ARCADIA PRIMER

[MBSE][LEC-007]

07:16

SUMMARY

REVIEW OF OUR APPROACH

07:16

3

ARCADIA METHOD

REF-006: VOIRIN, J.L. Model-based System and Architecture Engineering with the Arcadia Method. Elsevier, 2017. ISBN 978-0-0810-1794-4.

REF-007: ROQUES, P. Systems Architecture Modeling with the Arcadia Method – A Practical Guide to Capella. Elsevier, 2017. ISBN: 978-0-0810-1792-0

07:16

10

ARCADIA OPERATIONAL ANALYSIS

ARCADIA SYSTEM ANALYSIS

LOGICAL ARCHITECTURE

PHYSICAL ARCHITECTURE

EPBS (END PRODUCT BREAKDOWN STRUCTURE) AND INTEGRATION CONTRACTS

TRANSVERSAL MODELLING

Ending

07:16

197

WHAT IS SYSML?

SysML & ARCADIA

https://polarsys.org/capella/arcadia_capella_sysml_tool.html

REQUIREMENTS IN CAPELLA

Replicable Elements Collection (REC) e Replicas (RPL)

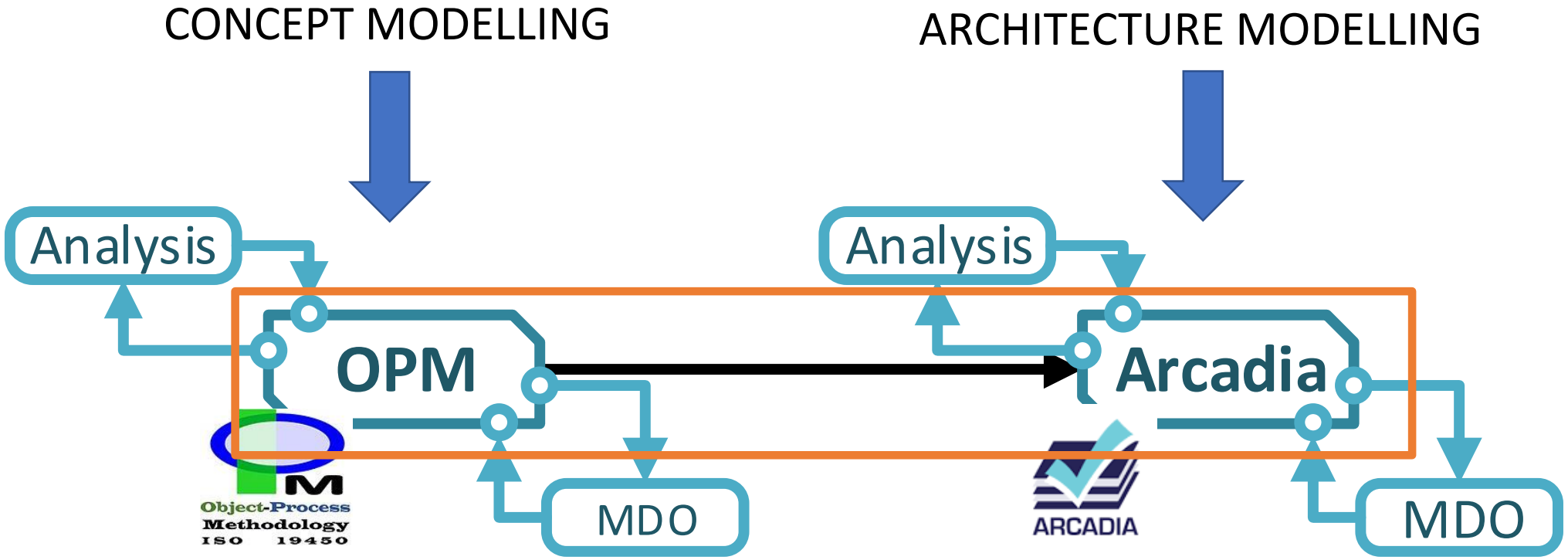
Written by Mateus S. Venturini

07:16

REVIEW OF OUR APPROACH

07:16

OUR APPROACH: OPM & ARCADIA HYBRID

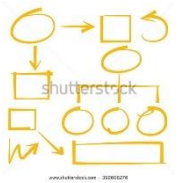


07:16

The start:
How do we explain ideas to each other?

Grab a pen and piece of paper, or a chalk and blackboard

Scribble shapes with names next to them



While talking, run lines with or without arrows among the shapes

Follow the reaction of the audience to see if idea is understood

Answer questions, continue scribbling…

07:16

The Start:

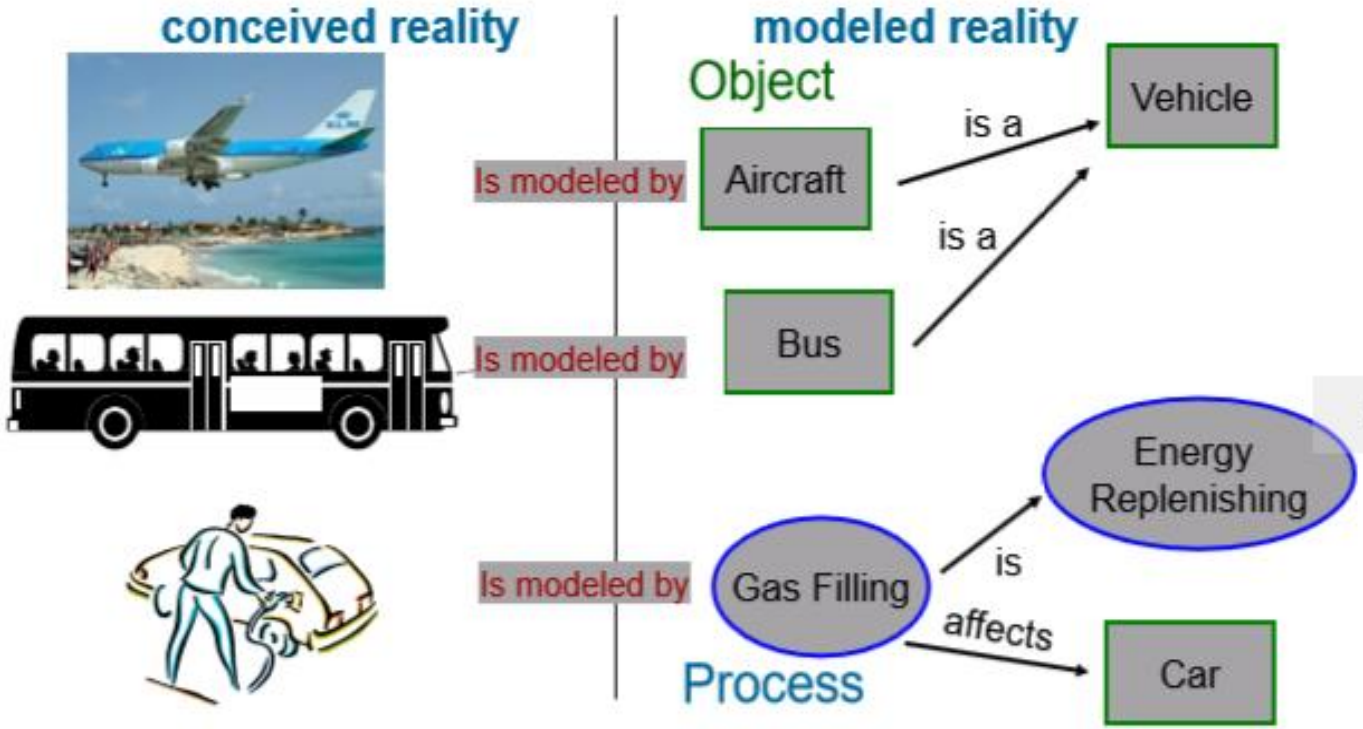
Convert — fragmented knowledge into explicit and integrated knowledge

+

Construct — concise models → Concept Models

+

Communicate — The model to stakeholders



- **simple** yet **expressive**, and
- **intuitive** yet **formal**

07:16

In OPM

- We are able to discuss **conceptual decisions**
  - Do a **fast state-machine simulation**
  - Identify **needs and some stakeholders**
  - **Identify main system elements to propose a CONOPs**
  - Proposal is to be a **formal language** that enables to capture a fairly amount of information to understand how the system will work.

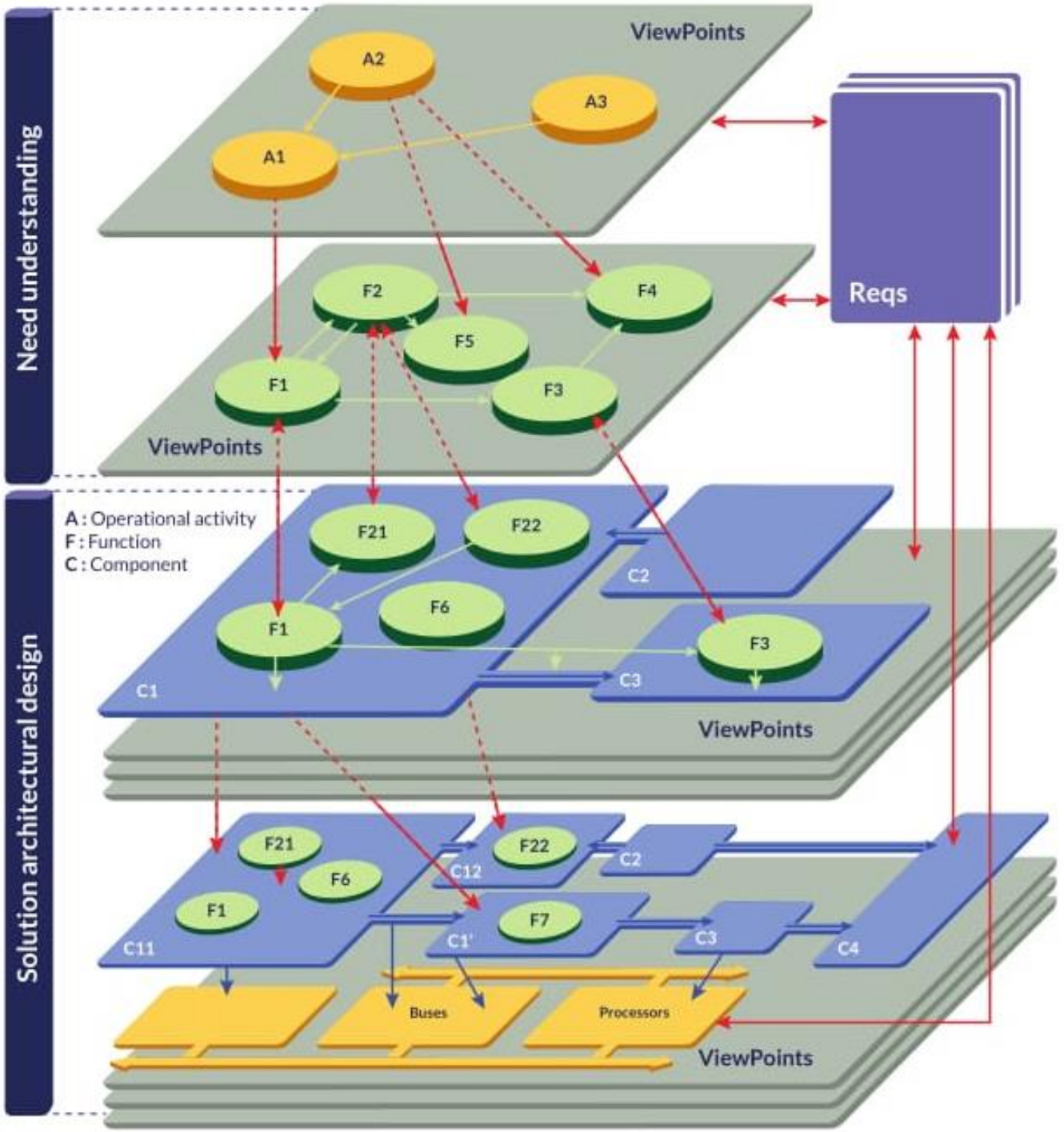
  - We could go from **need to screw** → but the method will get confused and lacks more capabilities when the complexity is **“heavily”** increased. (my opinion)

07:16

Evolving:
How can we construct the architecture?

- Analyzing stakeholders' necessities
- Analyzing the system functions that interact within the stakeholder’s necessities
- Creating a logical decomposition
- Creating a physical decomposition
- Distributing to the supply chain / development organizations

07:16



Operational Analysis
What the users of the system need to accomplish

Functional & Non Functional Need
What the system has to accomplish for the users

Logical Architecture
How the system will work to fulfill expectations

Physical Architecture
How the system will be developed and built

07:16

ARCADIA METHOD

REF-006: VOIRIN, J.L. Model-based System and Architecture Engineering with the Arcadia Method. Elsevier, 2017. ISBN 978-0-0810-1794-4.

REF-007: ROQUES, P. Systems Architecture Modeling with the Arcadia Method – A Practical Guide to Capella. Elsevier, 2017. ISBN: 978-0-0810-1792-0

07:16

- System engineers have been making use of modeling techniques for a long time. **Structured analysis and design technique** (SADT) and **structured analysis for real time** (SA/RT) are some of the best known of these, and date back to the 1980s. There are many other approaches based on Petri nets or **finite state machines**. However, these techniques are also limited by their range and expressivity, as well as by the difficulty in integrating them with other formalisms and with requirements.

- An interesting attempt was the publication of a UML variant for system engineering in 2006–2007. This new language, called SysML, was strongly inspired by version 2 of UML, but added the possibility of representing system requirements, non-software elements (mechanical, hydraulic, sensors, etc.), physical equations, continuous flows (matter, energy, etc.) and allocations. **Unfortunately, in practice it has been shown that the filiation of the SysML language to UML often leads to difficulty in terms of comprehension and use for system engineers who are not also computer scientists.**

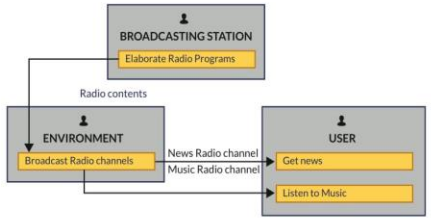
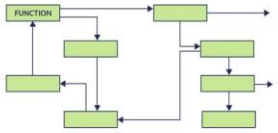
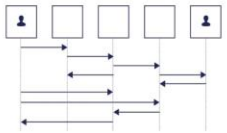
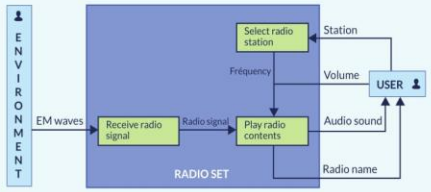
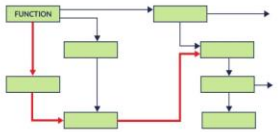
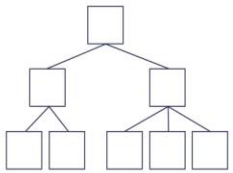
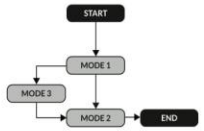
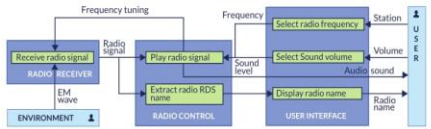
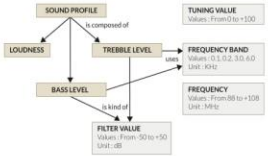
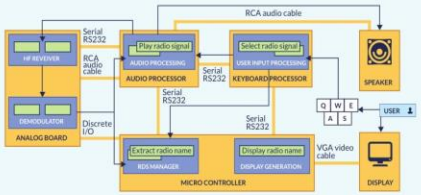
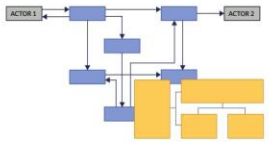
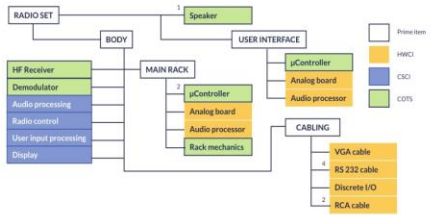
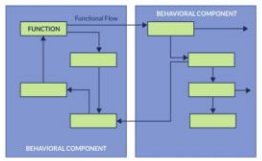
- This is the **reason** that led Thales to define the Arcadia method, along with its underlying formalism, for its own needs.
- It has been applied since 2011 in a growing number of projects across a great variety of domains (avionics, railway systems, defense systems in all fields, air traffic control, command control, area surveillance, complex sensor systems, satellite systems and ground stations, communications systems, etc.), and in many countries (France, Germany, United Kingdom, Italy, Australia, Canada, etc.).


Architect
Specialty engineering: safety, perf, security, ...
Functions
Safety
Performance
Human Factors
Security
IVVQ, Product Line, Cost...
ViewPoints
Product Line Manager, Etc.
Evaluation Rules
Sub-contractors
Solution Architecture

07:16

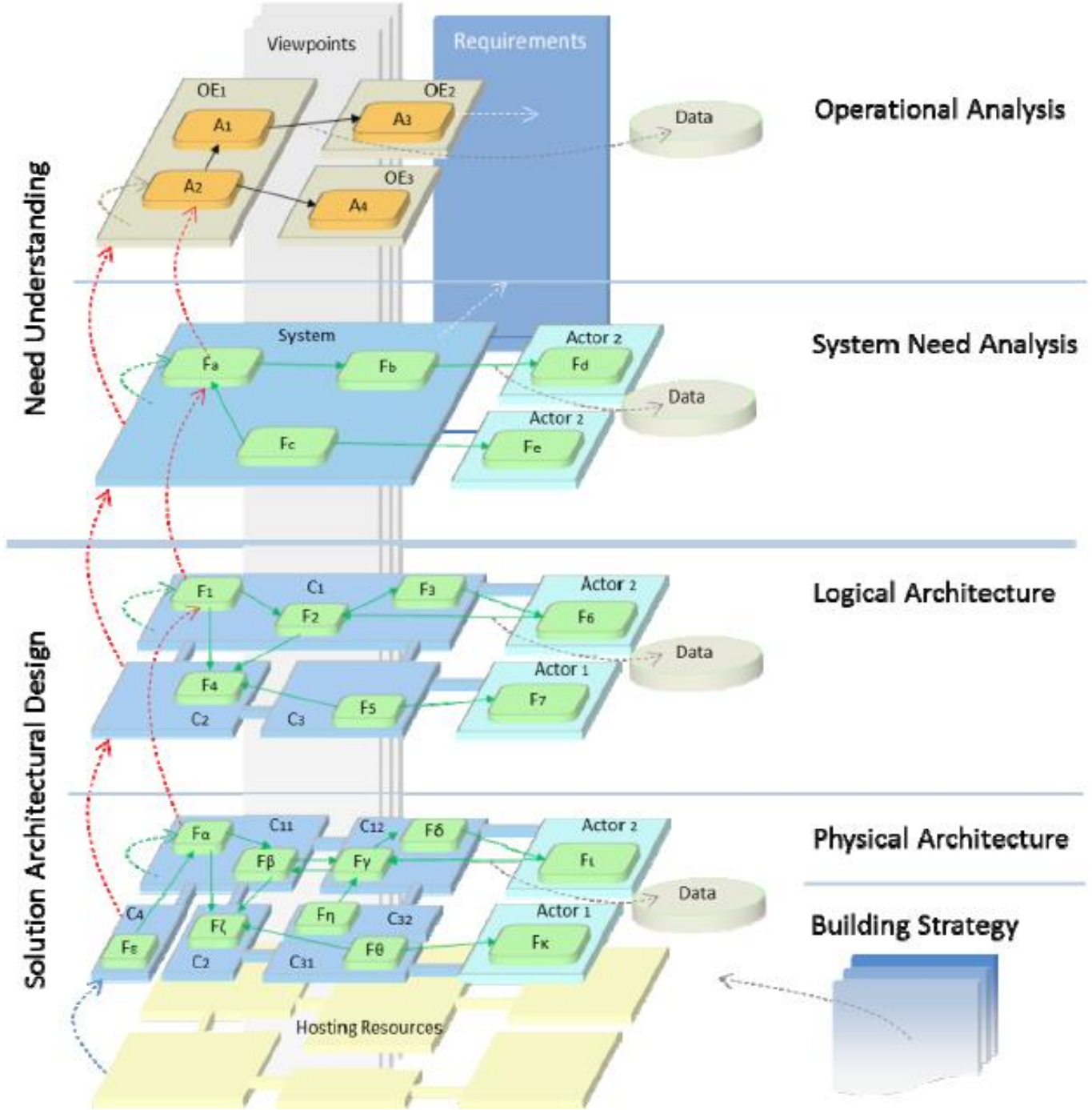
Founding principles

- all of the engineering stakeholders share the **same methodology**, the **same information**, **the same description of the need** and the product in the form of a shared model;
- each **specialized type of engineering** (for example security, performance, cost and mass) **is formalized as a "viewpoint"** in relation to the requirements from which the proposed architecture is then verified;
- the rules for the **anticipated verification of the architecture** are established in order to verify the architecture as soon as possible;
- **co-engineering between the different levels** of engineering is supported by the joint elaboration of models, and the models of the different levels and specialties are deducted/validated/linked one to the other.

METHOD STEPS	TASKS	SAMPLE MODEL	CONCEPTS	DESCRIPTION MEANS
NEED				
Customer Operational Need Analysis What the users of the system need to accomplish	✓ Define operational capabilities ✓ Perform an operational need analysis		- Operational capabilities - Actors, operational entities - Actor activities - Interactions between activities & actors - Information used in activities & interactions - Operational processes chaining activities - Scenarios for dynamic behaviour	 **Dataflow:** functions, op. activities interactions & exchanges **Scenarios:** actors, system, components interactions & exchanges 
System/ SW/HW Need Analysis What the system has to accomplish for the Users	✓ Perform a capability trade-off analysis ✓ Perform a functional and non-functional analysis ✓ Formalise and consolidate requirements		- Actors and system, capabilities - Functions of system & actors - Dataflow exchanges between functions - Functional chains traversing dataflow - Information used in functions & exchanges, data model - Scenarios for dynamic behaviour - Modes & states	 **Functional chains, operational processes** through functions & op. activities   **Modes & states** of actors, system, components **Breakdown** of functions & components
SOLUTIONS				
Logical Architecture Design How the system will work so as to fulfil expectations	✓ Define architecture drivers and viewpoints ✓ Build candidate architectural breakdowns in components ✓ Select best compromise architecture		*SAME CONCEPTS, PLUS :* - Components - Component ports and interfaces - Exchanges between components - Function allocation to components - Component interface justification by functional exchanges allocation	**Data model:** dataflow & scenario contents, definition & justification of interfaces 
Physical Architecture Design How the system will be developed & built	✓ Define architectural patterns ✓ Consider reuse of existing assets design a physical ✓ Design a physical reference architecture ✓ Validate and check it		*SAME CONCEPTS, PLUS :* - Behavioural components refining logical ones, and implementing functional behaviour - Implementation components supplying resources for behavioural components - Physical links between implementation components	 **Component wiring:** all kinds of components
Development Contracts What is expected from each designer/ sub-contractor	✓ Define a components IVVQ strategy ✓ Define & enforce a PBS and component integration contract		- Configuration items tree - Parts numbers, quantities - Development contract (expected behaviour, interfaces, scenarios, resource consumption, non-functional properties…)	**Allocation** of op.activities to actors, of functions to components, of behav.components to impl.components, of dataflows to interfaces, of elements to configuration items 

ARCADIA OPERATIONAL ANALYSIS

07:16



WHAT IS IN THE OPERATIONAL ANALYSIS (OA)

ARCADIA OPERATIONAL CONCEPTS:

OPERATIONAL DIAGRAMS

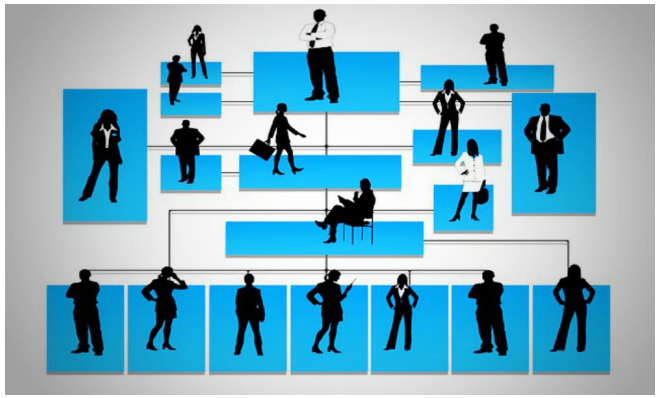
STEP BY STEP EXAMPLE

ARCADIA OPERATIONAL CONCEPTS:

07:16

- **Operational Capability**: capability of an organization to provide a high-level service leading to an **operational objective** being reached (*for example Provide weather forecasts, etc.*); - high-level objectives

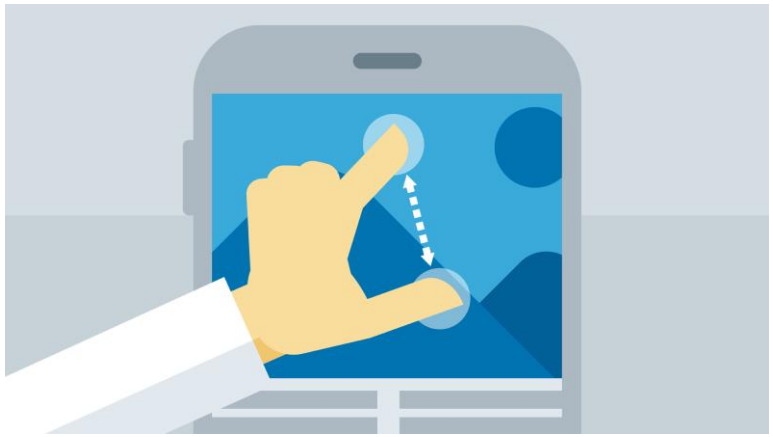
- **Operational Entity**: entity belonging to the real world (*organization, existing system, etc.*) whose role is to **interact with the system** being studied or **with its users** (*for example Crew, Ship, etc.*);



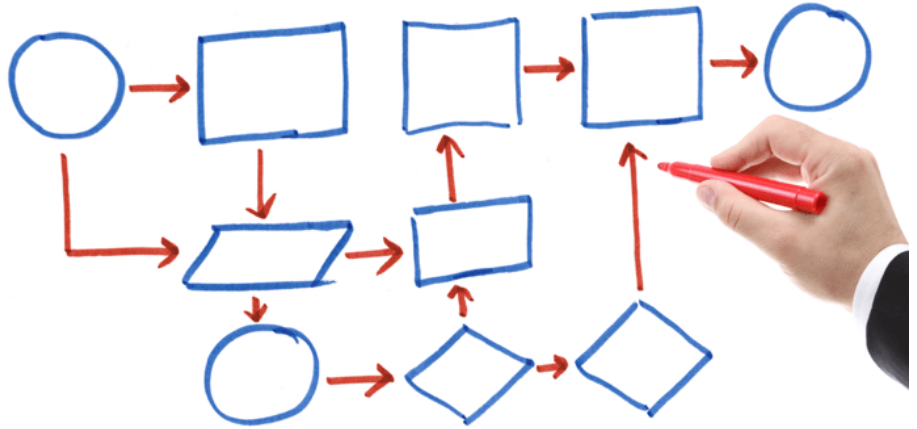
- **Operational Actor**: particular case of a (*human*) **non-decomposable operational entity** (*for example Pilot, etc.*);

- **Operational Activity**: process step carried out in order to **reach a precise objective** by an operational entity, which might need to use the future system in order to do so (*for example Detect a threat, Collect meteorological data, etc.*);

- **Operational Interaction**: **exchange of information** or of unidirectional matter **between operational activities** (for example meteorological data, etc.);

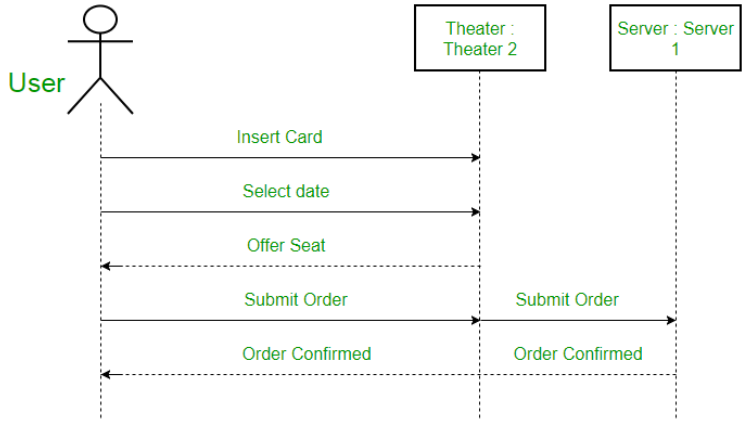


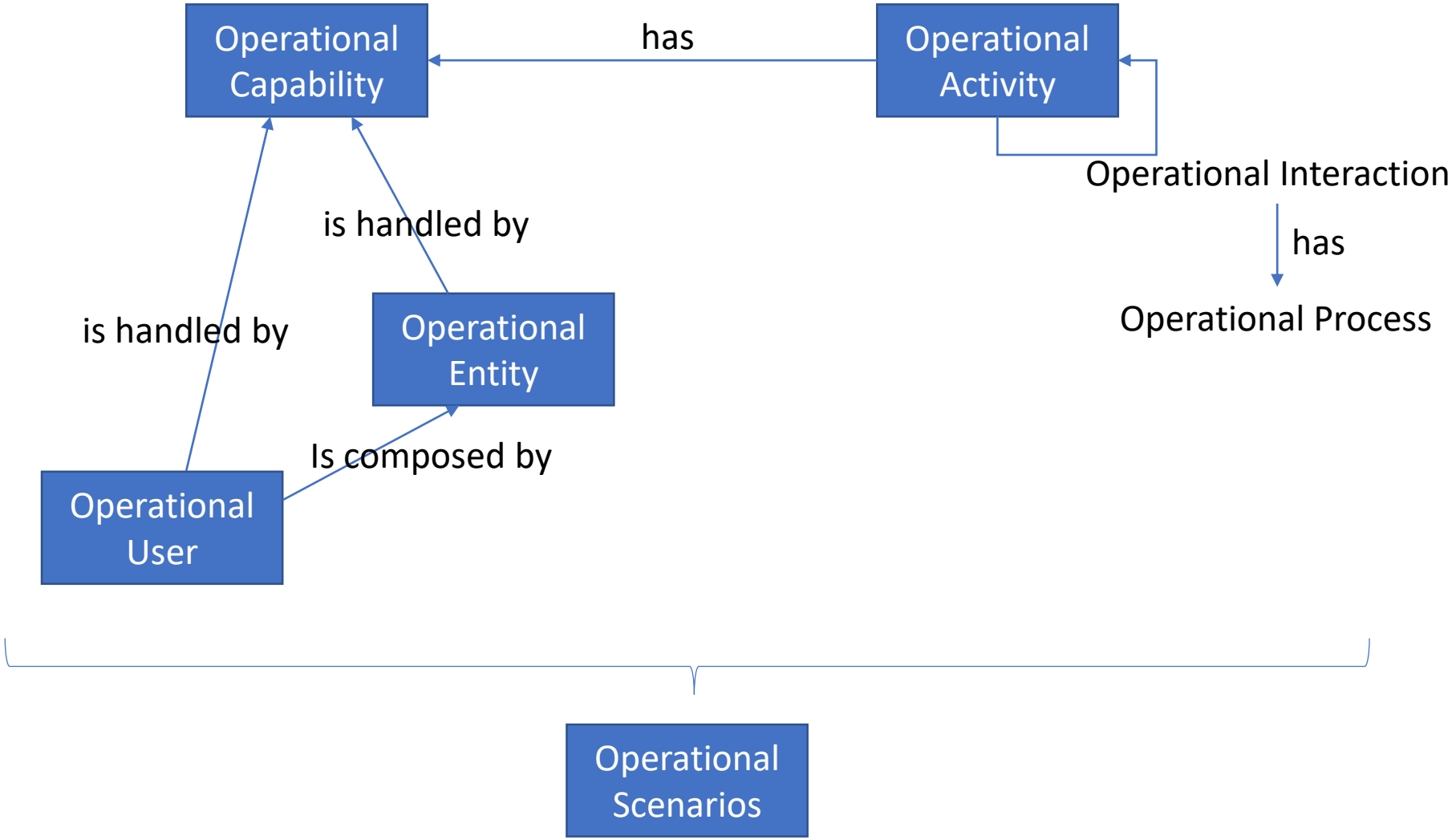
- **Operational Process**: **series** of **activities** and of **interactions** that contribute toward an operational capability.

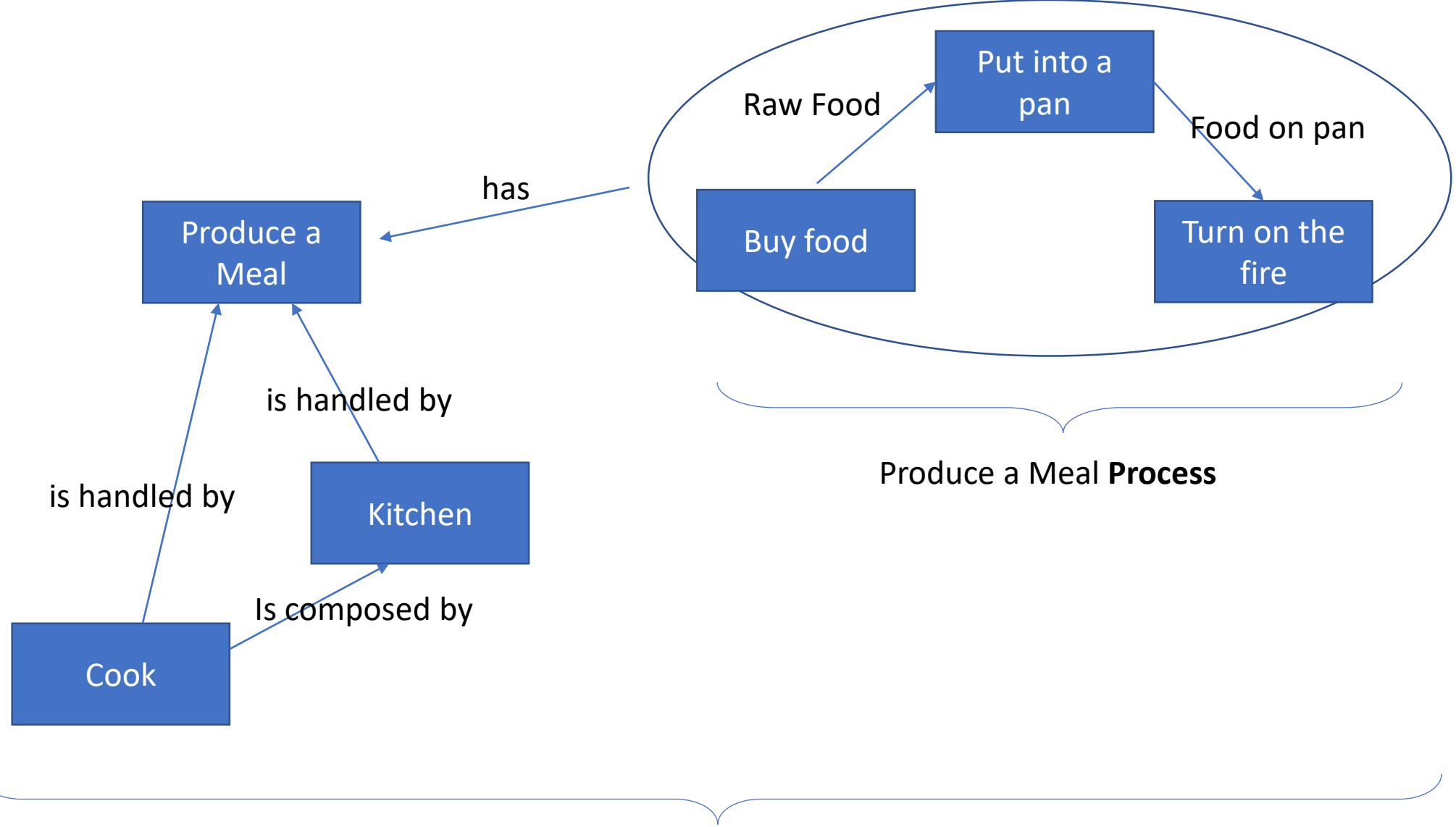


07:16

- **Operational Scenario**: scenario that **describes the behavior** of entities and and/or operational activities in the **context of an operational capability**. *It is commonly represented as a sequence diagram, with the vertical axis representing time.*




Operational Capability
has
Operational Activity
Operational Interaction
has
Operational Process
is handled by
is handled by
Operational Entity
Is composed by
Operational User
Operational Scenarios



1st you have to buy food

2nd put (the raw food) into a pan

3rd with the raw food into a pan turn on the fire

WHAT IS IN THE OPERATIONAL ANALYSIS (OA)

07:16

Operational Analysis

"What system users must achieve"

"What the users of the system need to accomplish"

- This perspective analyses the issue of operational users, by identifying actors that have to interact with the system, their goals, activities, constraints and the interaction conditions between them.
- Analysis of the issues of operational users by identifying the actors that must interact with the system, their activities and their interactions with each other.

- trying to best satisfy a customer need, without having an imposed system scope

- OA ***should not mention the system***, *so as not to bar itself from potentially interesting alternatives for achieving the satisfaction of customer needs*: it aims at understanding this need without any *a priori* assumptions about how the system will contribute thereto; this is to not restrict the scope of possibilities too quickly.

07:16

- EXAMPLE.– Suppose that the customer need is to be able to hang a mirror on a wall.

If this need is translated too quickly into "how to attach a dowel to the wall with a drill?"

- this prematurely excludes other possibilities (**such as using glue, for example**),
- and also criteria that would help guiding the process toward the right solution (**such as the need or not to be able to disassemble the mirror at a later time**).

07:16

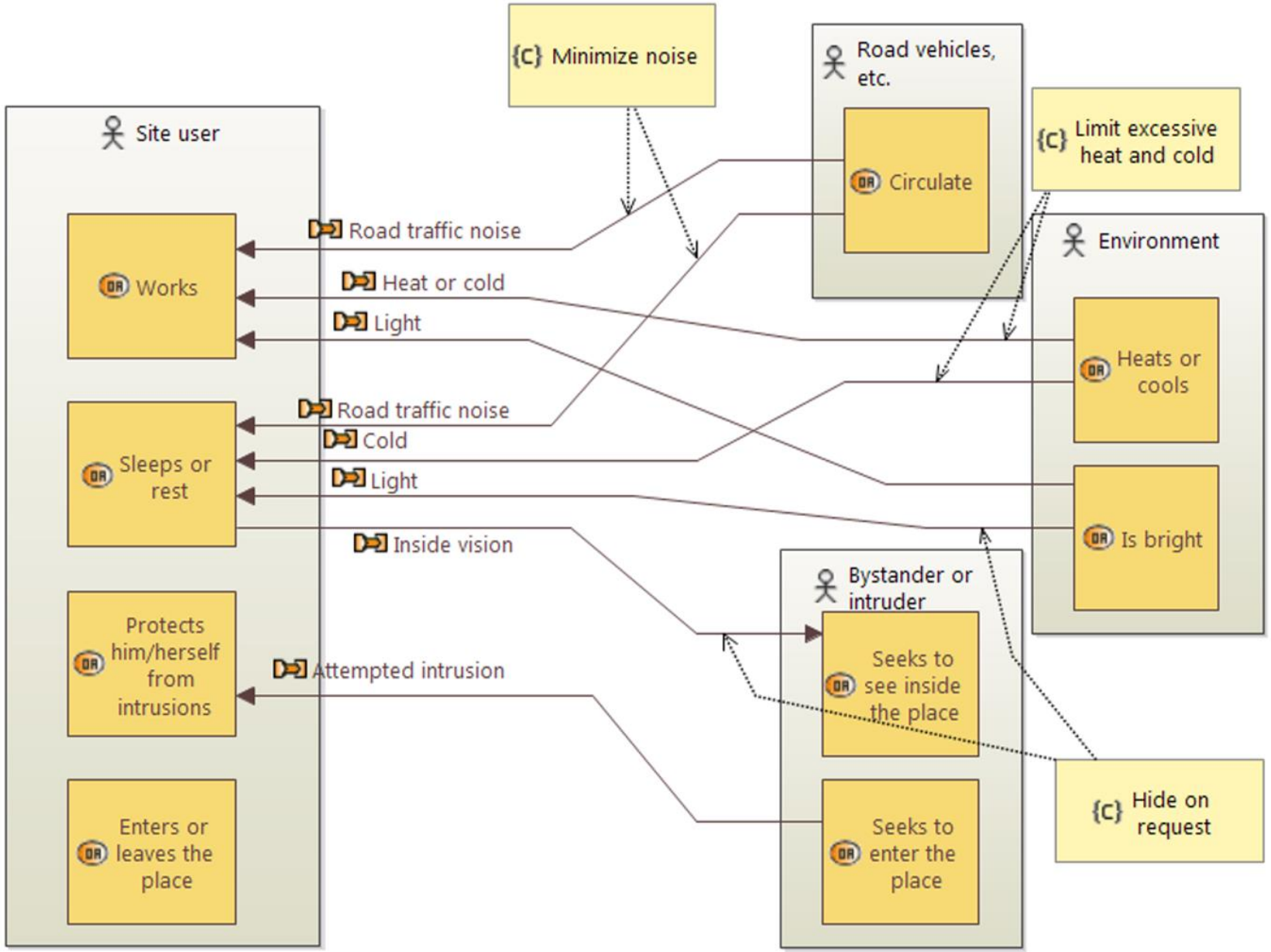
Define Missions and Required Operational Capabilities

- The **first step** consists of determining future system and environment users' missions – or more generally:
  - their motivations, expectations, goals, objectives, intentions, etc., as well as the **capabilities required to assume these missions**.

- **Existing constraints** on the execution of the mission must also be identified at all levels likely to impact it:
  - actor skills, operating modes and responsibilities, rules and associated procedures, existing means and systems, regulatory constraints, temporal and programmatic aspects, etc.

Perform operational needs analysis

- The goal is to capture the **conditions for the completion of a mission** previously identified, and those for the implementation of associated capabilities, mainly through the activities and interactions of the key players that contribute thereto.
- The **various situations** that directly shape and influence the missions, nominal or non-nominal, and the worst cases likely to be met, should be formalized. The analysis and the comparison of situations and conditions of missions must constitute a permanent concern; in fact, they are likely to guide both the needs analysis, to develop it by revealing constraints likely to have a high impact, but also the opportunities for development of processes, the principles behind the implementation of the mission, etc.
- The **different situations encountered during the mission are formalized in the form of scenarios that specify conditions for the implementation** of the required capabilities and the contributions of each stakeholder (actors, activities, interactions, etc.), as well as operational processes that implement activities contributing to a capability or part of the mission.


{C} Minimize noise
Road vehicles, etc.
Circulate
{C} Limit excessive heat and cold
Site user
Environment
Works
Road traffic noise
Heat or cold
Light
Heats or cools
Sleeps or rest
Road traffic noise
Cold
Light
Is bright
Inside vision
Bystander or intruder
Protects him/herself from intrusions
Attempted intrusion
Seeks to see inside the place
{C} Hide on request
Enters or leaves the place
Seeks to enter the place

07:16

OPERATIONAL DIAGRAMS

Describe the **state machine** of the system, specifying which are its modes and states. Among others, state machines can be created on the following kinds of elements: system, components, actors, classes (data), etc.

Describe the domain elements and the actually exchanged data.

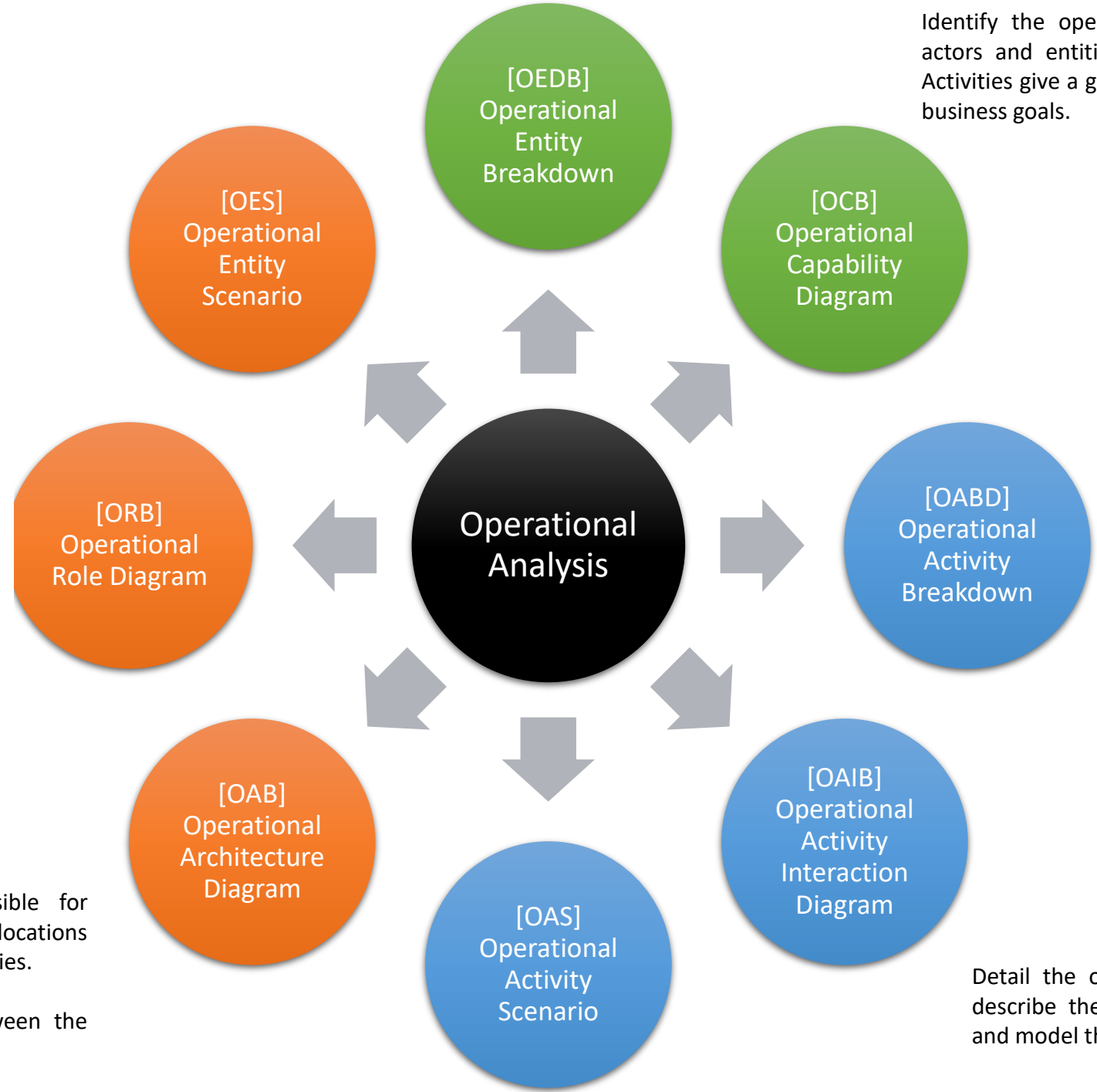
- **Domain Elements**: modeling elements of the domain should not be "polluted" by technical considerations (e.g.: internal representationof data, database storage, access methods, etc.). In the beginning, concentrate only on elements that provide high-level semantic related to the domain.
- **Data actually exchanged between components**: used for example to type parameters in interface operations. These data have to be unambiguousand consistent.

Both the domain elements and actual data are described in a Class diagram.

Operational actors and entities are responsible for performing the operational activities. Manage allocations and deduce communication means between entities.

07:16

Create Scenarios to illustrate interactions between the operational actors and entities

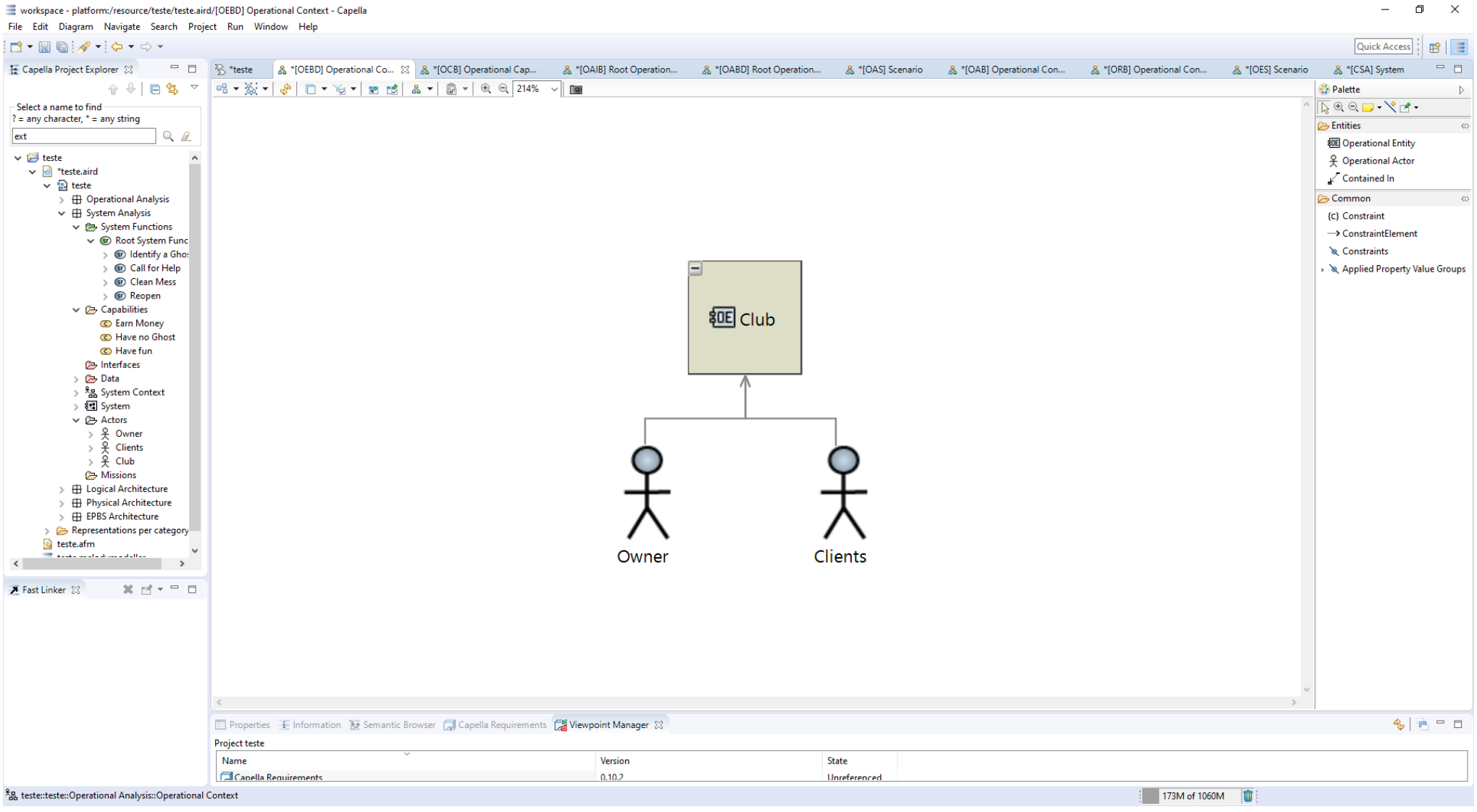


Identify the operational domain: who are the actors and entities, which are their purposes? Activities give a global view upon the operational business goals.

Detail the operational activity breakdown, describe the interactions between entities and model the operational processes

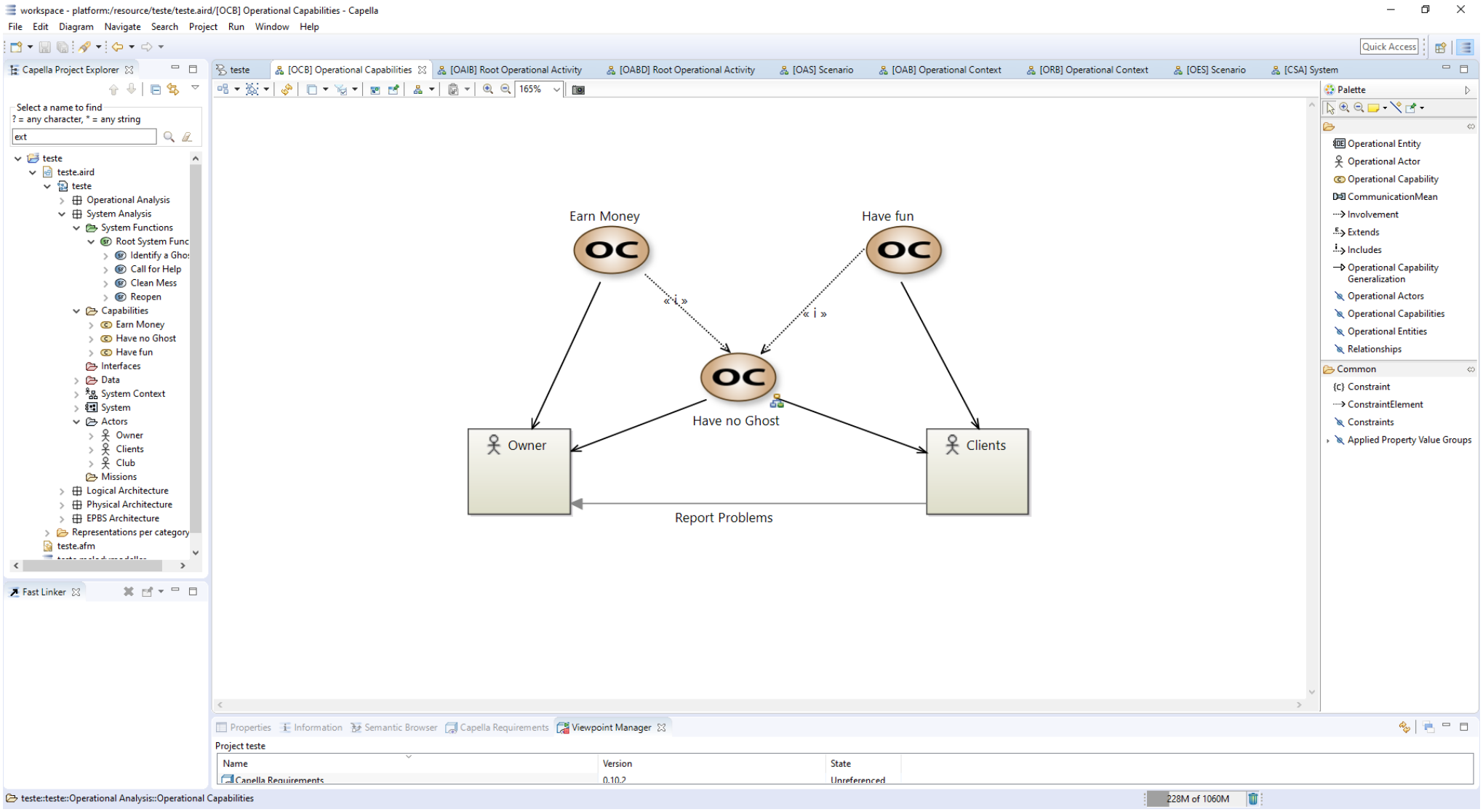
STEP BY STEP EXAMPLE

[OEDB] Operational Entity Breakdown

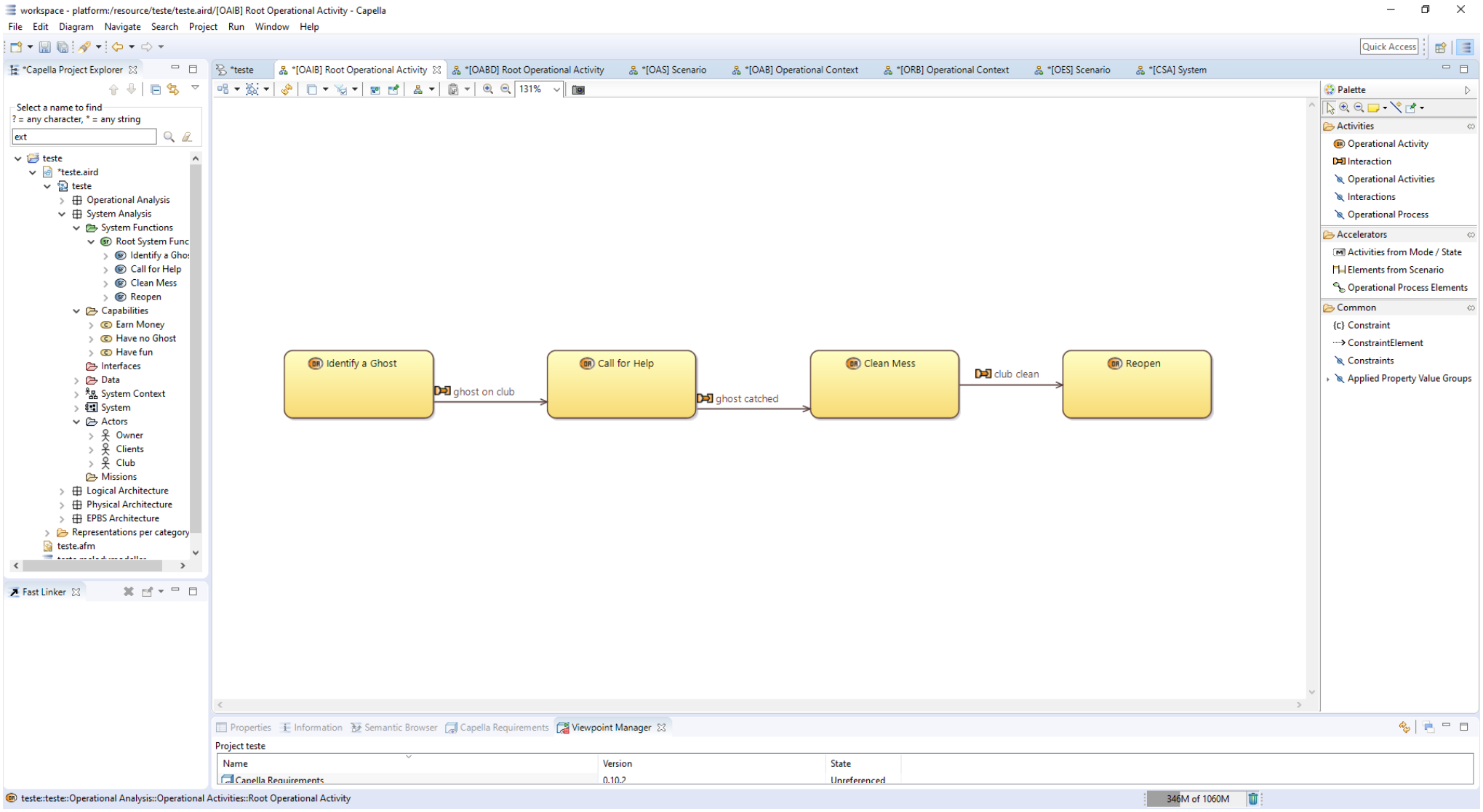


07:16

[OCB] Operational Capability Diagram

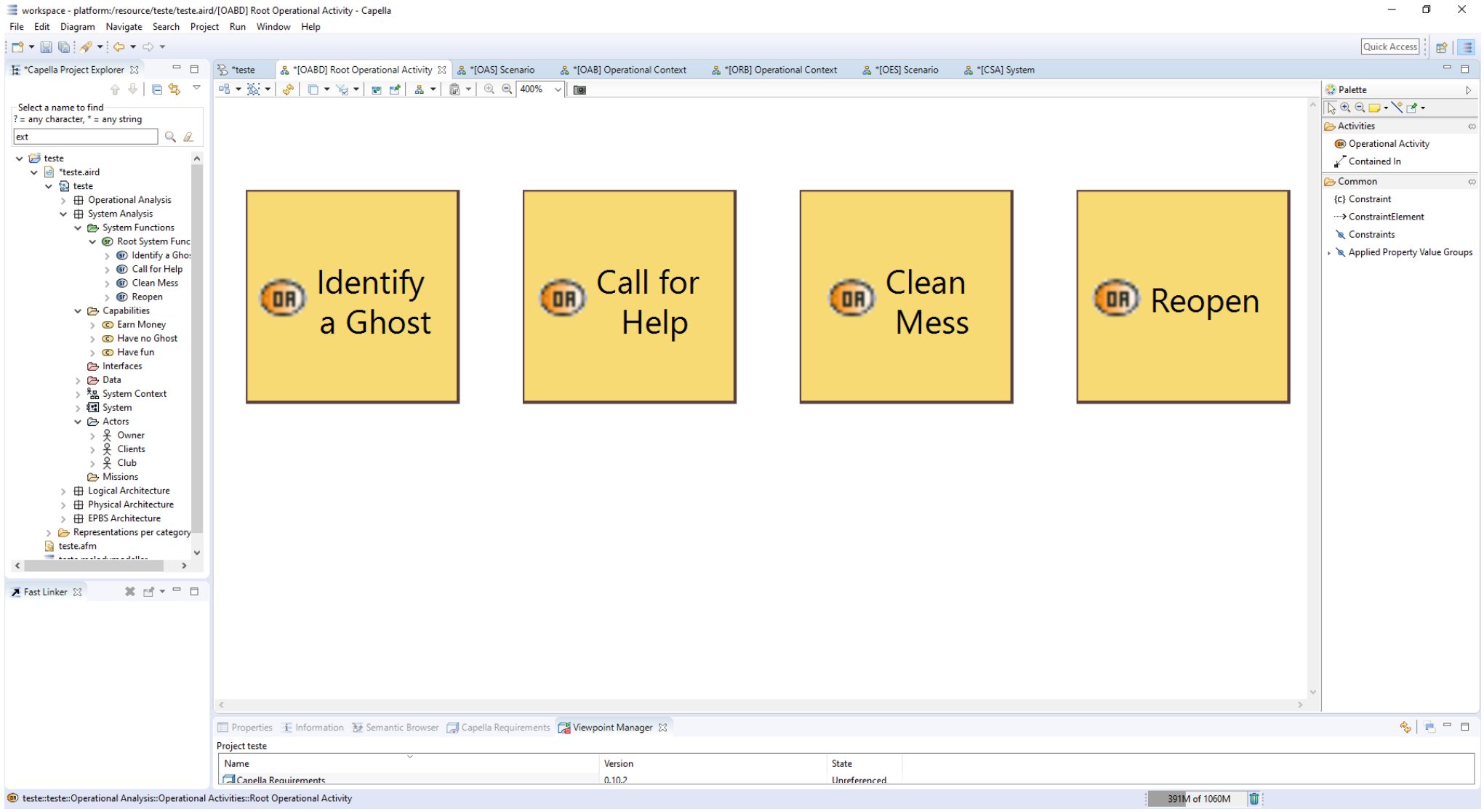


[OAIB] Operational Activity Interaction Diagram

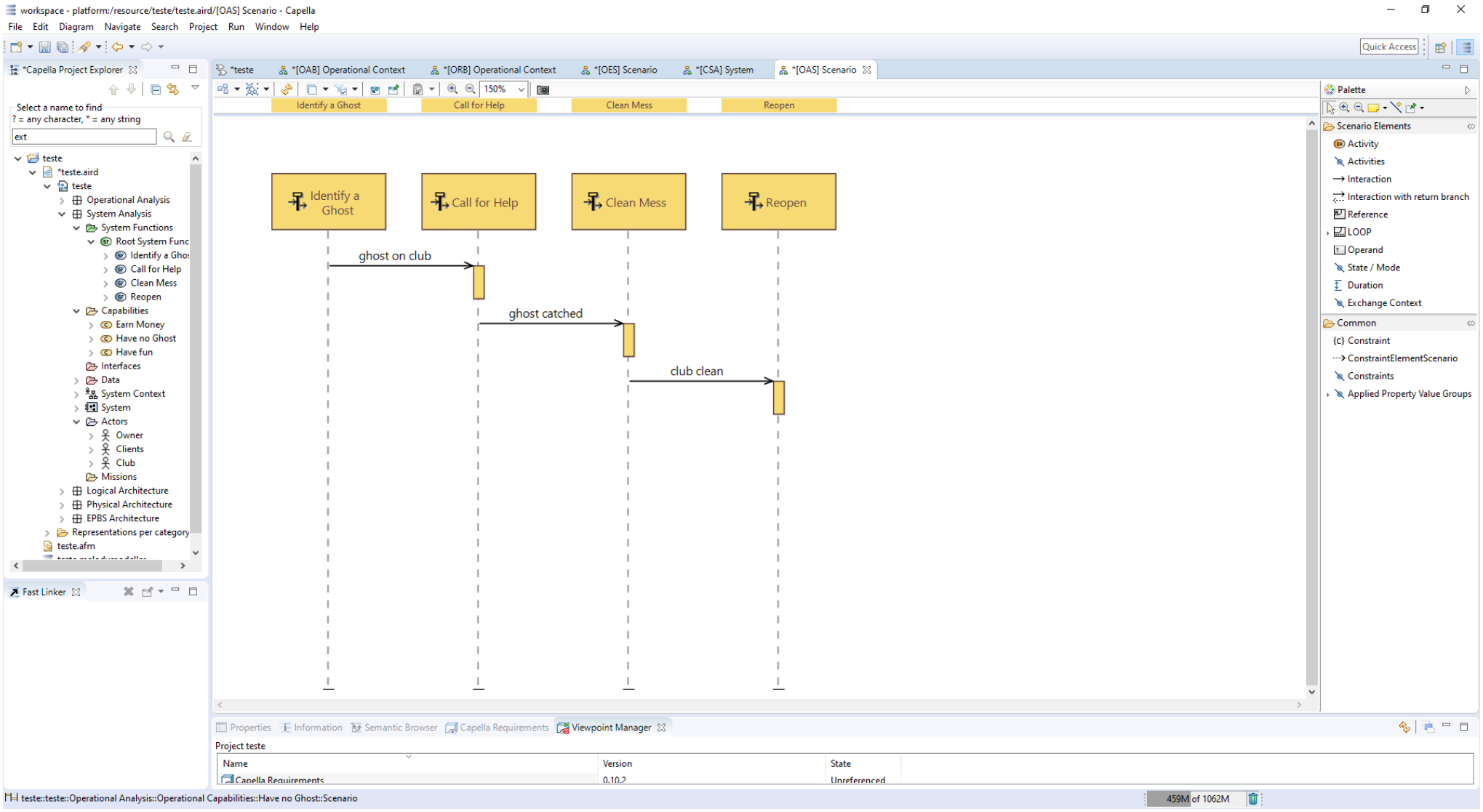


07:16

[OABD] Operational Activity Breakdown

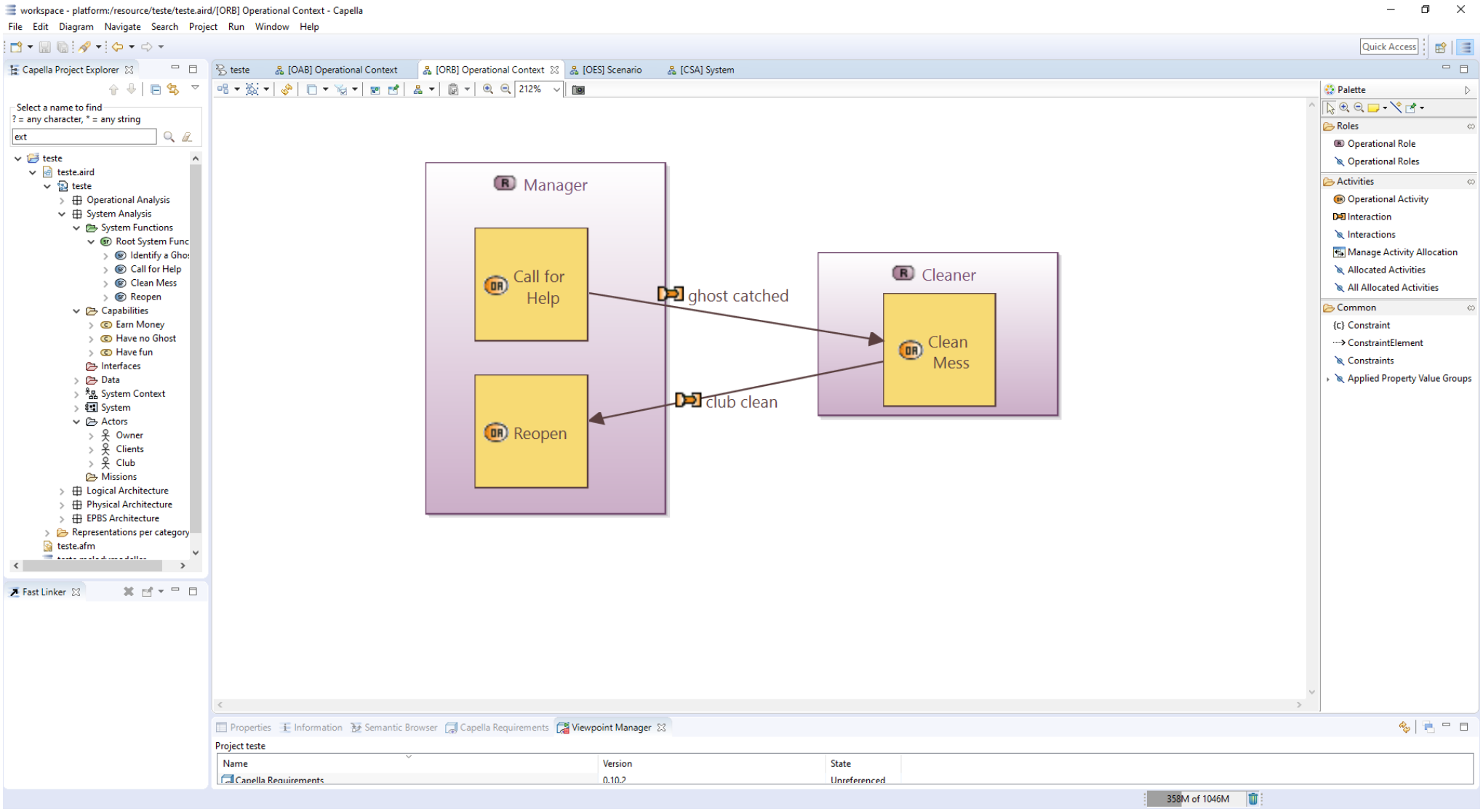


[OAS] Operational Activity Scenario

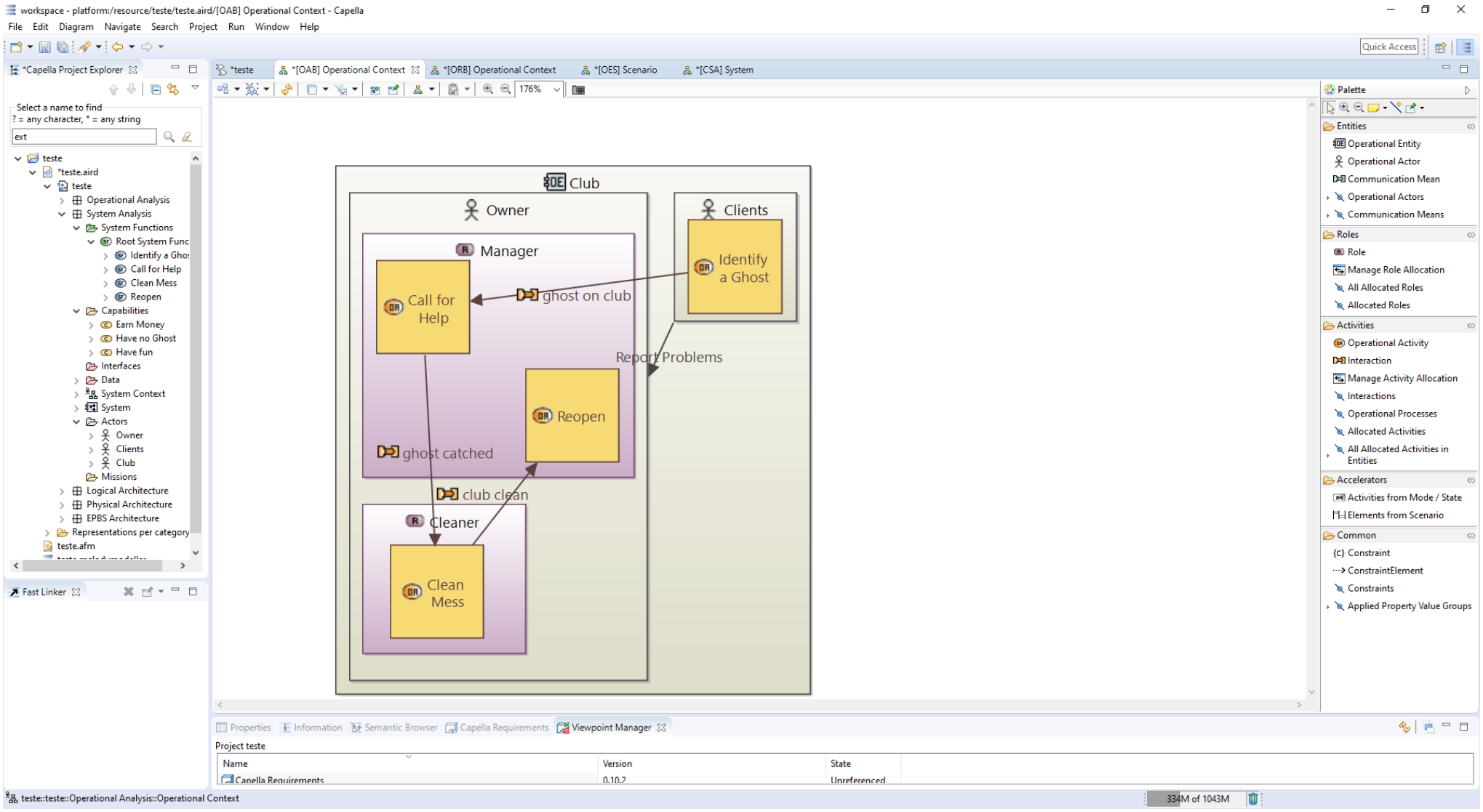


07:16

[ORB] Operational Role Diagram

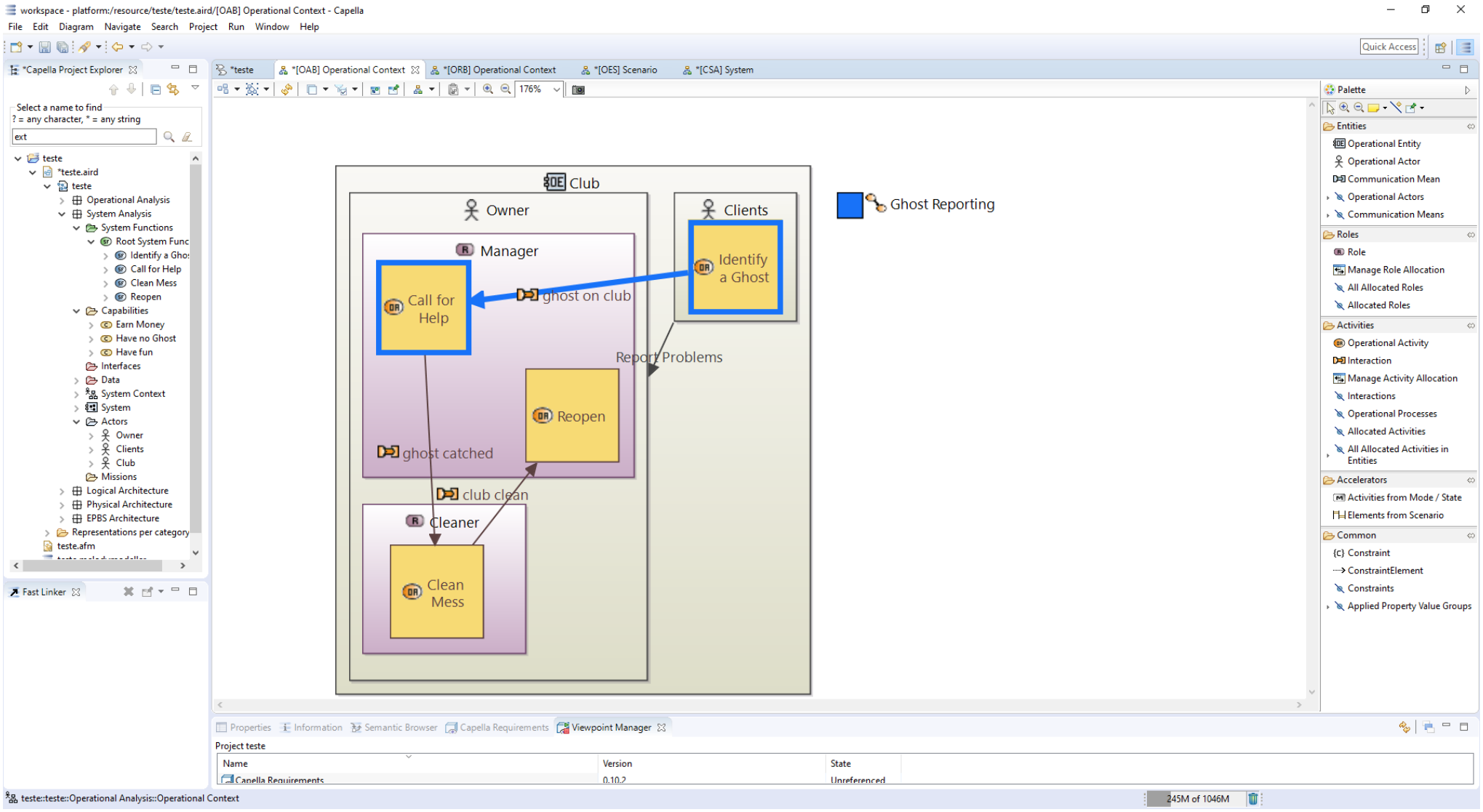


[OAB] Operational Architecture Diagram



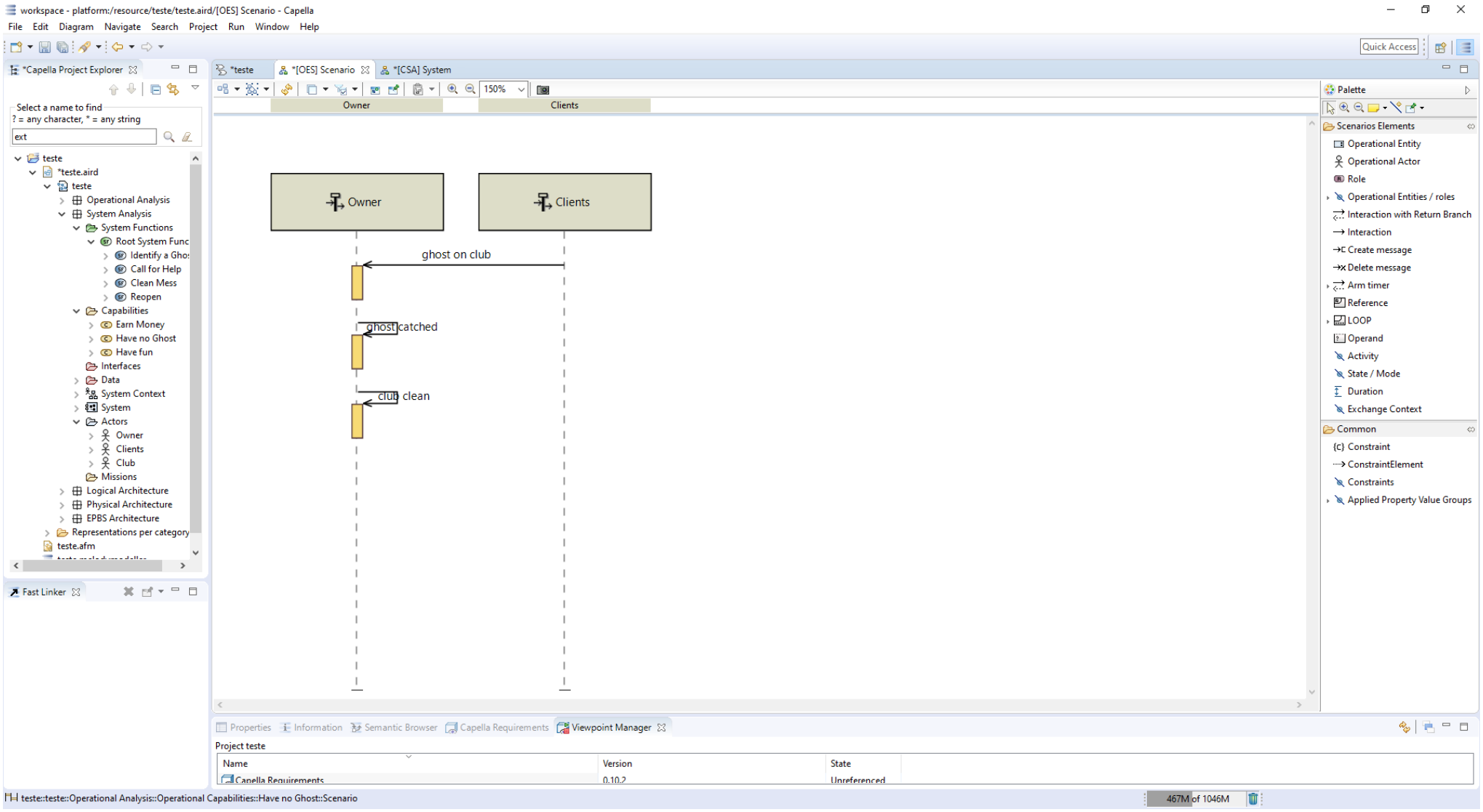
07:16

[OAB] Operational Architecture Diagram <<Operational Process>>



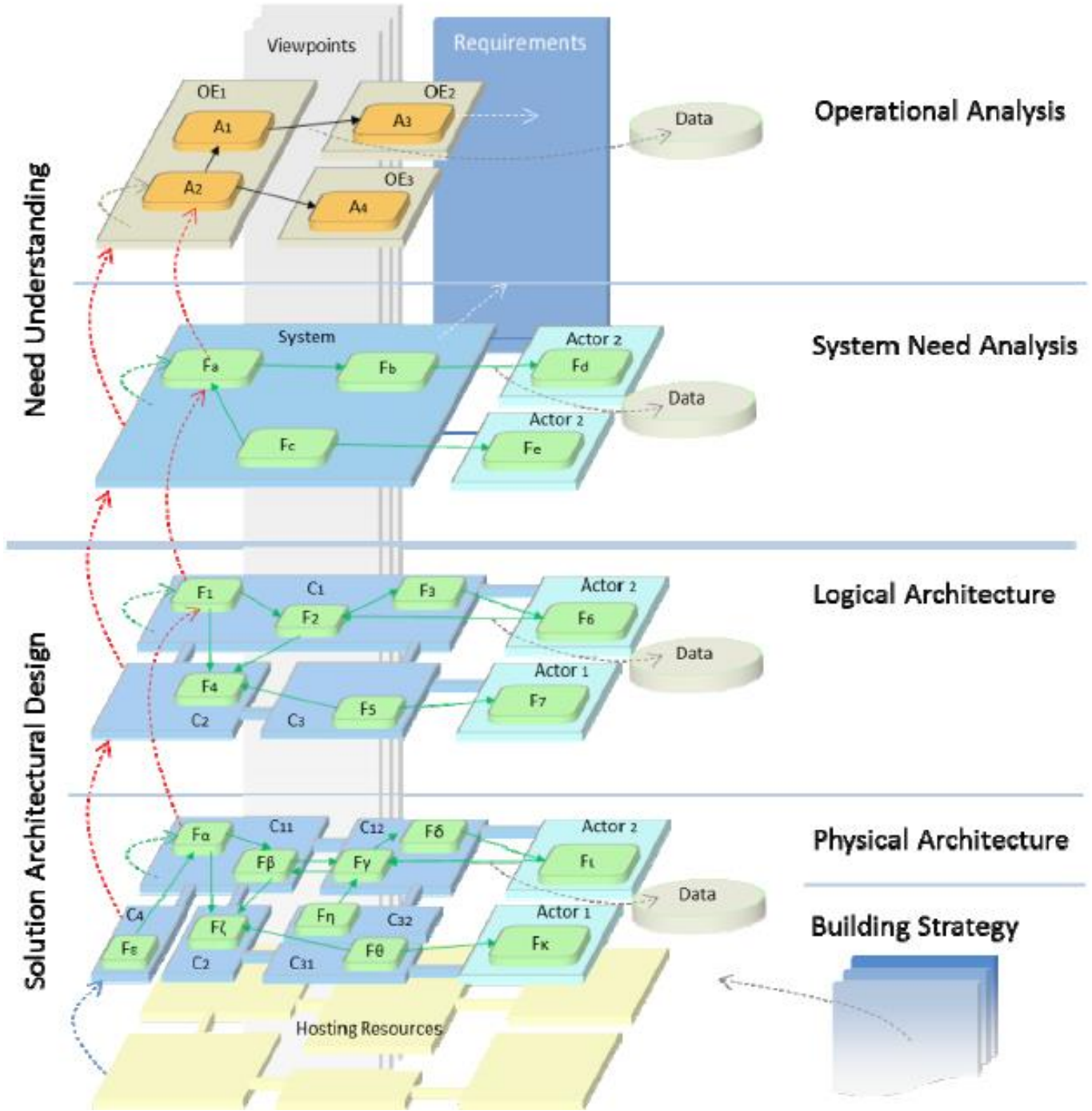
07:16

[OES] Operational Entity Scenario



07:16

ARCADIA SYSTEM ANALYSIS



WHAT IS IN THE SYSTEM ANALYSIS (SA)

ARCADIA SYSTEMIC CONCEPTS:

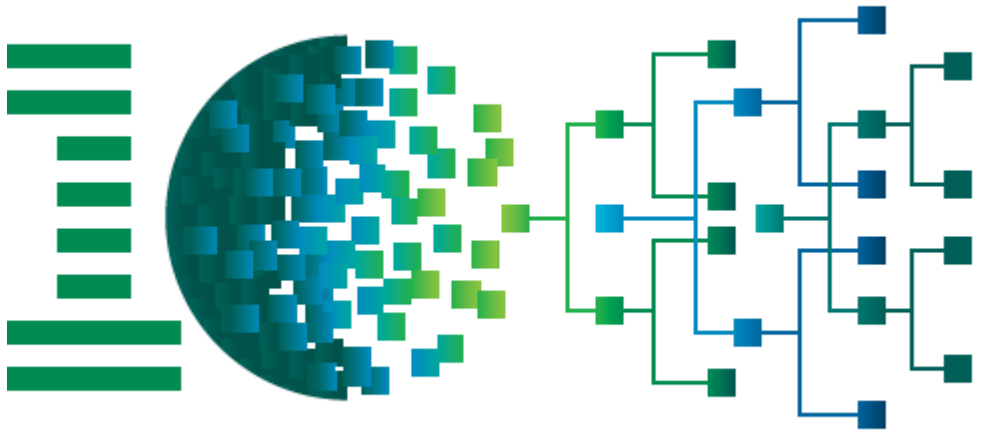
SYSTEM DIAGRAMS

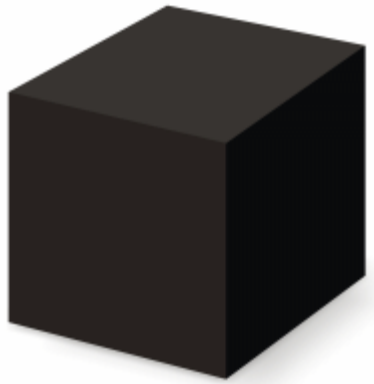
STEP BY STEP EXAMPLE

07:16

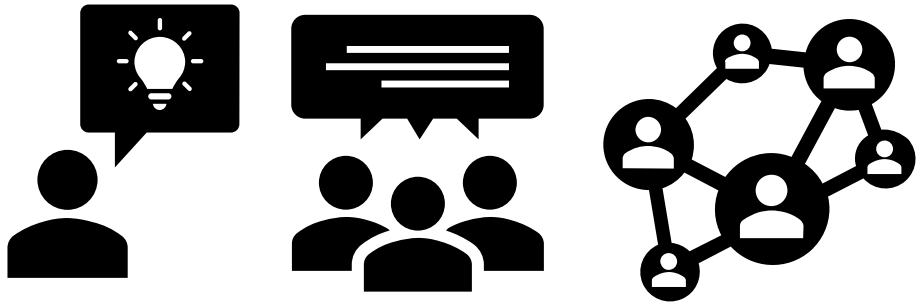
ARCADIA SYSTEMIC CONCEPTS:

- **System**: organized group of elements that function as a **unit (black box)** and **respond to the needs of the users**. The System owns **Component Ports** that allow it to interact with the external Actors;





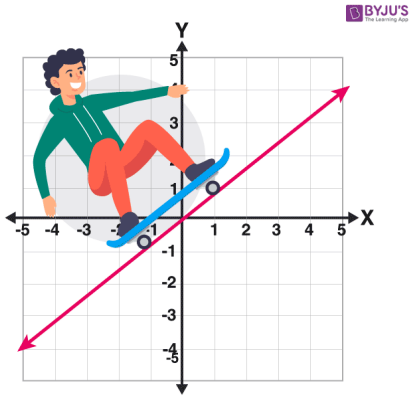
- **Actor**: any element that is **external to the System** (human or nonhuman) that **interacts with it**. (for example Pilot, Test operator, etc.);

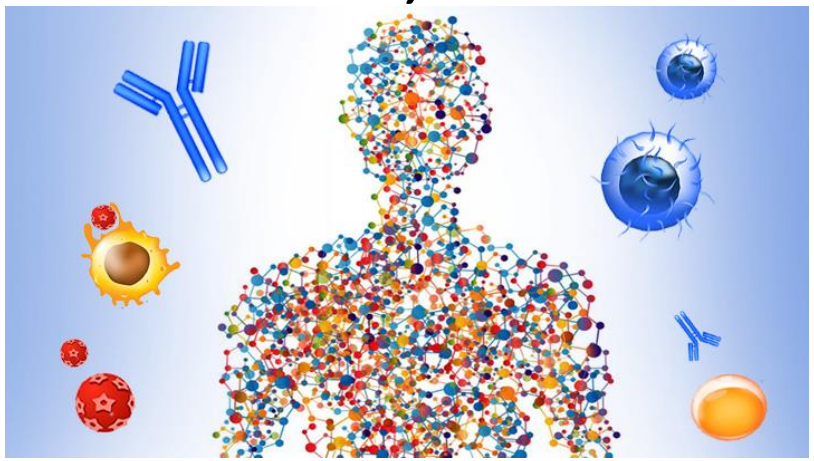


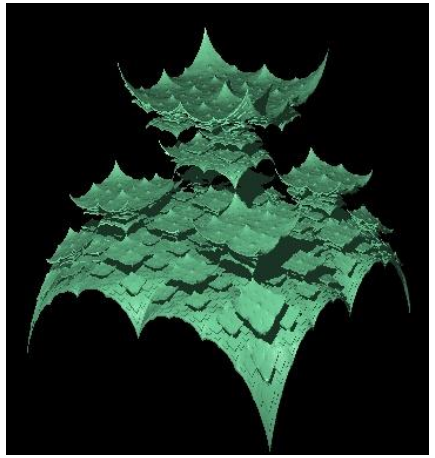
07:16

- **System Capability**: capability of the System to provide a highlevel **service** allowing it **to carry out an operational objective** (*for example provide meteorological data, etc.);*

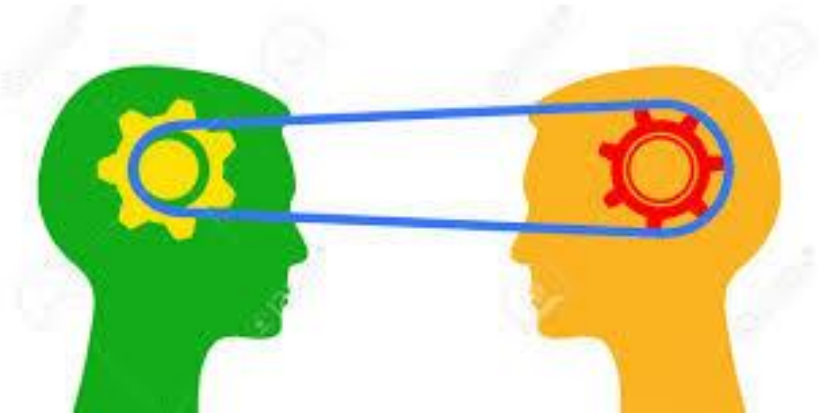
- **Function**: **behavior** or **service provided** by the System or by an Actor (for example detect a threat, measure altitude, etc.). A Function owns **Function Ports** that allow it to communicate with the other Functions. A Function can be split into subfunctions;

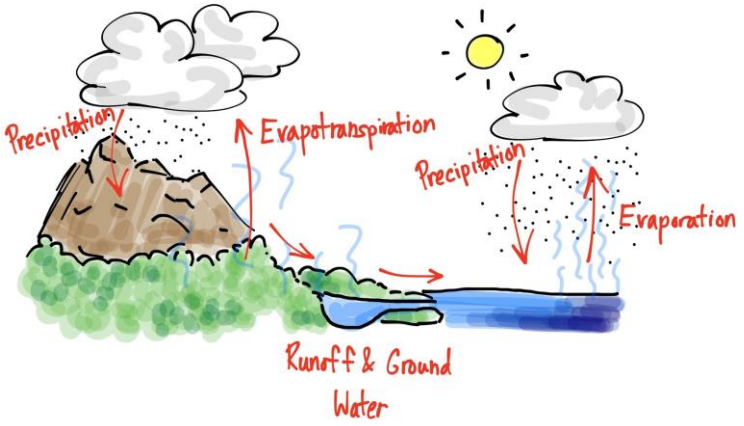




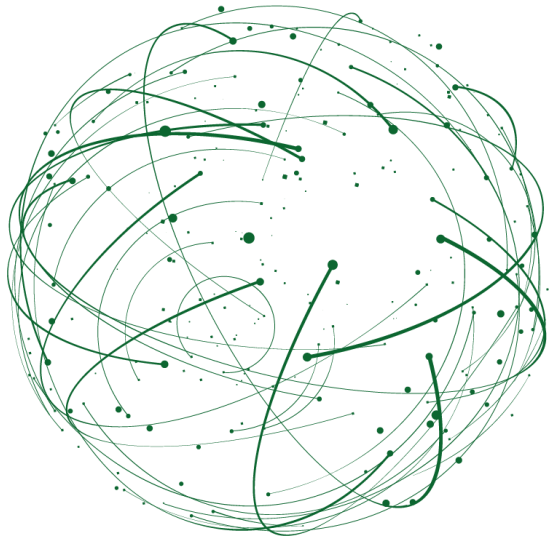


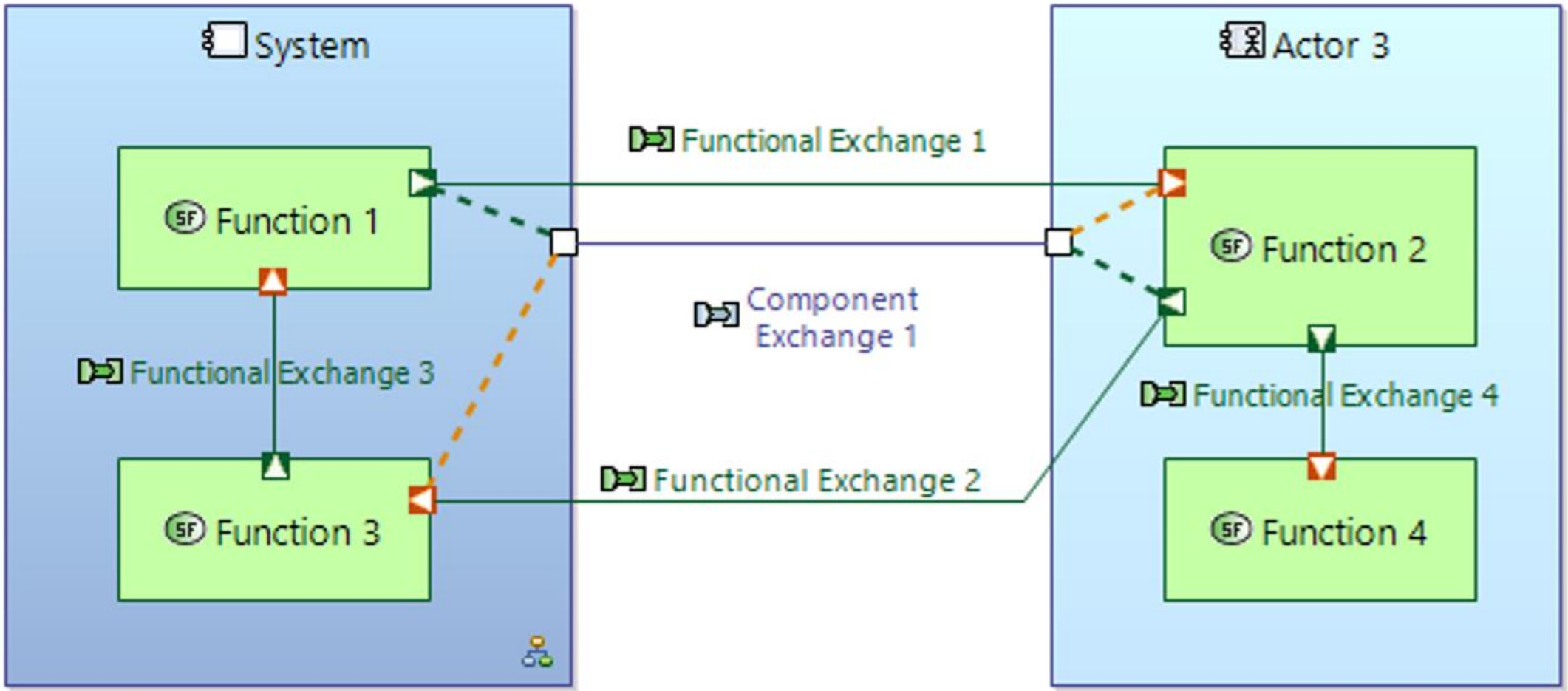
- **Functional Exchange**: **unidirectional exchange of information or of matter** between two Functions, linking two **Function Ports**;





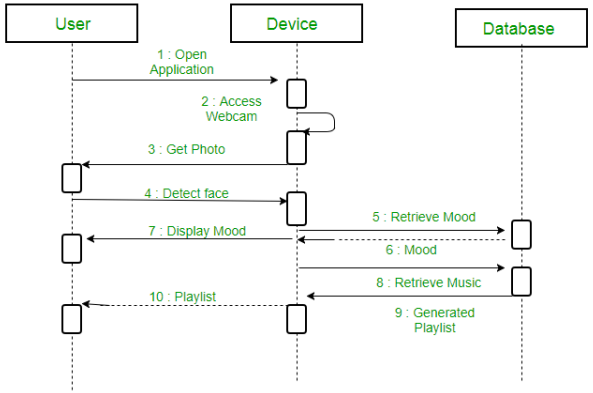
- **Component Exchange**: **connection** between the System and one of its external Actors, allowing **circulation of Functional Exchanges**;




System
Actor 3
Functional Exchange 1
Function 1
Function 2
Component
Exchange 1
Functional Exchange 3
Functional Exchange 4
Functional Exchange 2
Function 3
Function 4

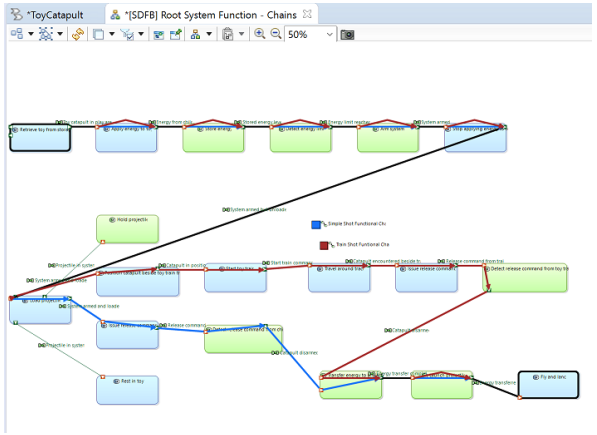
07:16

- **Scenario**: dynamic occurrence describing how the System and its Actors **interact** in the context of a System Capability. It is commonly represented in the form of a sequence diagram, with the vertical axis representing time;



07:16

- **Functional Chain**: element of the model that **enables a specific path to be designated among all possible paths** (using certain Functions and Functional Exchanges). This is particularly useful for assigning constraints (latency, criticality, etc.), as well as organizing tests.



WHAT IS IN THE SYSTEM ANALYSIS (SA)

07:16

System Analysis

"What the system must achieve for users"

"What the system has to accomplish for the users"

- The SA perspective defines the **expectations of the system**, that is to say **what the system has to perform for users**: it builds an external functional analysis, based on the OA and input textual requirements, to identify in response functions, services and expected system behaviors, necessary to its users.
  - external functional analysis as a response to identify the system functions needed by its users (e.g. "calculate the optimal path" and "detect a threat"), limited by the non-functional properties asked for.
- The System is identified as a modeling element at this level. It is a "black box" containing no other structural elements, only allocated Functions.

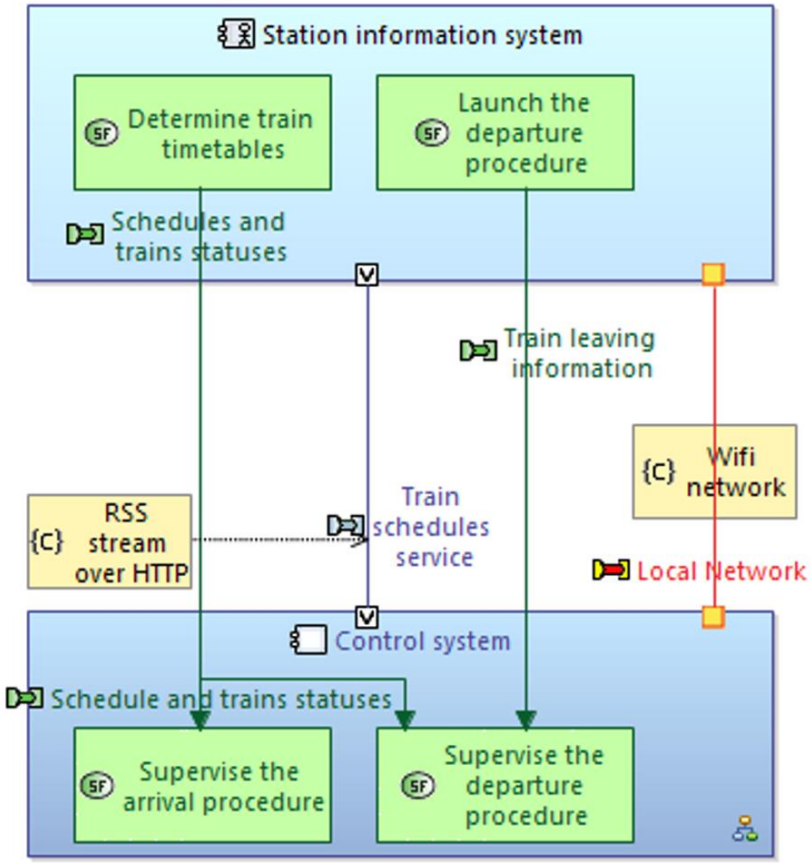
- The purpose of system needs analysis (referred to as SA further in the text) is to define the contribution expected of the system to users' needs, as they are described in the previous operational analysis (OA) and/or in the form of requirements expressed by the client.
- SA delimits the functions required of the system, distinguishing them from those assumed by the users or external systems.
- *It is essential to limit the functional analysis conducted in SA to the sole capture of the need, and only of need, excluding any implementation choice or details*. This allows freedom of choice to be maintained during the subsequent development of the solution,

Perform Capability Analysis

- Define the **essential characteristics** **necessary for the fulfillment of each operational capability** (the problem space), to uncover different alternative orientations likely to satisfy these required capabilities as well as the **criteria for associated appreciation and choice** (the solution space), and to compare these orientations to find the one(s) exhibiting the best compromise between the desirable characteristics.
- These parameters may concern the **functional contribution** **and the expected performance of the system**, obviously, but much further: organization, doctrines, procedures and users' roles, human factors, skills and training, logistics footprint and deployment conditions, hosting facilities, etc. **Quantitative and qualitative metrics** should be defined to evaluate the satisfaction conditions for each of these parameters.
- Capability analysis considers **much more general aspects** than the functional issues: as the client organization, organizational operating principles, roles and responsibilities, nature and infrastructure capacity, safety, human factors and users' skills/training, logistics, acquisition and operation costs, but also the potential complexity and implementational risks.

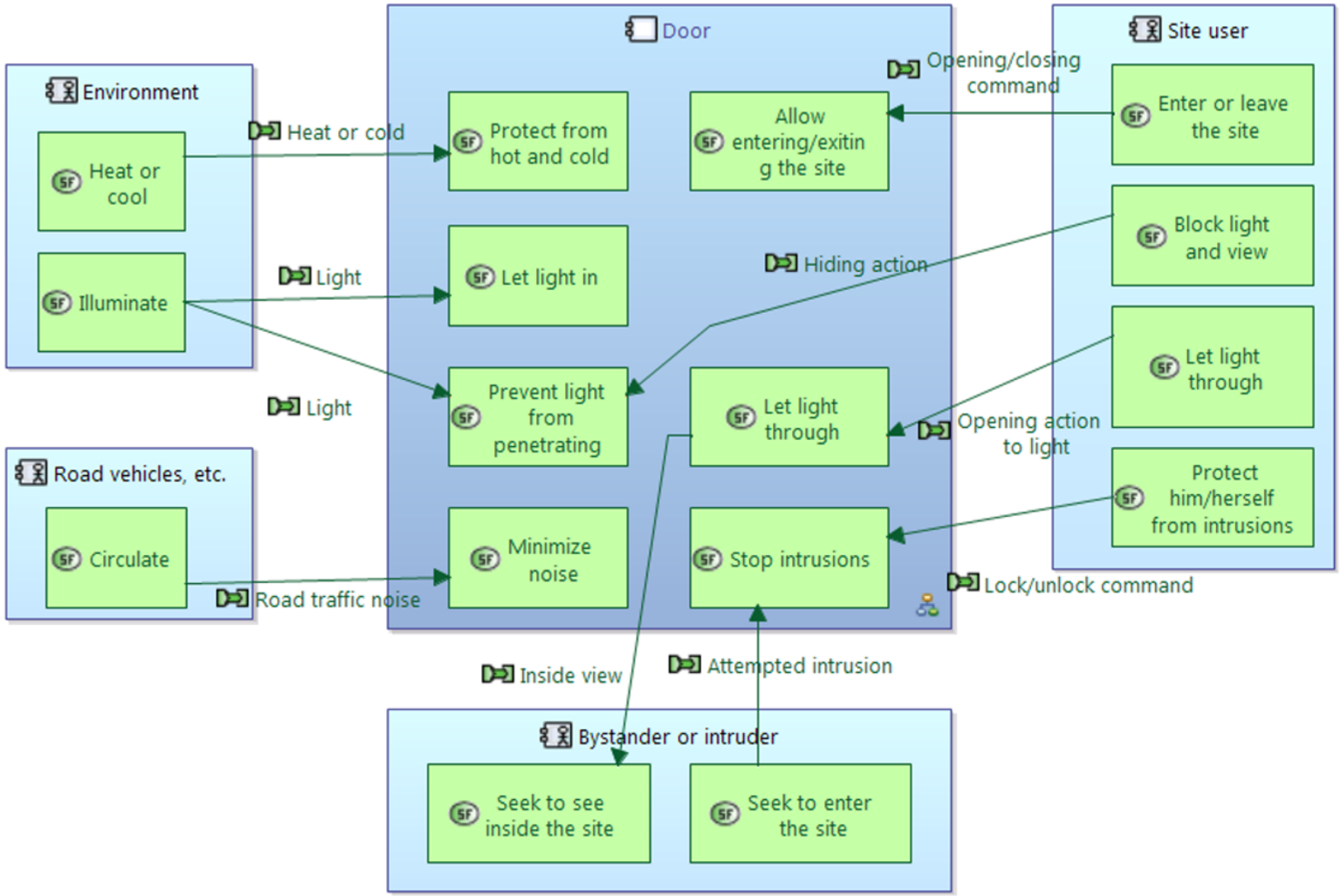
Perform Functional/non-Functional Need Analysis

- The intent is to **formalize the functional needs allocated to the system**, and to identify constraints, namely non-functional, to which it will have to respond through its use under operational conditions
- Assess the operational capabilities to which the system will have to contribute, taking the preliminary capability trade-off analysis (of “system capabilities”) into account - only **needs-related considerations** should be included in this perspective dedicated to the expression of the system needs as required by users
- In the event that actors or external systems are imposed by the client (or the state of the art) and exhibit a complex or critical level of interactions with the system, it is recommended to carry out minimal functional and non-functional analyses for these external systems or actors, and to compare them with the SA, to ensure the compatibility between the two. At this point, an **analysis of available interfaces is desirable**, to verify that planned functionalities and interactions will be possible.
- Another way to address the needs functional analysis consists of implementing each functional requirement into a few functions and exchanges between them (often the verbs of the requirement), the manipulated data (the names) and actors or external systems..



Formalize and Consolidate the System Needs

- The good understanding and consolidation of system needs rely on the three dimensions mentioned earlier, which are the **OA, requirements and the functional analysis** of the system need.
- It is through their *comparison* that consistency and completeness of the system need is assured: **Are all activities and operational processes correctly taken into account in the functional analysis? Are all functional requirements (or even nonfunctional) correctly captured? Is there any incompatibility between them?**
- It may even be the case that the functional needs analysis results in modifying the OA (e.g. changing an operator role for a more secure behavior, or reviewing the distribution of roles should an opportunity for system automation emerge); or alternatively, that the functional analysis reveals an inconsistency or something missing in the requirements.

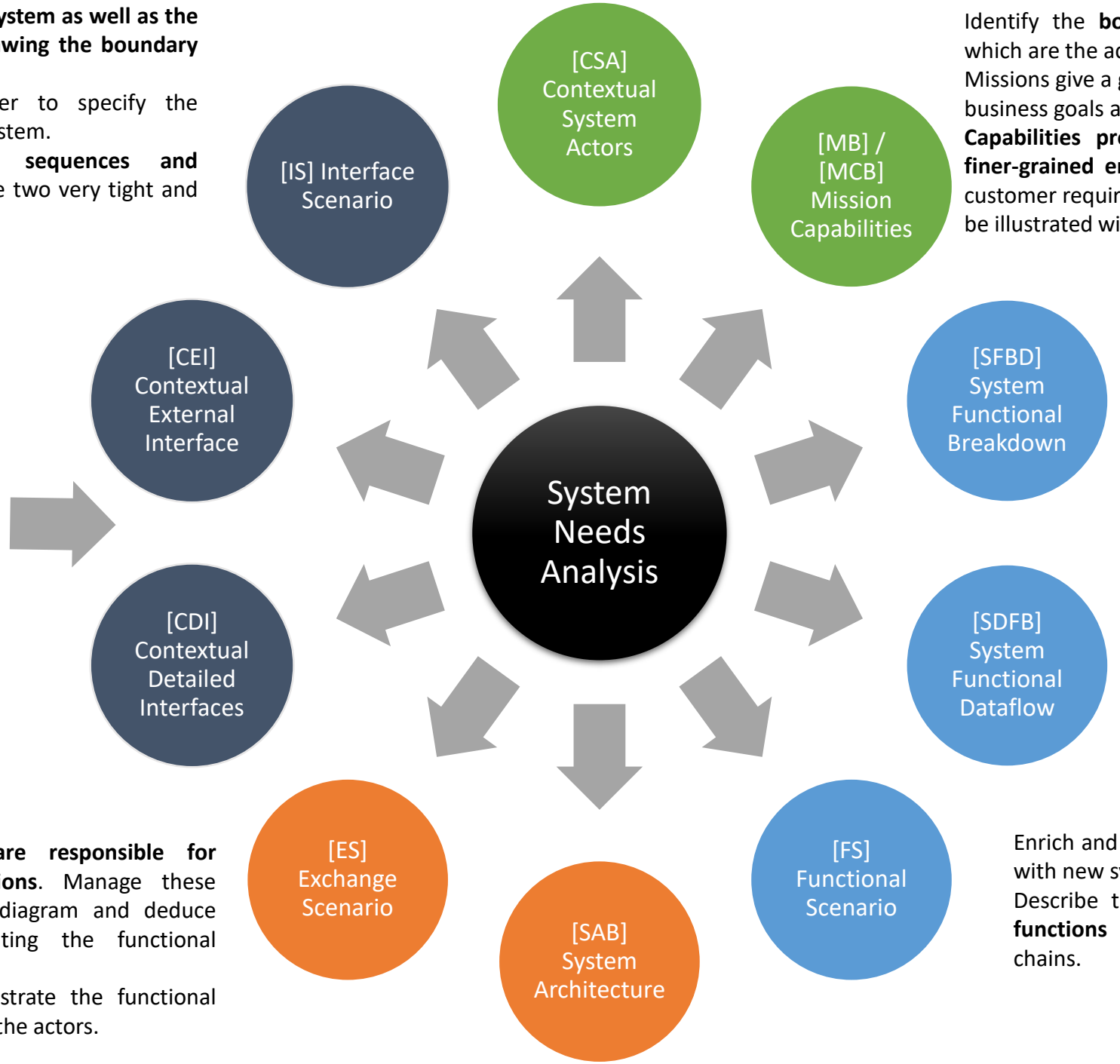

Door
Environment
Heat or cool
Illuminate
Heat or cold
Light
Light
Protect from hot and cold
Let light in
Prevent light from penetrating
Allow entering/exiting the site
Let light through
Minimize noise
Stop intrusions
Opening/closing command
Hiding action
Opening action to light
Lock/unlock command
Road traffic noise
Road vehicles, etc.
Circulate
Site user
Enter or leave the site
Block light and view
Let light through
Protect him/herself from intrusions
Inside view
Attempted intrusion
Bystander or intruder
Seek to see inside the site
Seek to enter the site

SYSTEM DIAGRAMS

Detail the interfaces of the system as well as the ones of the actors, thus drawing the boundary of the system.
Describe scenarios in order to specify the **dynamical behavior** of the system.
Defining the **interaction sequences and identifying the interfaces** are two very tight and iterative activities.

Identify the **boundaries of the system** : who which are the actors, which are their goals?
Missions give a global view upon the system main business goals and usages.
Capabilities provide a more operational and finer-grained enlightenment, directly related to customer requirements. Capabilities are meant to be illustrated with scenarios.

Initialization and automated update of the system analysis according to the breakdown of operational activities.
The initialization and automated updated of the system actors can also be automatically performed from selected operational entities / actors.
The transition tools create a first 1-1 traceability mapping between System Analysis and Operational Analysis. Use dedicated traceability matrices to modify the traceability relationships.

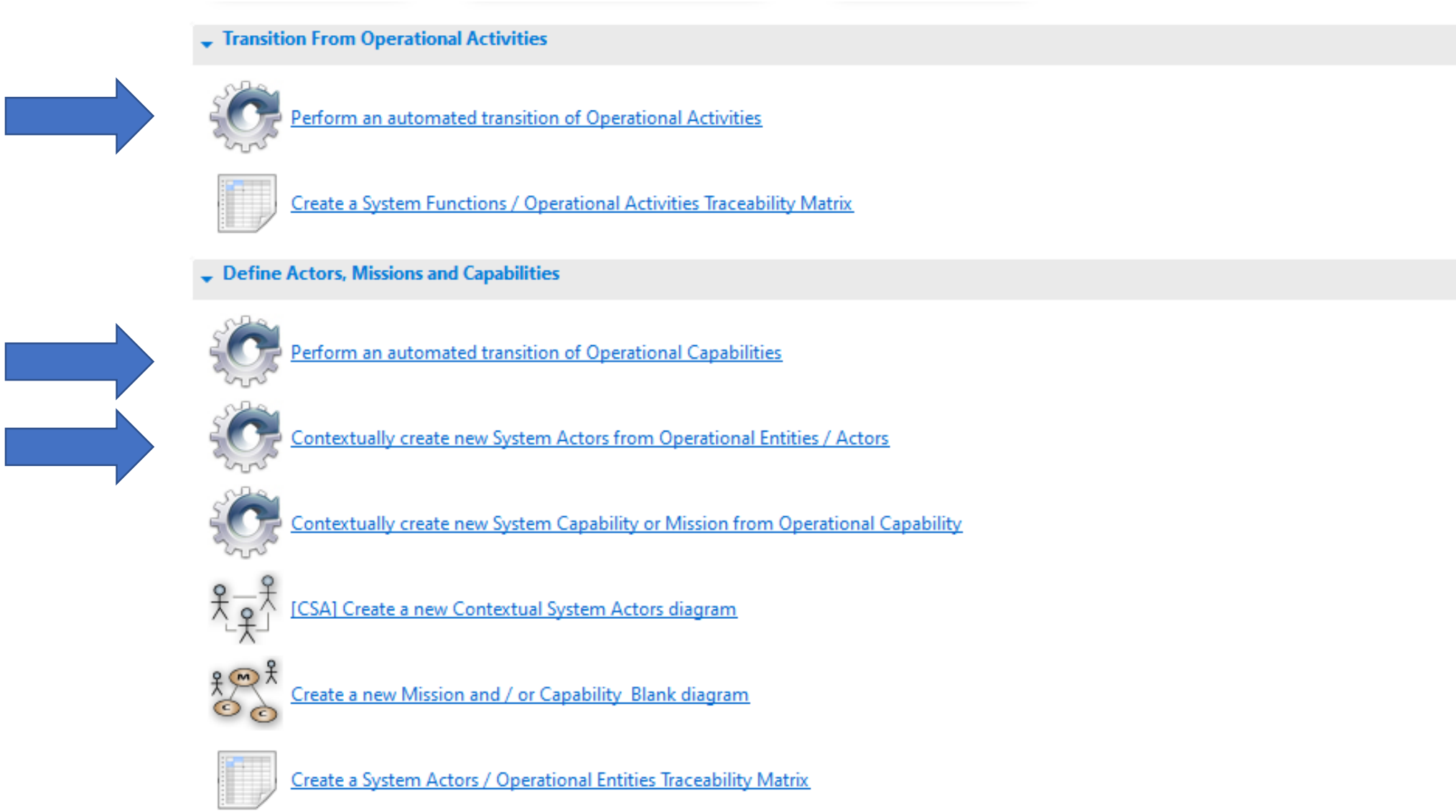


The **system and the actors are responsible for implementing the system functions**. Manage these allocations using an architecture diagram and deduce component exchanges implementing the functional exchanges.
Create dataflows scenarios to illustrate the functional exchanges between the system and the actors.

Enrich and details the functional breakdown with new system functions.
Describe the **data flows between system functions** and identify specific functional chains.

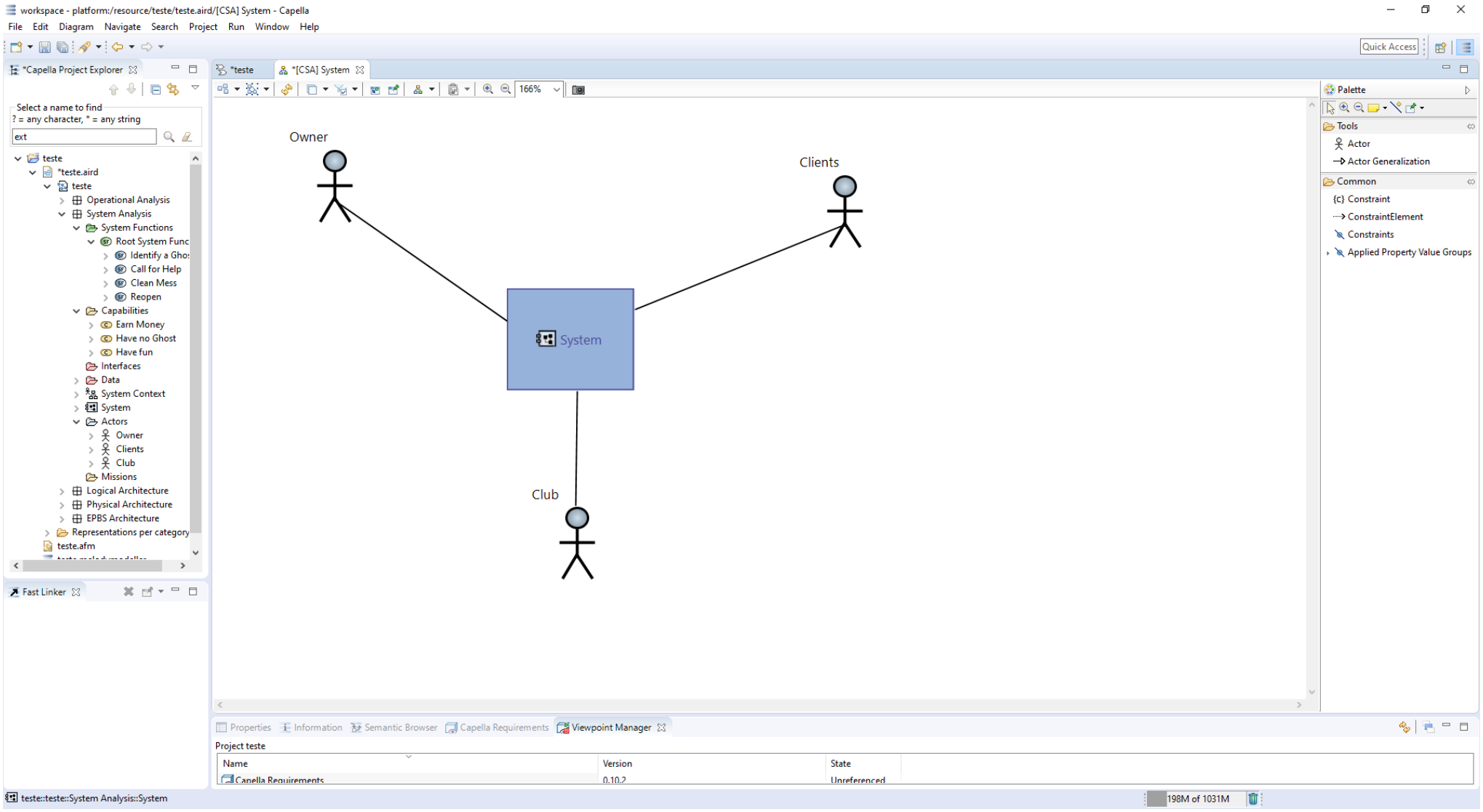
STEP BY STEP EXAMPLE

IMPORT DECISION FROM OA



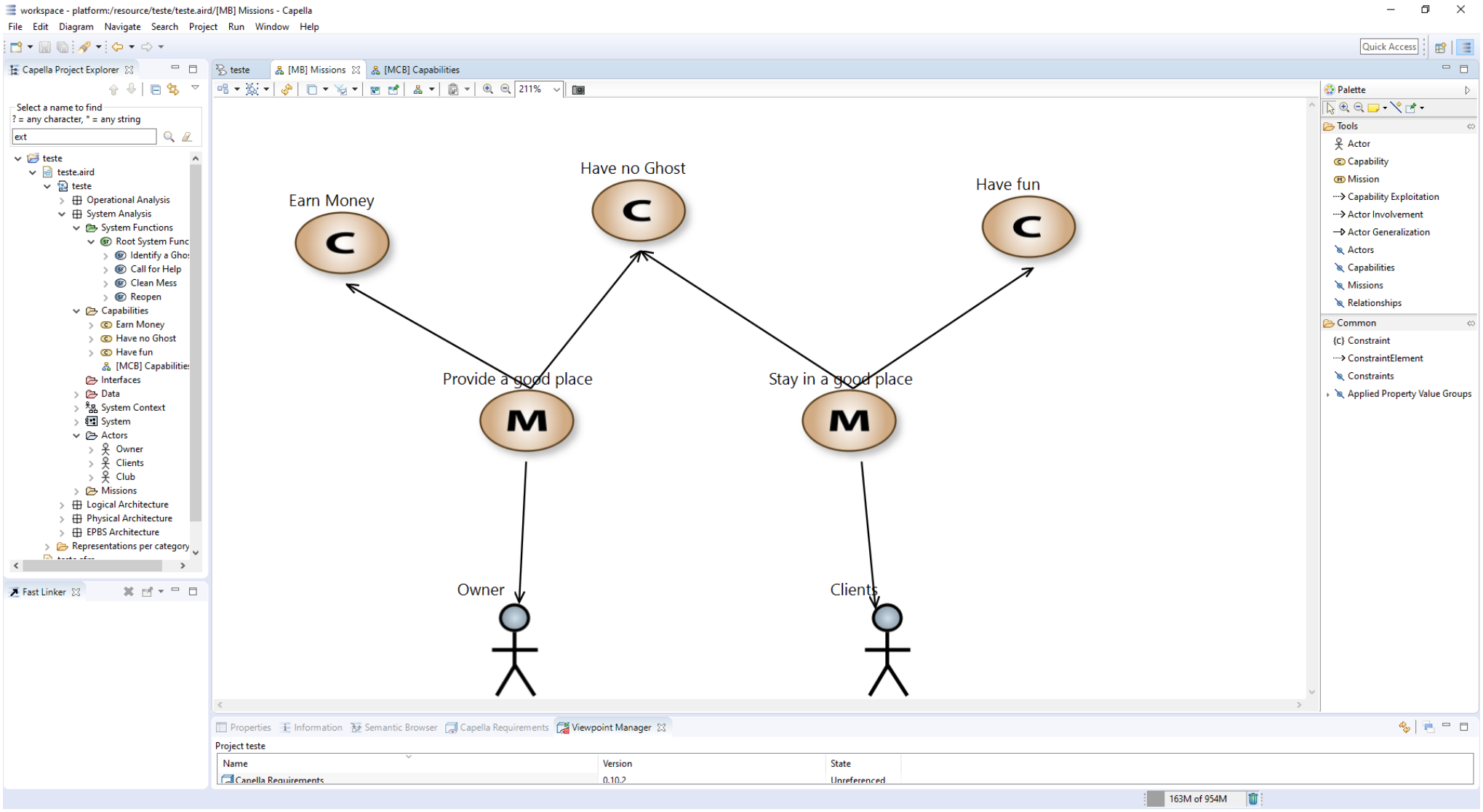
07:16

[CSA] Contextual System Actors



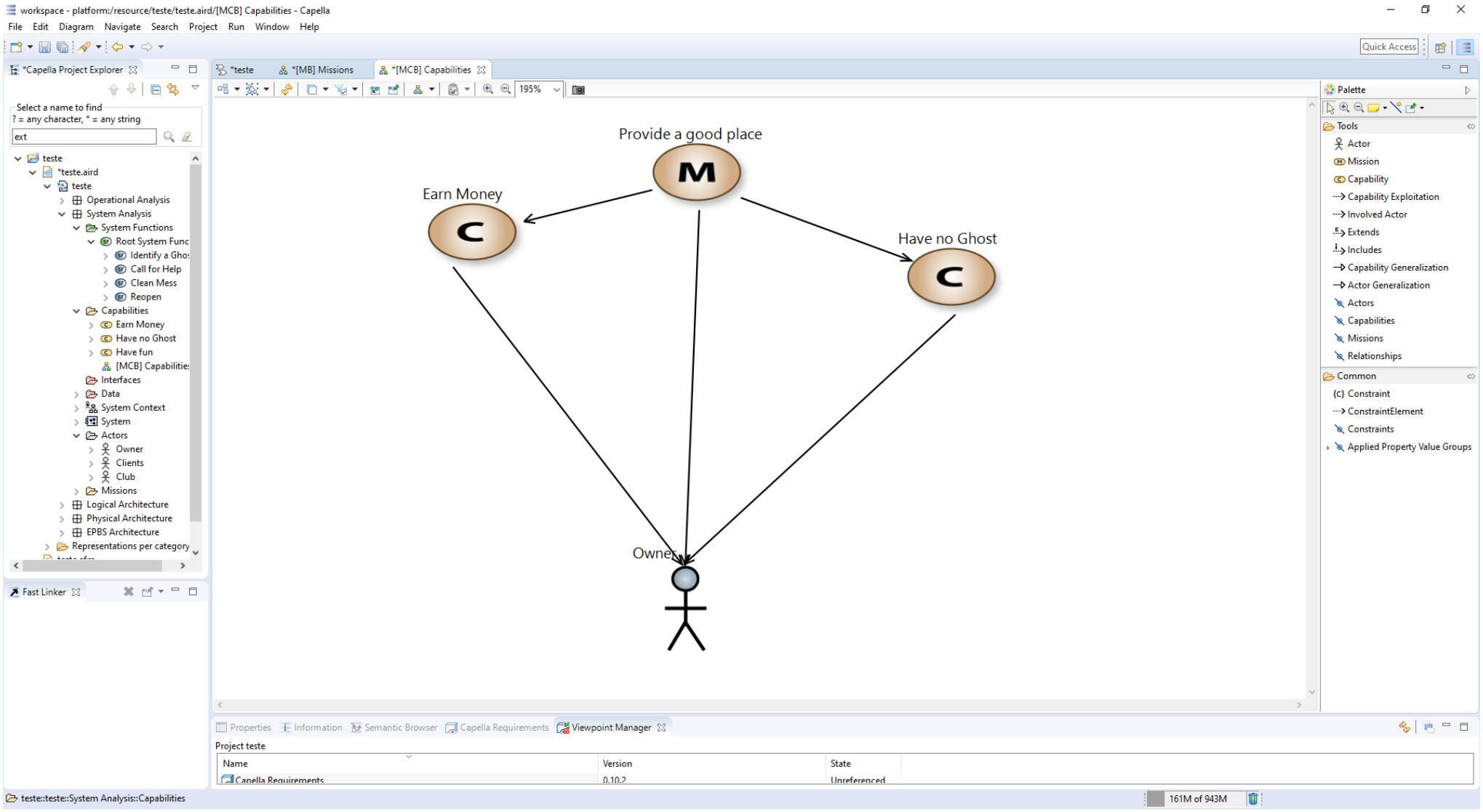
07:16

[MB] Mission (identify the mission related to the capability and the actors)



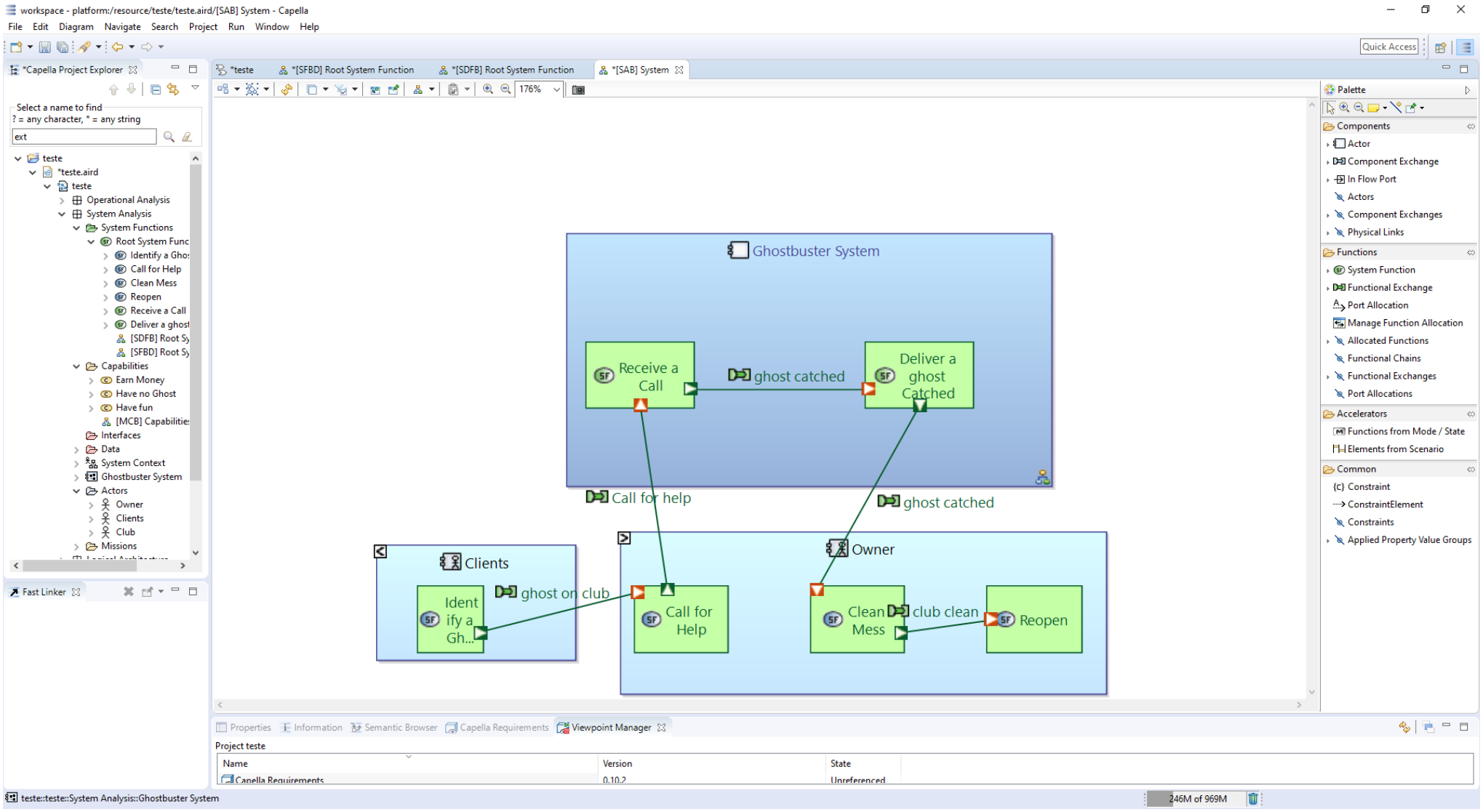
07:16

[MBC] Mission Capabilities

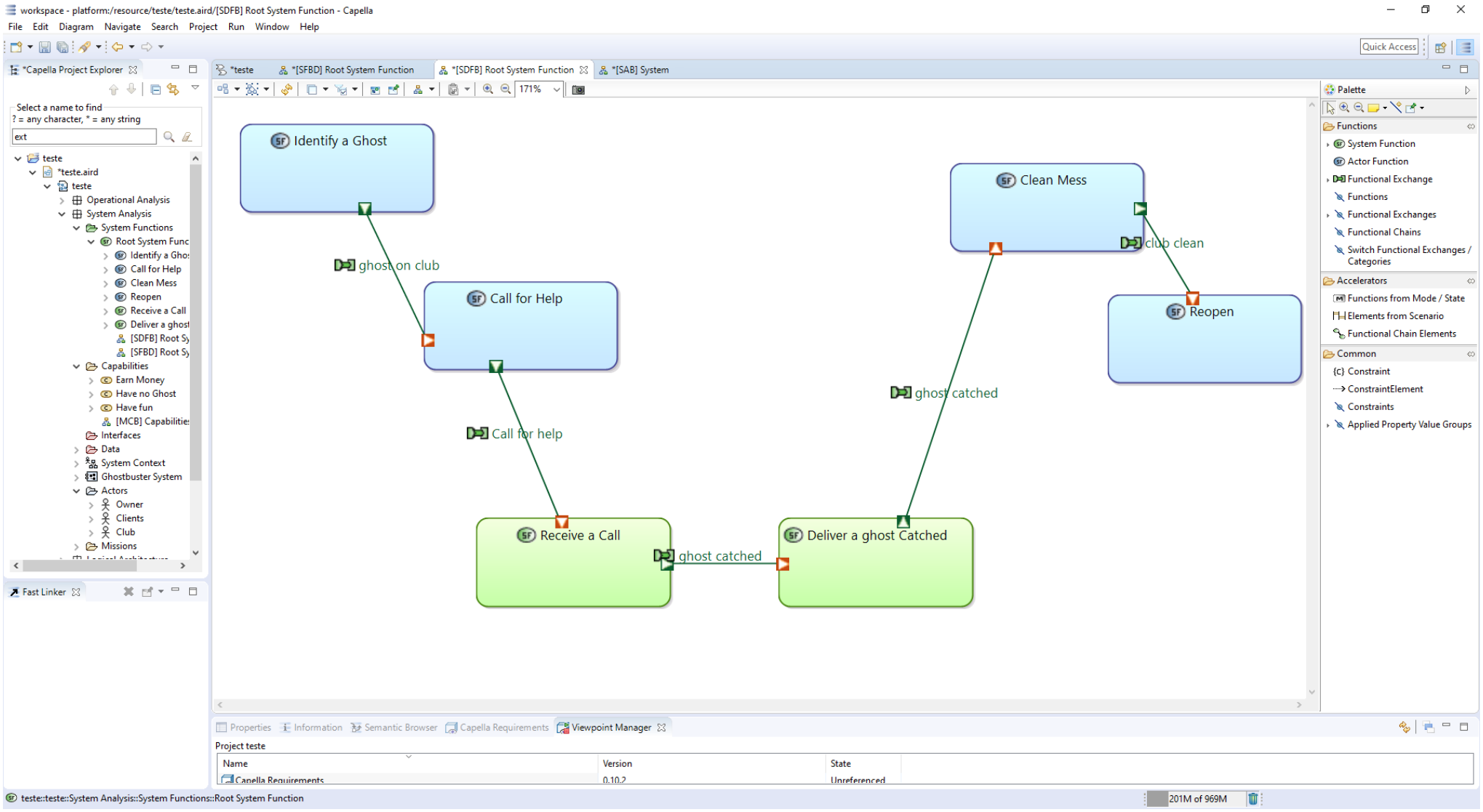


07:16

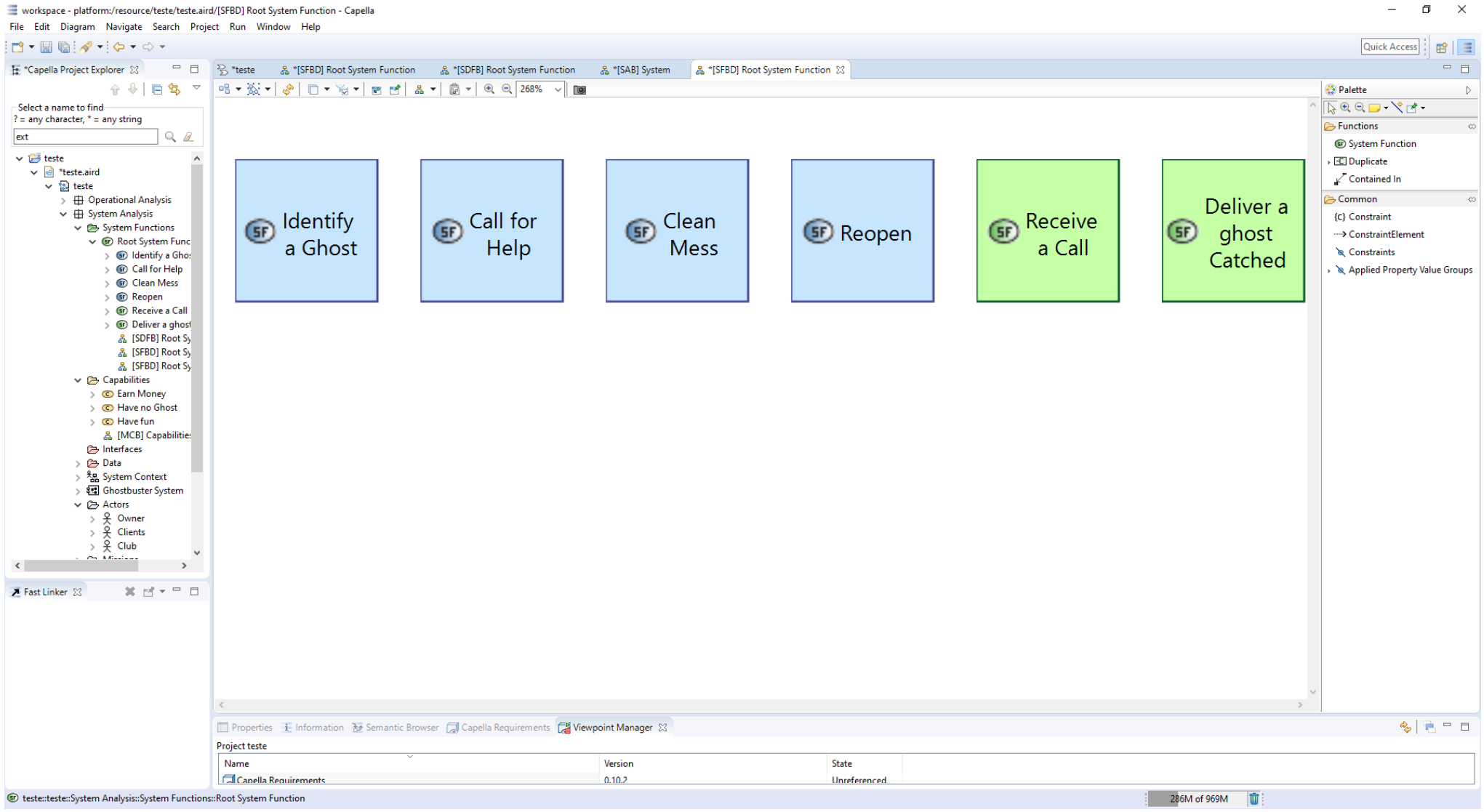
[SAB] System Architecture



[SDFB] System Dataflow

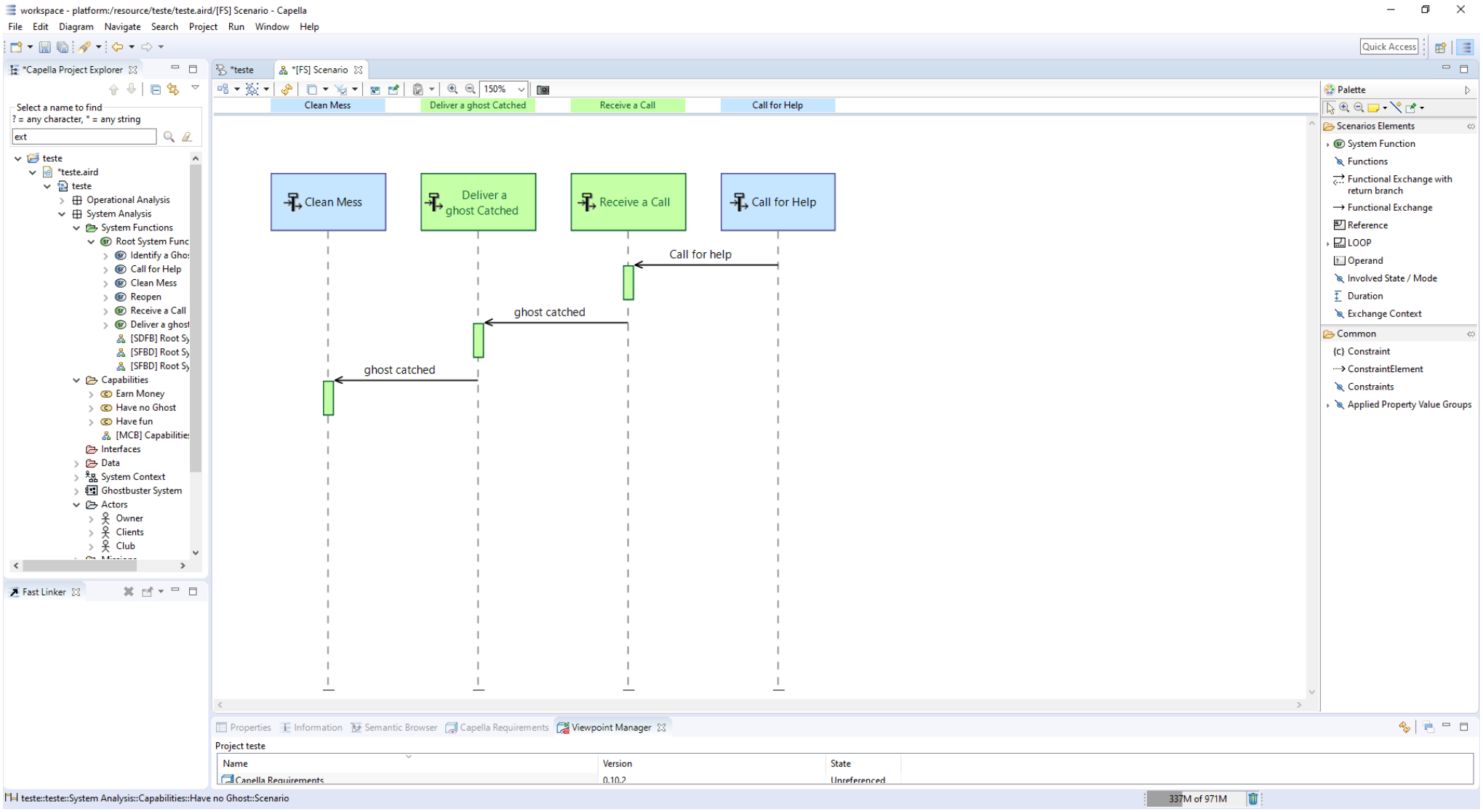


[SFBD] System Functional Breakdown



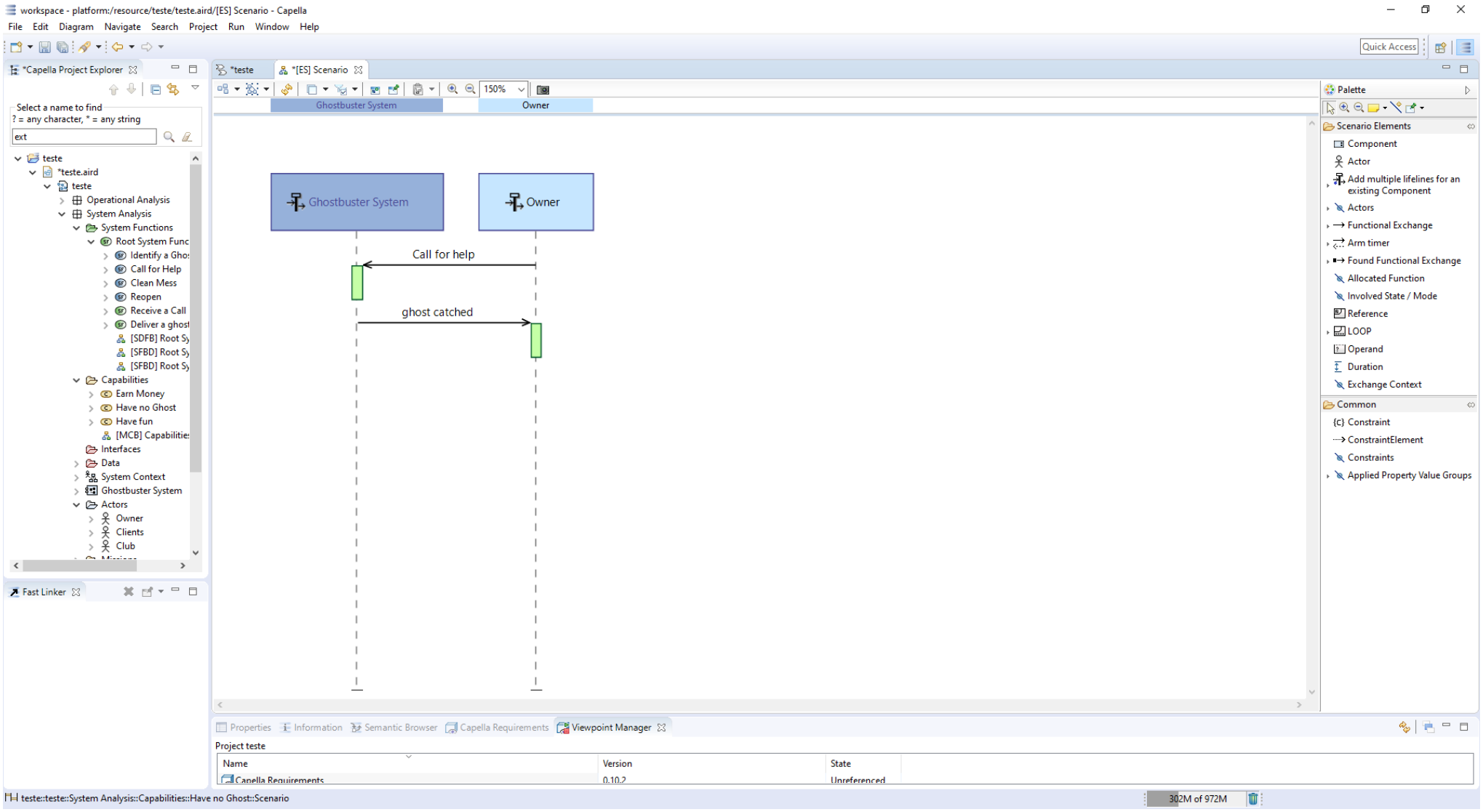
07:16

[FS] Functional Scenario



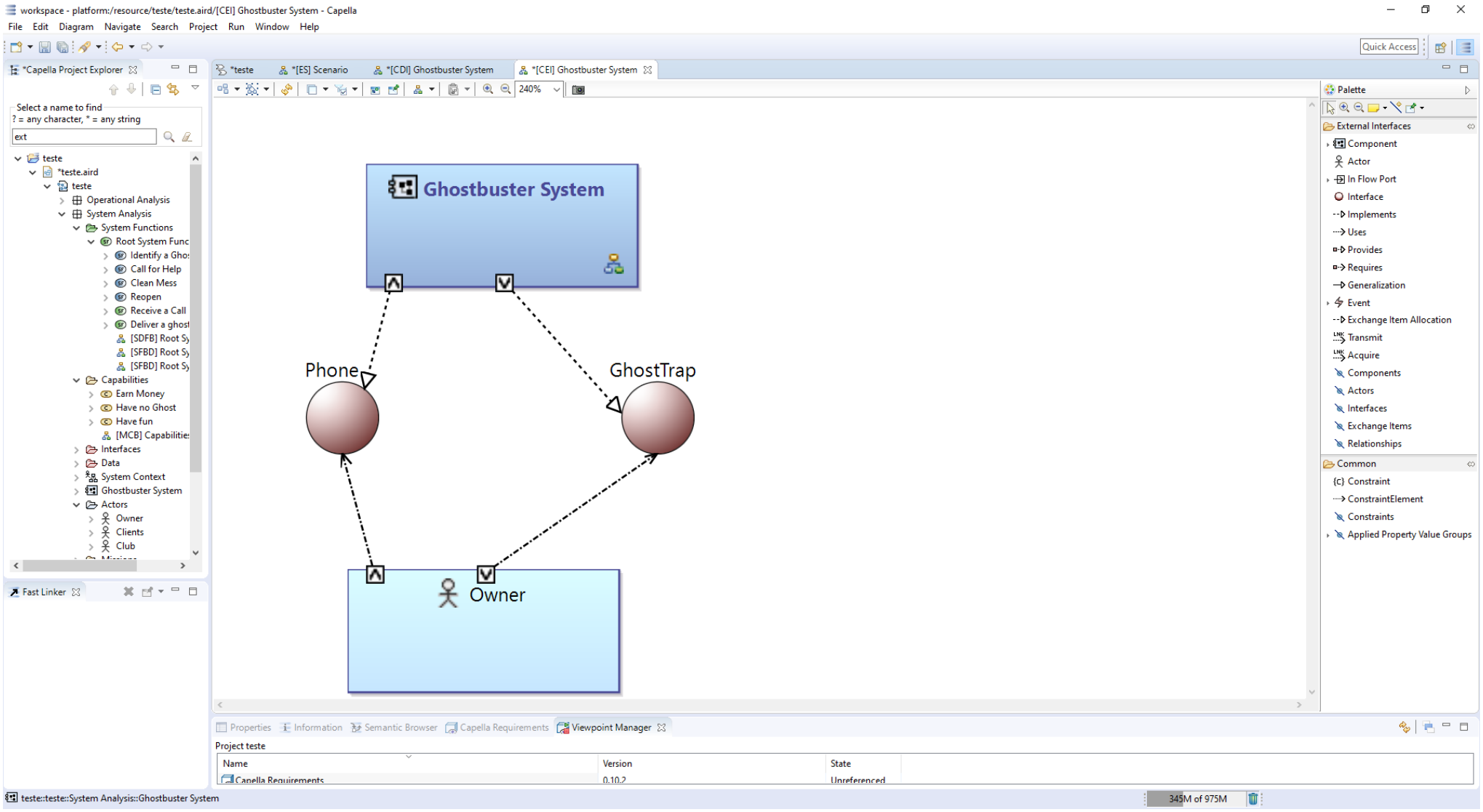
07:16

[ES] Exchange Scenario



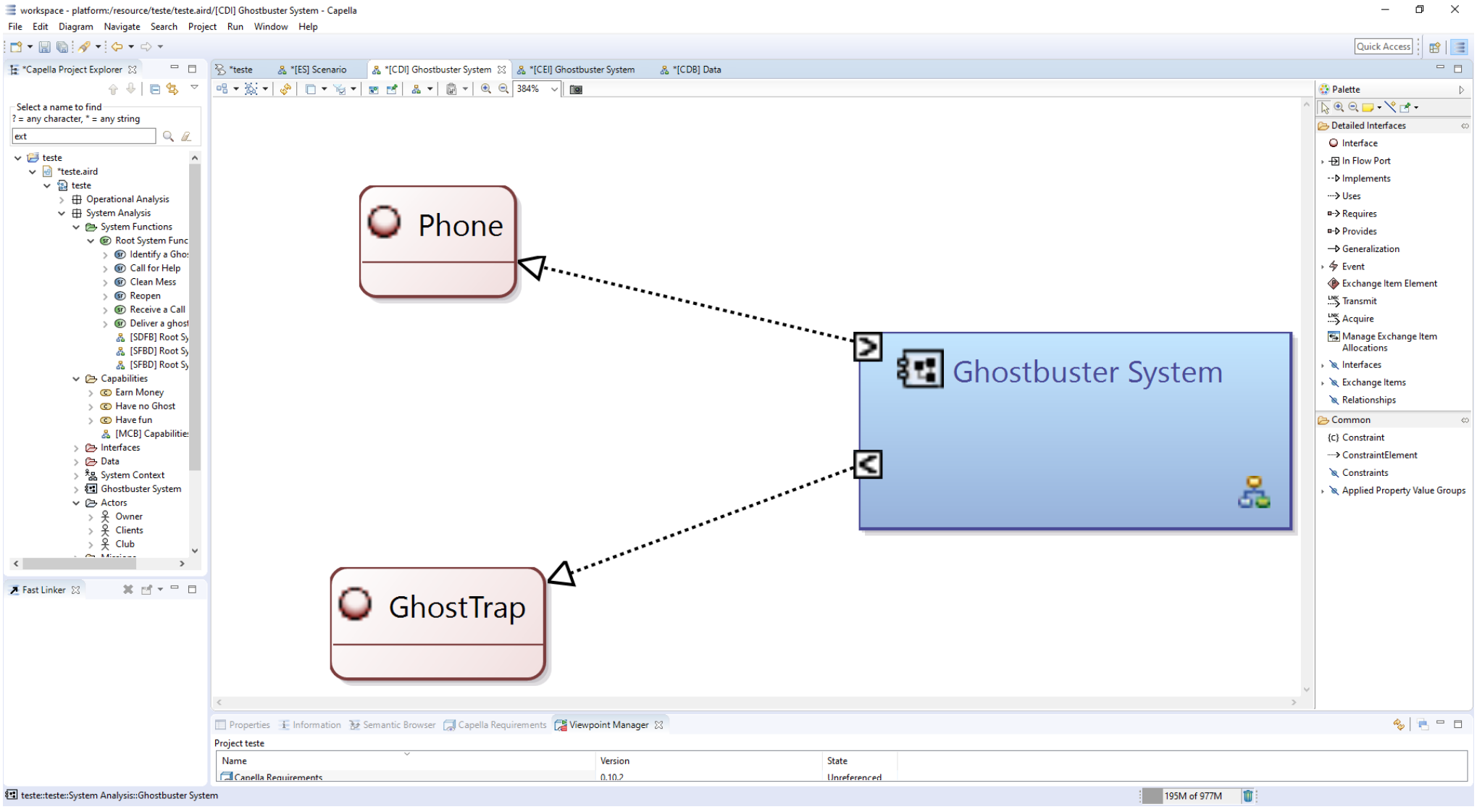
07:16

[CEI] Contextual External Interface



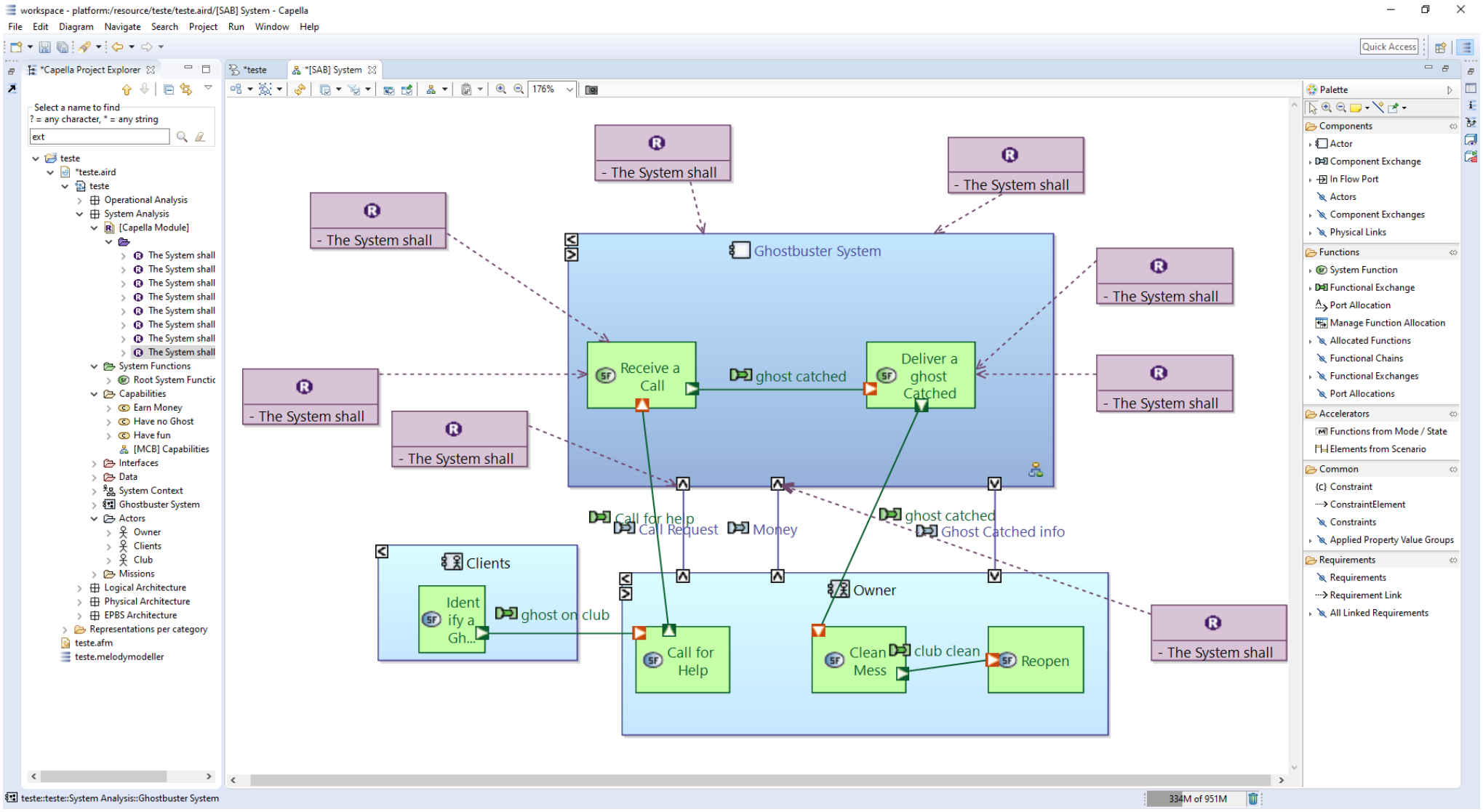
07:16

[CDI] Contextual Detailed Interface

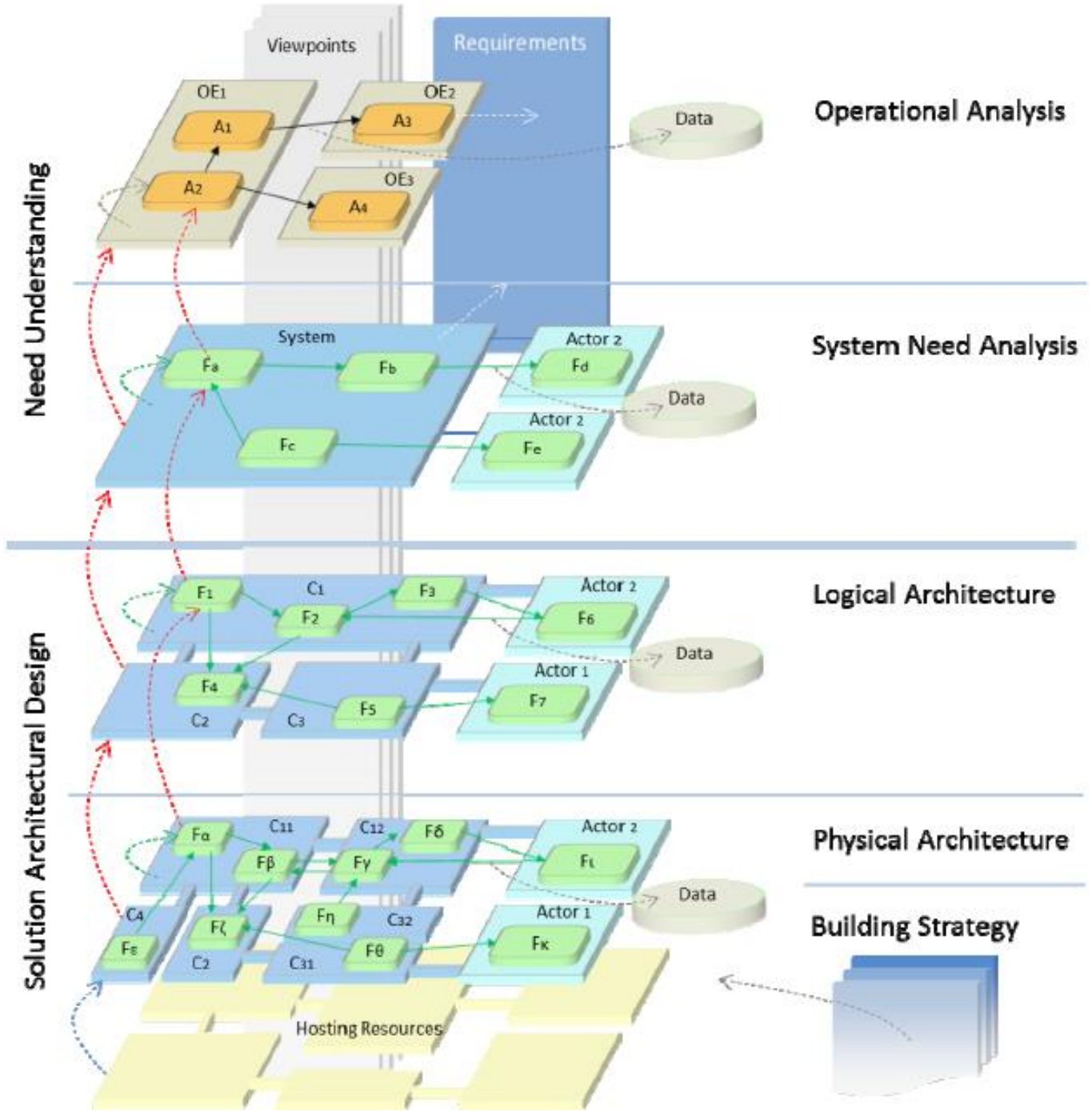


07:16

[SAB] System Architecture with Requirements



LOGICAL ARCHITECTURE



07:16

82

WHAT IS IN THE LOGICAL ARCHITECTURE (LA)?

LOGICAL ARCHITECTURE CONCEPTS

LOGICAL ARCHITECTURE DIAGRAMS

STEP BY STEP EXAMPLE

LOGICAL ARCHITECTURE CONCEPTS

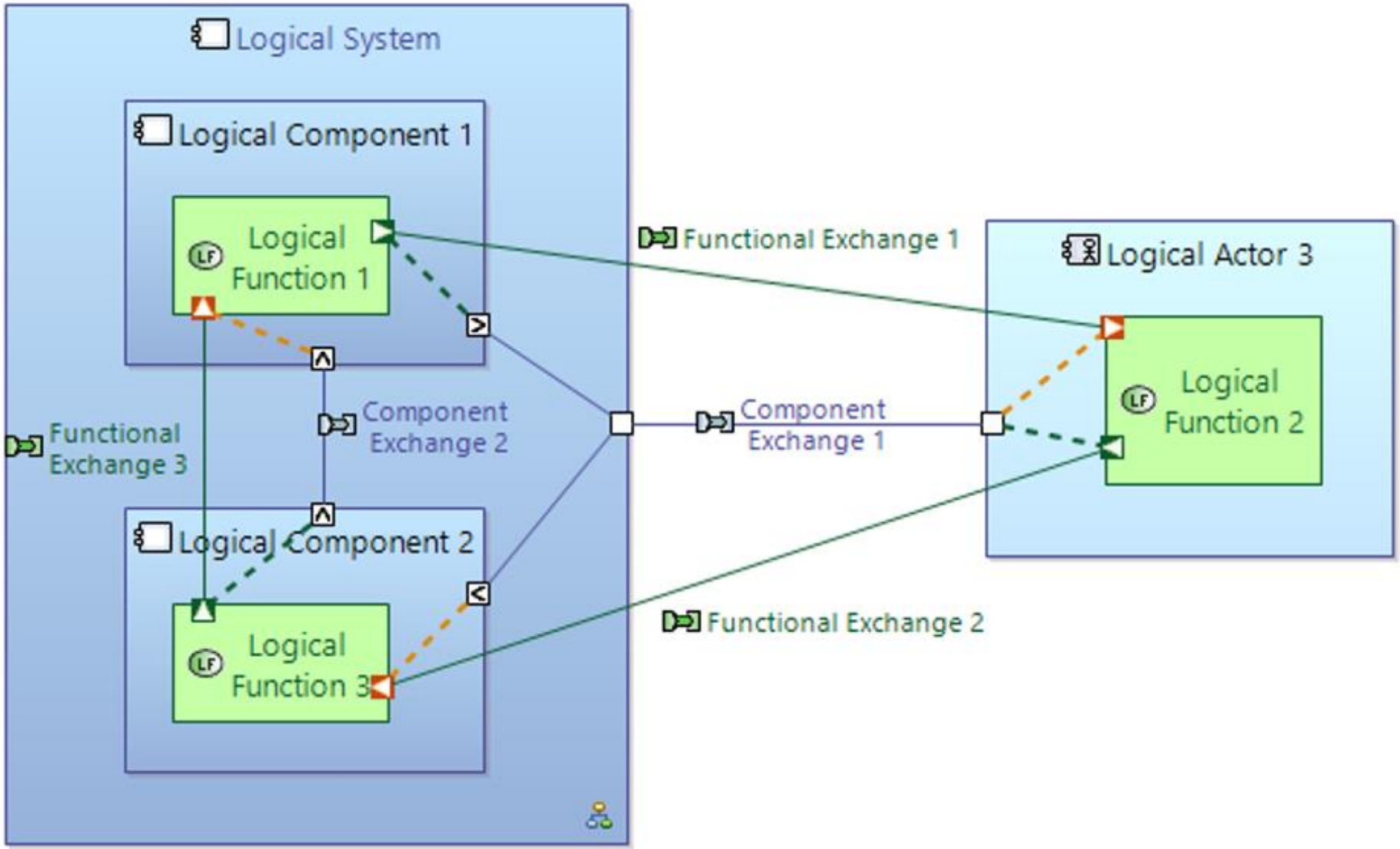
- **Logical Component**: structural element within the System, with **structural Ports** to interact with the other Logical Components and the external Actors. **A Logical Component can have one or more Logical Functions**. It can also be subdivided into Logical subcomponents;

- **Logical Actor**: any element that **is external to the System** (human or non-human) and that interacts with it (for example Pilot, Maintenance operator, etc.).

- **Logical Function**: **behavior or service** provided by a Logical Component or by a Logical Actor. A **Logical Function has Function Ports that allow it to communicate with the other Logical Functions**. A Logical Function can be subdivided into Logical subfunctions;

- **Functional Exchange**: a ***unidirectional*** exchange of information or matter between two Logical Functions, linking two Function Ports;

- **Component Exchange**: **connection between the Logical Components** and/or the Logical Actors, allowing **circulation of the Functional Exchanges**;


Logical System
Logical Component 1
Logical Function 1
Component Exchange 2
Functional Exchange 3
Logical Component 2
Logical Function 3
Functional Exchange 1
Component Exchange 1
Logical Actor 3
Logical Function 2
Functional Exchange 2

07:16

- **Logical Scenario**: **dynamic occurrence describing the interactions between Logical Components and Logical Actors** in the context of a Capability. It is commonly represented as a sequence diagram, with the vertical axis representing the time axis;

- **Functional Chain**: element of the model that enables a **specific path to be designated among all possible paths** (using certain Functions and Functional Exchanges). This is particularly useful for assigning constraints (latency, criticality, etc.), as well as organizing tests;

WHAT IS IN THE LOGICAL ARCHITECTURE (LA)?

07:16

Logical Architecture

"How the system will work to meet expectations"

"How the system will work to fulfill expectations"

- In response to the need expressed by the two previous perspectives, it enables the **first major choices of solution** design, first via an internal functional analysis of the system: it describes the functions to be performed and assembled in order to implement the service functions identified in the previous phase. It continues with the **identification of the operational components** implementing these solution functions, integrating the non-functional constraints that we chose to be addressed at this level.

07:16

- The level of Logical Architecture aims to identify **Logical Components inside the System** (“how the system will work to fulfill expectations”), their relations and their content, **independently of any considerations of technology or implementation**.
- Next an **internal functional analysis** of the system must be carried out: **the subfunctions required to carry out the System Functions** chosen during the previous phase must be identified; next, **a split into Logical Components to which these internal subfunctions will be allocated must be determined**, all the while integrating the nonfunctional constraints that have been chosen for processing at this level

- The definition of the LA (an activity often – and wrongly – designated “logical architecture” for convenience) **consists mainly of a comparison between the needs expressed in previous perspectives, a functional analysis** describing the system behavior chosen to satisfy requirements, and a **structural analysis intended to identify the components that will constitute the system**, taking the chosen constraints and structuring principles into account.
- The LA is therefore a **first general vision**, moderately detailed, somehow an abstraction, of what the architecture of the system will be

The main activities to be undertaken for the definition of the logical principle architecture are as follows:

- to define the **factors** impacting the architecture and analysis viewpoints;
- to define the principles underlying the **system behavior**;
- to build **component-based system** structuring alternatives;
- to select the **architecture alternative** offering the best compromise.

Definition of the factors impacting the architecture and analysis viewpoints

- Any **properly designed architecture satisfies several expectations and constraints of various kinds**, which constrain and influence or even direct its definition, and whose satisfaction should be verified as early as possible to minimize possible subsequent resumption costs.

- These factors that constrain the architecture depend largely on each domain, and each profession. As examples we mention: delivered services and costs of course, expected performance, safety of operations, privacy, ease of maintenance, life duration, energy or logistical footprint, availability, product policy, scalability, but also more “aesthetic” considerations such as customer satisfaction.

07:16

- **For each factor previously identified**, the associated constraints (especially nonfunctional and performance ones), which can be applied to the needs and the solution, **must be expressed and quantified by metrics**; **each candidate architecture will be analyzed according to this viewpoint**, to verify that good practice is correctly followed.

- These decisions reflect know-how, the craft, in addition to the creativity of the engineering team, and will guide the emergence of different alternatives as well as their comparison.
- **Imposed factors and design choices must be categorized by importance or priority**, in order to be able to arbitrate between them when they result in antagonistic properties, or when certain constraints will have to be released to find an acceptable compromise.

07:16

- *In the case of the traffic control system, the **first impact factor is obviously the safety of goods and people**. An additional factor involves **system operators**, their training and their required skills, the scope of their responsibility and the role that must be assigned to them. We should also take into account factors such as **environmental conditions, life duration, constraints on logistics and maintenance**.*
- *In the case of the traffic control system, let us mention the required reliability rate and the system failure probability, the capability to be able to operate in the event of partial failure of certain subsystems; the maximal eligible number of operators; extreme temperature ranges, humidity, resistance to possible salt sprays; etc.*

Definition of the behavior principles of the system

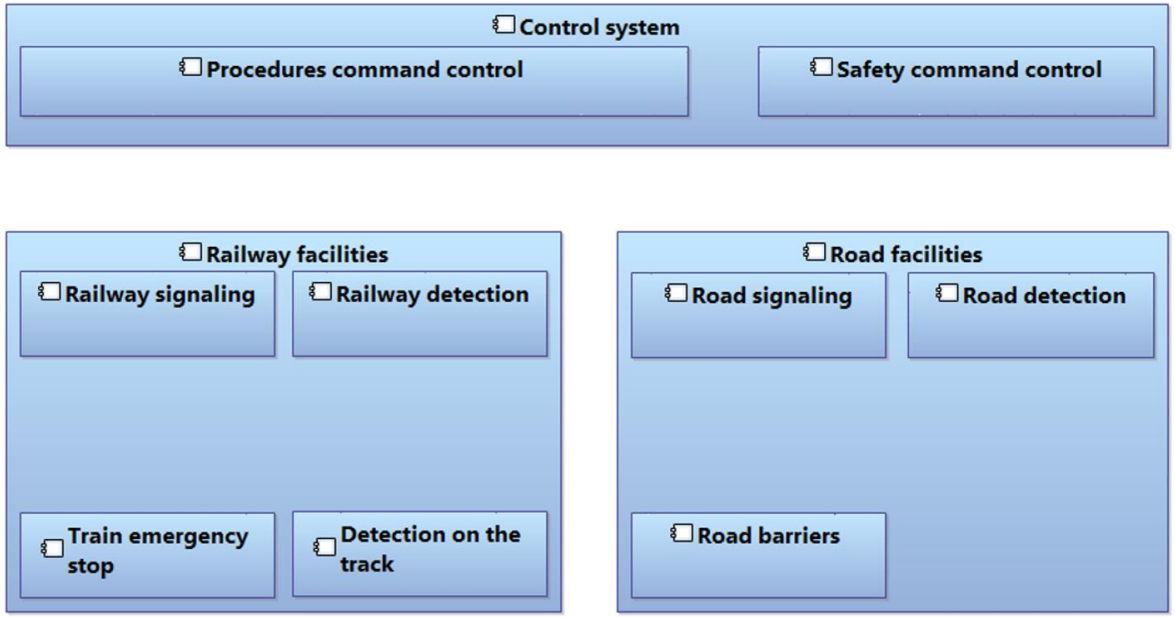
- The objective is to formalize the principles of the desired behavior of the system, and to non-functional, to which it has the responsibility to respond during its operation under operational conditions.
- *A common mistake consists of considering the behavior of the solution as a simple refinement of the previous functional expression of need at a finer level of detail. The solution design is much more than that: it is a take into account the constraints, namely "creative" definition effort of a behavior that meets the need (and that does not refine it),* detailing the processes and steps starting from the solicitations of the system, up to the provision of services, results or outputs, taking into account design decisions, mainly guided by the factors and constraints identified previously.

07:16

- 1 – identify and formalize **need items** captured. (tracing to SA)
- 2 – search for **possible functions** already in the LA that could also play a role to solve the need. (minimize functions)
- 3 – verify function **boundaries** to achieve what is expected of it.
  - Scenarios / chains will add light to design decisions or to the choice of product line.
- 4 – build a complete and coherent **global description** using the behavioral elements (scenarios/state machines)

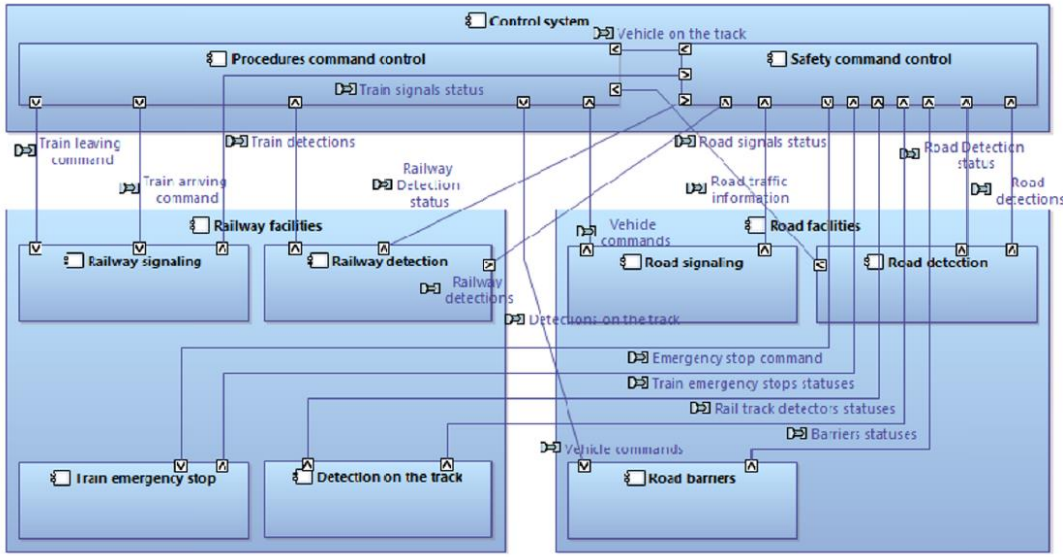
Construction of component-based system structuring alternatives

- This step should reveal a number of principle solutions, **describing the preliminary structure of the system**, built on the basis of the previous behavior, incorporating both non-functional associated constraints and the factors and design choices underlying it.
- The system is **broken down into principle components called logical components**. The term “component” is understood here in the general sense, as a constituent of the system at this level; it can later be implemented as a subsystem (or several), equipment, one or more mechanical parts or assemblies, one or more electronic cards, a software program itself eventually distributed or even a human contributor.

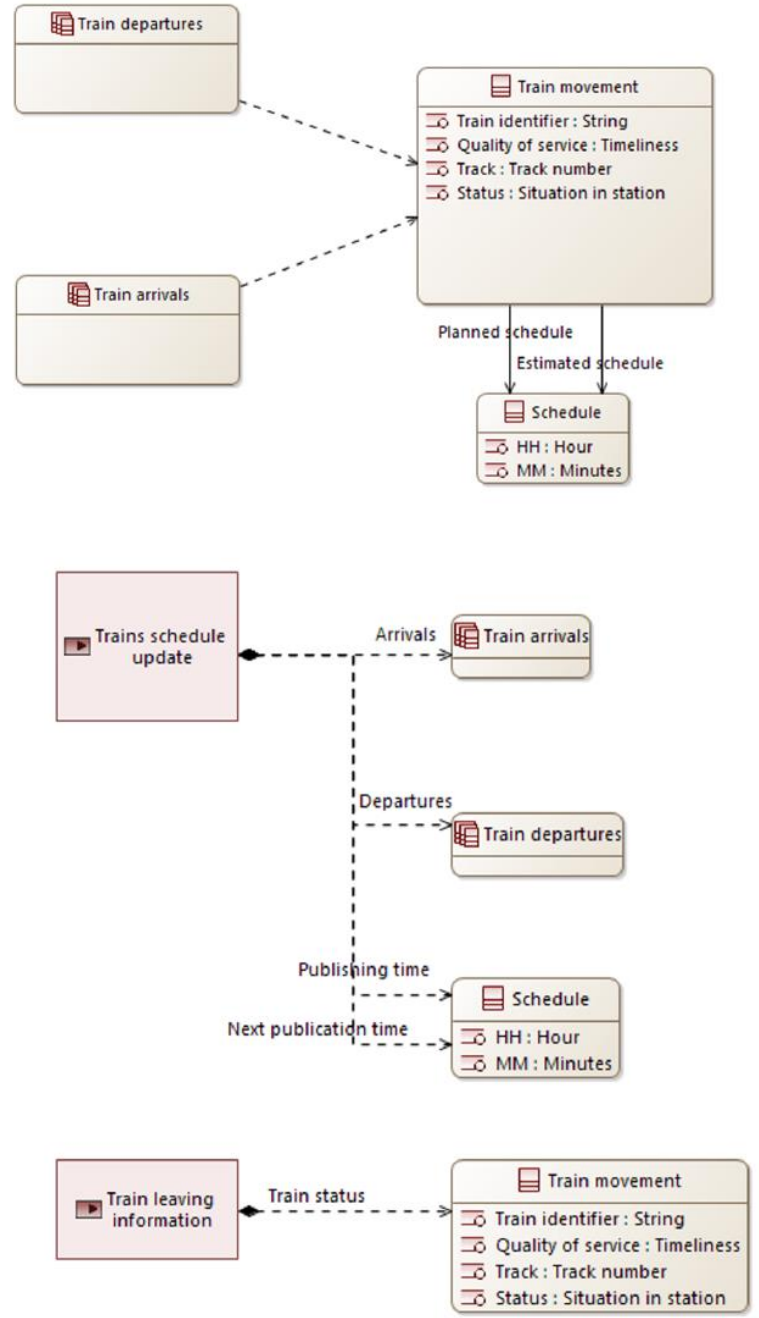


07:16

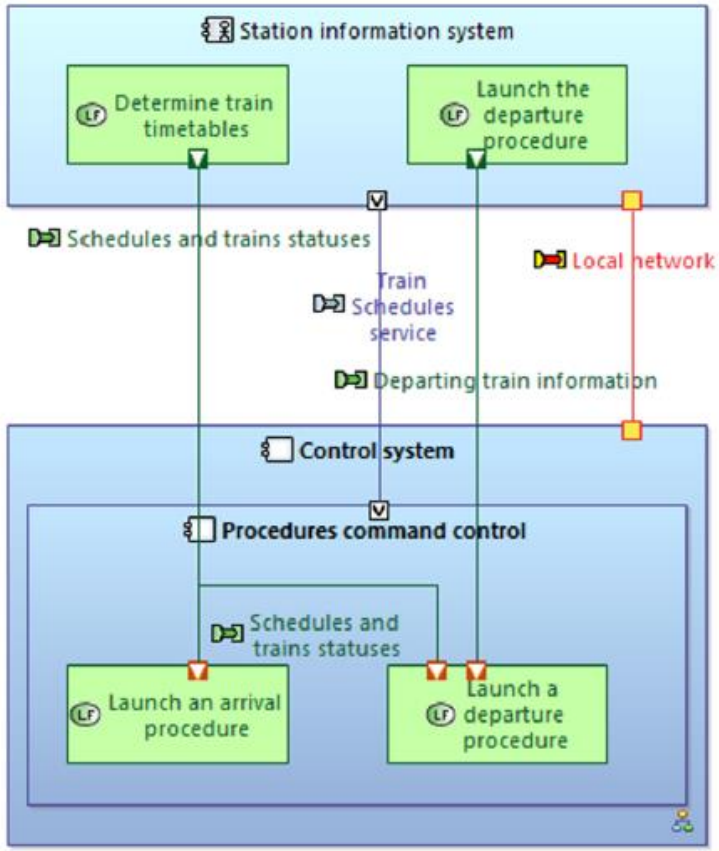
- The component building process consists of **grouping together or segregating the behavior functions** previously defined, according to the constraints and criteria imposed, in grouping sets that thus constitute the components. These latter can themselves be structured by subcomponents, according to the same types of criteria if necessary.
- It is recommended to submit each choice of functional grouping to the **multi-viewpoint analysis**

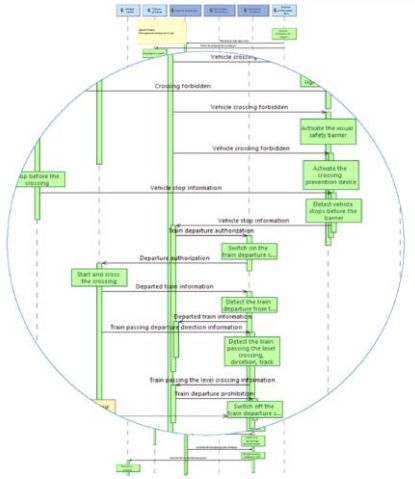


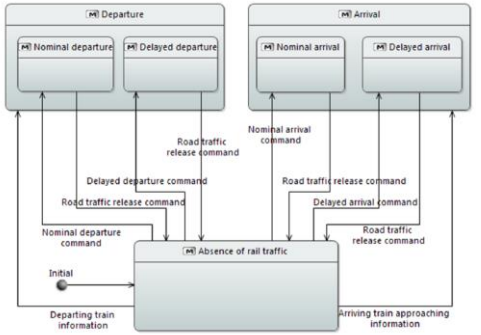
- The (preliminary) definition of interfaces between components (or with external actors) can be done at this level (or be postponed until the definition of the physical architecture): they are built based on the **functional exchanges linking the functions allocated to these components or actors, and exchanges data (and exchange elements) that these exchanges convey**; data and exchanges are mainly grouped according to semantic proximity or usage considerations.
- The actual exchanges between components are also achieved by way of grouping functional exchanges; combined with the capability to hide subcomponents in order to consider those of first level only, this also constitutes a level of synthesis or even of abstraction able to hide the complexity of functional exchanges, and to reason on several levels of detail.



- This static definition of interfaces most often must be accompanied by a **dynamic definition**, by creating scenarios at the boundaries of the components, and if necessary, state and modes machines associated with each contributor to exchanges and managing this dynamics of interfaces.
- Furthermore, **states and modes can be defined** and allocated to components, based on those implemented at the system level in the previous behavioral functional analysis, and consistent with them.



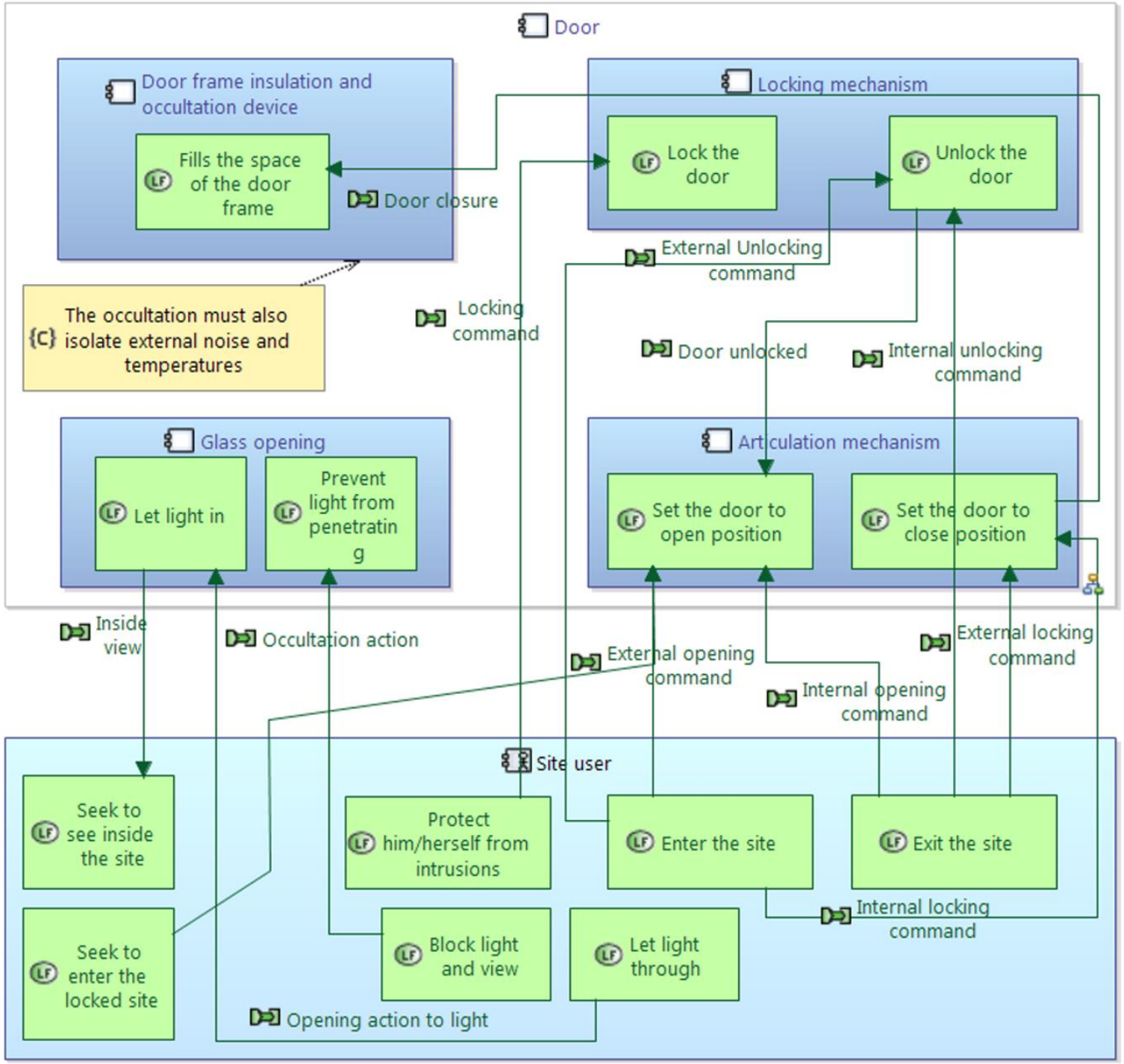




Selection of the architecture alternative offering the best trade-off

- The purpose of this activity is to find among previous candidate architectures the one that **represents the best trade-off** with respect to all viewpoints under consideration, and to justify its compliance to the need.
- Each alternative has in principle been evaluated based on the major viewpoints impacting it – and their relative importance – during its definition; the inadmissible nonconformities have been eliminated, but as the evaluation is rarely binary, the point is therefore now to **compare the "merits" of each candidate in a multi-criteria quantitative analysis**, of which previously identified viewpoint analyses, priorities and metrics are key elements.

07:16


Door
Door frame insulation and occultation device
Fills the space of the door frame
Door closure
The occultation must also isolate external noise and temperatures
Locking mechanism
Lock the door
Unlock the door
External Unlocking command
Locking command
Door unlocked
Internal unlocking command
Glass opening
Let light in
Prevent light from penetratin g
Articulation mechanism
Set the door to open position
Set the door to close position
Inside view
Occultation action
External opening command
Internal opening command
External locking command
Site user
Seek to see inside the site
Protect him/herself from intrusions
Enter the site
Exit the site
Internal locking command
Seek to enter the locked site
Block light and view
Let light through
Opening action to light

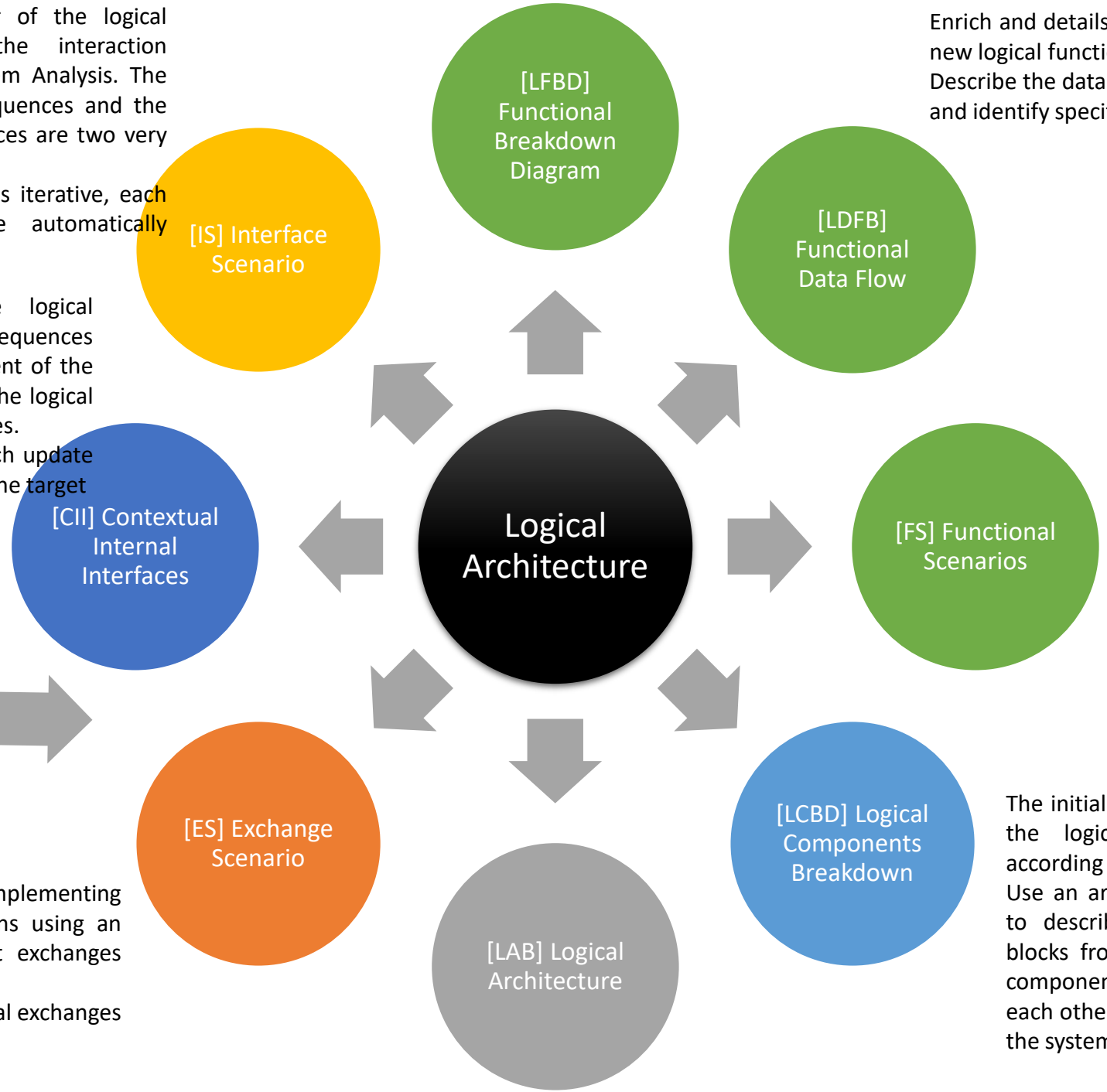
LOGICAL ARCHITECTURE DIAGRAMS

Specify the dynamical behaviour of the logical components by completing the interaction sequences coming from the System Analysis. The enrichment of the interaction sequences and the identification of the logical interfaces are two very tight and iterative activities.
The scenario refinement process is iterative, each update on a source can be automatically propagated to the target.

Specify the dynamical behaviour of the logical components by completing the interaction sequences coming from the System Analysis. The enrichment of the interaction sequences and the identification of the logical interfaces are two very tight and iterative activities.
The scenario refinement process is iterative, each update on a source can be automatically propagated to the target

Initialization and automated update of the logical functions according to the system functions
The transition tools create a first 1-1 traceability mapping between Logical Architecture and System Analysis. Use dedicated traceability matrices to modify the traceability relationships.

The logical components are responsible for implementing the logical functions. Manage these allocations using an architecture diagram and deduce component exchanges implementing the functional exchanges.
Create dataflows scenarios to illustrate functional exchanges between the components.

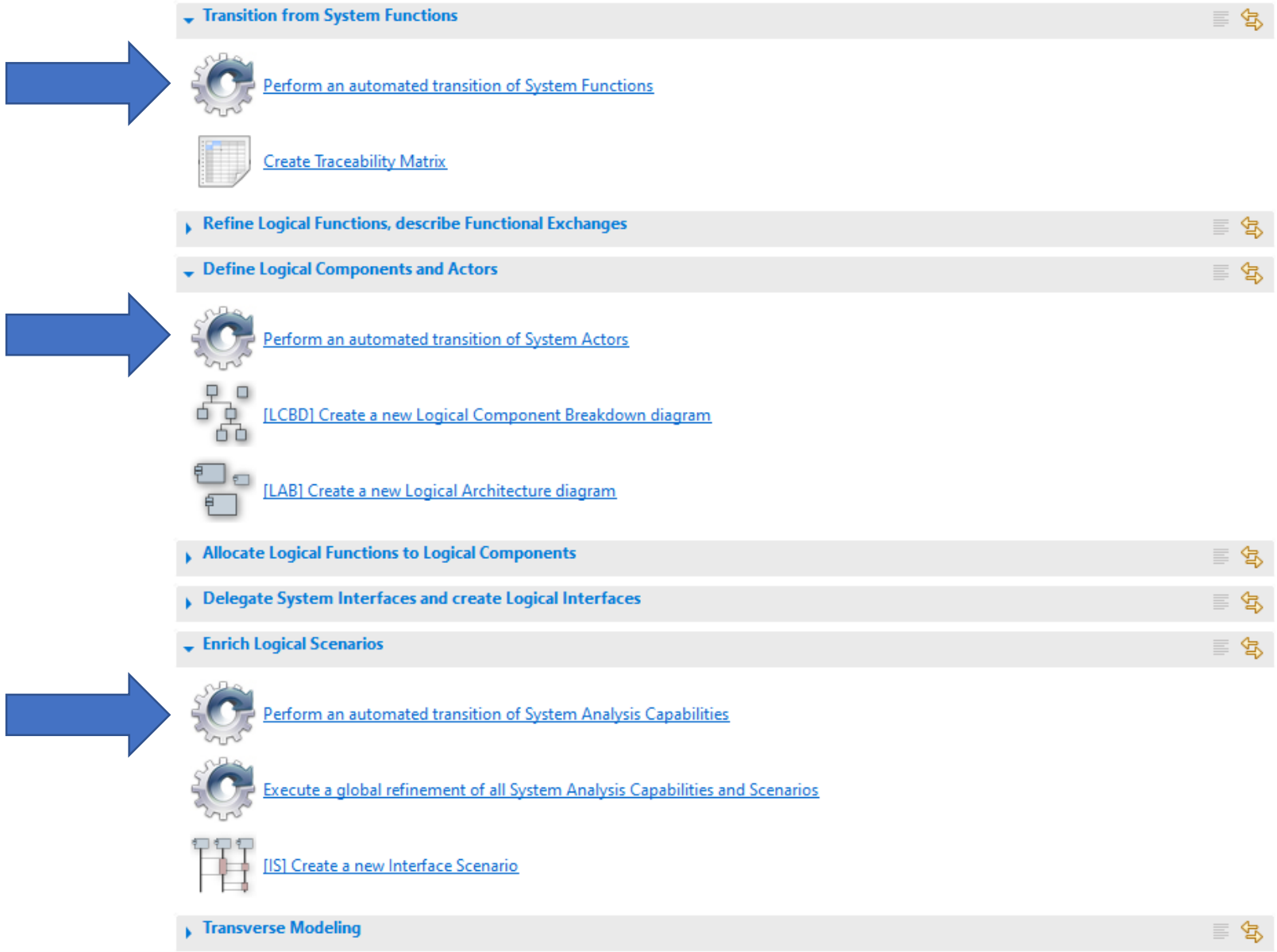


Enrich and details the functional breakdown with new logical functions.
Describe the data flows between logical functions and identify specific functional chains.

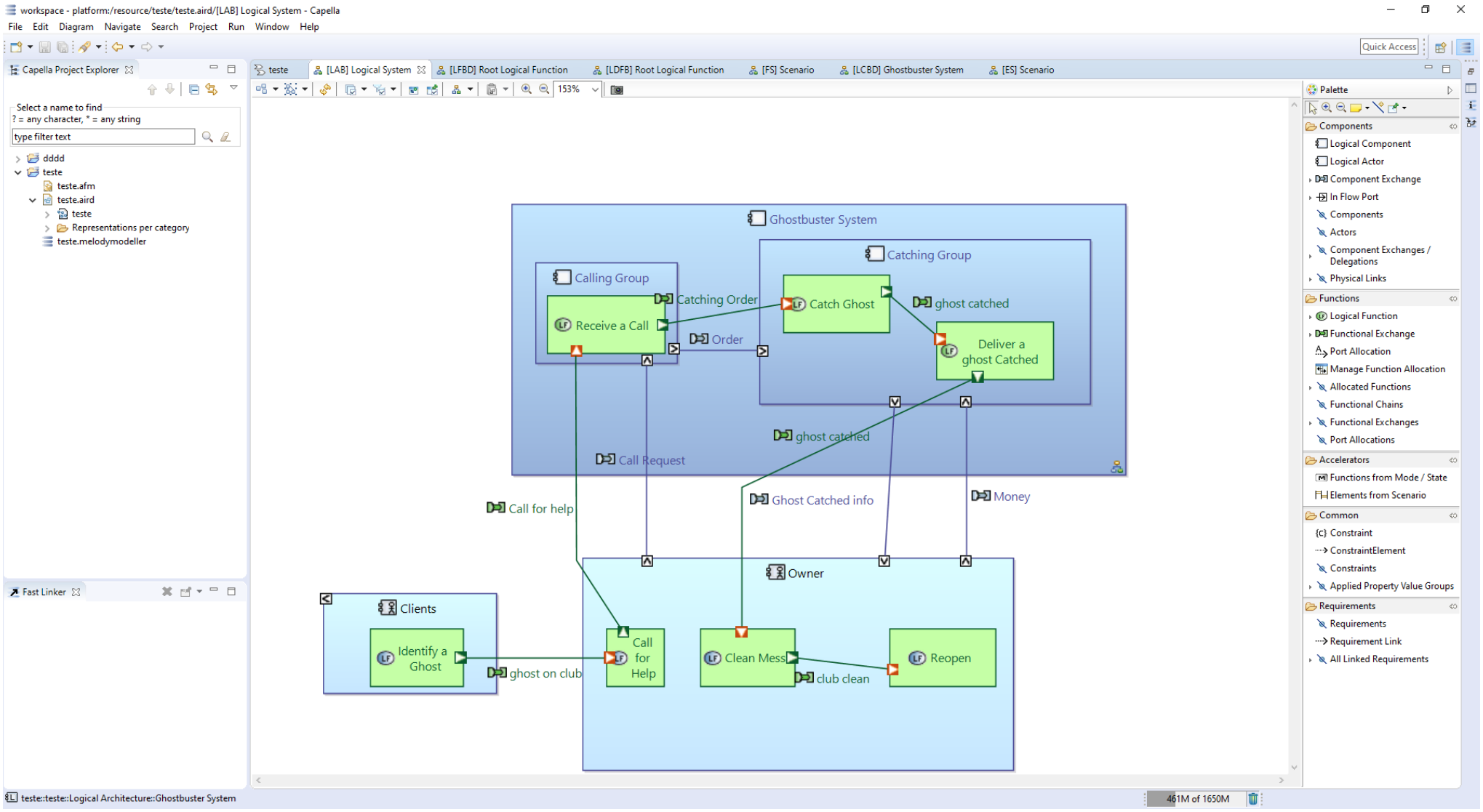
The initialisation and automated updated of the logical actors can be performed according to system actors.
Use an architecture or breakdown diagram to describe the system internal building blocks from a logical point of view.Logical components are intended to interact with each other to achieve the functional goals of the system.

STEP BY STEP EXAMPLE

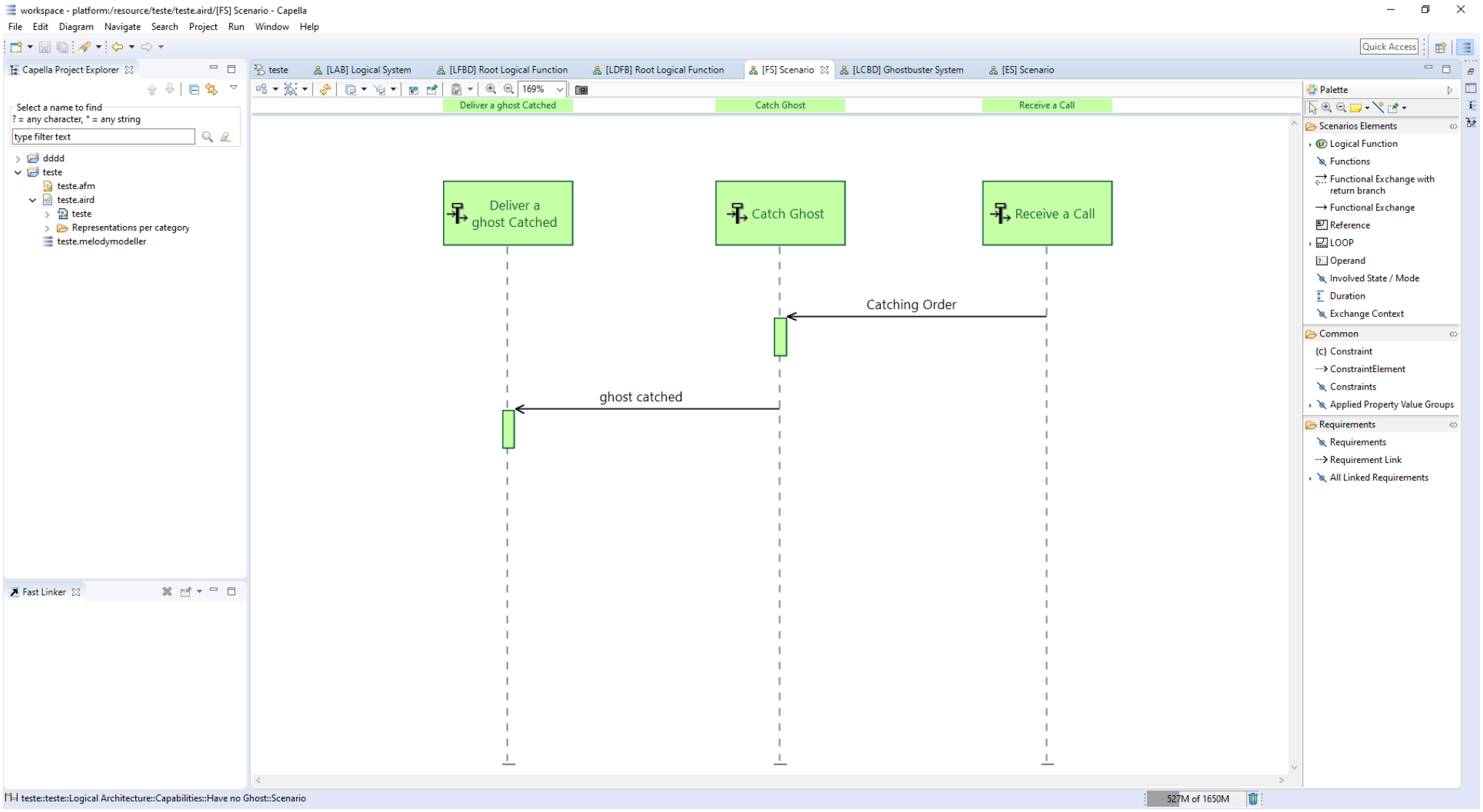
IMPORT DECISION FROM SA



[LAB] Logical Architecture

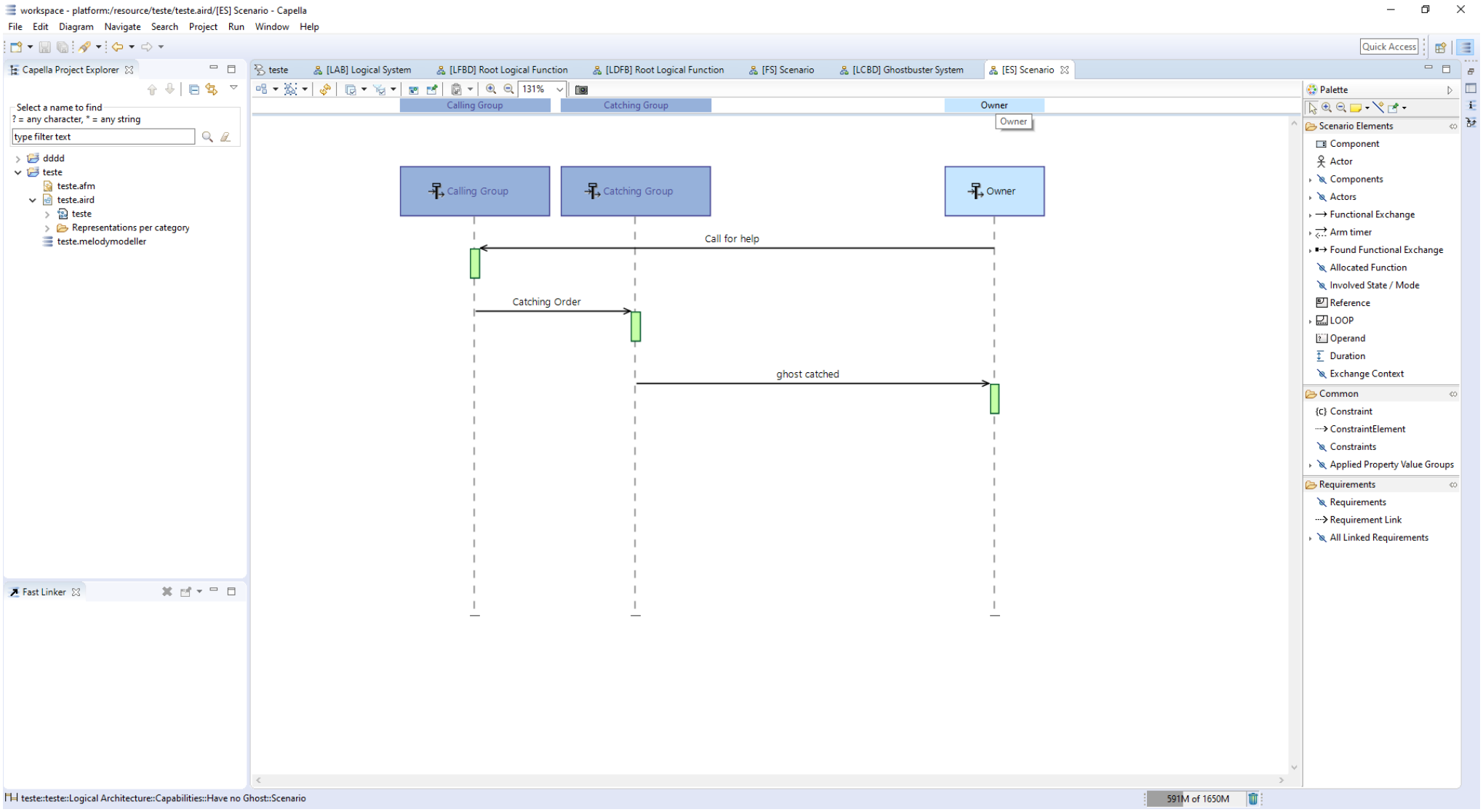


[FS] Functional Scenario



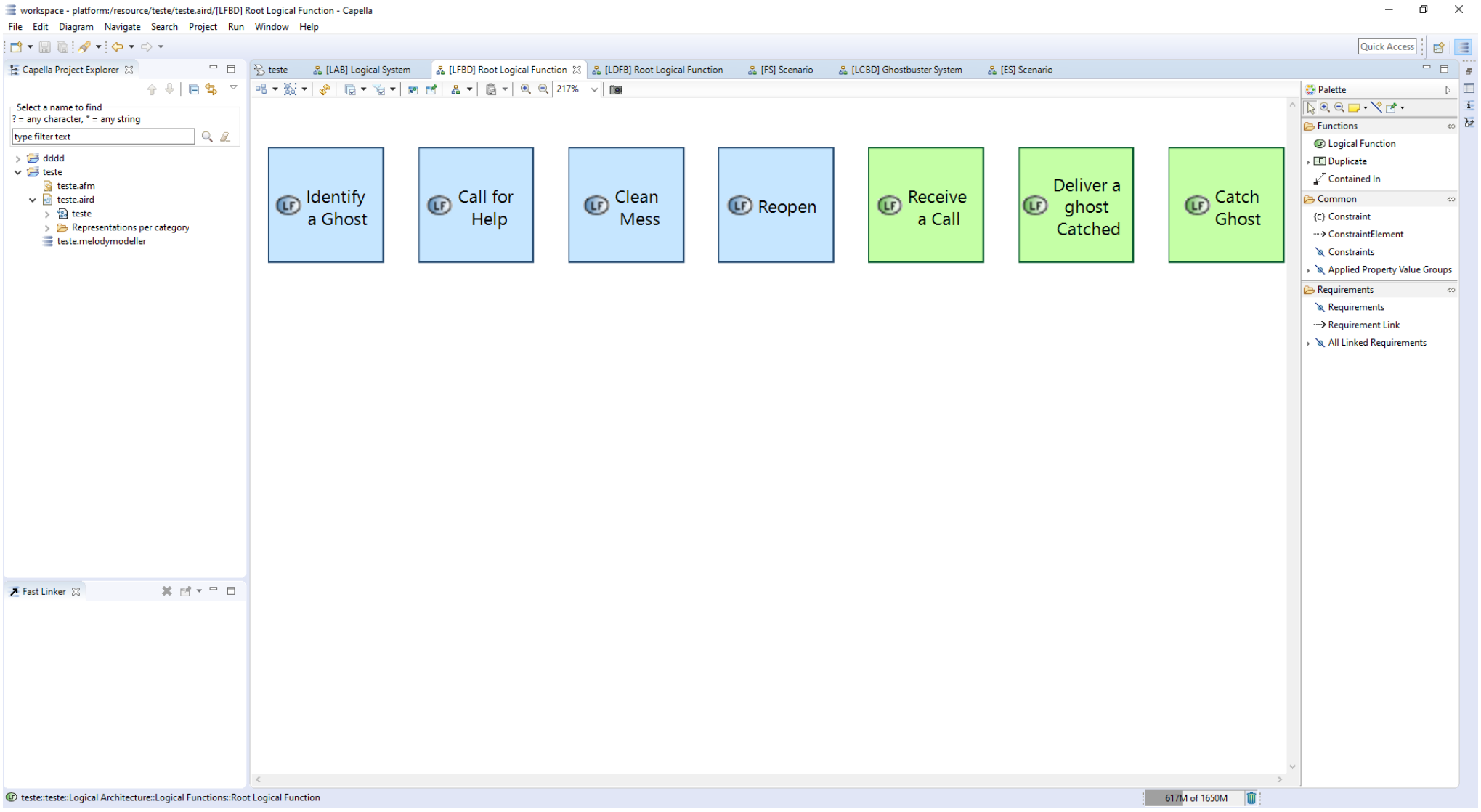
07:16

[ES] Exchange Scenario



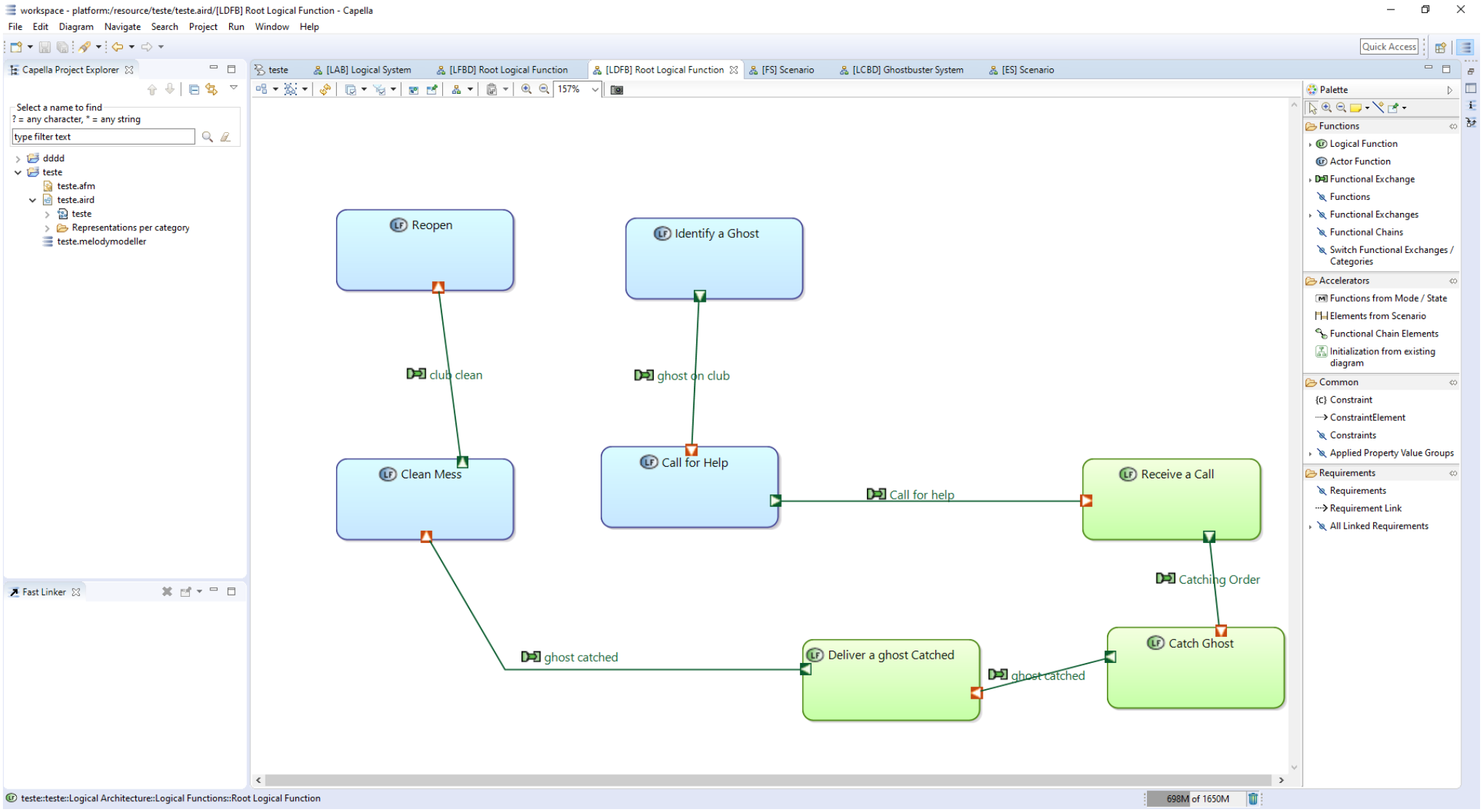
07:16

[LFBD] Logical Functional Break down



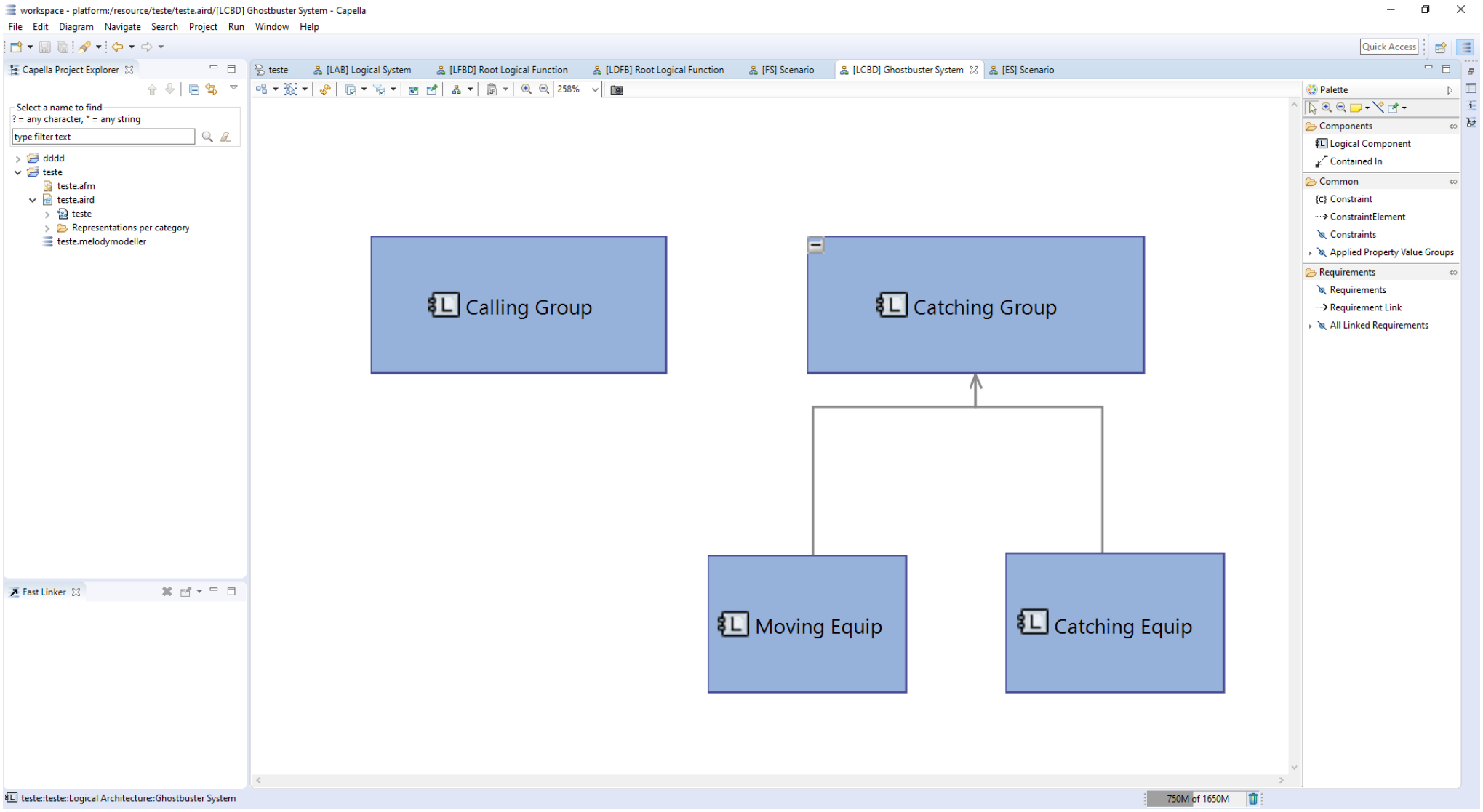
07:16

[LDFB] Logical Data Flow



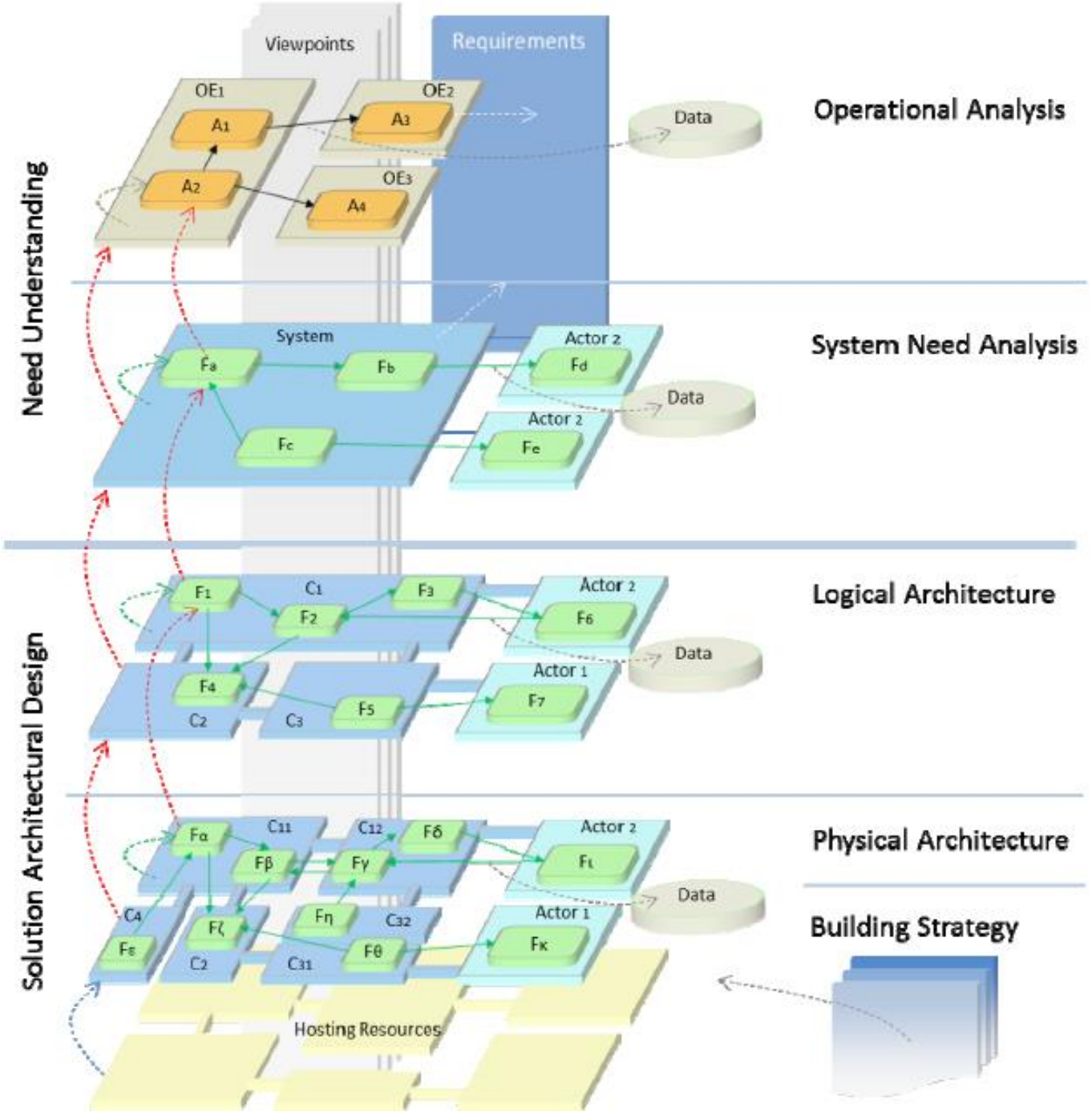
07:16

[LCBD] Logical Component Breakdown



07:16

PHYSICAL ARCHITECTURE



WHAT IS IN THE PHYSICAL ARCHITECTURE (LA)?

PHYSICAL ARCHITECTURE CONCEPTS

PHYSICAL ARCHITECTURE DIAGRAMS

STEP BY STEP EXAMPLE

PHYSICAL ARCHITECTURE CONCEPTS

- **Behavior Physical Component**: **Physical Component tasked with Physical Functions** and therefore carrying out part of the **behavior** of the System (for example software component, data server, etc.);

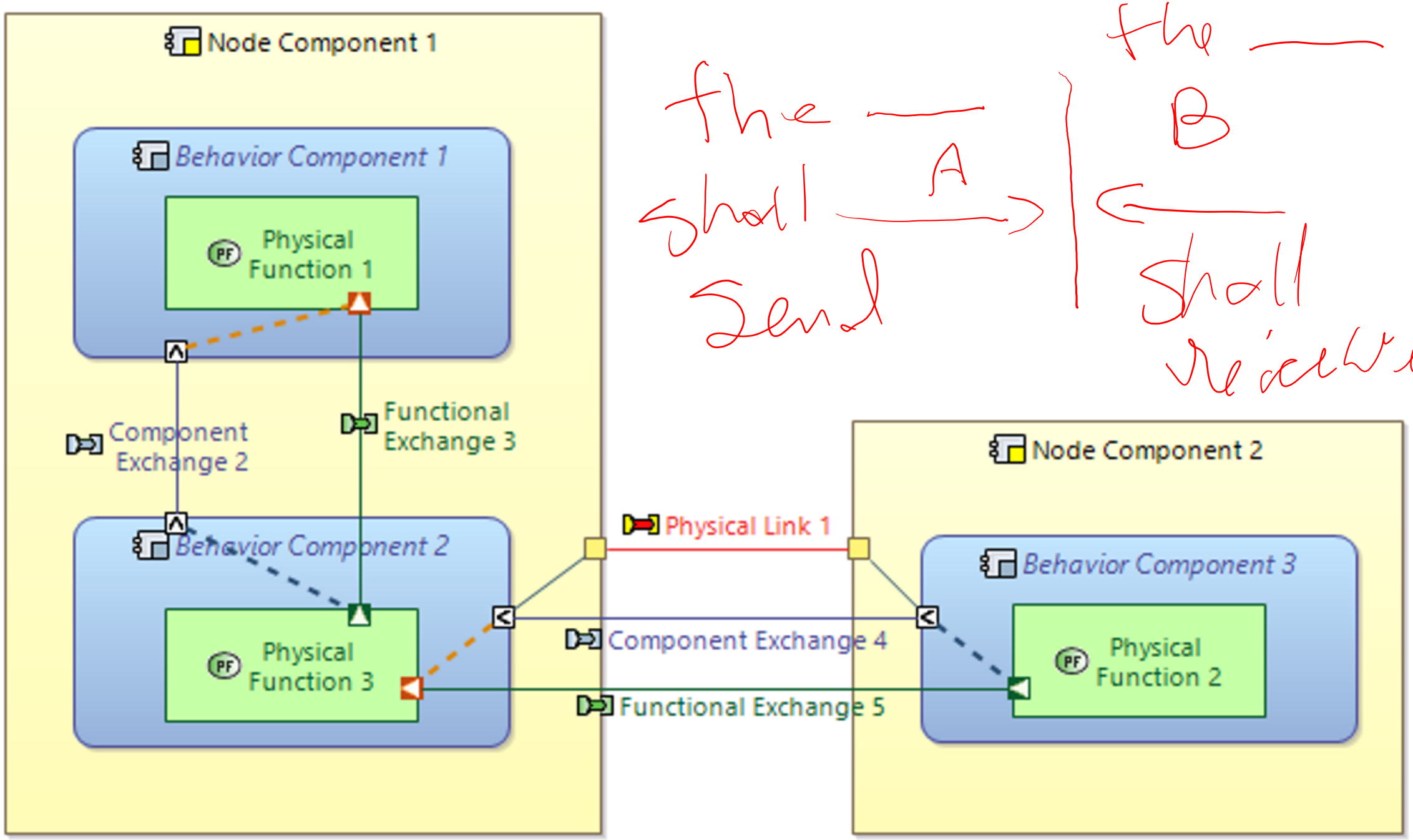
- **Node (or Implementation) Physical Component**: Physical Component that provides the **material resources needed for one or several Behavior Components** (for example processor, router, OS, etc.).

- At this level, the main concepts proposed by Arcadia are similar to those of the Logical Architecture: Physical Function, Functional Exchange, Physical Component, Physical Actor, etc. However, there are some additional concepts, notably:

- **Physical Port**: non-oriented **port that belongs to an Implementation Component** (or Node). The structural port (Component Port), on the other hand, has to belong to a Behavior Component;

- **Physical Link**: non-oriented **material connection between Implementation Components** (or Nodes). The Component Exchange remains a connection between Behavior Components. A Physical Link allows one or several Component Exchanges to take place (for example Ethernet cable, USB cable, etc.);

- **Physical Path**: **organized succession of Physical Links enabling a Component Exchange** to go through several Implementation Components (or Nodes).


Node Component 1
Behavior Component 1
Physical Function 1
Component Exchange 2
Functional Exchange 3
Behavior Component 2
Physical Function 3
Physical Link 1
Component Exchange 4
Functional Exchange 5
Node Component 2
Behavior Component 3
Physical Function 2

07:16

WHAT IS IN THE PHYSICAL ARCHITECTURE (LA)?

Physical Architecture

"how the system will be built"

- This perspective has the same objective as the logical architecture, except that it **defines the finalized architecture of the system**, as it should be completed and integrated. It **adds the functions required by the implementation and technical choices and reveals the behavioral components that perform these functions**. These behavioral components are then implemented using host implementation components that offer them the necessary material resource.
- Defines the solution at a ***sufficient level of detail to specify the developments and acquisitions of all subsystems (or components) to be implemented, and to define and orientate the system integration, verification and validation (IVV) phases***.

- It is often at this level only that choices and constraints are introduced related to implementation and production technologies, **to existing elements to be re-used**. Any ambiguities or inaccuracies that could still exist in the logical architecture (LA), if they did not impact its structuring, should this time be resolved, in order to constitute clear development contracts for the identified components.
- PA is the **privileged place of co-engineering** with subsystem engineering and software or hardware components.

The main activities to be undertaken for the definition of the finalized PA

- to define the structuring principles of the architecture and behavior;
- to detail and finalize the expected system behavior;
- to build and rationalize one or more possible system architectures;
- to select, complete and justify the system architecture retained.

Definition of the structuring principles of the architecture and behavior

- The major objective of the PA is to **minimize complexity through rationalizing**.
  - One of the most used means of rationalization consists of **reducing diversity and heterogeneity** within the solution, by searching for similarities and therefore possible architecture invariants (sometimes called "**patterns**") that can be applied more than once in the same manner – or configurable.
  - Another classic way to overcome complexity is based on the **separation of concerns and their containment** within parts of the architecture as separate as possible from each other.

07:16

Detail and finalization of the expected system behavior

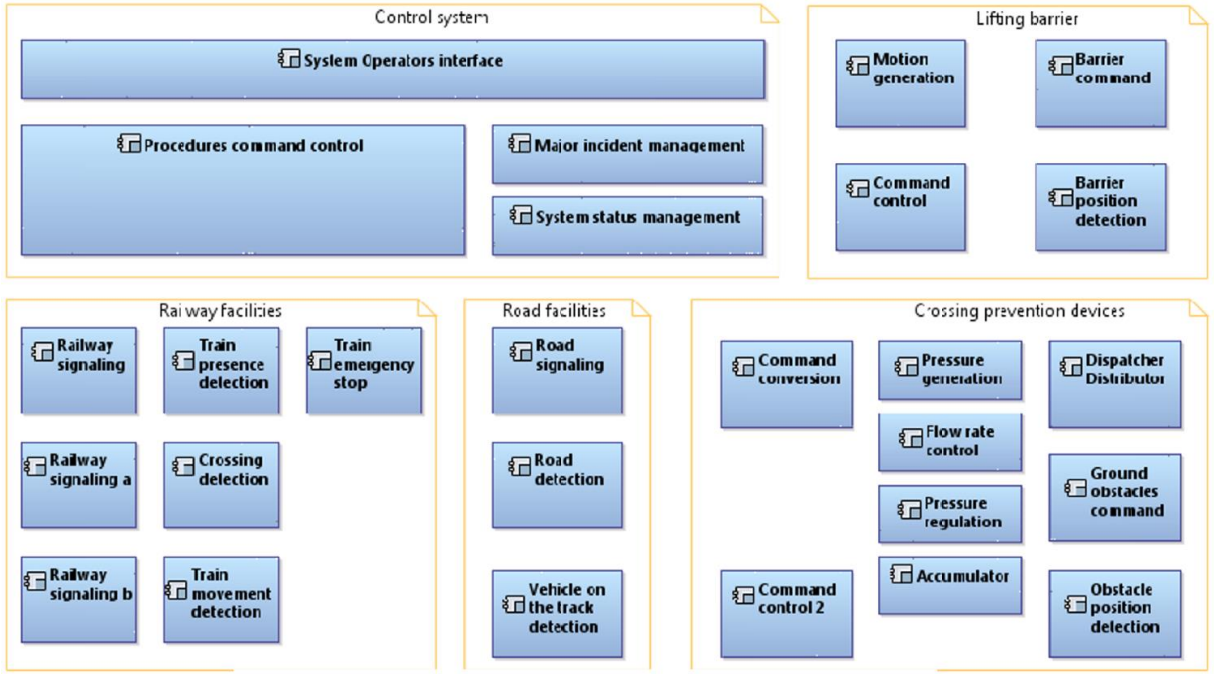
- Define the expected behavior of the system, to a **level of detail and validation** enough so that each of its components can be implemented (or selected and purchased), without any further risk or major questioning; this definition must of course demonstrate compliance with constraints, especially nonfunctional constraints, by which the system will have to abide when being used under operational conditions.

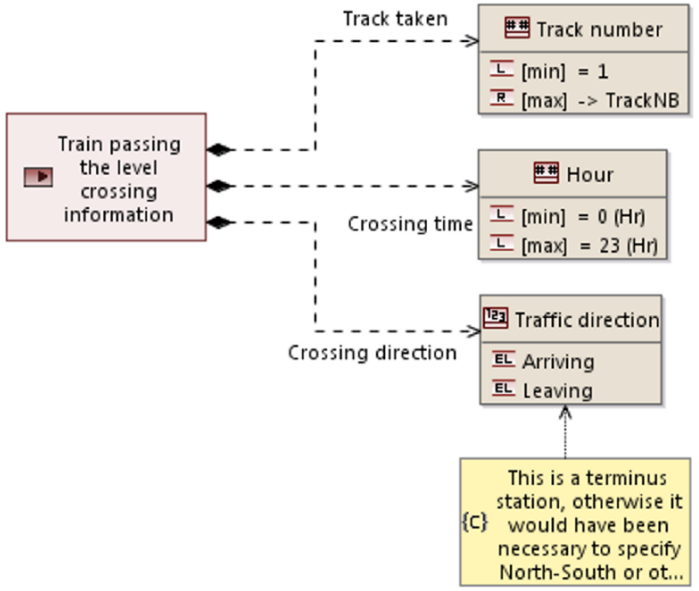
07:16

- In particular, the finalized behavior should **not necessarily be considered as a simple refinement** of that defined in the LA. The finalization of the chosen behavior in fact often constitutes a **re-designing**, which must result from the comparison between the principle behavior of the LA, and the implications of the principles chosen in the PA: **technological choices and adoption of standards, previous structuring principles, etc.**

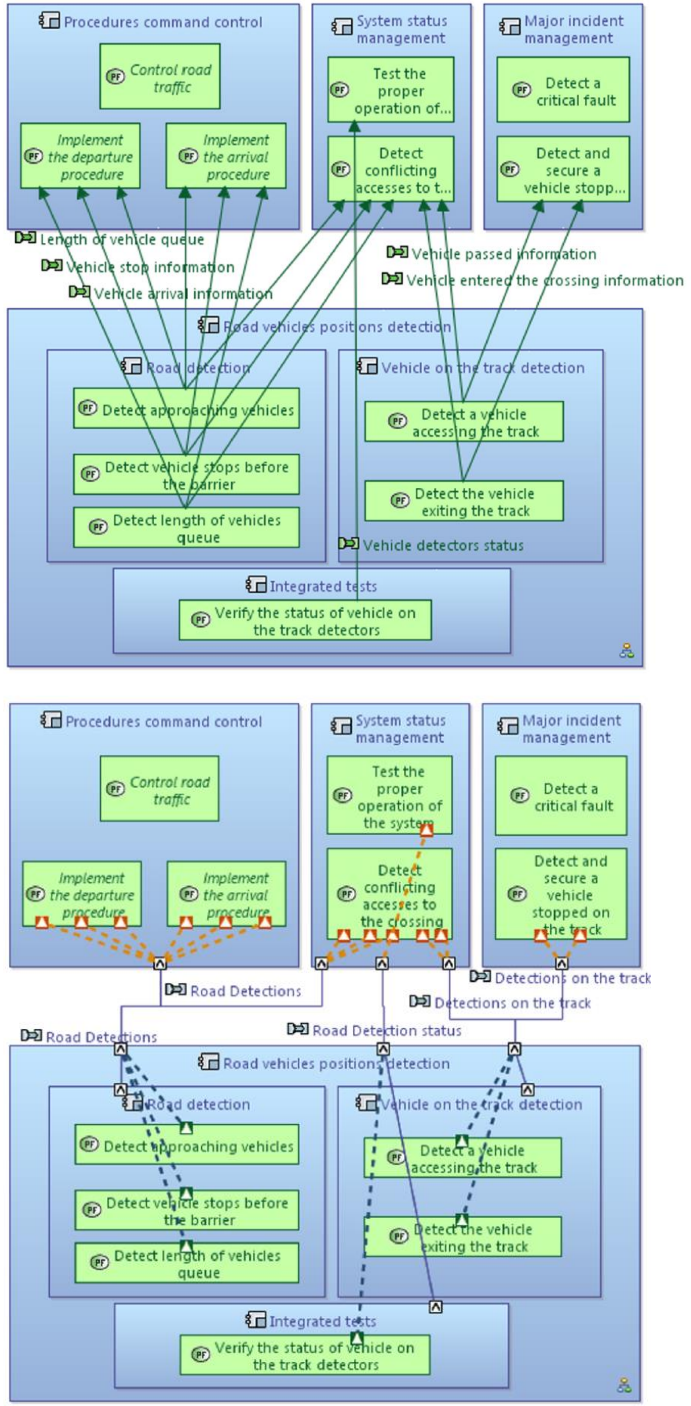
Construction and rationalization of one or more possible system architectures

- This step is intended to **define one or more solutions reflecting the structuring principles** defined in the LA, the previous finalized behavior, satisfying the expected non-functional constraints and applying technology and reuse choices decided in accordance with the structuring principles adopted.

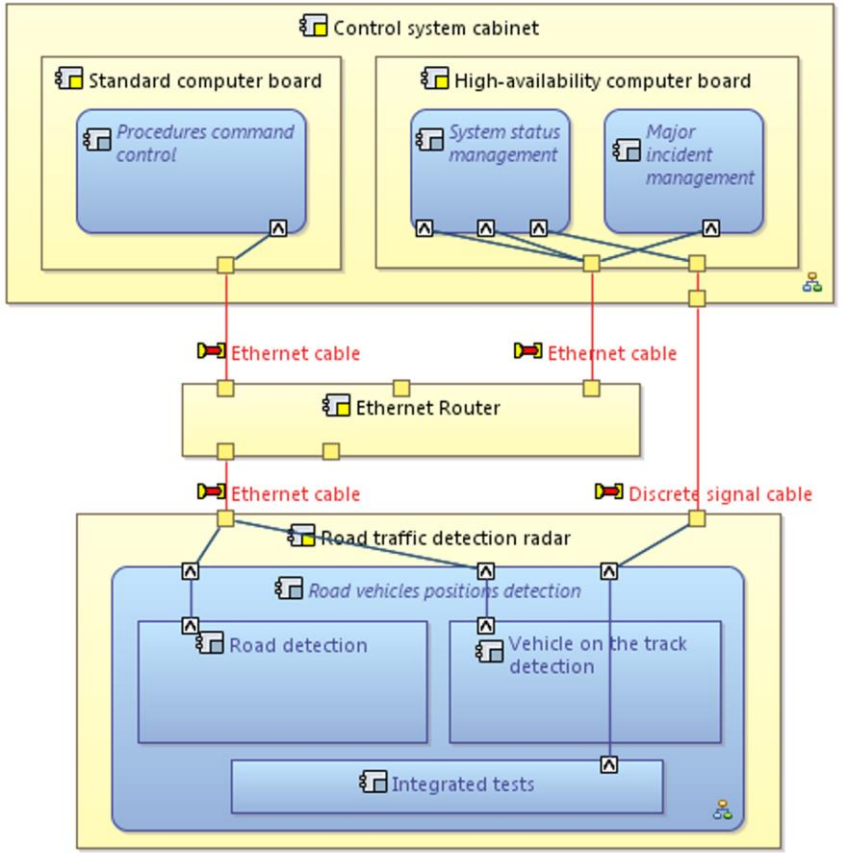




- In the simplest cases, or in systems with a physical or electrical dominant, the exchange items are often simple in their description and usage at this engineering and modeling level. However, for more complex exchange items, involving large numbers of exchange contents elements, it is desirable to be able to **structure a list of exchange items that can be extensive, by grouping them by type of service achieved**, for example. This is the role of the concept of an interface (also mainly present in software design).

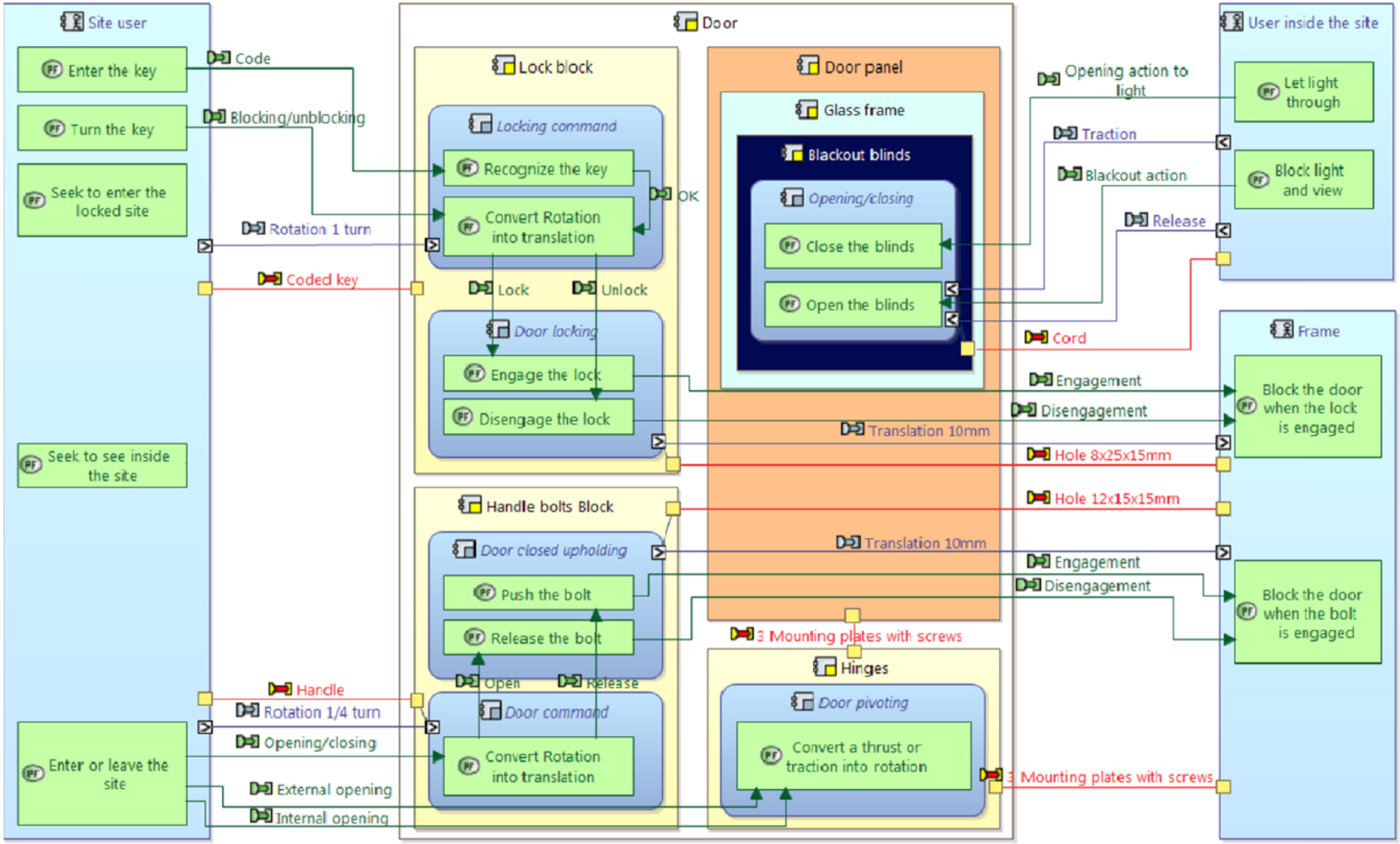


- The PA complements this behavioral description by way of the definition of implementation components, or **hosting physical components, containing behavioral components and forming the infrastructure of the system; the behavioral components are deployed on these host components**, which provide necessary resources for their behavior and hardware vectors (links) for their communications. It may thus consist of high-performance computers, resources for digital or analog processing, mechanical systems, evaporators, furnaces, chemical reactors, etc.
- **Hosting physical components are themselves connected by physical links**, reflecting the media that channel exchanges between behavioral components (a cabled network, a satellite link, a pipe or a mechanical shaft, for example).
- The same rationalization processes have to be performed for hosting physical components as for the behavior and behavioral components, in compliance with the established structuring principles.



Selection, completion and justification of the system architecture

- Finalize the choices among potential alternatives, and **verify that the retained alternative satisfies, possibly by means of an acceptable trade-off, all of the needs and constraints that have been imposed thereon**.
- For example, the implementation resources available may not be sufficient to support an expected behavior or associated properties (computational load too high for a given process in computers supporting it, temperature and pressure too high for a given pipe, etc.). This will lead to a redesigning of the architecture, including a redecomposition and a different distribution of behavioral components, or the use of other implementation resources (more powerful computers, more robust pipes).


Site user
Enter the key
Turn the key
Seek to enter the locked site
Seek to see inside the site
Enter or leave the site
Code
Blocking/unblocking
Rotation 1 turn
Coded key
Handle
Rotation 1/4 turn
Opening/closing
External opening
Internal opening
Door
Lock block
Locking command
Recognize the key
Convert Rotation into translation
OK
Lock
Unlock
Door locking
Engage the lock
Disengage the lock
Handle bolts Block
Door closed upholding
Push the bolt
Release the bolt
Open
Release
Door command
Convert Rotation into translation
Door panel
Glass frame
Blackout blinds
Opening/closing
Close the blinds
Open the blinds
3 Mounting plates with screws
Hinges
Door pivoting
Convert a thrust or traction into rotation
3 Mounting plates with screws
Translation 10mm
Hole 8x25x15mm
Hole 12x15x15mm
Translation 10mm
Engagement
Disengagement
Engagement
Disengagement
Cord
Opening action to light
Traction
Blackout action
Release
User inside the site
Let light through
Block light and view
Frame
Block the door when the lock is engaged
Block the door when the bolt is engaged

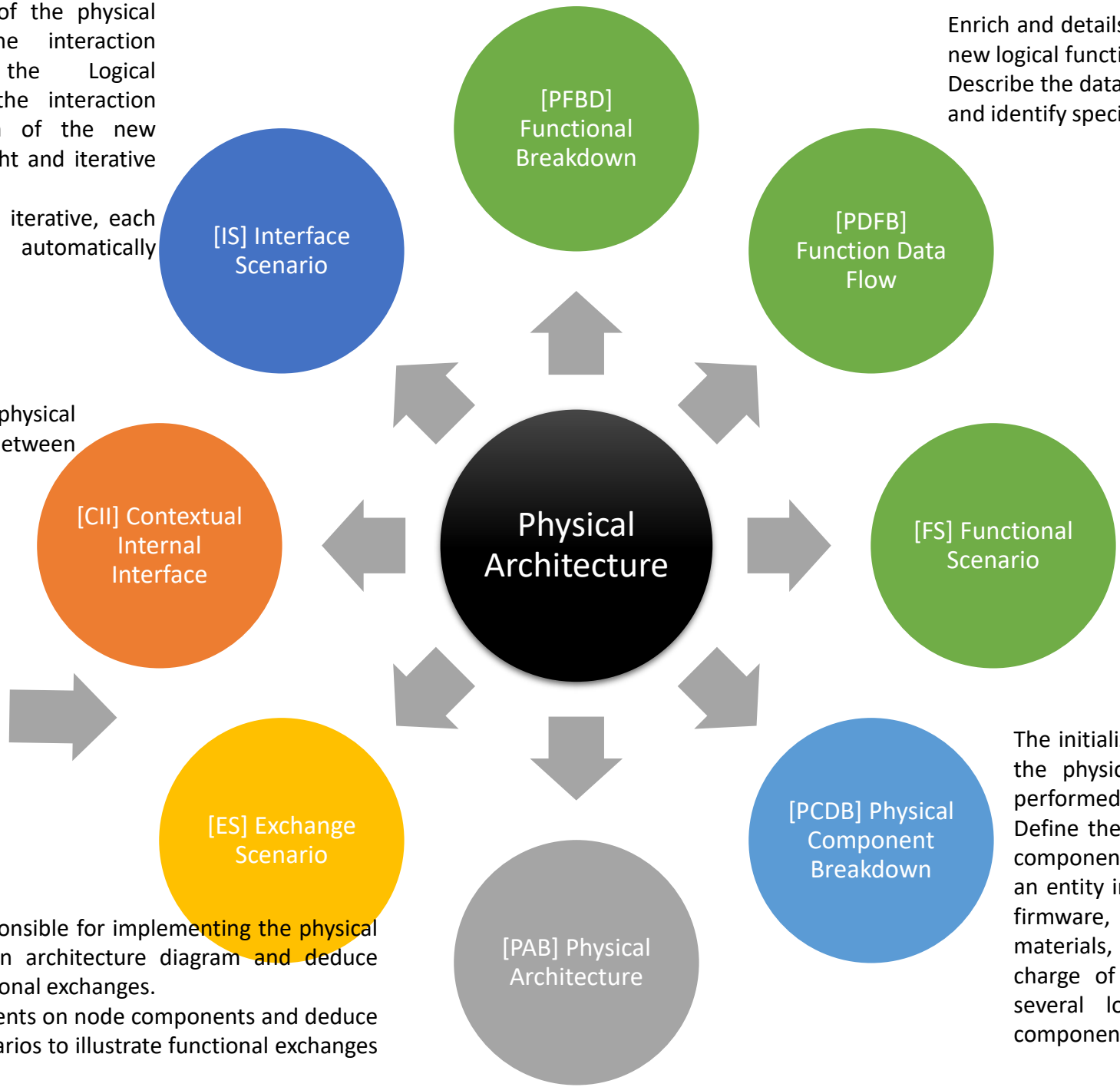
PHYSICAL ARCHITECTURE DIAGRAMS

Specify the dynamical behaviour of the physical components by completing the interaction sequences coming from the Logical Architecture.The enrichment of the interaction sequences and the identification of the new physical interfaces are two very tight and iterative activities.
The scenario refinement process is iterative, each update on a source can be automatically propagated to the target.

Enrich and details the functional breakdown with new logical functions.
Describe the data flows between logical functions and identify specific functional chains.

Delegate each logical interface to one physical component. Create new physical interfaces between components

Initialization and automated update of the logical functions according to the system functions
The transition tools create a first 1-1 traceability mapping between Logical Architecture and System Analysis. Use dedicated traceability matrices to modify the traceability relationships.



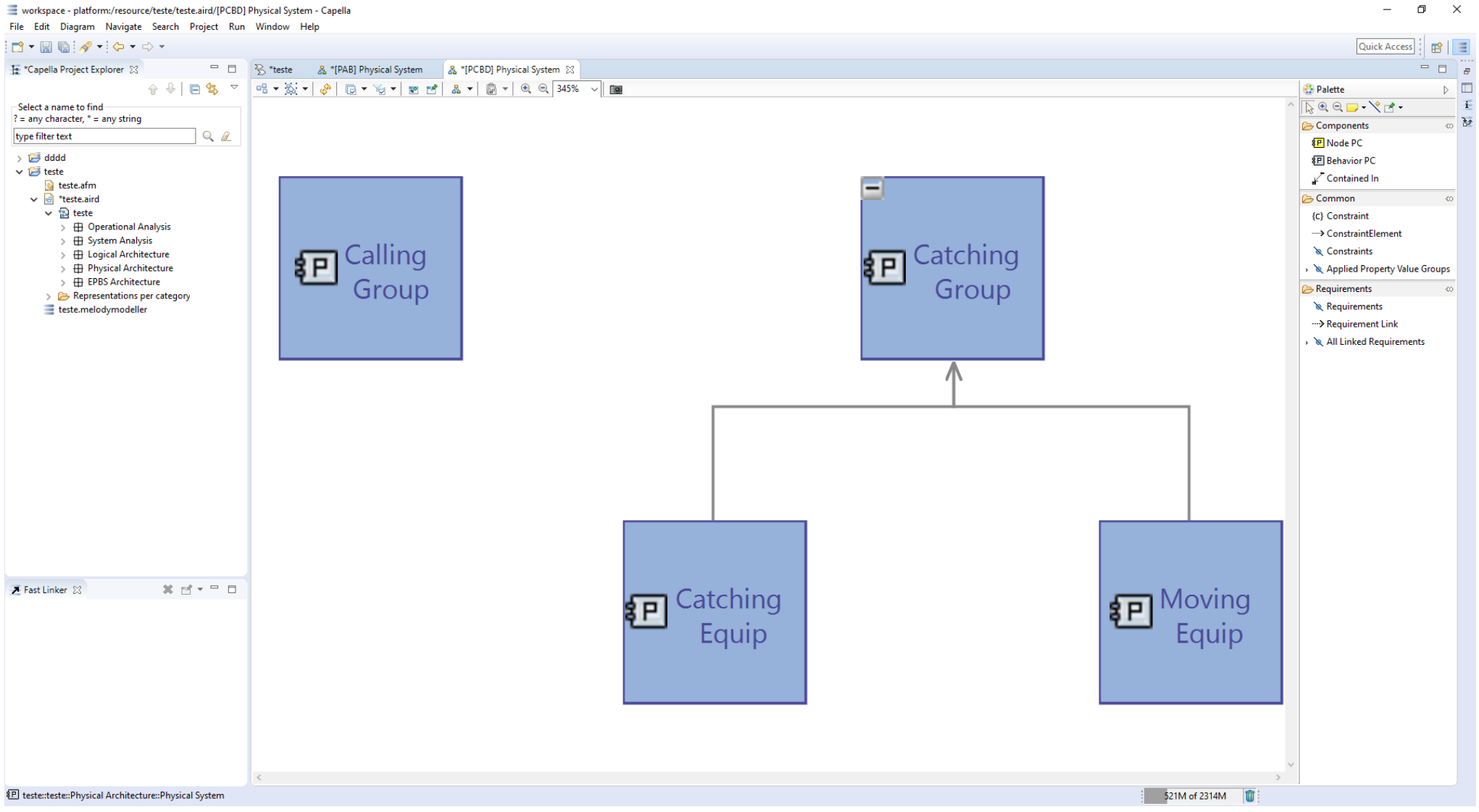
The initialization and automated updated of the physical actors can be automatically performed according to logical actors.
Define the physical components. A physical component is a physical representation of an entity in the system(hardware, software, firmware, personnel, facilities, data, materials, services and processes). It is in charge of the implementation of one or several logical components. A physical component can be Node or Behaviour.

The behavioural physical components are responsible for implementing the physical functions. Manage these allocations using an architecture diagram and deduce component exchanges implementing the functional exchanges.
Manage the deployment of behaviour components on node components and deduce physical links and paths. Create dataflows scenarios to illustrate functional exchanges between the components.

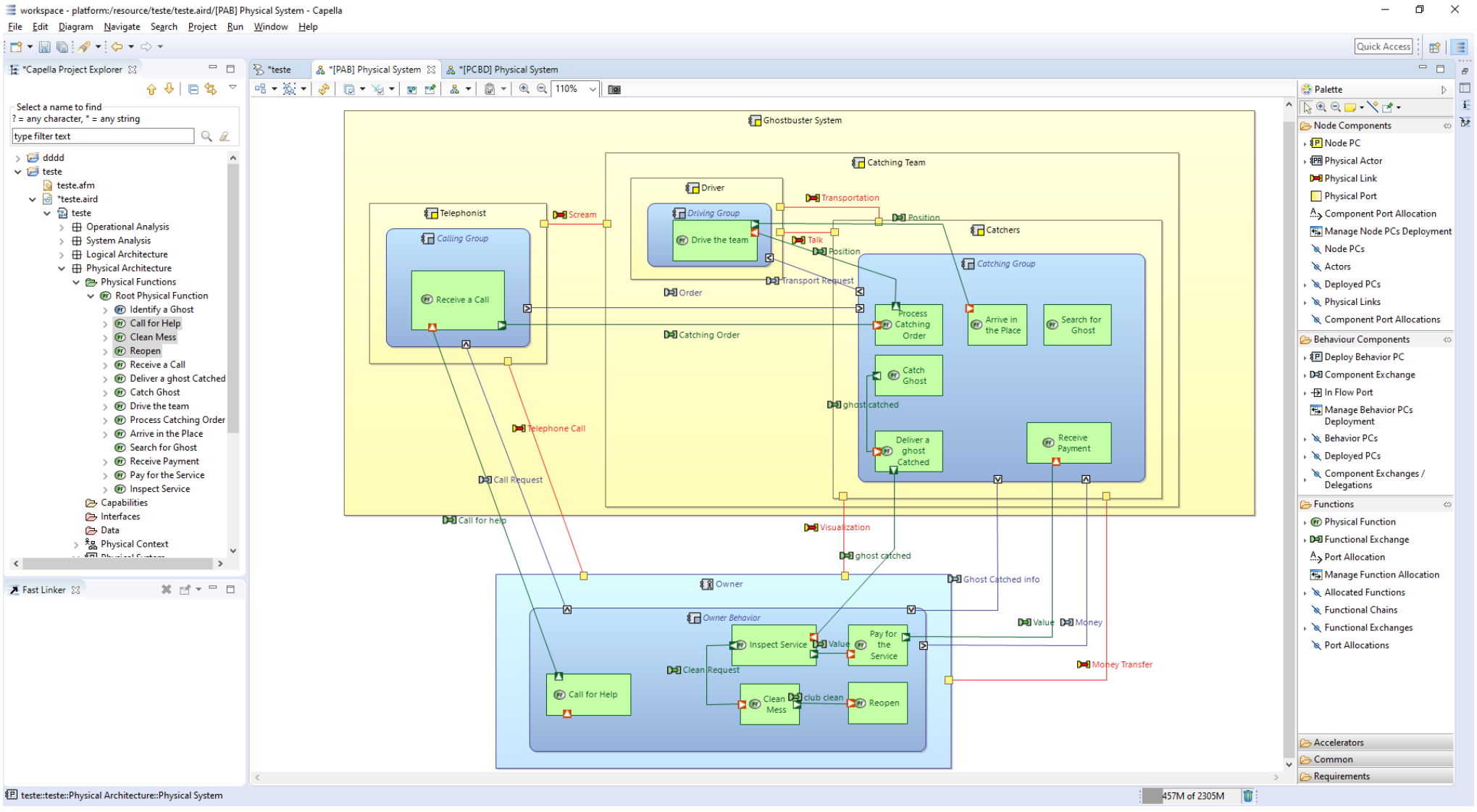
STEP BY STEP EXAMPLE

IMPORT DECISION FROM LA

[PCBD] Component Breakdown

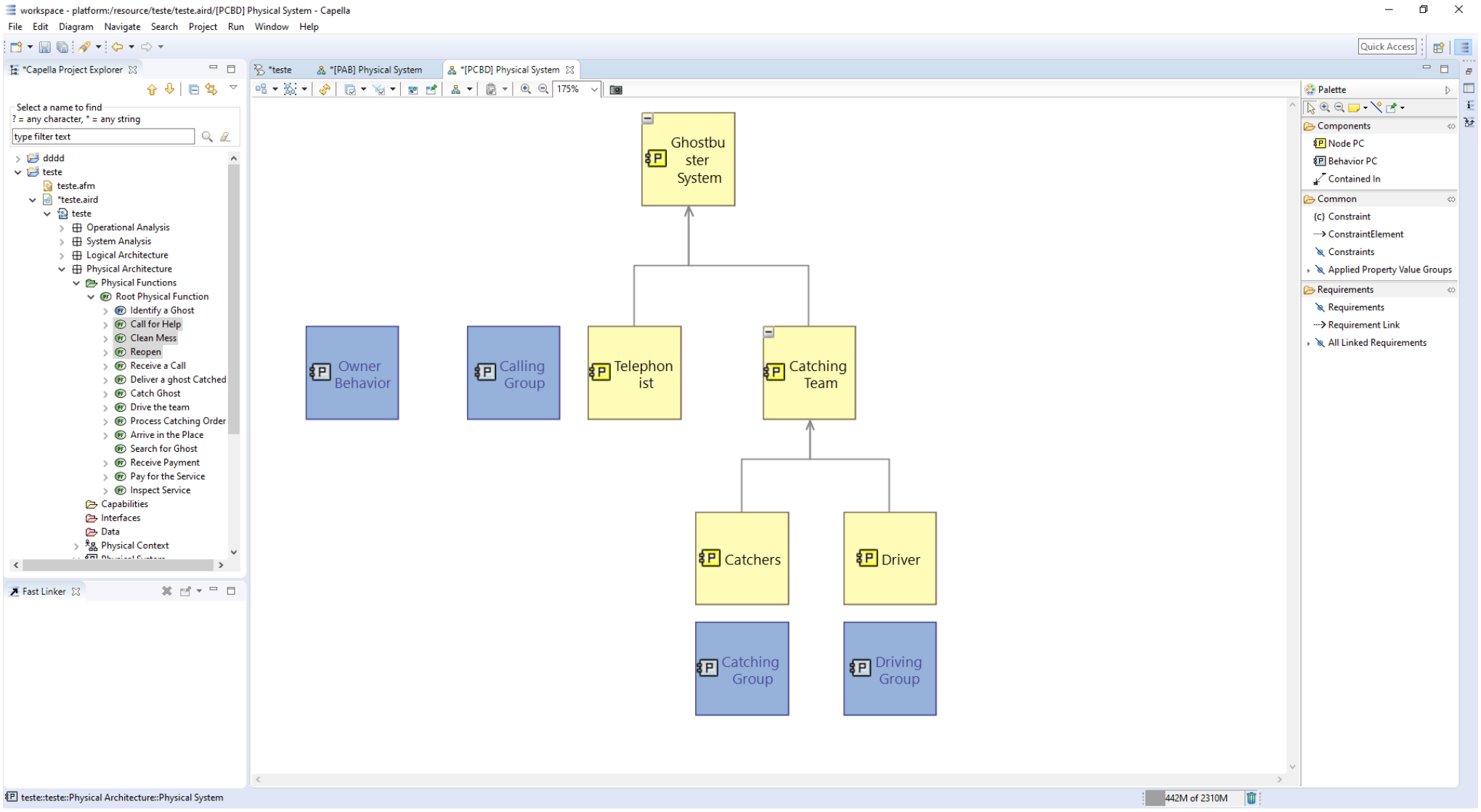


[PAB] Physical Architecture



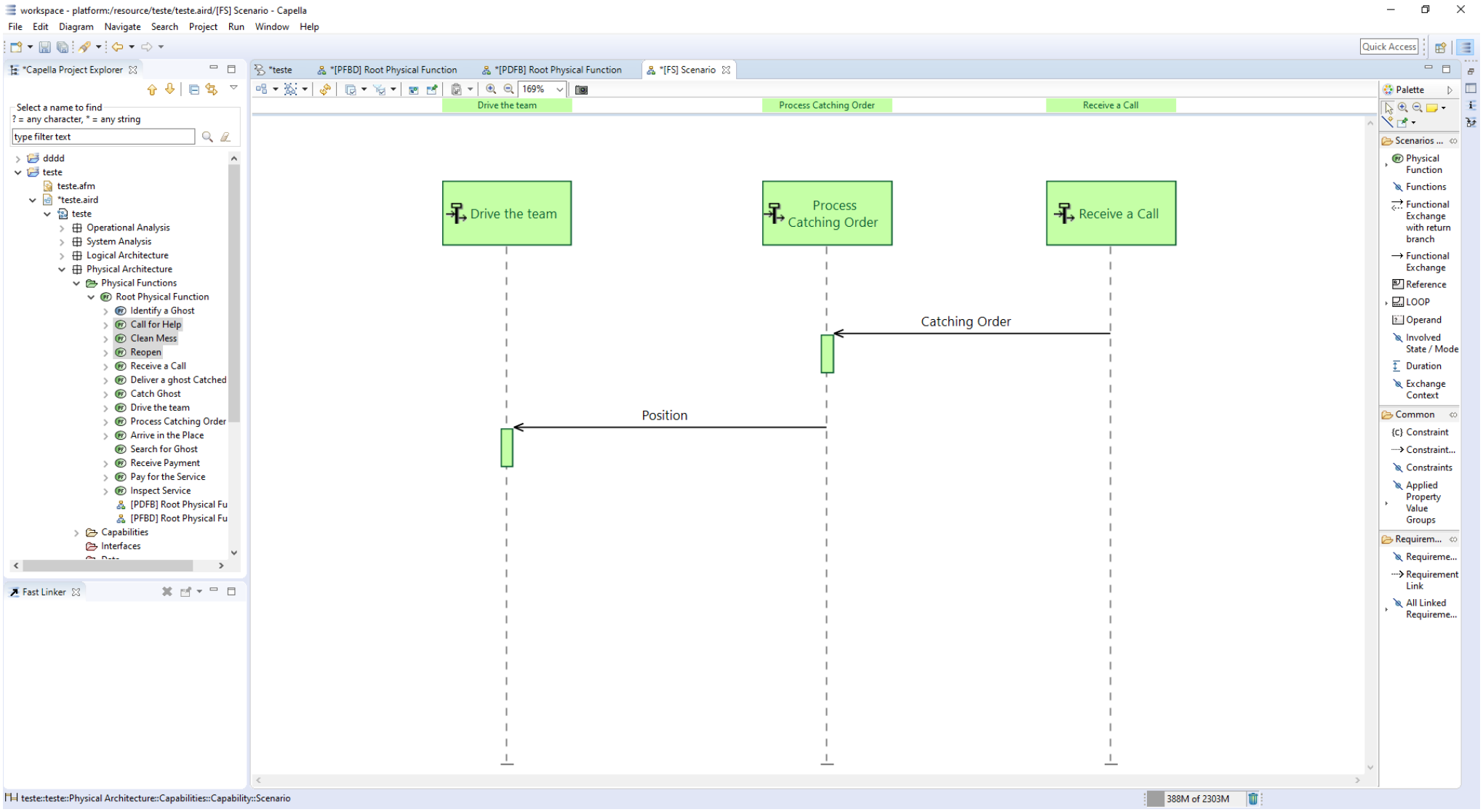
07:16

Review of Component Breakdown



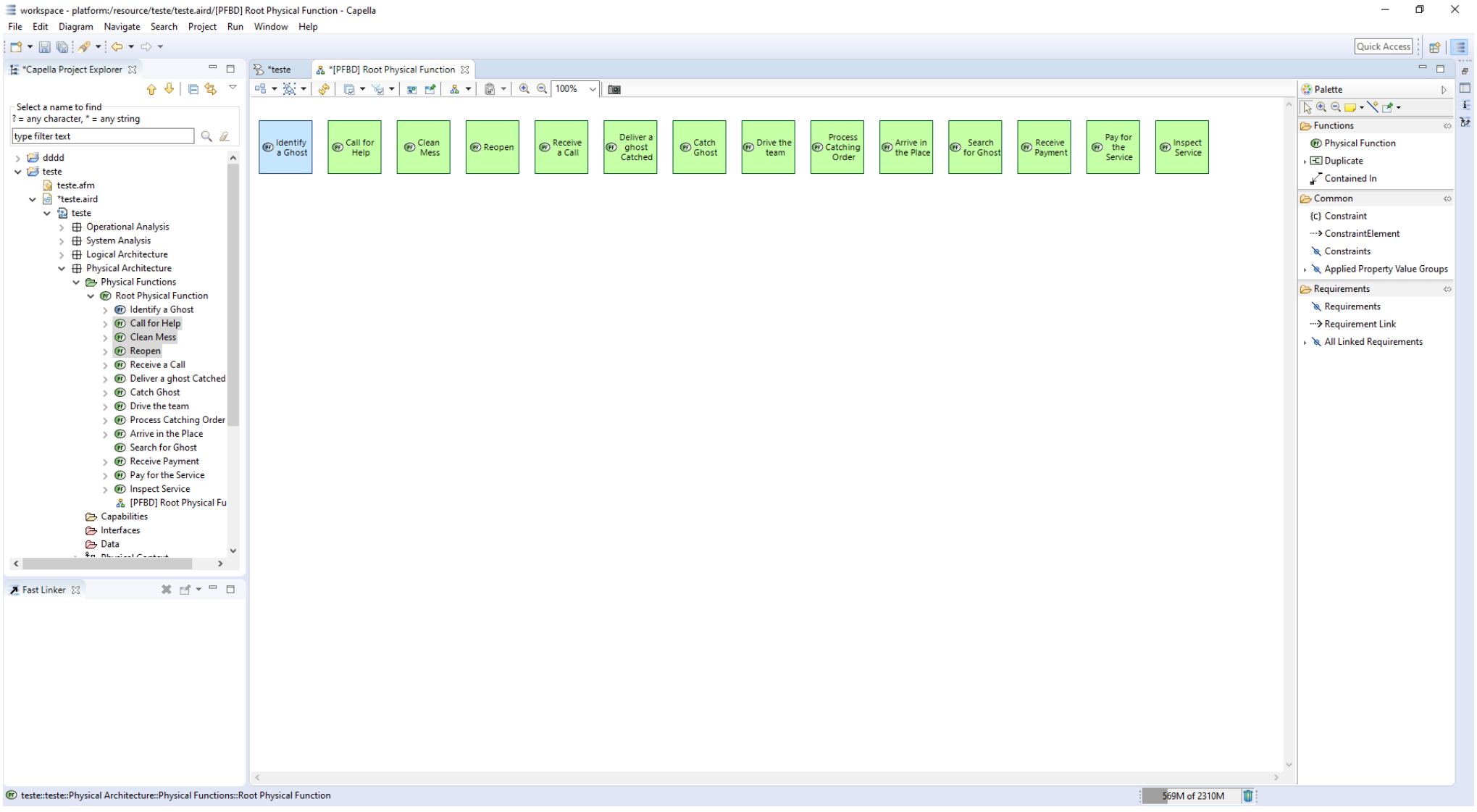
07:16

[FS] Functional Exchange



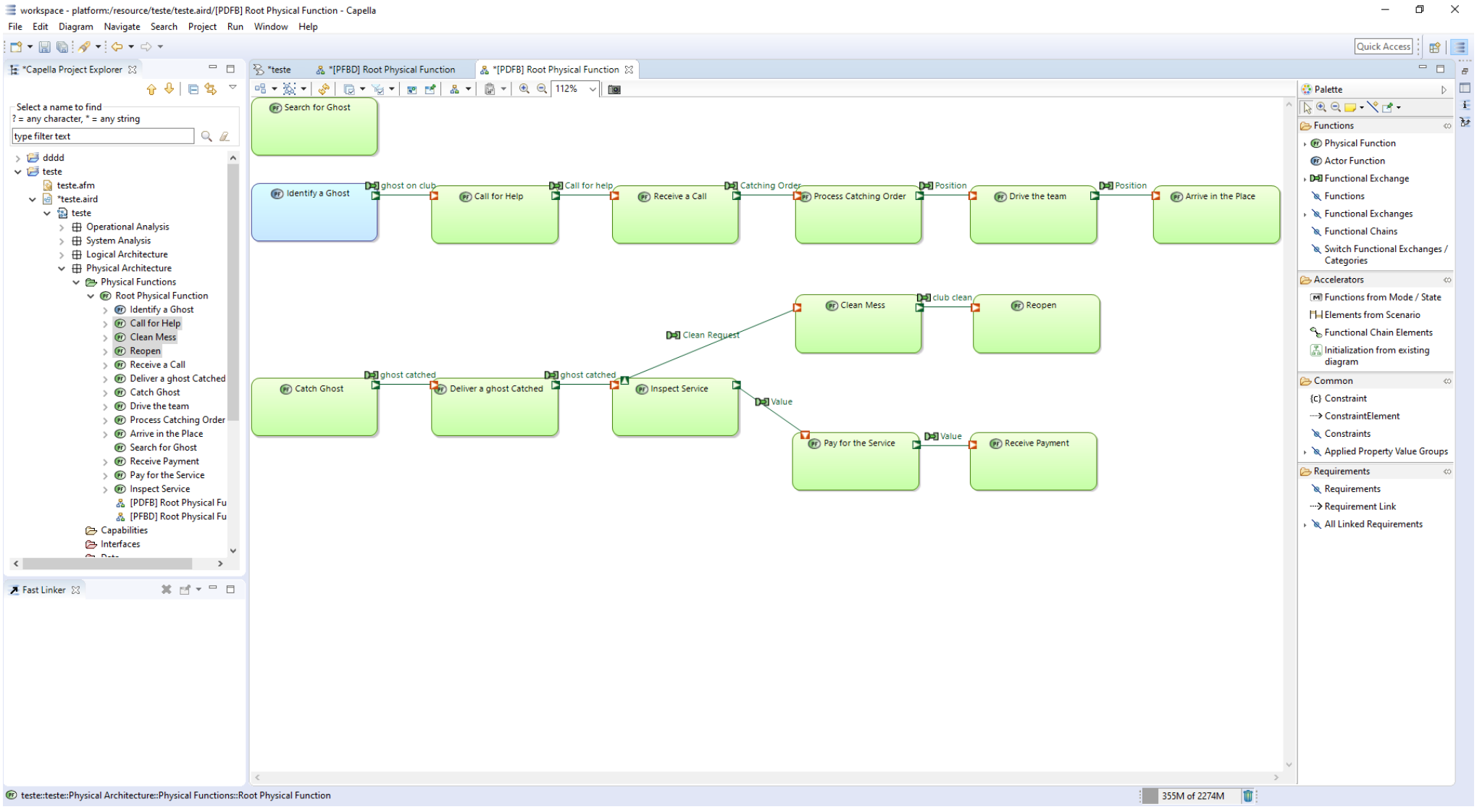
07:16

[PFBD] Functional Breakdown



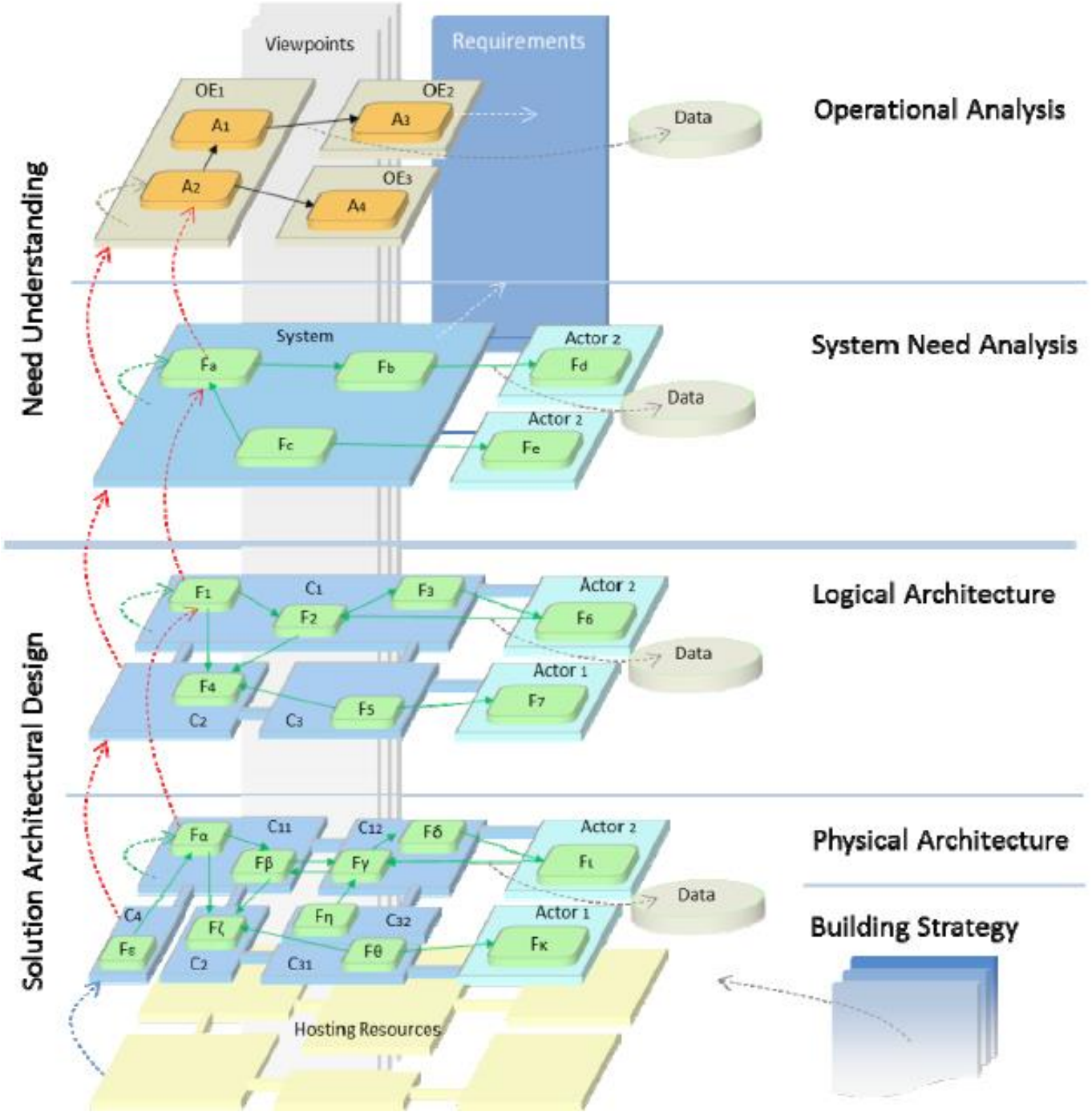
07:16

[PDFB] Physical Functional Data Flow



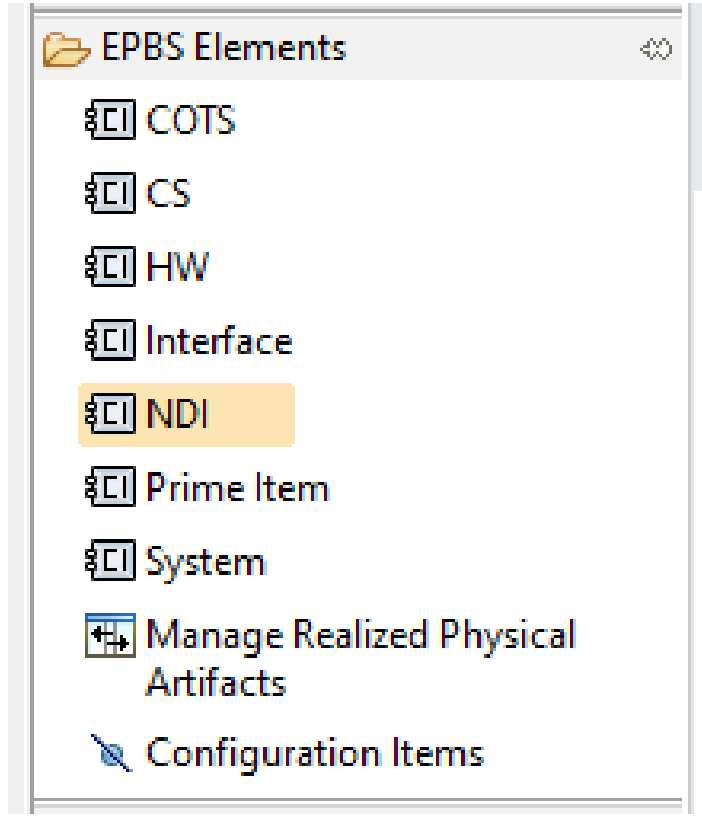
07:16

EPBS (END PRODUCT BREAKDOWN STRUCTURE) AND INTEGRATION CONTRACTS


Viewpoints
Requirements
Need Understanding
Solution Architectural Design
OE1
OE2
OE3
A1
A2
A3
A4
Data
Operational Analysis
System
Actor 2
Actor 2
Actor 1
Fa
Fb
Fc
Fd
Fe
Data
System Need Analysis
F1
F2
F3
F4
F5
F6
F7
C1
C2
C3
Logical Architecture
Fα
Fβ
Fγ
Fδ
Fε
Fζ
Fη
Fθ
Fι
Fκ
C11
C12
C4
C2
C31
C32
Physical Architecture
Building Strategy
Hosting Resources

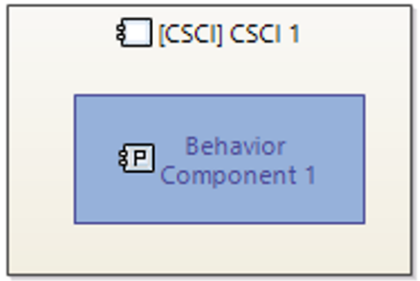
EPBS CONCEPTS

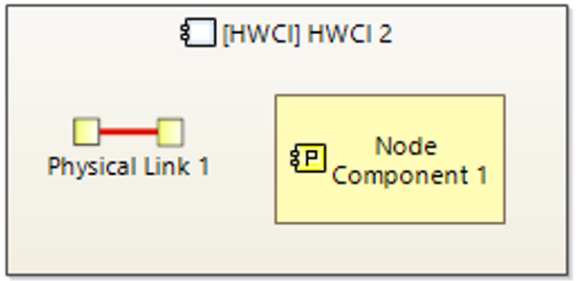
- **COTS CI**: component off the shelf configuration item.
- **CS CI**: computer software configuration item;
- **HW CI**: hardware configuration item;
- **Interface CI**: Interface Configuration Item
- **NDI CI**: non-developed configuration item;
- **Prime Item CI**: decomposable configuration item;
- **System CI**: system-type configuration item;

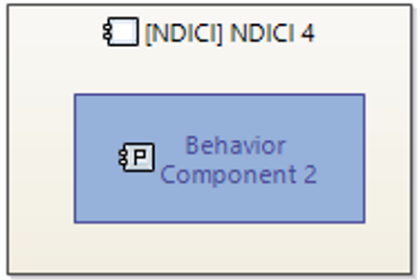


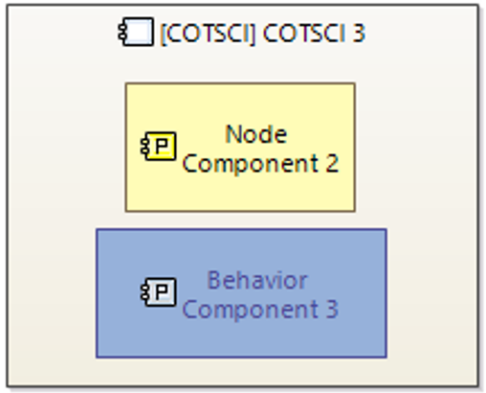
CONFIGURATION ITEMS

- The first one, **CSCI 1**, is a software Configuration Item, carrying out Behavior Component 1.
- The second item, **HWCI 2**, is a material Configuration Item, carrying out Node Component 1, as well as Physical Link 1.
- The third, **COTSCI 3**, is an off the shelf Configuration Item, carrying out both Node Component 2 as well as Behavior Component 3.
- Finally, the fourth, **NDICI 4** is a non-developed Configuration Item, carrying out Behavior Component 2.









HOW TO CREATE

PRODUCT BUILDING STRATEGY

"What is expected of each component, and the conditions of its integration into the system".

- Being the final stage of system design strictly speaking, this definition of the **Product Building Strategy (BS)** prepares later development, implementation, production, acquisition stages of subsystems or components identified in the physical architecture, and their integration, up to the qualification of the system in an operational environment.

- This level aims to deduce, from the Physical Architecture, the **conditions that each Component must satisfy to comply** with the constraints and **choice of design of the architecture** identified in the previous phases (*"what is expected from the provider of each component"*). The Physical Components are often grouped into larger Configuration Items that are easier to manage in terms of industrial organization and responsibilities.

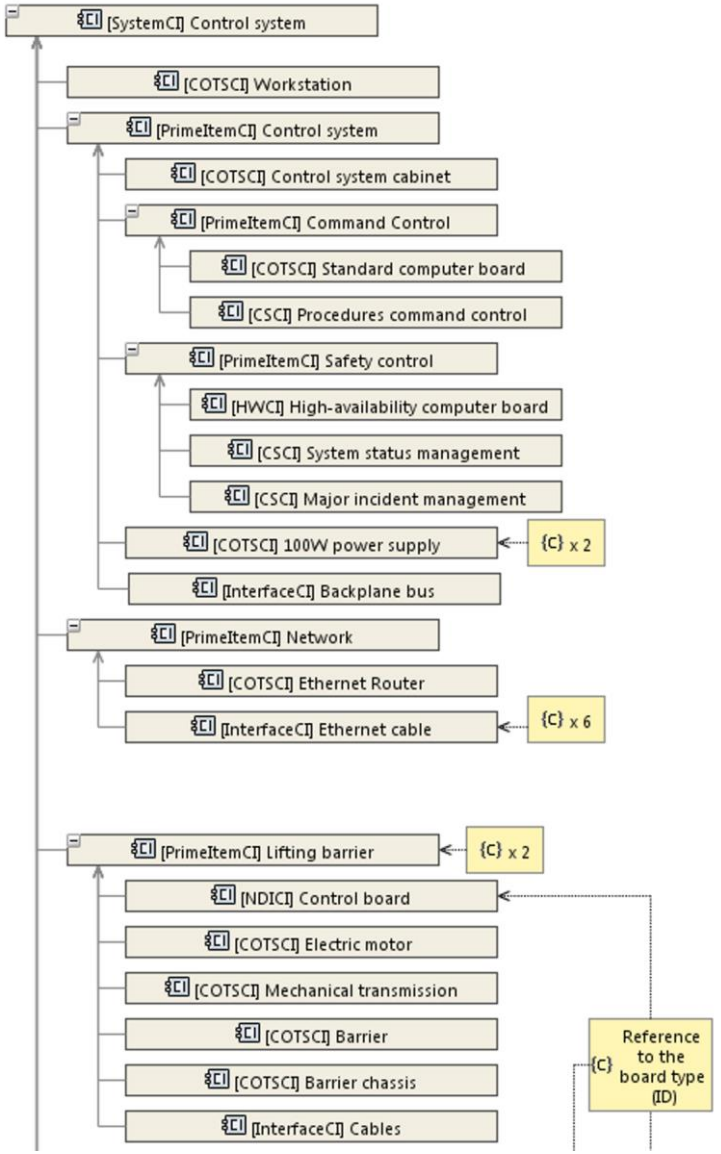
The main activities to carry out for the definition of development, acquisition and integration contracts are the following:

- to define the product breakdown structure;
- to finalize the development contracts of components to be implemented;
- to consolidate the definition of components to be acquired;
- to define the integration, verification and validation strategy and processes.

07:16

Definition of the product breakdown structure

- The **product breakdown structure lists the set of all of the concrete elements**, to be created or acquired, constituting the system as previously defined, and that will be the subject of the integration phase.
- Each item will have to be **managed as part of configurations** in order to identify its configuration state: its version, its parameters or potential adaptation, etc., in each of the system definitions to which it contributes. These elements are part of the product breakdown structure.



Finalization of development contracts of components to be implemented

- The **development technical contract of each component describes what is expected of its supplier** by the system engineering team, to satisfy the definition of the physical architecture produced and to secure later integration validation verification stages.
- In principle, compliance with this contract should **ensure that the IVV of the system will be performed without problem**, and that functional as well as nonfunctional needs will be satisfied.

07:16

For a behavioral component it describes

- the **functions** (or services)
- **interfaces** with its environment (other components, actors external to the system)
- the expected **dynamic behavior**, within the component and at its boundaries (shown by functional chains, scenarios, states machines, etc.)
- the different **versions** of the component to be delivered throughout the IVV
- the expectations from the IVV specific to this component may also be requested in the form of **scenarios or functional chains** to be demonstrated
- **interfaces** with the host component in which it is inserted
- the amount of **resources** of this host component and communications media that are allocated thereto
- the potential **contribution** of the component to the global data model of the system
- the **non-functional constraints** to which it will have to comply
- the possible **product line constraints** (optional parts of services and associated conditions)
- eventually, an extract of the **conditions of operational service** focused on the context of the component
- **textual requirements**, allocated to the component, and its accompanying definition above

For a hosting resource component it describes

- **links, ports or hardware interfaces** with its environment (other hosting resource components, actors external to the system)
- **links and interfaces** with the host component containing it
- the **resources** that it has to make available to behavioral components
- the **associated environmental and regulatory characteristics**
- **textual requirements**
- the **constraints of the product line**

07:16

Consolidation of the definition of components to be acquired

- The definition of the previous contract should in principle also apply to components that are not meant to be produced but acquired, and this can also apply to preexisting off-the-shelf components.
- It is strongly recommended to **give as much importance to the analysis of existing components (acquired or reused)** as to others, following the previous approach, most especially in the physical architecture.

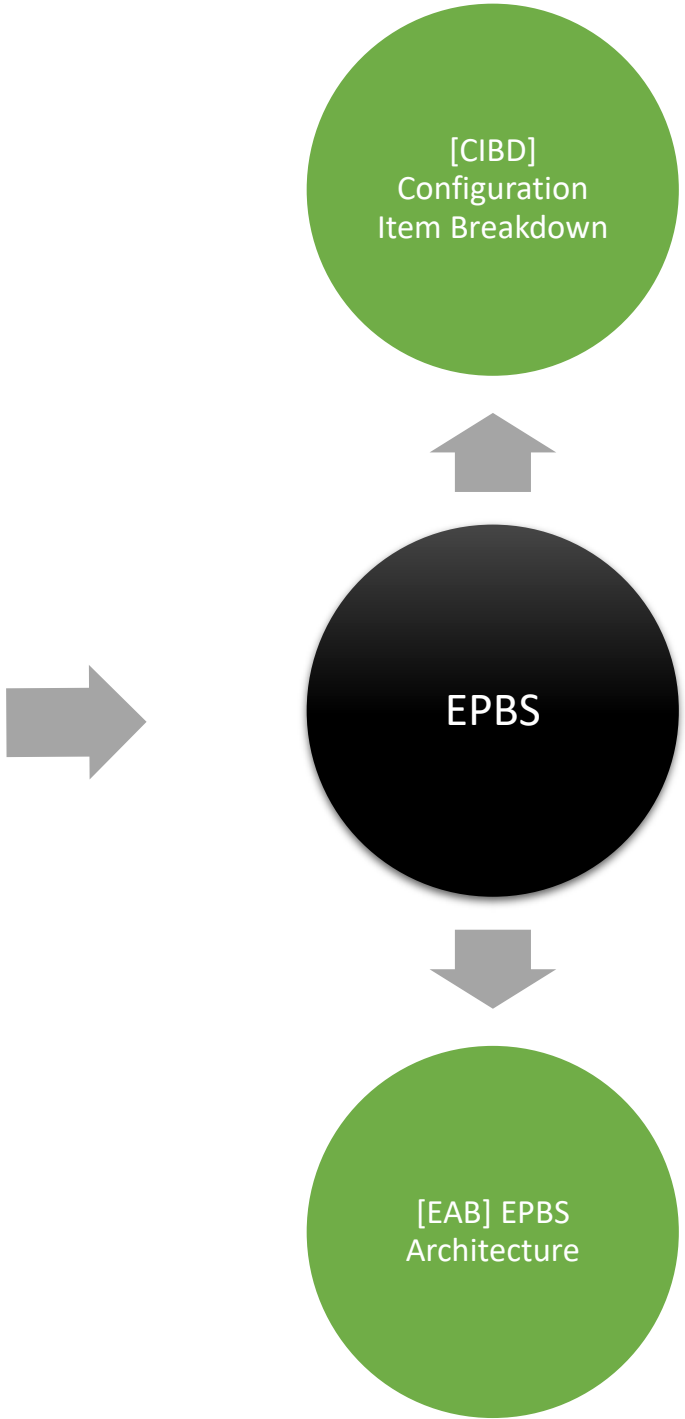
Definition of the IVV strategy

- The IVV strategy defines the **order in which operational and system capabilities will be delivered and verified**, the order in which components and their functions will be integrated and tested, and the conditions to achieve this: **namely, the nature of verification, the content of testing campaigns and required test means and testbeds**.
- The nature of the verifications demonstrating the adequacy of the system or product with the need is often characterized by the acronym IADT, for Inspection, Analysis, Demonstration and Testing.

- Demonstrations and tests will bring forward the demand exerted on the system by scenarios that will rely on those described in the model; the expected behaviors that the IVV will have to verify will be partially characterized by this same model, particularly by following the functional chains and the behavioral description that it comprises; similarly, the model provides invaluable help to the investigation, analysis of identified defects and their localization.
- In a number of cases, the **optimization of the IVV will require feedback into the architecture design in order to make it more suitable for stimulating**, observing and analyzing the functioning, but also for the progressiveness of tests or for facilitating the localization and containment of errors and defects.

EPBS DIAGRAMS

Initialization and automated update of the EPBS Architecture according to the Physical Architecture model. Define additional Configuration Items if necessary



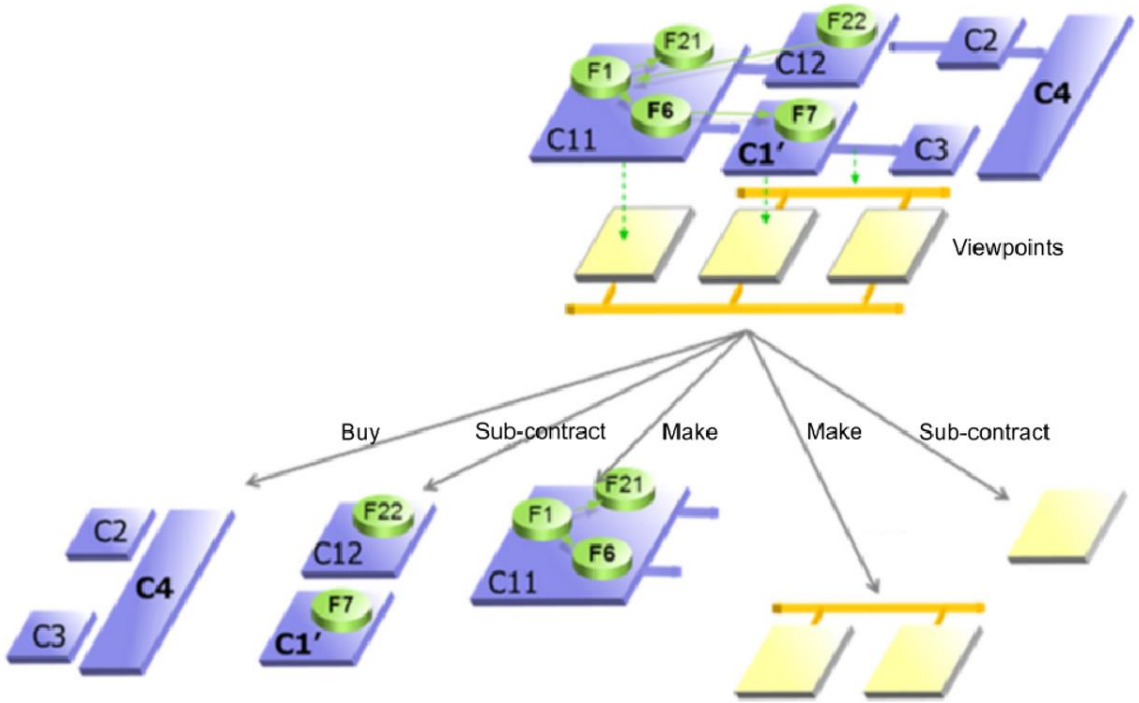
07:16

- NOTE – This level is much lower than the four previous ones, in terms of concepts, diagrams and methodological activities. In certain recent Arcadia presentations, it is no longer even represented as an engineering level, but rather as an “industrial organization” viewpoint on physical architecture.

STEP BY STEP EXAMPLE

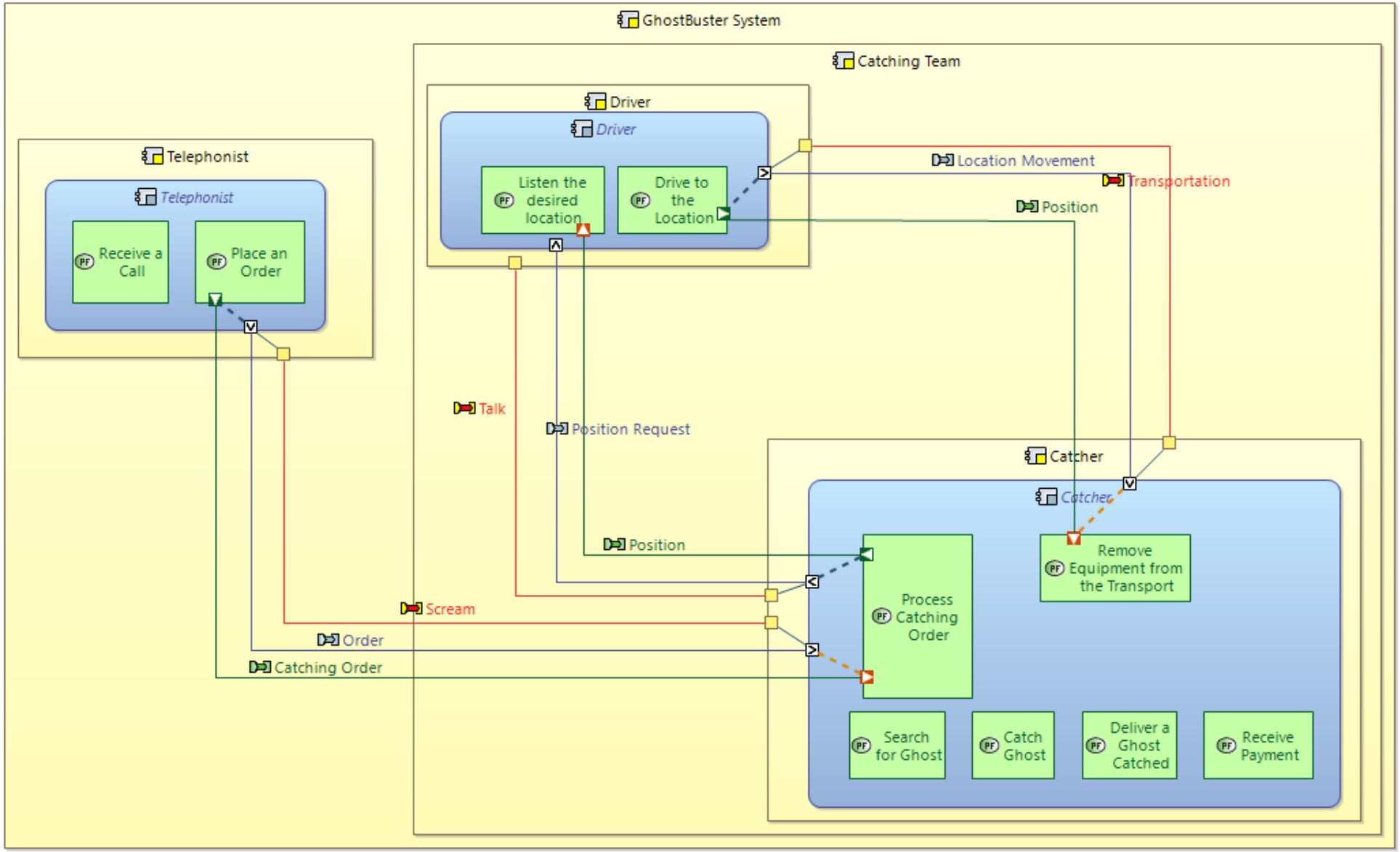
Moving from the Physical level to the EPBS level

- This level aims to use the **Physical Architecture to deduce the conditions that each component must fulfill to satisfy the design constraints and architecture choices**, identified in the previous phases ("*what is expected of the provider of each component*").
- The Physical Components are often gathered into **Configuration Items that are larger and more practical to manage**, in terms of industrial organization and responsibilities.
- The classic problem consists of asking ourselves whether we are going to **make the component**, **reuse a similar one**, **buy it off the shelf**, **or subcontract it out**, etc.

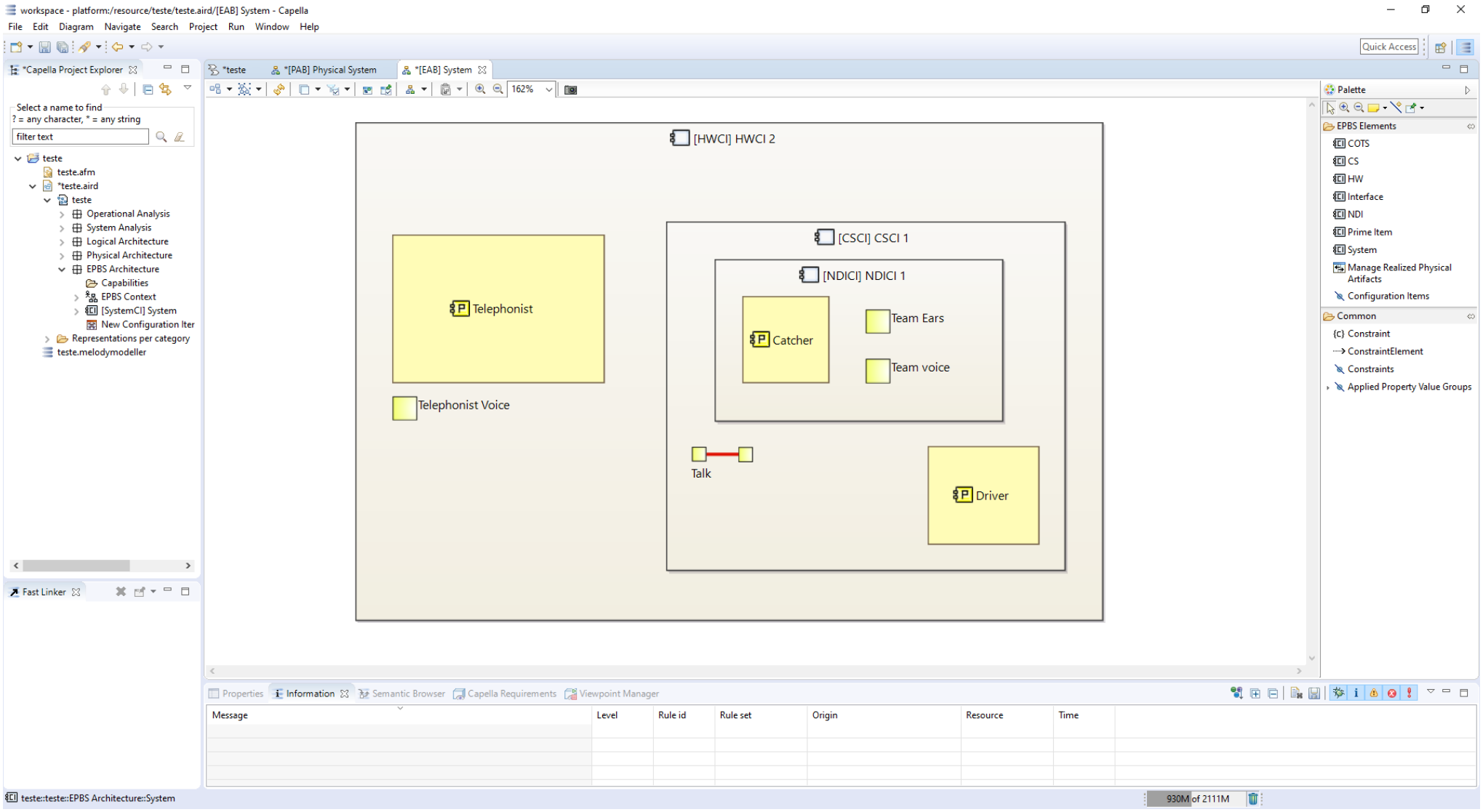


07:16

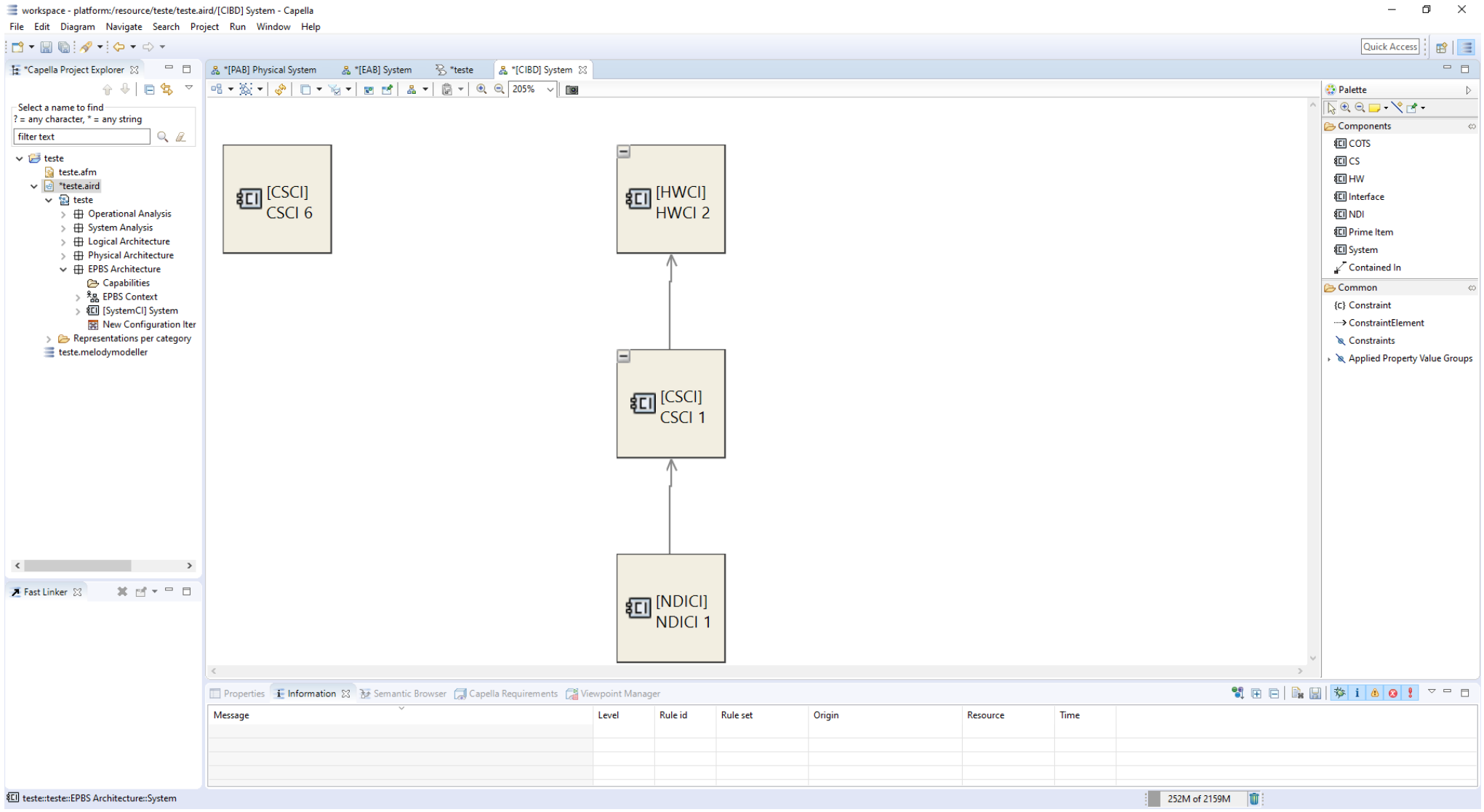
From the PAB of the System



[EAB] EPBS Architecture

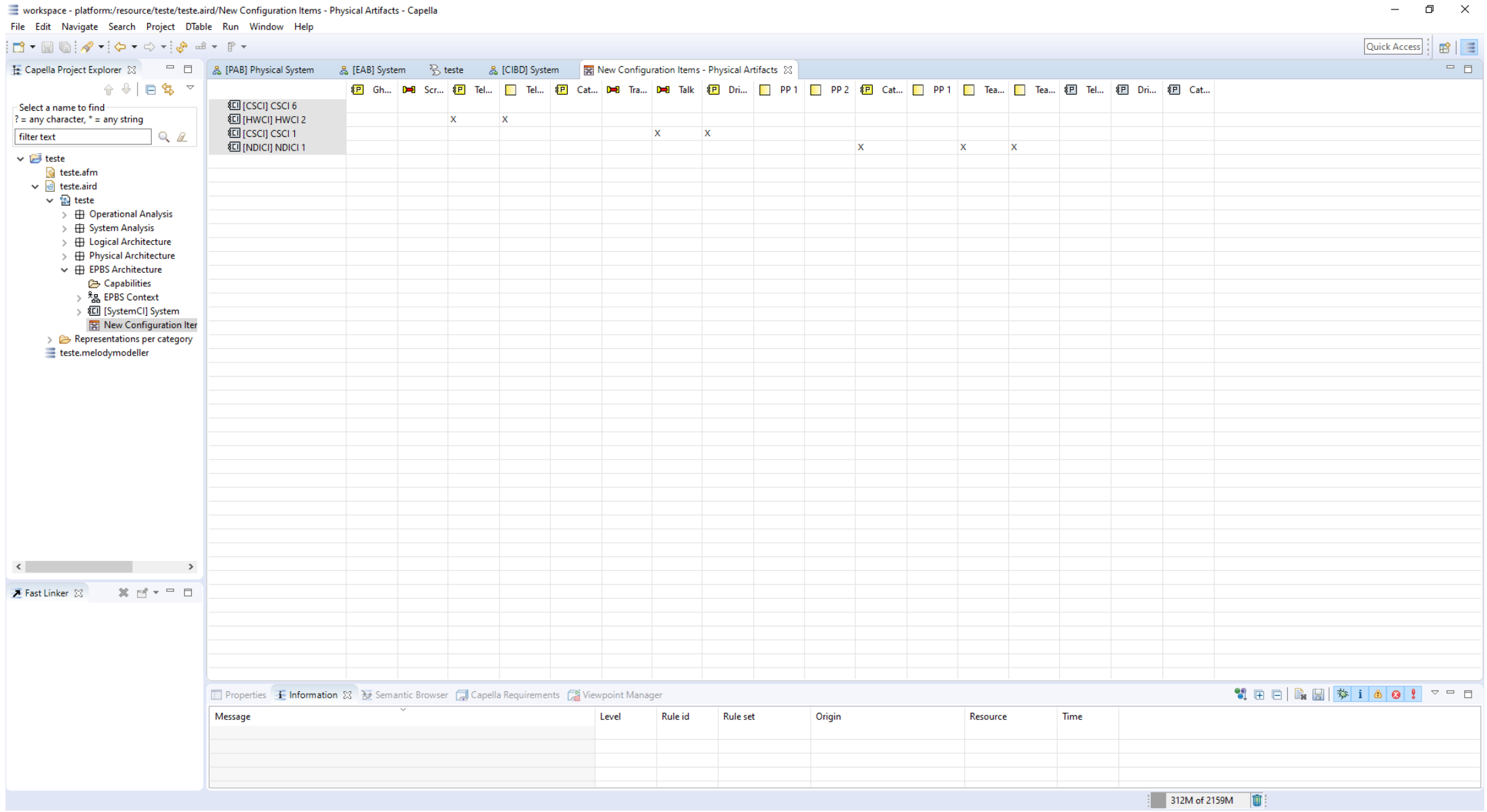


[CIBD] Configuration Items Breakdown



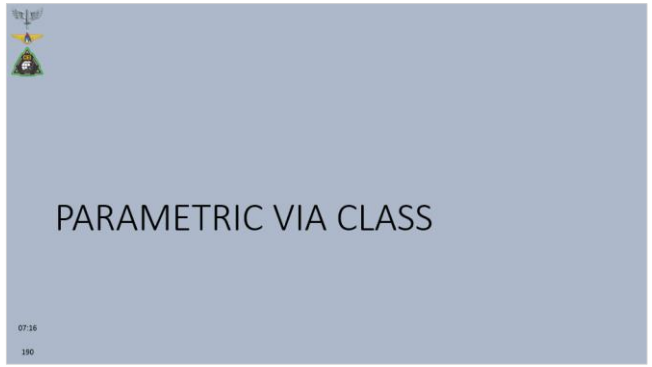
07:16

Traceability Matrix



TRANSVERSAL MODELLING

TRANSVERSAL MODELLING TOPICS



07:16

MODES AND STATES

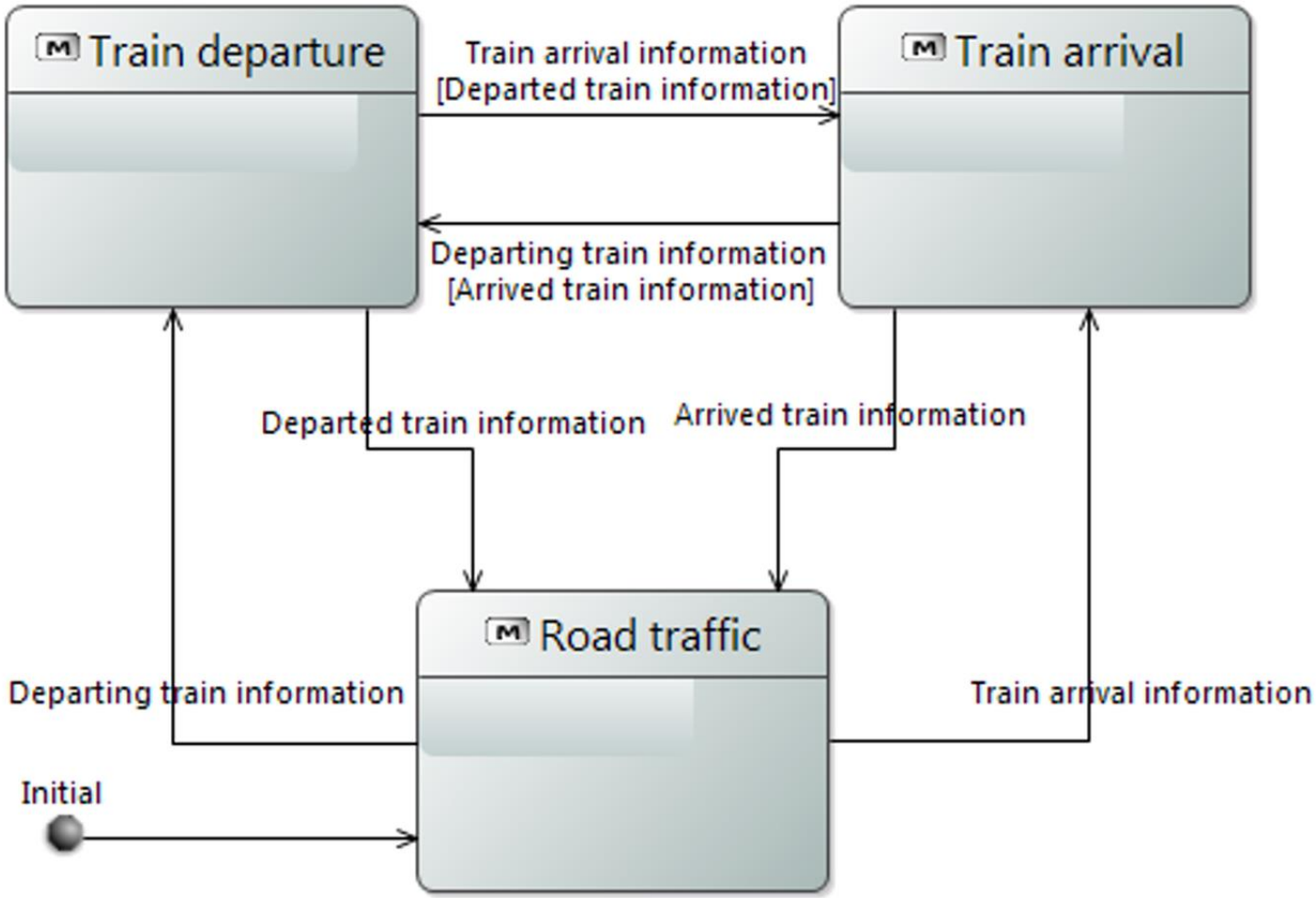
Modeling system modes, states, configurations with Arcadia and Capella: method and tool perspectives - 27th Annual INCOSE International Symposium (IS 2017)

07:16

MODES

- The definition of the **system's expected behavior** (or therefore, of one of the elements mentioned earlier) **in situations decided from the design is captured in the form of system *modes***; each mode is characterized principally by the functional content expected of the system in this mode (as a mnemonic, we talk of a "mode of life" to express the different expectations, priorities and activities in a life, and a "mode of transport" to indicate the means of travel). A mode can convey various concepts, such as a mission or process stage, a particular behavior required of the system, conditions of use such as a test or maintenance mode, a training mode, etc.

- *As its principal modes, the traffic control system will naturally have the modes characterizing the principal situations it should manage: train departure, train arrival and road traffic.*



STATES

- In the course of its life and use, the **system also passes through some states it undergoes** (we say “What a state you are in!” and we speak of a “state of alert or of emergency” to indicate an unexpected situation). Most often, a state characterizes mostly structural elements (presence or absence of a component, availability or breakdown, integrity or lack of it, availability of an external actor or loss of connection with it, etc.).
- **Transition from one state to another is often involuntary**, and will therefore result, for example, in a change in property for one or more elements in the system (availability/unavailability for example).

- *The level crossing can be found in a state occupied by a vehicle stuck on the track, or on the contrary, free (as expressed by the states of the control system itself). This situation is of course foreseen, but not on the initiative of the system, so it is undergone by the system, which must consequently react.*

CONFIGURATION

- To characterize the system when it is in a given mode or state, we will define the notion of *configuration*: a configuration identifies a set of model elements, of all types (for example functions, components, exchanges, etc.), globally involved in use of the configuration, at a given instant. A configuration can be attached to one or more modes and/or states.

- A configuration intended to describe the expectation of a mode will tend to be (though not exclusively) function dominant (capabilities, functions, exchanges, functional chains and scenarios, etc.) to express the expected functional content – or if it is easier to express, the functional content not present in this mode.
- A configuration intended to describe a state may be structural dominant (hosting physical components, physical links, indeed behavioral components hosted on the former, etc.), but could also include functional aspects, depending on the nature of the states considered (for example attack or failure scenarios, from a security viewpoint).

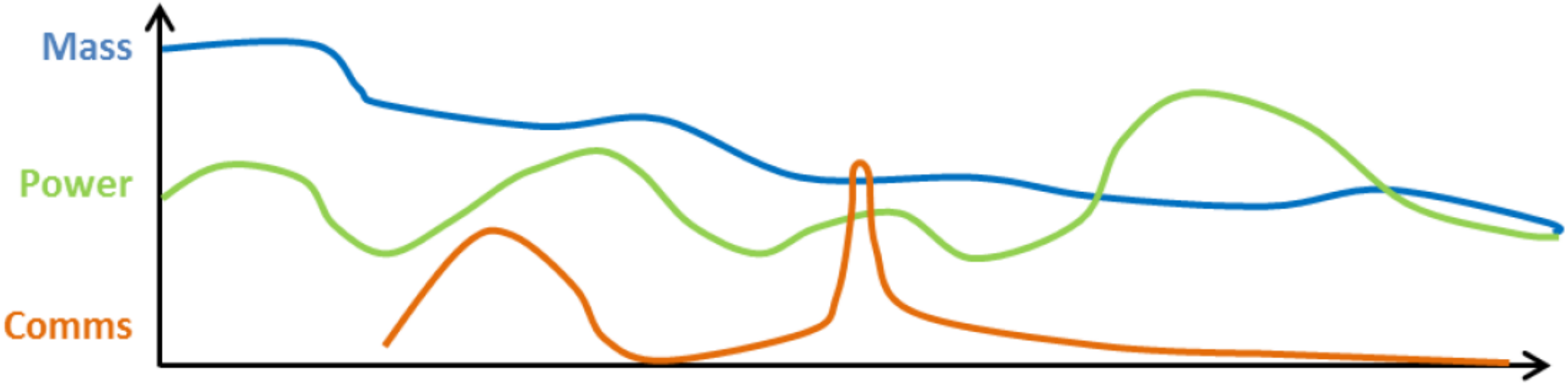
SCENARIO

- It is therefore necessary to define the combination of these states and modes to be able to study their consequences. For this, we will use the notion of *a situation of superposition*. A situation is defined as a logical combination of modes and states (for example (mode1 AND state1) OR (mode2 AND (state2 OR state3)), which would express the superposition of modes and states likely to occur at a given instant.
- A scenario can mention the transition from one situation of superposition to another, in the same way as it will mention changes of states and modes in the course of time.

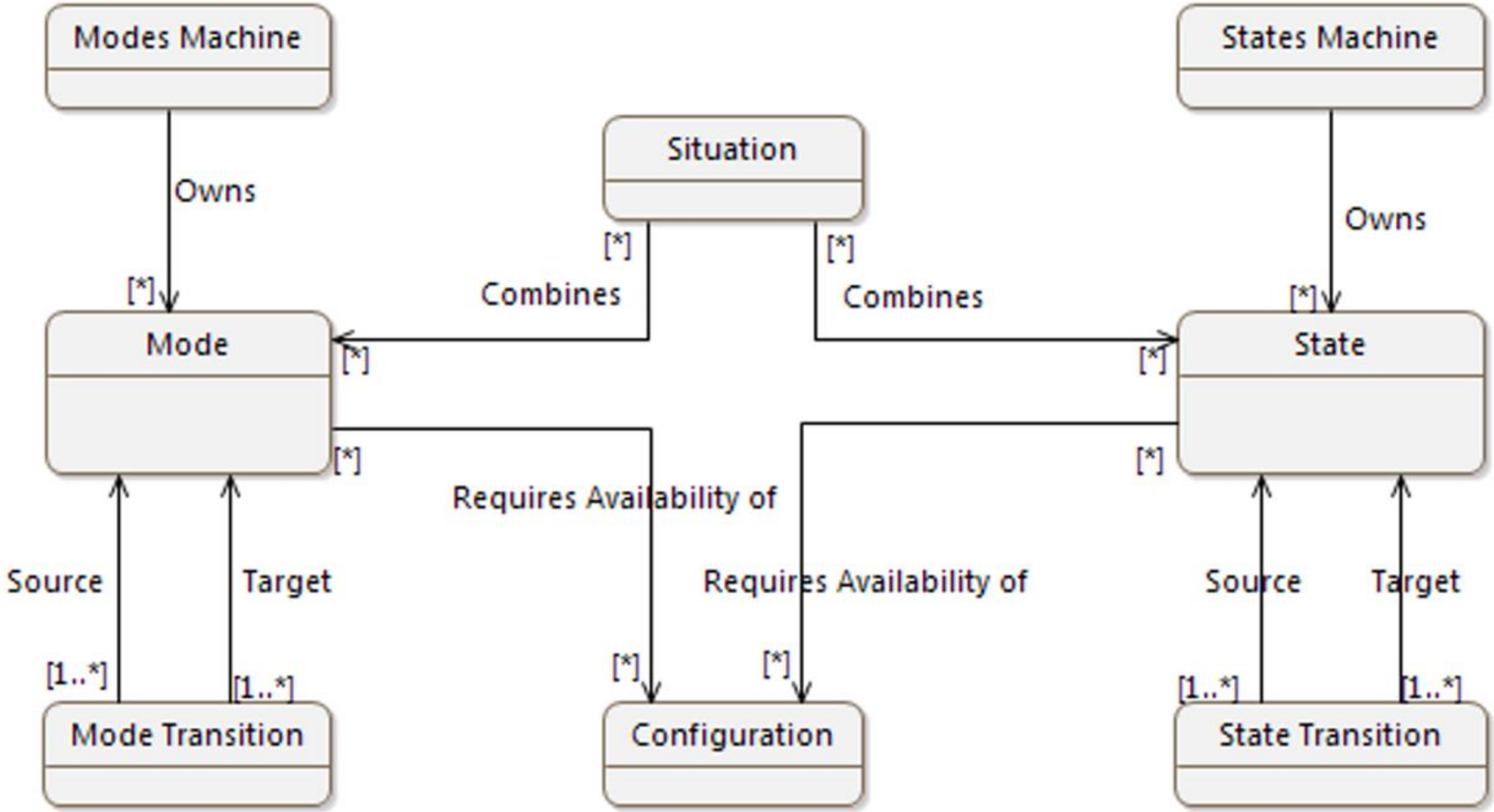
EXAMPLE OF SITUATIONS

Mission	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5	Phase 6

System		Mode 1	Mode 2	Mode 3	Mode 4	Mode 2	Mode 3	
Subsystems	1	Mode A	Mode B	Mode C	Mode A	Mode C	Mode A	
	2	Mode X	Mode Y	Mode Z		ModeX	Mode Y	
	3	Mode I	Mode J	Mode I	Mode J	Mode I	Mode J	



	MEANING OF STATES AND MODES
OA	describe either **general situations that the organization considered confronts** (usually rather states such as routine conditions, states of crisis, a situation where there is a lack of resources, for example), or the stages of a mission, or of the organization's normal functioning (usually rather modes, such as an airplane's or space launcher's stages of flight).
SA	describing the **expectation on the system**, as desired by the customer; they are most often perceived and employed by the final users. In particular, they capture the different modes and conditions of use required of the system in different situations, and feared situations, with the minimal behavior required when facing these situations
LA	the system states and modes respond this time to **design choices or constraints**. New modes and states reflecting the choices of solutions can appear, which cannot be linked to those of need analysis.
PA	applied to the system, but also to **each logical architecture component** and to the physical architecture components linked to it: modes and states, as well as the content of their associated configurations, should be coherent with traceability links (between functions, between components, between exchanges triggering transitions, etc.) between both architecture perspectives.

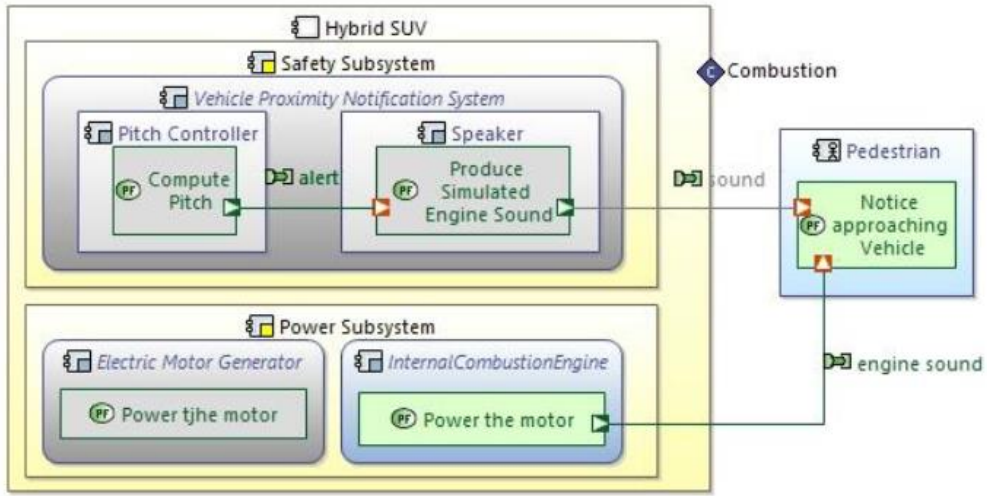

Modes Machine
States Machine
Situation
Owns
Owns
[*]
[*]
[*]
[*]
Combines
Combines
Mode
State
[*]
[*]
[*]
[*]
Requires Availability of
Requires Availability of
Source
Target
Source
Target
[1..*]
[1..*]
[1..*]
[1..*]
[*]
[*]
Mode Transition
Configuration
State Transition

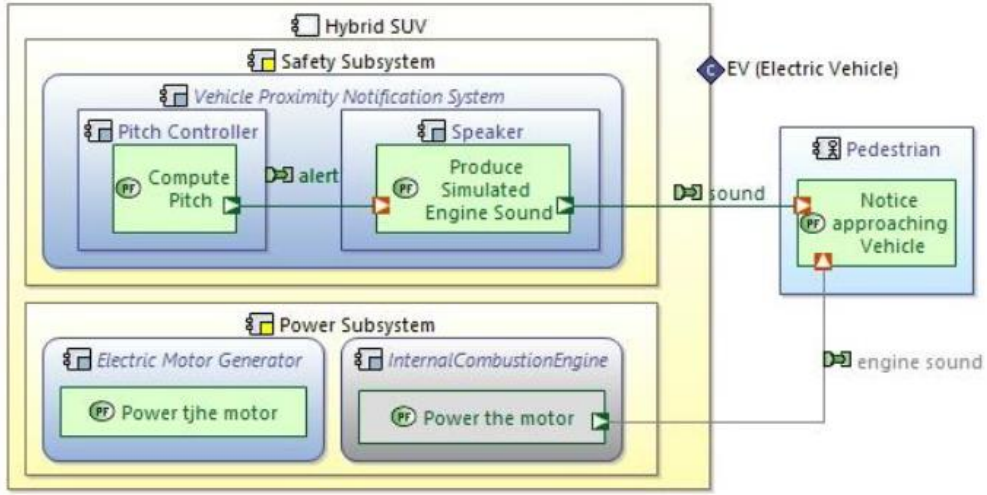
SUMMARY

- A **mode** is a **behavior expected of the system**, a component or also an actor or operational entity, in some chosen conditions.
- A **state** is a **behavior undergone by the system**, a component, an actor or an operational entity, in some conditions imposed by the environment.
- A **transition** is a **change from one mode to another** mode or from one state to another state (respectively, called the transition source and transition target).
- A **mode(s) machine** (or respectively, **state(s) machine**) is a **set of modes (or, respectively, states) linked to one another by transitions**. Modes and states cannot cohabit in the same machine.
- A **configuration** is a **set of model items that are globally available** or unavailable in a given context. A context can here be an active mode or state.
- A **situation** is a **combination of states and modes linked by Boolean operators** (of the type AND, OR, NOT), and representing the conditions of superposition of these states and modes simultaneously at a given instant.

ADD-ON on test to aid State Analysis (VPMS)

	EV ...	Hybrid	Combustion
safe vehicle approach			
safe pedestrian encounter in combustion configuration	X		
safe pedestrian encounter in EV configuration			X
Hybrid SUV			
Safety BCs			
Vehicle Proximity Notification System		X	X
Speaker			
Produce Simulated Engine Sound			
sound			
alert			
Pitch Controller			
bus			
Compute Pitch			
BrakeAssist			
Airbags			
Power Subsystem			
Power BCs			
Transmission			
PowerControlUnit			
InternalCombustionEngine	X		
FuelTankAssembly	X		
ElectricalPowerController			
Electric Motor Generator			X
Differential			
BatteryPack			

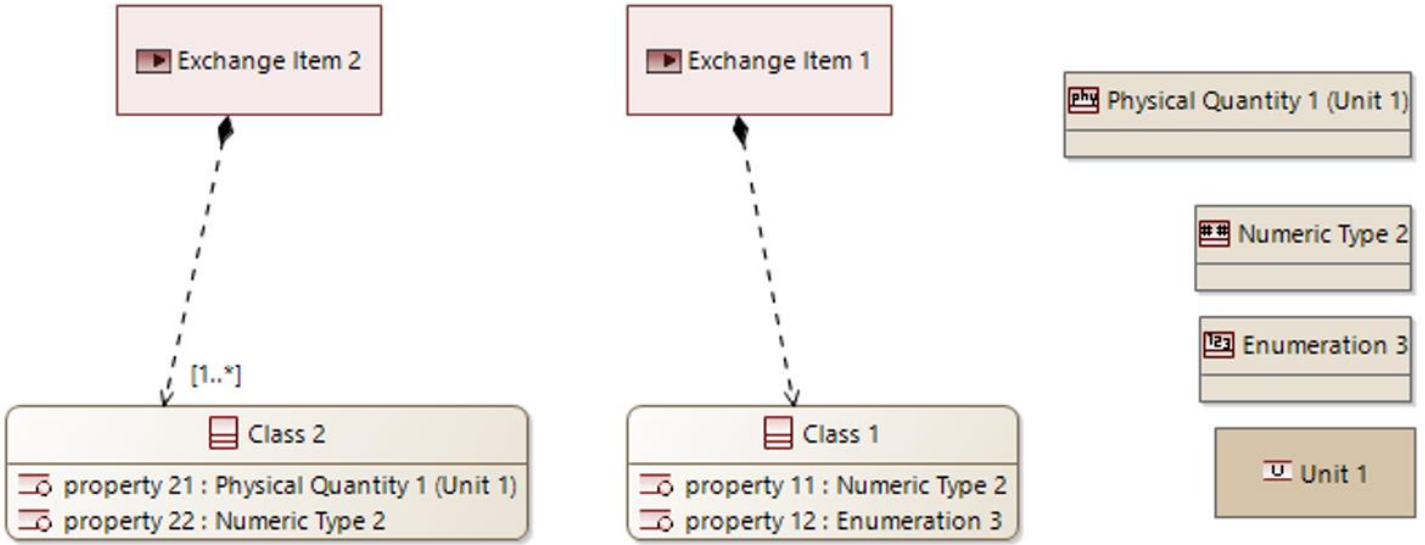




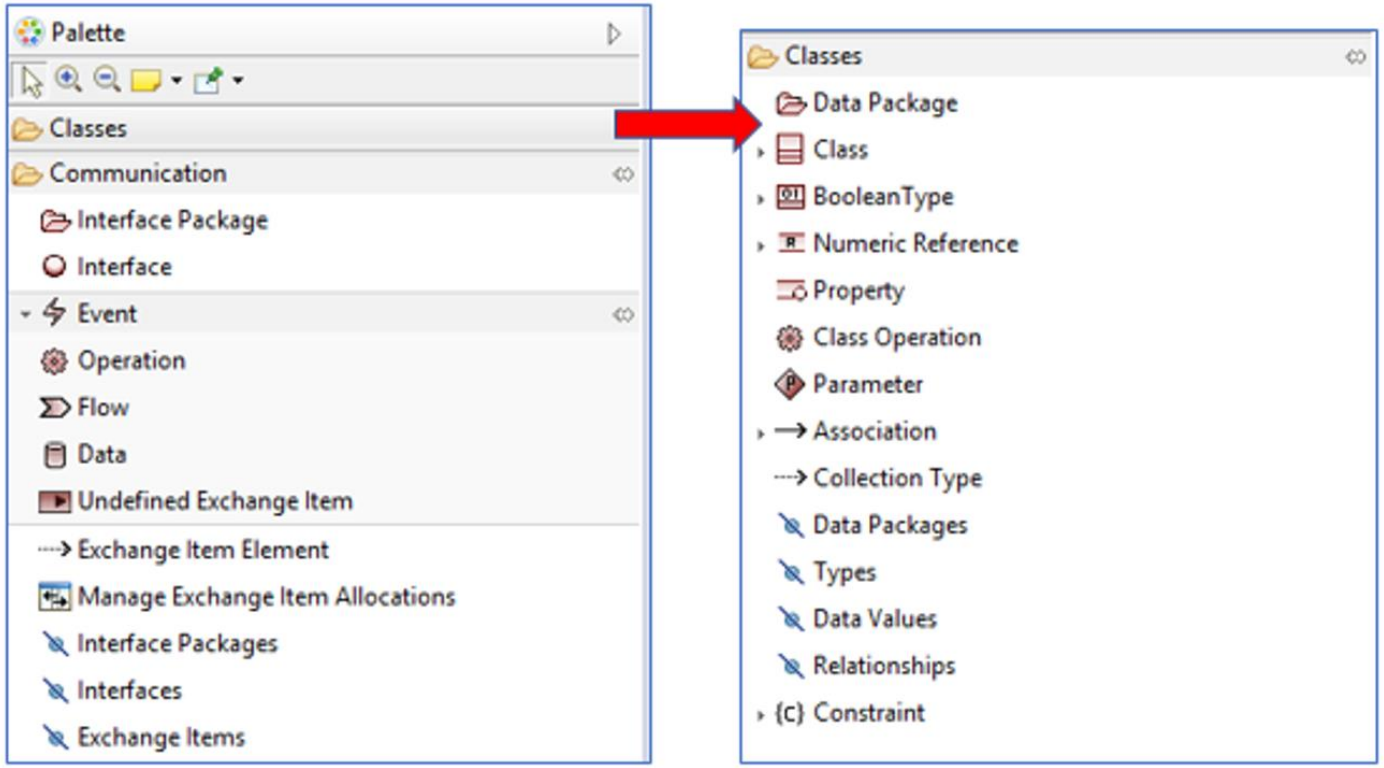
07:16

PARAMETRIC VIA CLASS

- Capella provides advanced mechanisms for modeling data structures at a stated level of precision and for linking them to Functional Exchanges, Component or Function Ports, Interfaces, etc.



- There are two main categories of concepts in this type of diagram:
  - Communication elements: EIs and Interfaces;
  - Type definitions: basic Types, Classes, relations between Classes.
- These two categories of concepts are taken into account in a division of the CDB's palette into two different groups.

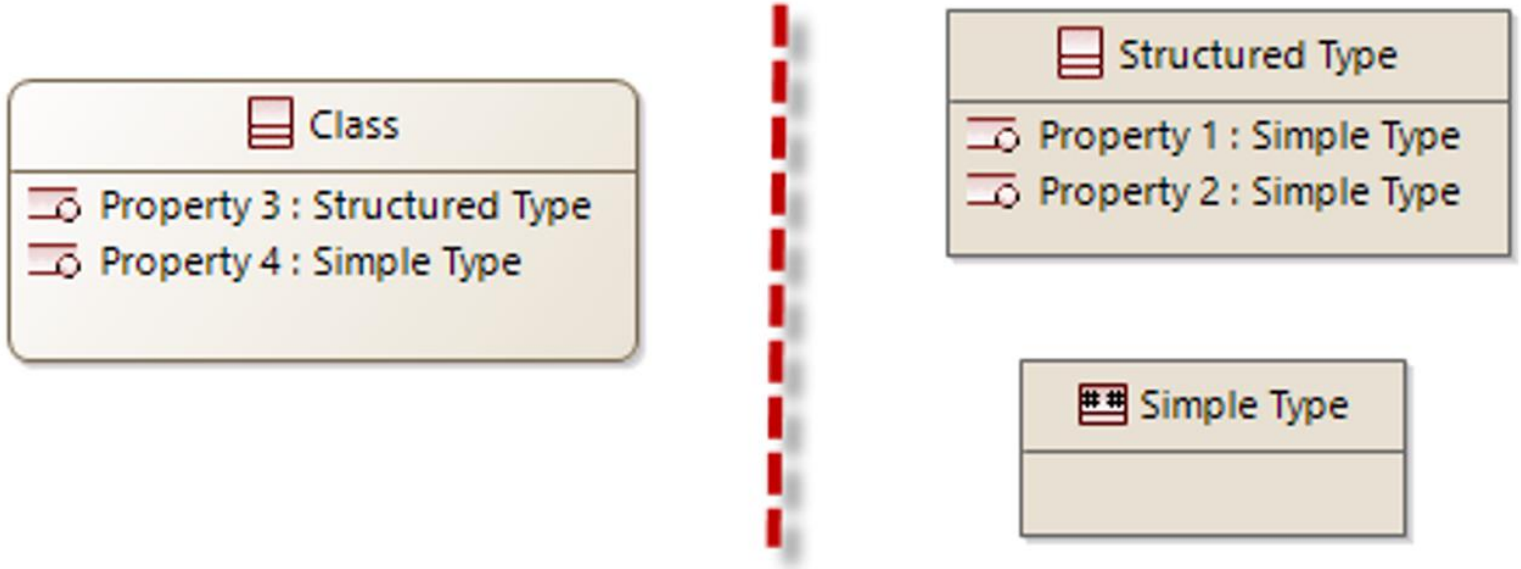


COMMUNICATION MECHANISMS

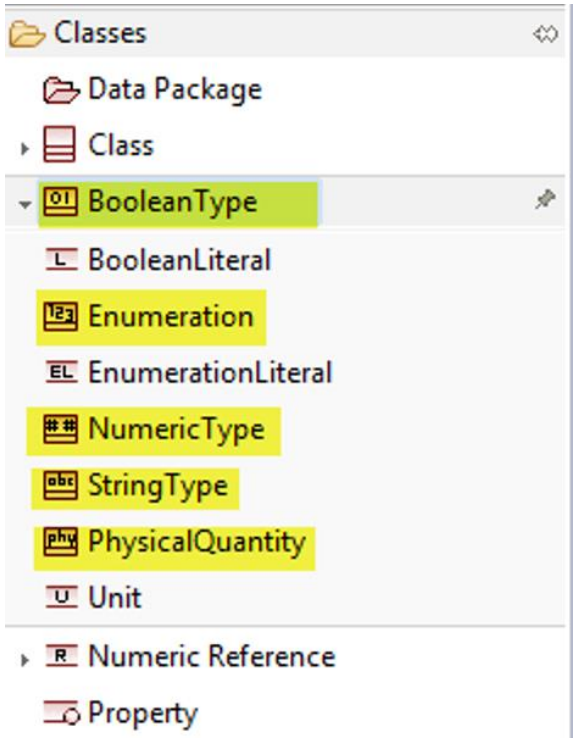
- EVENT: asynchronous mechanism where an event is sent by an element and received by one or several others;
- FLOW: flow of matter, energy, etc. or data;
- OPERATION: process carried out by an element and invoked by another;
- SHARED DATA: data modified by an element and read by others.

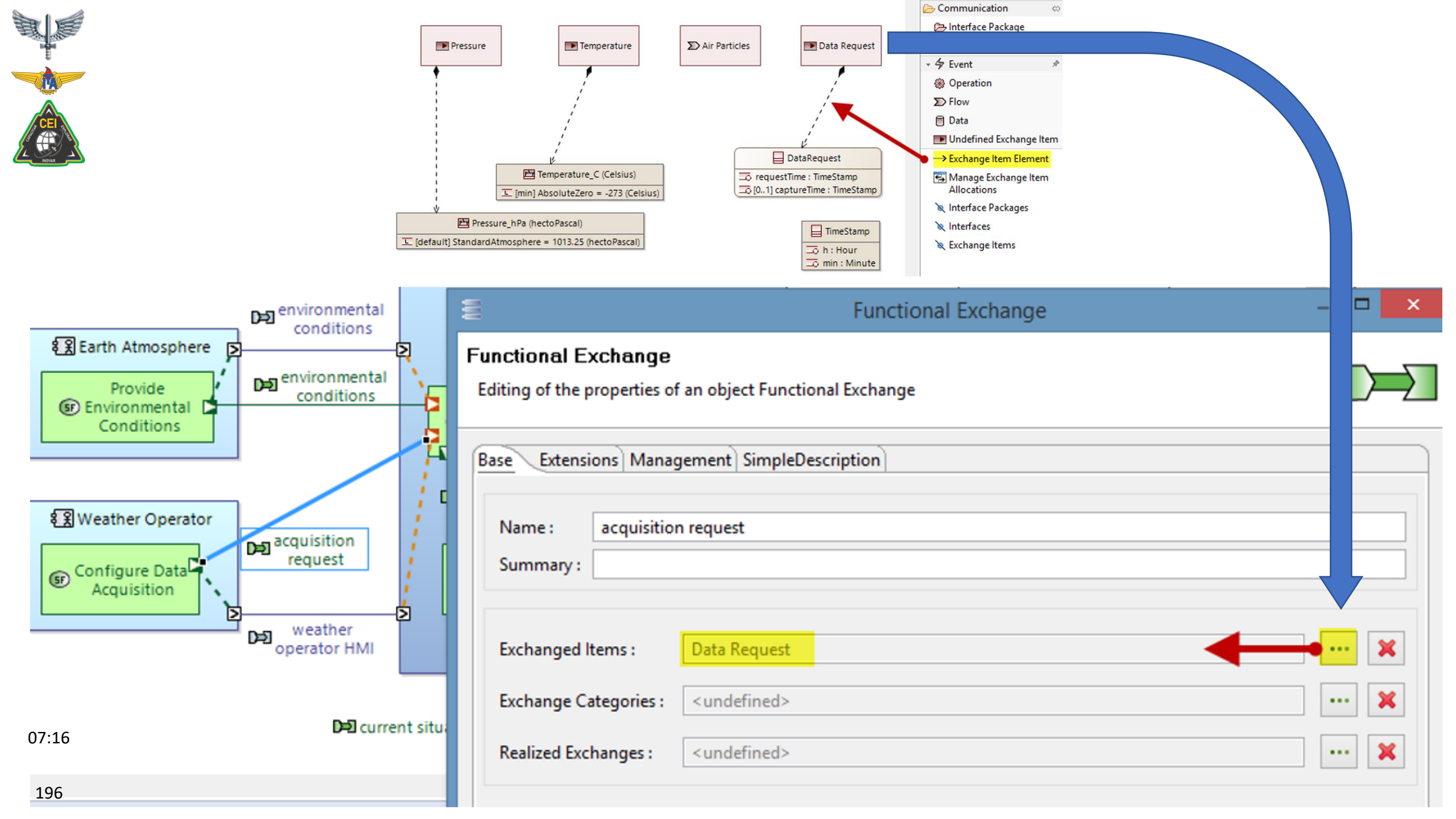
TYPES

- We shall look now at the definitions of the Types proposed by Capella: Classes, Structured Types, Simple Types. The vocabulary used comes from UML, and the very name of the “Class diagram” is a direct reference to it.



- The simple Types predefined by Capella are as follows: BooleanType, Enumeration, NumericType, StringType and PhysicalQuantity.
- Careful, BooleanLiteral and EnumerationLiteral help define Boolean Type and Enumeration, while Unit helps to define PhysicalQuantity.
- Simple types cannot have properties. If we want to define Structured Types, the Class button in the palette must be used, and then the Class must be specified as primitive (tick the box *Is Primitive*). The other “Primitive” Classes then play the role of structured Types, and can in turn type the properties of the “true” Classes.




Pressure
Temperature
Air Particles
Data Request
Temperature_C (Celsius)
[min] AbsoluteZero = -273 (Celsius)
Pressure_hPa (hectoPascal)
[default] StandardAtmosphere = 1013.25 (hectoPascal)
DataRequest
requestTime : TimeStamp
[0..1] captureTime : TimeStamp
TimeStamp
h : Hour
min : Minute
Communication
Interface Package
Event
Operation
Flow
Data
Undefined Exchange Item
Exchange Item Element
Manage Exchange Item Allocations
Interface Packages
Interfaces
Exchange Items
environmental conditions
Earth Atmosphere
Provide Environmental Conditions
environmental conditions
Weather Operator
Configure Data Acquisition
acquisition request
weather operator HMI
current situ
Functional Exchange
Functional Exchange
Editing of the properties of an object Functional Exchange
Base
Extensions
Management
SimpleDescription
Name :
acquisition request
Summary :
Exchanged Items :
Data Request
Exchange Categories :
<undefined>
Realized Exchanges :
<undefined>

07:16

Ending

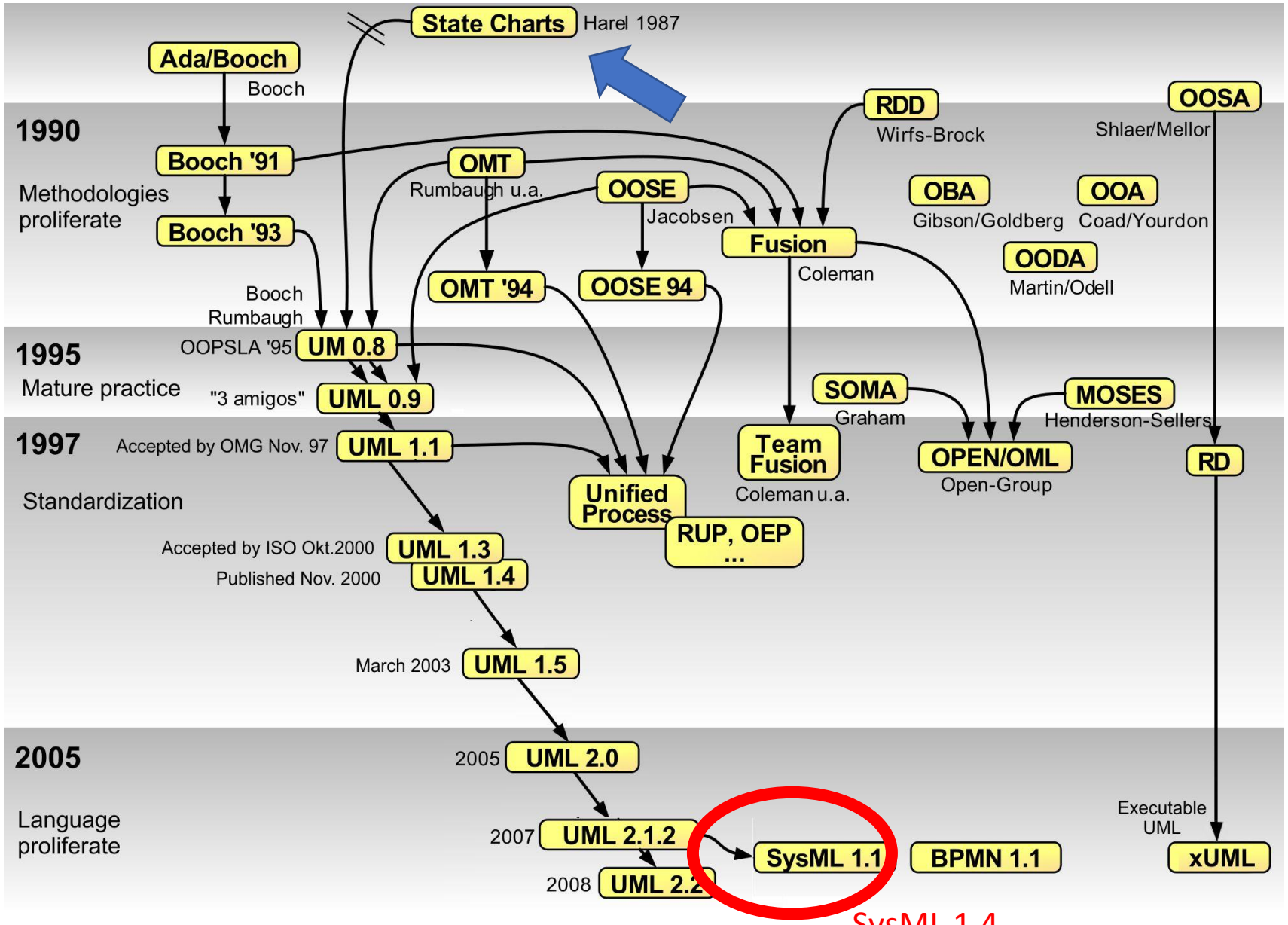
07:16

- It must be noted that the method does not always have to be top down in nature, but can also perfectly be bottom-up, for example if we start with an existing system that is to be worked on. The question relates more to architectural levels than to phases or steps.
- Moreover, not all architectural levels are mandatory for all projects. Operational Analysis, Logical Architecture and EPBS are considered to be optional, depending on the complexity of the system under study and the goals of the model.

BACKUP

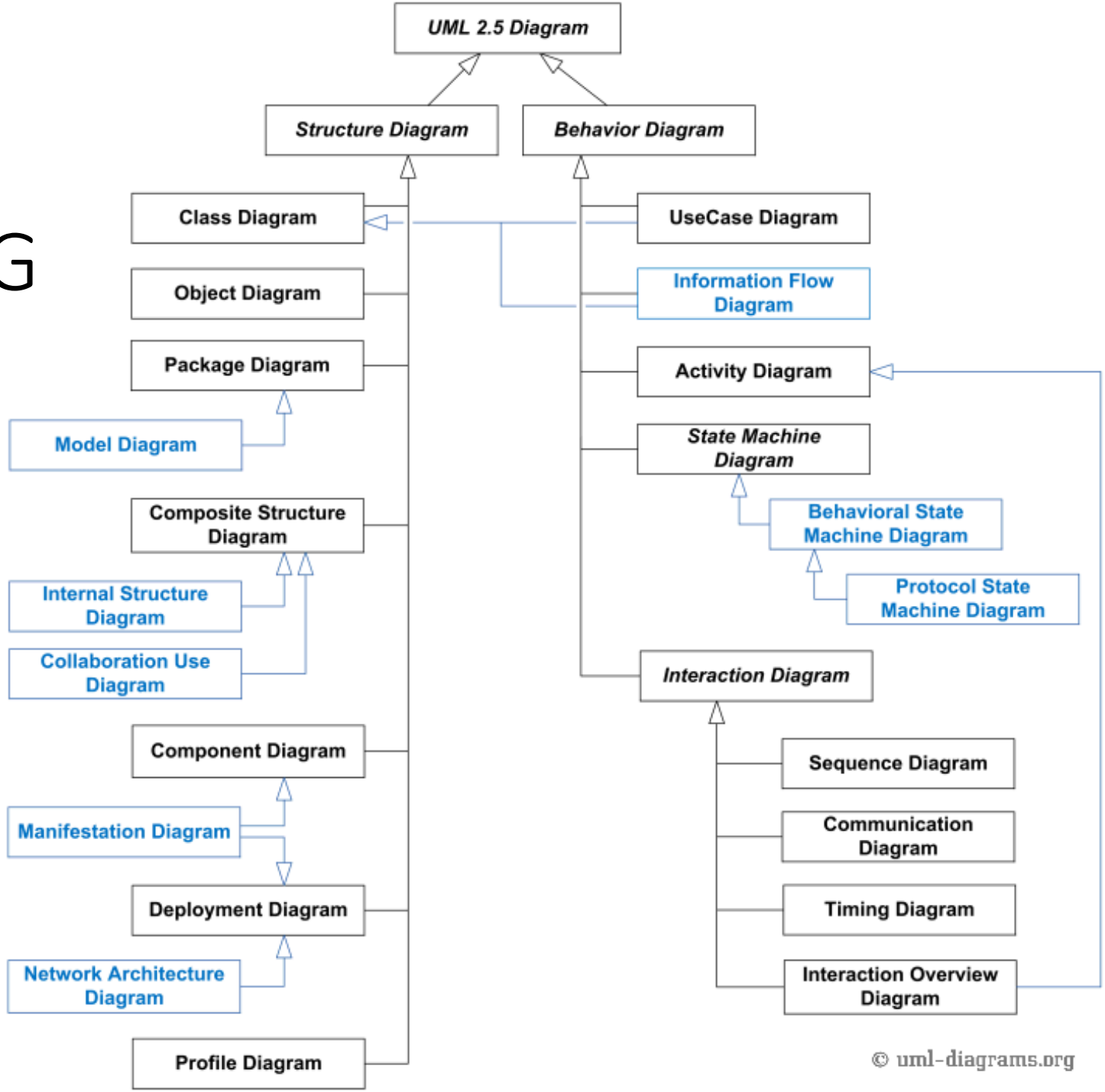
WHAT IS SYSML?

HISTORY OF OBJECT ORIENTED LANGUAGES

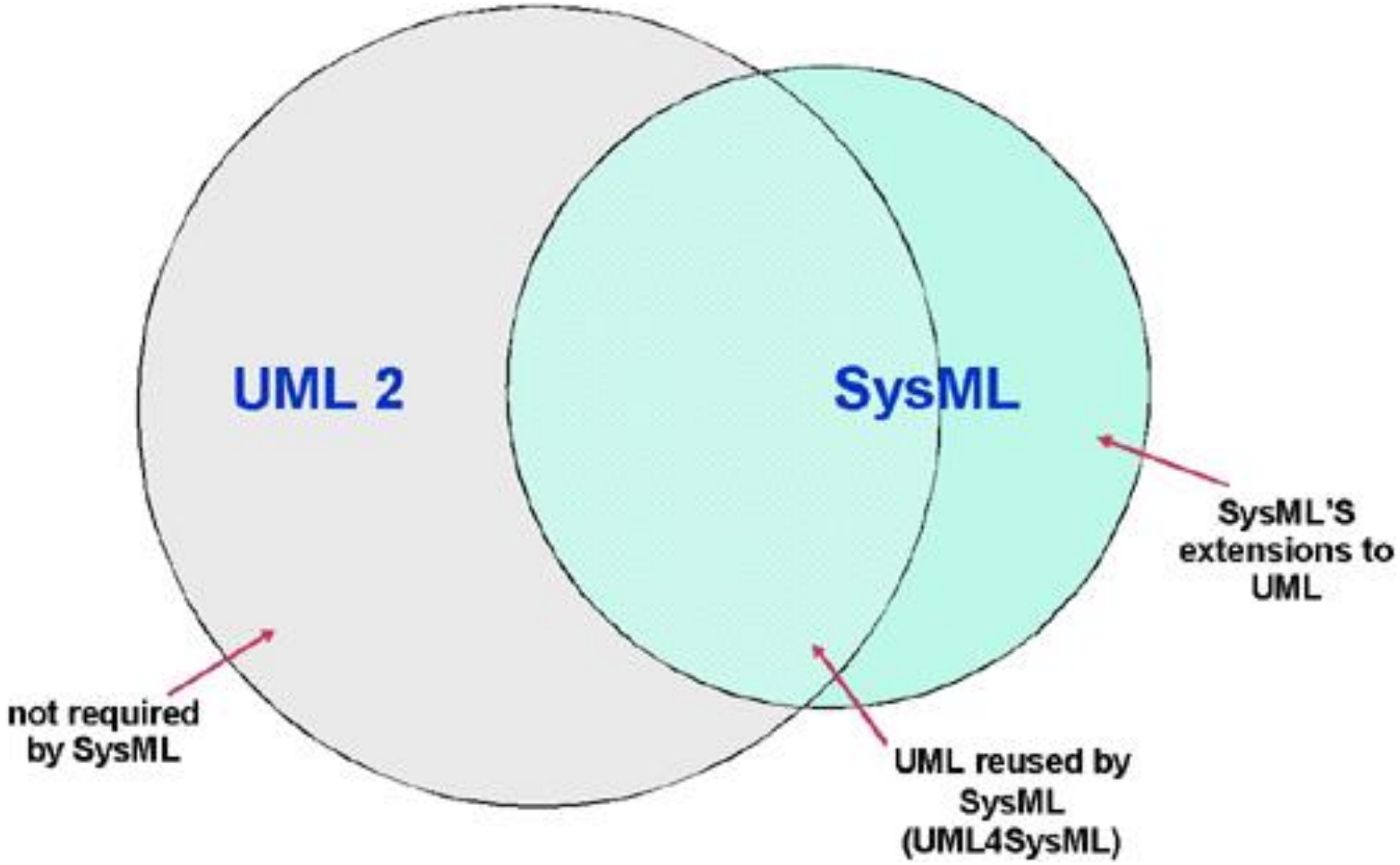


SysML 1.4

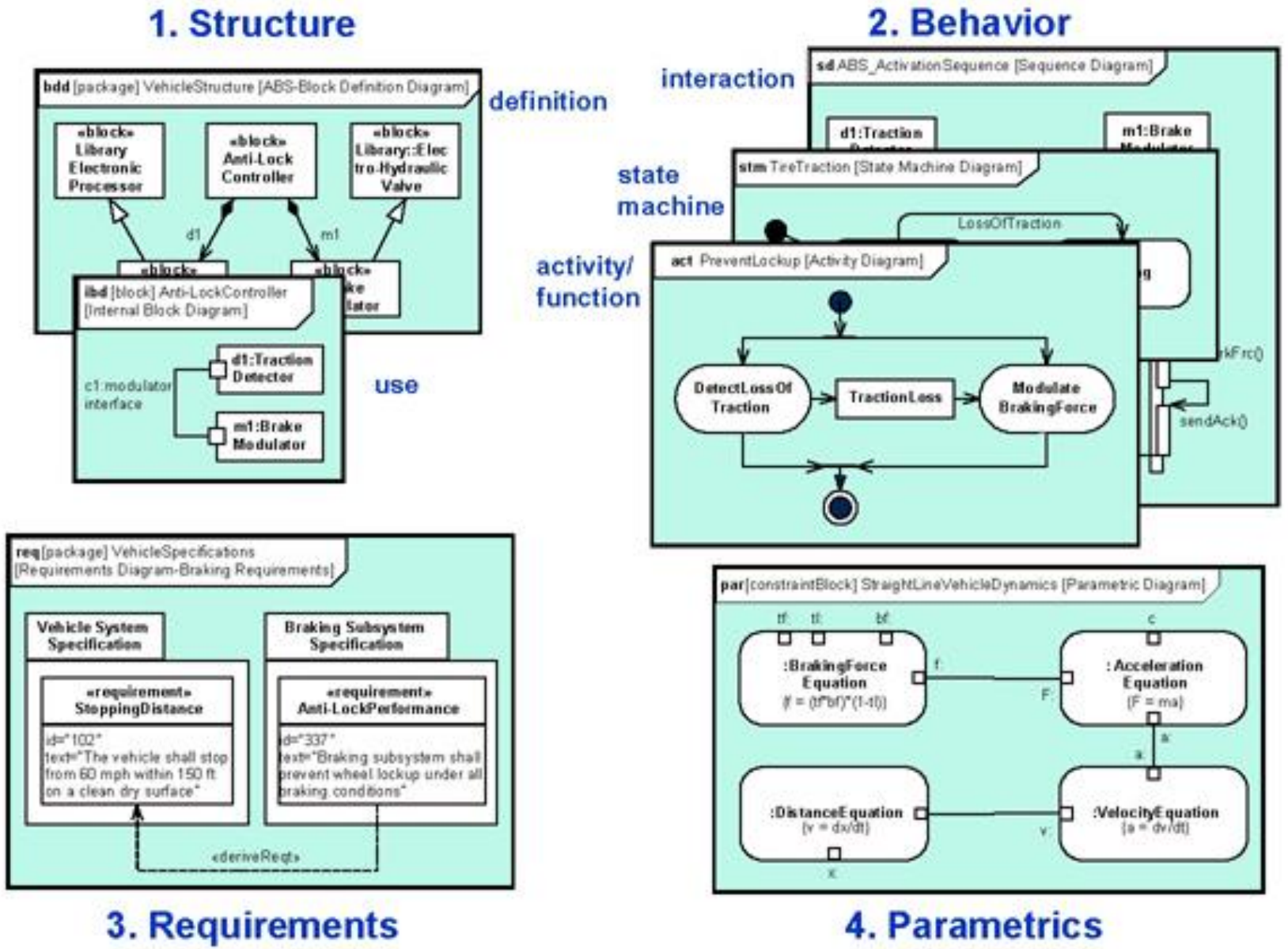
UML – UNIFIED MODELLING LANGUAGE



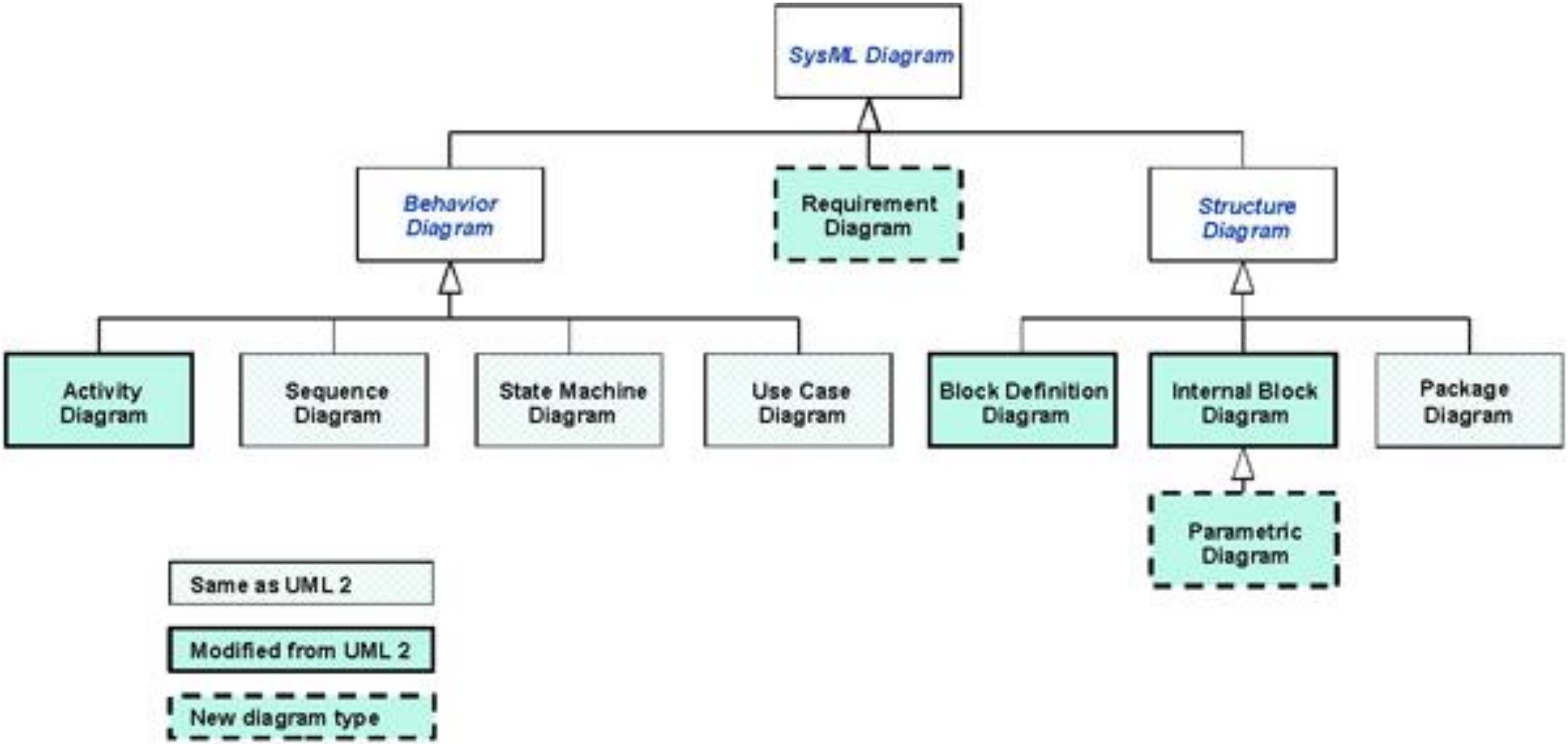
ADAPTATION OF UML TO SYSTEMIC DOMAIN



SysML – SYSTEM MODELLING LANGUAGE



SysML DIAGRAM BRANCHES



SysML

- Is a visual modelling language that provides
  - Semantics = meaning
  - Notation = representation of meaning
- Is not
  - a methodology or a tool
  - SysML is methodology and tool independent

Transition from OPM to SysML

Creating SysML Views from an OPM Model

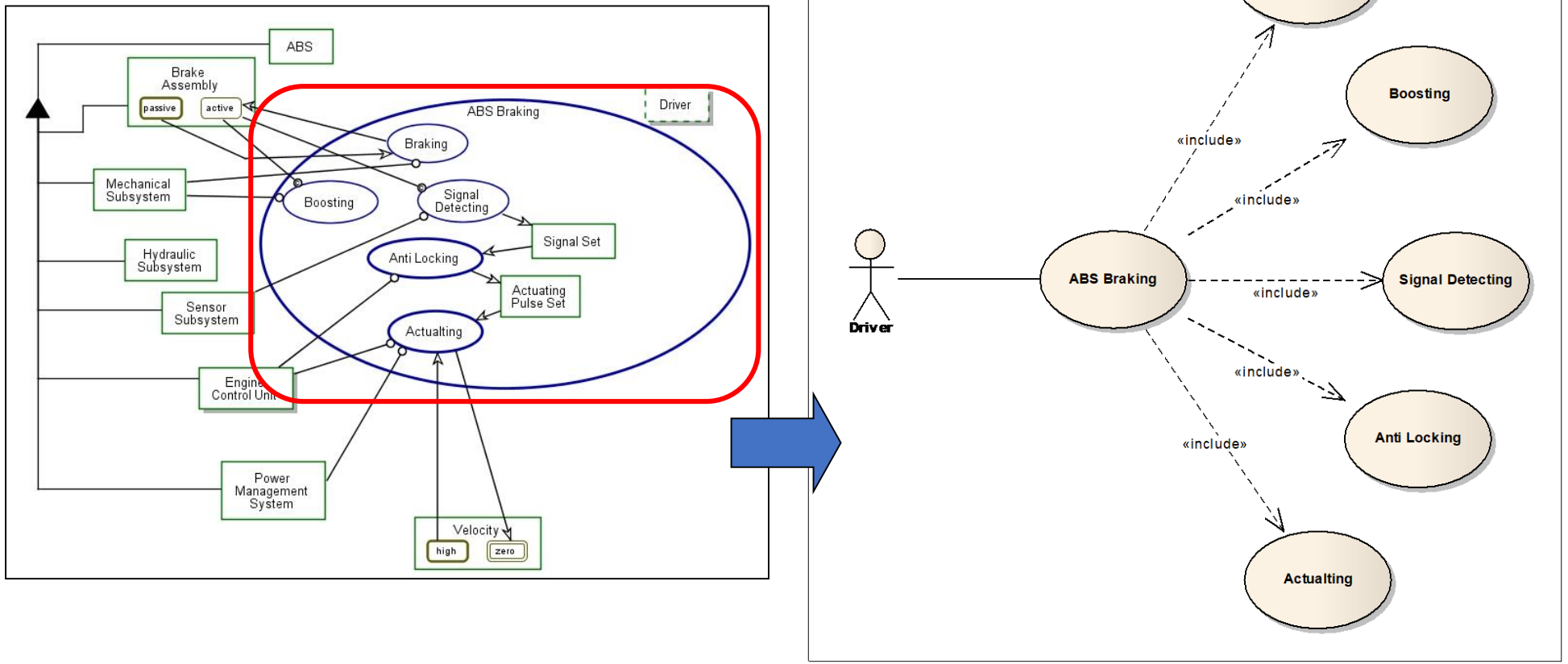
07:16

OPM to SysML Mapping Challenge

- The mapping is "**one-to-many**"
- Example – a Process in OPM can be mapped in SysML to one of the following:
  - **Use Case** (in a Use Case Diagram)
  - **Operation** of a block (in a Block Definition Diagram)
  - **Action** (in an Activity Diagram)
  - State transition **trigger** or in-state **activity** (in a State Machine Diagram)
  - **Message** (in a Sequence Diagram)

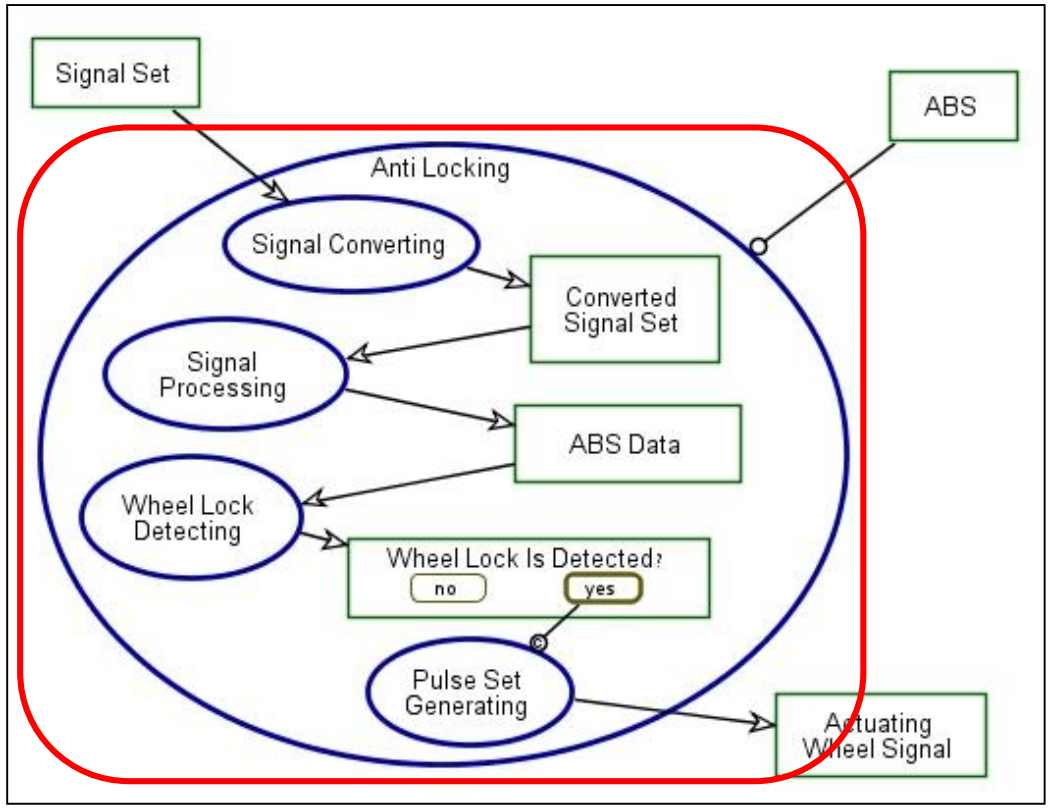
07:16

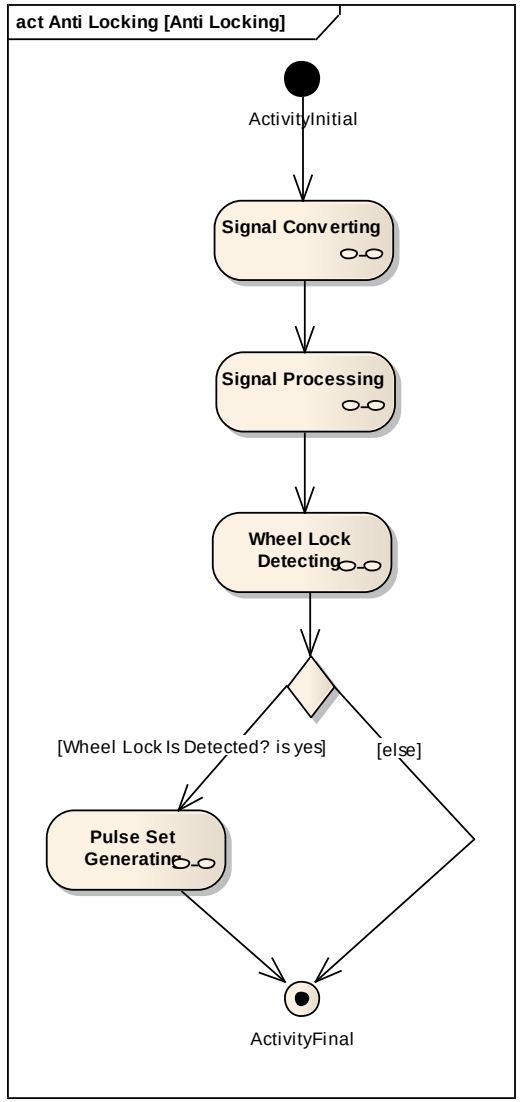
OPM-to-SysML IMPLEMENTATION - USE CASE EXAMPLE



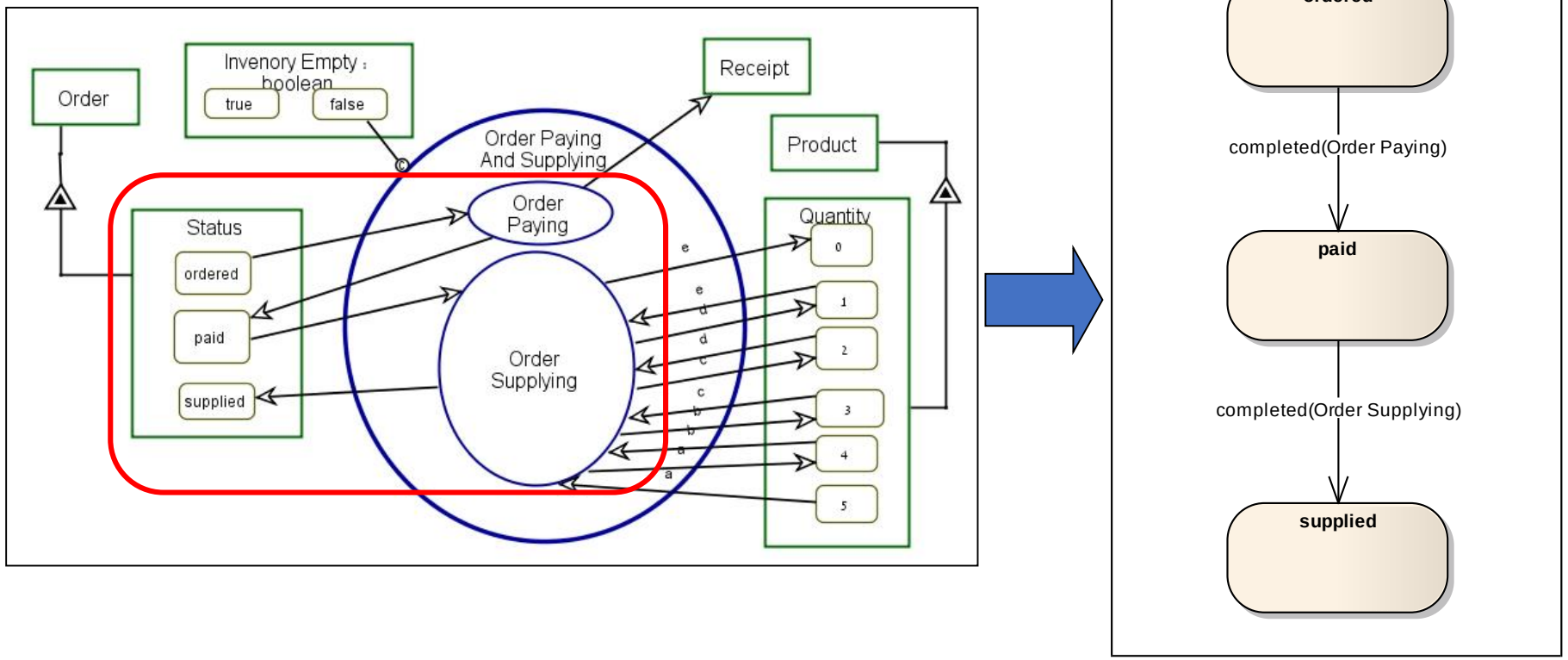
07:16

ACTIVITY/SEQUENCE DIAGRAM EXAMPLE

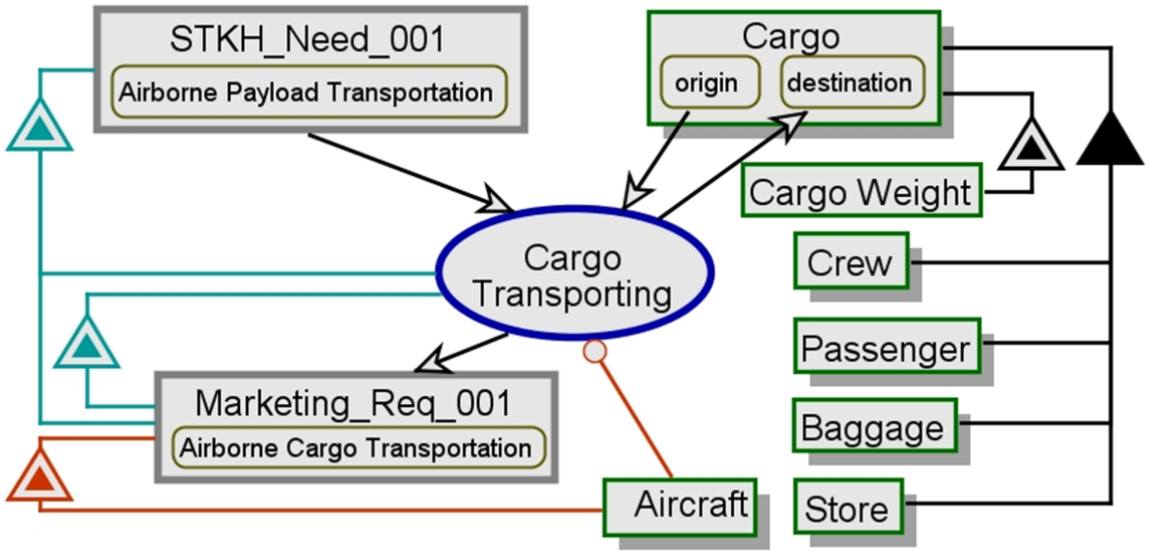




STATE MACHINE DIAGRAM EXAMPLE



REQUIREMENTS



STKH_Need_001 is Airborne Payload Transportation.
- Cargo Transporting consumes STKH_Need_001.
- Cargo Transporting yields Marketing_Req_001.
  - Marketing_Req_001 is Airborne Cargo Transportation.
    - Marketing_Req_001 exhibits Aircraft.
- Cargo Transporting requires Aircraft.
  - Aircraft is physical.
- Cargo Transporting changes Cargo from origin to destination.

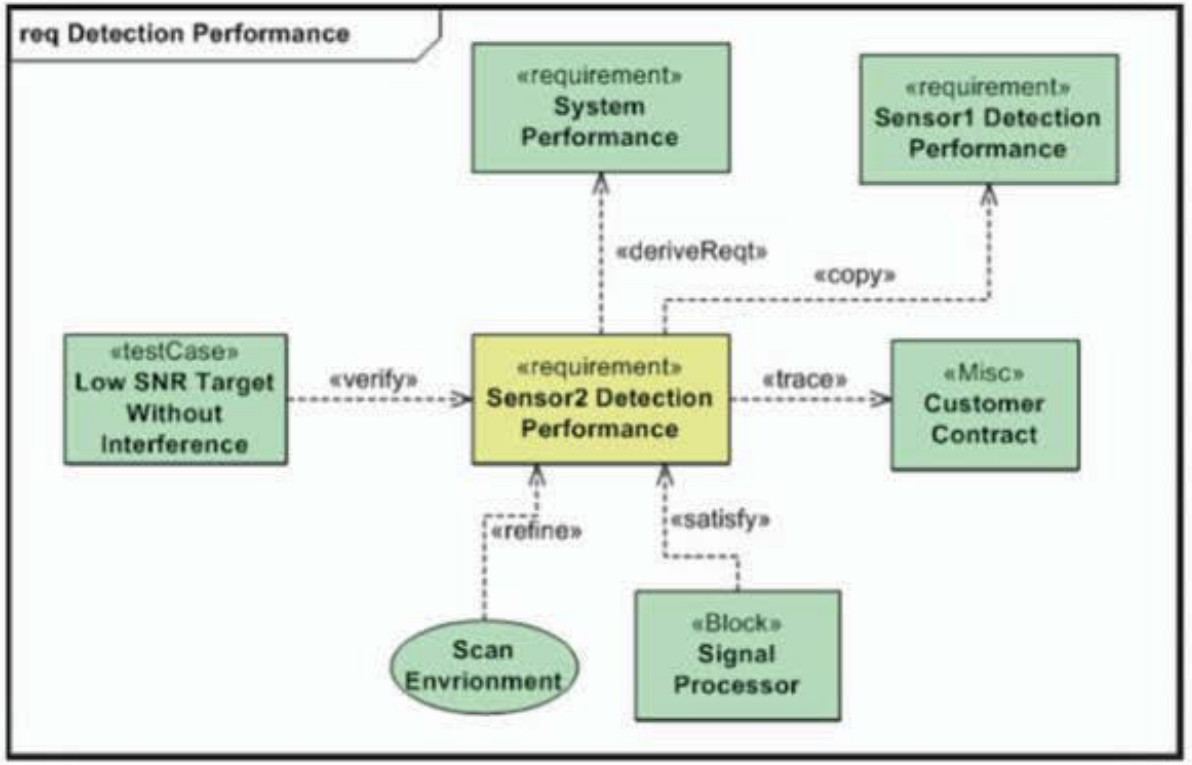
} Function Defining Requirements Identifying Requirements Allocating

STKH_Need_001 exhibits Marketing_Req_001, as well as Cargo Transporting.
- Cargo Transporting exhibits Marketing_Req_001.

} Traceability

Cargo is physical.
Cargo can be origin or destination.
Cargo exhibits Cargo Weight.
Cargo consists of Passenger, Baggage, Store, and Crew.
- Passenger is physical.
- Baggage is physical.
- Store is physical.
- Crew is physical.

} Configuration management

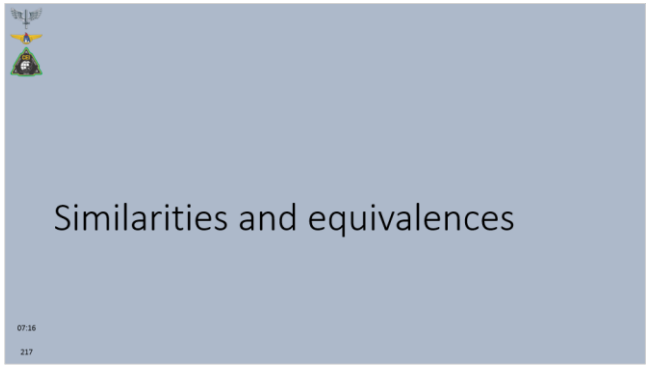


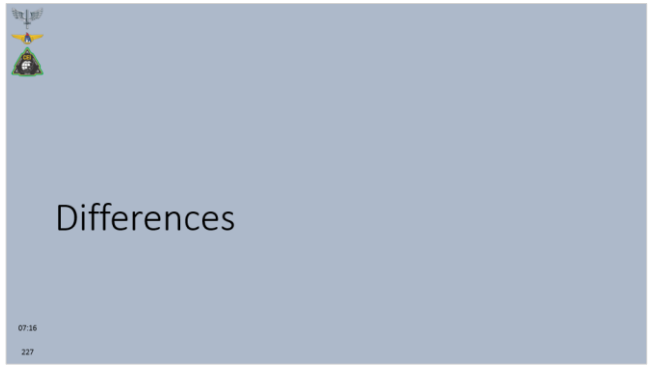
SysML & ARCADIA

https://polarsys.org/capella/arcadia_capella_sysml_tool.html

	SysML	Arcadia/Capella
Positioning	SysML is a standard and a general-purpose modeling language for modeling systems. SysML provides very rich and advanced expression means covering a very broad spectrum of applications, spanning from high-level architecture modeling to detailed design at the frontier of simulation.	Inspired by SysML concepts, the Arcadia/Capella solution focuses on the design of systems architectures. For the sake of an easier learning curve and because of the precise scope addressed by Arcadia/Capella, the expression means are sometimes reduced compared to SysML. The ultimate goal of Arcadia/Capella is to have systems engineers embrace the cultural change of MBSE rather than having modeling “experts” owning the model on behalf of systems engineers. Therefore, Arcadia/Capella are strongly driven by the current practices and concerns of system engineering practitioners.
Method	SysML is not associated to a particular method even though several engineering methods can be followed. As such, SysML only provides a vocabulary, but it does not specify when to use one concept or another, how to organize models, etc.	The Arcadia method enforces an approach structured on different engineering perspectives establishing a clear separation between system context and need modeling (operational need analysis and system need analysis) and solution modeling (logical and physical architectures), in accordance with the IEEE 1220 standard and covering parts of ISO/IEC/IEEE 15288.

	SysML	Arcadia/Capella
Language	Technically, the SysML language itself is defined as an extension of the Unified Modeling Language (UML). Both UML and SysML are general-purpose languages targeting wide spectrums of engineering domains and are relying on software-originated engineering paradigms using types, inheritance, etc.	The Arcadia concepts are mostly similar to the UML/SysML standard (about 75%) and the NATO Architecture Framework (NAF) standard (5%). Interoperability with SysML tools is possible using ad-hoc imports/exports. Because of the focus on architectural design, some of the SysML concepts have been simplified or specialized in order to better match the concepts system engineering practitioners already use in their engineering documents and assets. This is the case of the concepts related to functional analysis for instance.
Diagrams	SysML includes diagrams inherited from UML2 and adds new diagrams: • 4 diagrams are the same as UML2 diagrams (Sequence, State Machine, Use Case and Package); • 3 diagrams are extensions of UML2 diagrams (Activity, Block definition and Internal Block); • 2 diagrams are new diagram types (Requirement and Parametric).	Arcadia method is supported by various kinds of diagrams largely inspired by UML and SysML: • Architecture diagrams; • Dataflows diagrams; • Functional chains diagrams; • Sequence diagrams; • Tree diagrams; • Mode and States diagrams; • Classes and Interfaces diagrams.


Similarities and equivalences
07:16
217

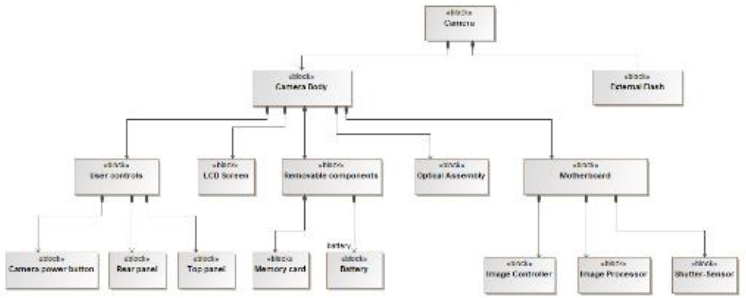

Differences
07:16
227

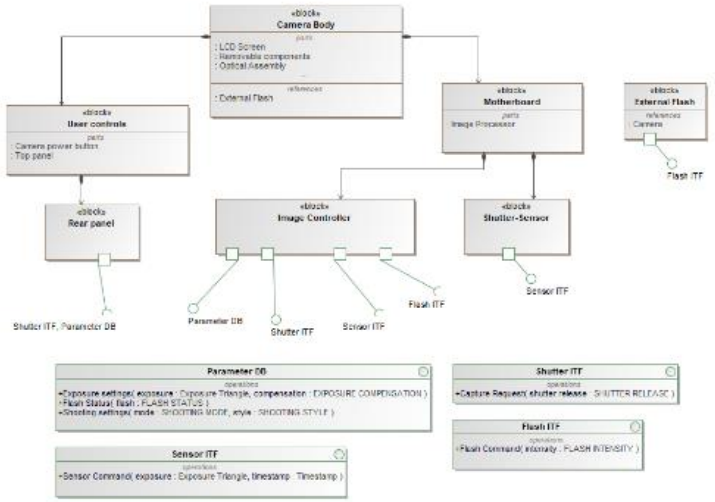
Similarities and equivalences

Block Definition Diagram

SysML

The Block Definition Diagram in SysML defines features of blocks and relationships between blocks such as associations, generalizations, and dependencies. It captures the definition of blocks in terms of properties and operations, and relationships such as a system hierarchy or a system classification tree.

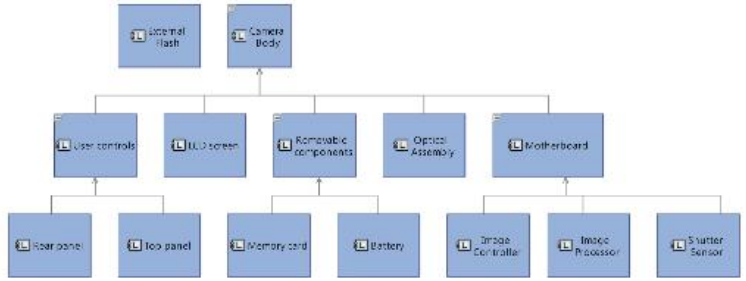




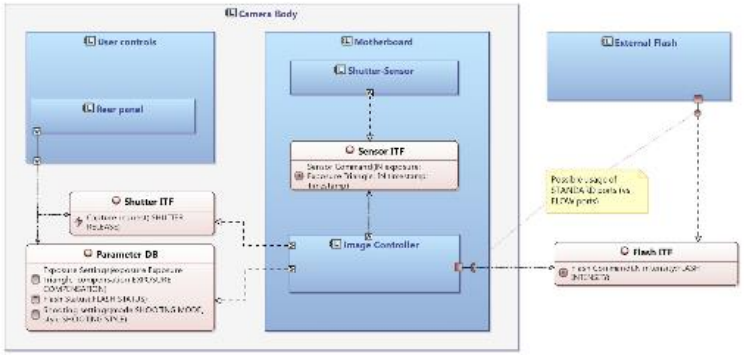
Arcadia/Capella

What is performed through Block Definition Diagrams in SysML is achieved in Arcadia through two kinds of diagrams:

- Component Breakdown Diagrams show the component hierarchy through a graphical tree.



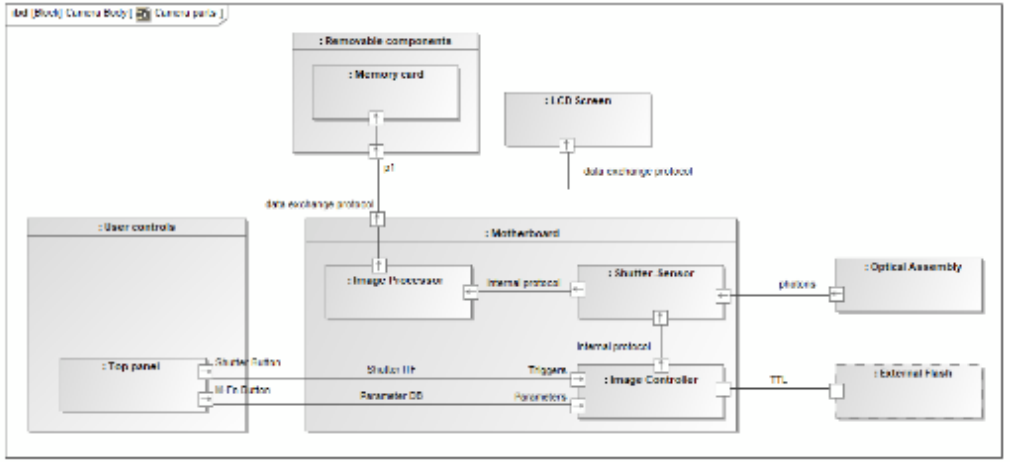
- Component Interface Diagrams show composition relationships between components through graphical containment and relationships between components and interfaces through ports. Component properties are not displayed graphically.



Internal Block Diagram

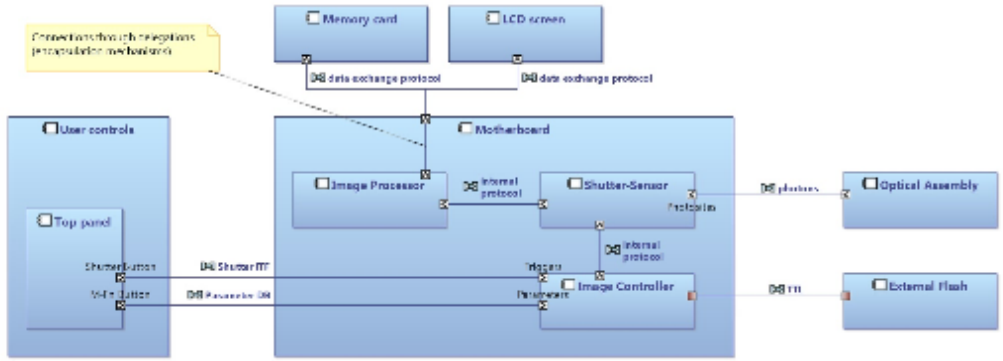
SysML

The Internal Block Diagram in SysML captures the internal structure of a block in terms of properties and connectors between properties.



Arcadia/Capella

Arcadia Architecture Diagrams describe the assembly of components in terms of internal breakdown and connections.



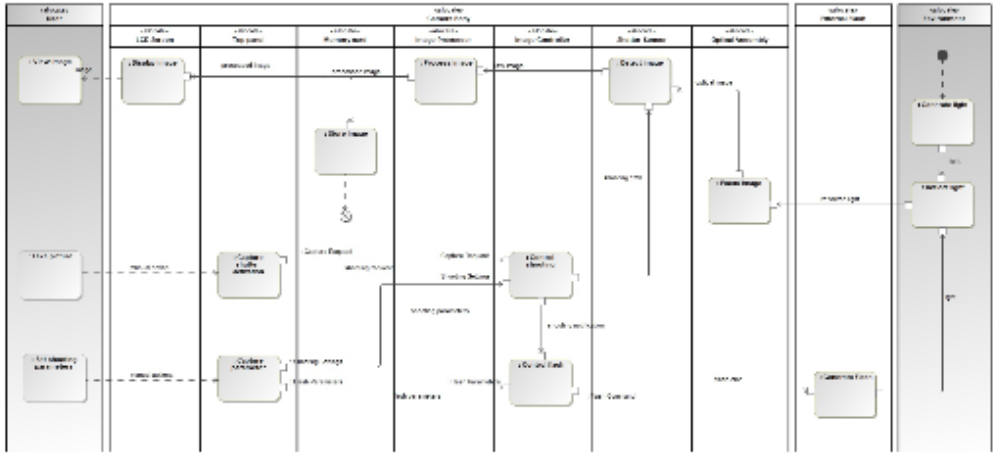
Note: See the dedicated section in Differences to understand how the concepts of parts, blocks and cardinalities are managed in Capella.

07:16

Activity Diagram

SysML

The Activity Diagram is a behavior diagram representing the flow of control and objects between activities.

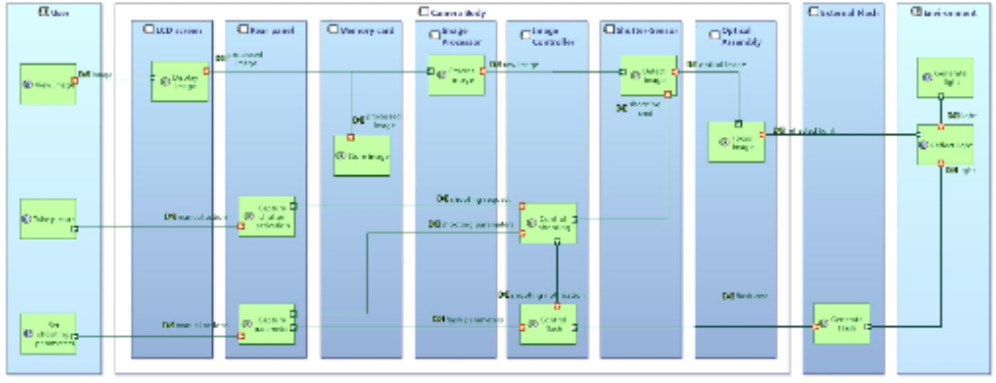


Arcadia/Capella

Capella functions naturally map to SysML activities/actions. However, while SysML Activity Diagrams are primarily intended to specify the control flows between activities, Capella Dataflows Diagrams only present the dependencies between functions with absolutely no semantics of control. The rationale for this choice is explained in this dedicated paper.

Allocating functions to components in Capella is similar to allocating actions to partitions representing blocks in SysML. Capella Architecture Diagrams ressemble to a mapping of SysML Activity Diagrams onto Internal Block Diagrams.

The following is a Capella Architecture Diagram voluntarily made similar to a SysML Activity Diagram where actions would be displayed in vertical partitions.



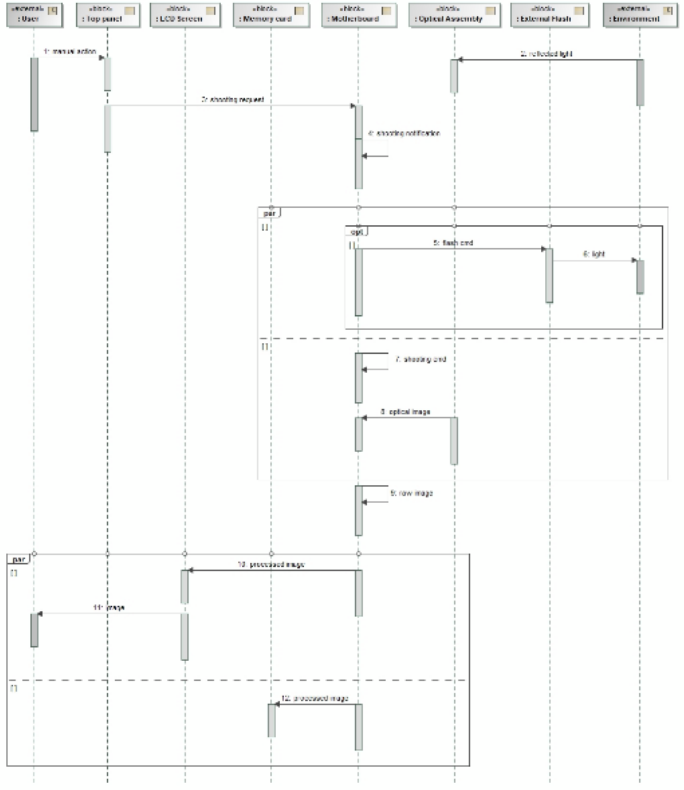
07:16

Sequence Diagram

SysML

Sequence Diagrams describe the interaction information with a focus on the time sequence.

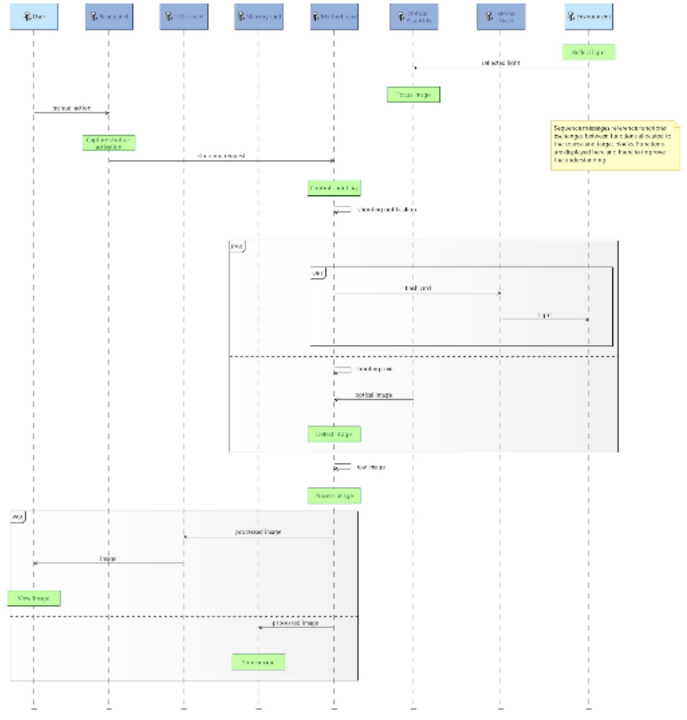
This diagram represents the sending and receiving of messages between the interacting entities called lifelines, where time is represented along the vertical axis. The sequence diagrams can represent highly complex interactions with special constructs to represent various types of control logic, reference interactions on other sequence diagrams, and decomposition of lifelines into their constituent parts.



Arcadia/Capella

The Capella underlying constructs of Sequence Diagrams are strictly mapped onto SysML ones. The differences reside in the variety of elements that can be referenced in a consistent manner by lifelines and sequence messages.

In the following figure, lifelines represent components (blocks in SysML) and sequence messages represent dependencies existing between the functions (actions/activities in SysML) respectively allocated to source and target components.



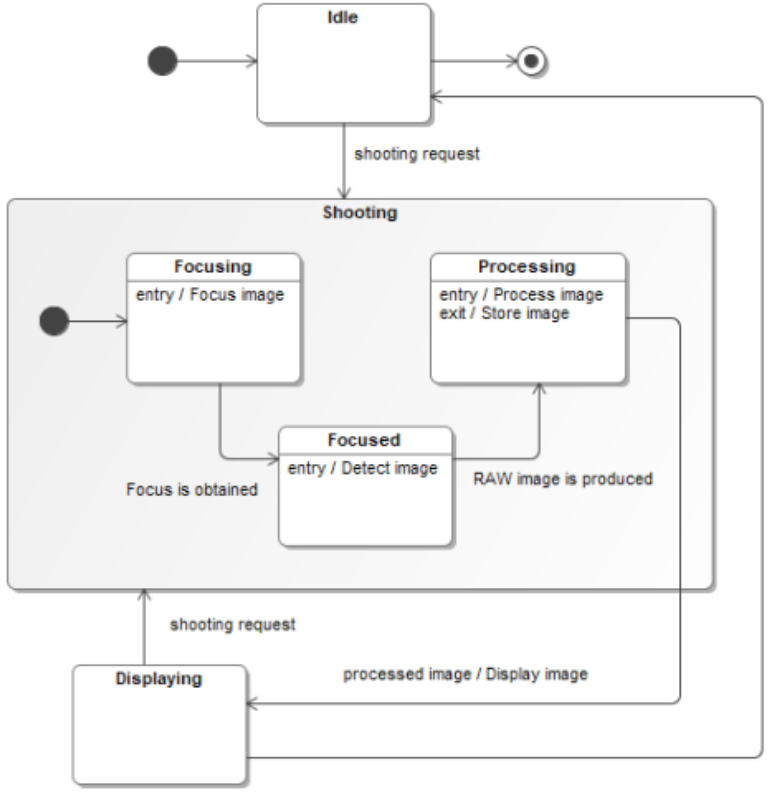
In the next Sequence Diagram lifelines represent functions (actions/activities in SysML) and sequence messages represent dataflows between these functions

07:16

State Machine Diagram

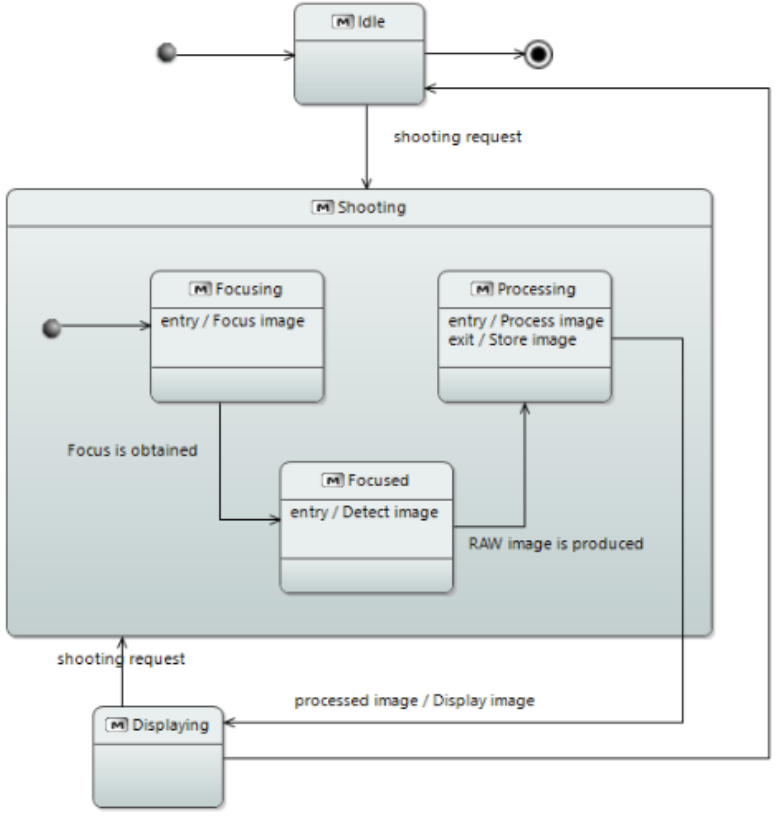
SysML

The State Machine Diagram is a behavior diagram describing the state transitions and actions that a system or its parts perform in response to events. It is used for representing behavior as the state history of an object in terms of its transitions and states.



Arcadia/Capella

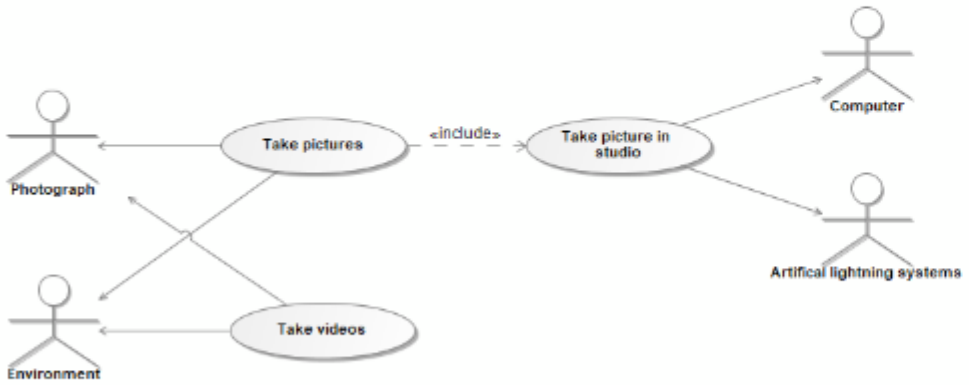
The Modes and States Machines in Capella are extremely close to SysML. The constructs are the same, but Capella adds a bit of semantics (difference between modes and states, articulation with functional analysis and interfaces).



Use Case Diagram

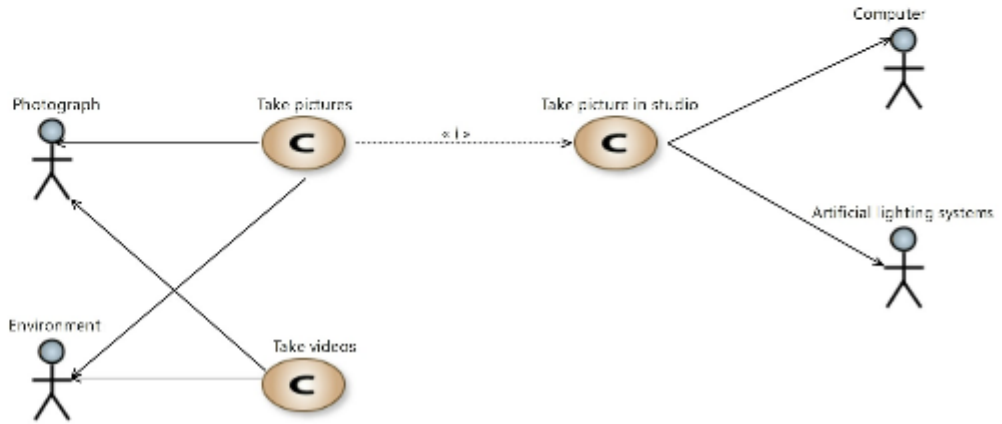
SysML

The Use Case Diagram is a method for describing the usages of a system. It represents a high-level description of functionalities that are achieved through interaction among a system (subject) and its actors (environment) to achieve a goal.



Arcadia/Capella

Capabilities in Capella are equivalent to SysML use cases and Capabilities Diagrams largely ressemble SysML Use Case diagrams. Capabilities are intensively used in Capella to organize the functional analysis: the involvement of stakeholders in a given capability is enriched by a specification of the stakeholder functions performed in the context of this capability.

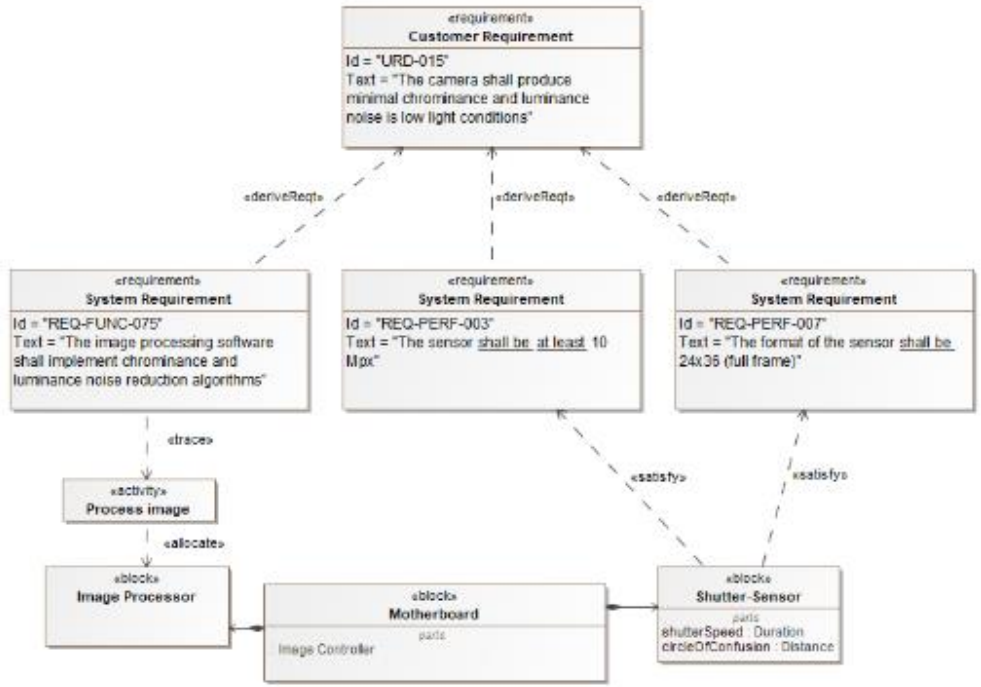


07:16

Requirement Diagram

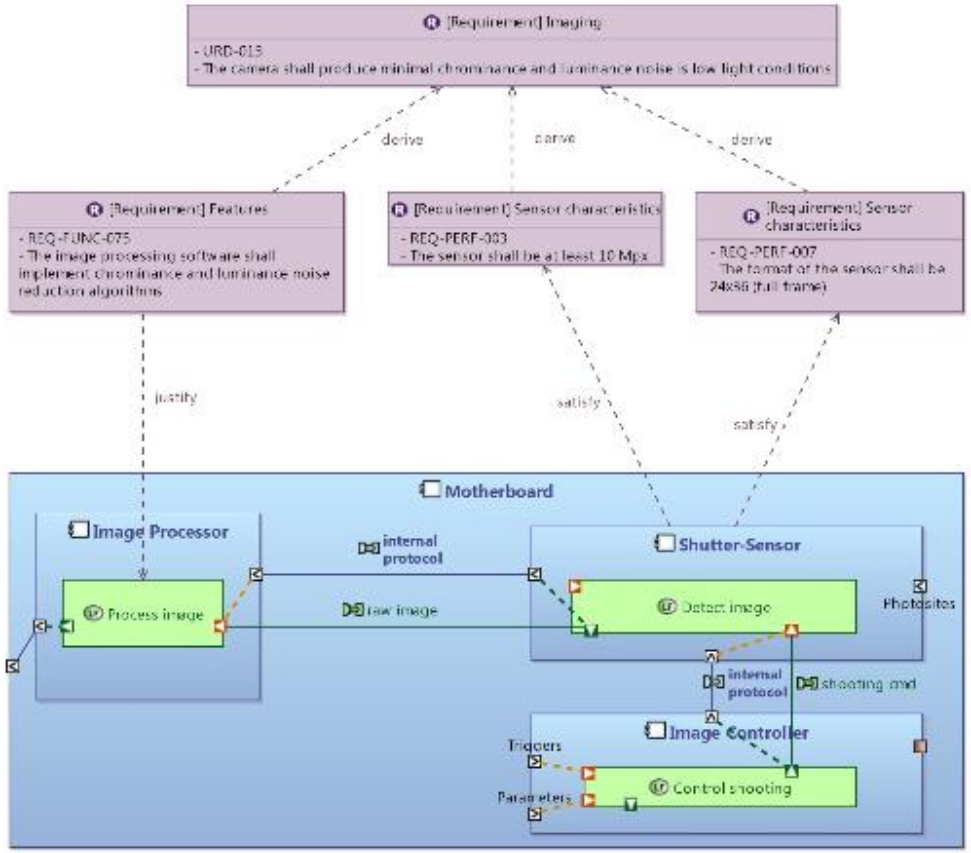
SysML

SysML includes a graphical construct to represent text based requirements and relate them to other model elements. The Requirements Diagram captures requirements hierarchies and requirements derivation, and the satisfy and verify relationships allow a modeler to relate a requirement to a model element that satisfies or verifies the requirements. The requirement diagram provides a bridge between the typical requirements management tools and the system models.



Arcadia/Capella

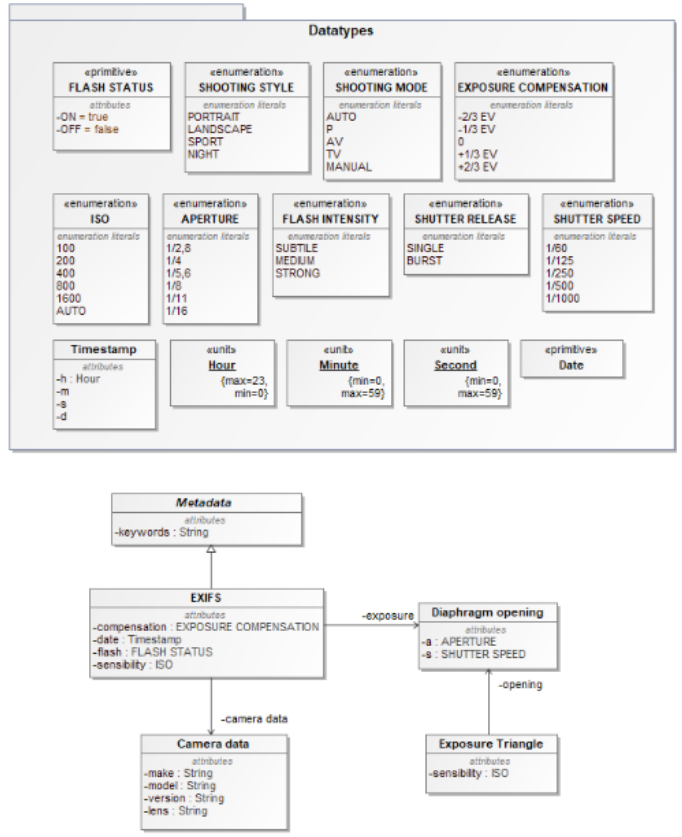
Textual requirements can be displayed in any Capella diagram. Relationships between requirements and model elements as well as between requirements can be shown. There is however no dedicated requirement diagram in Capella.



Class Diagrams

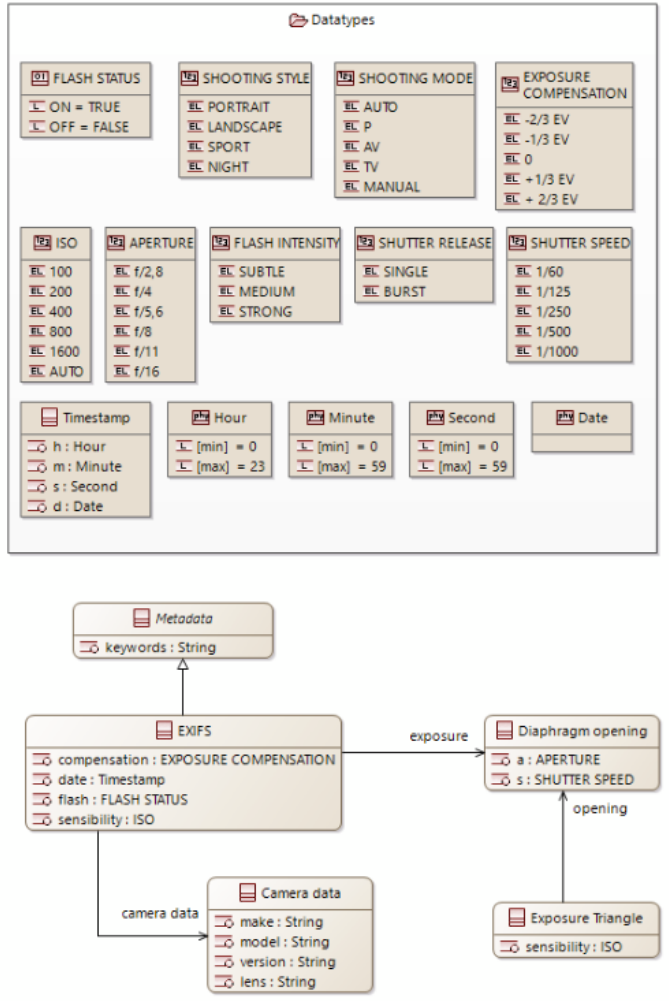
SysML

The UML Class Diagram does not belong to the official subset of UML diagrams available in SysML (it is replaced by the Block Definition Diagram which is based on the UML Class Diagram, with restrictions and extensions). However, it is presented here, as it is frequently used in addition to SysML diagrams.



Arcadia/Capella

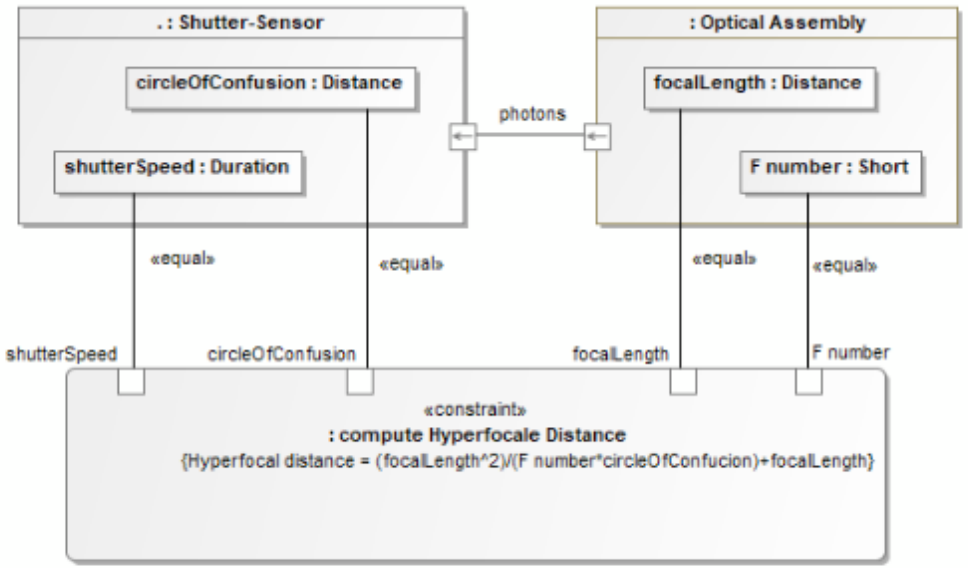
Capella class diagrams are fully aligned on UML class diagrams. Capella however adds a certain amount of construction rules (prohibiting dependency cycles for example, or enforcing certain visualization choices according to the properties of elements).



Parametric Diagrams

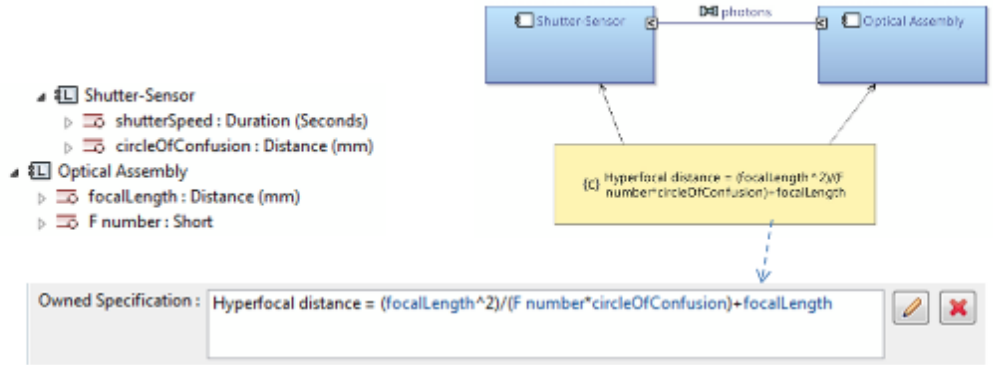
SysML

Parametric Diagrams are a restricted form of Internal Block Diagram that shows only the use of constraint blocks along with the properties they constrain within a context. Parametric Diagrams are used to support engineering analysis, such as performance or mass properties analysis.



Arcadia/Capella

While most of the underlying concepts are present in Capella (constraints, opaque expressions with assisted editing, parseable expressions, properties on elements, physical dimensions, etc.), no diagram is dedicated to their graphical display.



Note: Currently, Capella users typically use dedicated viewpoints (language and analyses extensions + graphical layers on top of existing diagrams) to evaluate their architecture against non-functional constraints. They rarely use the architecture models for simulation purposes. Should the end-user request them, parametric diagrams could be an easy addition to Capella.

07:16

Differences

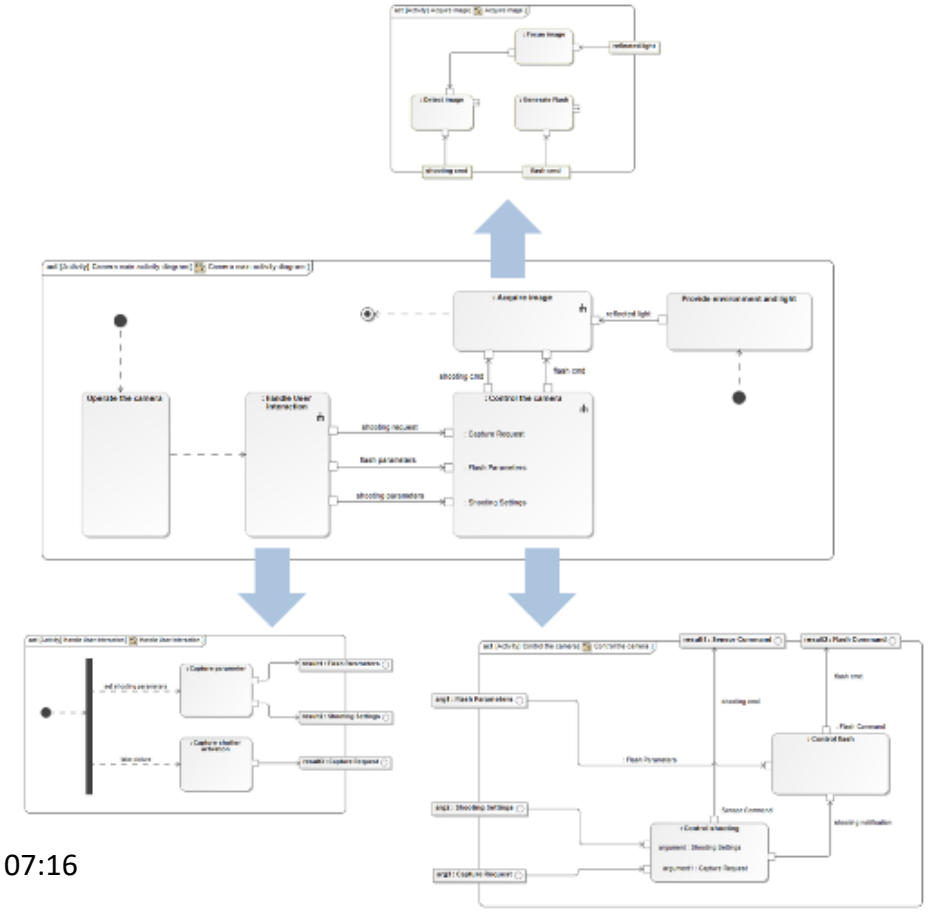
Functional Analysis

- Functional analysis is a classical technique broadly used by systems engineers. Arcadia and Capella provide methodological guidance and engineering helpers to support this technique that has been mostly left out of SysML V1.
- The mapping of Capella functions to SysML activity is the most natural one in terms of semantics. Capella functions are verbs specifying the actions expected from the component they are allocated to. This section describes the structural differences between SysML activities/actions and Capella functions.

07:16

SysML

The articulation between several Activity Diagrams relies on two major concepts: activities are described by different kinds of actions including some that can reference other activities, and the parameters of a given activity are connected (delegated) to the output or input pins of the actions describing it. This strong encapsulation mechanism favors the reuse of activity definitions in multiple contexts but imposes constraints on what can be represented in one single diagram and makes bottom-up workflows more difficult to implement.



07:16

Arcadia/Capella

There are three major differences between SysML Activity Diagrams and Capella dataflows:

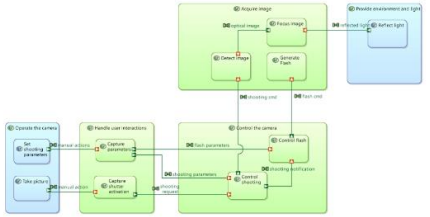
1. There is no control flow in Capella Dataflow Diagrams, meaning that there is no semantics of execution and there are no control nodes such as Join, Fork, etc.. The detailed rationale for the absence of control flows in Capella dataflow is explained in this dedicated paper.
2. The relationship between a function and its subfunctions is a direct containment
3. There is no delegation mechanism between functions at each level of decomposition in Capella. The rationale is detailed hereunder.

In a hierarchy of Capella functions, non-leaf functions are only "grouping" elements. This means:

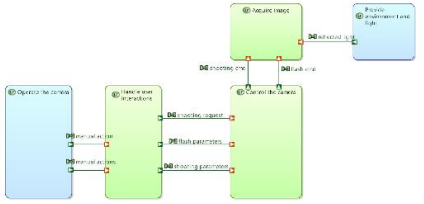
- Non-leaf functions are not supposed to have ports nor functional dependencies
- Non leaf-functions are not supposed to be allocated to components
- A leaf function can be connected freely to any other leaf function
- When a non-leaf function has ports, the design is considered non-finalized. The remaining ports are supposed to be moved towards a leaf function
- Low-levels dependencies between leaf functions are automatically displayed when intermediate/parent/non-leaf functions are displayed on a diagram.

The rationales for this modeling choice are multiple:

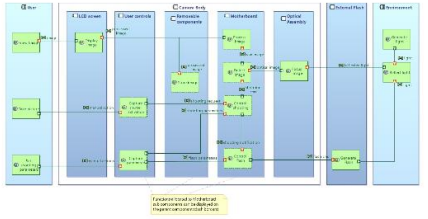
- This helps manage the complexity of functional trees by relieving engineers from the tedious task of maintaining the consistency of dependencies between several levels of decomposition
- This allows the immediate production of simplified views of the functional analysis
- This enables the easy combination between top-down and bottom-up workflows

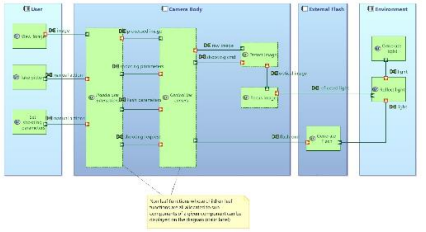


Capella leverages this language choice to provide several kinds of simplified views of the system architecture. The following diagram for example is automatically computed. Ports are displayed on non-leaf functions but still belong to children functions.



Similarly, graphical simplifications of Architecture Diagrams can be computed by automatically performing grouping at component and function levels.

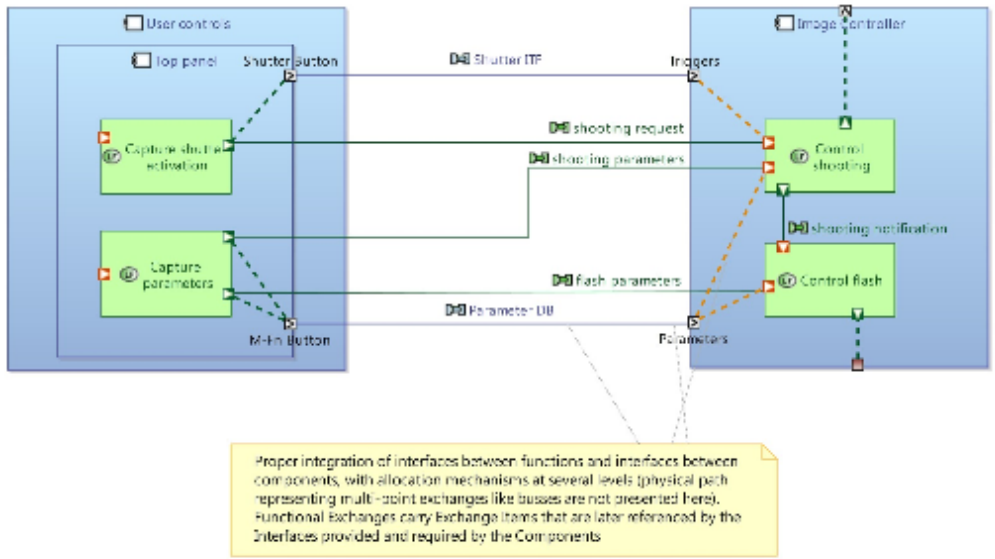




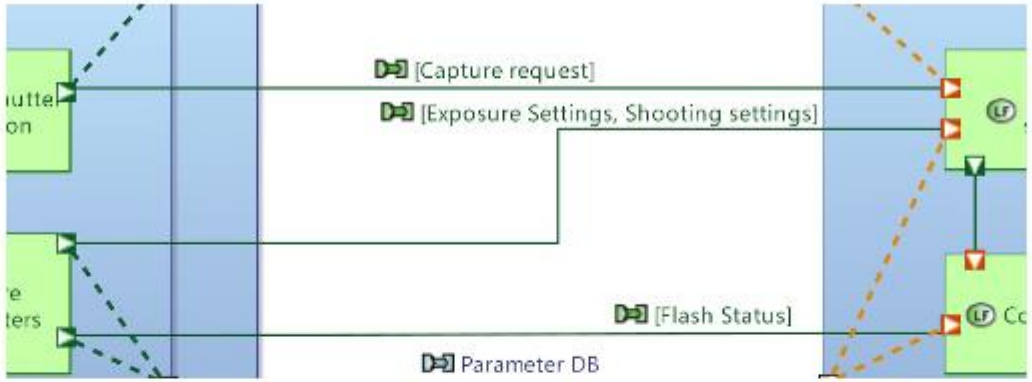
Integration Functions / Components / Interfaces

The biggest objective of the Arcadia method is to secure the architectural design activity through identification and justification of the interfaces. This is achieved by providing a global approach to conduct functional, structural, and interface modeling in parallel:

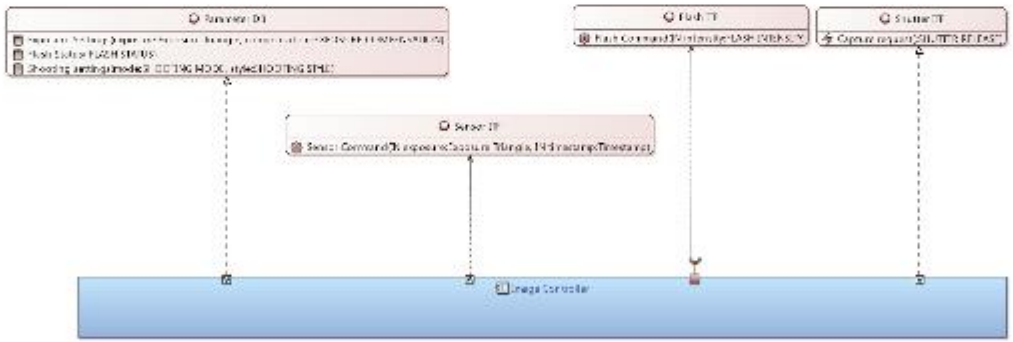
- Identification of the functional expectations of the subsystems (allocation of functions to components)
- Identification of the functional dependencies between the subsystems (specification of the exchanges between functions ideally with a structural description of the exchanged items)
- Allocation of functional dependencies to assembly relationships between subsystems (allocation of functional ports to component ports, allocation of functional exchanges to component exchanges, etc.)
- Specification of the interfaces provided and required through component ports (with a possible automated deduction based on all the above-mentioned specification)



Instead of showing the actual element name, the label of functional dependencies can show references towards the exchanged items.



The following diagram details the content of the interfaces between the components, deduced from the functional analysis and multiple allocations.



This integration of the functions / components / interfaces triptych is not straightforward to implement and enforce in SysML v1. This global approach promoted in Capella also comes with a set of assistance tooling enforcing the model correctness regarding this integration and providing automation means.

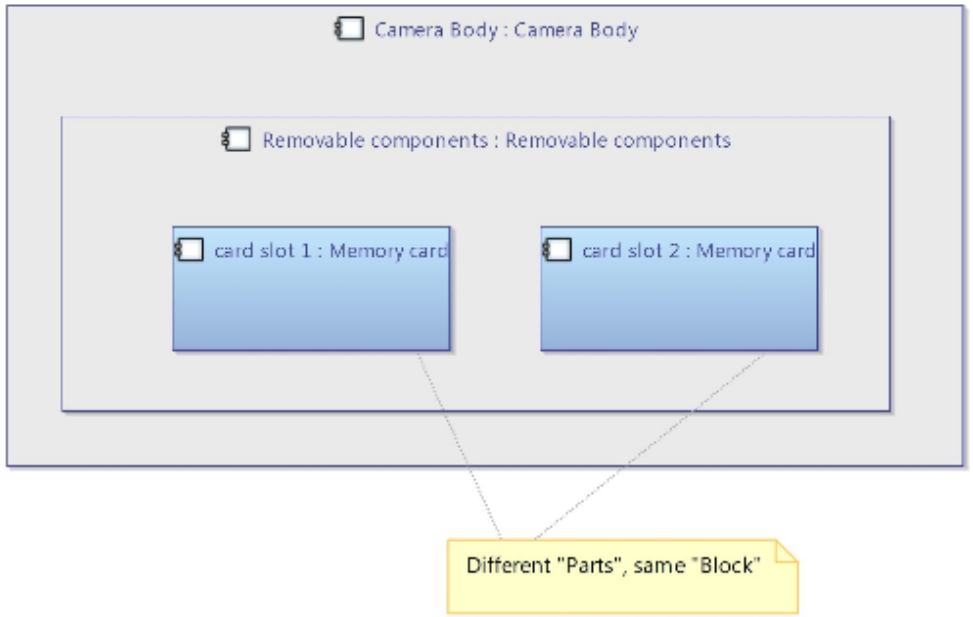
Note: The topic of a better integration between structure and behaviour is currently being investigated within the SysML v2 SST submission team.

07:16

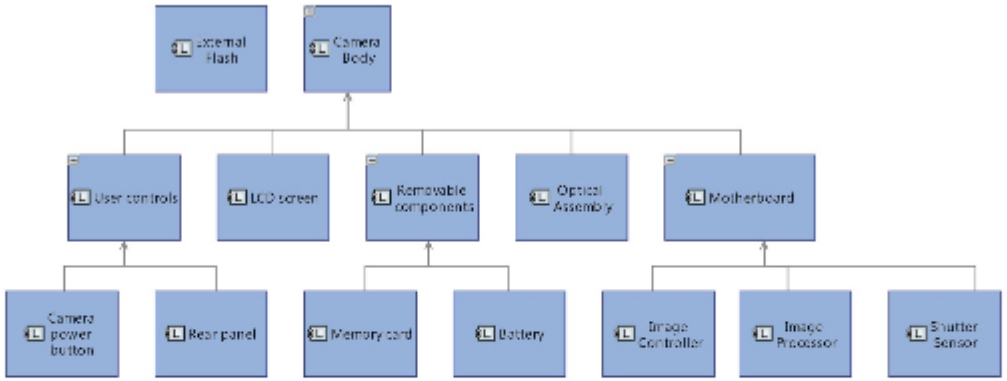
Management of "instances", or "definitions and usages"

The SysML Internal Block Diagram is dedicated to model the internal structure of a block. SysML relies on a generic block/part paradigm: in an Internal Block Diagram, a block can be decomposed into parts (usages) which are themselves typed by other blocks (definitions). A bicycle “block” has two parts “front wheel” and “rear wheel” which are both typed by the block “wheel”. The “wheel” definition is captured in one dedicated block and the same definition can be reused many times in the system through the part concept.

It is possible in Capella to use the same block/part paradigm than in SysML. The following diagrams show how the memory card compartment of the camera can have two slots. There is only one “Memory Card” component, but it is referenced twice. The component breakdown diagram shows the unicity of the “Memory Card” component.



07:16



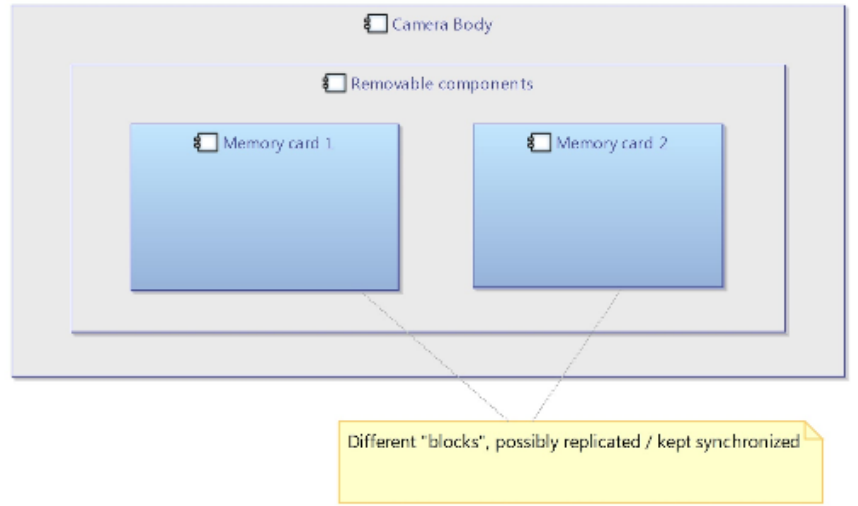
However, Capella is configured by default for an instance-driven modeling. Return of experience showed that systems engineers are not necessarily comfortable with the workflow of creating definition elements first (“blocks” or “components”) and then referencing them from specific usage elements (“parts”).

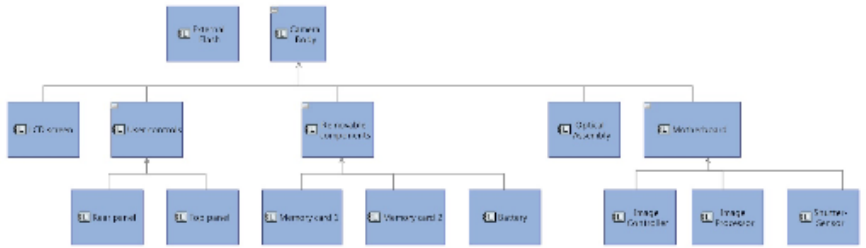
In addition, architectural design in Capella also consists in performing non functional analyses where it is critical to be able to distinguish the different occurrences of each element and to be able to give them different properties or values. For example, safety analyses typically require to distinguish between the execution of an “identical” function in two distinct components.

However, Capella is configured by default for an instance-driven modeling. Return of experience showed that systems engineers are not necessarily comfortable with the workflow of creating definition elements first (“blocks” or “components”) and then referencing them from specific usage elements (“parts”).

In addition, architectural design in Capella also consists in performing non functional analyses where it is critical to be able to distinguish the different occurrences of each element and to be able to give them different properties or values. For example, safety analyses typically require to distinguish between the execution of an “identical” function in two distinct components.

This means components and functions in Capella are by default considered as instances or usages.



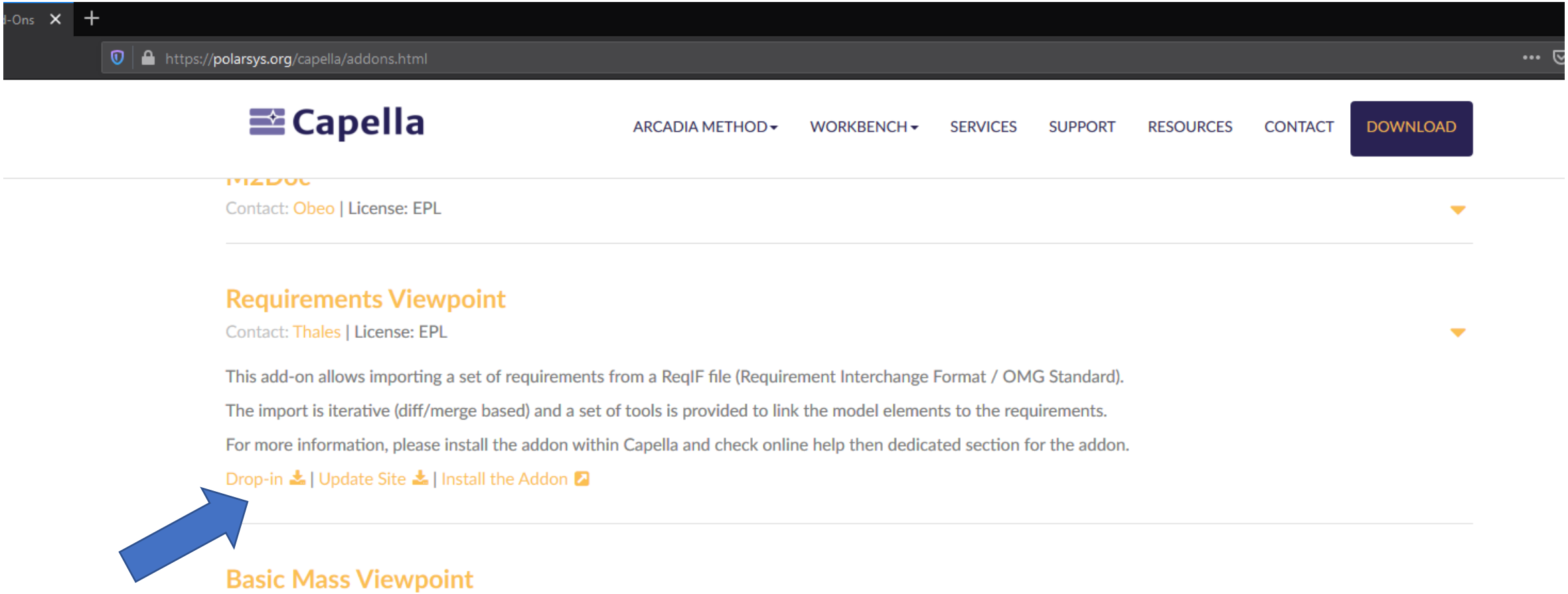


To support this approach, Capella provides automated mechanisms allowing the replication and synchronization of model elements (REC and RPL, for Records and Replica).

Note: This topic is known as “usage-based modeling” in the SysML v2 submission SST team, the goal being to have a language able to efficiently support multiple creation workflows. The outcome of this effort might at some point be taken into account in Capella.

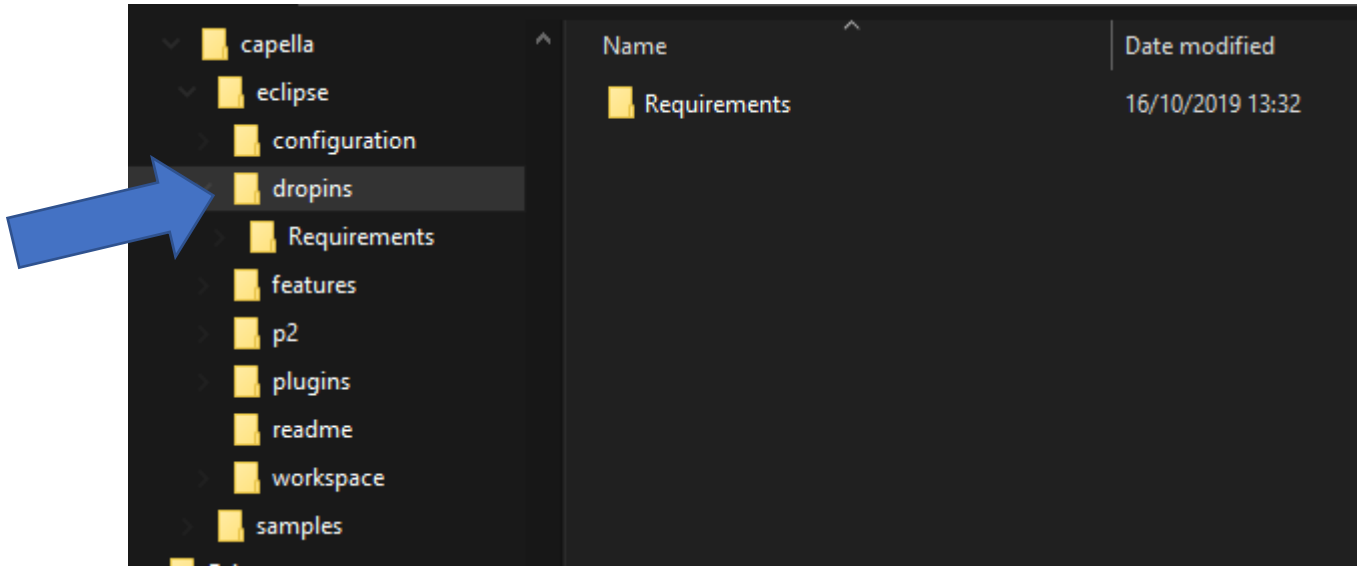
REQUIREMENTS IN CAPELLA

[IF NOT INSTALLED] ADD THE REQ ADDON



07:16

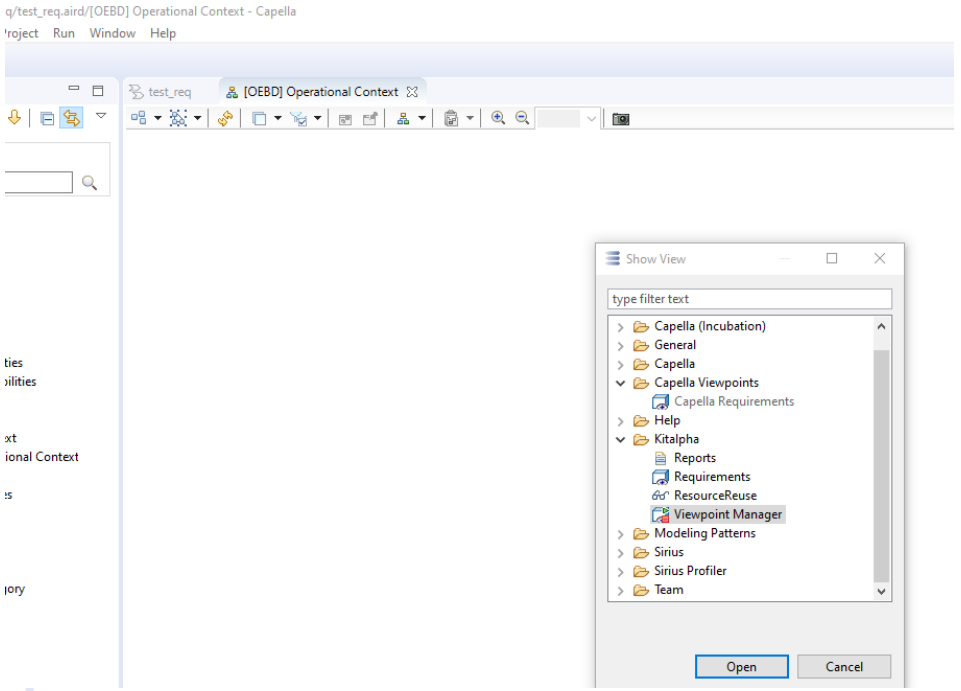
[IF NOT INSTALLED] UNZIP IN DROPIN FOLDER

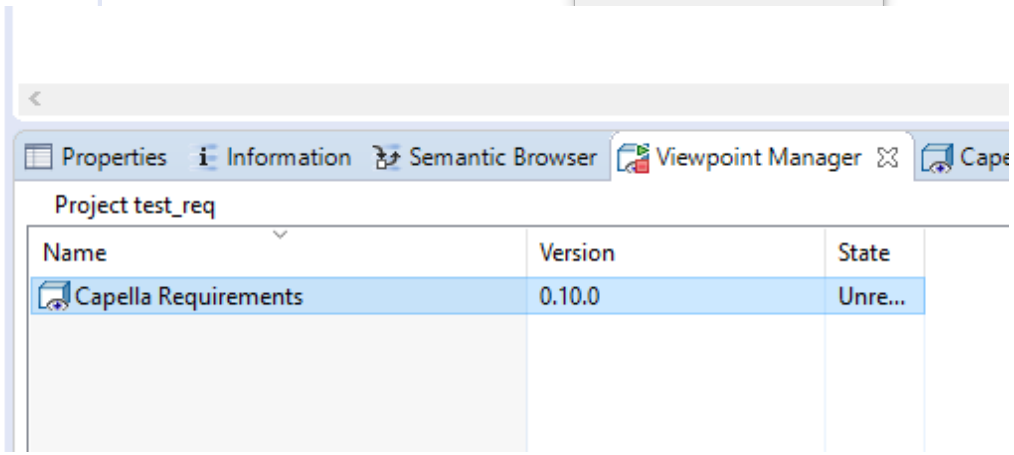


07:16

[IF NOT INSTALLED] Last Steps

- Start Capella
- Open the **Viewpoint Manager** view using **Window** menu then **Show View** and **Other...**
- Select **Viewpoint Manager** in **Kitalpha** directory and press **OK**
- The **Viewpoint Manager** view is displayed
- The viewpoints available in the platform are listed in this view.
- If using Capella version < 1.0.x
  - Right-click on the name of a viewpoint and select **Start** in order to start the viewpoint
- If using Capella version > 1.0.x
  - Select any model element (diagram element, element in the project explorer) related to your project
  - Right-click on the name of a viewpoint and select **Reference** in order to start the viewpoint



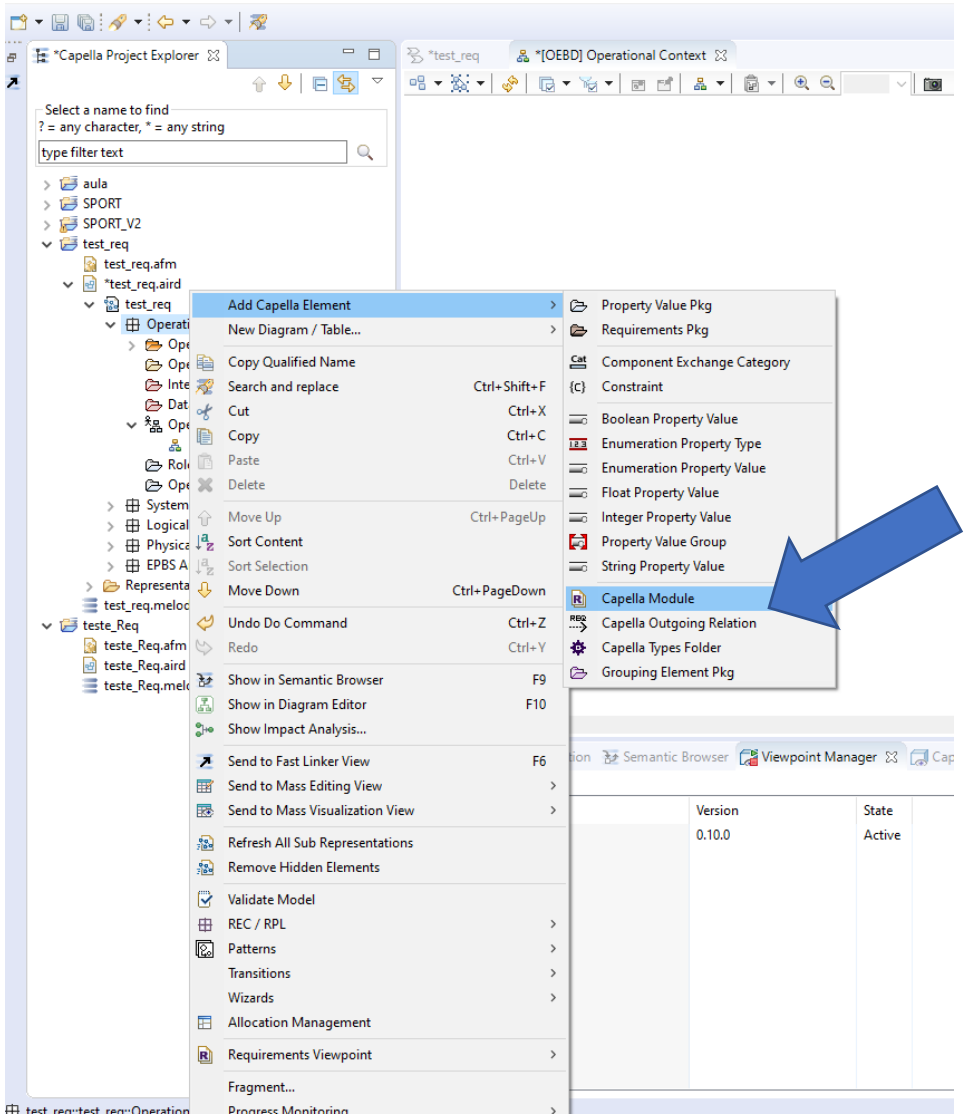


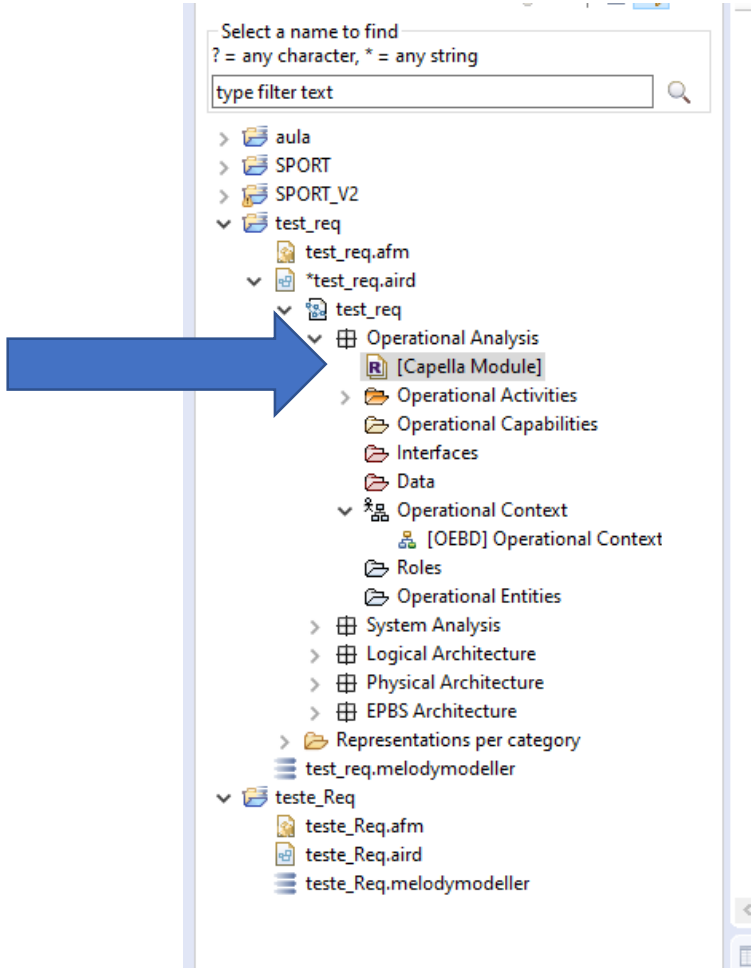
07:16

CAN BE USED IN MULTIPLE LAYER

- Operational Analysis Requirements
- System Analysis Requirements
- Logical Architecture Requirements
- Physical Architecture Requirements
- EPBS Architecture Requirements

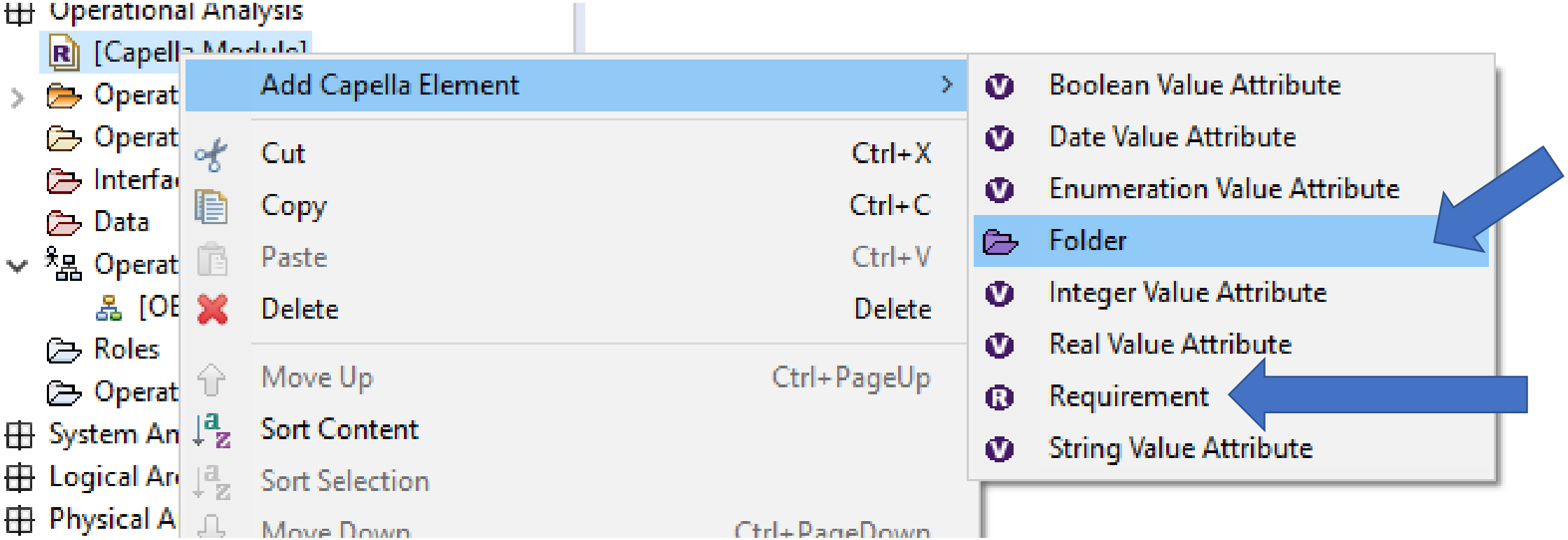
ADD A CAPELLA MODULE IN THE LAYER





07:16

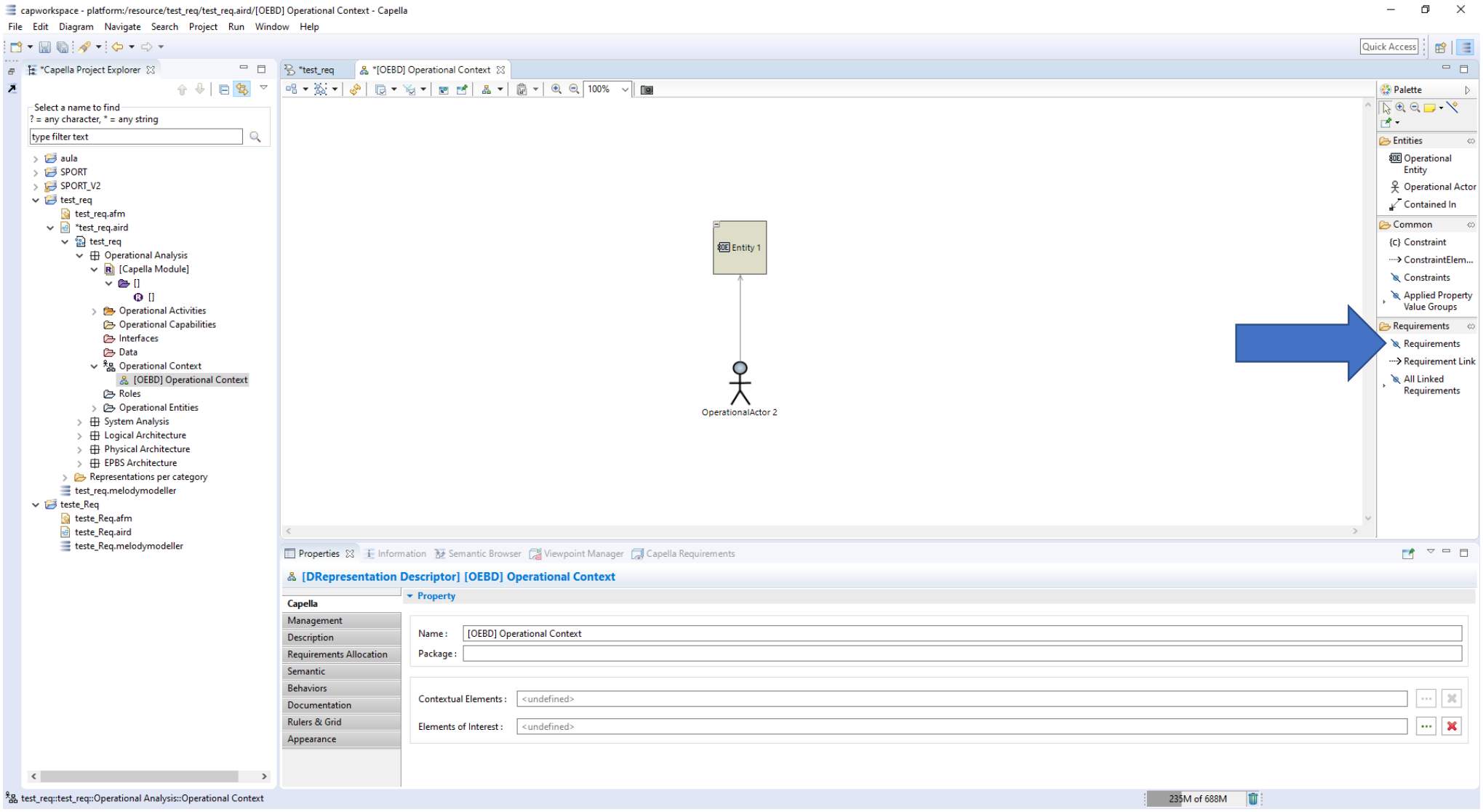
CREATE A REQUIREMENT FOLDER & REQUIREMENT



The only way to create requirements is through the Project Explorer.
[good side] Capella connects to Doors ($$$) to import requirements.

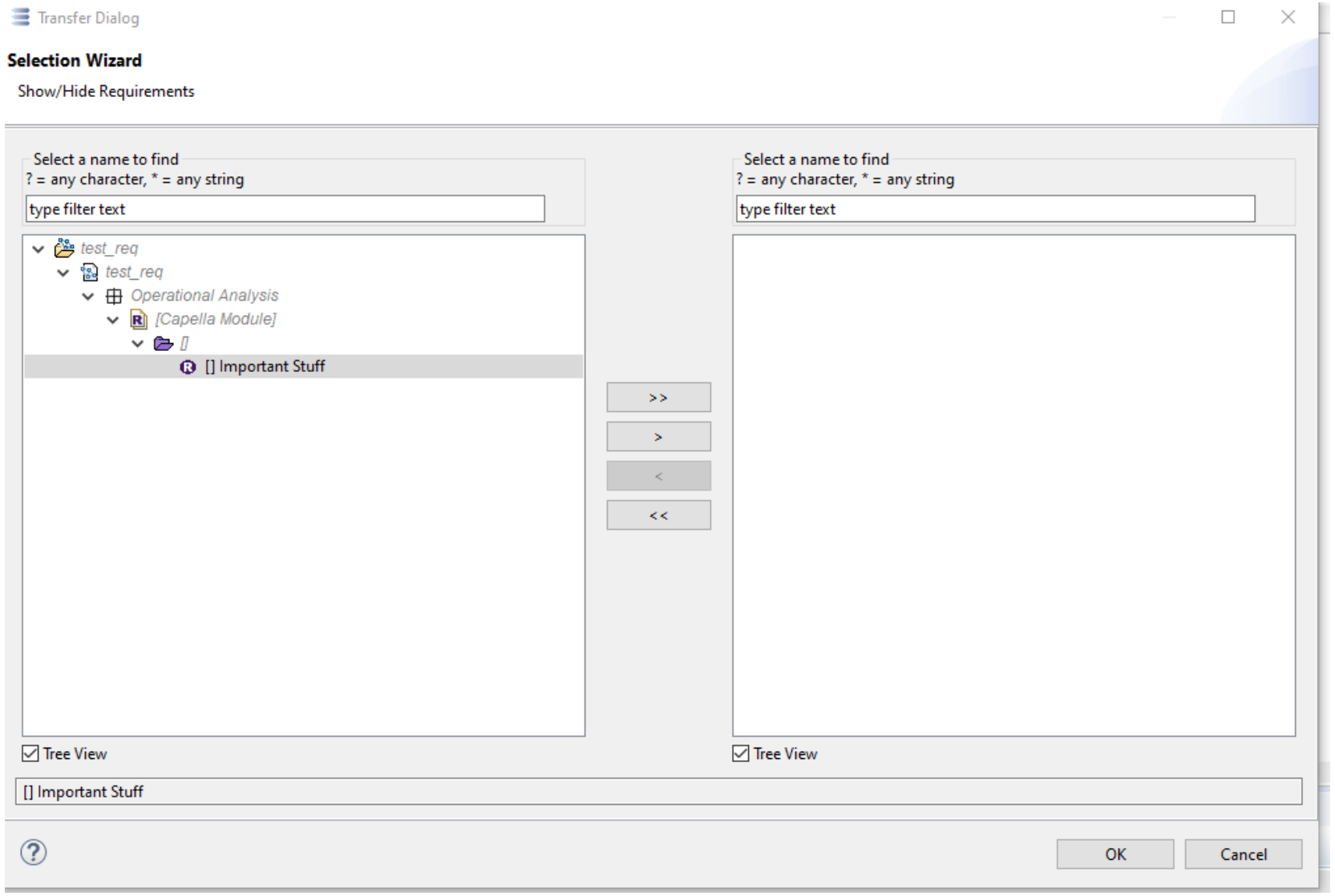
07:16

REQUIREMENTS CAN BE USED IN ANY VIEW

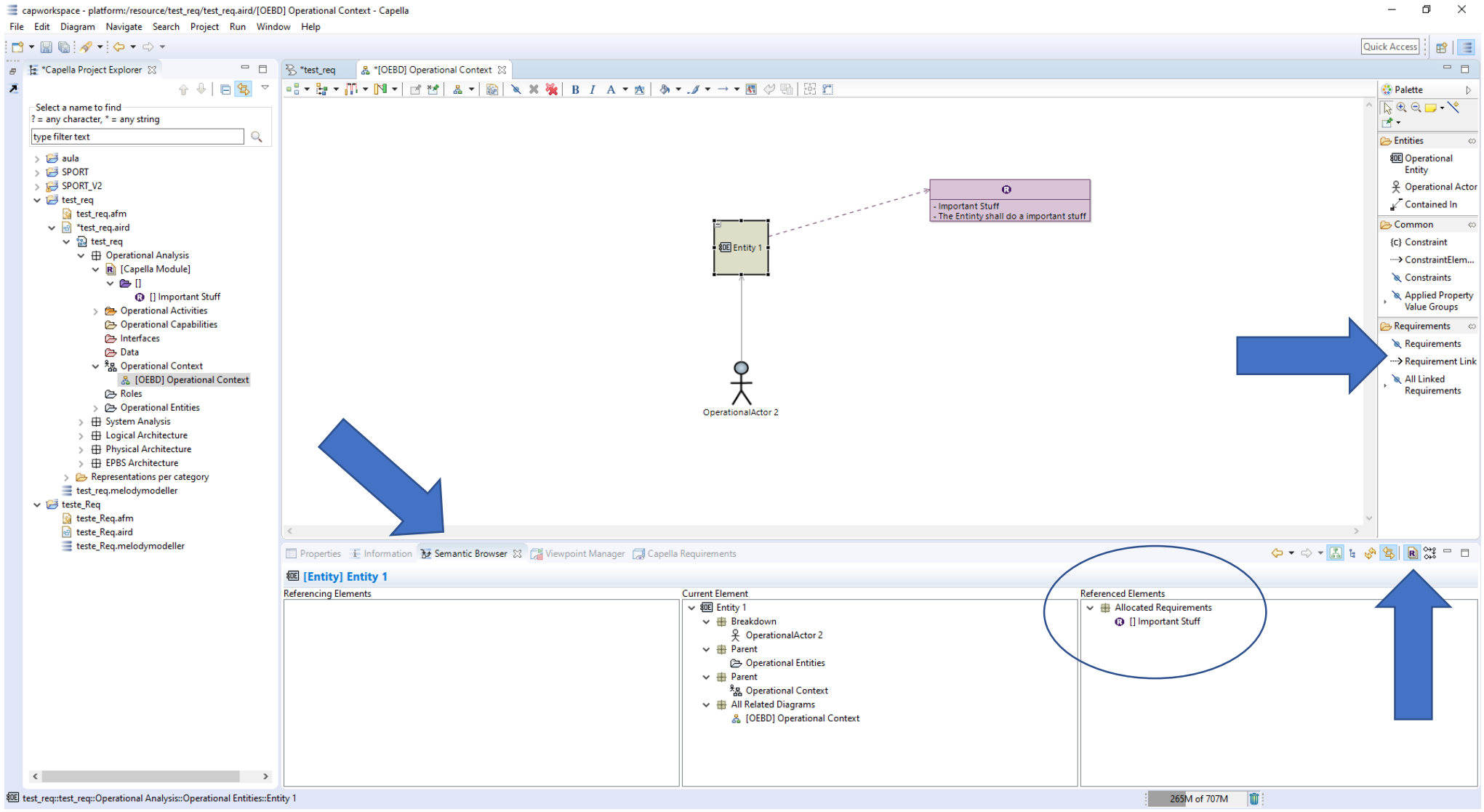


07:16

SELECT THE REQUIREMENTS THAT WANT TO USE IN THE VIEW.

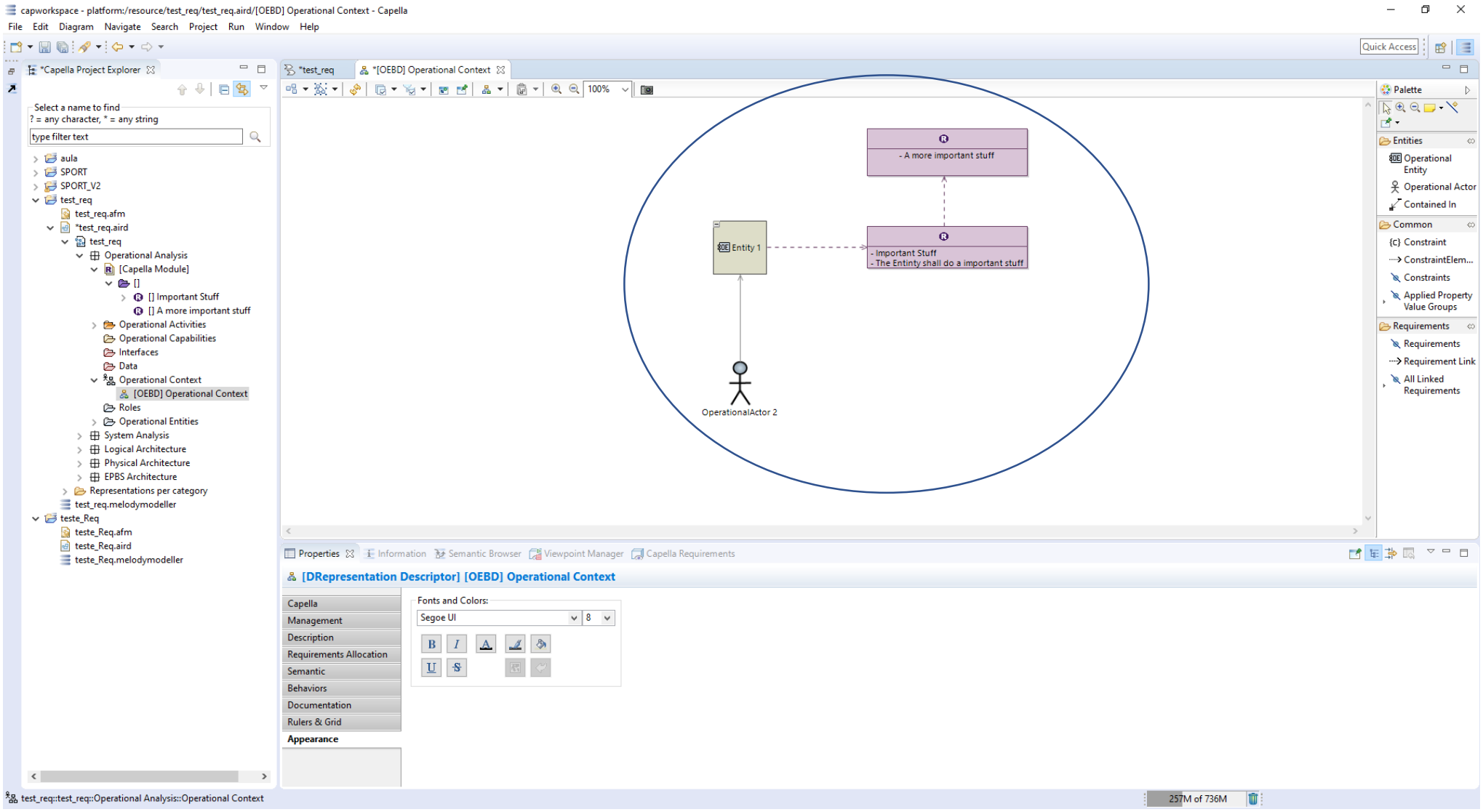


ADD A LINK / CHECK RELATIONS



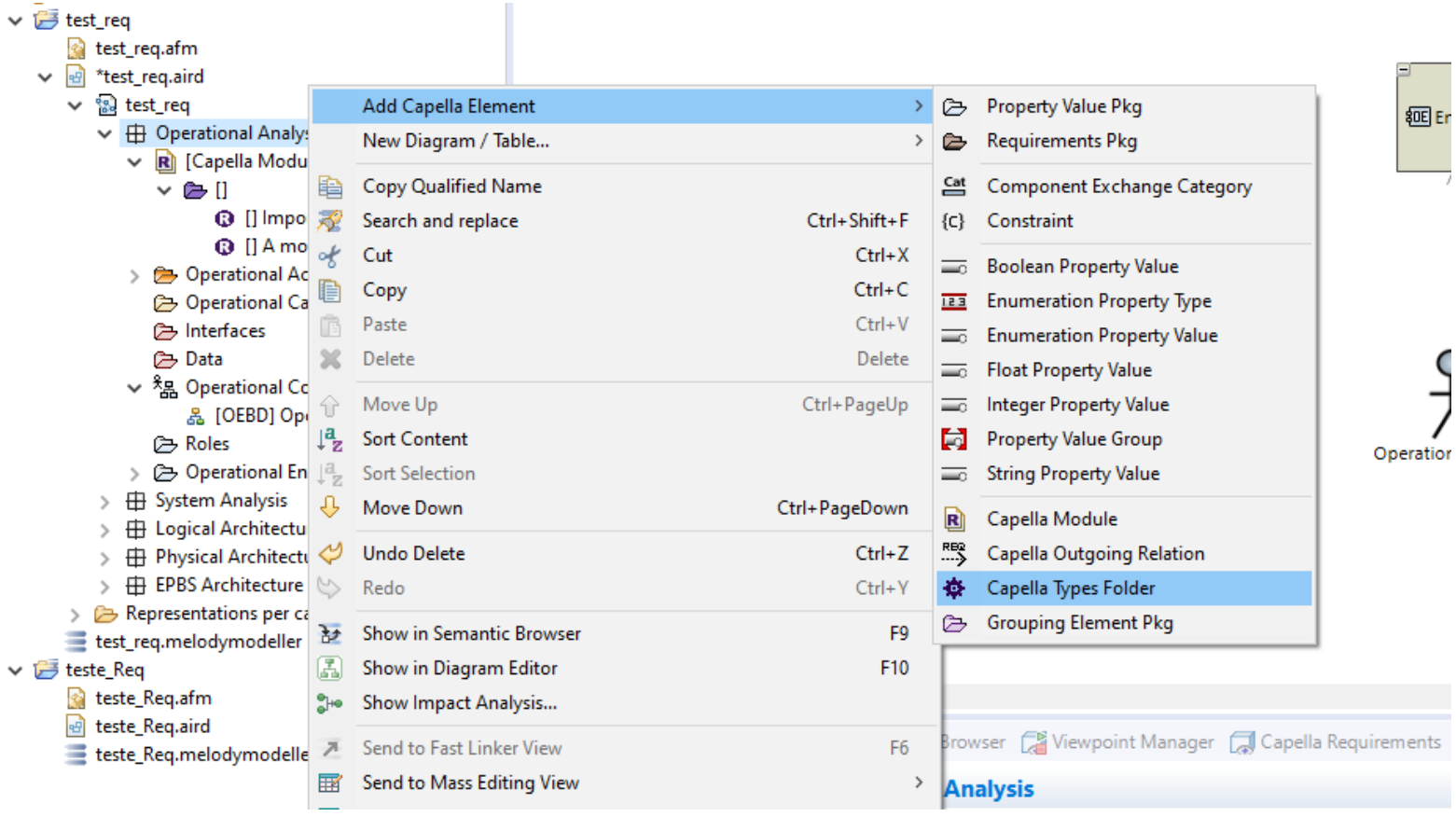
07:16

REQUIREMENT TREES



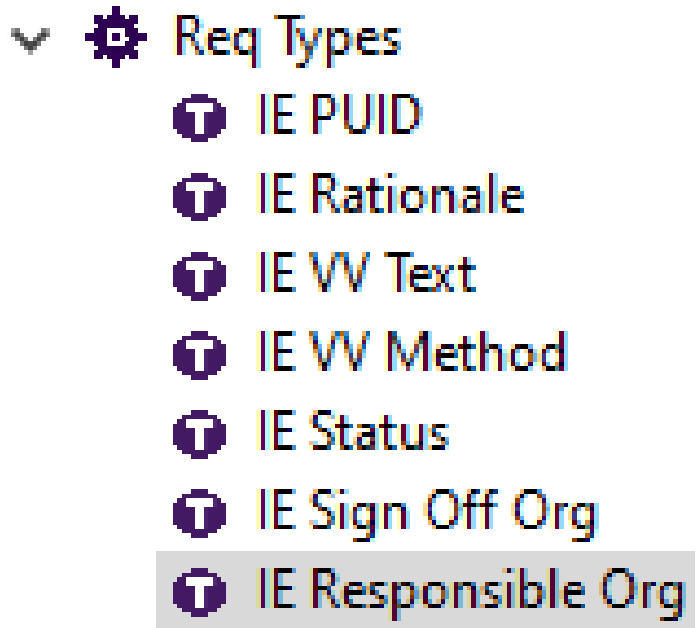
ADD REQUIREMENT METADATA

- It is required to create a new Type
- Create a Capella Types Folder → Rename Req Types

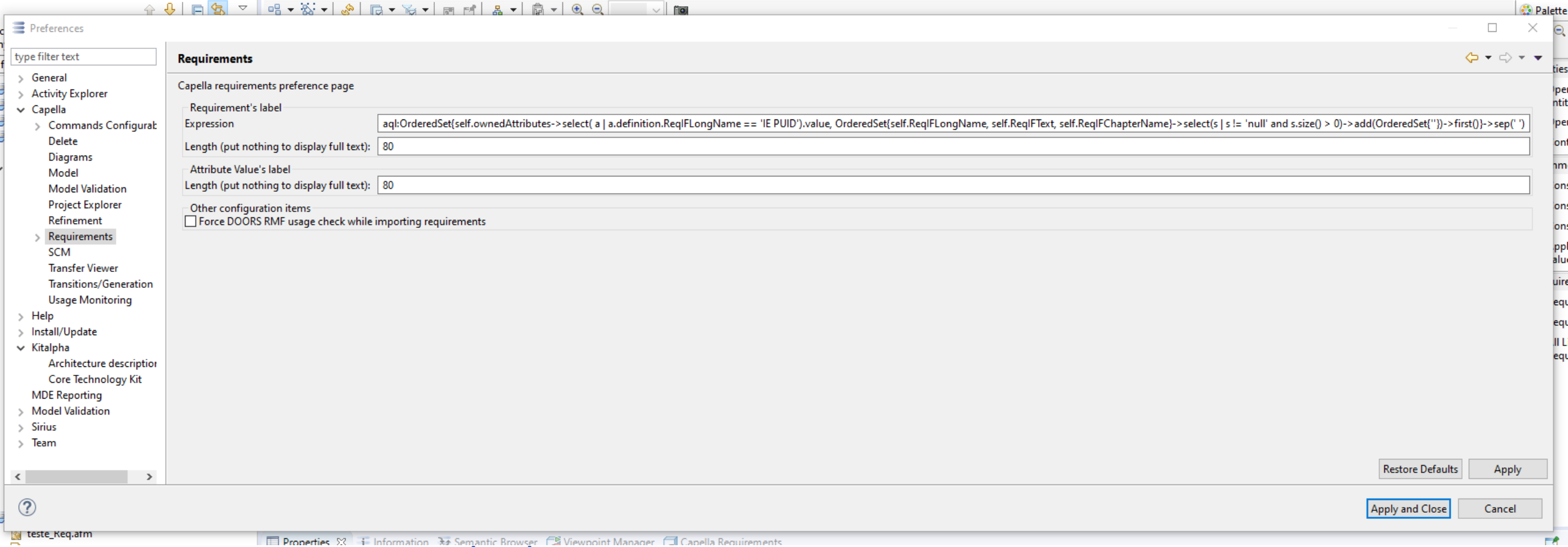


Requirement Data Type Definitions

- IE PUID (Requirement ID – name comes from DOORS)
- IE Rationale
- IE Verification Text
- IE Verification Method Expected
- IE Requirement Status
- IE Sign off Org
- IE Responsible Org



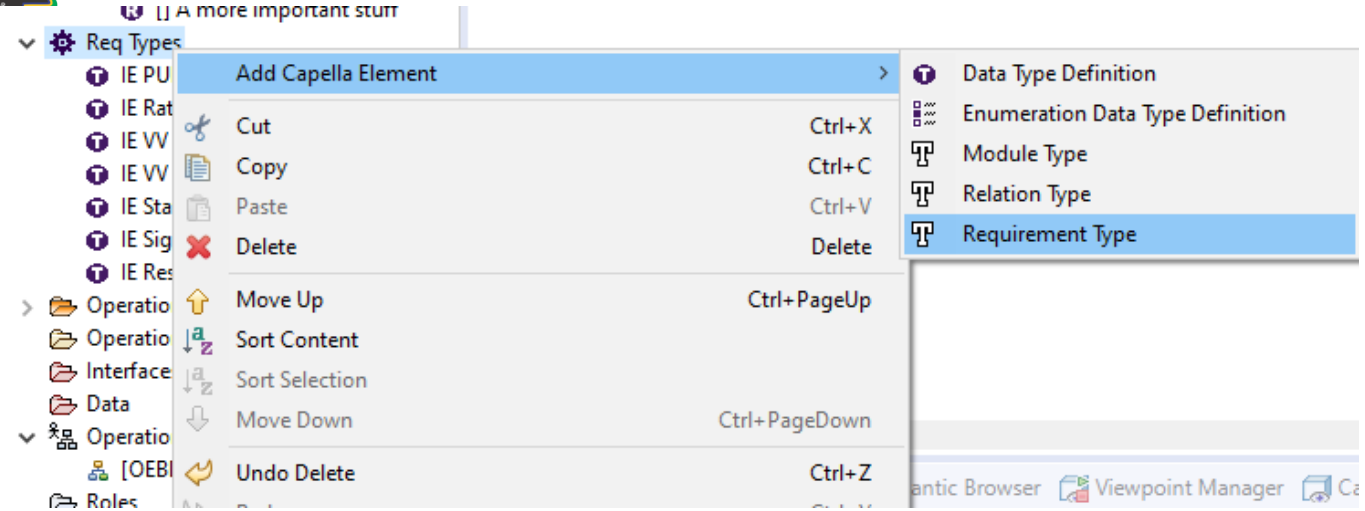
07:16

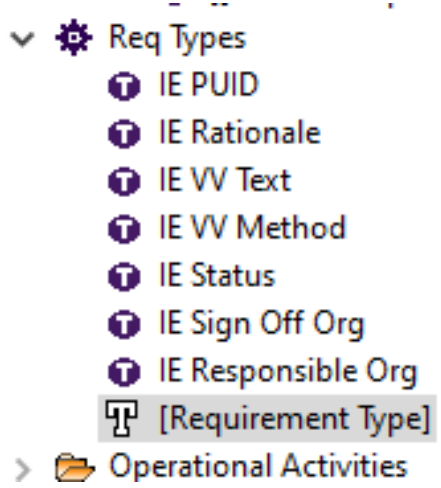


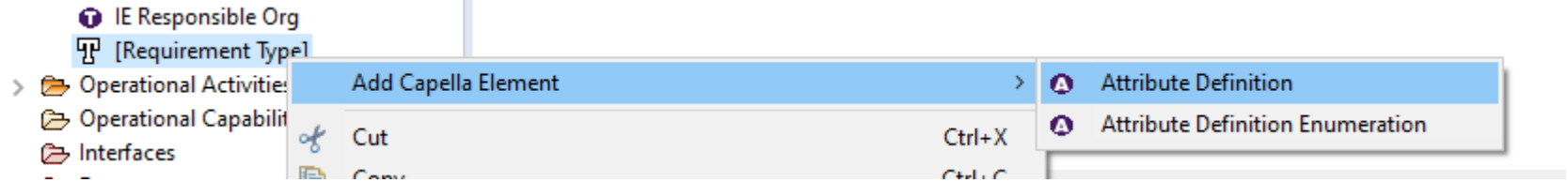
Annotation Query Language (AQL)

- aql:OrderedSet{self.ownedAttributes->select(a | a.definition.ReqIFLongName == '**IE PUID**').value, OrderedSet{self.ReqIFLongName, self.ReqIFText, self.ReqIFChapterName}->select(s | s != 'null' and s.size() > 0)->add(OrderedSet{''})->first()}->sep(' ')

Create the Requirement Type that include the Data types as Attributes

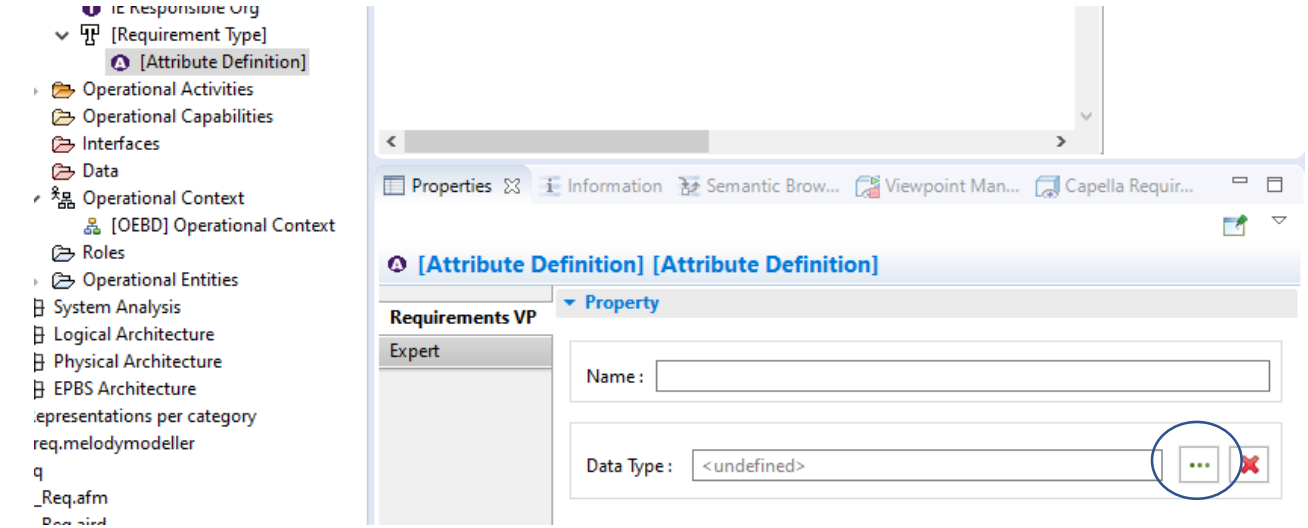


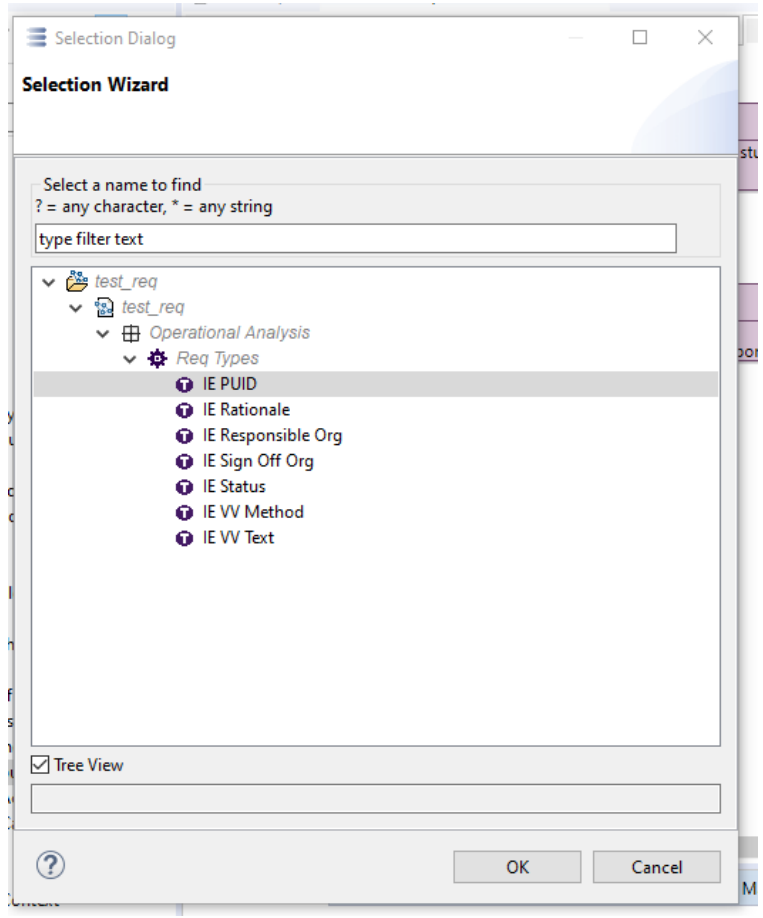


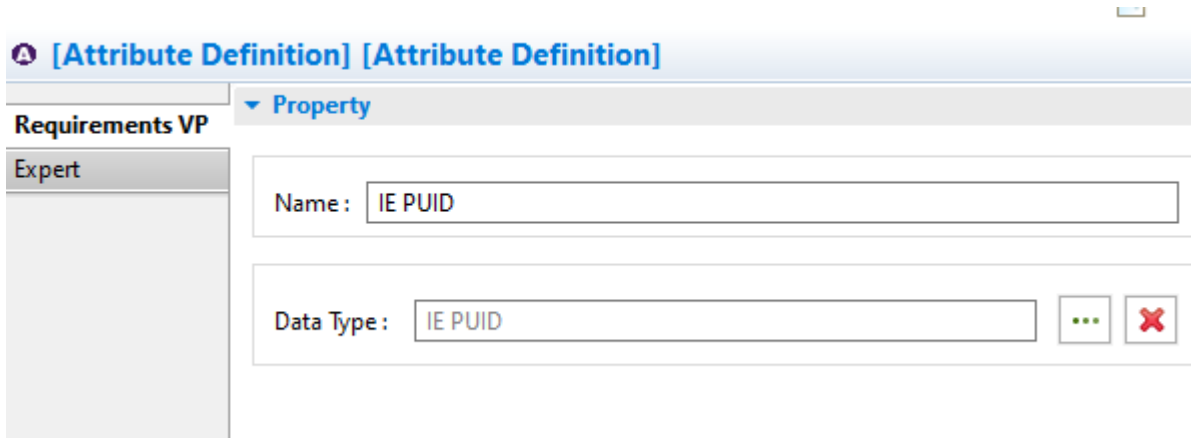


07:16

CONFIGURE THE ATTRIBUTE

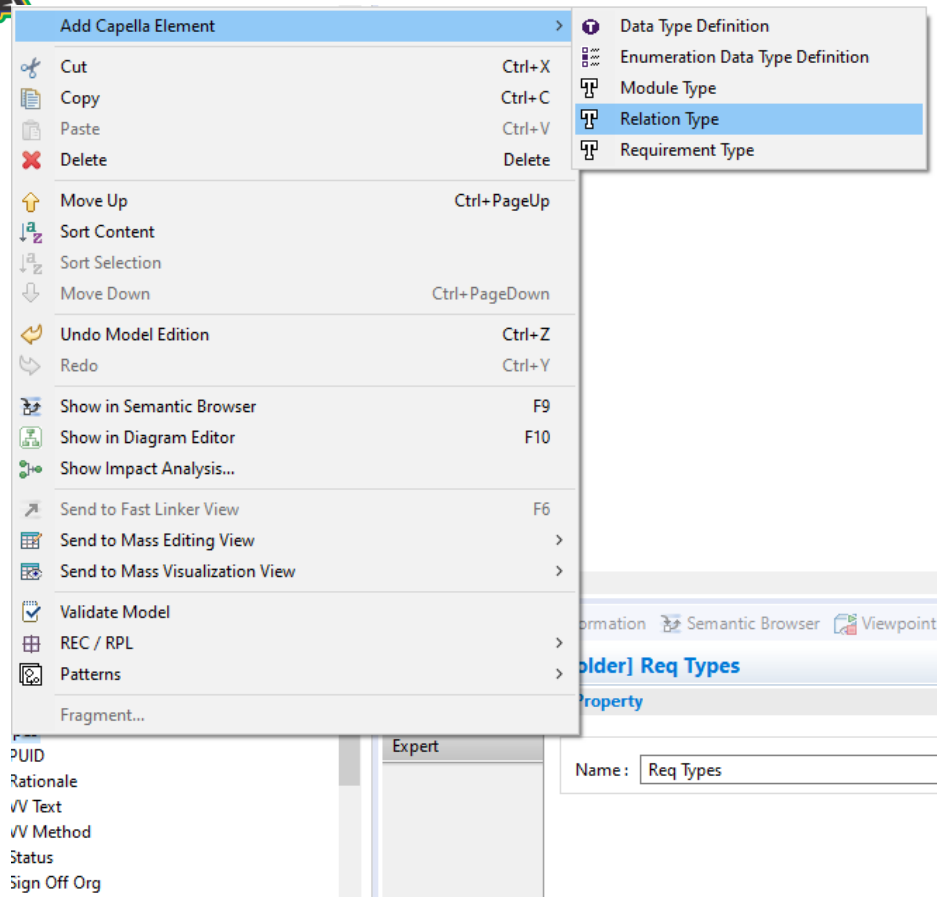


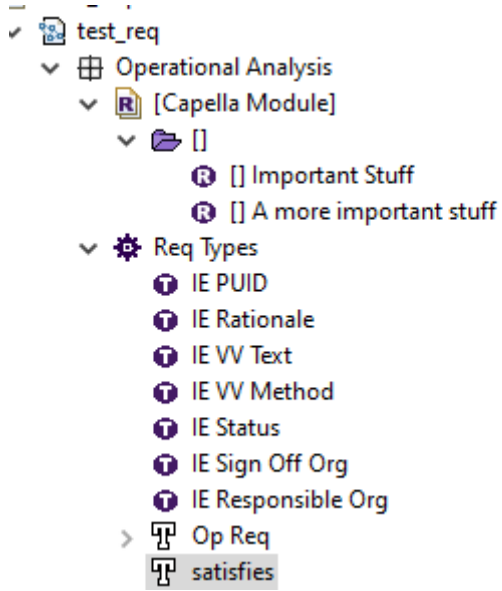


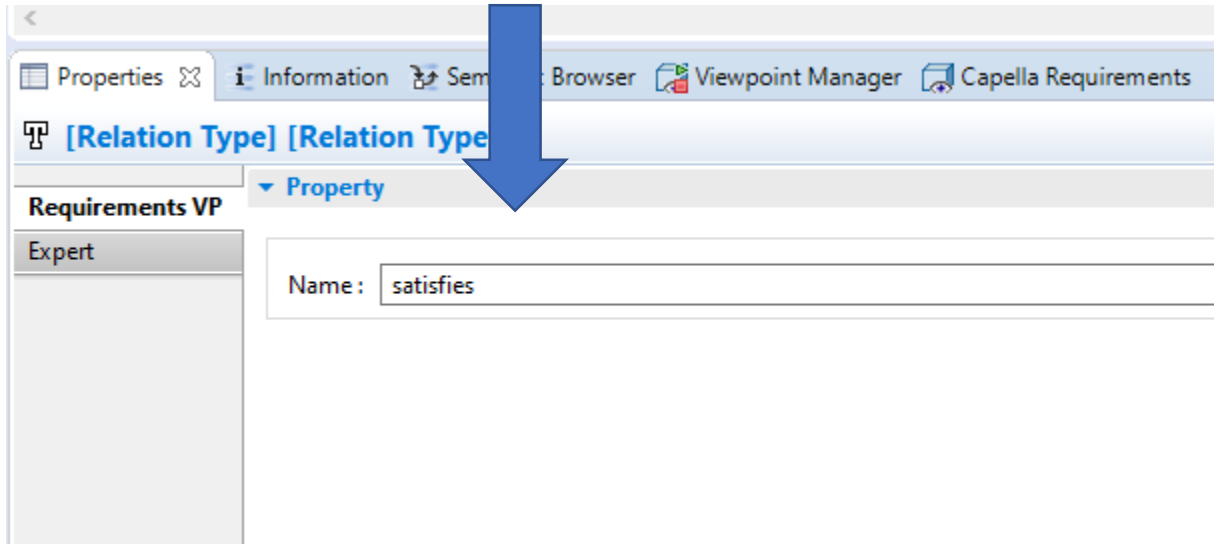


07:16

ADD RELATION METADATA

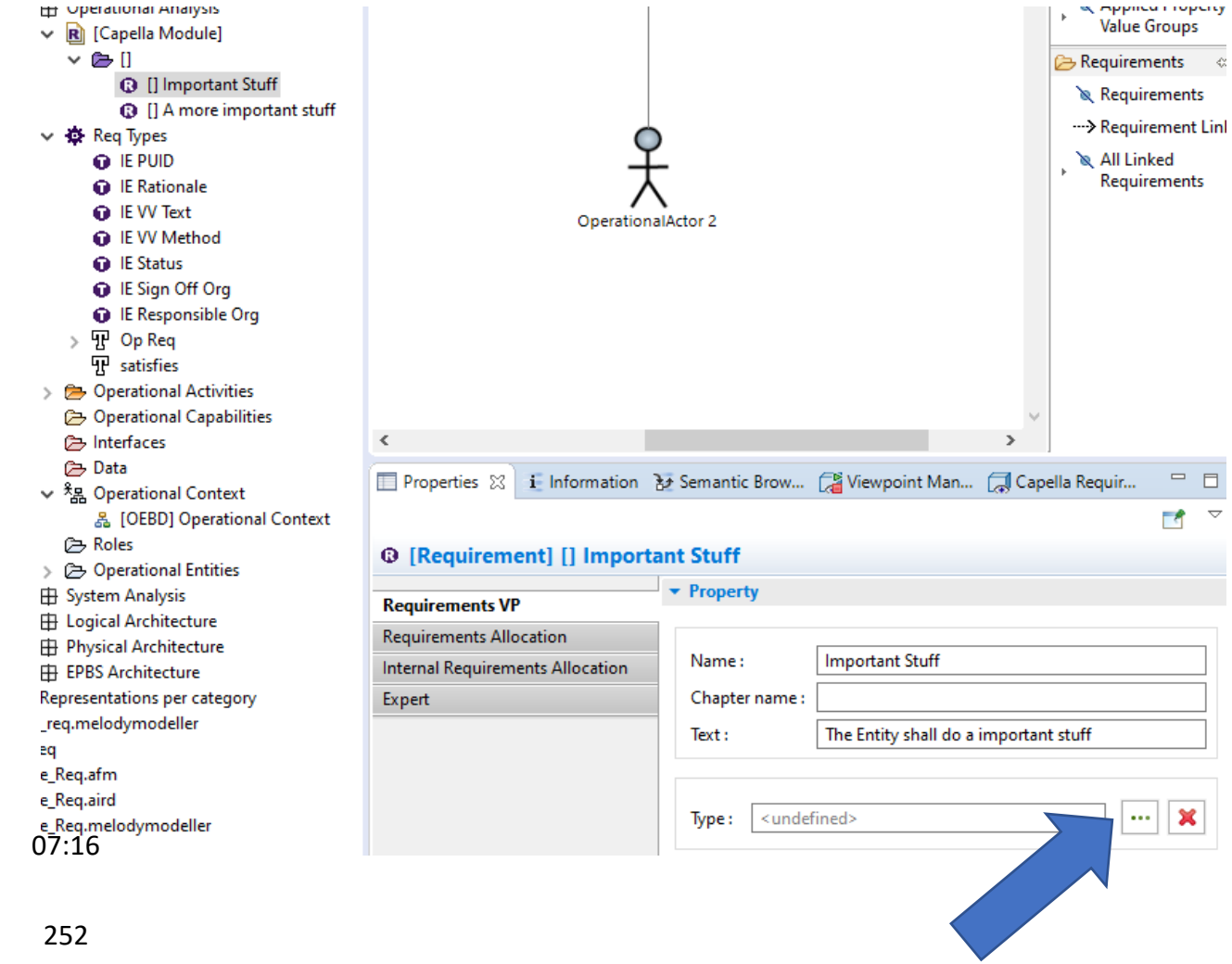


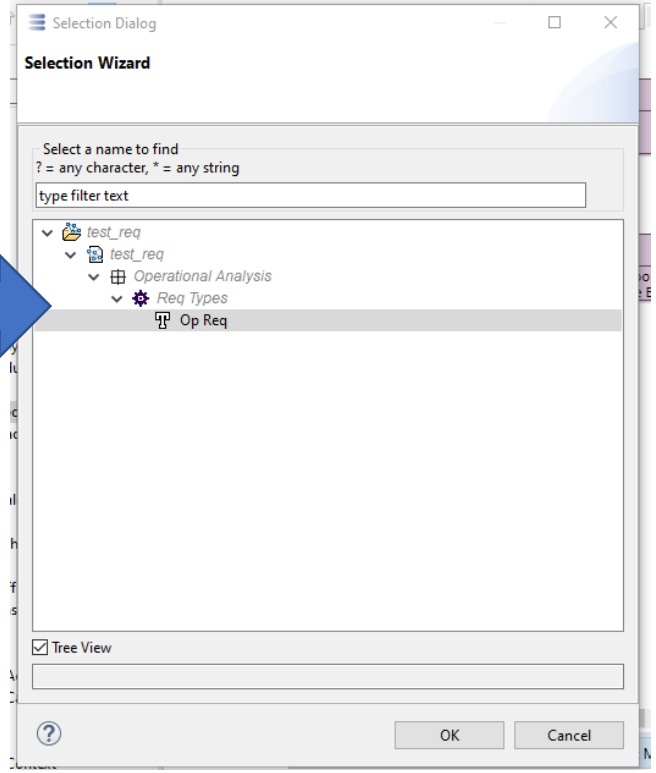




07:16

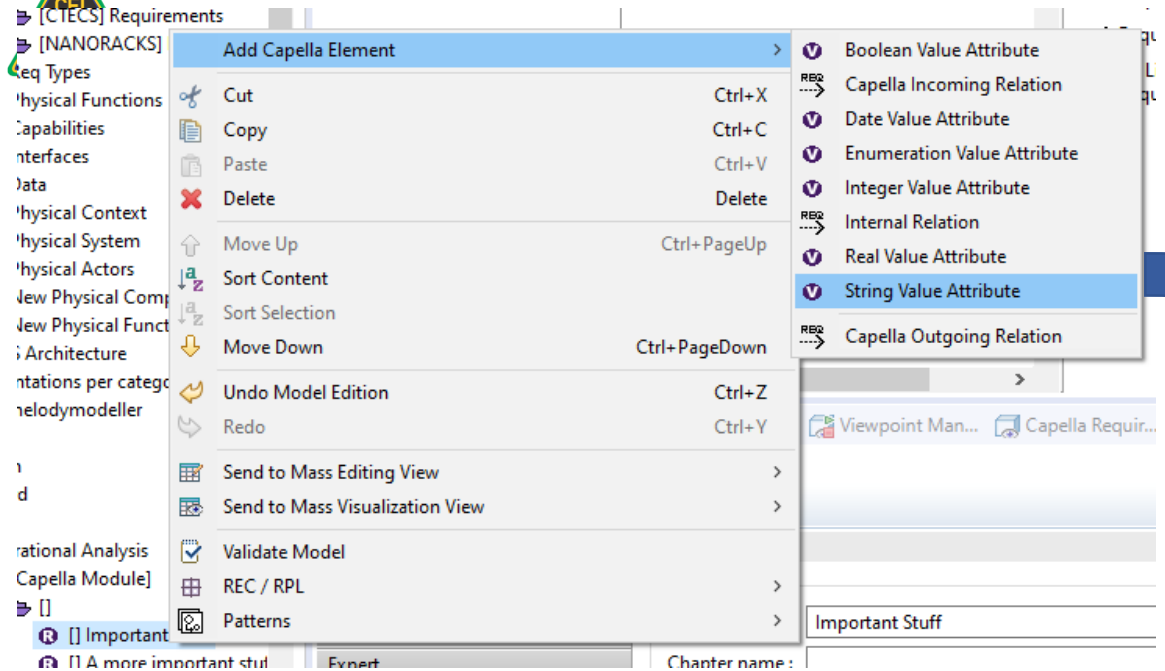
Apply to the Requirement Set

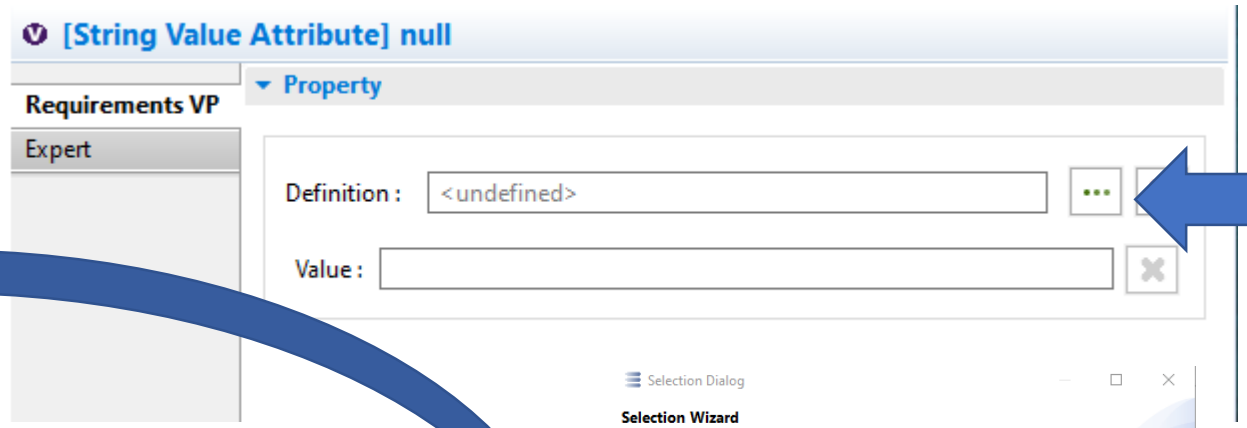


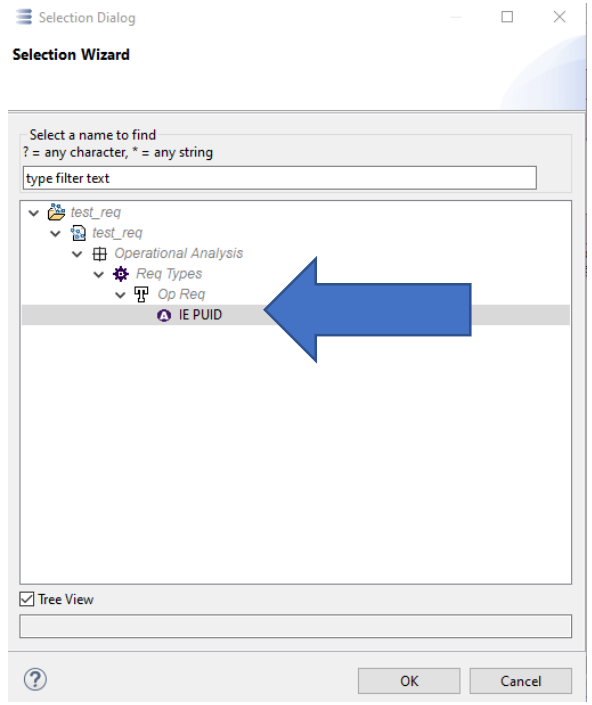


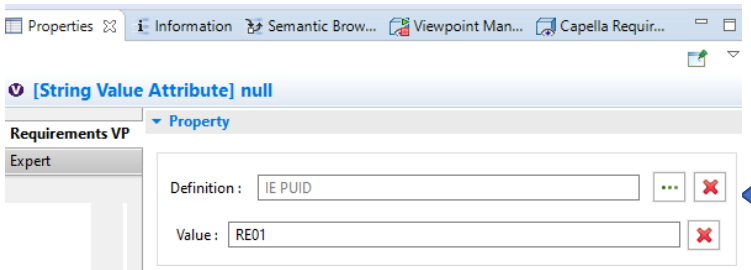
07:16

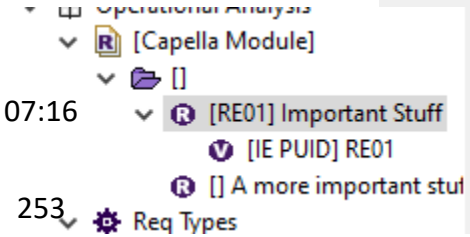
CREATING THE ATTRIBUTES











EACH LAYER HAS A “DEFAULT” REQ RELATION TABLE

- Operational:
  - Activities X Requirements
- System:
  - System Function X Requirements
- Logical
  - Logical Functions x Requirements
  - Logical Component x Requirements
  - Logical Architecture Requirement Refinements
- Physical
  - Physical Functions x Requirements
  - Physical Component x Requirements
- EPBS
  - Configuration Itens x Requirements
  - EPBS Requirement Refinements

07:16

EVERYTHING IS WRITTEN IN XMI

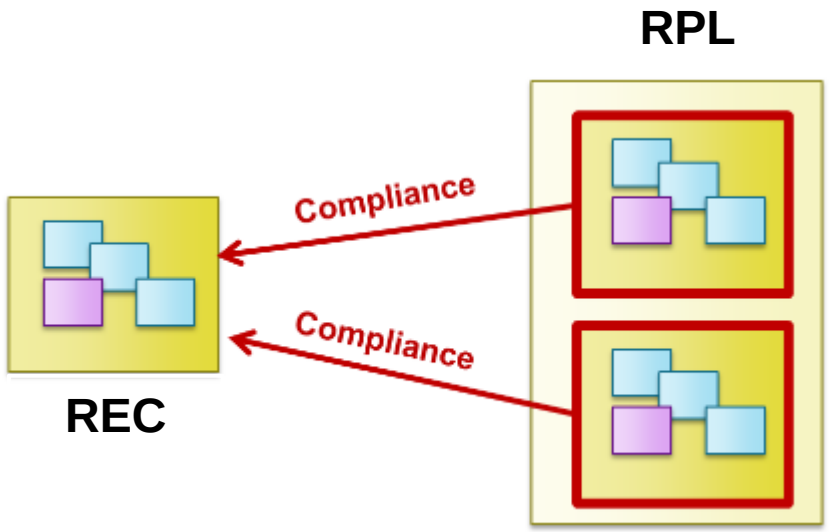
```
181  <ownedDiagramElements xmi:type="diagram:DNodeList" uid="_t5WE8PZlEem5oexdVu1e8g" name="RE01" incomingEdges="_t5dZsPZlEem5oexdVu1e8g">
182    <target xmi:type="Requirements:Requirement" href="test_req.melodymodeller#192c0814-e2aa-40f2-b5b9-09f1c3abf731"/>
183    <semanticElements xmi:type="Requirements:Requirement" href="test_req.melodymodeller#192c0814-e2aa-40f2-b5b9-09f1c3abf731"/>
184    <arrangeConstraints>KEEP_LOCATION</arrangeConstraints>
185    <arrangeConstraints>KEEP_SIZE</arrangeConstraints>
186    <arrangeConstraints>KEEP_RATIO</arrangeConstraints>
187    <ownedStyle xmi:type="diagram:FlatContainerStyle" uid="_t5WsAPZlEem5oexdVu1e8g" borderSize="1" borderSizeComputationExpression="1" borderColor="114,73,110" bac
188      <description xmi:type="style:FlatContainerStyleDescription" href="platform:/plugin/org.polarsys.capella.vp.requirements.design/description/CapellaRequirement
189    </ownedStyle>
190    <actualMapping xmi:type="description_1:ContainerMapping" href="platform:/plugin/org.polarsys.capella.vp.requirements.design/description/CapellaRequirements.ode
191    <ownedElements xmi:type="diagram:DNodeListElement" uid="_t5bkgPZlEem5oexdVu1e8g" name="- Important Stuff&#xA;- The Entity shall do a important stuff">
192      <target xmi:type="Requirements:Requirement" href="test_req.melodymodeller#192c0814-e2aa-40f2-b5b9-09f1c3abf731"/>
193      <semanticElements xmi:type="Requirements:Requirement" href="test_req.melodymodeller#192c0814-e2aa-40f2-b5b9-09f1c3abf731"/>
194      <ownedStyle xmi:type="diagram:Square" uid="_t5bkgfZlEem5oexdVu1e8g" showIcon="false" labelPosition="node">
195        <description xmi:type="style:SquareDescription" href="platform:/plugin/org.polarsys.capella.vp.requirements.design/description/CapellaRequirements.o
196      </ownedStyle>
197      <actualMapping xmi:type="description_1:NodeMapping" href="platform:/plugin/org.polarsys.capella.vp.requirements.design/description/CapellaRequirements.od
198    </ownedElements>
199  </ownedDiagramElements>
200  <ownedDiagramElements xmi:type="diagram:DEdge" uid="_t5dZsPZlEem5oexdVu1e8g" sourceNode="_nwDn8PZkEem5oexdVu1e8g" targetNode="_t5WE8PZlEem5oexdVu1e8g">
```

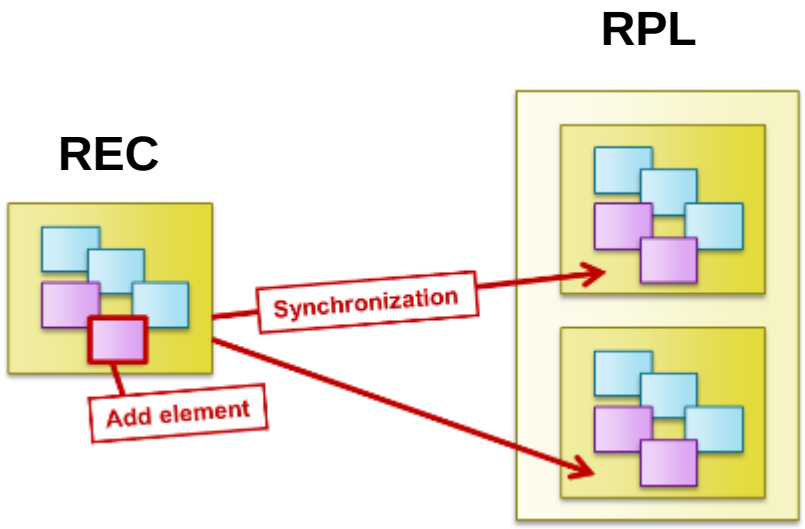
Replicable Elements Collection (REC) e Replicas (RPL)

Written by Mateus S. Venturini

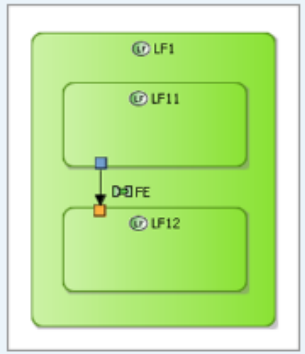
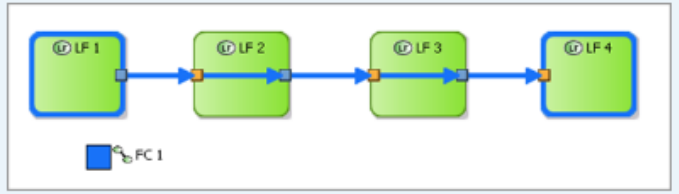
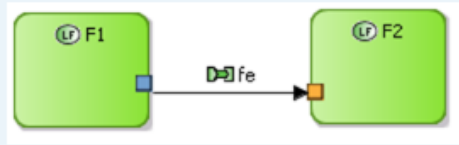
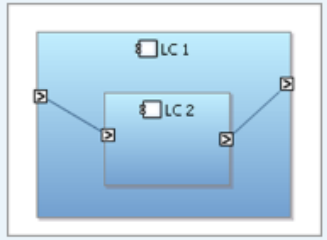
07:16

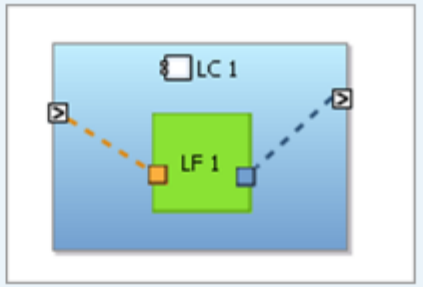
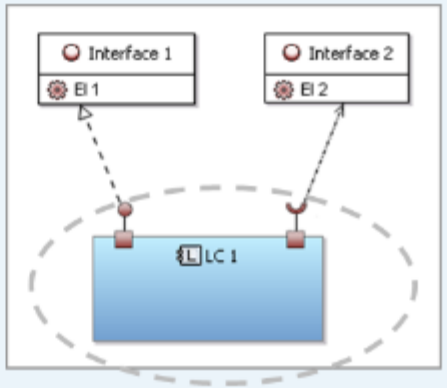
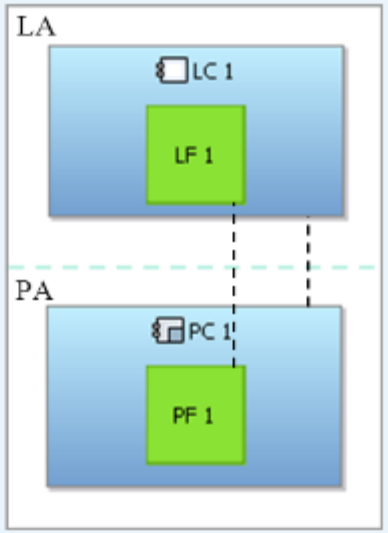
Definition


RPL
Compliance
Compliance
REC


RPL
REC
Synchronization
Add element

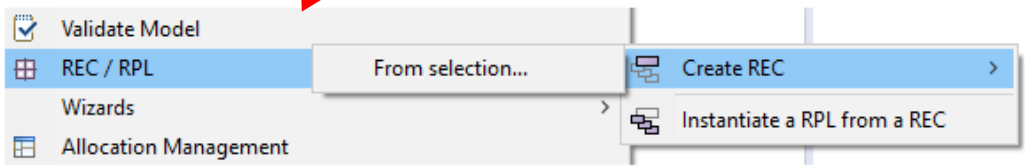
EXAMPLES

A Function and its sub functions (mono-root)	
A Functional Chain and the Functions it involves (multi-root)	
Two Functions and a Functional Exchange between them (multi-root)	
A Component and its Sub Components (mono-root)	

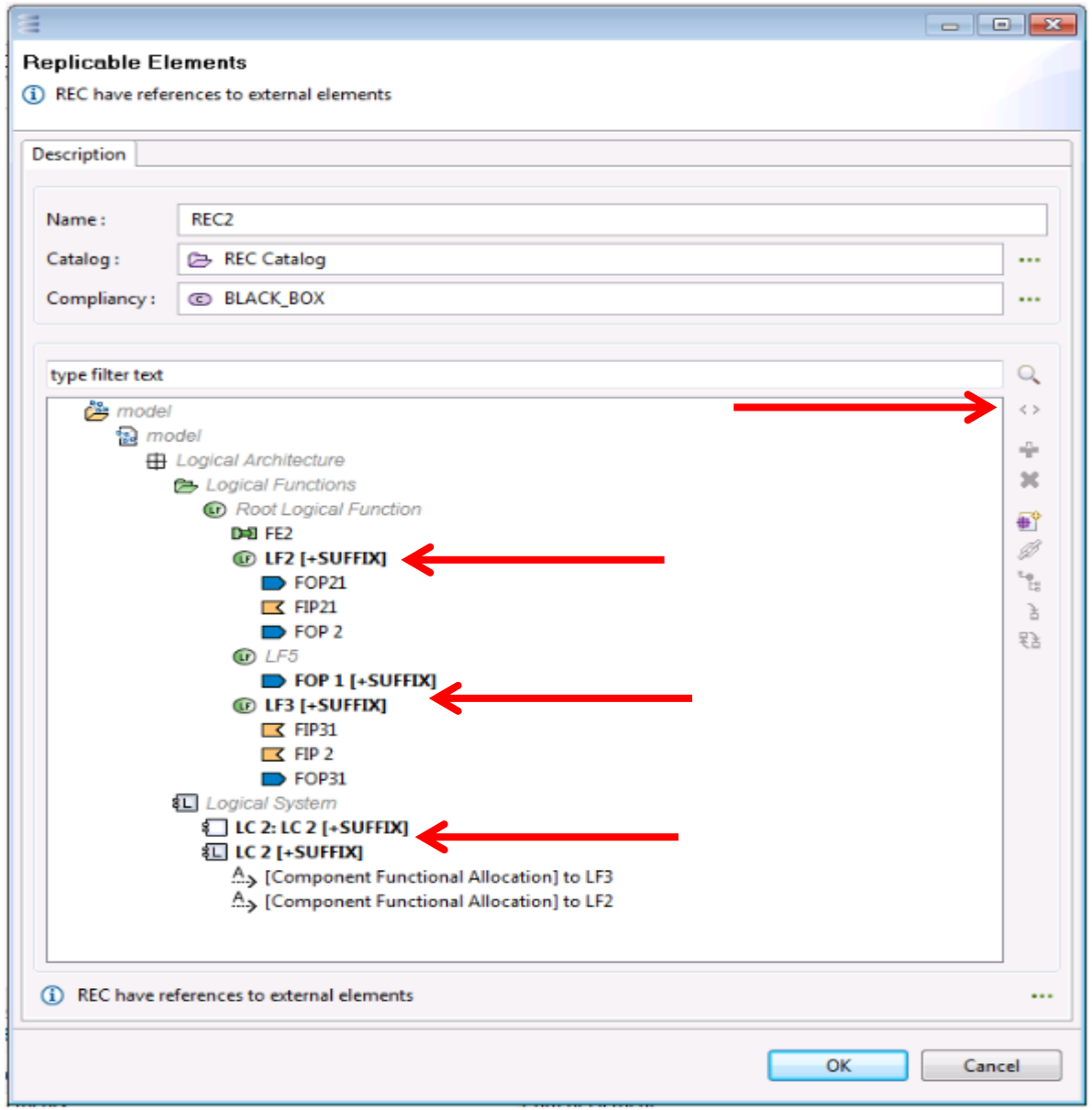
A Component and its allocated Functions (mono-root)	
A Component providing and requiring Interfaces located outside the REC (mono-root)	
A Physical Component and the Logical Components it realizes, including Functions and any other element (multi-root)	

CREATING

Right Click



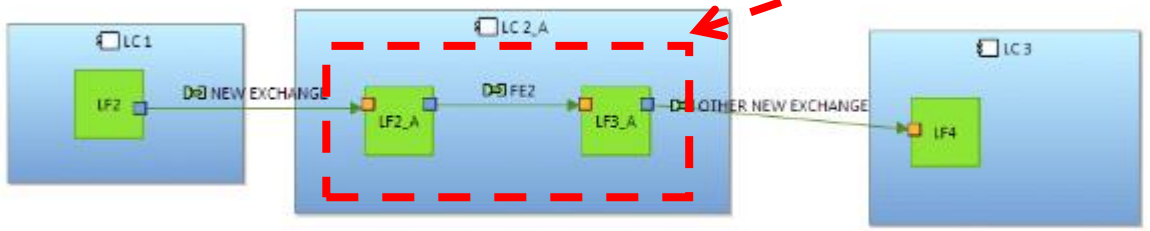
Select the components to replicate

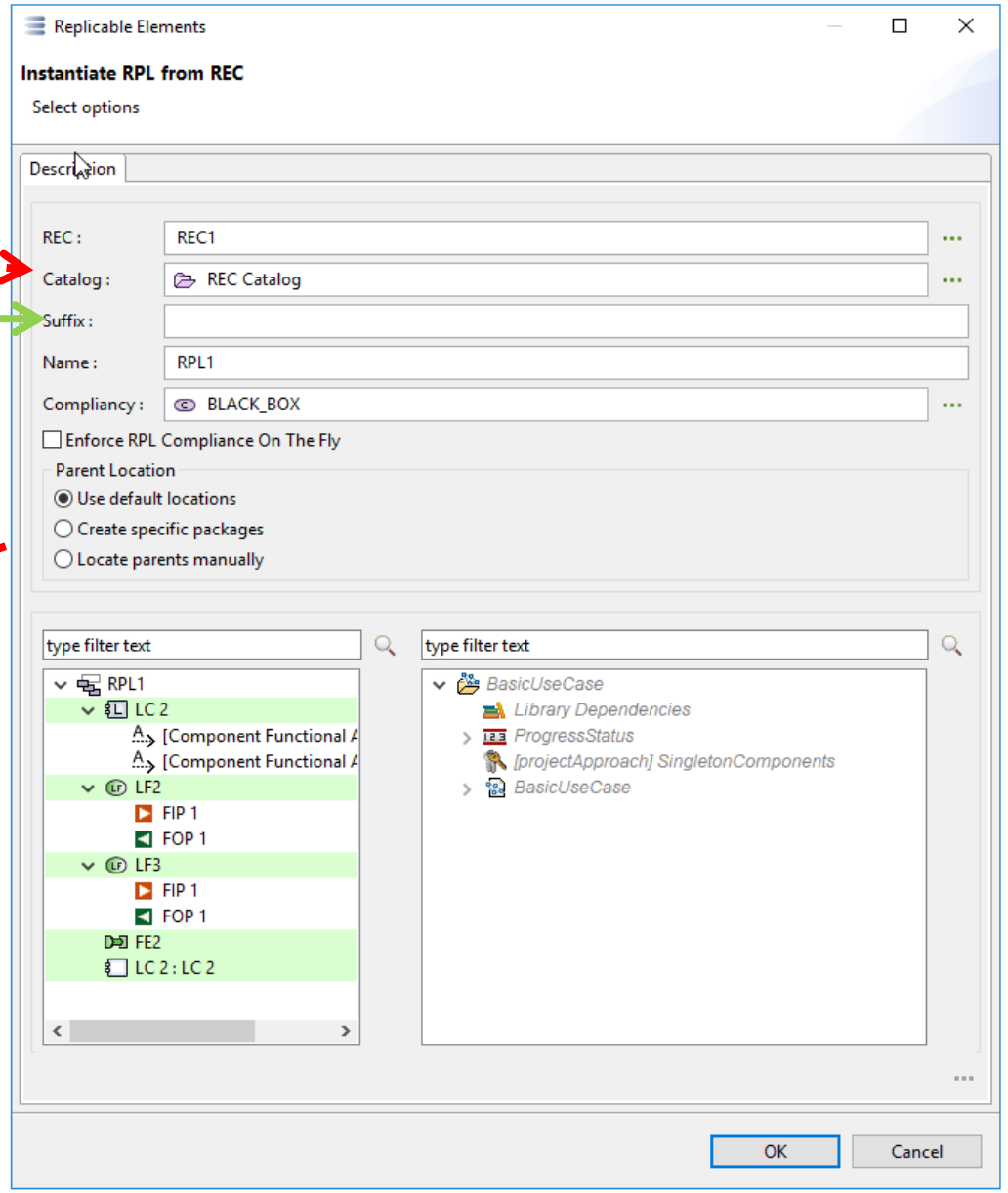


07:16

USING

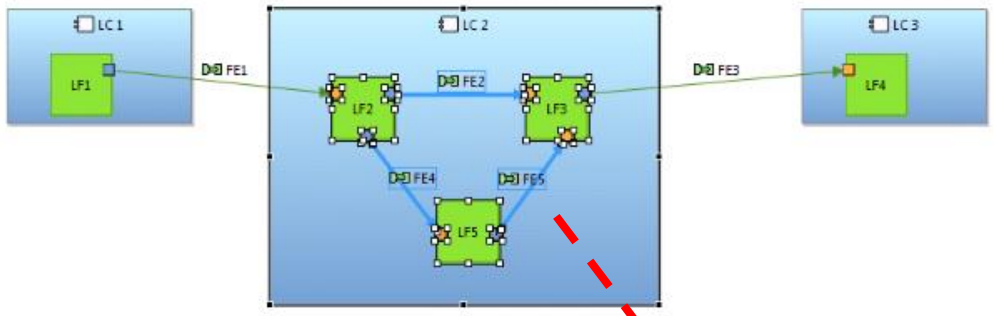

Validate Model
REC / RPL
Transitions
Wizards
Create REC
Instantiate a RPL from a REC

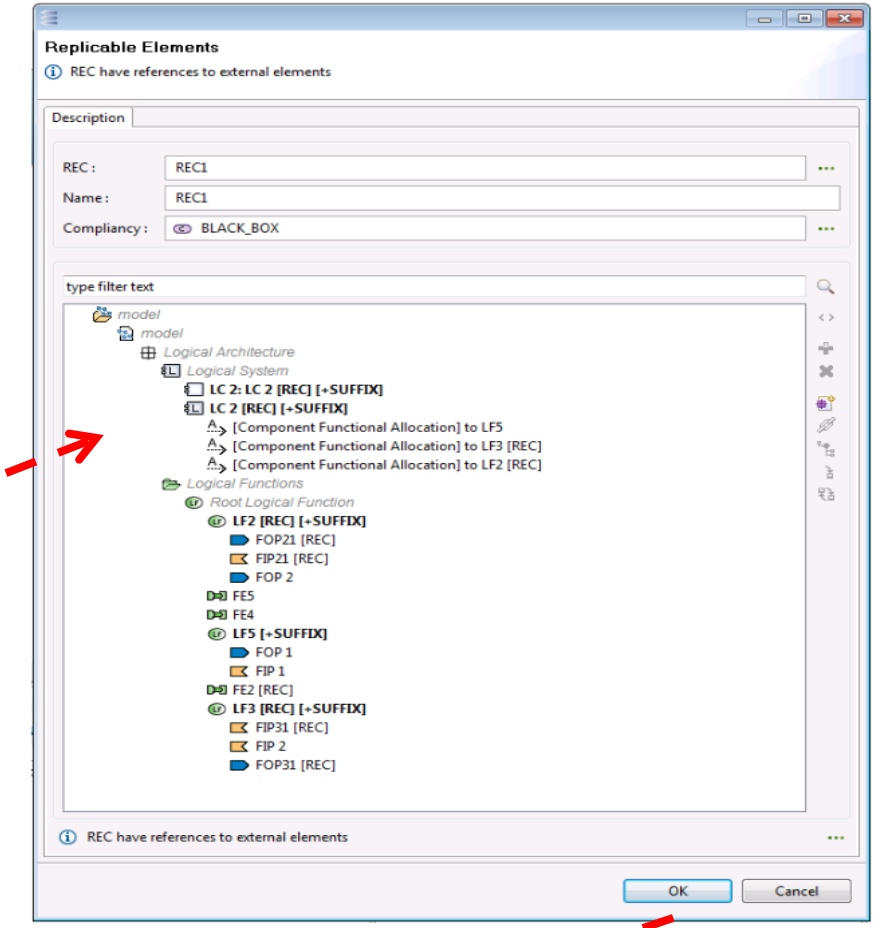

LC 1
LF2
NEW EXCHANGE
LC 2_A
LF2_A
FE2
LF3_A
OTHER NEW EXCHANGE
LC 3
LF4

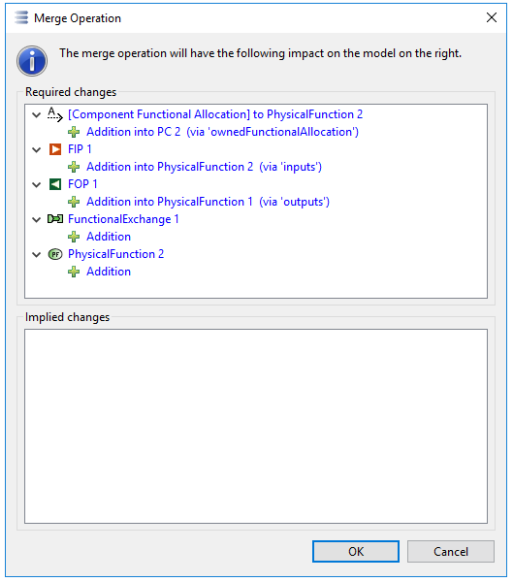

Replicable Elements
Instantiate RPL from REC
Select options
Description
REC :
REC1
Catalog :
REC Catalog
Suffix :
Name :
RPL1
Compliancy :
BLACK_BOX
Enforce RPL Compliance On The Fly
Parent Location
Use default locations
Create specific packages
Locate parents manually
type filter text
RPL1
LC 2
[Component Functional A
[Component Functional A
LF2
FIP 1
FOP 1
LF3
FIP 1
FOP 1
FE2
LC 2 : LC 2
type filter text
BasicUseCase
Library Dependencies
ProgressStatus
[projectApproach] SingletonComponents
BasicUseCase
OK
Cancel

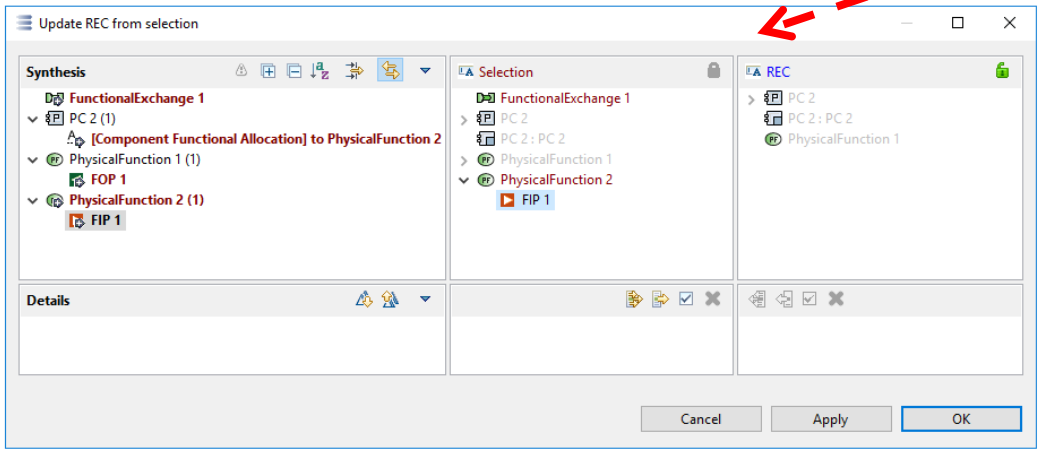
07:16

UPDATE REC from the RPL



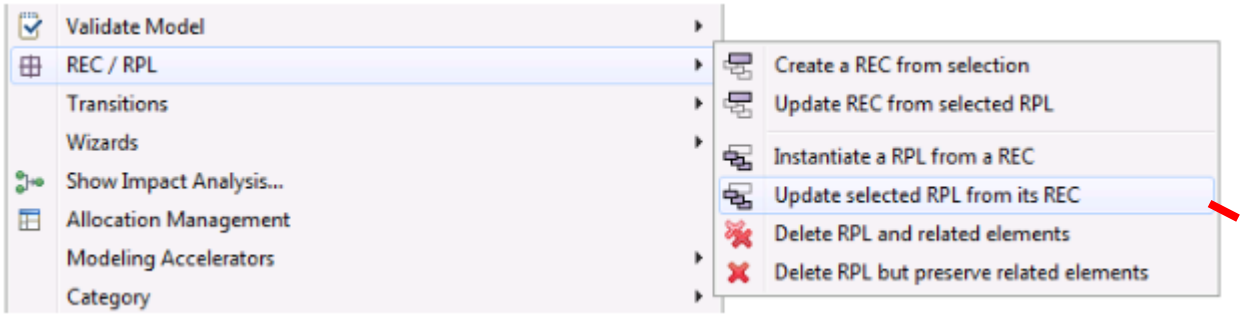


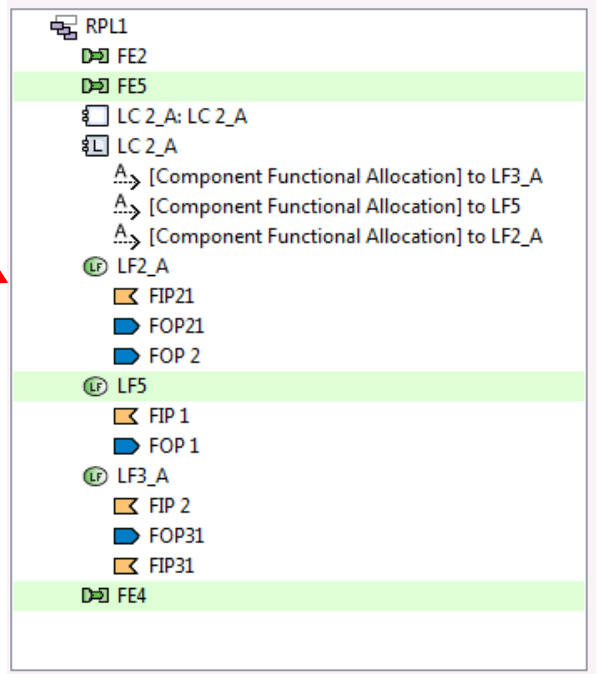


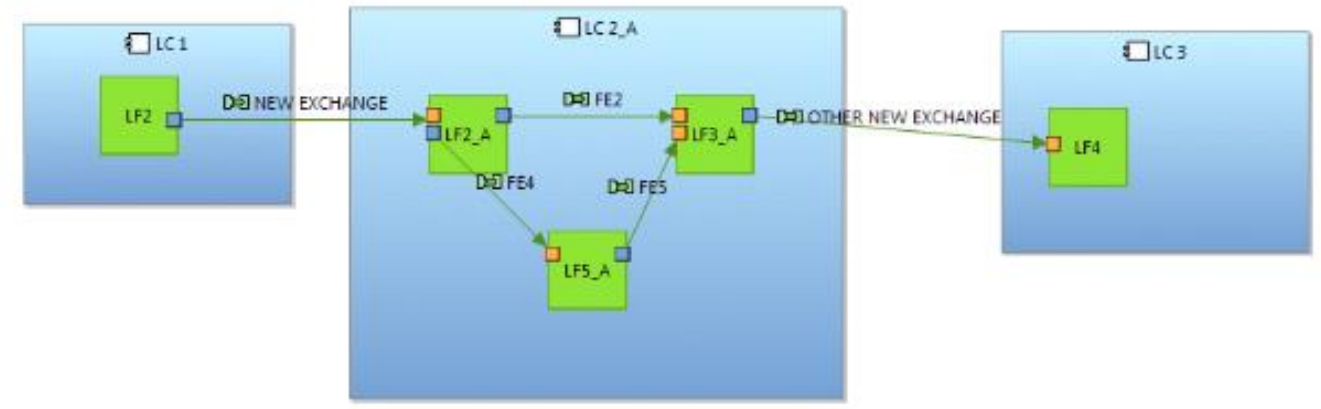


07:16

Update RPL from the REC







07:16

REFERENCES OF THE REC->RPL

- **[HOW TO] Replicate model elements in Capella (4’25”)**
- https://www.youtube.com/watch?v=h-ax61eVlxM

- **Webinar - Strategies and tools for model reuse with Capella (58’23”)**
- https://www.youtube.com/watch?v=I28EhAXe-i8

- **In-Flight Entertainment System (IFE) – Example**
- https://download.eclipse.org/capella/samples/1.3.1/InFlightEntertainmentSystem.zip

- **Capella Help – Replicable Elements**