



# ReqCycle V0.7.4 Starting guide

1st of February 2014

Raphaël Faudou  
[raphael.faudou@samarès-engineering.com](mailto:raphael.faudou@samarès-engineering.com)

# Agenda

- **Changes from last release**
- **Motivations, overview and current state**
- **Installation**
- **Big picture and detailed usage step by step**
  1. Data model – scopes and types
  2. Requirement import, creation and visualization
  3. Configuration and creation of traceability links
  4. Capture and management of traceability links
  5. Export of requirements and links (doc, matrix,...)
- **Teamwork support**
- **Questions / contacts**





# Changes from last release

Raphaël Faudou  
[raphael.faudou@samarès-engineering.com](mailto:raphael.faudou@samarès-engineering.com)

# Changes from V0.7.3



- See V0.7.4 release notes to see all changes in code
- New and modified slides
  - Installation steps : OCL support
  - Step 1 conclusion: file restart
  - Step 2: Import SysML requirements





# Motivations for ReqCycle

Raphaël Faudou

[raphael.faudou@samarès-engineering.com](mailto:raphael.faudou@samarès-engineering.com)

- Provide an open source solution supporting all activities concerning requirements in the development of an industrial product or system...

- Capture, elicit, prioritize, select, transform...
- Define, derive, refine, verify quality, validate content relevance...
- Classify, prioritize, allocate, put in baseline...
- Verify traceability and maintain it on change requests

... especially in the context of Model Based System Engineering where requirements can be textual or model elements => require 2 technical services

- Identify/extract requirements from model
- Identify/extract requirement traceability links from a set of models

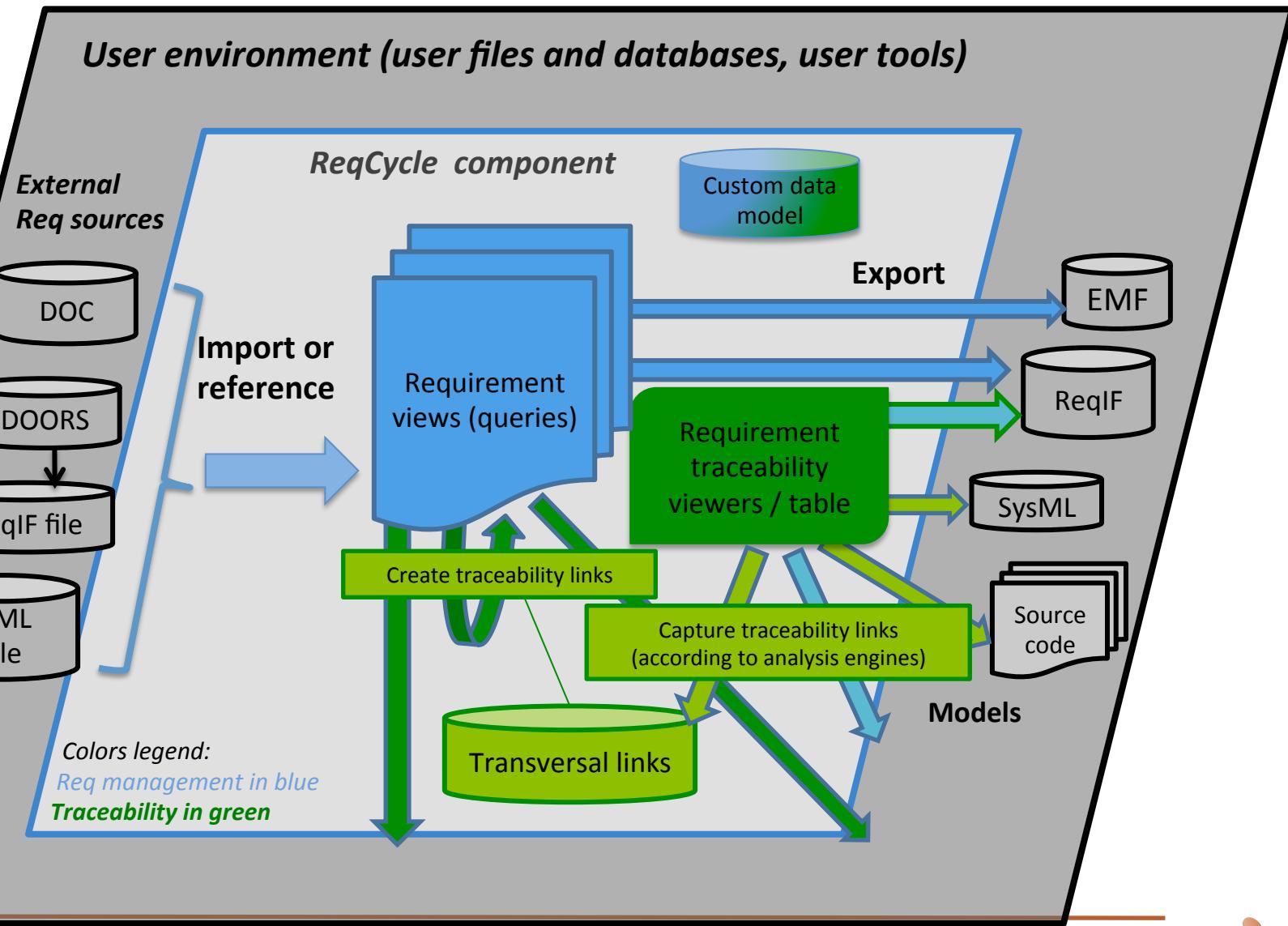


# Motivations

- **Provide a flexible solution able to adapt to any organization**
  - Can organize requirements with a user-defined data model
  - Can drive traceability through a user-defined process
  - Can import or reference requirements managed in any requirement database
  - Can capture existing requirement traceability links from any language/tool



# ReqCycle overview



## Current state

- ReqCycle does not yet implement requirement engineering (planned for 2015)
- ReqCycle provides means to create a custom data model to manage requirements and traceability relationships
- ReqCycle can import requirements coming from different external sources or can create requirements
  - External requirement sources already supported: ReqIF, EMF models



- **ReqCycle can create traceability links between requirements or between one requirement and a model element according to the traceability relationships defined in the custom data model**
  - EMF UML and SysML models supported for now
- **ReqCycle can capture existing traceability links or any link from models and display extended traceability (all links from one element to another)**
  - UML/SysML, OCL, XCos and Java traceability analysers
- **ReqCycle cannot yet export requirement traceability in EMF format (soon)**





# ReqCycle installation

Available for Windows, Linux and MacOS  
Checked for Windows 7 64 bits and MacOS 10.8 64 bits

Raphaël Faudou  
[raphael.faudou@samarès-engineering.com](mailto:raphael.faudou@samarès-engineering.com)

## 1. Ensure you get a Java Runtime Environment V1.6 or higher installed

- a. In terminal view type and run command line “java –version”
- b. If version is lower than 1.6 or if you do not have java installed, download and install it according to your operating system:  
<http://www.java.com/fr/download/>
  - Note: for Windows 64 bits, you might need to install a Java 64 bits runtime to fully benefit from 64 bits

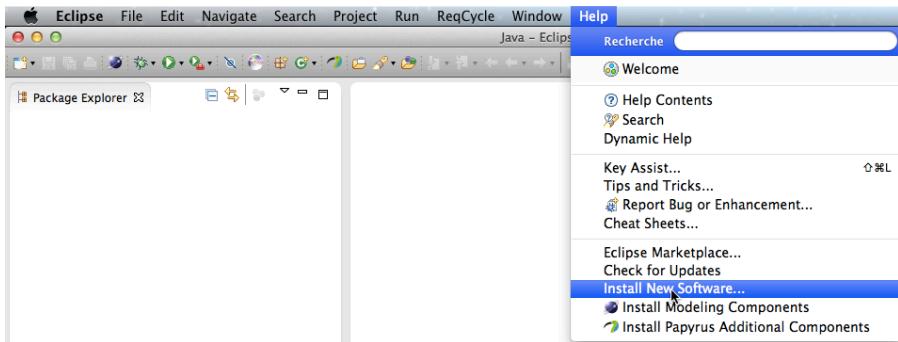
## 2. Download Eclipse Kepler Modeling SR1

- a. <http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/keplerr>
- b. Choose your installation package according to your operating system (installation packages differ for 32 bits or 64 bits)

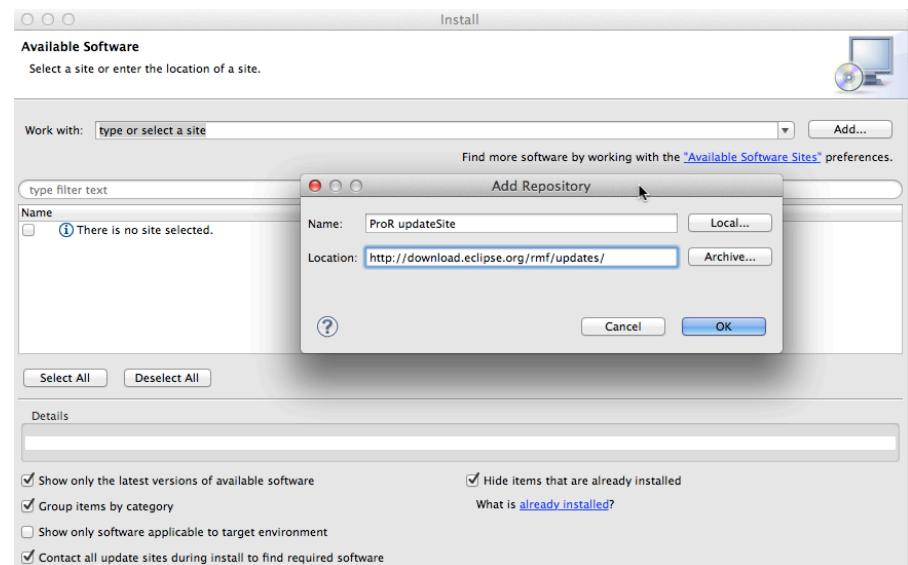


# Installation steps – ReqIF support (1)

- Launch Eclipse application
- Go to Help>new software

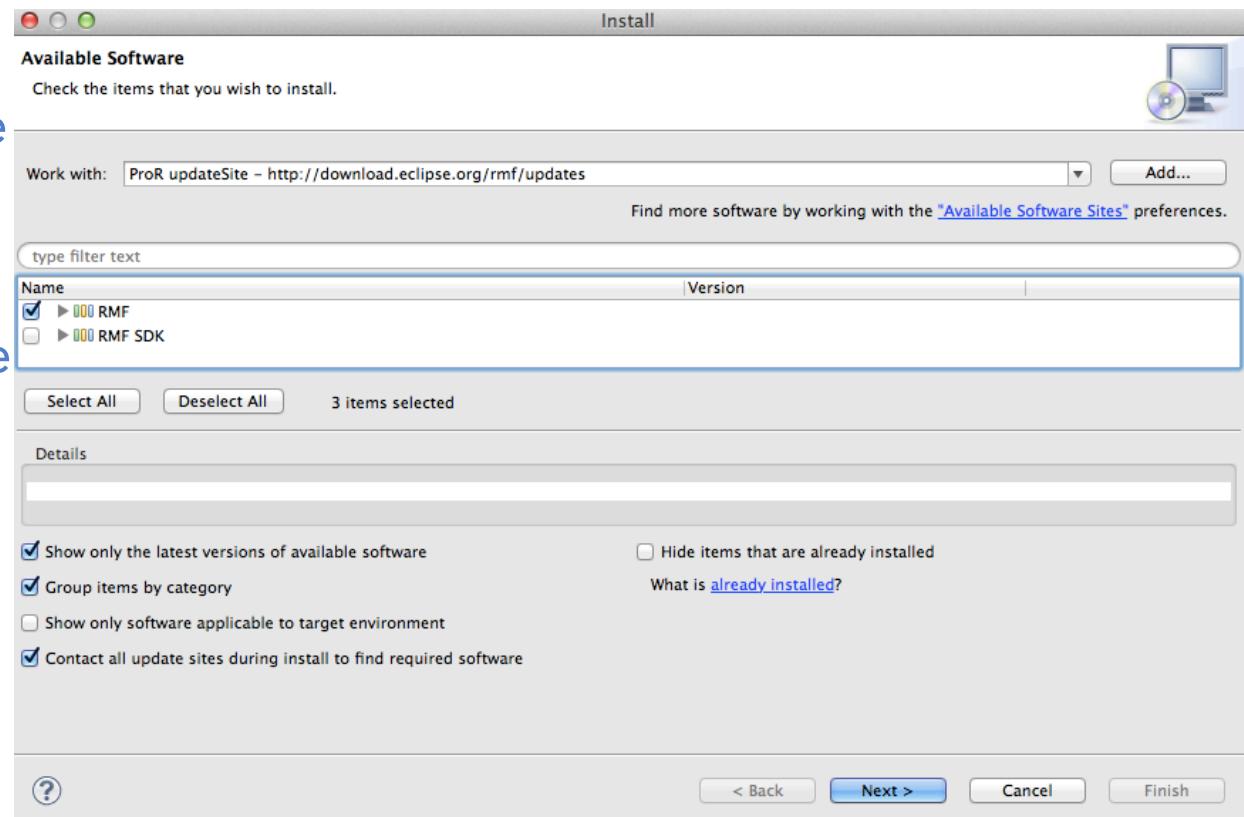


- Click "Add" button
  - Fill name and location
  - <http://download.eclipse.org/rmf/updates/>
  - Click "OK"



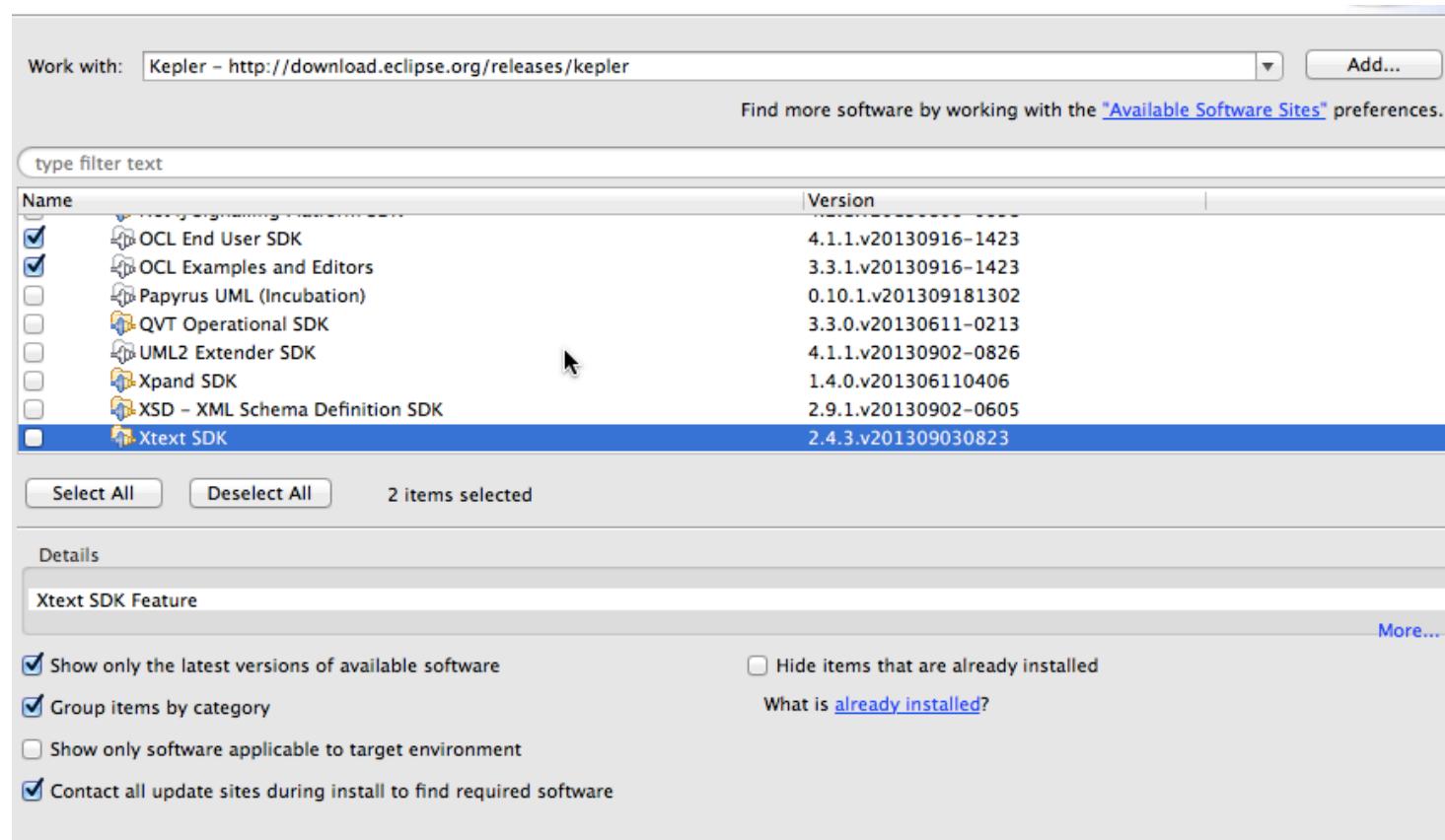
# Installation steps – ReqlF support (2)

- Only select « RMF » and follow wizard with check boxes filled as below
  - Next,
  - Accept license
  - Finish
  - OK to install
  - Restart Eclipse



# Installation steps - OCL support

- Install new... and select Kepler update
- Choose “Modeling” project and select OCL as below

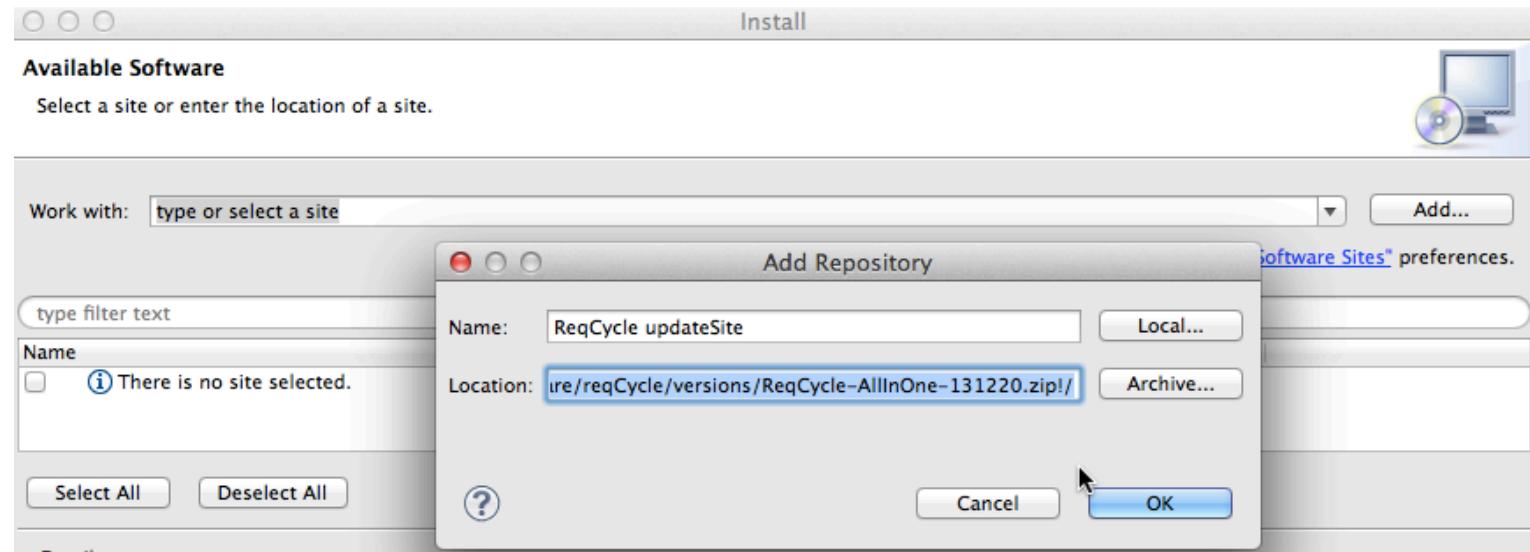


## 1. Download ReqCycle updateSite

- <https://github.com/raphaelfaudou/ReqCycle/blob/master/releases/ReqCycle-AllInOne-131220.zip>

## 2. In Eclipse, menu Help>new software

- Add ReqCycle update site (select .zip with “archive” button)

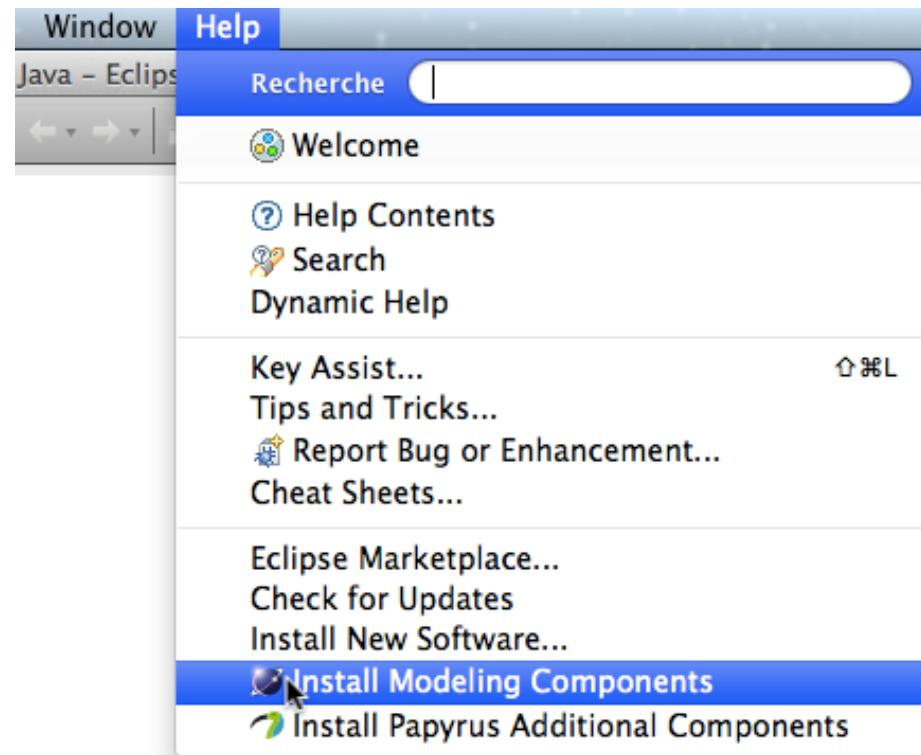


- OK - follow wizard (includes Eclipse restart) - That's it ☺



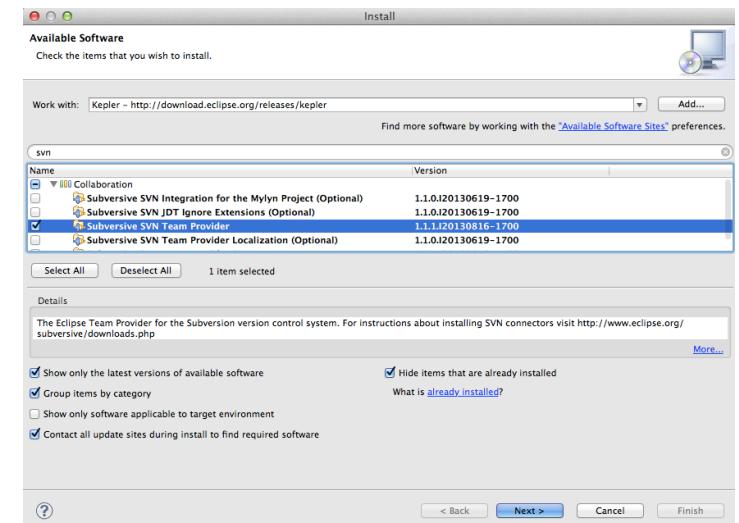
# Installation steps – MDT Papyrus

- You can then install MDT Papyrus to create links between requirements and Papyrus models...
  - Menu Help>Install Modeling Components
  - Choose Papyrus
  - Follow wizard



# Installation steps – SVN connectors

- You can install SVN team and connectors to put your requirements and models in version management in a SVN repository
  - Team plugins are part of Kepler release: install new software, select Kepler and then "collaboration>subversive SVN team provider")



- Install connectors with update site:
  - <http://community.polarion.com/projects/subversive/download/eclipse/3.0/kepler-site/>
- In case of issues, full Installation instructions are detailed here:  
<http://www.polarion.com/products/svn/subversive/download.php>





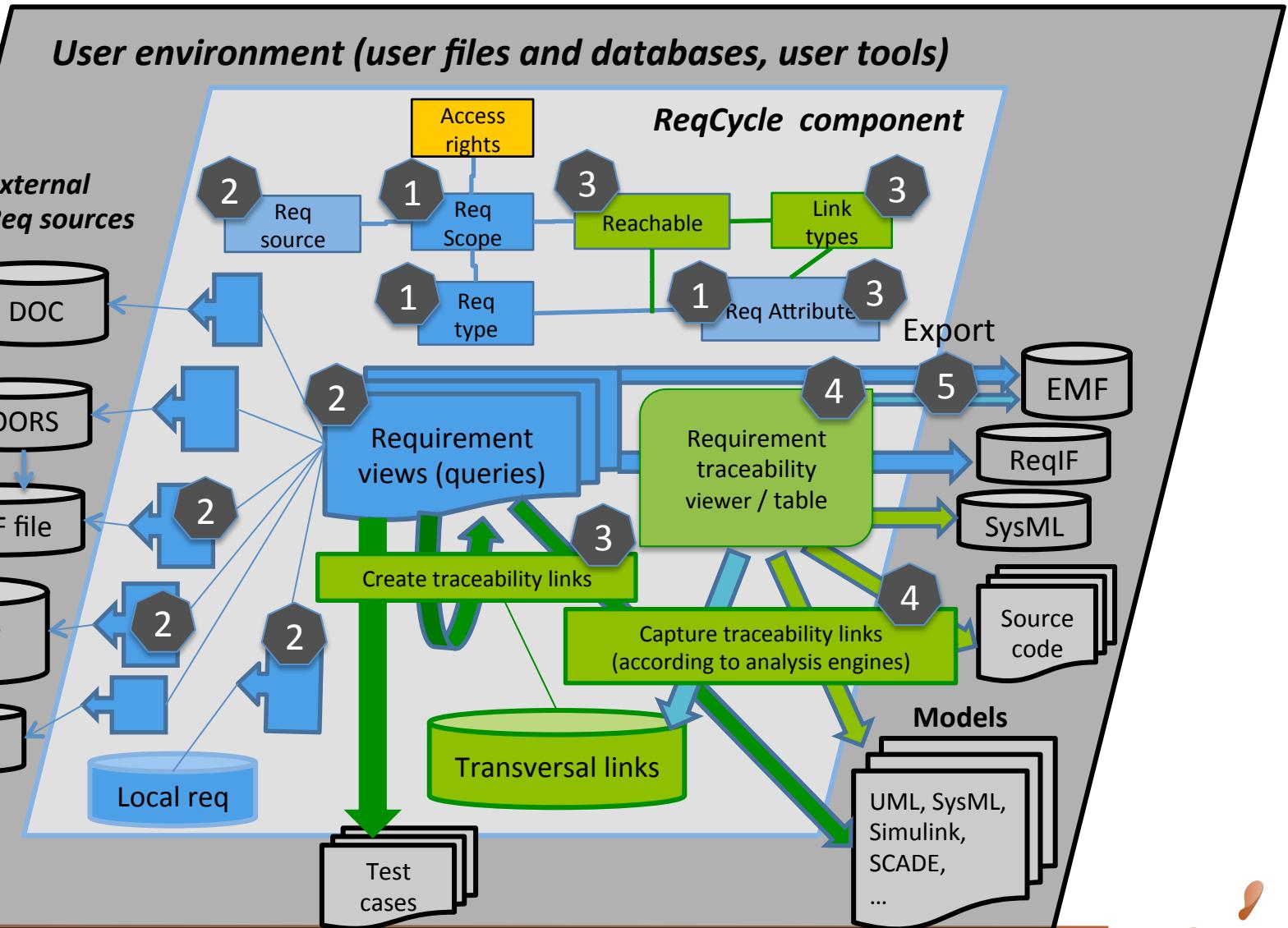
# Big picture and process overview

Raphaël Faudou

[raphael.faudou@samarès-engineering.com](mailto:raphael.faudou@samarès-engineering.com)

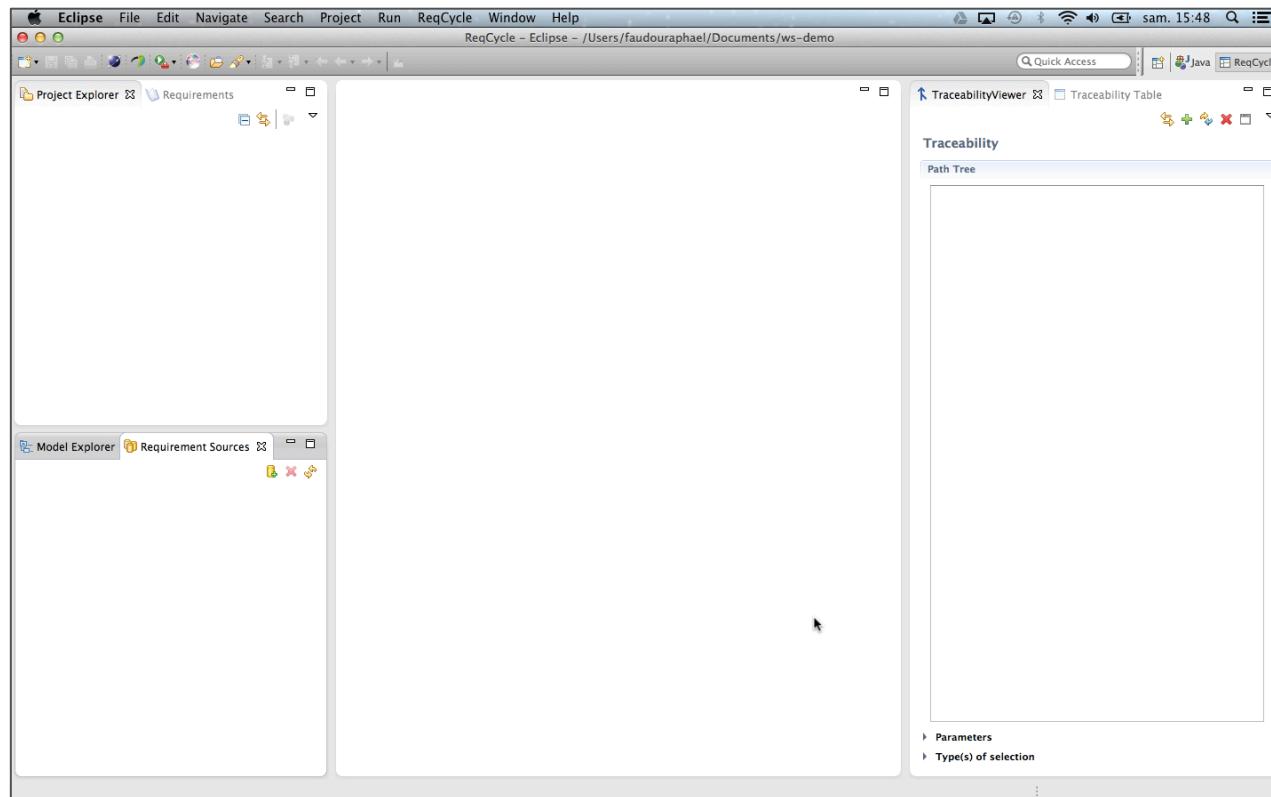
# Big picture and process overview

1. Define custom data model
2. Import/create / view and edit requirements
3. Configure and create traceability links
4. Capture / view traceability
5. Export requirements and traceability links



# Requisites - ReqCycle perspective

- Once ReqCycle is installed (see previous slides), launch Eclipse, choose a workspace and switch to ReqCycle perspective
  - Workbench is then organized in different views (requirement sources, editors, traceability viewer...) that will be presented in next slides





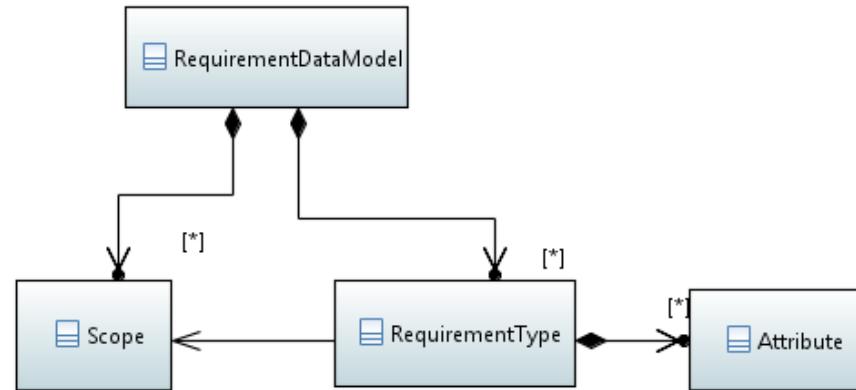
# 1st step - Requirement data model

## Concepts and screenshots

Raphaël Faudou  
[raphael.faudou@samarès-engineering.com](mailto:raphael.faudou@samarès-engineering.com)

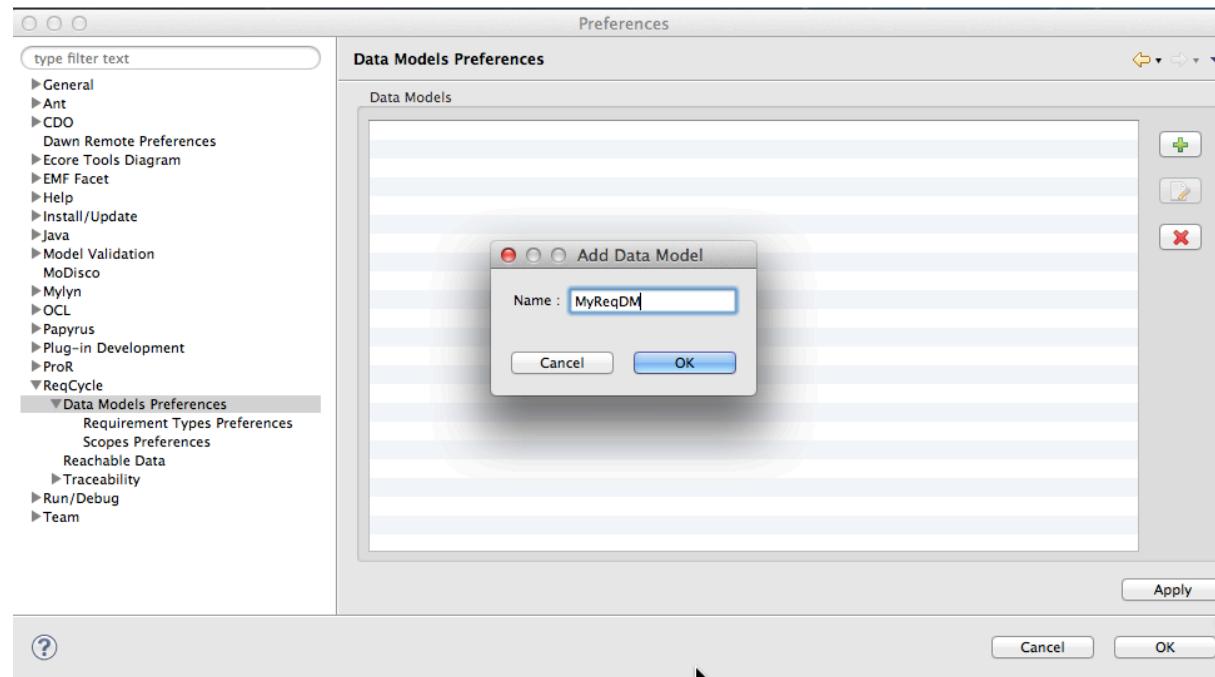
# Requirement custom data model

- A ReqCycle data model is defined with **scopes** (abstraction level/organization) and requirement types (nature/structure of requirements)
- **Scopes allow creating classification and allocation of teams**
  - Ex: Customer, SRS, SSRS, SSHA, SubS1, SubS2, SW-A, SW-B...
- **Types allow managing different natures and structures of requirements (with different attributes)**
  - Ex: Functional Req, Performance Req, Safety Req, Constraint...



# Create data model

- Go to main menu Preferences>ReqCycle
- In Data Models Preferences page, click “+” button and create a data model “MyReqDm” – click “OK”

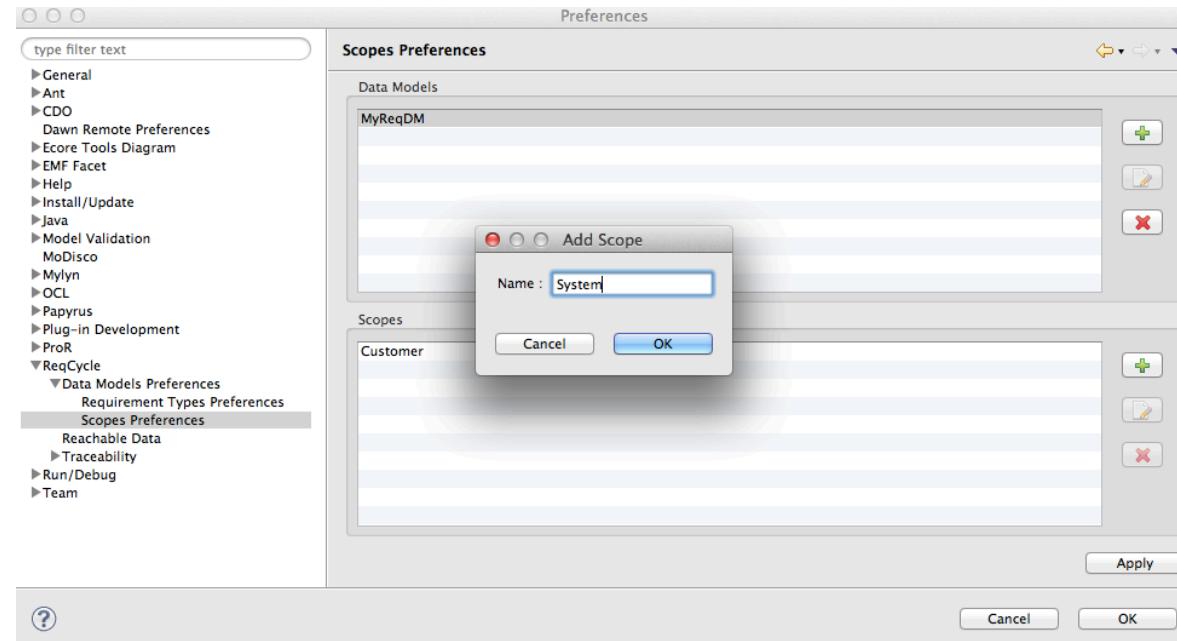


- Then select newly created data model and click « apply » button to register it



# Create scopes for this data model

- Go to Scopes Preferences page
- Select “MyReqDM” data model and add scope
  - For example “Customer” and “System”

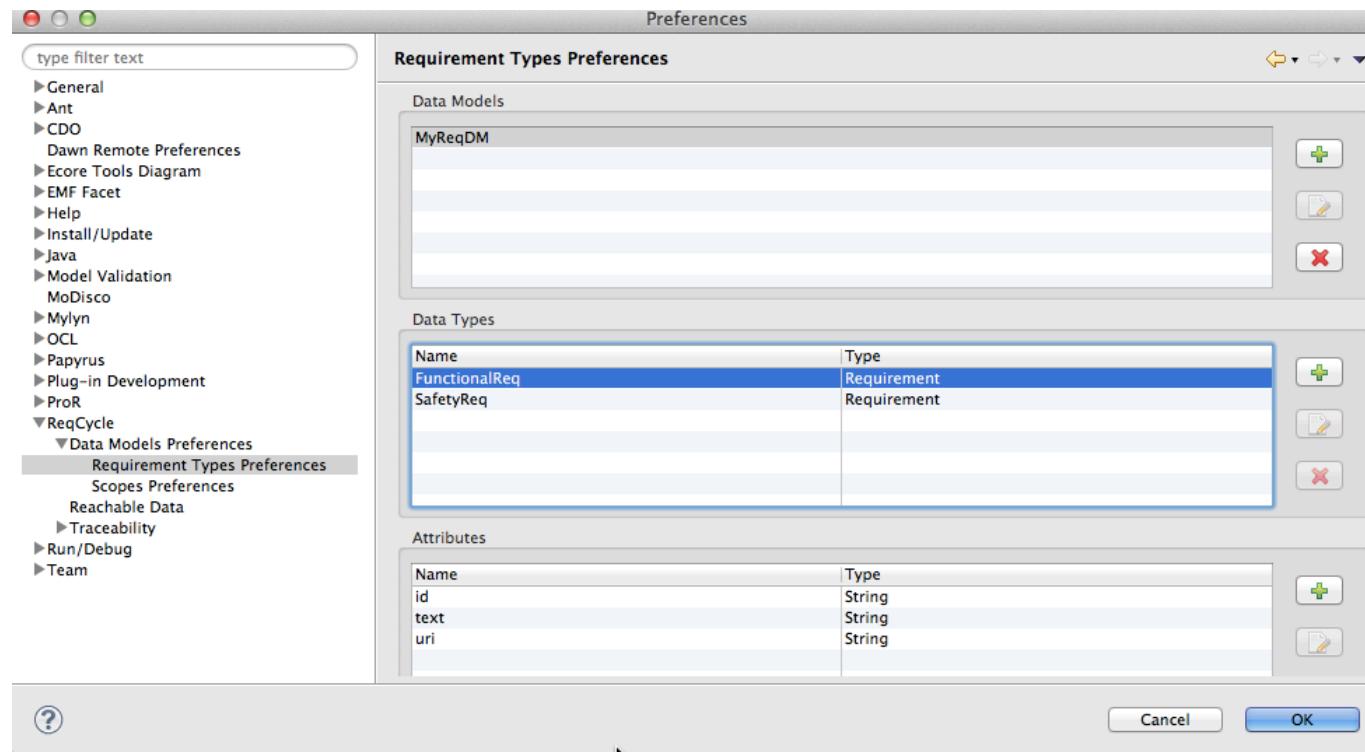


- Then click « apply » button



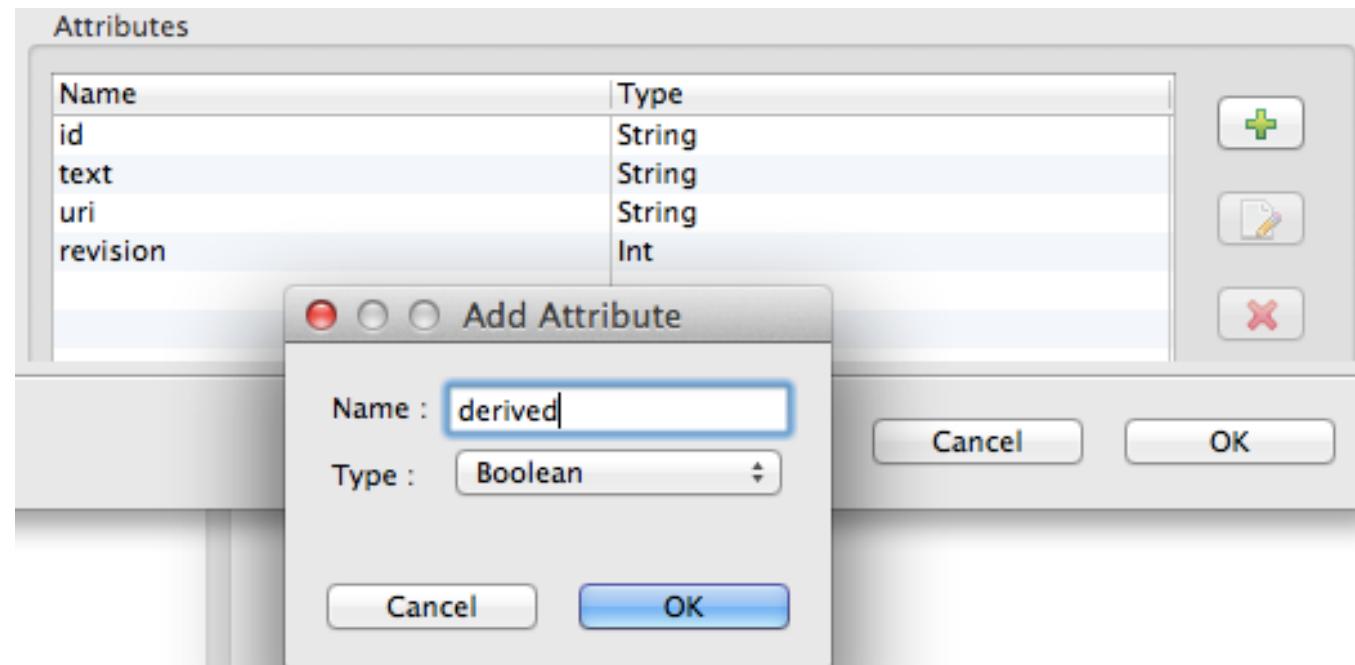
# Create requirement types

- Go to Requirement types Preferences page
- Select “MyReqDM” data model and add data types
  - For example “FunctionalReq” and “SafetyReq”



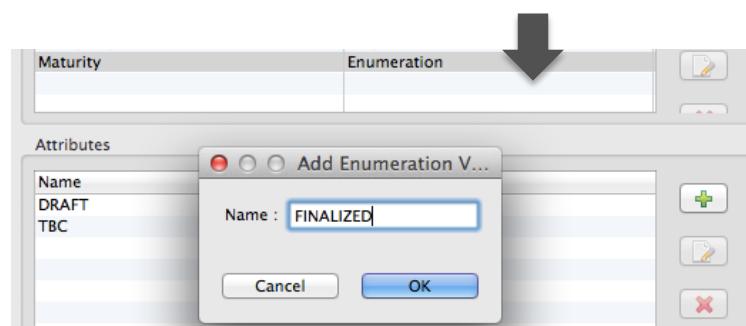
# Add requirement types attributes (1)

- Select one requirement type (data type)
- Can add attributes of predefined types
  - Examples: “revision” of type “int” and “derived” (bool)

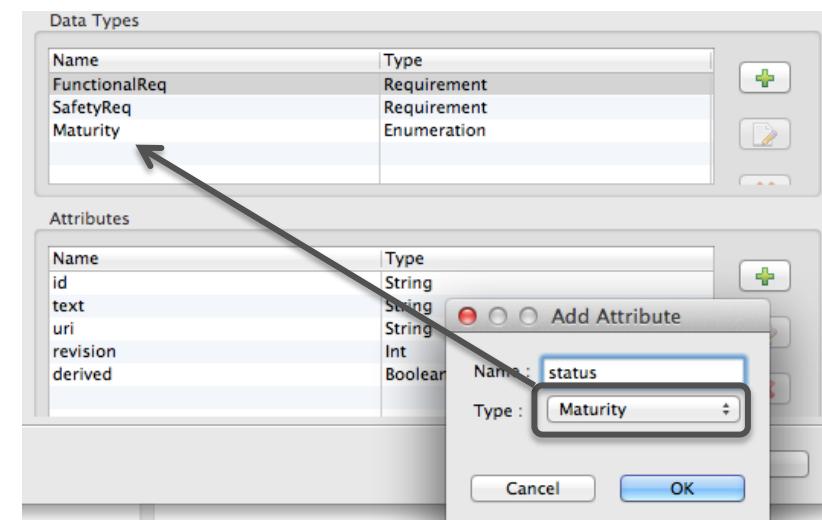
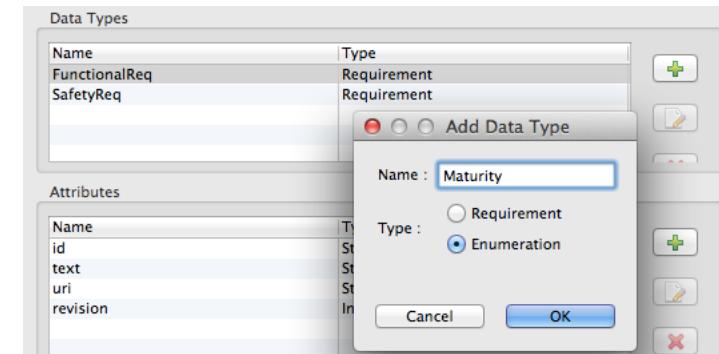


# Add requirement types attributes (2)

- Can also define (new) enumerations
  - Add new data type and select “enumeration”
  - Example: Maturity
  - Then select enum data type
  - Add values from attributes area

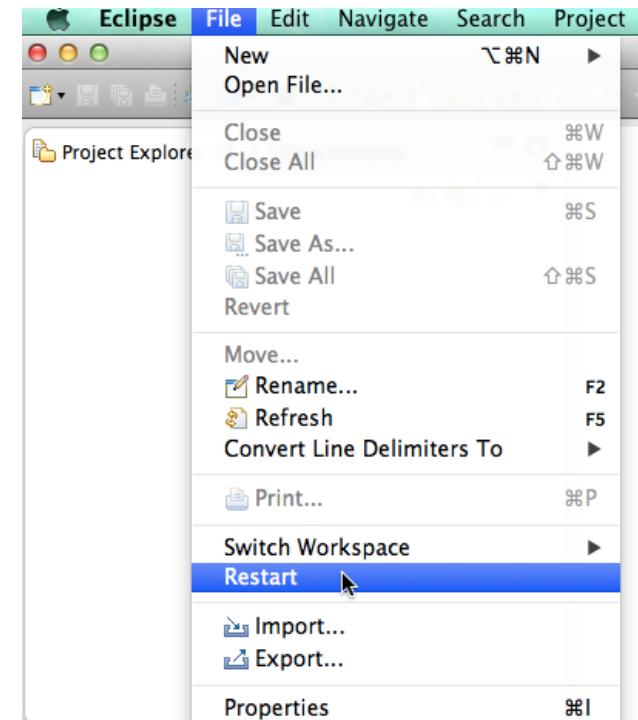


- Now can define a new attribute typed with this enum



# Step 1 conclusion

- Now we have seen how to create a custom data model with scopes, requirements types and attributes
- Restart your platform to register (persist) this requirement data model
  - Menu File>Restart
- In step 2 we will see how to integrate external requirements or create own requirements that conform to this data model





## 2nd step – Import /reference / create view / edit / filter requirements

Concepts and screenshots

Raphaël Faudou

[raphael.faudou@samarès-engineering.com](mailto:raphael.faudou@samarès-engineering.com)

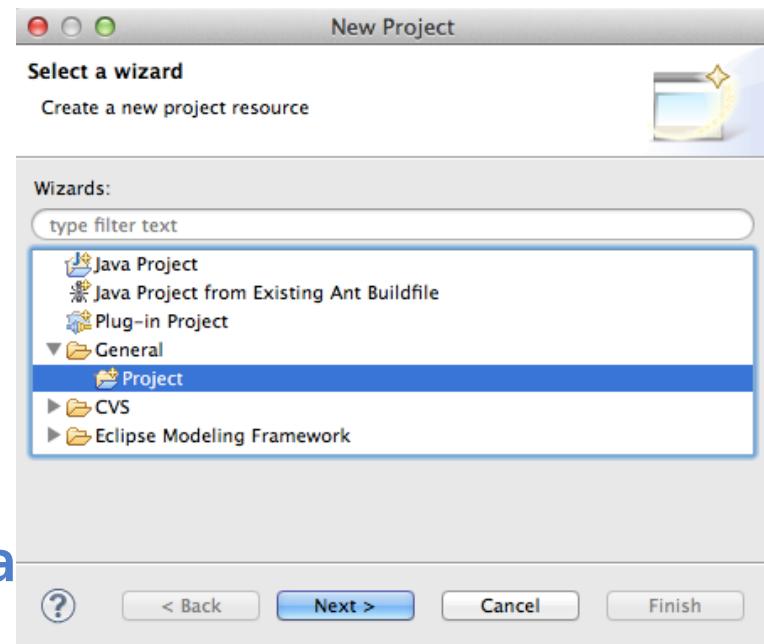
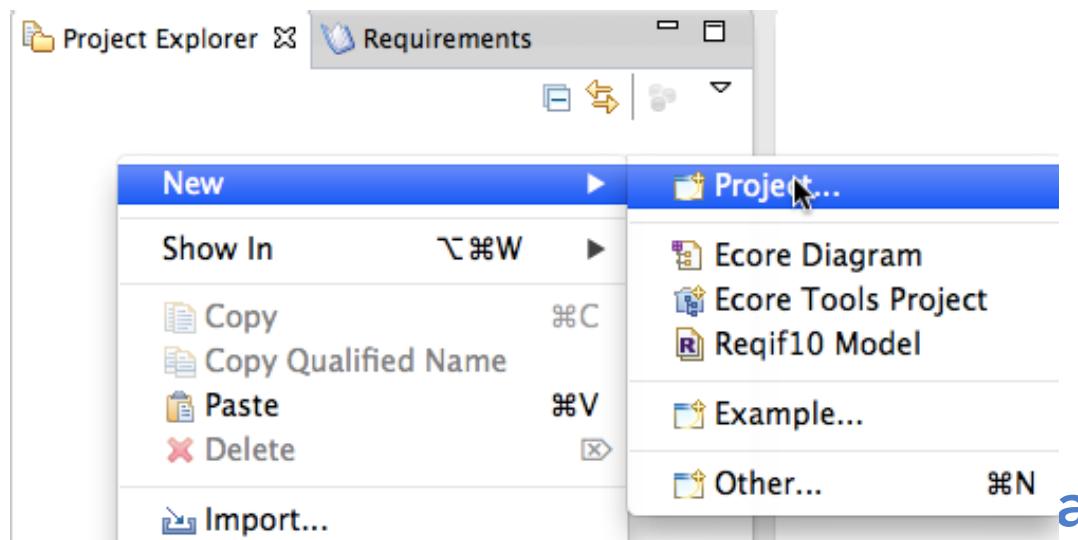
# Retrieve customer requirements

- Generally customer requirements have been edited through a requirement tool (Doors™, Requisite Pro™..) or through a document (Word™, Excel™, OpenOffice)
- We want to retrieve those requirements and visualize them into ReqCycle so that we can easily formalize them, refine them, implement them and verify them with different system development artefacts (other requirements, model elements, documents, source code...)
  - Next slides show how to do that practically



# Eclipse project to host requirements

- We create an Eclipse project to store customer requirements

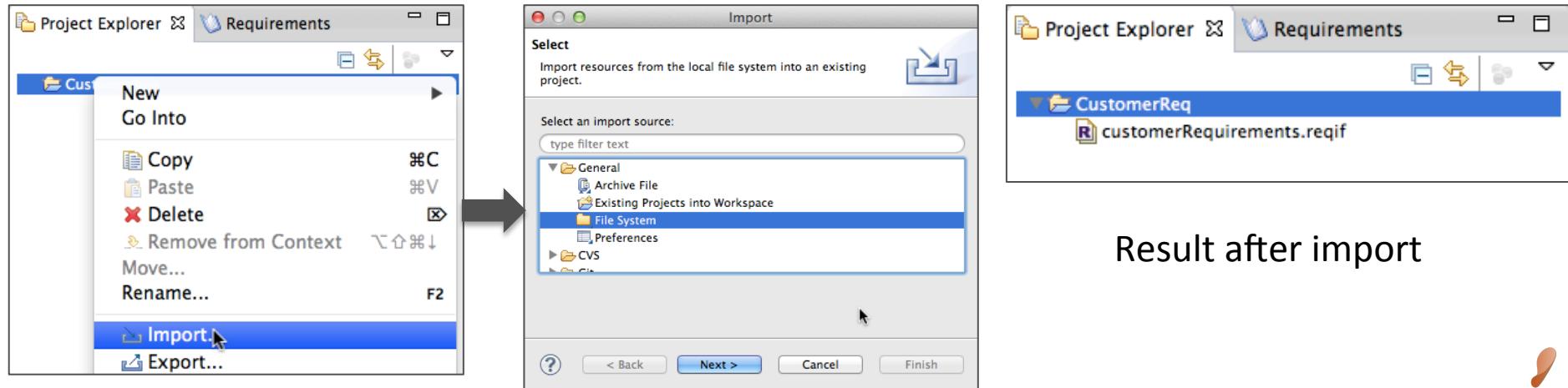


- Give "CustomerReq" as name
- OK – project is created



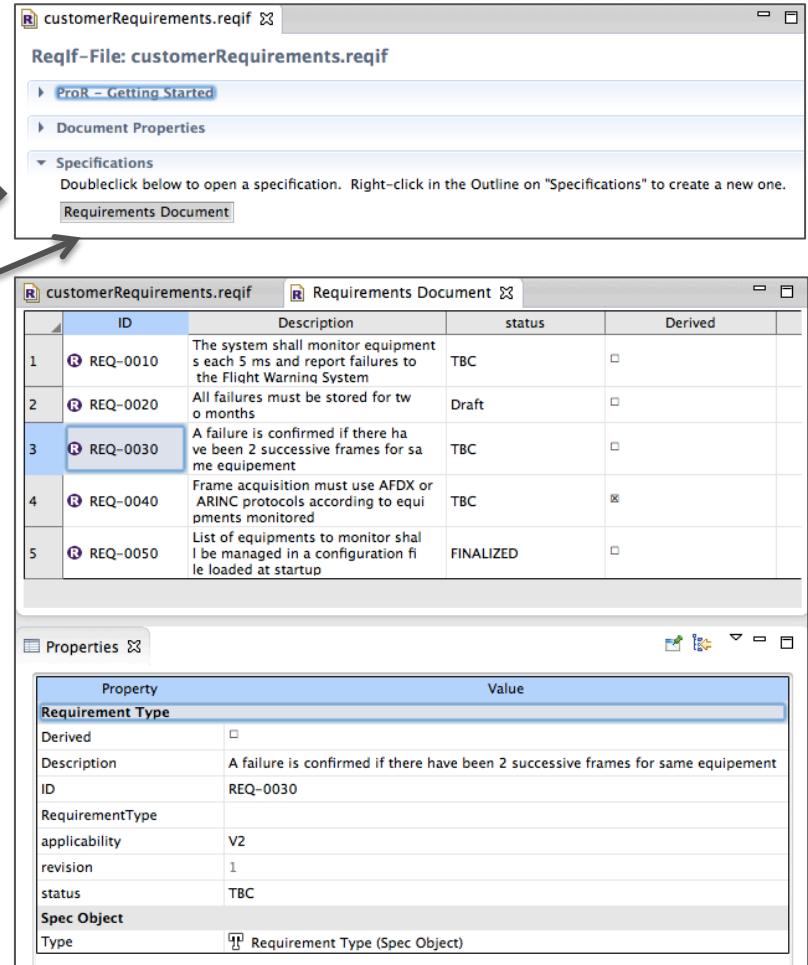
# Store customer requirements

- We start with requirements formalized in ReqIF format
  - ReqIF is OMG standard Interchange format for requirements. All requirement tools should now support ReqIF export
- Select your Eclipse project and then import customer requirements .ReqIF file. Sample file available here:
  - <https://github.com/raphaelfaudou/ReqCycle/blob/master/resources/sample/customerRequirements.reqif>



# Visualize .ReqIF requirements

- Thanks to ProR Eclipse tool you can see your ReqIF requirements in a table
  - Double click on the .reqIF file and ProR editor opens
  - Double click on a ReqIF specification to see the requirements in a custom table
  - Right click to open the properties view and select one requirement -> all requirement attributes are listed



The screenshot shows the ProR Eclipse interface for managing .reqIF requirements. At the top, there's a navigation bar with tabs like 'ProR – Getting Started' and 'Document Properties'. Below that is a section titled 'Specifications' with a note about double-clicking to open specifications or right-clicking to create new ones. A 'Requirements Document' tab is selected. The main area displays a table of requirements:

ID	Description	status	Derived
1 REQ-0010	The system shall monitor equipments each 5 ms and report failures to the Flight Warning System	TBC	<input type="checkbox"/>
2 REQ-0020	All failures must be stored for two months	Draft	<input type="checkbox"/>
3 REQ-0030	A failure is confirmed if there have been 2 successive frames for same equipment	TBC	<input type="checkbox"/>
4 REQ-0040	Frame acquisition must use AFDX or ARINC protocols according to equipments monitored	TBC	<input checked="" type="checkbox"/>
5 REQ-0050	List of equipments to monitor shall be managed in a configuration file loaded at startup	FINALIZED	<input type="checkbox"/>

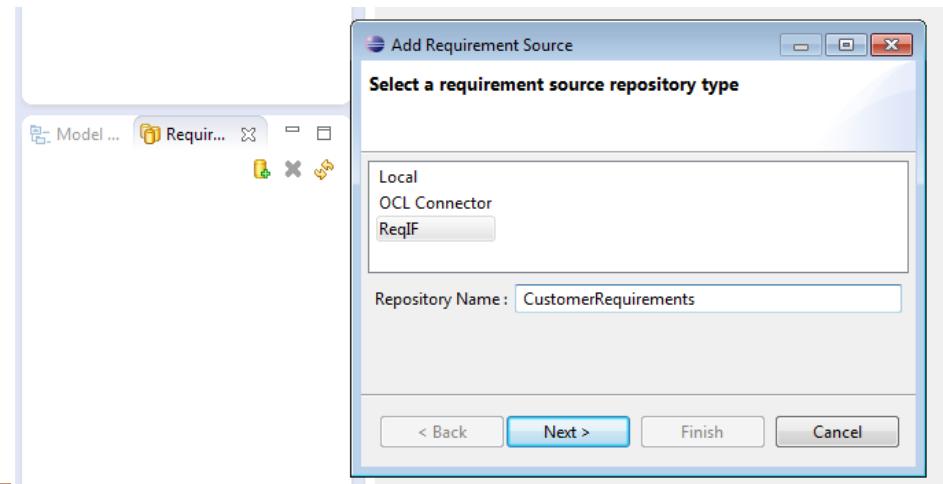
Below the table is a 'Properties' view showing detailed information for requirement ID 3:

Property	Value
Requirement Type	
Derived	<input type="checkbox"/>
Description	A failure is confirmed if there have been 2 successive frames for same equipment
ID	REQ-0030
RequirementType	
applicability	V2
revision	1
status	TBC
Spec Object	
Type	Requirement Type (Spec Object)



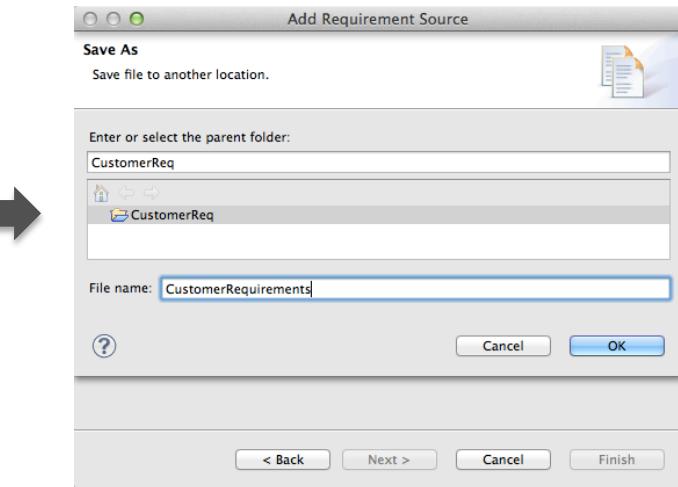
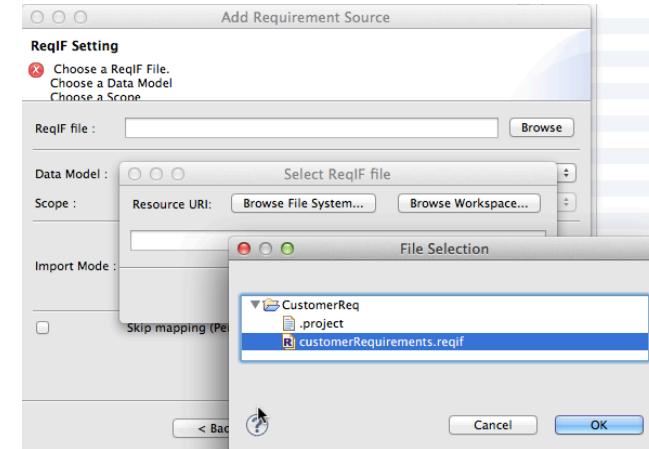
# Import ReqIF requirements in ReqCycle (1)

- Now we want to integrate those requirements with our custom requirement data model (MyReqDM – see step 1)
  - ReqIF file does not contain any links with this data model, so we must perform a mapping (import)
- Go to requirement sources view
  - Add requirement source (+) and choose ReqIF connector
  - Give name for this source (ex: customerRequirements)
  - Next



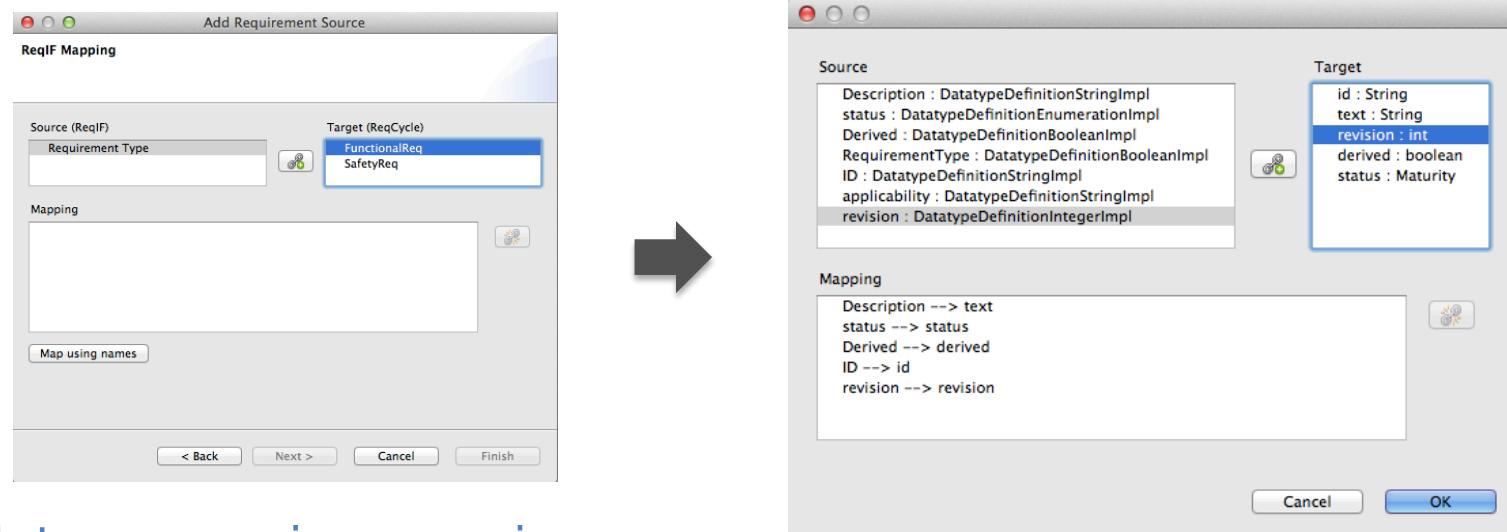
# Import ReqIF requirements in ReqCycle (2)

- Select .reqIF file with “browse” and then “browse workspace”
- Select data model and scope
  - “MyReqDM” for data model
  - “Customer” for scope
- Select destination file
  - Select project and then put a name (same than source)
  - Note: will be filled automatically in a future release
- Next



# Import ReqIF requirements in ReqCycle (3)

- Map ReqIF specifications to a requirement type
  - For instance « Requirement Type » to « FunctionalReq »
  - Click the « map » button (between source and target): a new screen opens that lets you map attributes

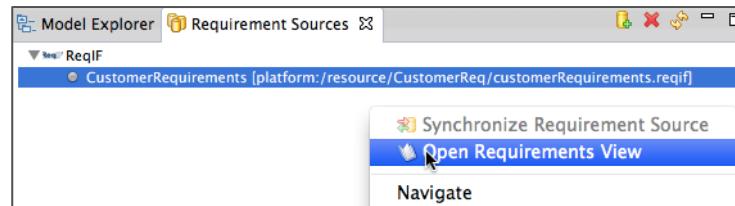


- Note concerning mapping:
  - In this example, « Applicability » has no mapping in target: you can cancel import and update your data model with new attribute 'applicability' (string) and then do the mapping (import) again to get applicability mapped

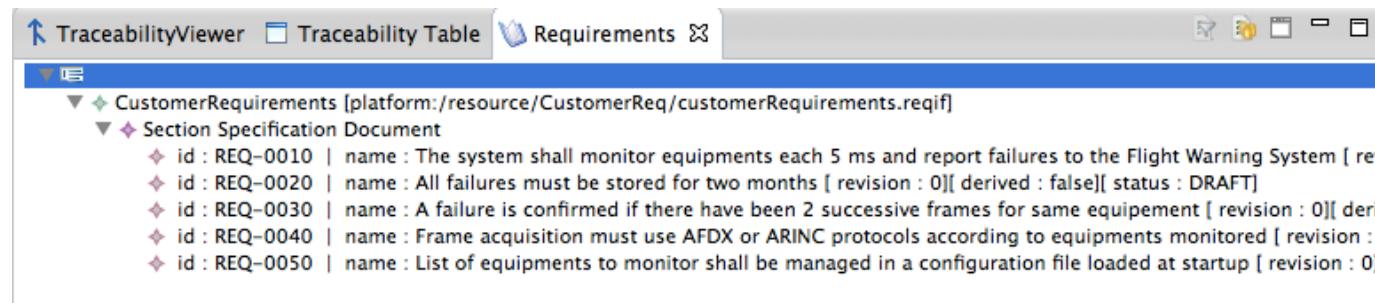


# See mapping result

- « CustomerRequirements » source has been created
  - Right click > « open in Requirement view »



- A Requirements treeview (readOnly) is created and that can be expanded

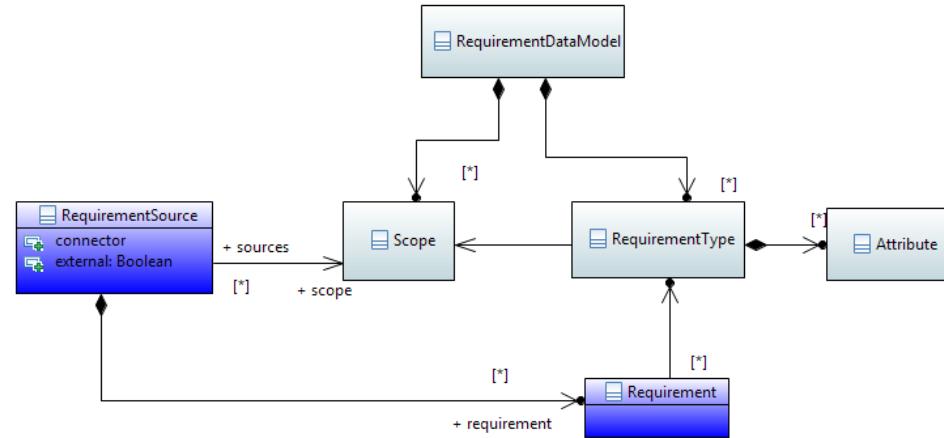


- Note: this editor is poor compared to what ProR can bring but ReqCycle needs an editor able to show all requirements (not only ReqIF) and linked to requirement data model: so we can not use ProR editor for now - This is an area where we intend to find some synergy with ProR team



# And now?

- Now we have integrated some customer requirements in our custom data model



- If you need to integrate more customer requirements, you can create a new requirement source
  - New connectors for external sources are planned for future releases: Word, Excel, openOffice, Topcased Req
  - If needed you can update your data model (new scopes, new requirements types, attributes) with current limits: labels cannot yet be renamed (later) and no deletion for now (some rules still to specify)
- Then time has come to trace our customer requirements

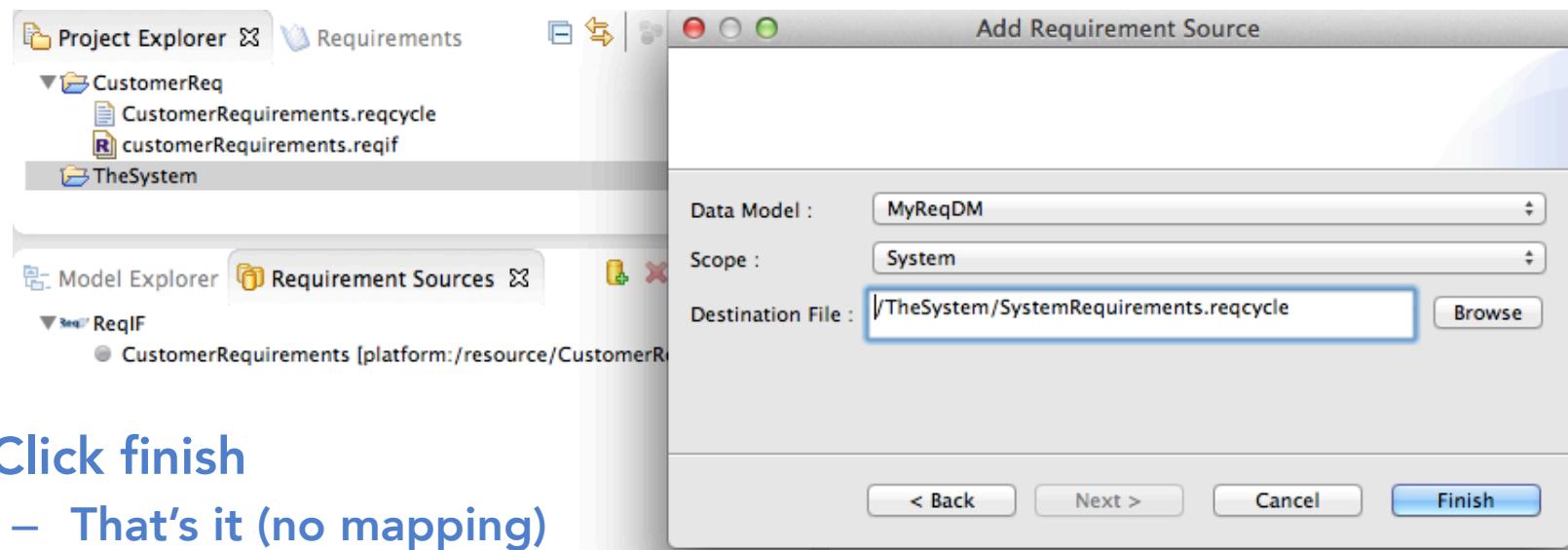


- If customer requirements already provide good definition of target system, you can switch to modelling world to refine and implement those requirements. This is first option.
  - To do that, you can jump to step 3 – configure traceability
- Else, you might consider that you need to refine customer requirements through System textual requirements. This is second option.
  - Next slides show how to create your own textual requirements



# Create textual requirements (1)

- Create a new Eclipse project “TheSystem” to differentiate customer requirements and System requirements that you are about to define
- Go to Requirement sources view and add source
  - Choose “Local” requirement source connector
  - Fill input parameters

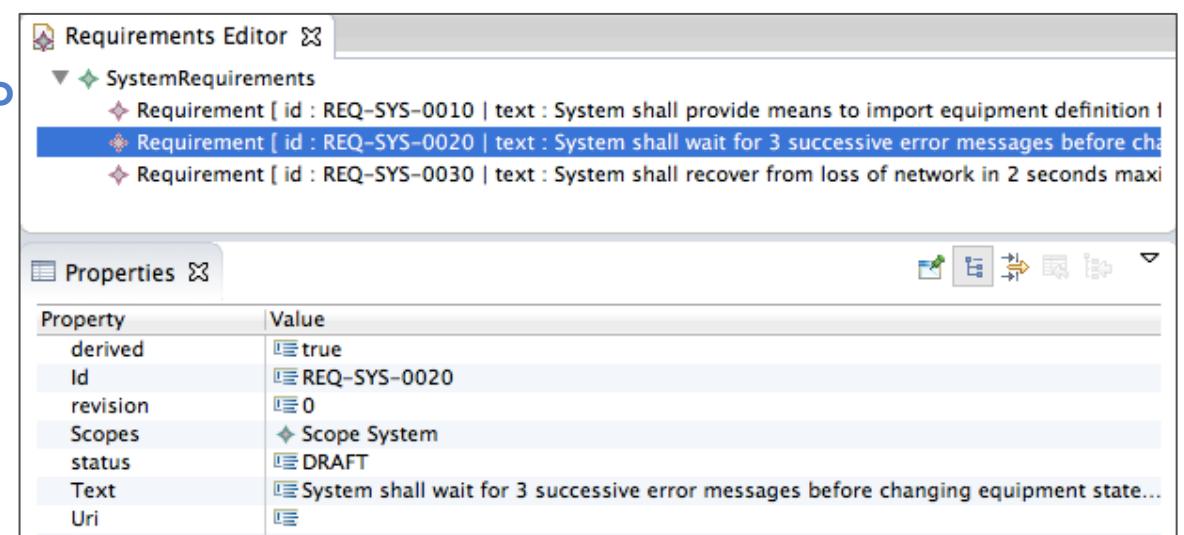
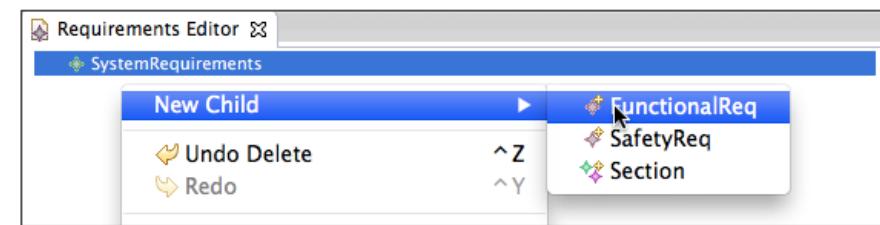
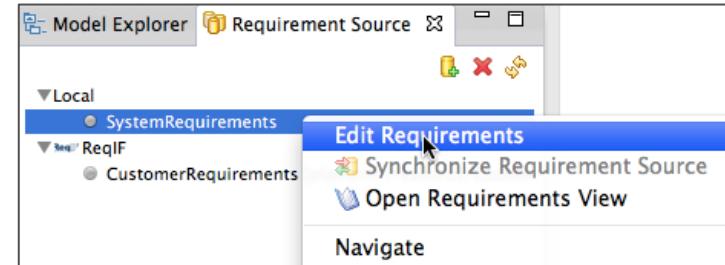


- Click finish
  - That's it (no mapping)



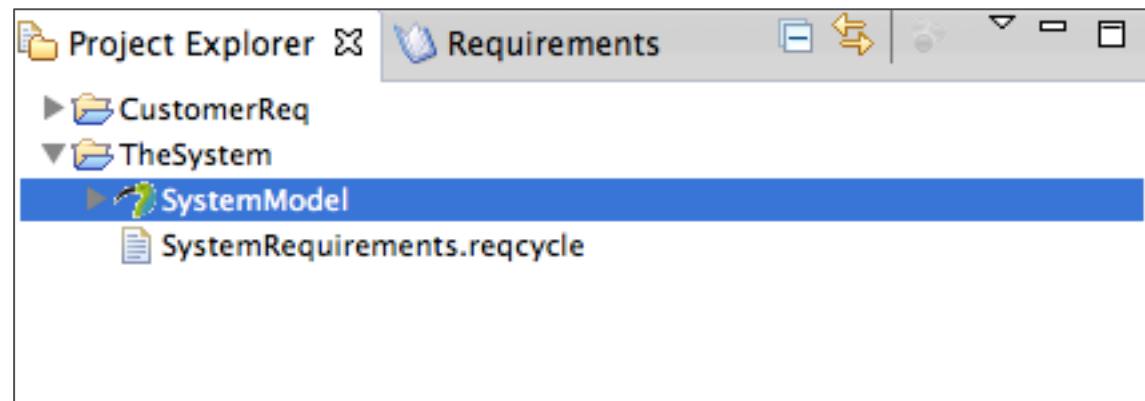
# Create textual requirements (2)

- Now you can edit those requirements (right click)
- A minimalist Requirement editor lets you create requirements according to types defined in data model
- Attribute values are set through the property view
  - Note: contextually to selected requirement



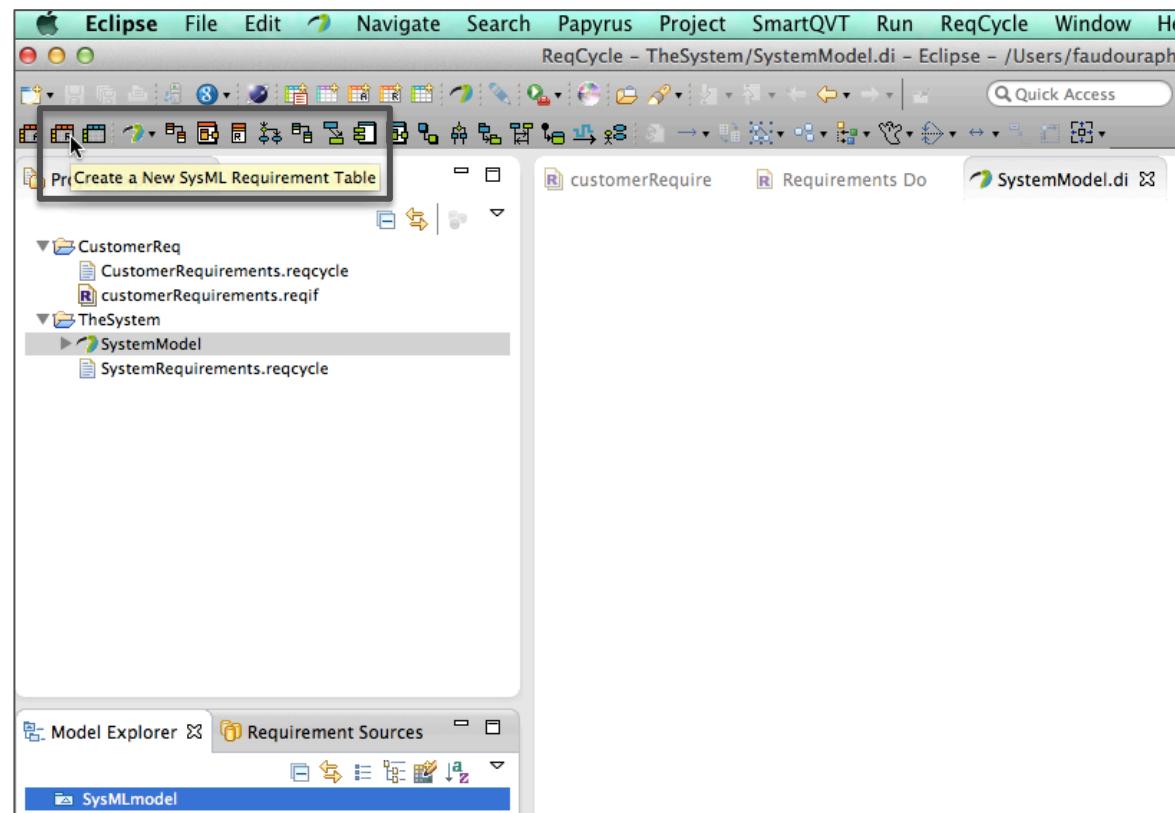
# Import SysML requirements (1)

- Can import textual requirements created in SysML
  - OCL requirement connector can extract them
- Let us create a Papyrus SysML model
  - Select “TheSystem” project, right click>new other... papyrus model
  - Give name “SystemModel”
  - Select “SysML” model type and finish



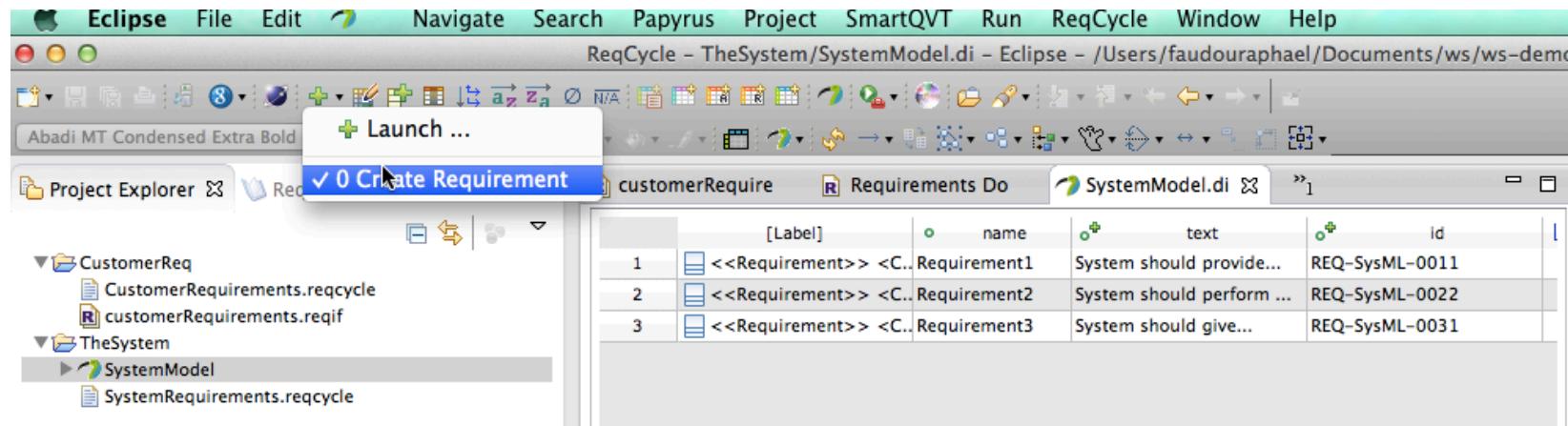
# Import SysML requirements (2)

- Open papyrus model, select root node in model explorer view and create a requirements table

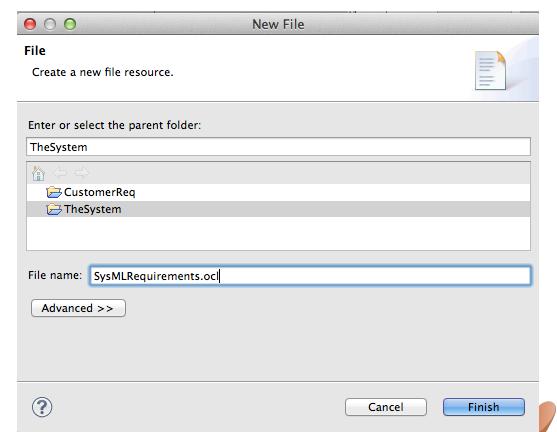


# Import SysML requirements (3)

- Create requirements and edit id and text



- Then let us create the OCL file that will define rules to extract requirements
  - Select « TheSystem » project
  - Right click new file
  - Give SysMLRequirements.ocl name



# Import SysML requirements (4)

- Edit OCL file as below
  - Rules extract requirements and their attributes (as defined in requirement datamodel)

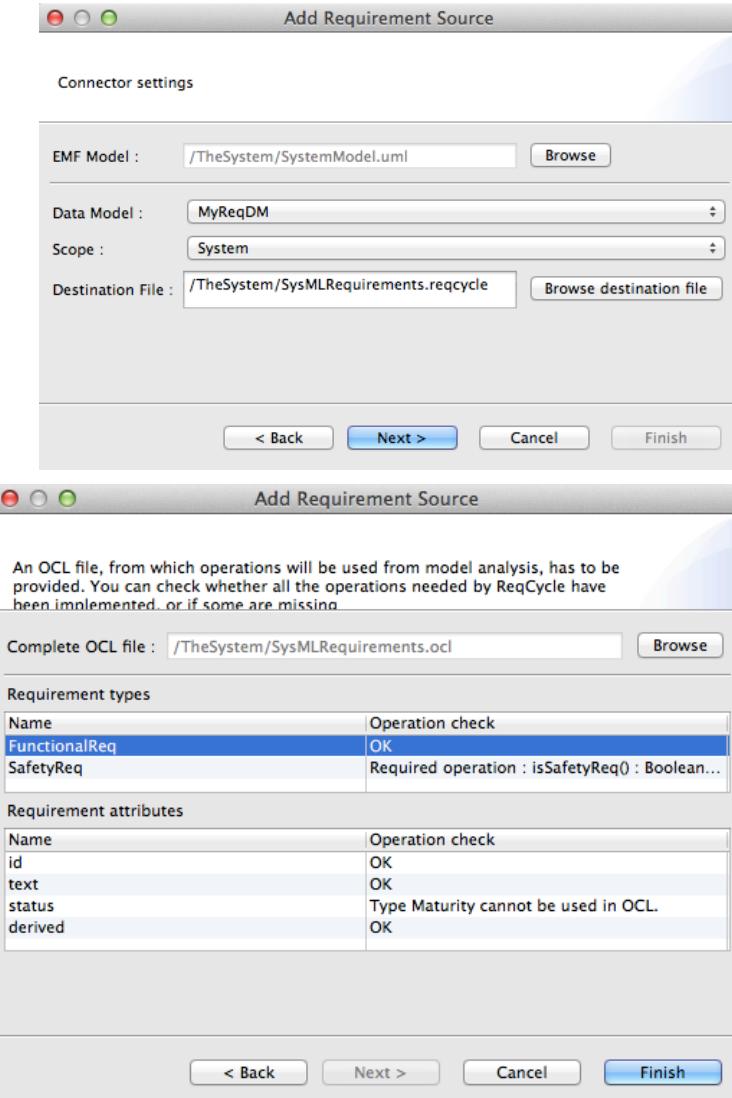
```
import uml : 'http://www.eclipse.org/uml2/4.0.0/UML#/'  
import 'http://www.eclipse.org/papyrus/0.7.0/SysML#/'  
  
context Requirement  
def: isFunctionalReq() : Boolean = true  
def: getId(): String = self.id  
def: getText() : String = self.text  
def: getDerived() : Boolean = false  
def: getUri() : String = 'not managed'  
def: getStatus() : String = 'statut'
```

- Note: enumerations are not yet managed



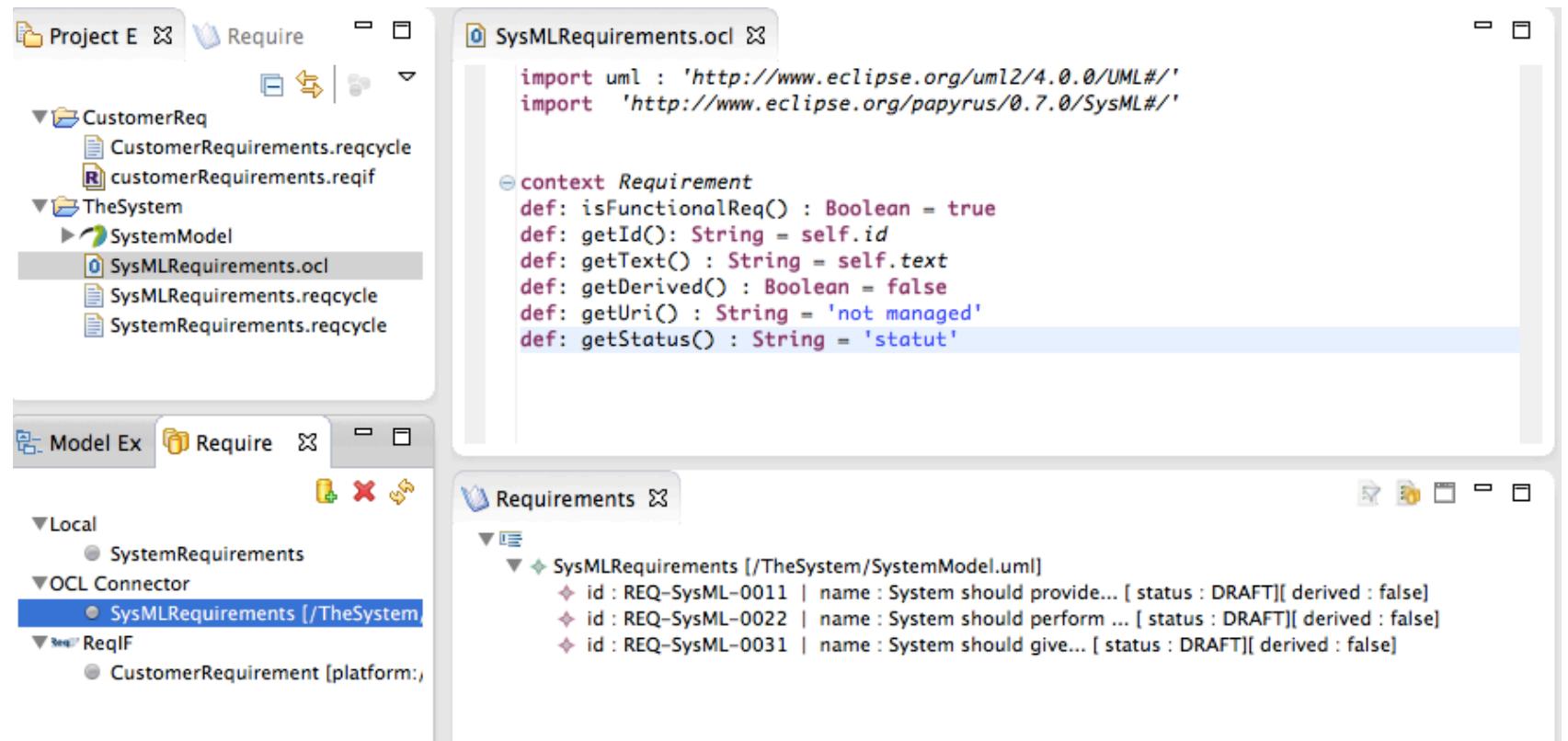
# Import SysML requirements (5)

- Create the OCL requirement source
  - “SysMLRequirements” as name
  - “SystemModel.uml” as EMF model
  - “MySystem>” project to host destination file
  - Next
  - Select “SysMLRequirements.ocl”
  - Click “FunctionalReq”
    - All attributes are listed
    - OK is put if there is a correct OCL definition
    - Recall: enum attributes are not yet managed
- Click Finish, that's it



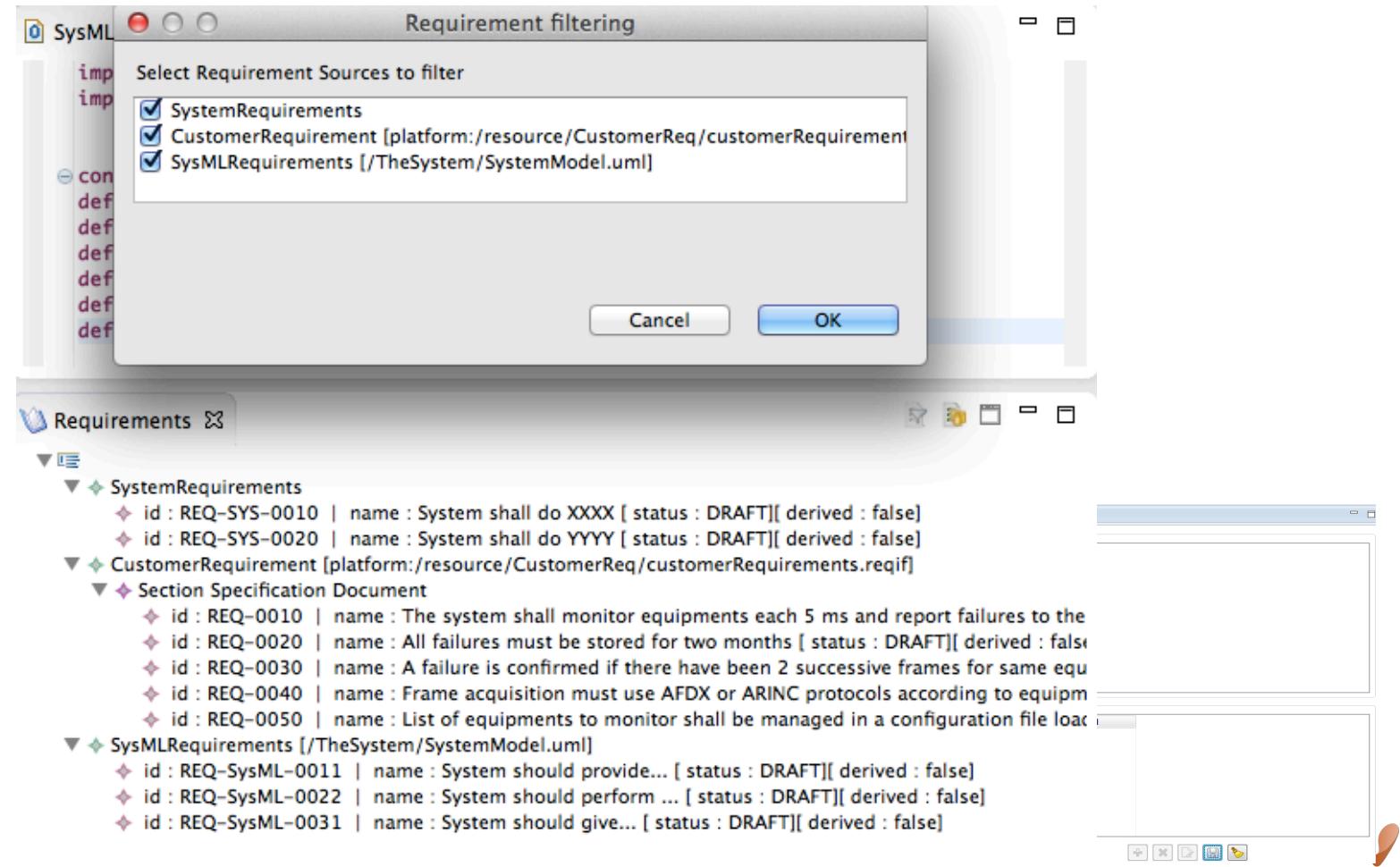
# Import SysML requirements (6)

- Now open your requirement source (contextual menu)
- Requirements are there



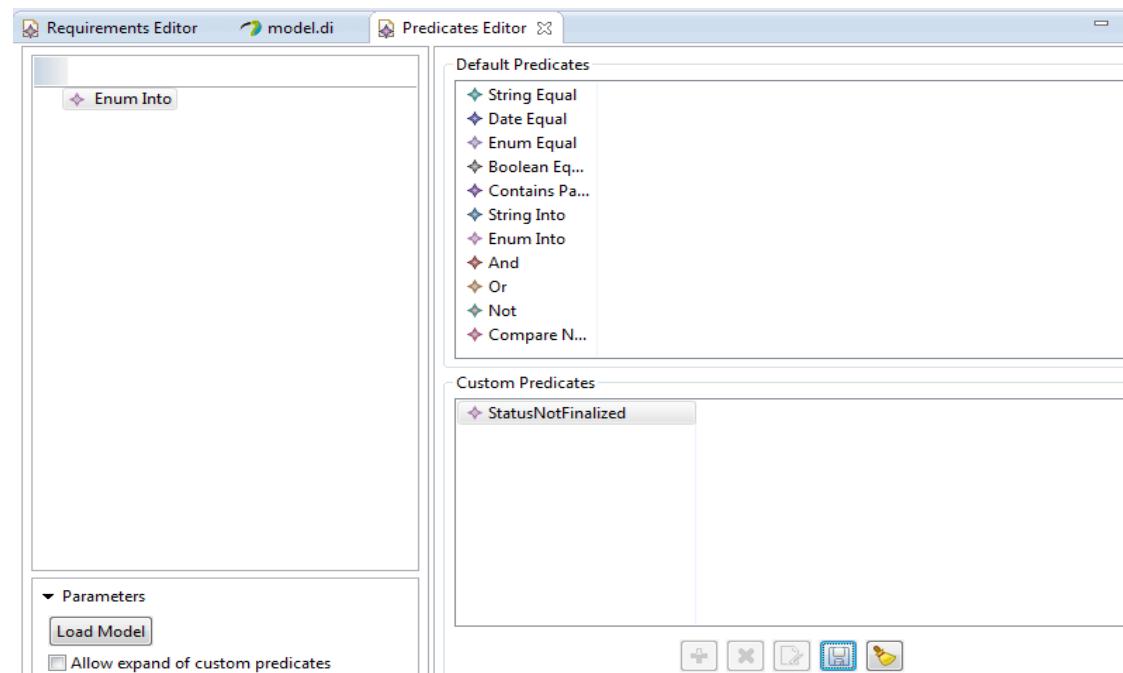
# Queries/filters on requirements

- We can already filter requirements on their source



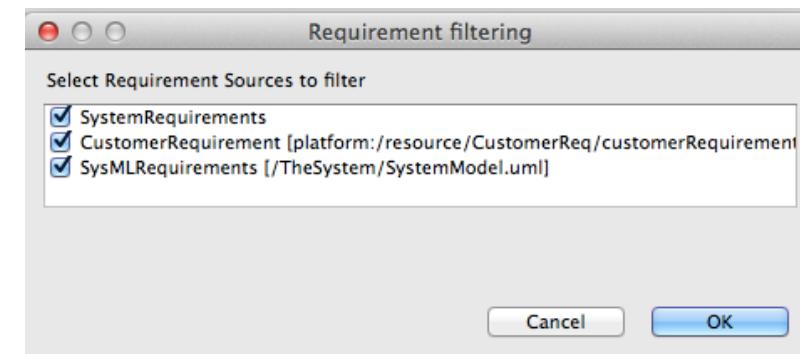
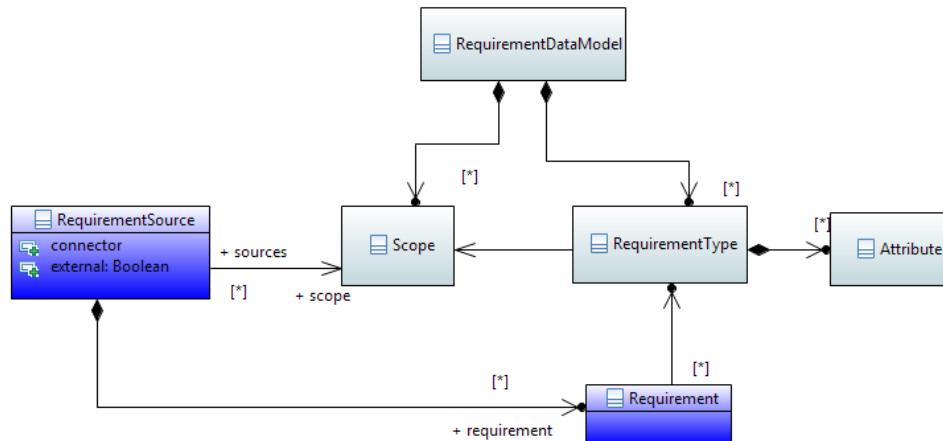
# Queries/filters on requirements

- In a future release we will have ability to define predicates (queries) to filter views
  - Feature partially developed
  - Requires more tests



# Step 2 conclusion

- Requirements (external or local, ReqIF or SysML) can be imported or created in ReqCycle so that they conform to the requirement data model defined in step 1
  - We can filter them and show them in requirements views



- Now we are going to show in step 3 how to trace them





# 3rd step – configure traceability links

Concepts and screenshots

Raphaël Faudou

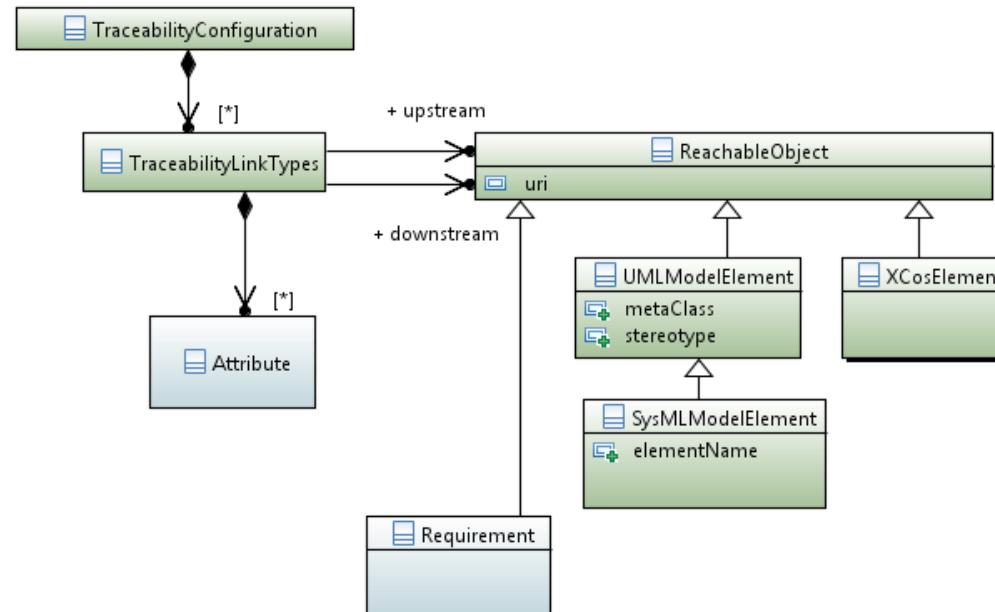
[raphael.faudou@samarès-engineering.com](mailto:raphael.faudou@samarès-engineering.com)

- 2 kinds of traceability engines: generic and specific
- Generic engine able to create any link
  - Between reachable data and reachable data (see later)
  - Can restrict possible links by configuration or checks
  - Currently requires specific GUI for Drag and Drop support
- Specific analysis engines able to capture existing links
  - One by existing tool/language
  - First targets are UML/SysML and Java, next is Xcos
  - Adding a target consists in implementing two interfaces
- Note: 3<sup>rd</sup> step focuses on link creation (generic engine), capture of existing links is presented in step 4.



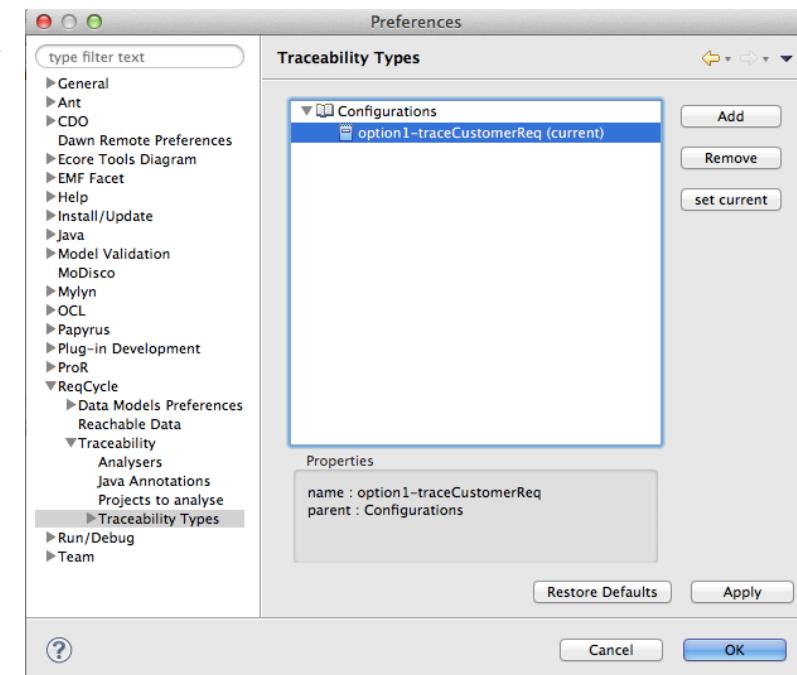
# Traceability configuration

- By default, reachable objects known by ReqCycle are
  - Requirements, UML, SysML and Xcos model elements
- Can capture other reachable objects with new plugins
  - Have to implement 3 extensions (interfaces)



# Configuration to trace customer req.

- In preferences>ReqCycle>Traceability>Traceability types
  - Select “Configurations” root node and click “add”
  - Give name (for instance “option1-traceCustomerReq”) - OK
  - Click “set current”
    - To activate it
  - Click “apply”
    - To register data

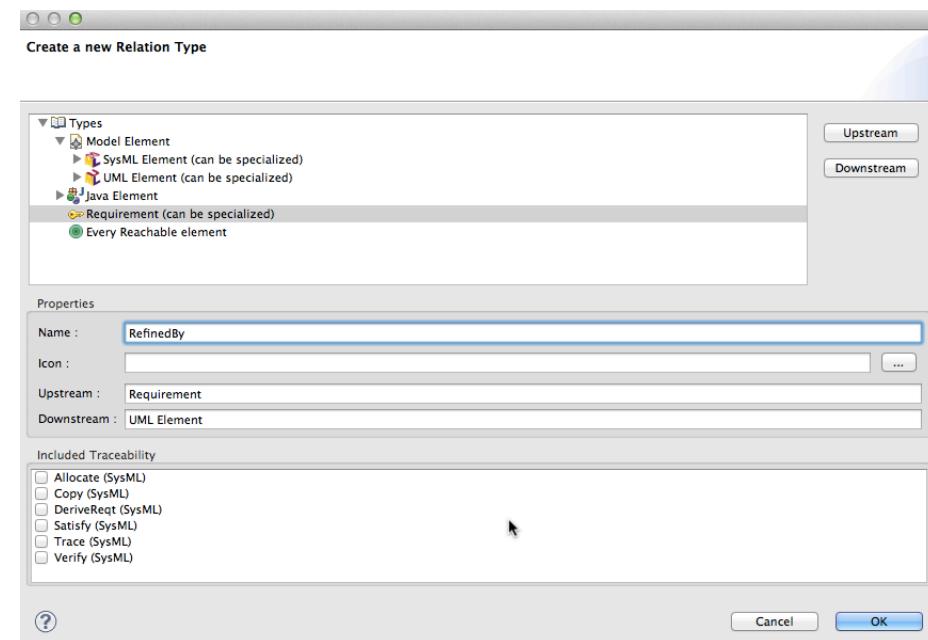


- To trace our customer requirements we want to define several kinds of links. For instance:
  - “RefinedBy” that will concern UML (and SysML) elements
  - “ImplementedBy” that can concern any reachable data
  - “VerifiedBy” that only concerns SysML test cases
- Next slides show how to do that



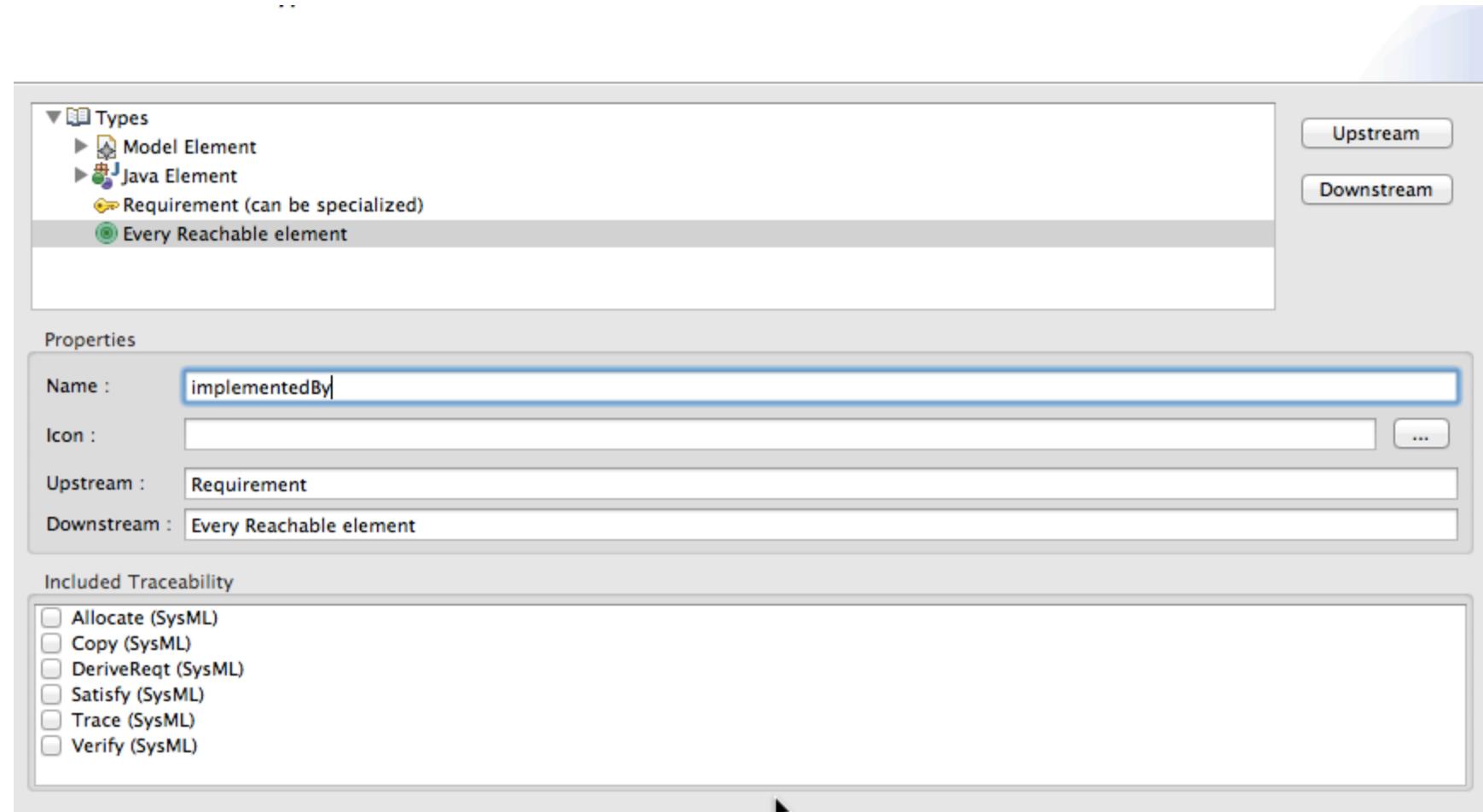
# Create “RefinedBy” traceability link type

- Select upstream data type - click “Upstream” button
  - Here we choose “Requirement”
- Select downstream data type - click “Downstream” button
  - Here we choose “UML element”



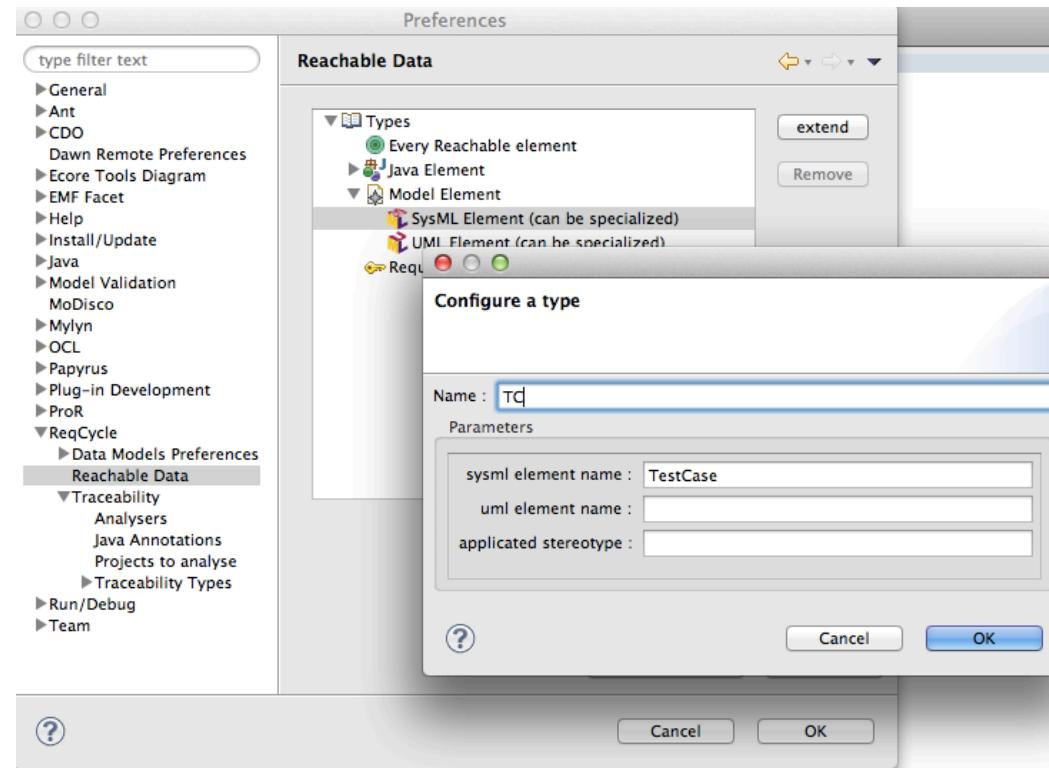
- Give name
  - “RefinedBy”
- OK and then Apply

# “ImplementedBy” traceability type

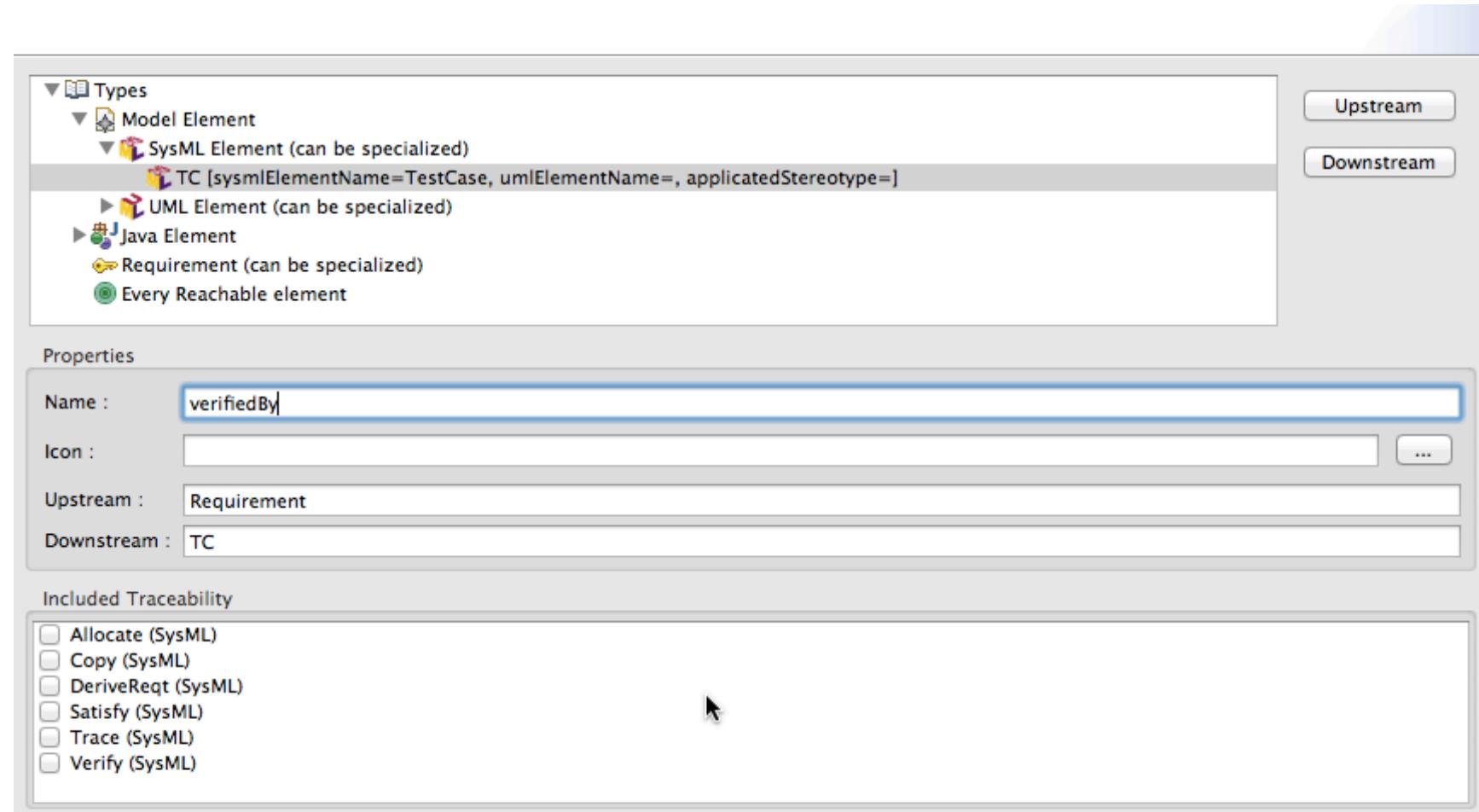


# Create “VerifiedBy” traceability type (1)

- “VerifiedBy” first requires defining “TC” reachable data
  - Go to preferences>ReqCycle>Reachable Data
  - Select SysML Element and click “extend” to define “TC”



# Create “VerifiedBy” traceability type (2)

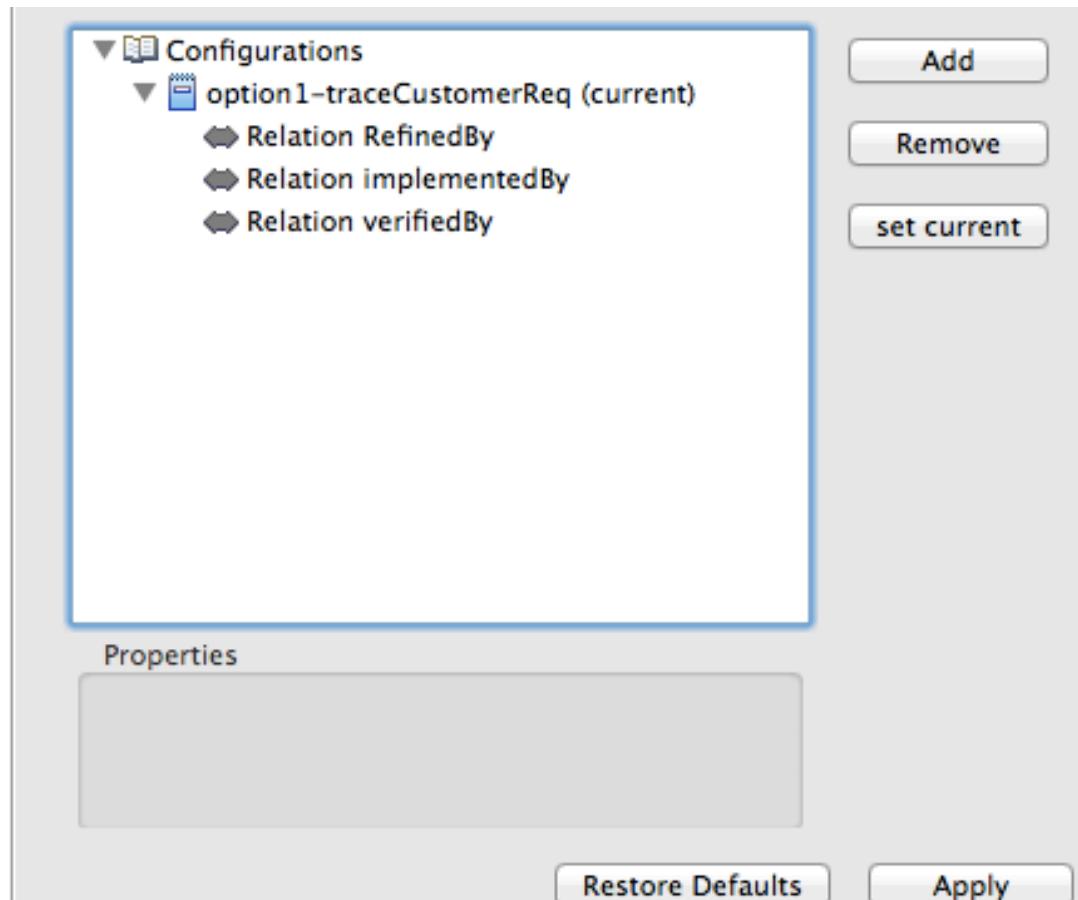


- OK and then Apply



# Summary of “traceCustomerReq”

- ▶ Ant
- ▶ CDO
- ▶ Dawn Remote Preferences
- ▶ Ecore Tools Diagram
- ▶ EMF Facet
- ▶ Help
- ▶ Install/Update
- ▶ Java
- ▶ Model Validation
- ▶ MoDisco
- ▶ Mylyn
- ▶ OCL
- ▶ Papyrus
- ▶ Plug-in Development
- ▶ ProR
- ▼ ReqCycle
  - ▶ Data Models Preferences
  - ▶ Reachable Data
  - ▼ Traceability
    - ▶ Analysers
    - ▶ Java Annotations
    - ▶ Projects to analyse
    - ▶ Traceability Types
- ▶ Run/Debug



- Now let us create traceability links that conform to this configuration



# Create traceability links (1)

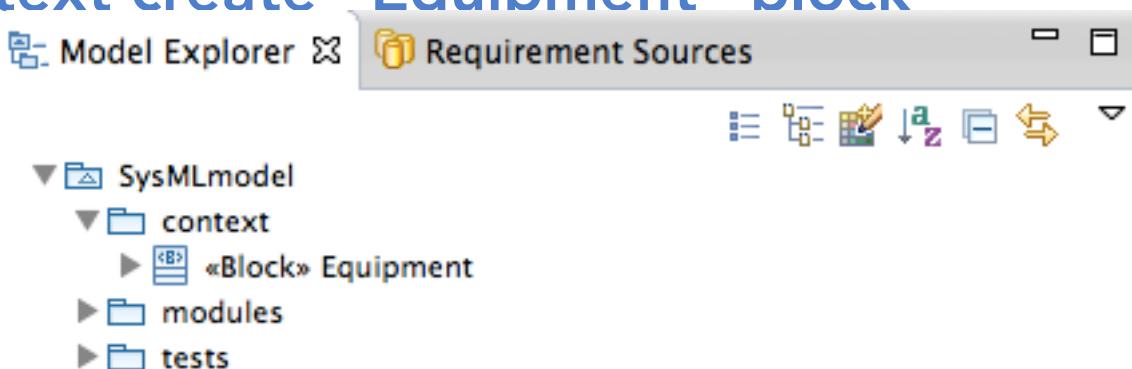


- Now that we have defined and applied a configuration and set it “current”, we can trace links
- We will use our Papyrus “System model”
- Note: go to step 2 (import sysML requirement)s to see how to create a SysML model



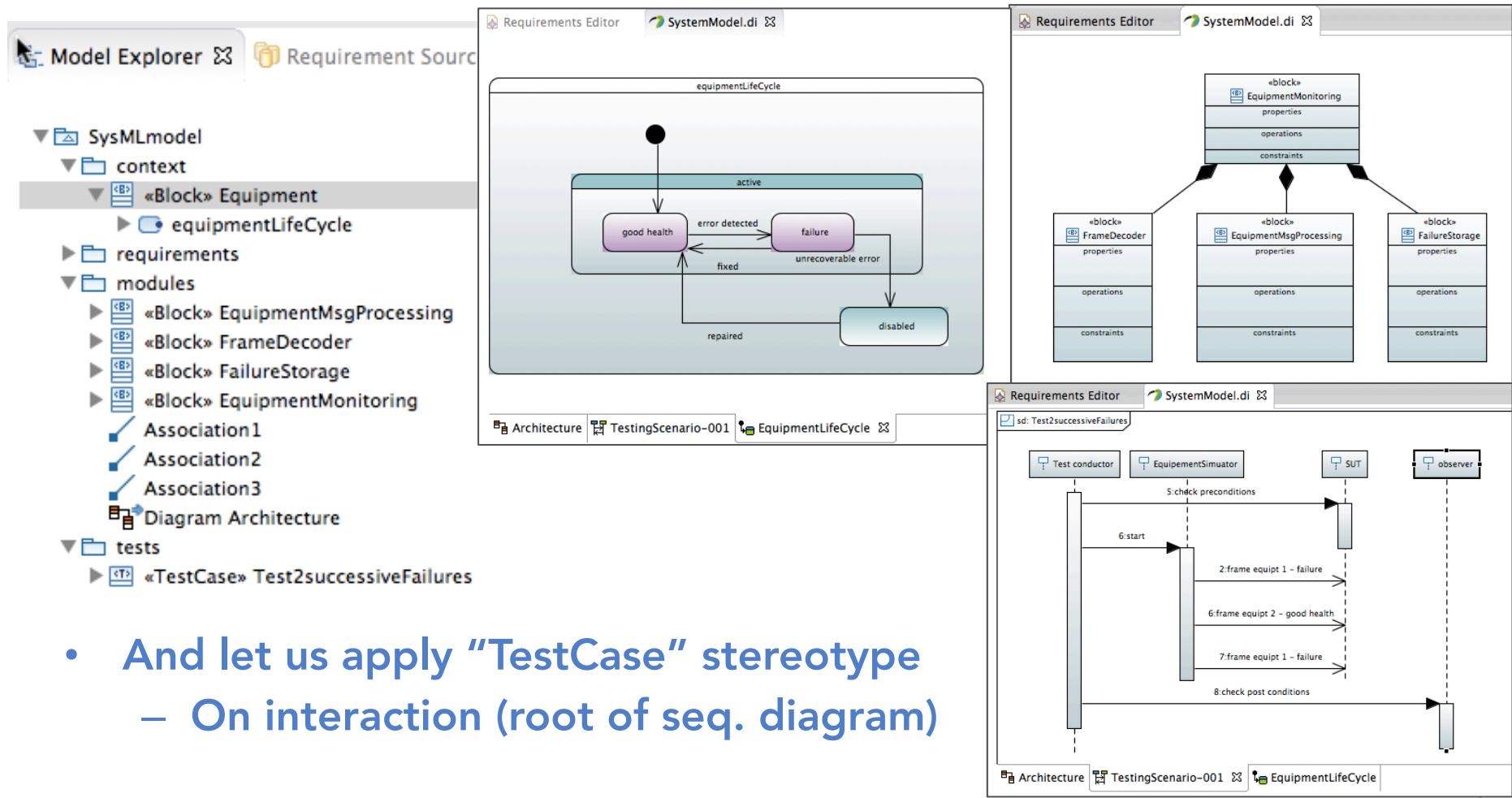
# Create traceability links (2)

- Let us create 3 packages
  - “context” to refine equipment lifecycle
  - “modules” to formalize logical architecture
  - “tests” to formalize a few testing scenarios
- In context create “Equipment” block



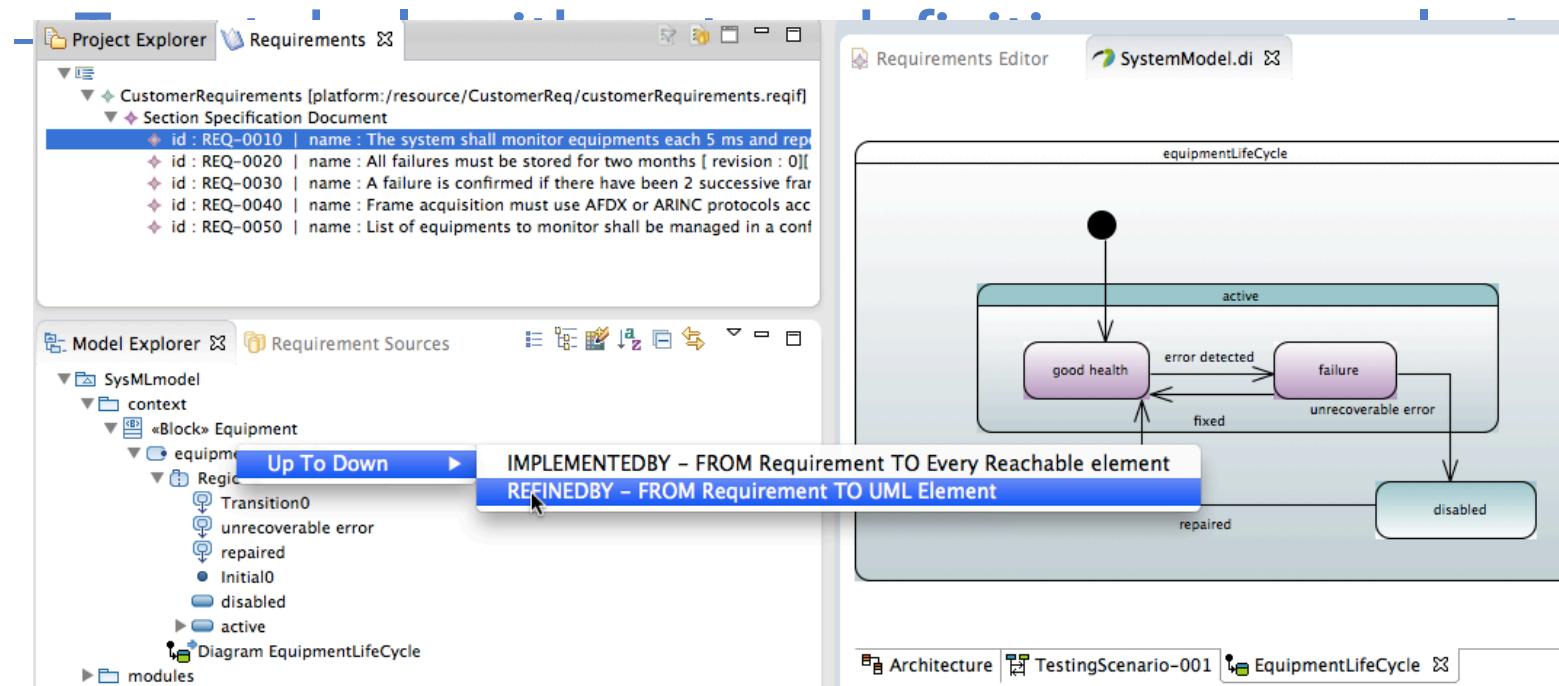
# Create traceability links (3)

- Let us create following diagrams to get some contents



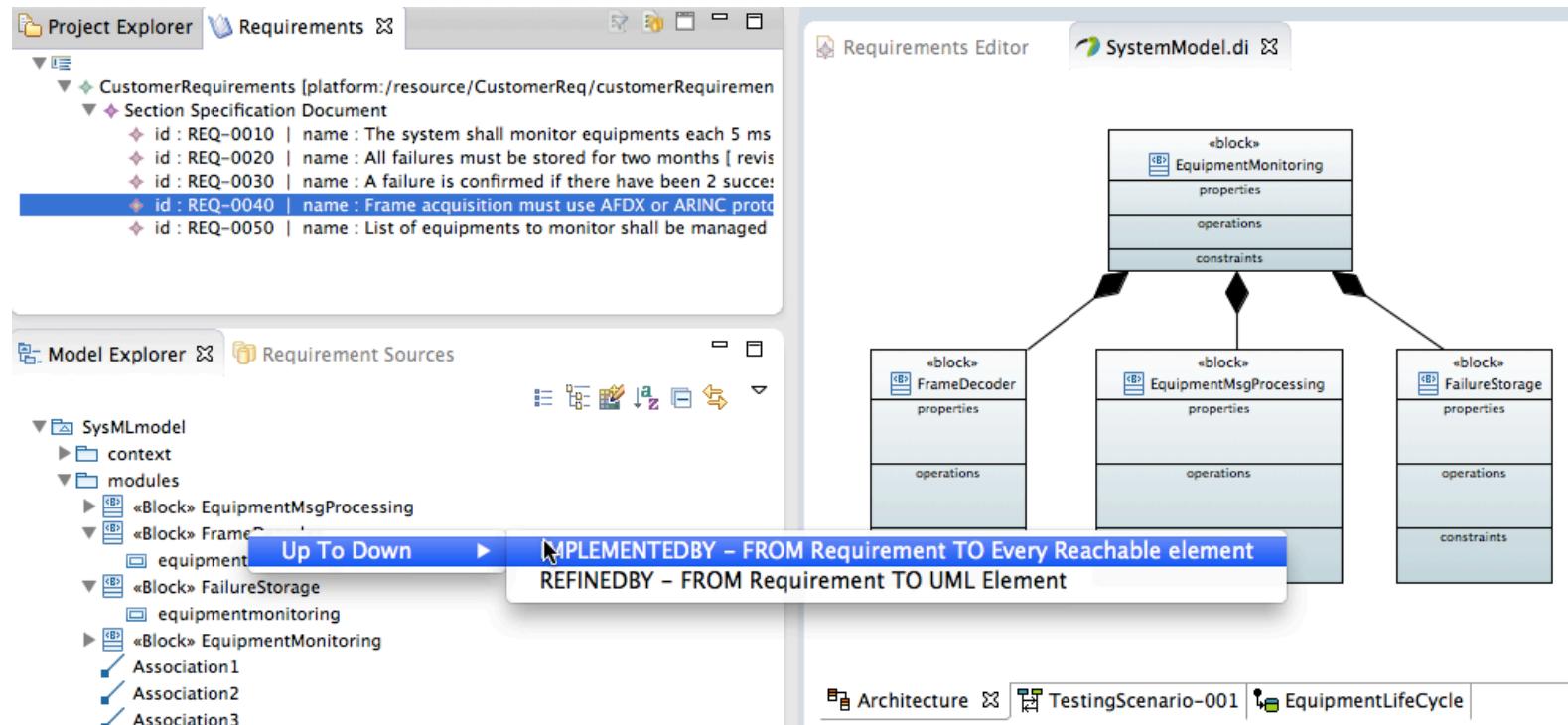
# Create traceability links (4)

- D&D one customer requirement on model explorer
  - For instance “REQ-0010” on “equipmentLifeCycle”
  - As target element is reachable and UML, 2 links are available



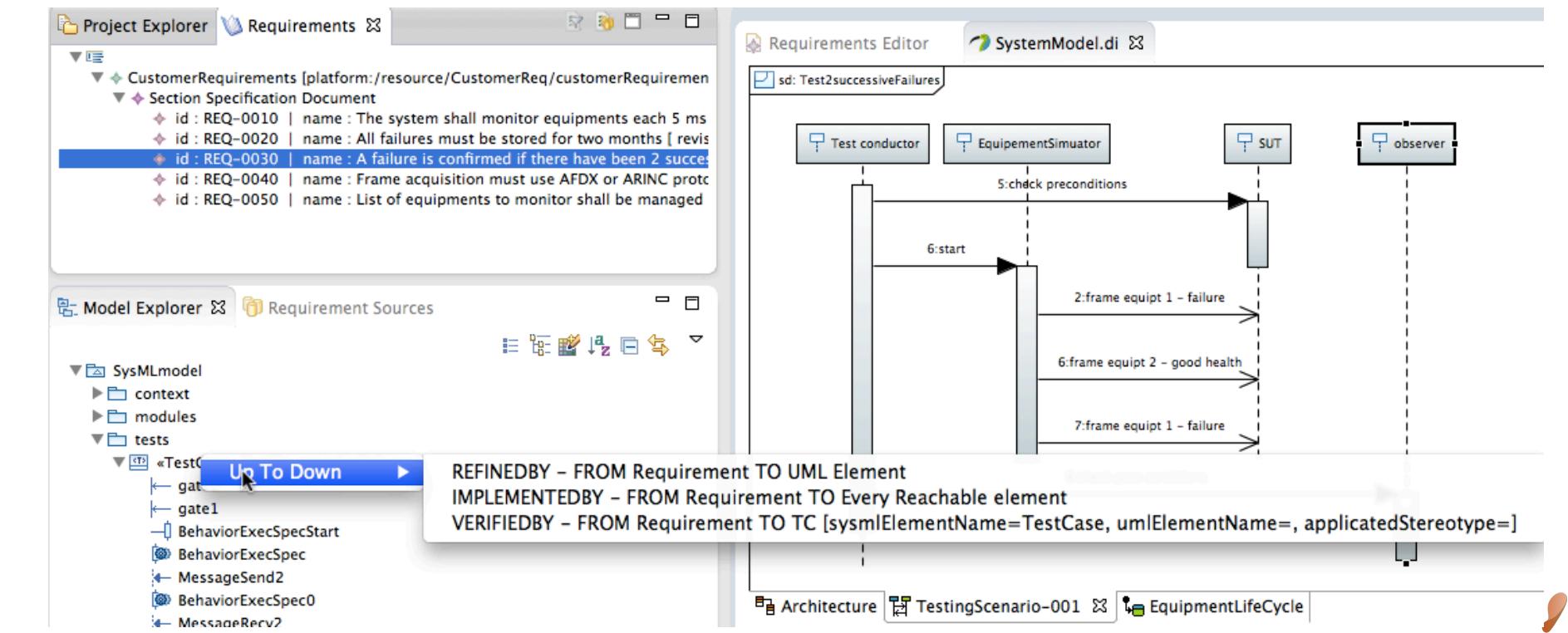
# Create traceability links (5)

- Now let us create an “implementedBy” link
  - For instance “REQ-0040” on “FrameDecoder” block



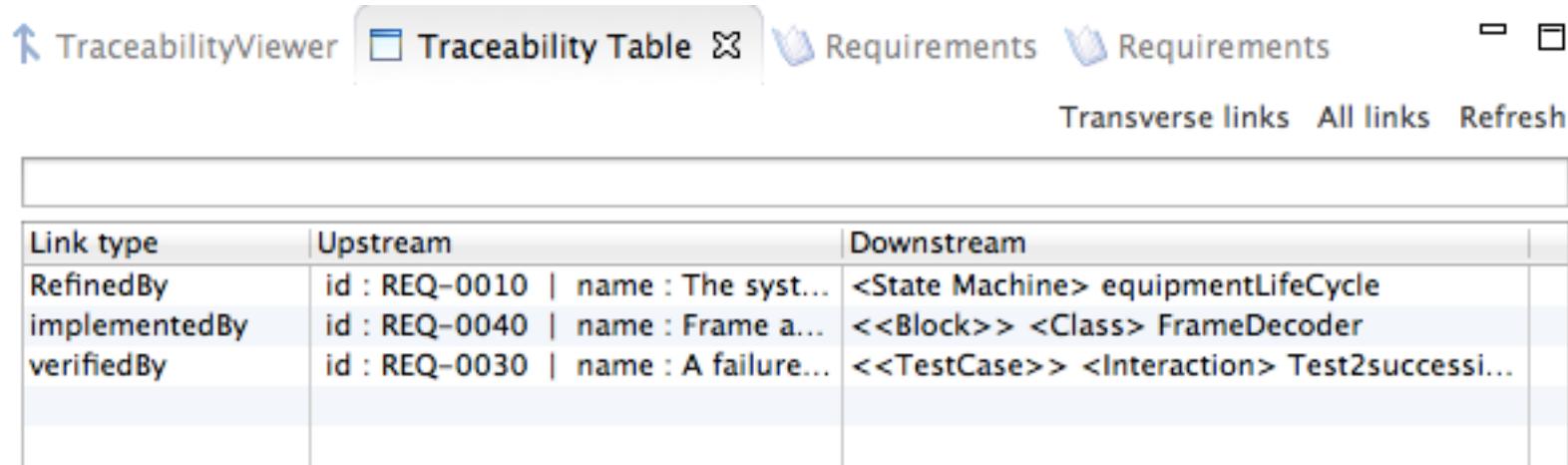
# Create traceability links (6)

- Now let us create a “verifiedBy” link
  - For instance “REQ-0030” on “Test2successiveFailures” TC
  - You can notice a new “VerifiedBy” link available...



# See and manage traceability links

- So far we have traced 3 customer requirements



The screenshot shows a software interface titled "TraceabilityViewer". At the top, there are tabs: "TraceabilityViewer" (selected), "Traceability Table" (highlighted with a blue border), and two "Requirements" tabs. Below the tabs are three buttons: "Transverse links", "All links", and "Refresh". The main area is a table with three columns: "Link type", "Upstream", and "Downstream". The table contains three rows of data:

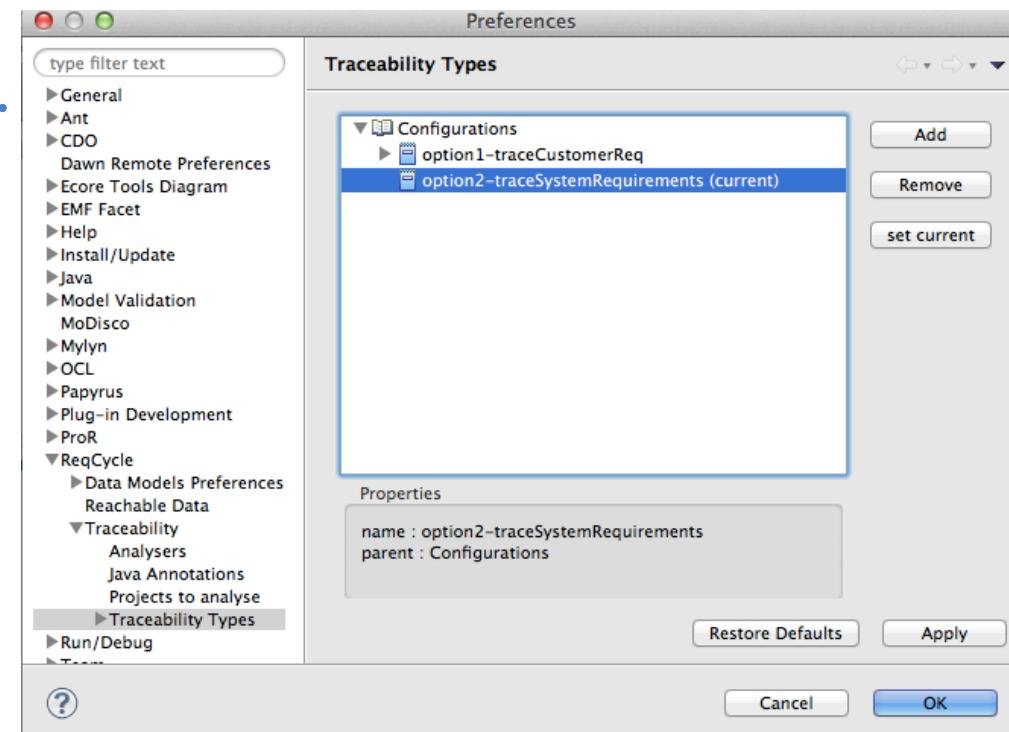
Link type	Upstream	Downstream
RefinedBy	id : REQ-0010   name : The syst...	<State Machine> equipmentLifeCycle
implementedBy	id : REQ-0040   name : Frame a...	<<Block>> <Class> FrameDecoder
verifiedBy	id : REQ-0030   name : A failure...	<<TestCase>> <Interaction> Test2successi...

- You can jump to step 4 to see how to visualize traceability links created or captured, in a table or a treeview...  
... or you can continue step 3 to see how to refine customer requirements into textual system requirements (option 2)



# Configuration to trace System req.

- In preferences>ReqCycle>Traceability>Traceability types
  - Select “Configurations” root node and click “add”
  - Give name (for instance “option2-traceSystemReq”) - OK
  - Click “set current”
    - To activate this conf.
  - Click “apply”
    - To register data



# Traceability links for System Req.

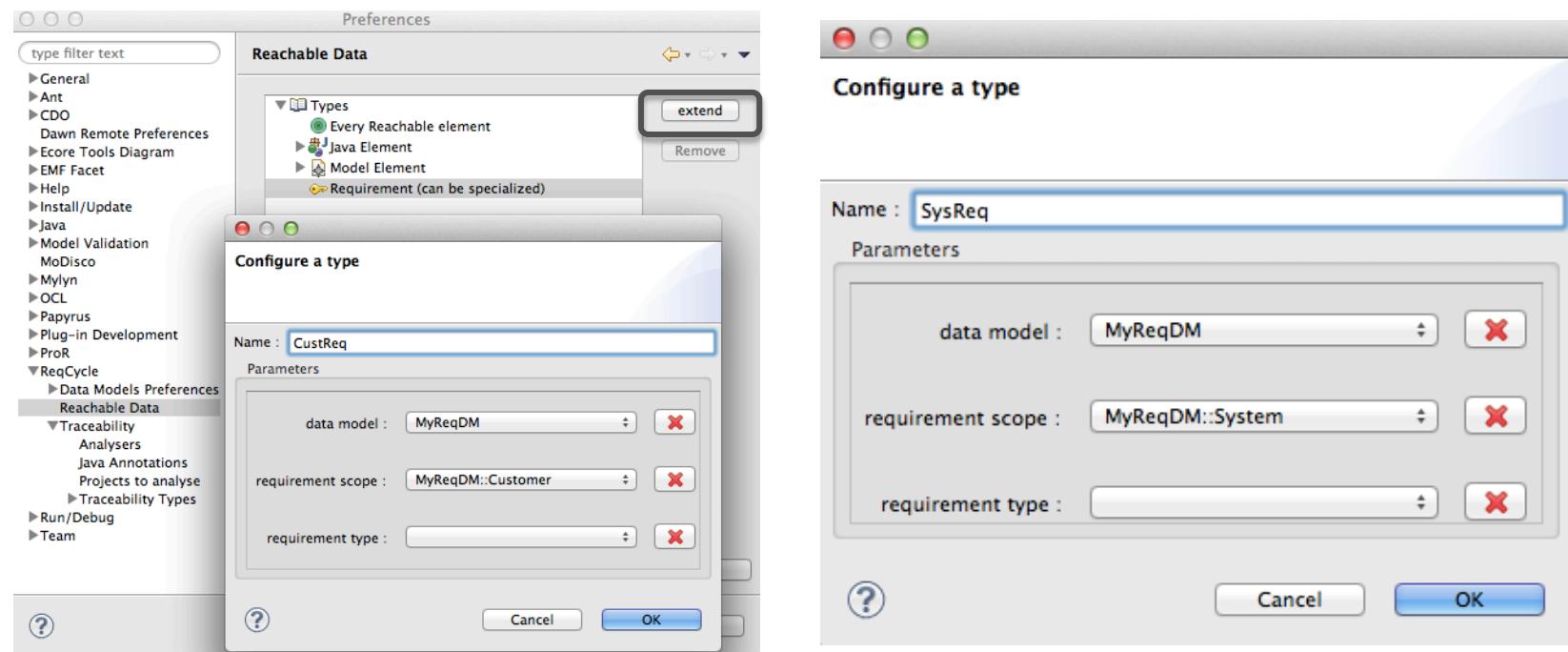


- To trace our system requirements we want to define several kinds of links. For instance:
  - “Refines” that will concern Customer requirements
  - “ImplementedBy” that can concern any reachable data
  - “VerifiedBy” that only concerns SysML test cases
- Next slides show how to do “Refines” link type
  - For other relationships, do same as in previous slides



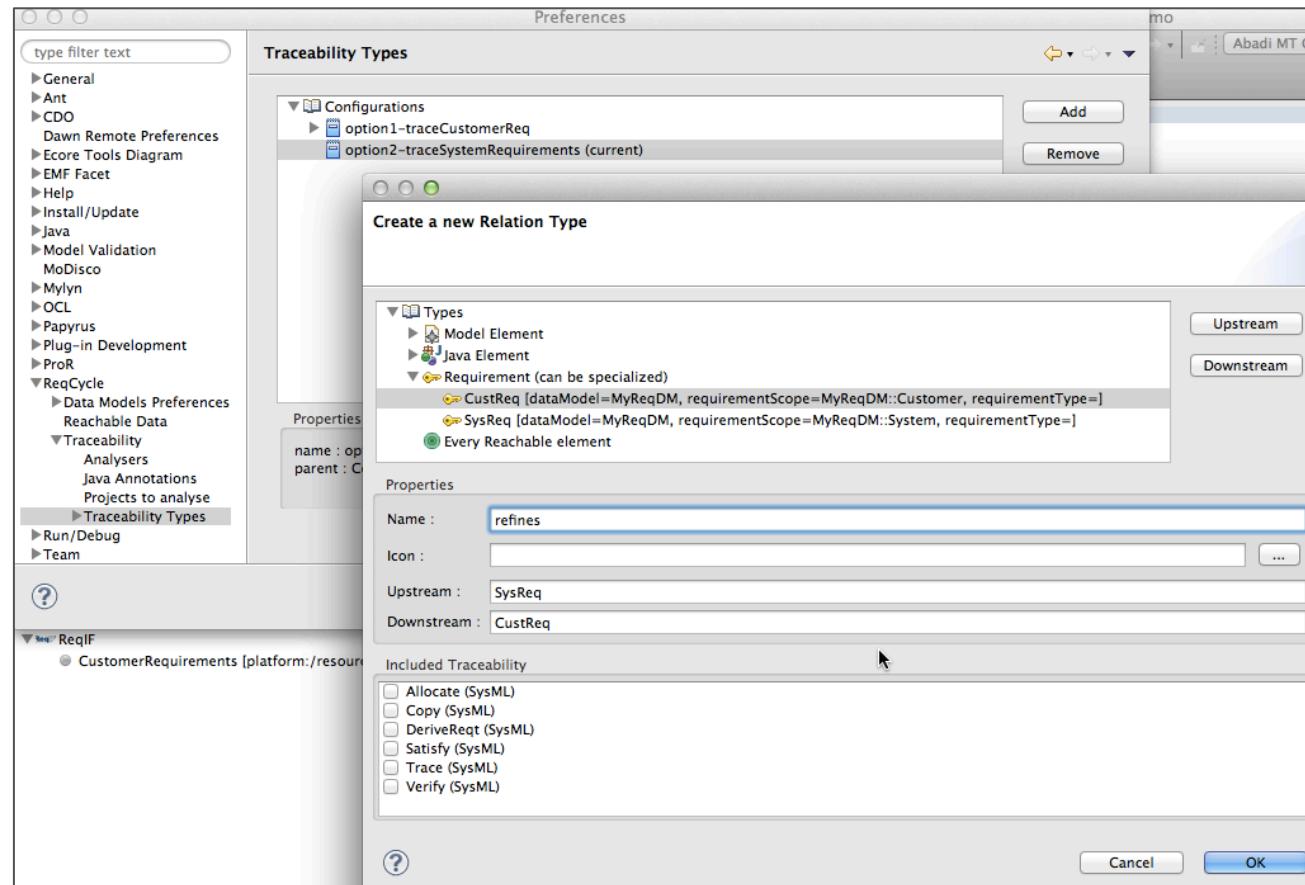
# Refine a customer req into system req

- At first sight we could simply define a Req2Req relation
  - Upstream = Requirement, downstream = Requirement
  - But it would allow creating relations between customer req...
- Let us rather define CustReq. And SysReq reachable types



# Define Refine link type

- Now we can define precisely our “refines” link type

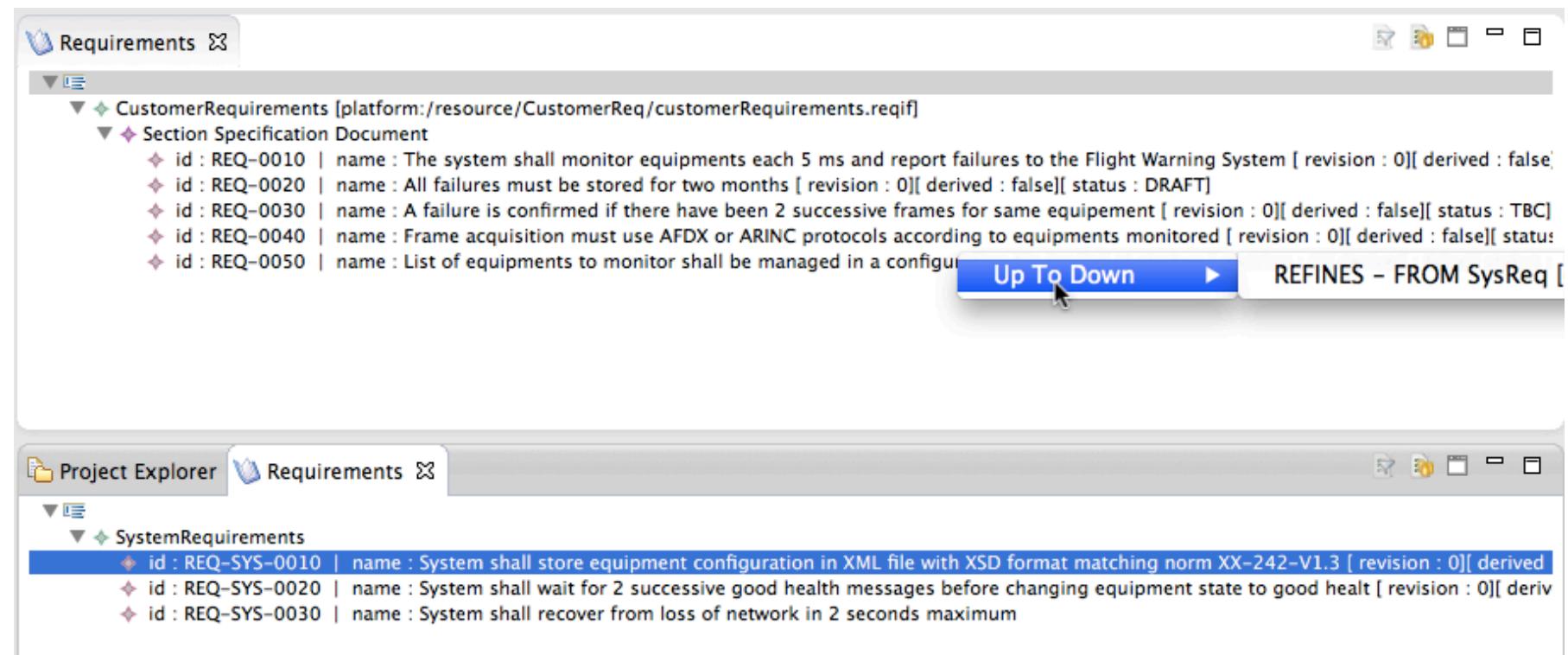


- OK and then do not forget to apply to register the link



# Refine a customer req with a system req

- You can now drag and drop a system requirement to a customer requirement
  - Note: no possible link between customer reqs

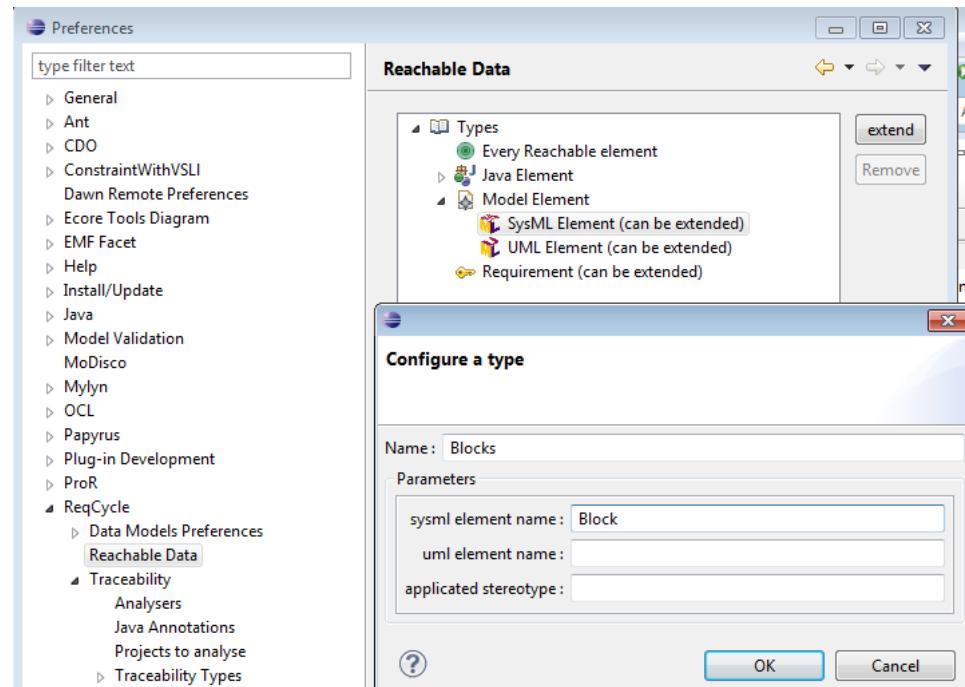


# Restrict link target

- Can restrict target reachable data
  - Go to ReqCycle preferences>Reachable data
  - Extend UML or SysML model element

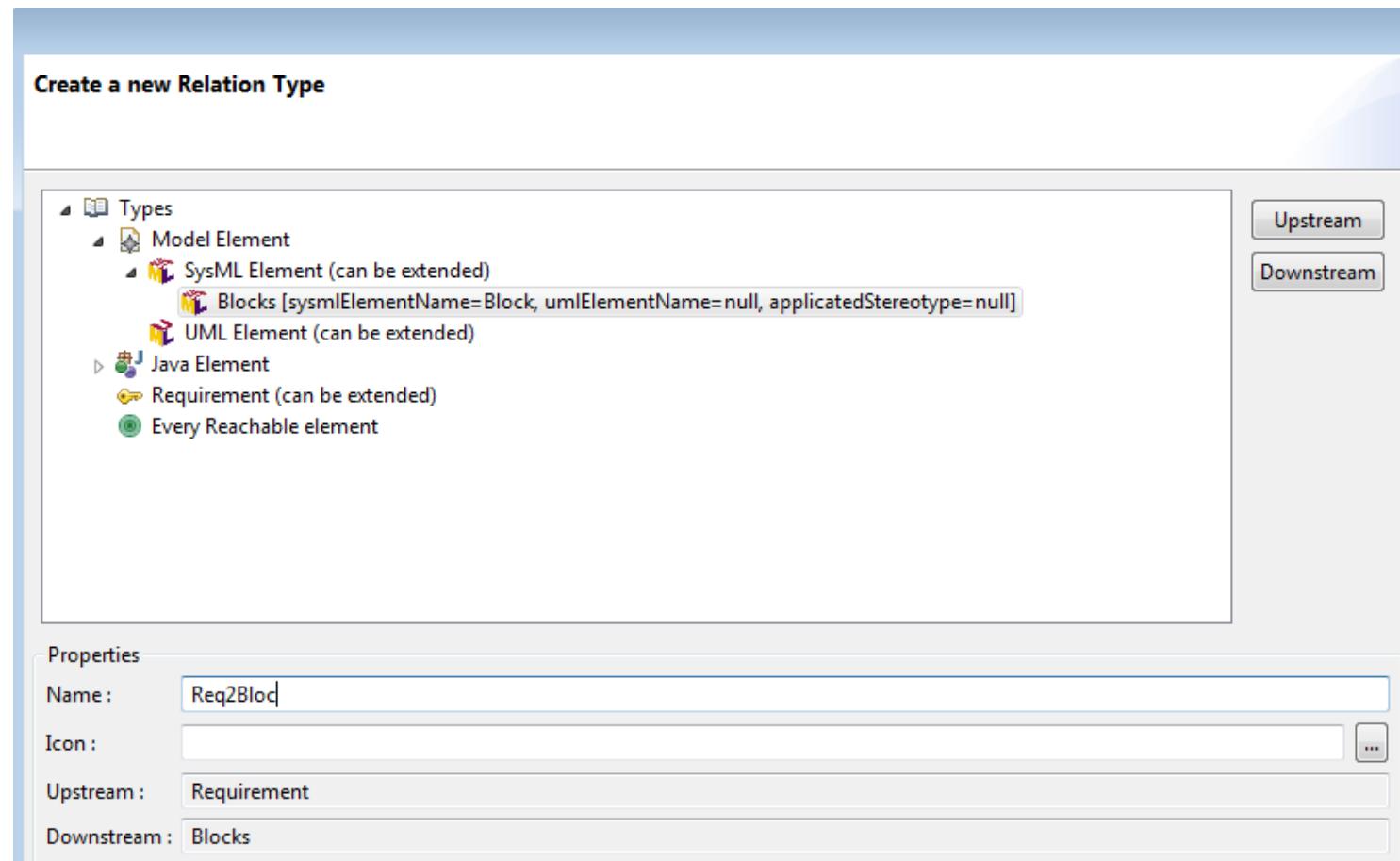
- Example
  - Blocs

- Note
  - Use “apply” on new type and then “OK”

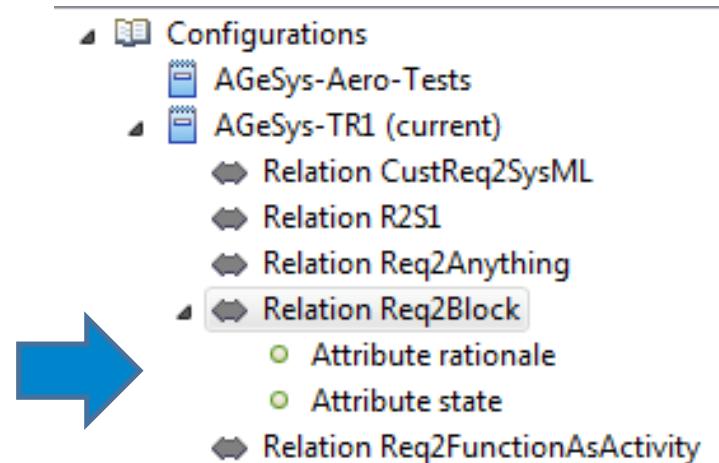
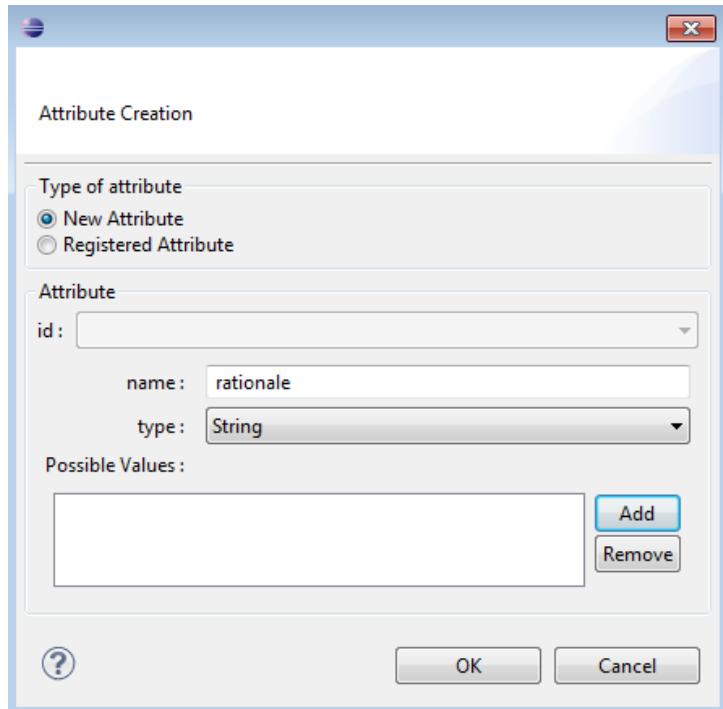


# Restrict links to blocks - definition

- Can then create a link type from any requirement to any SysML block



- Possibility to add link attributes



- Note : development partially done but not enough tested



# Summary of 3<sup>rd</sup> step

- We have demonstrated we could create some links between requirements and other reachable elements
  - For instance between ReqlF requirements and UML elements
  - Even reachable data belong to different languages/formats
- We have shown we could drive the creation of links through the configuration of link types and requirement data model
  - Limit links available according to the traceability link types
  - Limit link source/target to some requirement scopes or types
- Now we are going to see in step 4 how to capture and display traceability links whatever the way its was created





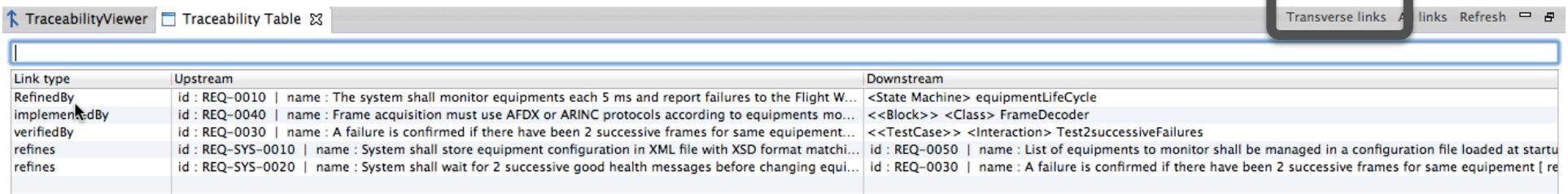
# 4th step – Capture / display traceability

## Concepts and screenshots

Raphaël Faudou  
[raphael.faudou@samarès-engineering.com](mailto:raphael.faudou@samarès-engineering.com)

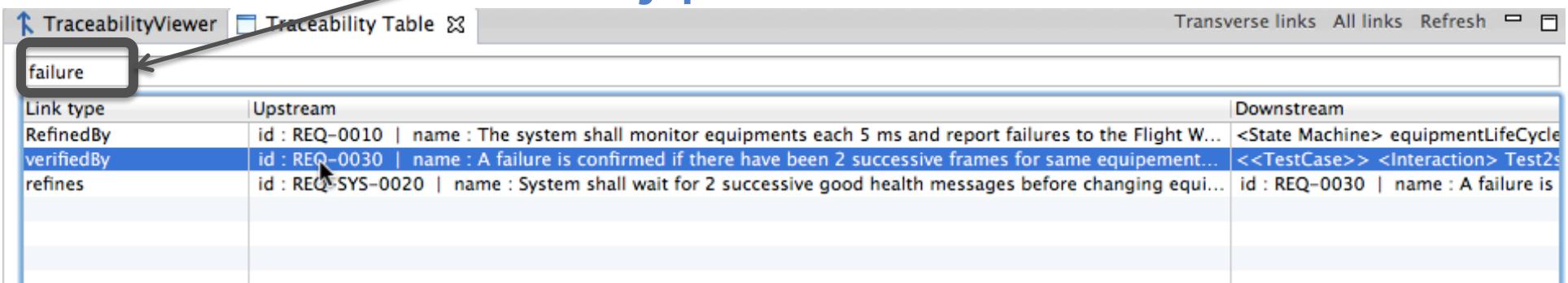
# Traceability table

- Apply “Transverse link” button
  - Requires selecting an Eclipse project



Link type	Upstream	Downstream
RefinedBy	id : REQ-0010   name : The system shall monitor equipments each 5 ms and report failures to the Flight W...	<State Machine> equipmentLifeCycle
implementedBy	id : REQ-0040   name : Frame acquisition must use AFDX or ARINC protocols according to equipments mo...	<>Block><Class> FrameDecoder
verifiedBy	id : REQ-0030   name : A failure is confirmed if there have been 2 successive frames for same equipement...	<<TestCase>> <Interaction> Test2successiveFailures
refines	id : REQ-SYS-0010   name : System shall store equipment configuration in XML file with XSD format matchi...	id : REQ-0050   name : List of equipments to monitor shall be managed in a configuration file loaded at startu...
refines	id : REQ-SYS-0020   name : System shall wait for 2 successive good health messages before changing equi...	id : REQ-0030   name : A failure is confirmed if there have been 2 successive frames for same equipement [ re...

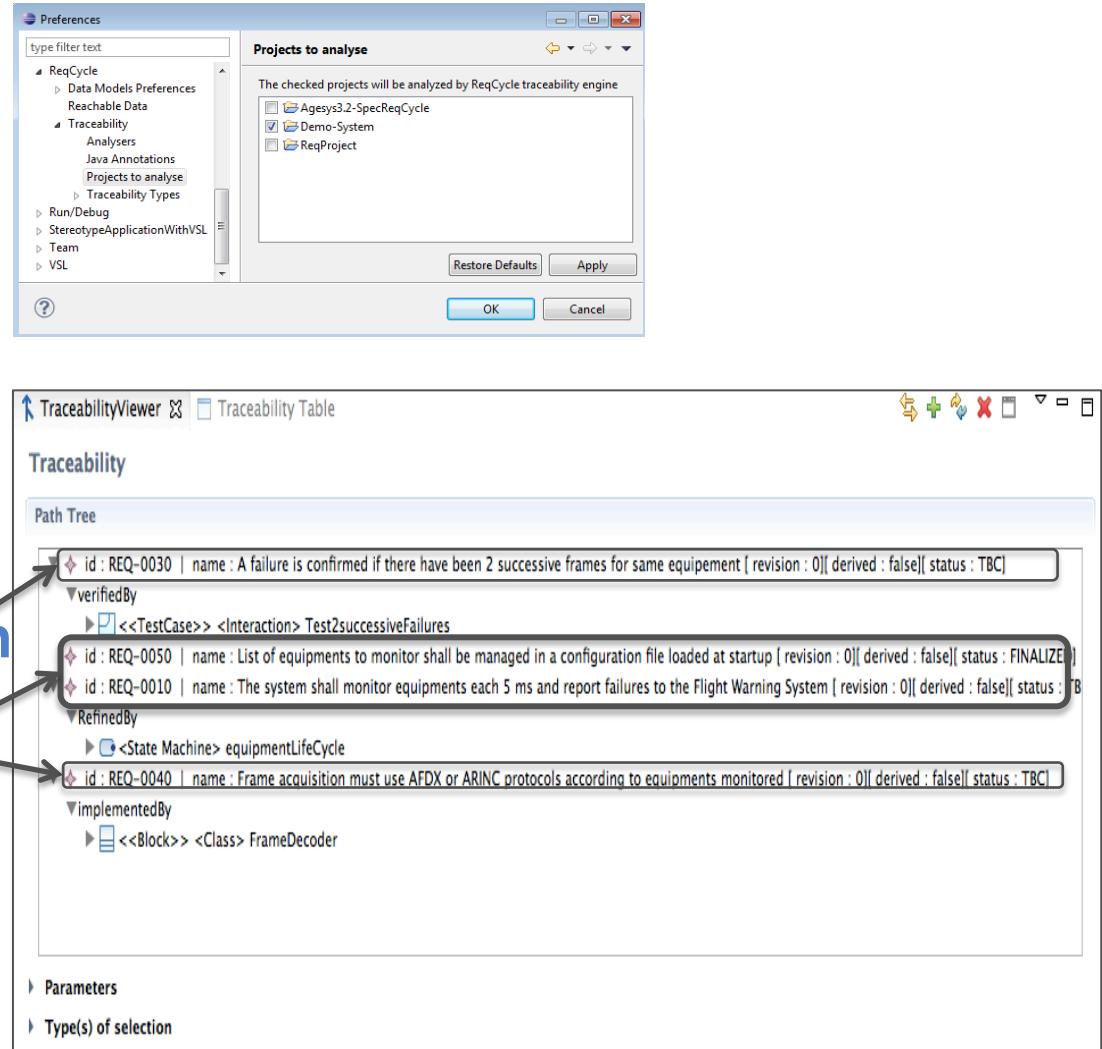
- Can delete a link (contextual menu)
- Can search on any part of text



Link type	Upstream	Downstream
RefinedBy	id : REQ-0010   name : The system shall monitor equipments each 5 ms and report failures to the Flight W...	<State Machine> equipmentLifeCycle
verifiedBy	id : REQ-0030   name : A failure is confirmed if there have been 2 successive frames for same equipement...	<<TestCase>> <Interaction> Test2successiveFailures
refines	id : REQ-SYS-0020   name : System shall wait for 2 successive good health messages before changing equi...	id : REQ-0030   name : A failure is confirmed if there have been 2 successive frames for same equipement [ re...

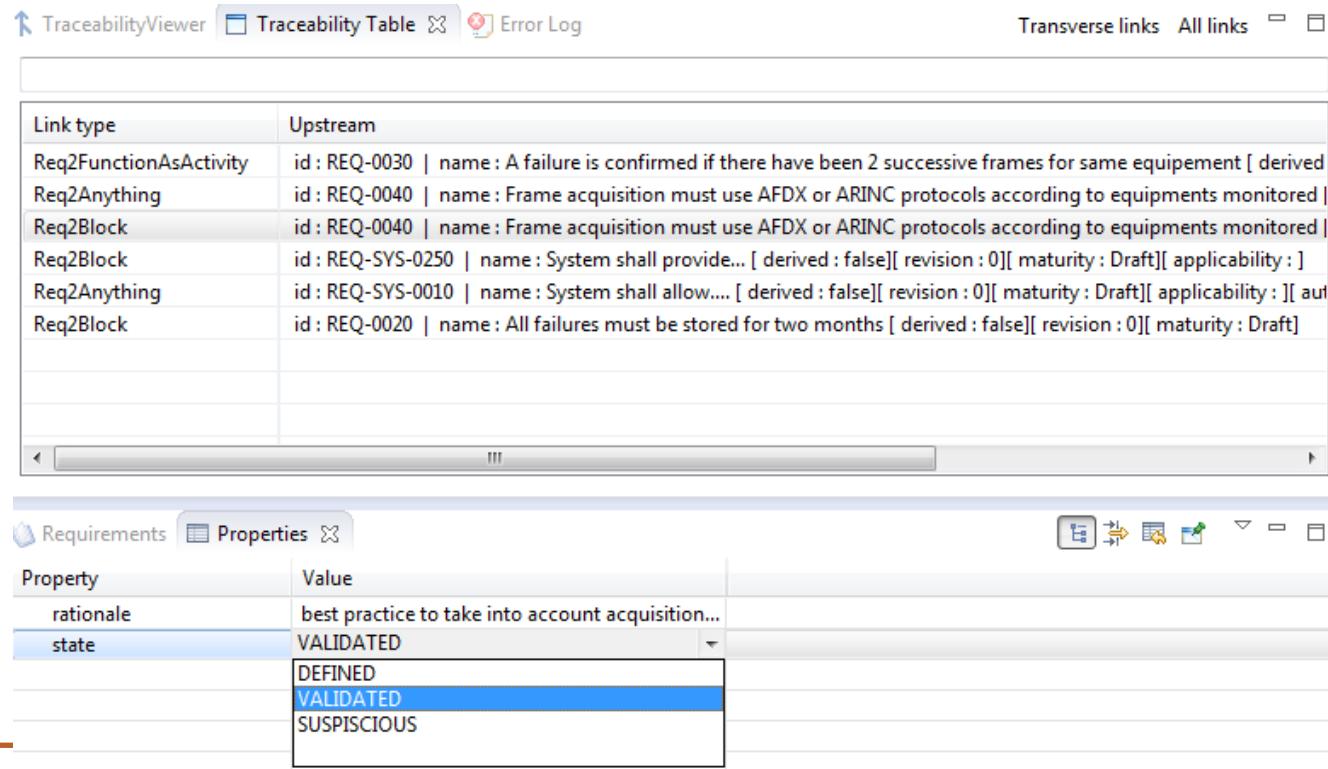
# Traceability viewer

- In ReqCycle preferences, select projects to analyse
- Then open traceability Viewer view
- DnD requirements in Path Tree area
  - Traceability is captured in real time and displayed



# In the future...

- Ability to edit link attributes
  - A first step toward link validation and management of link changes



The screenshot shows a software interface for managing requirements and links. At the top, there are tabs for "TraceabilityViewer", "Traceability Table", and "Error Log". To the right of the tabs are buttons for "Transverse links" and "All links".

The main area contains a table with the following data:

Link type	Upstream
Req2FunctionAsActivity	id : REQ-0030   name : A failure is confirmed if there have been 2 successive frames for same equipement [ derived
Req2Anything	id : REQ-0040   name : Frame acquisition must use AFDX or ARINC protocols according to equipments monitored
Req2Block	id : REQ-0040   name : Frame acquisition must use AFDX or ARINC protocols according to equipments monitored
Req2Block	id : REQ-SYS-0250   name : System shall provide... [ derived : false][ revision : 0][ maturity : Draft][ applicability : ]
Req2Anything	id : REQ-SYS-0010   name : System shall allow.... [ derived : false][ revision : 0][ maturity : Draft][ applicability : ][ aut
Req2Block	id : REQ-0020   name : All failures must be stored for two months [ derived : false][ revision : 0][ maturity : Draft]

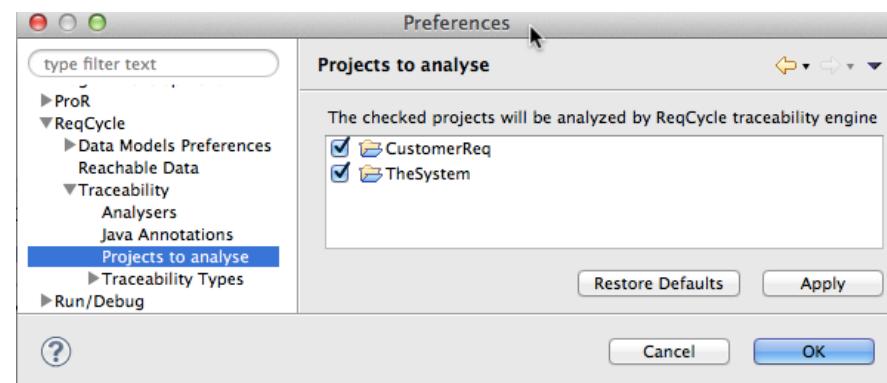
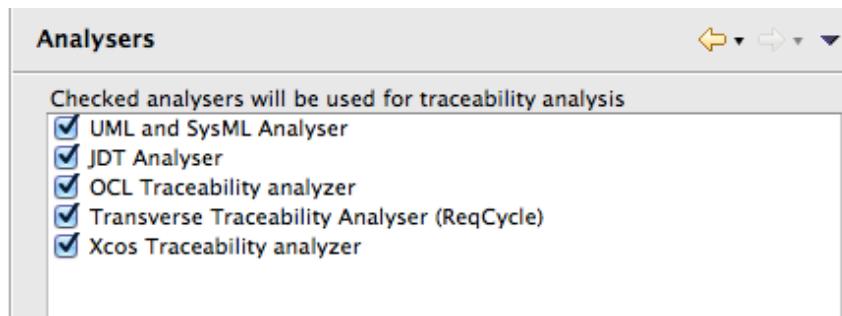
Below the table is a "Properties" panel with the following data:

Property	Value
rationale	best practice to take into account acquisition...
state	VALIDATED

The "state" row has a dropdown menu open, showing three options: "DEFINED", "VALIDATED" (which is highlighted in blue), and "SUSPICIOUS".

# Capture existing links

- ReqCycle can capture existing links created in different languages
  - “satisfy” or “allocate” links in a SysML model
  - “@method” in java code
  - <TraceExtRef> in .xcos model file
  - ...
- First select some traceability analysers registered in your platform
  - ReqCycle preferences>Traceability>Analysers

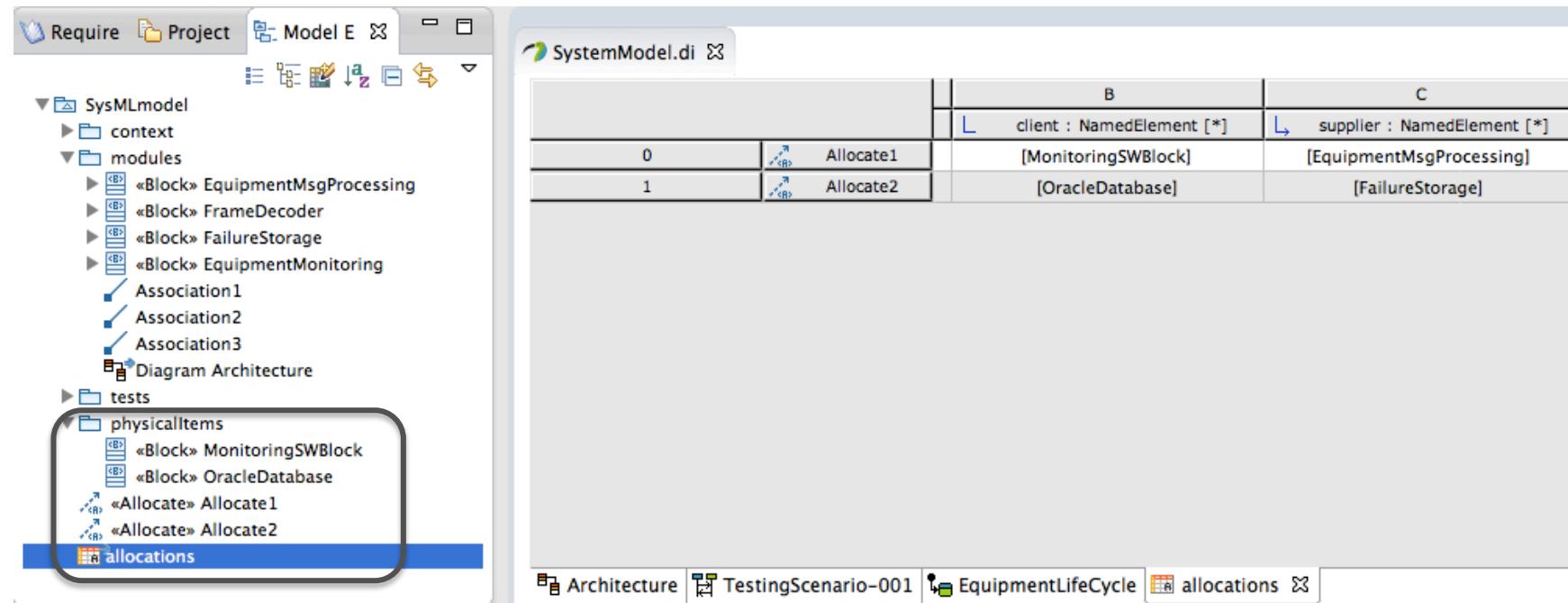


- Then choose projects to analyse



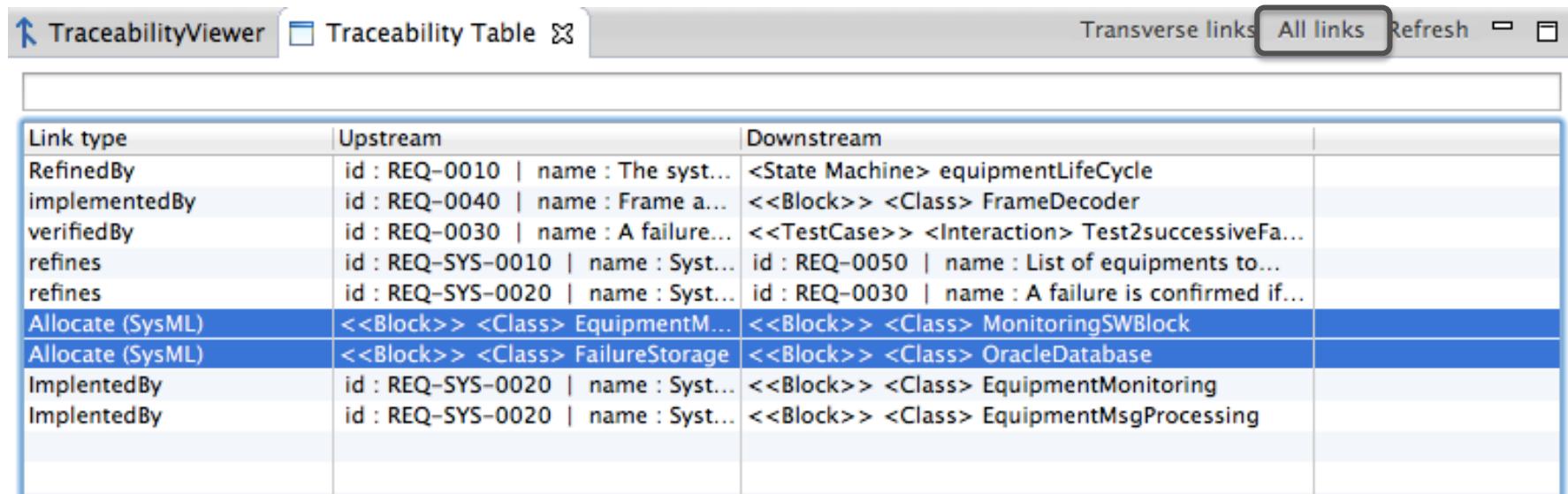
# Let us create SysML “allocate” links

- First open your Papyrus SysML system model
- Then create physical items and two allocations from logical blocks to physical blocks (with a table) as below



# Capture “allocate” SysML links

- Open the traceability table views and click “all links”
  - Allocate links have been captured

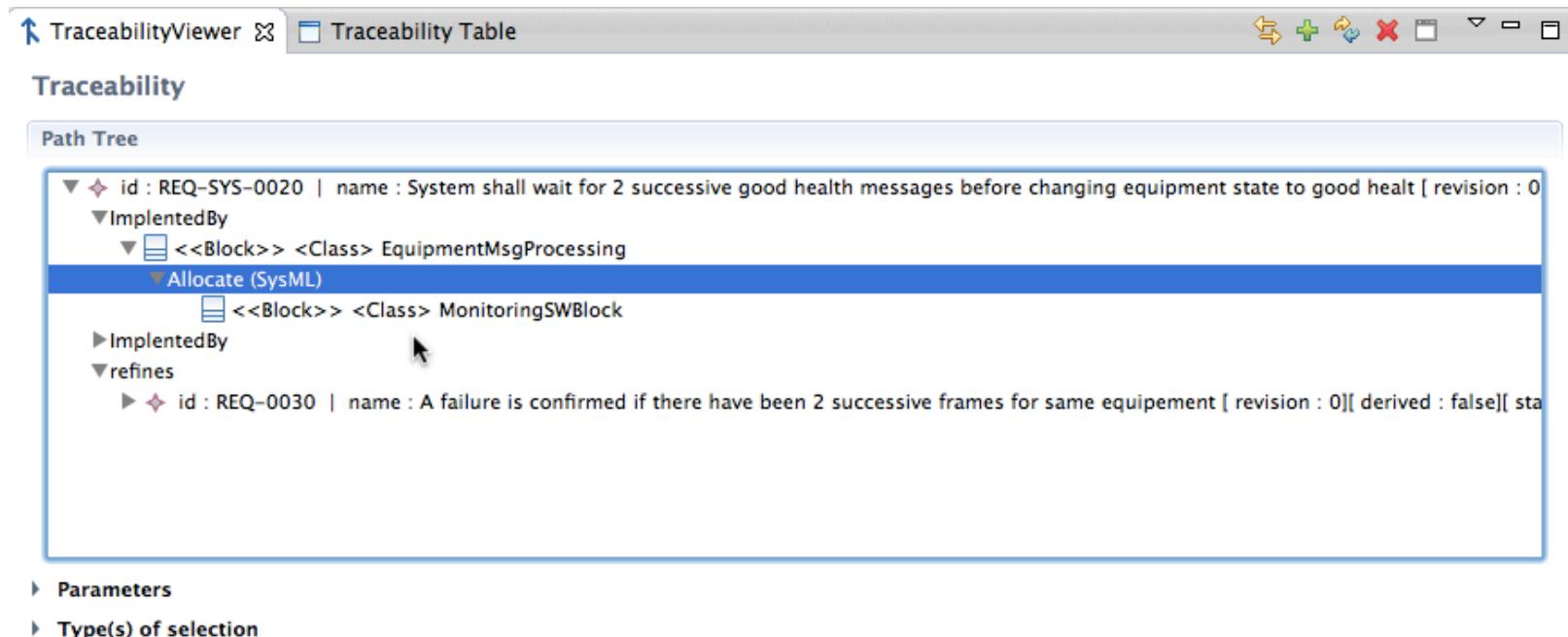


Link type	Upstream	Downstream
RefinedBy	id : REQ-0010   name : The syst...	<<State Machine> equipmentLifeCycle
implementedBy	id : REQ-0040   name : Frame a...	<<Block>> <Class> FrameDecoder
verifiedBy	id : REQ-0030   name : A failure...	<<TestCase>> <Interaction> Test2successiveFa...
refines	id : REQ-SYS-0010   name : Syst...	id : REQ-0050   name : List of equipments to...
refines	id : REQ-SYS-0020   name : Syst...	id : REQ-0030   name : A failure is confirmed if...
Allocate (SysML)	<<Block>> <Class> EquipmentM...	<<Block>> <Class> MonitoringSWBlock
Allocate (SysML)	<<Block>> <Class> FailureStorage	<<Block>> <Class> OracleDatabase
ImplementedBy	id : REQ-SYS-0020   name : Syst...	<<Block>> <Class> EquipmentMonitoring
ImplementedBy	id : REQ-SYS-0020   name : Syst...	<<Block>> <Class> EquipmentMsgProcessing



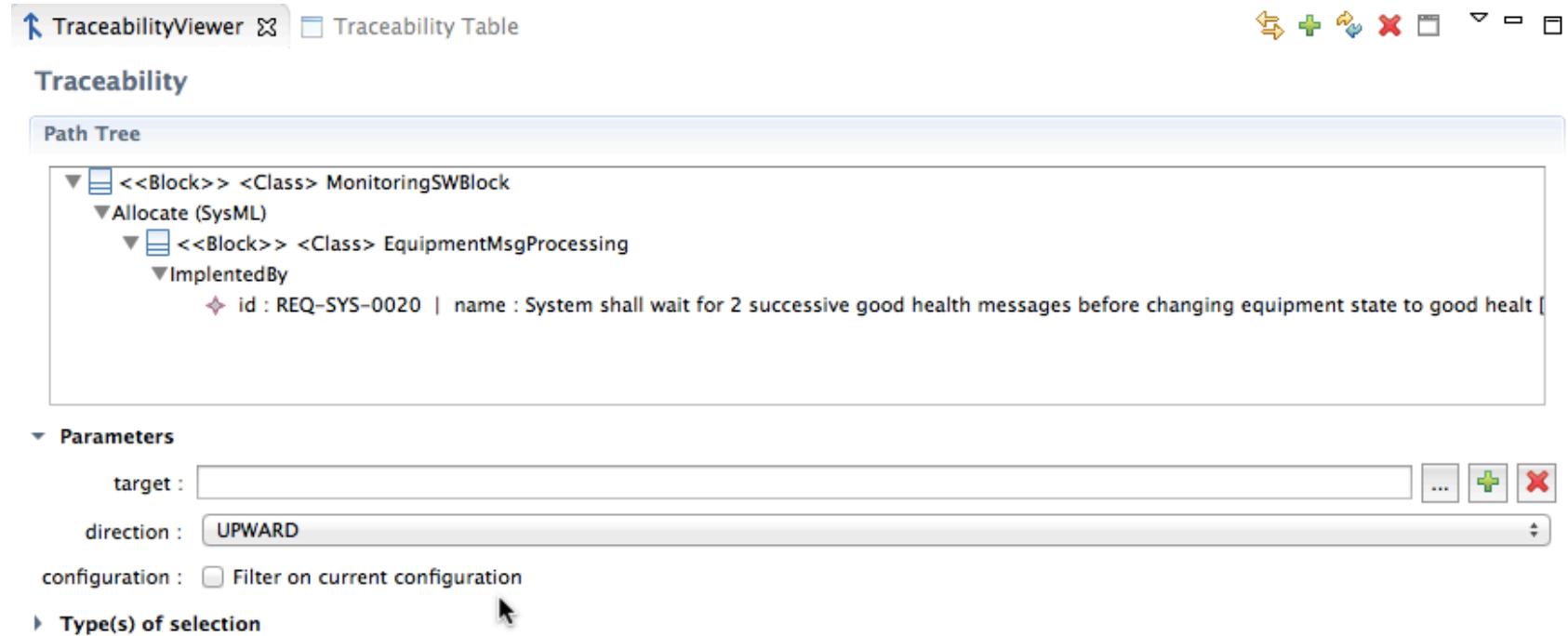
# Display “extended” traceability

- Traceability viewer can also show all captured links and provide what we call “extended” traceability
  - Example with a system requirement implemented by a logical bloc that is allocated on a physical item



# Upward traceability

- Can change direction to upward
  - And then start from a block or any model element...



The screenshot shows the TraceabilityViewer application interface. At the top, there are tabs for "TraceabilityViewer" and "Traceability Table". To the right is a toolbar with various icons for navigation and management. The main area is titled "Traceability" and contains a "Path Tree" section. The tree shows a path starting from a "MonitoringSWBlock" (represented by a blue folder icon) and branching down through an "Allocate (SysML)" relationship to an "EquipmentMsgProcessing" block (also a blue folder icon). This path is further detailed under the "ImplementedBy" relationship, which includes a requirement entry: "id : REQ-SYS-0020 | name : System shall wait for 2 successive good health messages before changing equipment state to good health [...]" (indicated by three ellipsis dots). Below the tree, there are sections for "Parameters" (with fields for "target" and "direction set to "UPWARD"), "configuration" (with a checkbox for "Filter on current configuration"), and "Type(s) of selection" (with a dropdown menu). The bottom right corner of the slide features a decorative graphic of three orange footprints.



# 5th step – export requirement and traceability links

Concepts and screenshots

Raphaël Faudou

[raphael.faudou@samarès-engineering.com](mailto:raphael.faudou@samarès-engineering.com)

# Export requirements



- Not yet supported
- Planned for mid 2014



# Export of traceability links



- Not yet supported
- Planned for mid 2014





# Teamwork support

Concepts and screenshots

Raphaël Faudou

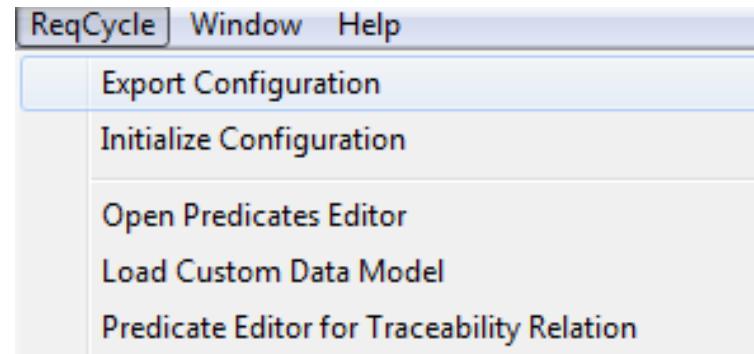
[raphael.faudou@samarès-engineering.com](mailto:raphael.faudou@samarès-engineering.com)

- Can share ReqCycle configuration
  - Export/Import
- Can put requirements file and traceability file to a SVN repository
  - Commit/update – manage versions
- Will be possible (in the future) to synchronize requirements and traceability links at fine-grained level with diff/merge approach
  - Not yet operational (bugs)



# Share ReqCycle configuration

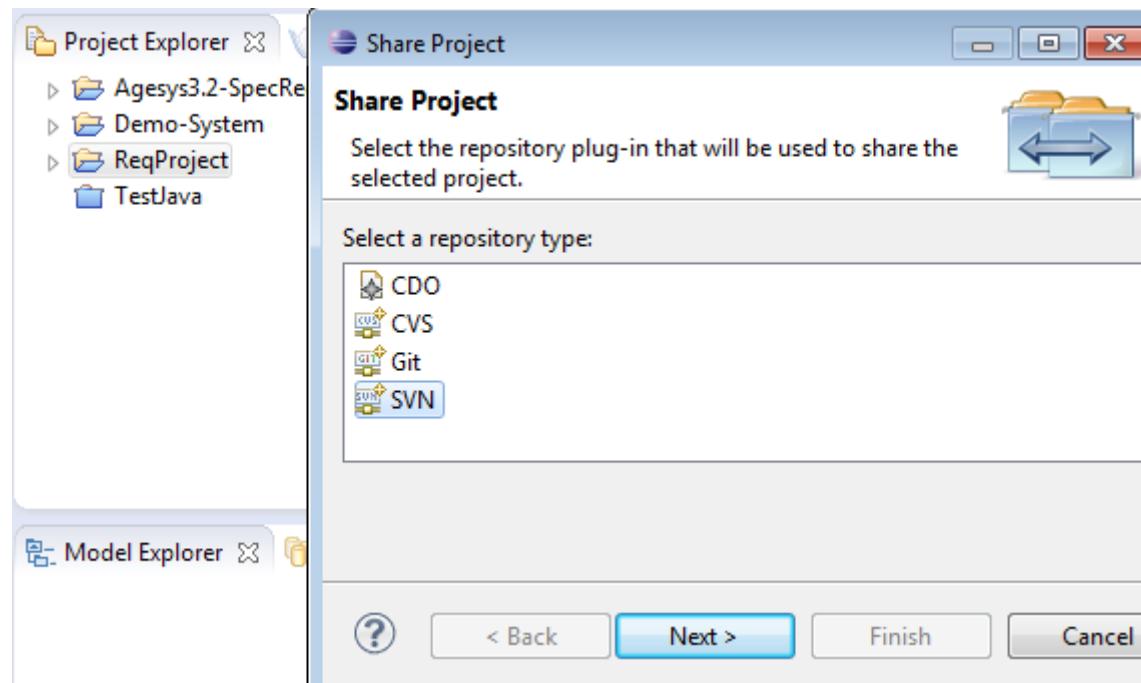
- Can export ReqCycle full configuration as a file
  - Data saved = Data models + reachable types + traceability configurations + requirement sources
  - Menu “ReqCycle”>export configuration
  - Save it as a file – can then be distributed to others



- Can then initialize configuration in an other workspace

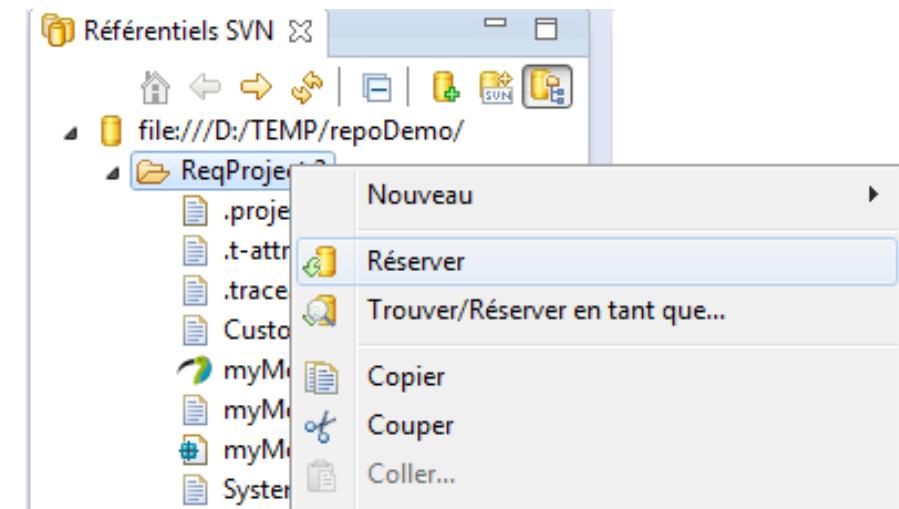
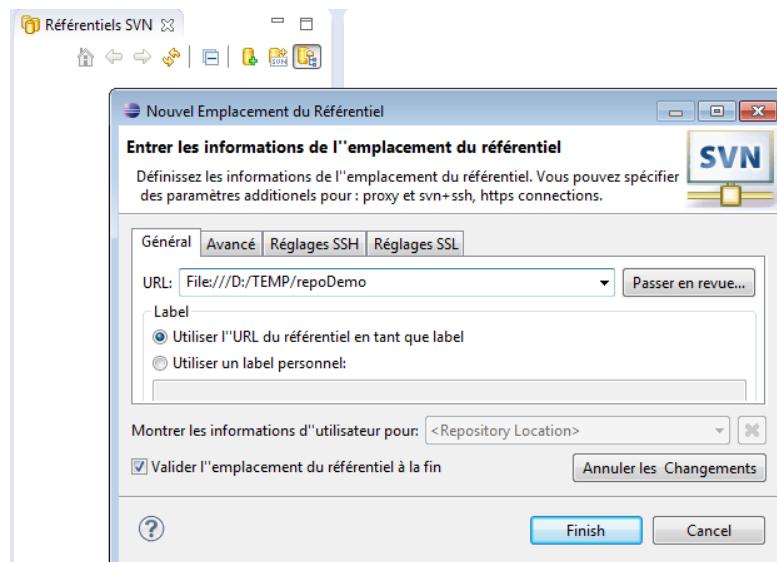
# Put requirements in SVN

- Just share your Eclipse project (containing requirements) in SVN



# Get requirements from SVN

- In other workspace, retrieve requirement projet put previously in SVN
  - SVN perspective: repository access
  - Then checkout



- Restart : req and links are there



## More information...

- You can go to ReqCycle github before the project joins Eclipse PolarSys (currently in proposal phase)
  - <https://github.com/raphaelfaudou/ReqCycle>
- Contacts for roadmap and support
  - [Raphael.faudou@samarès-engineering.com](mailto:Raphael.faudou@samarès-engineering.com)
- List of PolarSys ReqCycle initial committers
  - [Raphael.faudou@samarès-engineering.com](mailto:Raphael.faudou@samarès-engineering.com)
  - [Anass.radouani@atos.net](mailto:Anass.radouani@atos.net)
  - [Mathieu.velten@atos.net](mailto:Mathieu.velten@atos.net)

