Tutorial to create a basic viewpoint like this one

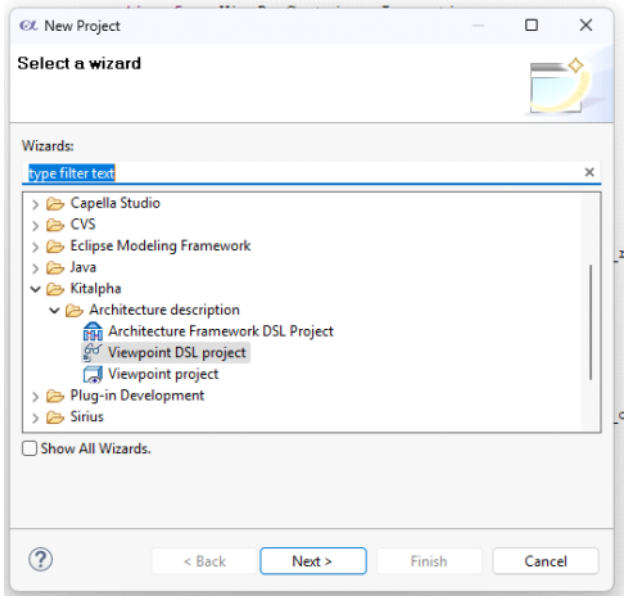
01 October 2025 19:16

Disclaimer: I'm not a developer, and I do not have a lot of experience with coding languages, except for Matlab, which I didn't use here. So this viewpoint will be extremely basic, also because the documentation is scarce and the help is non-existent (yes, there is the Capella forum, but nobody answers questions on the Capella Studio section).

Phase 1: DATA

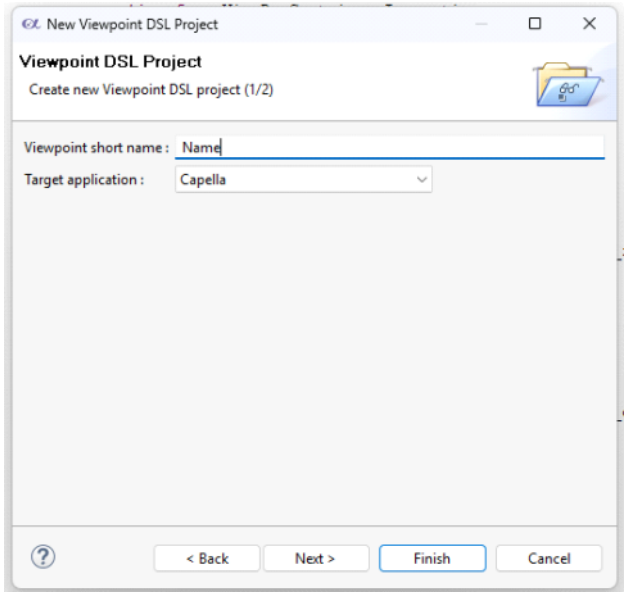
The objective of this viewpoint is to create a faster and standardised way to create requirements that could easily fill a Verification Matrix. So the idea was to create a new Class with attributes that will fill the Verification Matrix.

First things first: create the viewpoint. After downloading Capella Studio (Capella MBSE Tool - Download) and choosing the workspace, you need to create a new project:



Select Viewpoint DSL Project, click Next

.



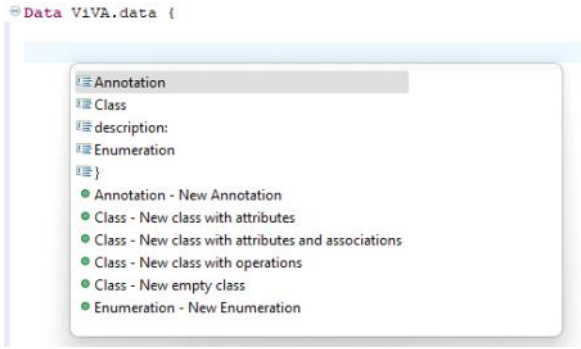
Insert your name and select the target application: Capella. Click Finish.

Automatically, you will be in front of the Name.spec.vptext page (my viewpoint is called ViVA):

```
*ViVA.spec.vptext | ViVA.diagram.vptext | ViVA.ui.vptext | ViVA.data.vptext
/**
 * Copyright (c) PolarSys, 2025. All rights reserved.
 *
 *  Viewpoint ViVA
 *  @author: Chiara.My
 *  @date: 01/10/2025
 *
 */
Viewpoint ViVA {
    name: "ViVA"
    Data ViVA.data
}
```

Put your pointer on Data and click on F3 to open the ViVA.data.vptext. I suggest reading a little bit around the documentation first, but here is where the classes and all your data will be created.

Let's start creating a new Class inside the brackets. To get suggestions on the possible option you can use, click on Ctrl+Spacebar. You will get something like this:



A Class is like a blueprint for creating objects. It defines the structure and behaviour that the objects (sometimes called "instances") will have.
Inside these classes, you can add:

- Icons
- Attributes
- Associations
- Annotations
- etc

To be able to add icons, you need to first add them in the icons folder inside the org.polarys.capella.vp.viva.vpdsl folder (just drop them there). Then you can recall them. Be mindful of the image's size (tip: 16x16 pixels is perfect).

Inside Attributes, put all the information that your class will have, inside associations, the connections with other objects.
The attributes will have different types, they can be strings, lists, floats, integers, …. Choose what you need. A specific type that I will use is Enumeration, this type will give you the possibility to choose from a list (eg the Enumeration) that you will define after the Class. For my viewpoint, I need certain information inside my requirements, so my Class looks like this:

```
Class VivaRequirement {
    icon: "verification.png"
    extends pa.PhysicalNode

    Attributes:
        identification type ecore.EString
        nameReq type ecore.EString
        text type ecore.EString
        justification type ecore.EString
        ecss enum ECSSdivision
        remarks type ecore.EString
        verificationmodel enum VerificationModel
        closeOutReference type ecore.EString
        verificationComment type ecore.EString
        verificationStatus enum VerificationStatus
        rfdRfw type ecore.EString

    Associations:
        rodAss contains [0,*] ROD
        anAss contains [0,*] Analysis
        inAss contains [0,*] Inspection
        teAss contains [0,*] Test
        pcRef refers [0,*] external modellingcore.ModelElement
        reqReqAss refers [0,*] VivaRequirement
}
```

With these enumerations defined:

```
Enumeration ECSSdivision {
 empty ,Mission, Functional, Operational, Design, Environmental, Interface, Physical, Configuration, PA, ILS, "Human Factor", Verification
}

Enumeration VerificationModel {
empty, EM, EQM, "EQM-T", FM, PFM
}

Enumeration VerificationStatus {
Open, Closed
}

Enumeration ComplianceStatus {
empty, Compliance, "Non Compliance",  "Non Applicable"
}

Enumeration VerificationStage {
empty, Qualification, Acceptance
}
```

As you can see, the first option is "empty". If you leave it like this, your option will actually show the word empty. We will turn back on this later.

Then I defined the Verification Methods that a Requirement can have (following the ECSS standards). You can either define an abstract class and then create new classes with the types of the abstract class, or create different classes all similar.

The first option would look like this:

```
Class abstract{
    abstract: true

    Attributes:
        stage enum VerificationStage
        compliance enum ComplianceStatus
        complianceComment type ecore.EString
        justification type ecore.EString
}

Class ROD{
    superClass abstract
}

Class Analysis{
    superClass abstract
}

Class Inspection{
    superClass abstract
}

Class Test{
    superClass abstract
}
```

While the second option would look like this:

```
Class ROD {
    icon: "rod.png"
    Attributes:
        stage enum VerificationStage
        compliance enum ComplianceStatus
        complianceComment type ecore.EString
        justification type ecore.EString
}

Class Analysis {
    icon: "analysis.png"
    Attributes:
        stage enum VerificationStage
        compliance enum ComplianceStatus
        complianceComment type ecore.EString
        justification type ecore.EString
}

Class Inspection {
    icon: "inspection.png"
    Attributes:
        stage enum VerificationStage
        compliance enum ComplianceStatus
        complianceComment type ecore.EString
        justification type ecore.EString
}

Class Test {
    icon: "test.png"
    Attributes:
        stage enum VerificationStage
        compliance enum ComplianceStatus
        complianceComment type ecore.EString
        justification type ecore.EString
        specification type ecore.EString
}
```

It is recommended to use the first option, but, I had some troubles in later development (for my own lack of information), so I followed the second option.

Some more comments:
Inside the Associations group, you need to put the classes that you want to connect. For example, I put 4 connections with my 4 verification methods (you can just

put one if you use the abstract Class) because I want to assign a verification Method to my requirement. You can also just put an enumeration, but I decided to go this way for 2 big reasons:

1. I want to add more information inside the verification methods classes
2. I want to have a visual representation of some information (you'll see later)

To be able to assign classes to my Requirement Class, I used "contains". Then I used "refers" for two edges that will be created later. Edges are basically arrows that can connect elements. In my project, I want to be able to connect requirements to each other ("reqReqAss refers [0,*] VivaRequirement"), but I also want to connect requirements to elements in my diagram ("pcRef refers [0,*] external modellingcore.ModelElement ").

In the end, when you have defined all your data, you can save the page (Ctrl+S).

My final code:

```
Data ViVA.data {
Class VivaRequirement {
icon: "verification.png"
extends pa.PhysicalNode
Attributes:
reqIdentification type ecore.EString
nameReq type ecore.EString
text type ecore.EString
justification type ecore.EString
ecss enum ECSSdivision
remarks type ecore.EString
verificationmodel enum VerificationModel
closeOutReference type ecore.EString
verificationComment type ecore.EString
verificationStatus enum VerificationStatus
rfdRfw type ecore.EString
Associations:
rodAss contains [0,*] ROD
anAss contains [0,*] Analysis
inAss contains [0,*] Inspection
teAss contains [0,*] Test
pcRef refers [0,*] external modellingcore.ModelElement
reqReqAss refers [0,*] VivaRequirement
}
Class ROD {
icon: "rod.png"
Attributes:
stage enum VerificationStage
compliance enum ComplianceStatus
complianceComment type ecore.EString
justification type ecore.EString
}
Class Analysis {
icon: "analysis.png"
Attributes:
stage enum VerificationStage
compliance enum ComplianceStatus
complianceComment type ecore.EString
justification type ecore.EString
}
Class Inspection {
icon: "inspection.png"
Attributes:
stage enum VerificationStage
compliance enum ComplianceStatus
complianceComment type ecore.EString
justification type ecore.EString
}
Class Test {
icon: "test.png"
Attributes:
stage enum VerificationStage
compliance enum ComplianceStatus
complianceComment type ecore.EString
justification type ecore.EString
specification type ecore.EString
}
Enumeration ECSSdivision {
empty ,Mission, Functional, Operational, Design, Environmental, Interface,
Physical, Configuration, PA, ILS, "Human Factor", Verification
}
Enumeration VerificationModel {
empty, EM, EQM, "EQM-T", FM, PFM
}

Enumeration VerificationStatus {
Open, Closed
}
Enumeration ComplianceStatus {
empty, Compliance, "Non Compliance",  "Non Applicable"
}
Enumeration VerificationStage {
empty, Qualification, Acceptance
}
}
```

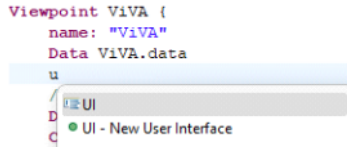
BIG DISCLAIMER!!!
The name that you give to the Requirement Class will influence the Verification Matrix (if you decide to use it)

UI: User Interface

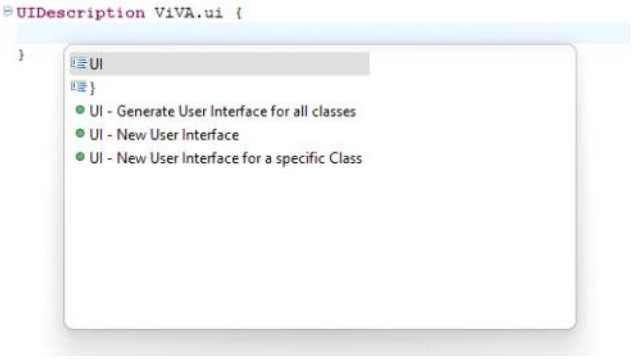
02 October 2025 12:00

Phase 2: User Interface

Now we have to define how the user will interact with our Classes and its attributes. In the .spec.vptext page, under Data, type UI:



Click on New User Interface, a new tab will open, just click on Finish.
The .ui.vptext will be automatically open. This is the easiest part because clicking on "Generate User Interface for all classes". It will be automatically done.



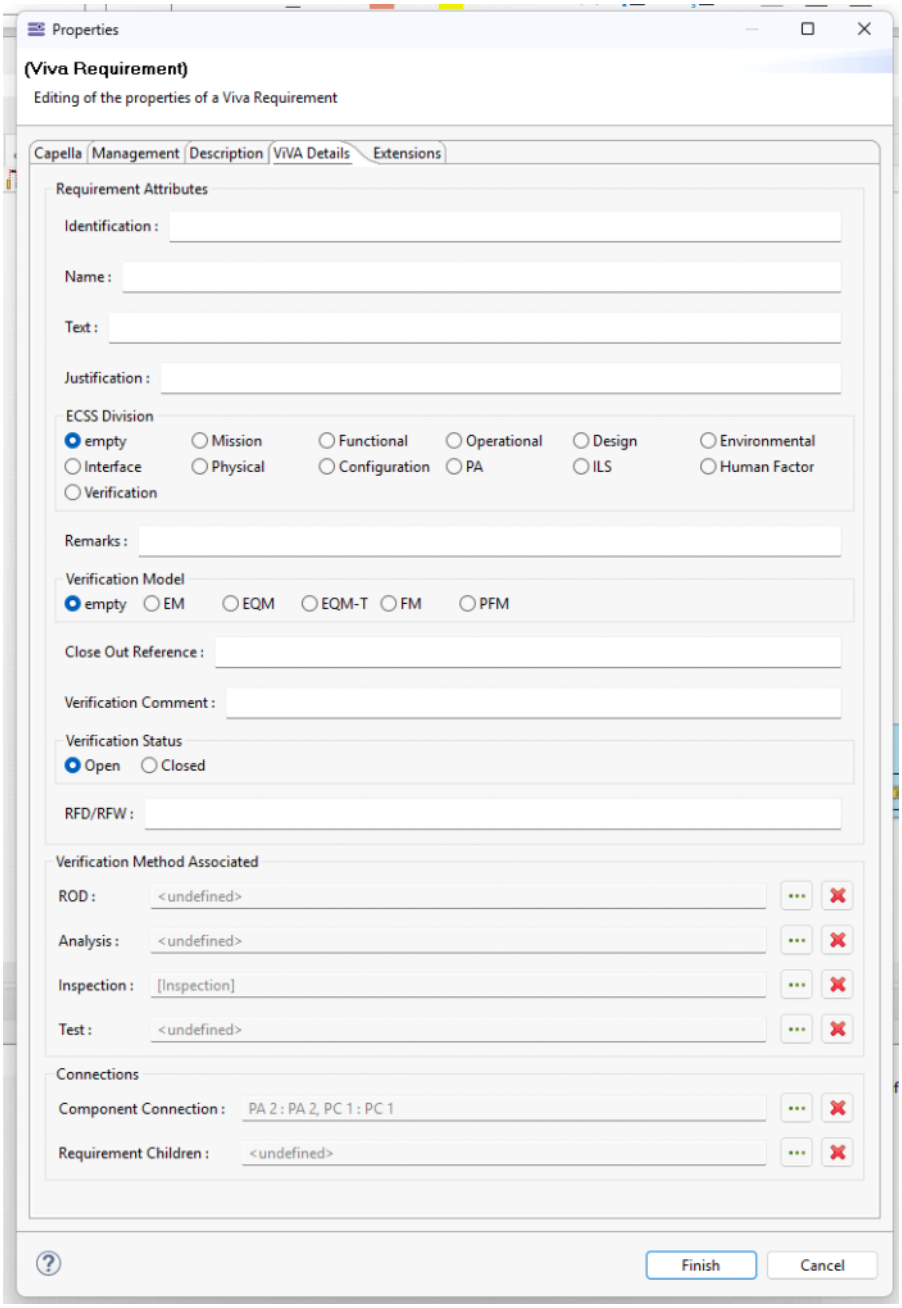
Now you can easily customise your information.
Main stuff you can easily change:

- Labels
- Different containers
- Types -> it's gonna be text for type strings, but for enumeration can be customized in radiobox, checkbox, multipleChoiceList, etc. find the best for you.

You can choose to not show some info, to create multiple containers or sections.
This is my code and the result once the viewpoint is ready to be used:

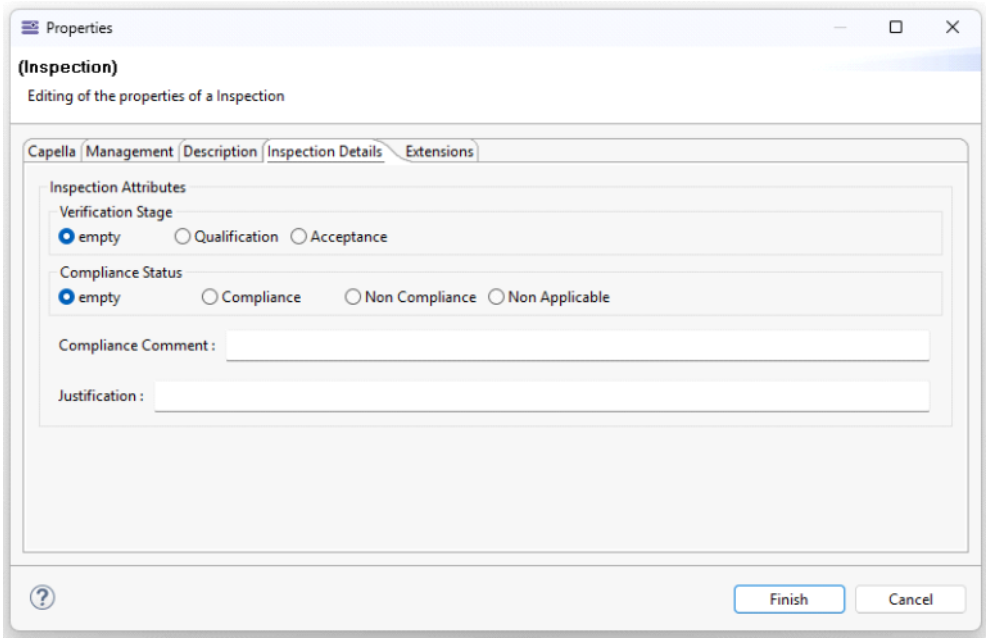
```
UIDescription ViVA.ui {
UI ViVA_VivaReq {
label: "ViVA Details"
Container ViVA_VivaReq_Section {
Container ViVA_VivaReq_AttributeGroup {
label: "Requirement Attributes"
Field reqIdentificationField label: "Identification" type text , mapped-to ViVA.data.VivaRequirement.identification
Field nameReqField label: "Name" type text , mapped-to ViVA.data.VivaRequirement.nameReq
Field ^textField label: "Text" type text , mapped-to ViVA.data.VivaRequirement.^text
Field justificationField label: "Justification" type text , mapped-to ViVA.data.VivaRequirement.justification
Field ecssField label: "ECSS Division" type radiobox , mapped-to ViVA.data.VivaRequirement.ecss
Field remarksField label: "Remarks" type text , mapped-to ViVA.data.VivaRequirement.remarks
Field verificationmodelField label: "Verification Model" type radiobox , mapped-to ViVA.data.VivaRequirement.verificationmodel
Field closeOutReferenceField label: "Close Out Reference" type text , mapped-to ViVA.data.VivaRequirement.closeOutReference
Field verificationCommentField label: "Verification Comment" type text , mapped-to ViVA.data.VivaRequirement.verificationComment
Field verificationStatusField label: "Verification Status" type radiobox , mapped-to ViVA.data.VivaRequirement.verificationStatus
Field rfdRfwField label: "RFD/RFW" type text , mapped-to ViVA.data.VivaRequirement.rfdRfw
}
Container ViVA_VivaReq_AssociationGroup {
label: "Verification Method Associated"
Field rodAss label: "ROD" type multipleChoiceList , mapped-to ViVA.data.VivaRequirement.rodAss
Field anAss label: "Analysis" type multipleChoiceList , mapped-to ViVA.data.VivaRequirement.anAss
Field insAss label: "Inspection" type multipleChoiceList , mapped-to ViVA.data.VivaRequirement.inAss
Field tesAss label: "Test" type multipleChoiceList , mapped-to ViVA.data.VivaRequirement.teAss
}
Container NewName_Requirement_ConnectionGroup{
label: "Connections"
Field pcRefAssociation label: "Component Connection" type multipleChoiceList , mapped-to ViVA.data.VivaRequirement.pcRef
Field reqReqAssAssociation label: "Requirement Children" type multipleChoiceList , mapped-to ViVA.data.VivaRequirement.reqReqAss
}
}
}
UI ViVA_ROD {
label: "ROD Details"
Container ViVA_ROD_Section {
Container ViVA_ROD_AttributeGroup {
label: "ROD Attributes"
Field stageField label: "Verification Stage" type radiobox , mapped-to ViVA.data.ROD.stage
Field complianceField label: "Compliance Status" type radiobox , mapped-to ViVA.data.ROD.compliance
Field complianceCommentField label: "Compliance Comment" type text , mapped-to ViVA.data.ROD.complianceComment
Field justificationField1 label: "Justification" type text , mapped-to ViVA.data.ROD.justification
}
}
}
UI ViVA_Analysis {
label: "Analysis Details"
```

```
Container ViVA_Analysis_Section {
Container ViVA_Analysis_AttributeGroup {
label: "Analysis Attributes"
Field stageField2 label: "Verification Stage" type radiobox , mapped-to ViVA.data.Analysis.stage
Field complianceField3 label: "Compliance Status" type radiobox , mapped-to ViVA.data.Analysis.compliance
Field complianceCommentField4 label: "Compliance Comment" type text , mapped-to ViVA.data.Analysis.complianceComment
Field justificationField5 label: "Justification" type text , mapped-to ViVA.data.Analysis.justification
}
}
}
UI ViVA_Inspection {
label: "Inspection Details"
Container ViVA_Inspection_Section {
Container ViVA_Inspection_AttributeGroup {
label: "Inspection Attributes"
Field stageField6 label: "Verification Stage" type radiobox , mapped-to ViVA.data.Inspection.stage
Field complianceField7 label: "Compliance Status" type radiobox , mapped-to ViVA.data.Inspection.compliance
Field complianceCommentField8 label: "Compliance Comment" type text , mapped-to ViVA.data.Inspection.complianceComment
Field justificationField9 label: "Justification" type text , mapped-to ViVA.data.Inspection.justification
}
}
}
UI ViVA_Test {
label: "Test Details"
Container ViVA_Test_Section {
Container ViVA_Test_AttributeGroup {
label: "Test Attributes"
Field stageField10 label: "Verification Stage" type radiobox , mapped-to ViVA.data.Test.stage
Field complianceField11 label: "Compliance Status" type radiobox , mapped-to ViVA.data.Test.compliance
Field complianceCommentField12 label: "Compliance Comment" type text , mapped-to ViVA.data.Test.complianceComment
Field justificationField13 label: "Justification" type text , mapped-to ViVA.data.Test.justification
Field specificationField label: "Test Specification" type text , mapped-to ViVA.data.Test.specification
}
}
}
}
```



You can see the 3 different containers, and the "empty" slot still written as empty (see the Java file to understand how to modify this)

Inside one of the verification methods:



Diagram

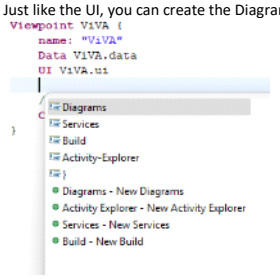
02 October 2025 12:00

Phase 3: Diagrams

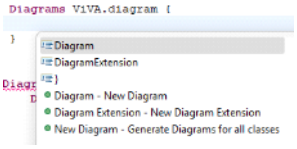
DISCLAIMER: Speaking with Obeo employees, I discovered that this method to create Diagrams and Diagram Extensions is now deprecated. Instead, they use the .odiagram interface. I will also use it later, but starting from zero from that one caused me some troubles, so I prefer to use this way to obtain a stronger base and then modify it there.

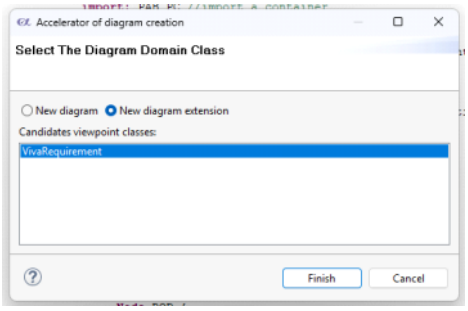
Now, the last important step is to customise the diagram and the visual representation of the viewpoint.

Just like the UI, you can create the Diagram page in the .spec.vptext and then modify it



Inside Diagram, select New Diagram and then New Diagram Extension





Now, just like for the UI, you have a base that you can modify. Since this is a Diagram Extension, it means that you're using an already existing diagram from Capella, so you need to specify both the extended diagram and then you need to import a container. The first diagram extension I creat ed was in the Physical level so we'll start from there. This is how I completed the info:

```
Diagrams ViVA.diagram {
    DiagramExtension "diagramExtension_physical" {
        extended-diagram: PhysicalArchitectureBlank //extended diagram

        Mapping {
            Container VivaReqContainer {
                import: PAB_PC //import a container
```

Something really important here is that if you leave the code like this, the viewpoint will actually already work! BUT with t hese settings, you can only create requirement elements inside the container you imported. I wanted to have requirements out of the element and then connect the m.
To do this, you need to remove the requirement node from the container section, so that it exists as a standalone container. Inside it, we can place the other nodes (since we want them to be contained within the main requirement node). If you prefer to keep the nodes separate, simply remov e them from the Contains section (see the result of the code below).

Main section you can intervene:

- Representation -> both labels and Style
  - The labels can be either static, with a string (like "ROD"), or they can be one of the attributes you declared (like `content: ViVA.data.VivaRequirement.identification`), or you can create customised Java codes to make it dynamic
  - You can also have more than one representation, with conditions.

To be able to connect your requirement to other elements, you need an Edge element, which will be our arrows. The first one w ill connect 2 requirements, to establish a sort of traceability or parent-child relations:

```
Edge Requirement_reqReqAss{
association-context: ViVA.data.VivaRequirement.reqReqAss
source: VivaReqContainer
target: VivaReqContainer
Representation {
Style {
end-decorator: InputArrow
color: black
}
}
}
```

Really important to set the right association-context (here you see the association we declared in the Data file), and the Source and Target of our edge.

Then, to be able to create an edge between my requirement element and a general Capella element, I imported a container as a "placeholder" and then connected the edge to it. Again: if you use the .odesign, this is not necessary.

```
Container ComponentPlaceholder{
import: PAB_PC
}
```

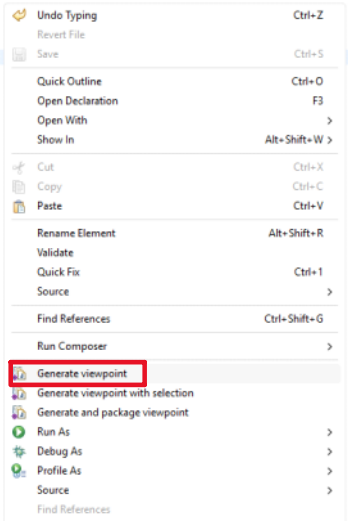
```
Edge Requirement_compEdge{
association-context: ViVA.data.VivaRequirement.pcRef
source: VivaReqContainer
target: ComponentPlaceholder
Representation {
Style {
line-style: dash
end-decorator: InputArrow
color: gray
}
}
}
```

Lastly, you need to declare the actions to be able to actually use the viewpoint. So we need Create, Drop and Delete actions, for all our elements (except for the placeholder container)

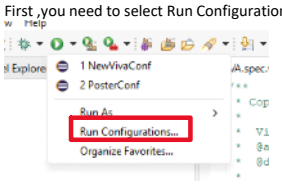
```
Actions {
/* VivaReqContainer Actions*/
Create VivaReq_CT_5{
label: "Requirement" action-for: VivaReqContainer
}
Drop VivaReq_DR_5{
action-for: VivaReqContainer
}
Delete VivaReq_DL_5{
action-for: VivaReqContainer
}
/*ROD Actions */
Create ROD_CT_7{
label: "ROD" action-for: VivaReqContainer.ROD
}
Drop ROD_DT_7{
action-for: VivaReqContainer.ROD
}
/*Analysis Actions */
Create Analysis_CT_7{
label: "Analysis" action-for: VivaReqContainer.Analysis
}
Drop Analysis_DT_7{
action-for: VivaReqContainer.Analysis
}
/*Inspection Actions */
Create Inspection_CT_7{
label: "Inspection" action-for: VivaReqContainer.Inspection
}
Drop Inspection_DT_7{
action-for: VivaReqContainer.Inspection
}
/*Test Actions */
Create Test_CT_7{
label: "Test" action-for: VivaReqContainer.Test
}
Drop Test_DT_7{
action-for: VivaReqContainer.Test
}
Create reqedge {
label: "Requirement Children" action-for: Requirement_reqReqAss
}
Drop reqedgedrop {
action-for: Requirement_reqReqAss
}
Create compedge {
label: "Component Connection" action-for: Requirement_compEdge
}
Drop compedgedrop {
action-for: Requirement_compEdge
}
}
```

So now, to obtain your viewpoint first, you need to go to the main page and generate it (right click):

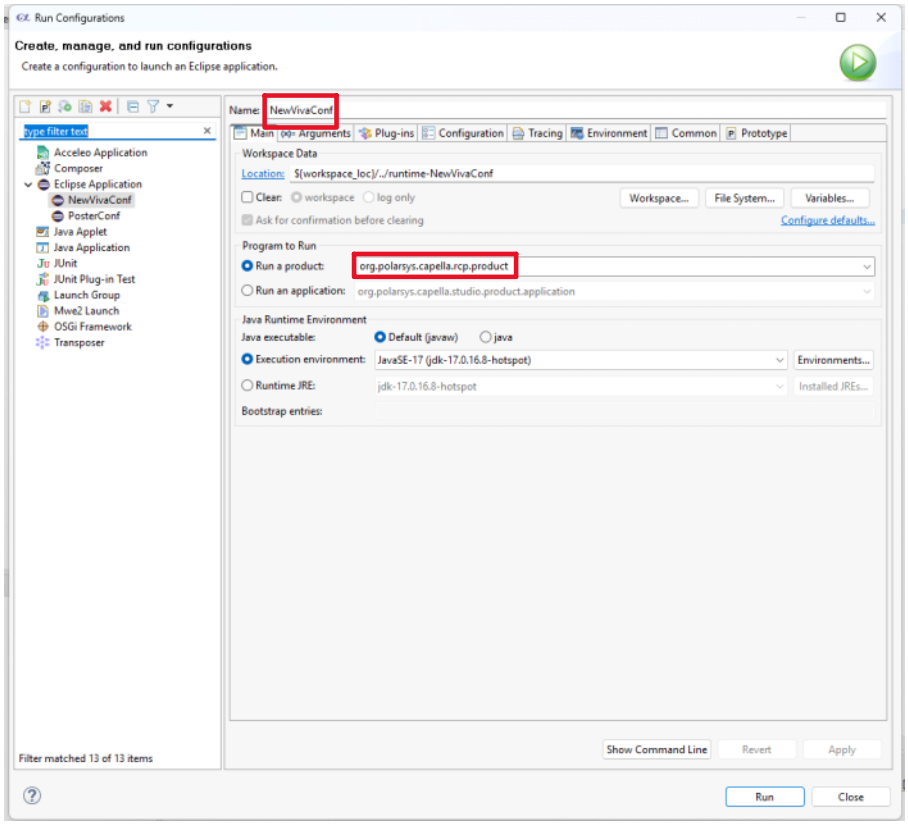
```
 *
 */
Viewpoint ViVA {
    name: "ViVA"
    Data ViVA.data
    UI ViVA.ui
    Diagrams ViVA.diagram
    Configuration ViVA.conf
}
```



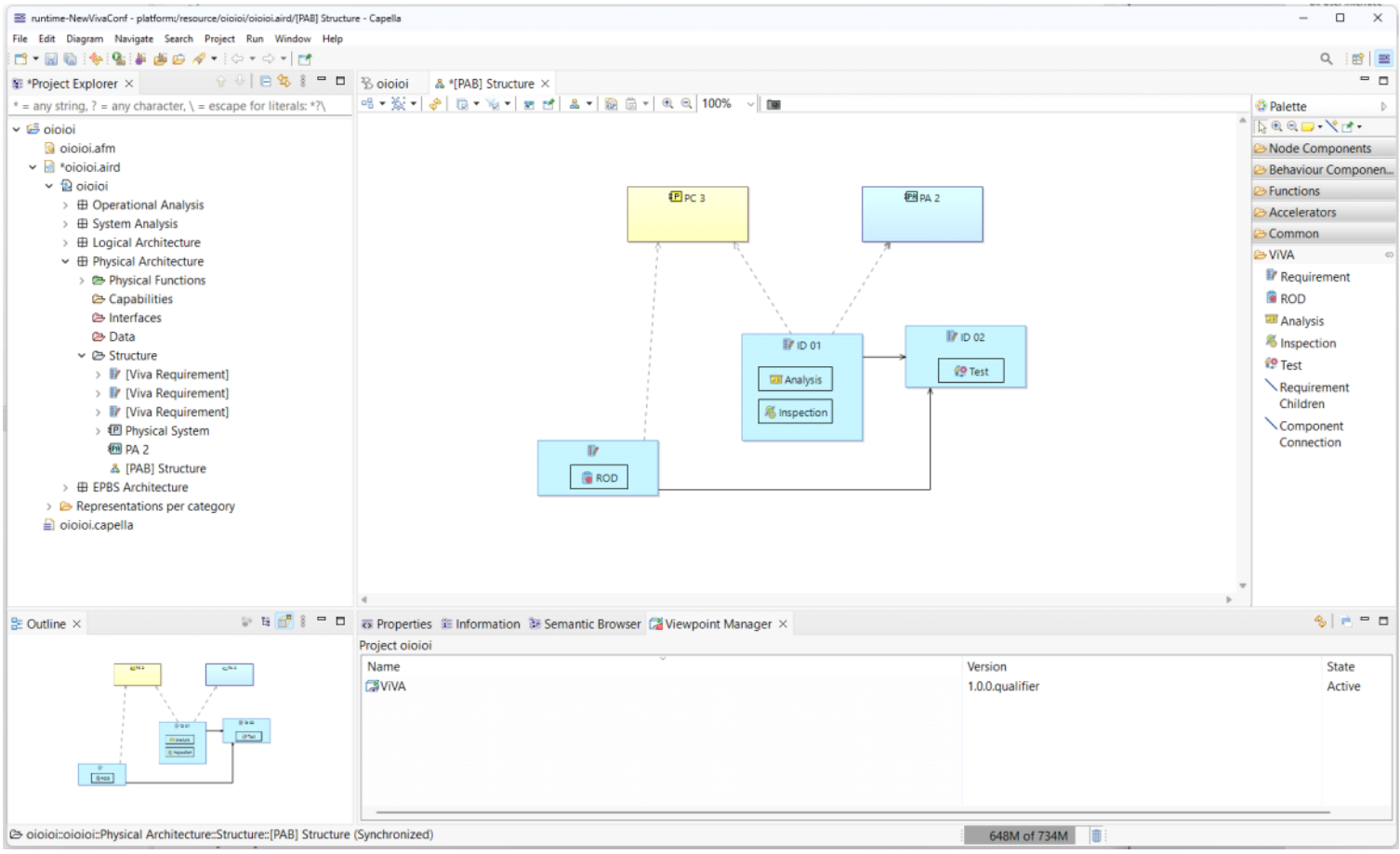
After 1 life, more or less, when the generation is completed, you can Run.
First ,you need to select Run Configuration to create a new configuration (I already have two):



Then, inside, you need to give it a name, and select the right "Run a product". You want to select the exact same one you see here. Then you click Run, and after 3 years, a Capella window will open, and you can create a new Capella project. Now you need to follow the usual steps to operat e your viewpoint (Window -> Show View -> Viewpoint Manager, then you need to reference your viewpoint. Then you need to activate your layer, in this case only in th e PAB diagram will be possible).



After all these steps, you will end up with something like this:



As it is, it's a simple functioning viewpoint. Here are some problems:

- Behaviour PC are not visible-> this will be fixed using the .odesign file
- All the Requirements will display only the ID we assign them, not the name or the text, but we can improve it with a simple Java code
- As you can see on the left, all the requirements, even if they have names and IDs, will display the Class's name. We can create a code that automatically copies the ID of our requirement in the Name field generated by Capella so that it will be visible in the Project explorer
- We need to create more diagram extensions for all the Capella levels
- We can make Java code to make the visualisation of the viewpoint more dynamic

I suggest first finishing creating all the codes you want, then copy-paste for all the diagram extensions you need, and then, when you're absolutely sure you will not need to change anything anymore, pass to fix stuff in the .odesign. Trust me, you will understand better later.

The result now will look like this:

```
Diagrams ViVA.diagram {
DiagramExtension "diagramExtension_physical" {
extended-diagram: PhysicalArchitectureBlank //extended diagram
Mapping {
Container VivaRequirementContainer {
import: PAB_PC //import a container
domain-context: ViVA.data.VivaRequirement
provided-by association external emde.ExtensibleElement.ownedExtensions
Representation {
Label {
content: "VivaRequirement"
police: black
}
Style {
FlatStyle {
border: blue
background: light_blue
foreground: white
}
}
}
Contains {
Node ROD{
domain-context: ViVA.data.ROD
provided-by association ViVA.data.VivaRequirement.rodAss
Representation {
Label {
content: "ROD"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
} // close ROD
Node Analysis{
domain-context: ViVA.data.Analysis
provided-by association ViVA.data.VivaRequirement.anAss
Representation {
Label {
content: "Analysis"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
} //close analysis
Node Inspection{
domain-context: ViVA.data.Inspection
provided-by association ViVA.data.VivaRequirement.inAss
Representation {
Label {
content: "Inspection"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
} //close inspection
Node Test{
domain-context: ViVA.data.Test
provided-by association ViVA.data.VivaRequirement.teAss
Representation {
Label {
content: "Test"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
} //close test
} //close contains
} //close container
Container ComponentPlaceholder{
import: PAB_PC
}
Edge VivaRequirement_reqReqAss {
association-context: ViVA.data.VivaRequirement.reqReqAss
source: VivaRequirementContainer
target: VivaRequirementContainer
Representation {
Style {
end-decorator: InputArrow
color: black
}
}
}
Edge VivaRequirement_compAss {
association-context: ViVA.data.VivaRequirement.pcRef
source: VivaRequirementContainer
target: ComponentPlaceholder
Representation {
Style {
line-style: dash
end-decorator: InputArrow
color: gray
}
}
}
} //close mapping
Actions {
/* VivaRequirementContainer Actions*/
Create VivaRequirement_CT_19{
label: "Requirement" action-for: VivaRequirementContainer
}
Drop VivaRequirement_DR_19{
action-for: VivaRequirementContainer
}
Delete VivaRequirement_DL_19{
action-for: VivaRequirementContainer
}
/*ROD Actions */
Create ROD_CT_21{
```

```
label: "ROD" action-for: VivaRequirementContainer.ROD
}
Drop ROD_DT_21{
action-for: VivaRequirementContainer.ROD
}
/*Analysis Actions */
Create Analysis_CT_22{
label: "Analysis" action-for: VivaRequirementContainer.Analysis
}
Drop Analysis_DT_22{
action-for: VivaRequirementContainer.Analysis
}
/*Inspection Actions */
Create Inspection_CT_23{
label: "Inspection" action-for: VivaRequirementContainer.Inspection
}
Drop Inspection_DT_23{
action-for: VivaRequirementContainer.Inspection
}
/*Test Actions */
Create Test_CT_24{
label: "Test" action-for: VivaRequirementContainer.Test
}
Drop Test_DT_24{
action-for: VivaRequirementContainer.Test
}
/* Requirement Edge Actions */
Create reqedge {
label: "Requirement Children" action-for: VivaRequirement_reqReqAss
}
Drop reqedgedrop {
action-for: VivaRequirement_reqReqAss
}
ReconnectEdge reqedgerecon {
action-for: VivaRequirement_reqReqAss
}

/* Component Edge Actions */
Create compedge {
label: "Component Connection" action-for: VivaRequirement_compAss
}
Drop compedgedrop {
action-for: VivaRequirement_compAss
}
ReconnectEdge compedgerecon{
action-for:VivaRequirement_compAss
}
} //close actions
} // close DiagramExtension
} // close Diagrams
```

Java codes

02 October 2025 12:00

Phase 4: Java Codes

Welcome.
So now, the funny part.
This perfect software runs on a domain-specific language (DSL) provided by the Viewpoint Development Framework (VPD/VPDSL) (the one we used in the previous pages), but can also implement Services using Java code.

I want to create 4 main codes:

- One that can change the Representation (eg: the colour) of our element, when something inside is selected
- One that can both display the requirement information in the format ID - Name : Text, and at the same time copy the ID in the "Name" space that Capella automatically creates, so that it can be visible in the Project Explorer

The first thing to do is to create the code, so we need to go back to the diagram file. Here I want a code that makes my Requ irement turns grey when they're closed. So I create another representation, under the one I already have, I put the Style I want to obtain and the condition:

```
Representation{
condition: Java ReqClose
Label {
content: ViVA.data.VivaRequirement.identification
police: black
}
Style {
FlatStyle {
border: black
background: light_gray
foreground: black
}
}
}
```

So now, when the condition will be satisfied, my element will turn grey. We need now to save and generate the viewpoint. Insi de the Project Explorer, we will find the place where the code is recalled:
Org.polarys.capella.vp.yourname.**design** -> src -> org.polarys.capella.vp.yourname.**design.service.nodes** -> namecontainer_Service.java

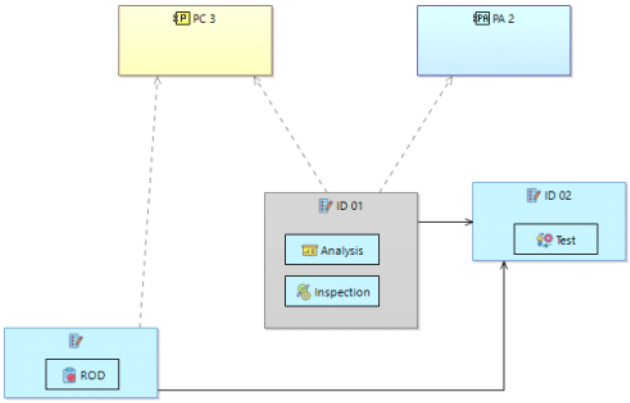
We will find something like this:

```
/**
 * <!-- begin-user-doc -->
 * This class is an implementation of the Sirius JavaExtension '<em><b>[org.polarsys.capella.vp.viva.design.service.nodes.VivaReqContainer_Service] </b>
</em>'.
 * <!-- end-user-doc -->
 * <p>
 * </p>
 *
 * @generated
 */

public class VivaReqContainer_Service {
	/**
	* <!-- begin-user-doc -->
	* <!-- end-user-doc -->
	* @param eObject : the current semantic object
	* @param view : the current view
	* @param container : the semantic container of the current object
	* @generated
	*/
	public boolean ReqClose(EObject eObject, EObject view, EObject container) {
		// TODO Auto-generated method stub
		// Ensure that you remove @generated or mark it @generated NOT
		throw new UnsupportedOperationException();
	}
}
```

The first thing to do is to put @generate NOT over the service method ReqClose, in this way it will not be erased every time you generate your viewpoint. Then, inside the code we put our check. My code is this:

```
public class VivaReqContainer_Service {
	/**
	* <!-- begin-user-doc -->
	* <!-- end-user-doc -->
	* @param eObject : the current semantic object
	* @param view : the current view
	* @param container : the semantic container of the current object
	* @generated NOT
	*/
	public boolean ReqClose(EObject eObject, EObject view, EObject container) {
		if (eObject == null)
			return false;

		// Veirfy "verificationStatus" exist
		if (eObject.eClass().getEStructuralFeature("verificationStatus") != null) {
			Object value = eObject.eGet(eObject.eClass().getEStructuralFeature("verificationStatus"));
			if (value != null) {
				try {
					java.lang.reflect.Method getLiteralMethod = value.getClass().getMethod("getLiteral");
					Object literalValue = getLiteralMethod.invoke(value);
					return "Closed".equals(literalValue);
				} catch (Exception e) {
					return "Closed".equals(value.toString());
				}
			}
		}
		return false;

	}
}
```

We can see that it works by running our code:



A similar code can be created for the other classes. In my case, I want my Verification method Classes to turn Red when they' re Not compliant, Green when they're compliant, and blue when they're not Applicable.
So my diagram code, for the ROD node, will look like this:

```
Node ROD {
domain-context: ViVA.data.ROD
provided-by association ViVA.data.VivaRequirement.rodAss
Representation {
Label {
content: "ROD"
police: black
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
Representation {
condition: Java MethodCompliant
Label {
content: "ROD"
police: black
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: light_green
form: Square
}
}
}
Representation {
condition: Java MethodNonCompliant
Label {
content: "ROD Non Compliant"
police: black
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: light_red
form: Square
}
}
}
Representation {
condition: Java MethodNA
Label {
content: "ROD Non Applicable"
police: white
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: dark_blue
form: Square
}
}
}
} // close rod
```

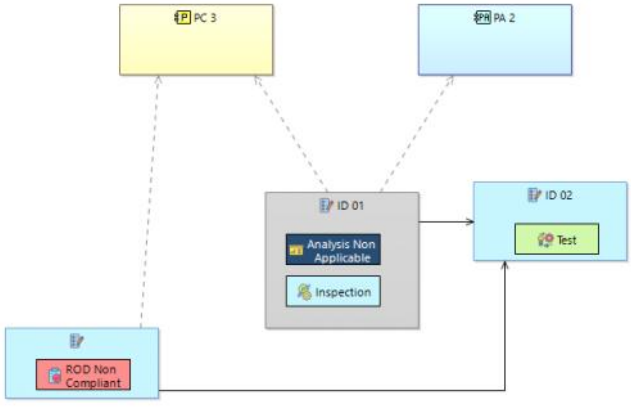
Yes, you need a "basic" representation if none of those is selected (the empty option)

And then, after I generate the viewpoint, I will find the Java code where the previous one was. My final code is this one:

```
public class ROD_Service {
	/**
	* <!-- begin-user-doc -->
	* <!-- end-user-doc -->
	* @param eObject : the current semantic object
	* @param view : the current view
	* @param container : the semantic container of the current object
	* @generated NOT
	*/
	public boolean MethodCompliant(EObject eObject, EObject view, EObject container) {
		if (eObject == null)
			return false;

		if (eObject.eClass().getEStructuralFeature("compliance") != null) {
			Object value = eObject.eGet(eObject.eClass().getEStructuralFeature("compliance"));
			if (value != null) {
```

```
                        try {
                              java.lang.reflect.Method getLiteralMethod = value.getClass().getMethod("getLiteral");
                              Object literalValue = getLiteralMethod.invoke(value);
                              return "Compliance".equals(literalValue);
                        } catch (Exception e) {
                              return "Compliance".equals(value.toString());
                        }
                  }
            }
            return false;
      }

      /**
      * <!-- begin-user-doc -->
      * <!-- end-user-doc -->
      * @param eObject : the current semantic object
      * @param view : the current view
      * @param container : the semantic container of the current object
      * @generated NOT
      */
      public boolean MethodNonCompliant(EObject eObject, EObject view, EObject container) {
            if (eObject == null)
                  return false;

            if (eObject.eClass().getEStructuralFeature("compliance") != null) {
                  Object value = eObject.eGet(eObject.eClass().getEStructuralFeature("compliance"));
                  if (value != null) {

                        try {
                              java.lang.reflect.Method getLiteralMethod = value.getClass().getMethod("getLiteral");
                              Object literalValue = getLiteralMethod.invoke(value);
                              if ("Non Compliance".equals(literalValue)) {
                                    return true;
                              }
                        } catch (Exception e) {

                              if ("Non Compliance".equals(value.toString())) {
                                    return true;
                              }
                        }
                  }
            }
            return false;

      }

      /**
      * <!-- begin-user-doc -->
      * <!-- end-user-doc -->
      * @param eObject : the current semantic object
      * @param view : the current view
      * @param container : the semantic container of the current object
      * @generated NOT
      */
      public boolean MethodNA(EObject eObject, EObject view, EObject container) {
            if (eObject == null)
                  return false;

            if (eObject.eClass().getEStructuralFeature("compliance") != null) {
                  Object value = eObject.eGet(eObject.eClass().getEStructuralFeature("compliance"));
                  if (value != null) {
                        try {
                              java.lang.reflect.Method getLiteralMethod = value.getClass().getMethod("getLiteral");
                              Object literalValue = getLiteralMethod.invoke(value);
                              return "Non Applicable".equals(literalValue);
                        } catch (Exception e) {
                              return "Non Applicable".equals(value.toString());
                        }
                  }
            }
            return false;
      }
}
```

Then you just need to repeat this step for all the verification methods. The result will look like this:



Now the second type of code we need to create is the one that will customize our visualization of the id and at the same time will copy our id in the Name section of Capella. In the diagram page, let's change the label settings to this:

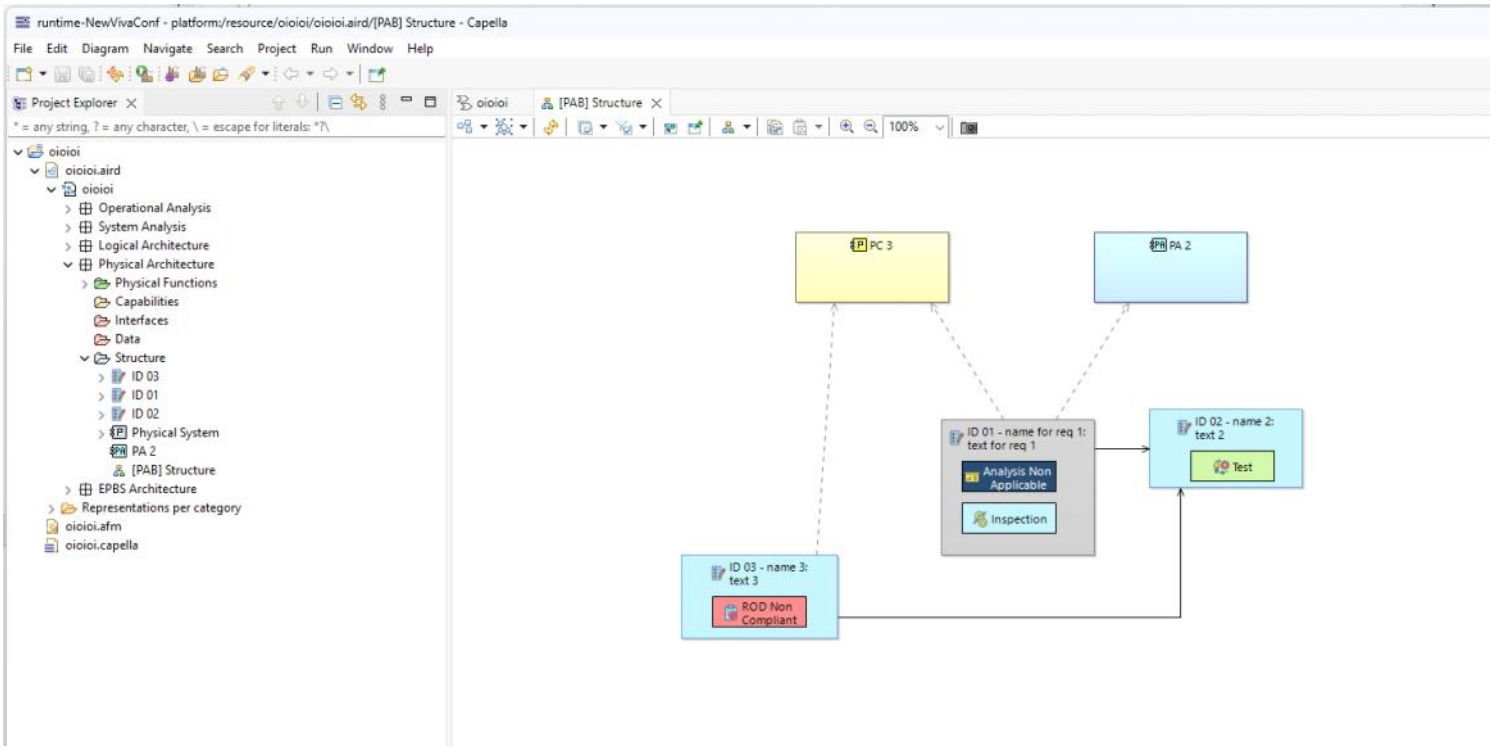
```
Container VivaReqContainer {
import: PAB_PC //import a container
domain-context: ViVA.data.VivaRequirement
provided-by association external emde.ExtensibleElement.ownedExtensions
Representation{
Label {
content: Java Reqlabel
police: black
}
Style {
FlatStyle {
border: blue
background: light_blue
foreground: light_blue
```

```
}
}
}
Representation{
condition: Java ReqClose
Label {
content: Java Reqlabel
police: black
}
Style {
FlatStyle {
border: black
background: light_gray
foreground: black
}
}
}
```

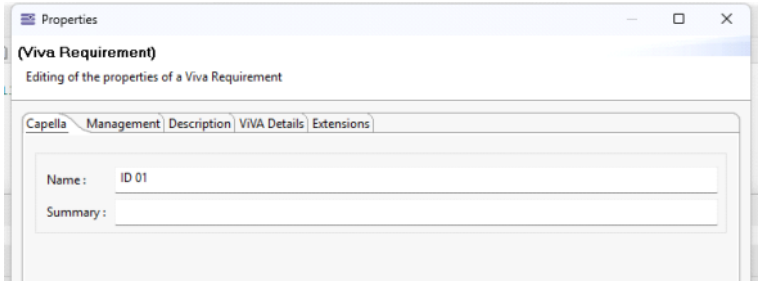
Then, generate the viewpoint, and we can write our code in the same place we wrote the ReqClose one. My code looks like this:

```
/**
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @param eObject : the current semantic object
	 * @param diagram : the current DSemanticdiagram
	 * @param view : the current View for witch the label is calculated
	 * @generated NOT
	 */
	public String Reqlabel(EObject eObject, DDiagram diagram, DDiagramElement view) {
		if (eObject == null) {
		return "";
	  }

	  // Read identification, nameReq and text
	  Object idVal = eObject.eGet(eObject.eClass().getEStructuralFeature("reqIdentification"));
	  Object nameVal = eObject.eGet(eObject.eClass().getEStructuralFeature("nameReq"));
	  Object textVal = eObject.eGet(eObject.eClass().getEStructuralFeature("text"));

	  String id = idVal != null ? idVal.toString() : "";
	  String name = nameVal != null ? nameVal.toString() : "";
	  String text = textVal != null ? textVal.toString() : "";

	  // Copy id into the "name" feature (if it exists)
	  if (!id.isEmpty() && eObject.eClass().getEStructuralFeature("name") != null) {
		  eObject.eSet(eObject.eClass().getEStructuralFeature("name"), id);
	  }

	  // Build label in the format: ID - Name: Text
	  StringBuilder sb = new StringBuilder();

	  if (!id.isEmpty()) {
		  sb.append(id);
	  }

	  if (!name.isEmpty()) {
		  if (sb.length() > 0) {
			  sb.append(" - ");
		  }
		  sb.append(name).append(":");
	  }

	  if (!text.isEmpty()) {
		  // Start a new line after ':'
		  sb.append("\n").append(text);
	  }

	  return sb.toString().trim();
	}
```

This is the result when running our code:



As you can see, both the display is what we wanted and in the Project Explorer we can see the id of our requirement. This is because the id is automatically copied in the Name section:



Now, we still have to fix the display of the "empty" enumerations, we actually want the field to be empty. To do this, we nee d to find the right Java code responsible for this representation. This can be found in:
Org.polarys.capella.vp.yourname.**model** -> src -> org.polarys.capella.vp.yourname.**yourname** -> enumerationName.java

Fo rexample, for the ECSS Division enumeration, I find something like this:

```
package org.polarsys.capella.vp.viva.ViVA;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;

import org.eclipse.emf.common.util.Enumerator;

/**
 * <!-- begin-user-doc -->
 * A representation of the literals of the enumeration '<em><b>ECS Sdivision</b></em>',
 * and utility methods for working with them.
 * <!-- end-user-doc -->
 * @see org.polarsys.capella.vp.viva.ViVA.ViVAPackage#getECSSdivision()
 * @model
 * @generated
 */
public enum ECSSdivision implements Enumerator {
	/**
	 * The '<em><b>Empty</b></em>' literal object.
	 * <!-- begin-user-doc -->
	 * <!-- end-user-doc -->
	 * @see #EMPTY_VALUE
	 * @generated
	 * @ordered
	 */
	EMPTY(0, "empty", "empty"), //$NON-NLS-1$ //$NON-NLS-2$
```

As you can see, you just need to replace the word Empty with an empty string (do not forget to put @generated NOT or it will not work). Repeat as you need.

Another side note: if you just modify the Java code (or the .odesign), you do not need to Generate every time you change some thing, just Run it. But if you change the .vptext you will need to generate every time.

Now you just need to copy-paste the final diagram for the other levels you want to extend. Pay attention: when you copy-paste to create also other diagram extensions for other levels, you need to change 4 lines:

- The name of the diagram extension
- The reference to the right extended diagram
- The import in the requirement container
- The import in the PlaceholderContainer

My final .diagram.vptext :

```
Diagrams ViVA.diagram {
DiagramExtension "diagramExtension_physical" {
extended-diagram: PhysicalArchitectureBlank //extended diagram
Mapping {
Container VivaRequirementContainer {
import: PAB_PC //import a container
domain-context: ViVA.data.VivaRequirement
provided-by association external emde.ExtensibleElement.ownedExtensions
Representation {
```

```
Label {
content: Java ReqLabel
police: black
}
Style {
FlatStyle {
border: blue
background: light_blue
foreground: light_blue
}
}
}
Representation {
condition: Java ReqClose
Label {
content: Java ReqLabel
police: black
}
Style {
FlatStyle {
border: blue
background: light_gray
foreground: black
}
}
}
Contains {
Node ROD{
domain-context: ViVA.data.ROD
provided-by association ViVA.data.VivaRequirement.rodAss
Representation {
Label {
content: "ROD"
police: black
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
Representation {
condition: Java MethodCompliant
Label {
content: "ROD"
police: black
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: light_green
form: Square
}
}
}
Representation {
condition: Java MethodNonCompliant
Label {
content: "ROD Non Compliant"
police: black
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: light_red
form: Square
}
}
}
Representation {
condition: Java MethodNA
Label {
content: "ROD Non Applicable"
police: white
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: dark_blue
form: Square
}
}
}
} // close ROD
Node Analysis{
domain-context: ViVA.data.Analysis
provided-by association ViVA.data.VivaRequirement.anAss
Representation {
Label {
content: "Analysis"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
Representation {
condition: Java MethodCompliant
Label {
content: "Analysis"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_green
form: Square
}
}
}
Representation {
condition: Java MethodNonCompliant
```

```
Label {
content: "Analysis Non Compliant"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_red
form: Square
}
}
}
Representation {
condition: Java MethodNA
Label {
content: "Analysis Non Applicable"
police: white
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: dark_blue
form: Square
}
}
}
} //close analysis
Node Inspection{
domain-context: ViVA.data.Inspection
provided-by association ViVA.data.VivaRequirement.inAss
Representation {
Label {
content: "Inspection"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
Representation {
condition: Java MethodCompliant
Label {
content: "Inspection"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_green
form: Square
}
}
}
Representation {
condition: Java MethodNonCompliant
Label {
content: "Inspection Non Compliant"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_red
form: Square
}
}
}
Representation {
condition: Java MethodNA
Label {
content: "Inspection Non Applicable"
police: white
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: dark_blue
form: Square
}
}
}
} //close inspection
Node Test{
domain-context: ViVA.data.Test
provided-by association ViVA.data.VivaRequirement.teAss
Representation {
Label {
content: "Test"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
Representation {
condition: Java MethodCompliant
Label {
content: "Test"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_green
form: Square
}
```

```
}
}
Representation {
condition: Java MethodNonCompliant
Label {
content: "Test Non Compliant"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_red
form: Square
}
}
}
Representation {
condition: Java MethodNA
Label {
content: "Test Non Applicable"
police: white
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: dark_blue
form: Square
}
}
}
} //close test
} //close contains
} //close container
Container ComponentPlaceholder{
import: PAB_PC
}
Edge VivaRequirement_reqReqAss {
association-context: ViVA.data.VivaRequirement.reqReqAss
source: VivaRequirementContainer
target: VivaRequirementContainer
Representation {
Style {
end-decorator: InputArrow
color: black
}
}
}
Edge VivaRequirement_compAss {
association-context: ViVA.data.VivaRequirement.pcRef
source: VivaRequirementContainer
target: ComponentPlaceholder
Representation {
Style {
line-style: dash
end-decorator: InputArrow
color: gray
}
}
}
} //close mapping
Actions {
/* VivaRequirementContainer Actions*/
Create VivaRequirement_CT_19{
label: "Requirement" action-for: VivaRequirementContainer
}
Drop VivaRequirement_DR_19{
action-for: VivaRequirementContainer
}
Delete VivaRequirement_DL_19{
action-for: VivaRequirementContainer
}
/*ROD Actions */
Create ROD_CT_21{
label: "ROD" action-for: VivaRequirementContainer.ROD
}
Drop ROD_DT_21{
action-for: VivaRequirementContainer.ROD
}
/*Analysis Actions */
Create Analysis_CT_22{
label: "Analysis" action-for: VivaRequirementContainer.Analysis
}
Drop Analysis_DT_22{
action-for: VivaRequirementContainer.Analysis
}
/*Inspection Actions */
Create Inspection_CT_23{
label: "Inspection" action-for: VivaRequirementContainer.Inspection
}
Drop Inspection_DT_23{
action-for: VivaRequirementContainer.Inspection
}
/*Test Actions */
Create Test_CT_24{
label: "Test" action-for: VivaRequirementContainer.Test
}
Drop Test_DT_24{
action-for: VivaRequirementContainer.Test
}
/* Requirement Edge Actions */
Create reqedge {
label: "Requirement Children" action-for: VivaRequirement_reqReqAss
}
Drop reqedgedrop {
action-for: VivaRequirement_reqReqAss
}
ReconnectEdge reqedgerecon {
action-for: VivaRequirement_reqReqAss
}

/* Component Edge Actions */
Create compedge {
label: "Component Connection" action-for: VivaRequirement_compAss
}
Drop compedgedrop {
action-for: VivaRequirement_compAss
}
ReconnectEdge compedgerecon{
action-for:VivaRequirement_compAss
}
} //close actions
} // close DiagramExtension
DiagramExtension "diagramExtension_logical" {
extended-diagram: LogicalArchitectureBlank //extended diagram
Mapping {
Container VivaRequirementContainer {
```

```
import: LABLogicalComponent //import a container
domain-context: ViVA.data.VivaRequirement
provided-by association external emde.ExtensibleElement.ownedExtensions
Representation {
Label {
content: Java ReqLabel
police: black
}
Style {
FlatStyle {
border: blue
background: light_blue
foreground: light_blue
}
}
}
Representation {
condition: Java ReqClose
Label {
content: Java ReqLabel
police: black
}
Style {
FlatStyle {
border: blue
background: light_gray
foreground: black
}
}
}
Contains {
Node ROD{
domain-context: ViVA.data.ROD
provided-by association ViVA.data.VivaRequirement.rodAss
Representation {
Label {
content: "ROD"
police: black
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
Representation {
condition: Java MethodCompliant
Label {
content: "ROD"
police: black
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: light_green
form: Square
}
}
}
Representation {
condition: Java MethodNonCompliant
Label {
content: "ROD Non Compliant"
police: black
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: light_red
form: Square
}
}
}
Representation {
condition: Java MethodNA
Label {
content: "ROD Non Applicable"
police: white
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: dark_blue
form: Square
}
}
}
} // close ROD
Node Analysis{
domain-context: ViVA.data.Analysis
provided-by association ViVA.data.VivaRequirement.anAss
Representation {
Label {
content: "Analysis"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
Representation {
condition: Java MethodCompliant
Label {
content: "Analysis"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_green
form: Square
}
```

```
}
}
Representation {
condition: Java MethodNonCompliant
Label {
content: "Analysis Non Compliant"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_red
form: Square
}
}
}
Representation {
condition: Java MethodNA
Label {
content: "Analysis Non Applicable"
police: white
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: dark_blue
form: Square
}
}
}
} //close analysis
Node Inspection{
domain-context: ViVA.data.Inspection
provided-by association ViVA.data.VivaRequirement.inAss
Representation {
Label {
content: "Inspection"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
Representation {
condition: Java MethodCompliant
Label {
content: "Inspection"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_green
form: Square
}
}
}
Representation {
condition: Java MethodNonCompliant
Label {
content: "Inspection Non Compliant"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_red
form: Square
}
}
}
Representation {
condition: Java MethodNA
Label {
content: "Inspection Non Applicable"
police: white
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: dark_blue
form: Square
}
}
}
} //close inspection
Node Test{
domain-context: ViVA.data.Test
provided-by association ViVA.data.VivaRequirement.teAss
Representation {
Label {
content: "Test"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
Representation {
condition: Java MethodCompliant
Label {
content: "Test"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
```

```
border-color: black
background: light_green
form: Square
}
}
}
Representation {
condition: Java MethodNonCompliant
Label {
content: "Test Non Compliant"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_red
form: Square
}
}
}
Representation {
condition: Java MethodNA
Label {
content: "Test Non Applicable"
police: white
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: dark_blue
form: Square
}
}
}
} //close test
} //close contains
} //close container
Container ComponentPlaceholder{
import: LABLogicalComponent
}
Edge VivaRequirement_reqReqAss {
association-context: ViVA.data.VivaRequirement.reqReqAss
source: VivaRequirementContainer
target: VivaRequirementContainer
Representation {
Style {
end-decorator: InputArrow
color: black
}
}
}
Edge VivaRequirement_compAss {
association-context: ViVA.data.VivaRequirement.pcRef
source: VivaRequirementContainer
target: ComponentPlaceholder
Representation {
Style {
line-style: dash
end-decorator: InputArrow
color: gray
}
}
}
} //close mapping
Actions {
/* VivaRequirementContainer Actions*/
Create VivaRequirement_CT_19{
label: "Requirement" action-for: VivaRequirementContainer
}
Drop VivaRequirement_DR_19{
action-for: VivaRequirementContainer
}
Delete VivaRequirement_DL_19{
action-for: VivaRequirementContainer
}
/*ROD Actions */
Create ROD_CT_21{
label: "ROD" action-for: VivaRequirementContainer.ROD
}
Drop ROD_DT_21{
action-for: VivaRequirementContainer.ROD
}
/*Analysis Actions */
Create Analysis_CT_22{
label: "Analysis" action-for: VivaRequirementContainer.Analysis
}
Drop Analysis_DT_22{
action-for: VivaRequirementContainer.Analysis
}
/*Inspection Actions */
Create Inspection_CT_23{
label: "Inspection" action-for: VivaRequirementContainer.Inspection
}
Drop Inspection_DT_23{
action-for: VivaRequirementContainer.Inspection
}
/*Test Actions */
Create Test_CT_24{
label: "Test" action-for: VivaRequirementContainer.Test
}
Drop Test_DT_24{
action-for: VivaRequirementContainer.Test
}
/* Requirement Edge Actions */
Create reqedge {
label: "Requirement Children" action-for: VivaRequirement_reqReqAss
}
Drop reqedgedrop {
action-for: VivaRequirement_reqReqAss
}
ReconnectEdge reqedgerecon {
action-for: VivaRequirement_reqReqAss
}

/* Component Edge Actions */
Create compedge {
label: "Component Connection" action-for: VivaRequirement_compAss
}
Drop compedgedrop {
action-for: VivaRequirement_compAss
}
ReconnectEdge compedgerecon{
action-for:VivaRequirement_compAss
}
} //close actions
} // close DiagramExtension
```

```
DiagramExtension "diagramExtension_system" {
extended-diagram: SystemArchitectureBlank //extended diagram
Mapping {
Container VivaRequirementContainer {
import: SystemSystem //import a container
domain-context: ViVA.data.VivaRequirement
provided-by association external emde.ExtensibleElement.ownedExtensions
Representation {
Label {
content: Java ReqLabel
police: black
}
Style {
FlatStyle {
border: blue
background: light_blue
foreground: light_blue
}
}
}
Representation {
condition: Java ReqClose
Label {
content: Java ReqLabel
police: black
}
Style {
FlatStyle {
border: blue
background: light_gray
foreground: black
}
}
}
Contains {
Node ROD{
domain-context: ViVA.data.ROD
provided-by association ViVA.data.VivaRequirement.rodAss
Representation {
Label {
content: "ROD"
police: black
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
Representation {
condition: Java MethodCompliant
Label {
content: "ROD"
police: black
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: light_green
form: Square
}
}
}
Representation {
condition: Java MethodNonCompliant
Label {
content: "ROD Non Compliant"
police: black
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: light_red
form: Square
}
}
}
Representation {
condition: Java MethodNA
Label {
content: "ROD Non Applicable"
police: white
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: dark_blue
form: Square
}
}
}
} // close ROD
Node Analysis{
domain-context: ViVA.data.Analysis
provided-by association ViVA.data.VivaRequirement.anAss
Representation {
Label {
content: "Analysis"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
Representation {
condition: Java MethodCompliant
Label {
content: "Analysis"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
```

```
border-color: black
background: light_green
form: Square
}
}
}
Representation {
condition: Java MethodNonCompliant
Label {
content: "Analysis Non Compliant"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_red
form: Square
}
}
}
Representation {
condition: Java MethodNA
Label {
content: "Analysis Non Applicable"
police: white
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: dark_blue
form: Square
}
}
}
} //close analysis
Node Inspection{
domain-context: ViVA.data.Inspection
provided-by association ViVA.data.VivaRequirement.inAss
Representation {
Label {
content: "Inspection"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
Representation {
condition: Java MethodCompliant
Label {
content: "Inspection"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_green
form: Square
}
}
}
Representation {
condition: Java MethodNonCompliant
Label {
content: "Inspection Non Compliant"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_red
form: Square
}
}
}
Representation {
condition: Java MethodNA
Label {
content: "Inspection Non Applicable"
police: white
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: dark_blue
form: Square
}
}
}
} //close inspection
Node Test{
domain-context: ViVA.data.Test
provided-by association ViVA.data.VivaRequirement.teAss
Representation {
Label {
content: "Test"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
Representation {
condition: Java MethodCompliant
Label {
content: "Test"
police: black
position: node
```

```
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_green
form: Square
}
}
}
Representation {
condition: Java MethodNonCompliant
Label {
content: "Test Non Compliant"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_red
form: Square
}
}
}
Representation {
condition: Java MethodNA
Label {
content: "Test Non Applicable"
police: white
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: dark_blue
form: Square
}
}
}
} //close test
} //close contains
} //close container
Container ComponentPlaceholder{
import: SystemSystem
}
Edge VivaRequirement_reqReqAss {
association-context: ViVA.data.VivaRequirement.reqReqAss
source: VivaRequirementContainer
target: VivaRequirementContainer
Representation {
Style {
end-decorator: InputArrow
color: black
}
}
}
Edge VivaRequirement_compAss {
association-context: ViVA.data.VivaRequirement.pcRef
source: VivaRequirementContainer
target: ComponentPlaceholder
Representation {
Style {
line-style: dash
end-decorator: InputArrow
color: gray
}
}
}
} //close mapping
Actions {
/* VivaRequirementContainer Actions*/
Create VivaRequirement_CT_19{
label: "Requirement" action-for: VivaRequirementContainer
}
Drop VivaRequirement_DR_19{
action-for: VivaRequirementContainer
}
Delete VivaRequirement_DL_19{
action-for: VivaRequirementContainer
}
/*ROD Actions */
Create ROD_CT_21{
label: "ROD" action-for: VivaRequirementContainer.ROD
}
Drop ROD_DT_21{
action-for: VivaRequirementContainer.ROD
}
/*Analysis Actions */
Create Analysis_CT_22{
label: "Analysis" action-for: VivaRequirementContainer.Analysis
}
Drop Analysis_DT_22{
action-for: VivaRequirementContainer.Analysis
}
/*Inspection Actions */
Create Inspection_CT_23{
label: "Inspection" action-for: VivaRequirementContainer.Inspection
}
Drop Inspection_DT_23{
action-for: VivaRequirementContainer.Inspection
}
/*Test Actions */
Create Test_CT_24{
label: "Test" action-for: VivaRequirementContainer.Test
}
Drop Test_DT_24{
action-for: VivaRequirementContainer.Test
}
/* Requirement Edge Actions */
Create reqedge {
label: "Requirement Children" action-for: VivaRequirement_reqReqAss
}
Drop reqedgedrop {
action-for: VivaRequirement_reqReqAss
}
ReconnectEdge reqedgerecon {
action-for: VivaRequirement_reqReqAss
}

/* Component Edge Actions */
Create compedge {
label: "Component Connection" action-for: VivaRequirement_compAss
}
Drop compedgedrop {
action-for: VivaRequirement_compAss
}
ReconnectEdge compedgerecon{
```

```
action-for:VivaRequirement_compAss
}
} //close actions
} // close DiagramExtension
DiagramExtension "diagramExtension_operational" {
extended-diagram: OperationalActivityInteractionBlank //extended diagram
Mapping {
Container VivaRequirementContainer {
import: OAIBOperationalActivity //import a container
domain-context: ViVA.data.VivaRequirement
provided-by association external emde.ExtensibleElement.ownedExtensions
Representation {
Label {
content: Java ReqLabel
police: black
}
Style {
FlatStyle {
border: blue
background: light_blue
foreground: light_blue
}
}
}
Representation {
condition: Java ReqClose
Label {
content: Java ReqLabel
police: black
}
Style {
FlatStyle {
border: blue
background: light_gray
foreground: black
}
}
}
Contains {
Node ROD{
domain-context: ViVA.data.ROD
provided-by association ViVA.data.VivaRequirement.rodAss
Representation {
Label {
content: "ROD"
police: black
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
Representation {
condition: Java MethodCompliant
Label {
content: "ROD"
police: black
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: light_green
form: Square
}
}
}
Representation {
condition: Java MethodNonCompliant
Label {
content: "ROD Non Compliant"
police: black
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: light_red
form: Square
}
}
}
Representation {
condition: Java MethodNA
Label {
content: "ROD Non Applicable"
police: white
position: node
alignment: center
}
Style {
BasicStyle {
border-color: black
background: dark_blue
form: Square
}
}
}
} // close ROD
Node Analysis{
domain-context: ViVA.data.Analysis
provided-by association ViVA.data.VivaRequirement.anAss
Representation {
Label {
content: "Analysis"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
Representation {
condition: Java MethodCompliant
Label {
content: "Analysis"
police: black
position: node
```

```
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_green
form: Square
}
}
}
Representation {
condition: Java MethodNonCompliant
Label {
content: "Analysis Non Compliant"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_red
form: Square
}
}
}
Representation {
condition: Java MethodNA
Label {
content: "Analysis Non Applicable"
police: white
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: dark_blue
form: Square
}
}
}
} //close analysis
Node Inspection{
domain-context: ViVA.data.Inspection
provided-by association ViVA.data.VivaRequirement.inAss
Representation {
Label {
content: "Inspection"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
Representation {
condition: Java MethodCompliant
Label {
content: "Inspection"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_green
form: Square
}
}
}
Representation {
condition: Java MethodNonCompliant
Label {
content: "Inspection Non Compliant"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_red
form: Square
}
}
}
Representation {
condition: Java MethodNA
Label {
content: "Inspection Non Applicable"
police: white
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: dark_blue
form: Square
}
}
}
} //close inspection
Node Test{
domain-context: ViVA.data.Test
provided-by association ViVA.data.VivaRequirement.teAss
Representation {
Label {
content: "Test"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_blue
form: Square
}
}
}
Representation {
condition: Java MethodCompliant
```

```
Label {
content: "Test"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_green
form: Square
}
}
}
Representation {
condition: Java MethodNonCompliant
Label {
content: "Test Non Compliant"
police: black
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: light_red
form: Square
}
}
}
Representation {
condition: Java MethodNA
Label {
content: "Test Non Applicable"
police: white
position: node
alignment: left
}
Style {
BasicStyle {
border-color: black
background: dark_blue
form: Square
}
}
}
} //close test
} //close contains
} //close container
Container ComponentPlaceholder{
import: OAIBOperationalActivity
}
Edge VivaRequirement_reqReqAss {
association-context: ViVA.data.VivaRequirement.reqReqAss
source: VivaRequirementContainer
target: VivaRequirementContainer
Representation {
Style {
end-decorator: InputArrow
color: black
}
}
}
Edge VivaRequirement_compAss {
association-context: ViVA.data.VivaRequirement.pcRef
source: VivaRequirementContainer
target: ComponentPlaceholder
Representation {
Style {
line-style: dash
end-decorator: InputArrow
color: gray
}
}
}
} //close mapping
Actions {
/* VivaRequirementContainer Actions*/
Create VivaRequirement_CT_19{
label: "Requirement" action-for: VivaRequirementContainer
}
Drop VivaRequirement_DR_19{
action-for: VivaRequirementContainer
}
Delete VivaRequirement_DL_19{
action-for: VivaRequirementContainer
}
/*ROD Actions */
Create ROD_CT_21{
label: "ROD" action-for: VivaRequirementContainer.ROD
}
Drop ROD_DT_21{
action-for: VivaRequirementContainer.ROD
}
/*Analysis Actions */
Create Analysis_CT_22{
label: "Analysis" action-for: VivaRequirementContainer.Analysis
}
Drop Analysis_DT_22{
action-for: VivaRequirementContainer.Analysis
}
/*Inspection Actions */
Create Inspection_CT_23{
label: "Inspection" action-for: VivaRequirementContainer.Inspection
}
Drop Inspection_DT_23{
action-for: VivaRequirementContainer.Inspection
}
/*Test Actions */
Create Test_CT_24{
label: "Test" action-for: VivaRequirementContainer.Test
}
Drop Test_DT_24{
action-for: VivaRequirementContainer.Test
}
/* Requirement Edge Actions */
Create reqedge {
label: "Requirement Children" action-for: VivaRequirement_reqReqAss
}
Drop reqedgedrop {
action-for: VivaRequirement_reqReqAss
}
ReconnectEdge reqedgerecon {
action-for: VivaRequirement_reqReqAss
}

/* Component Edge Actions */
Create compedge {
label: "Component Connection" action-for: VivaRequirement_compAss
}
```

```
Drop compedgedrop {
action-for: VivaRequirement_compAss
}
ReconnectEdge compedgerecon{
action-for:VivaRequirement_compAss
}
} //close actions
} // close DiagramExtension
} // close Diagrams
```

Phase 5: Odesign

In my personal experience, this is a great tool, but hard to manage if you do not know where to put your hands.
As I said before, all the design implementation could have been done already here, without the need to modify the .vptext. Since I discovered it too late, and I already understood how to manage the other codes, I decided to keep it that way.
Another tip, as you will see in a minute, if you change the .odesign, the .vptext will not be automatically updated. So, I suggest having everything ready and not changing it anymore before using this one. Otherwise, you will need to begin every time from the start.

You may have noticed that, if you create the diagram extension for the Physical Level, the Behaviour PC are not visible, even though they are created in the model. This happens because the .diagram.vptext file is old and doesn't work perfectly, so we can fix that using the .odesign.

The .odesgin file can be founded under:
Org.polarys.capella.vp.yourname.**design** -> description -> yourname.odesign

To be sure that you modification will be saved, first you need to open the Configuration file you can find in the main page:

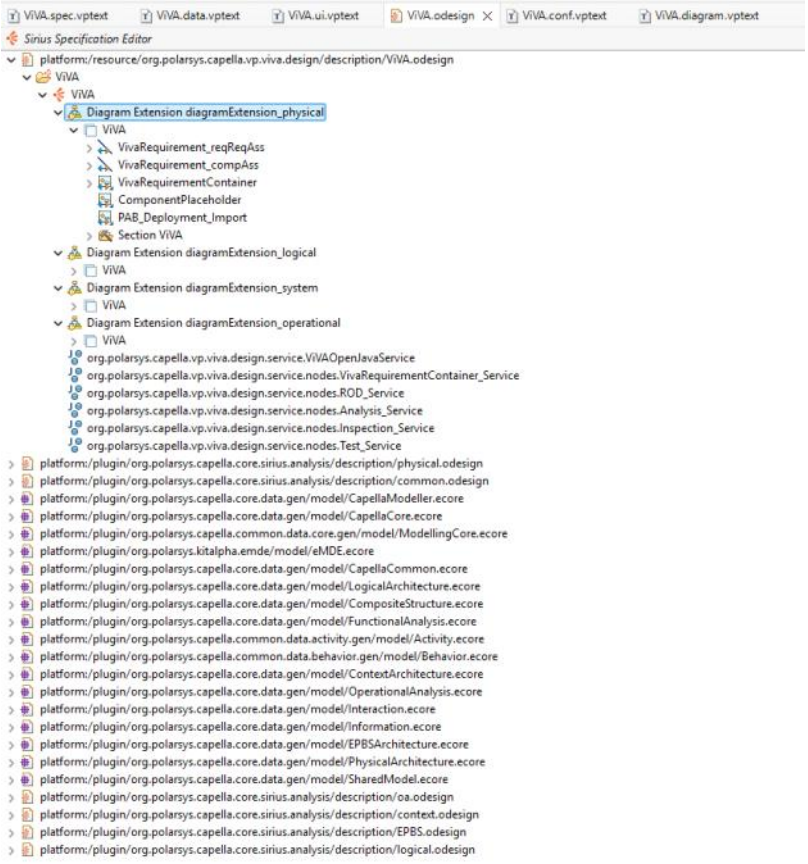
```
Viewpoint ViVA {
name: "ViVA"
Data ViVA.data
UI ViVA.ui
Diagrams ViVA.diagram
Configuration ViVA.conf
}
```

Here you need to turn

```
diagram {
OverwriteOdesign: true
}
```

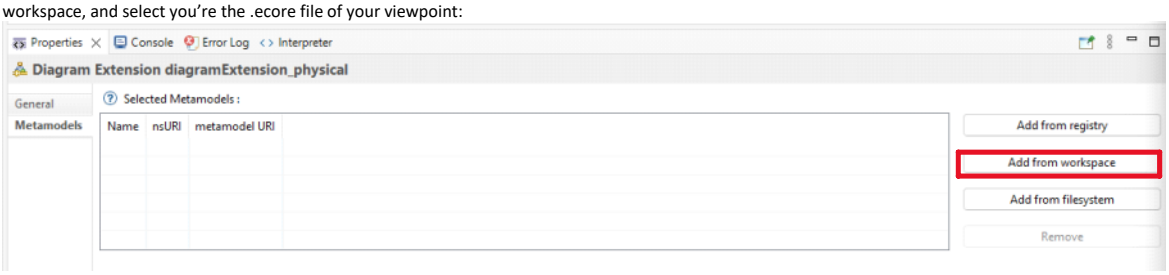
From true -> to false

Now, the .odesign file, will look like this:

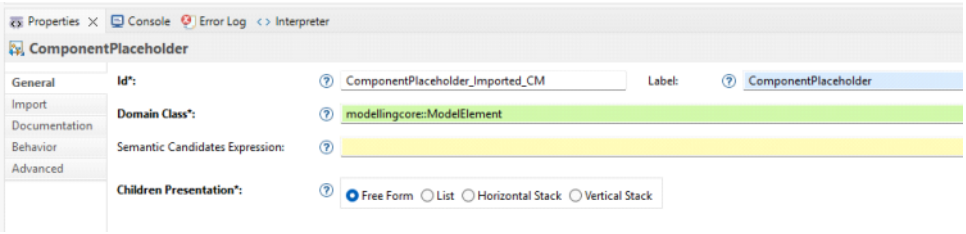


The first thing to fix is connected to the BPC, as I said before. To do so there's a couple of step to do.

First we need to reference our Metamodel in every diagram extension. Just select you the diagram and inside Properties, in Metamodels, select Add from workspace, and select you're the .ecore file of your viewpoint:

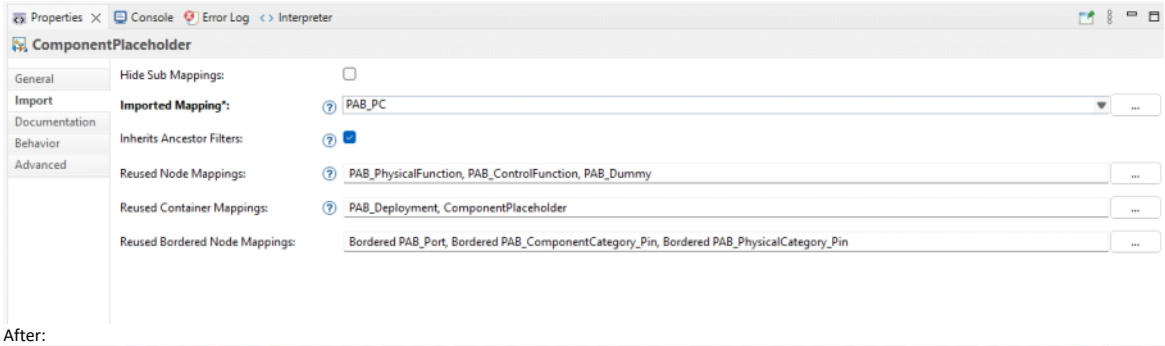


Then, in Component Placeholder, the first thing is to change the Domain Class from cs.Part to modellingcore::ModelElement (the Obeo engineers told me it's better).

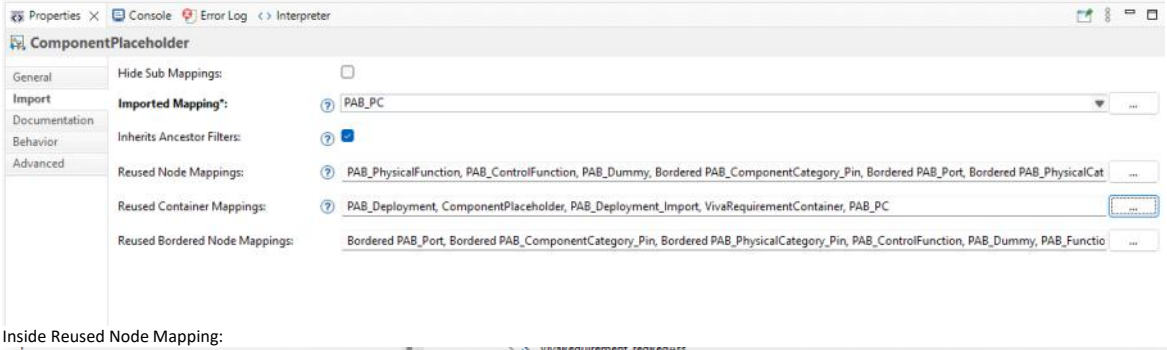


Then we go to the Import Page. Here, in all three Reused xx Mapping, I filtered the element by levels and I added all the physical elements **RELATED TO THE PAB DIAGRAM**, so we can connect requirements to everything (just click on the 3 dots on the right and select, then click Add). You'll see in behaviour all the new functions added.

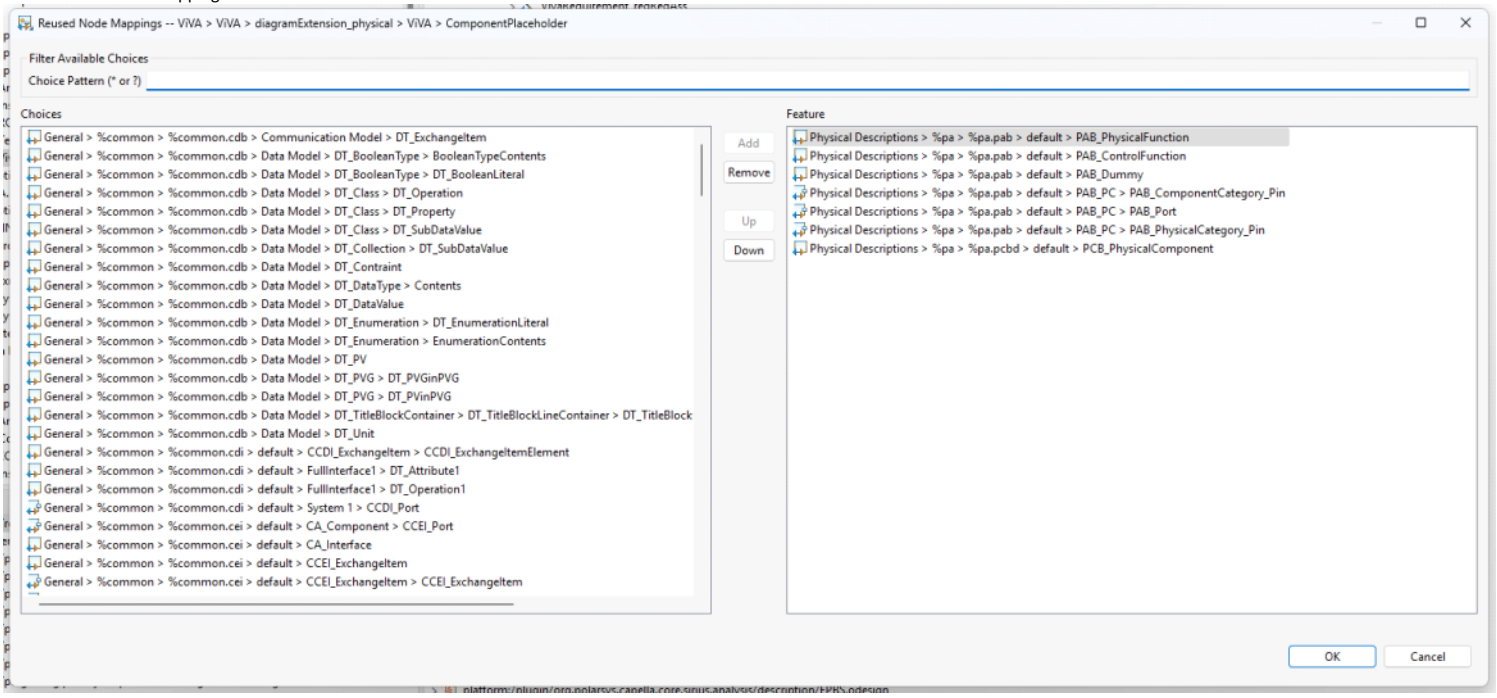
Before:



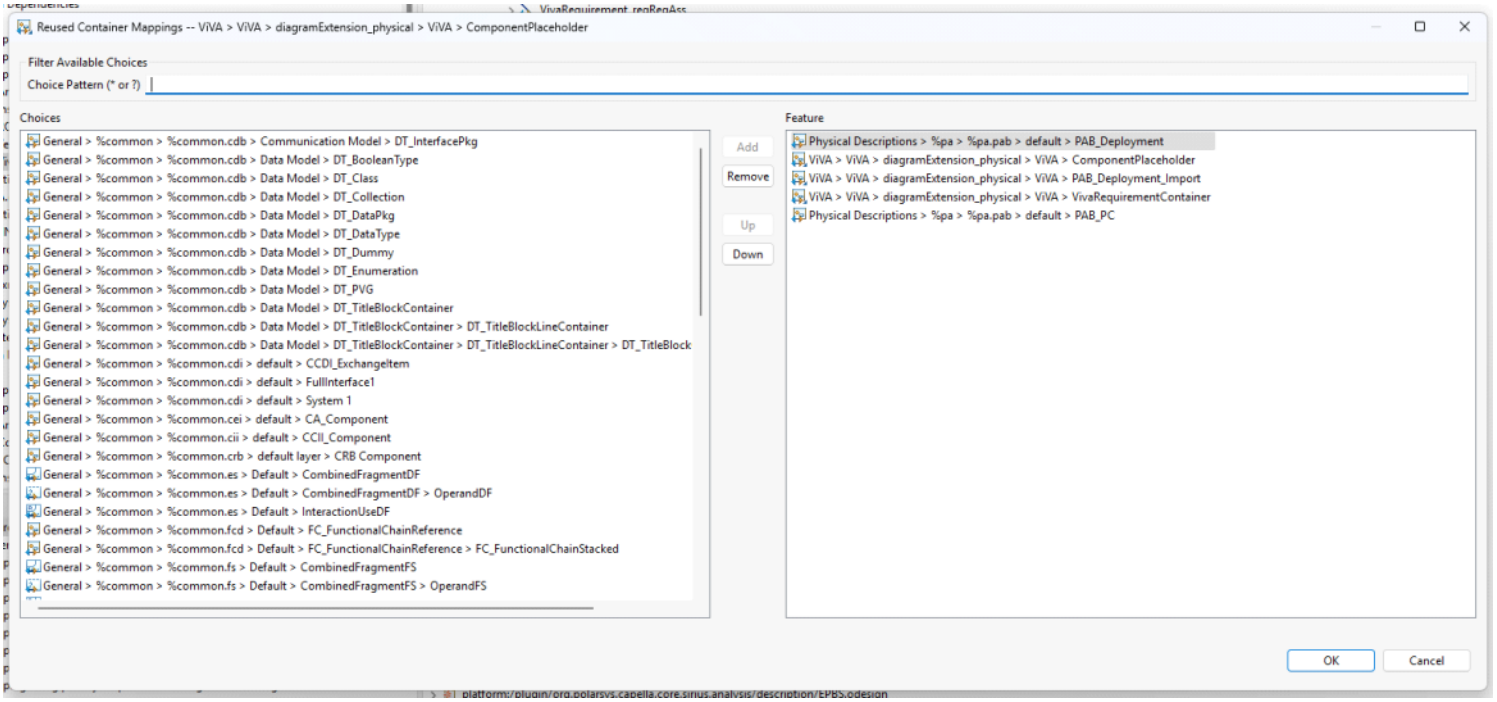
After:



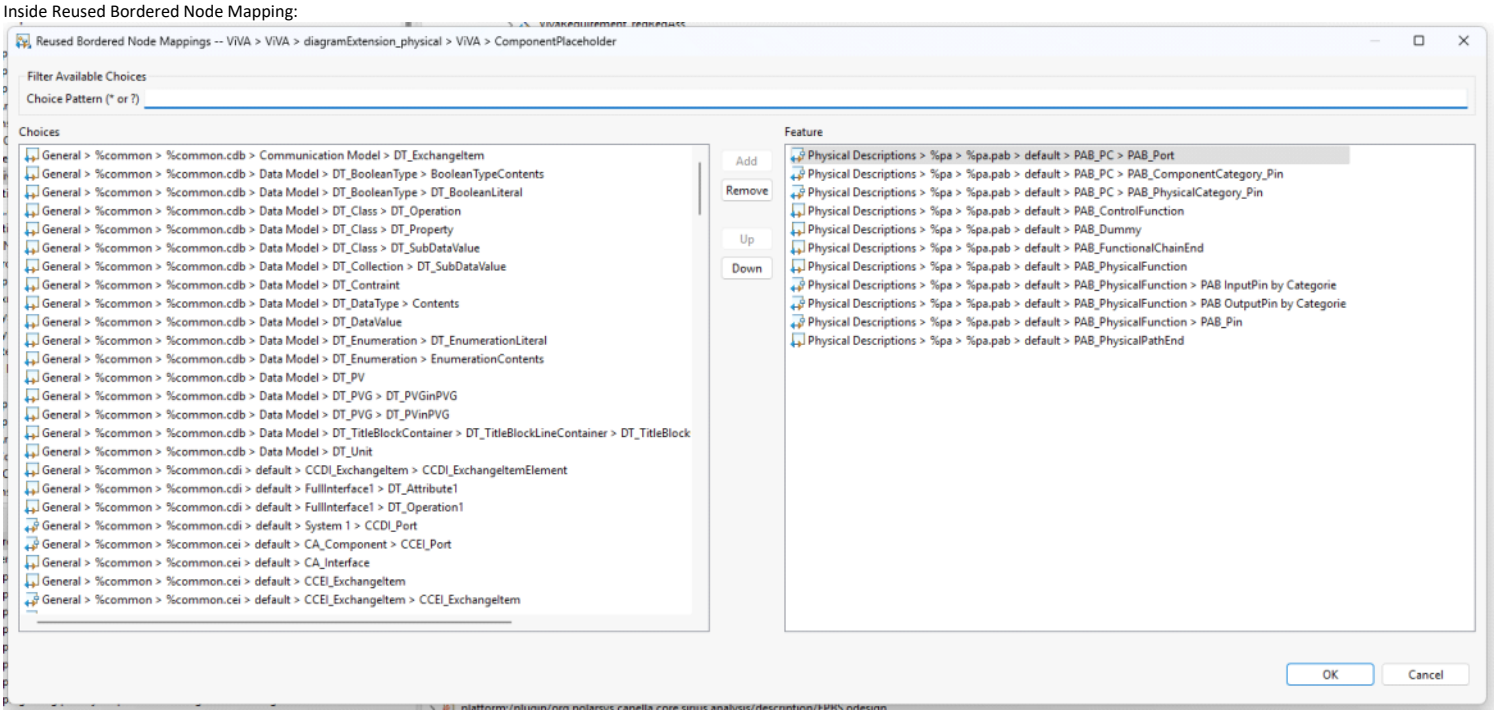
Inside Reused Node Mapping:



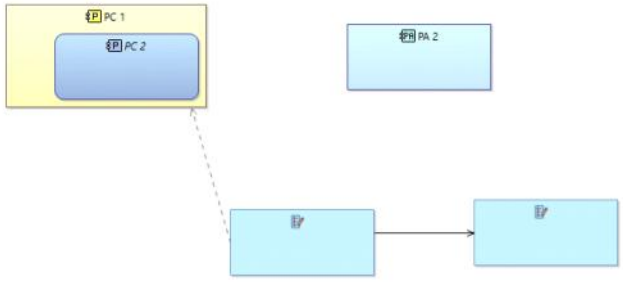
Inside Reused Container Mapping:



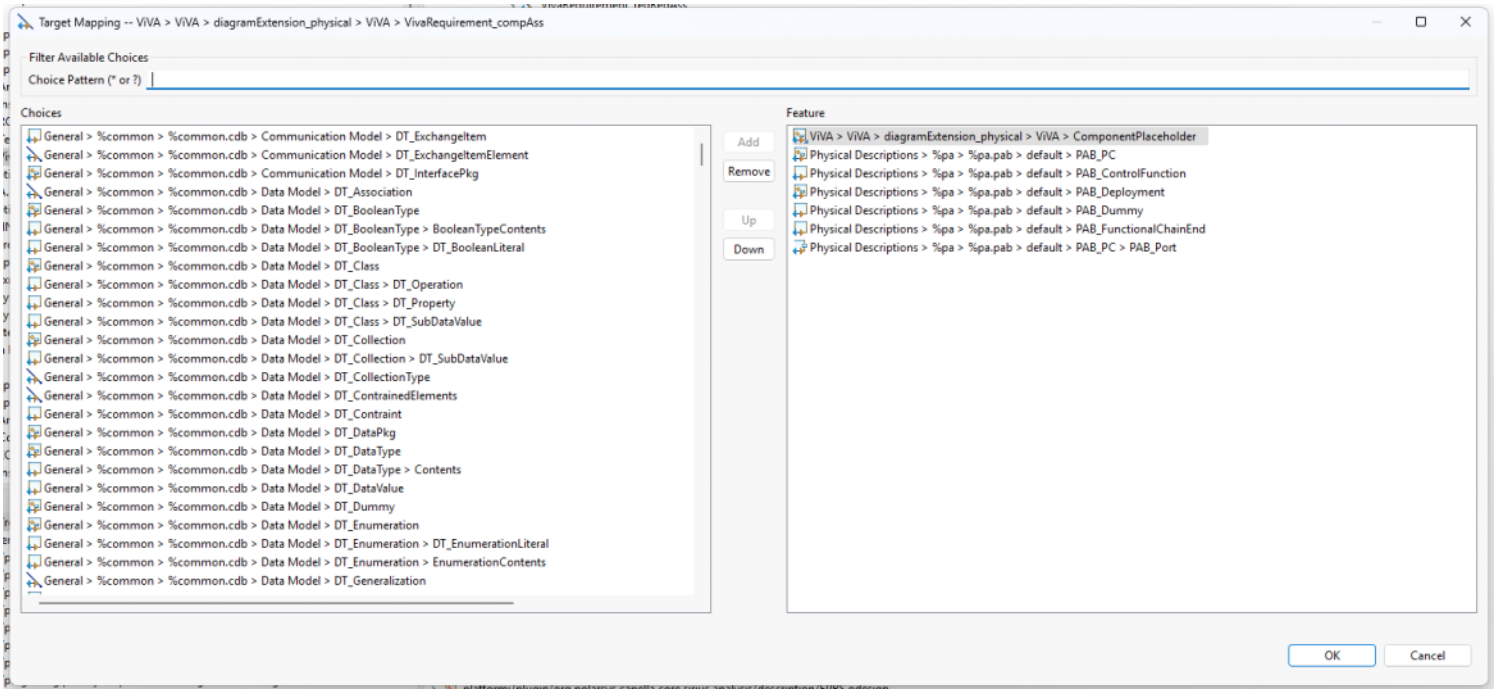
Inside Reused Bordered Node Mapping:



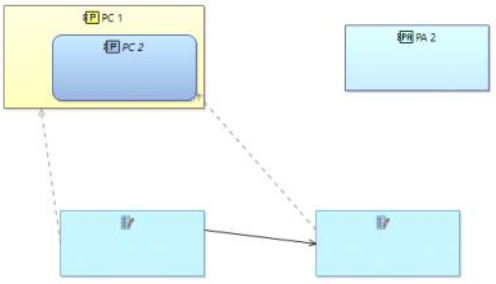
As you can see now, all the elements in our diagram are visible, without bugs.
P.s. if you decide to just use the .odesign file without the .diagram.vptext, you don't need to create a component placeholder, just add everything in your edge properties.



To be able to connect also your requirement to the PC2 you need to add the same information in the edge properties page, inside the Target Mapping section:



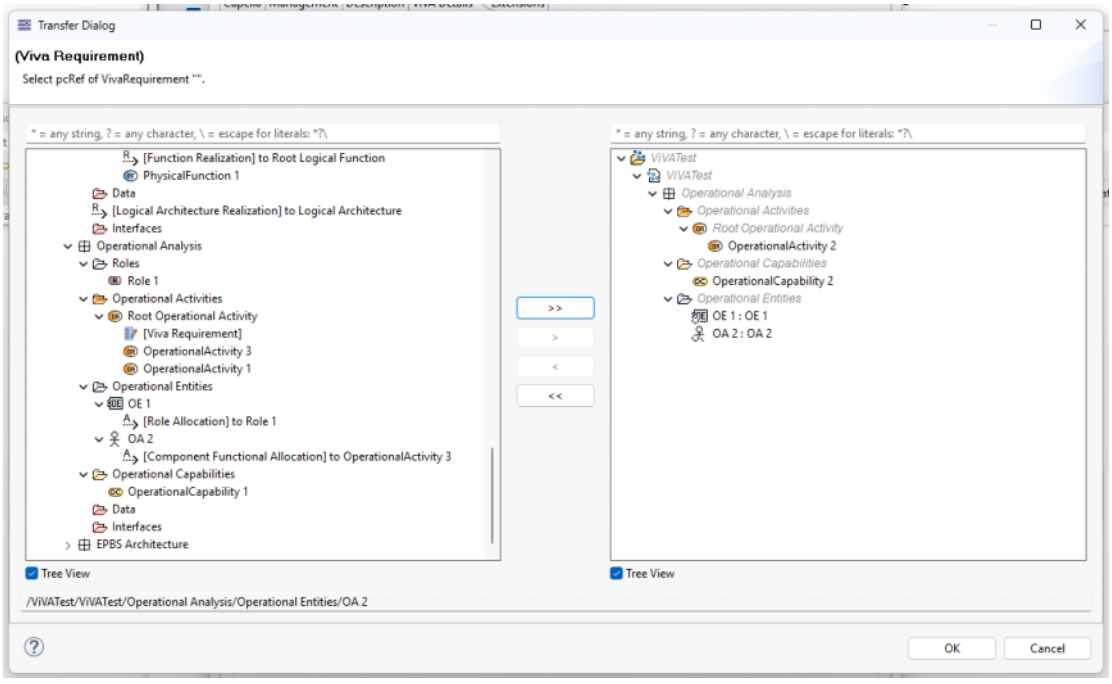
Now it works:



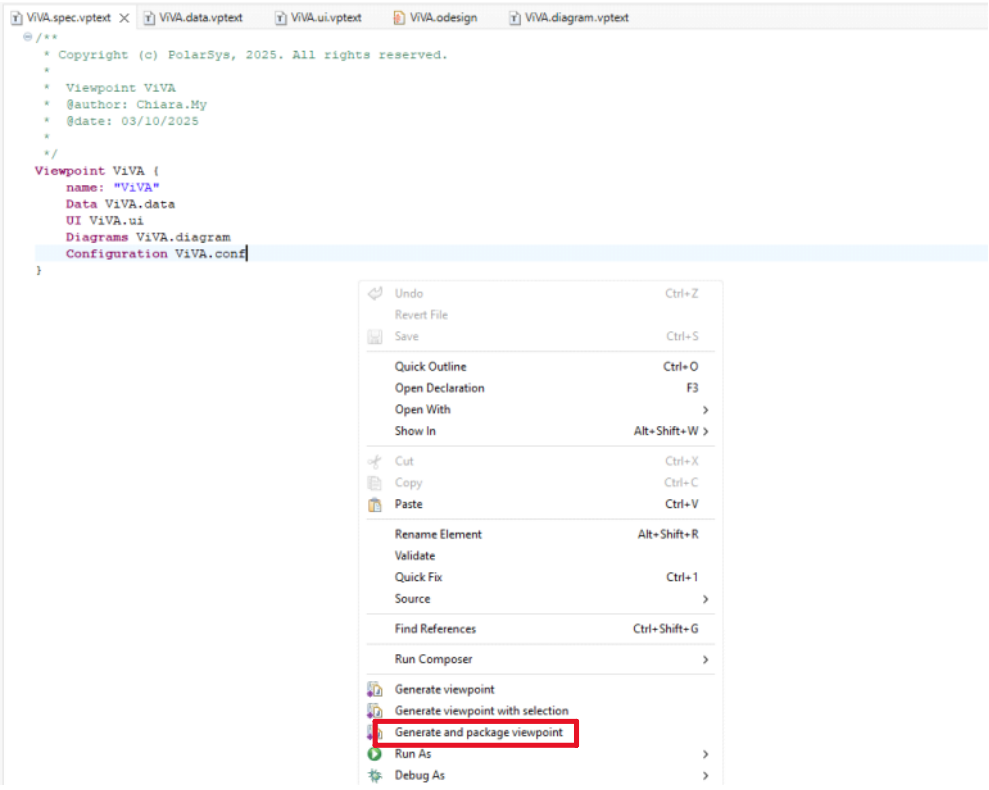
In the LAB diagram, we don't see problems, if not for the connection not available with functions. This has actually to be fixed for all diagrams, if someone wants to.

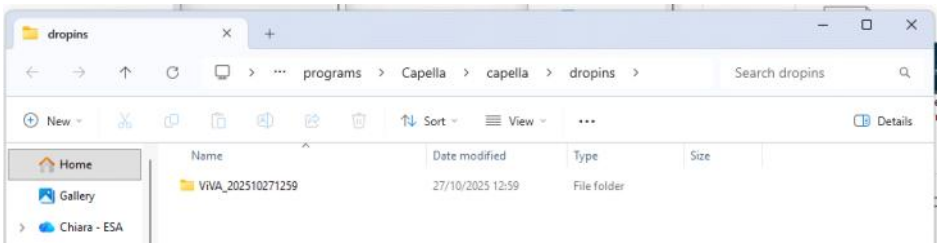
In the SAB diagram, we can see that the connection with System Actors is not possible. With the same process we already saw, we can fix it.

In the OAIB diagram, we don't see problems either. Unfortunately, the option to create the diagram extension for the OAB was not available.
You can still connect everything you need to just inside the properties of your requirement:



When your viewpoint is ready, you just need to click on "Generate and package viewpoint" and then copy/paste your folder inside the "dropins" folder in Capella.


ViVA.spec.vptext
ViVA.data.vptext
ViVA.ui.vptext
ViVA.odesign
ViVA.diagram.vptext
/**
* Copyright (c) PolarSys, 2025. All rights reserved.
*
* Viewpoint ViVA
* @author: Chiara.My
* @date: 03/10/2025
*
*/
Viewpoint ViVA {
name: "ViVA"
Data ViVA.data
UI ViVA.ui
Diagrams ViVA.diagram
Configuration ViVA.conf
}
Undo Ctrl+Z
Revert File
Save Ctrl+S
Quick Outline Ctrl+O
Open Declaration F3
Open With
Show In Alt+Shift+W
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Rename Element Alt+Shift+R
Validate
Quick Fix Ctrl+1
Source
Find References Ctrl+Shift+G
Run Composer
Generate viewpoint
Generate viewpoint with selection
Generate and package viewpoint
Run As
Debug As


dropins
programs
Capella
capella
dropins
Search dropins
New
Sort
View
Details
Home
Gallery
Chiara - ESA
Name
Date modified
Type
Size
ViVA_202510271259
27/10/2025 12:59
File folder

References/Sources

02 October 2025 11:33

Some sources and links that could be useful:

Eclipse Capella Forum - Eclipse Capella Forum
Tutorials · eclipse-capella/capella-studio Wiki
quality_assessment_capella_viewpoint/README.md at master · houssamkanso/quality_assessment_capella_viewpoint
Eclipse Capella™
eclipse-capella/capella-basic-vp
Xtext - Eclipse Support
Viewpoint development · eclipse-kitalpha/kitalpha Wiki
Eclipse Sirius Documentation
Creation of viewpoint with Capella Studio 1.1
Viewpoint: the making of. Customizing Capella with Capella Studio in 20 minutes
Textual DSL for AF and Viewpoint DSL-v0.5.5