www.thalesgroup.com

Transposer

Kitalpha

PolarSys

OPEN

Version 1.0.0

THALES

Agenda

1 **Introduction**

2 **Transposer**

What is Transposer?
Principles
Implementation

3 **Examples**

Agenda

1 **Introduction**

2 **Transposer**

What is Transposer?
Principles
Implementation

3 **Examples**

Purpose

Context

- Several model transformation tools still exist (e.g., ATL, QVT)
- Then, why a new one with Transposer?

Purpose

Dramatically reducing the effort for the developer to:

- Create transformations
- Incrementally add new rules
- Change the transformation order

Agenda

1 **Introduction**

2 **Transposer**

What is Transposer?

Principles

Implementation

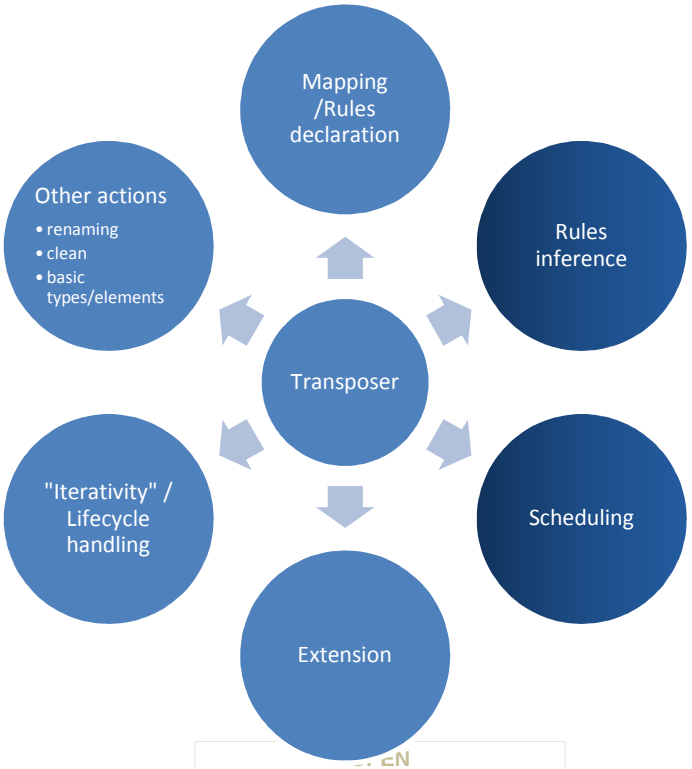
3 **Examples**

What is Tranposer?

Features

A declarative formalism – Declaration with plugins yet

Transposer provides a mechanism of rule inference and scheduling
Transposer supports incremental transformations
For bridges between different metamodels



Agenda

1 **Introduction**

2 **Transposer**

What is Transposer?

Principles

Implementation

3 **Examples**

Declaration structure of model transformation

Mapping:

Objective: declaration of a transformation mapping

- Extension point ID : ID of the declared mapping
- Mapping purpose : Application “Goal”
- Mapping name
- Description
- Domain: declaration of the source metaclasses concerned by the mapping
- Context: global context of the mapping

Mapping Element:

Objective: declaration of the mapping decomposition in a set of elements, each describing a metaclass mapping

- Name
- Domain metaclass: source metaclass to be mapped

Mapping Rules:

Objective: declaration of the rule set which maps the source metaclass into target metaclasses

- Name
- Rule: Java implementation rule
  - Nota: There is a distinction when there is a cycle to map a source metaclass (e.g., Element references a set of Elements)
- Context: local rule context

Java Rule

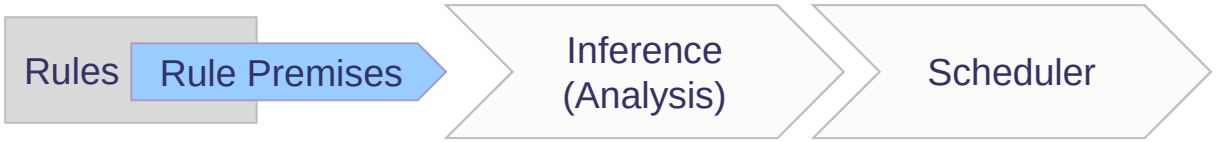
Java Rule:

Objective: implementation of the transformation

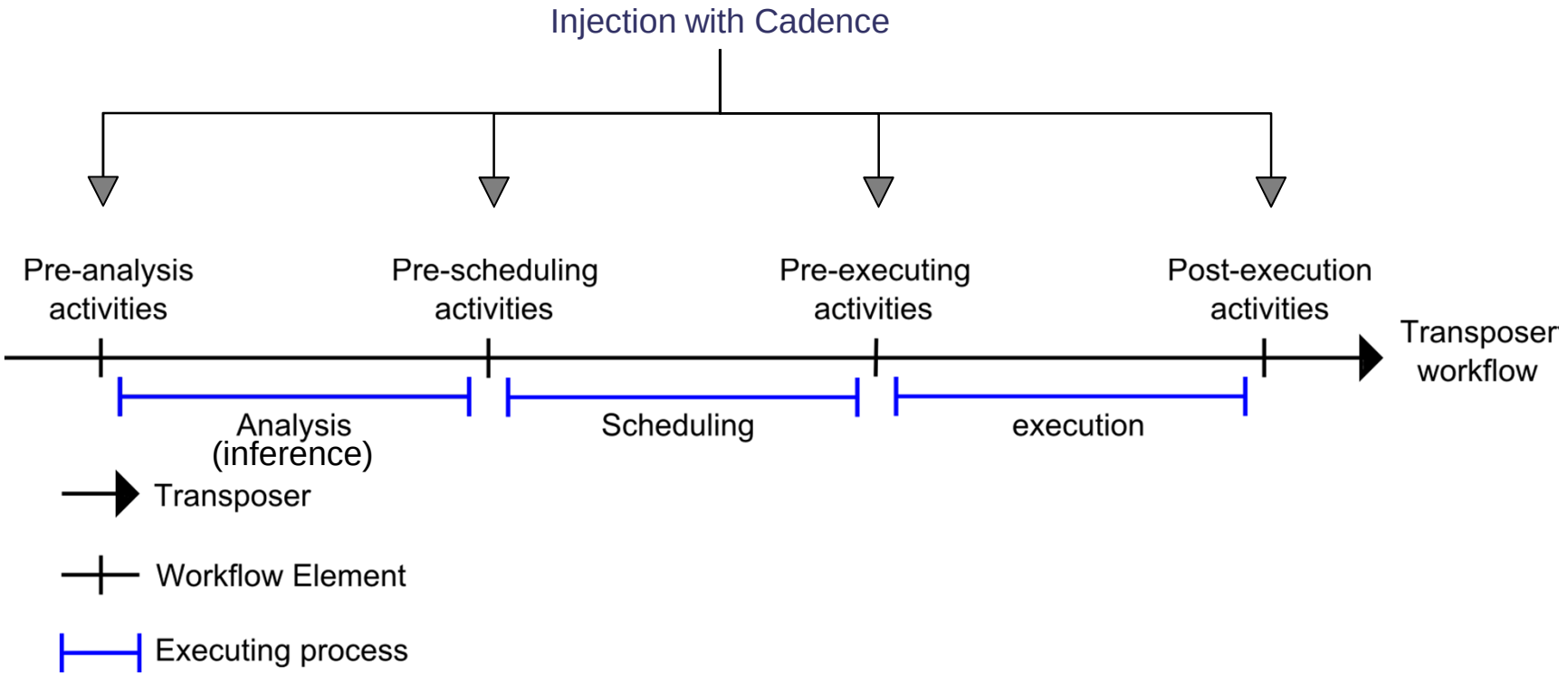
- A set of *premises* to apply the rule: it defines the precedence order to map a source metaclass. For instance, a Class must be mapped in the target location before its Attributes and Operations.
- The *Apply method* to map a source metaclass into target metaclasses

The mechanisms of Inference and Scheduling

- A Java rule declares *premises* in order to manage constraints of precedence order between source metaclasses to be mapped.
- The *Inference function* computes all the mapping dependencies.
- From this computation, the *scheduler* schedules the order to apply the mapping rules.



Workflow



Agenda

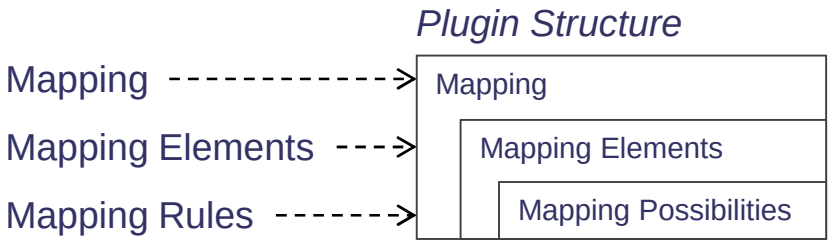
1 **Introduction**

2 **Transposer**

What is Transposer?
Principles
Implementation

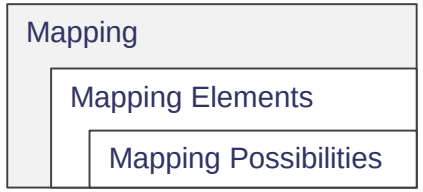
3 **Examples**

Implementation



Plugin declaration 1/3

Plugin Structure



Contribution: Mapping

Extension point: org.polarsys.kitalpha.transposer.rules.handler.mapping
There is only one extension point for all the declaration

General information

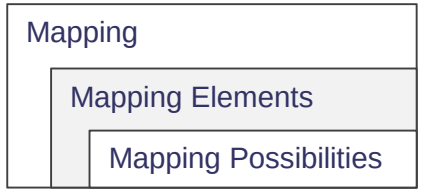
Field	Description	Required/Optional
ID (String)	ID of the mapping contribution	Required
Name (String)	Name of the mapping contribution	Optional

Mapping information

Field	Description	Required/Optional
mappingPurpose (String)	Purpose	Required
mappingName (String)	Name	Required
description (String)	Description	Optional
Private (Boolean)	The mapping is not showed on the user interface	Optional
domainHelper (IDomainHelper)	Class which implements IDomainHelper	Required if there is no mapping extension
Context (Icontext)	Class which implements IContext	Optional
ExtendedMappingExtensionID (String)	ID of the mapping contribution to extend	Optional

Plugin declaration 2/3

Plugin Structure

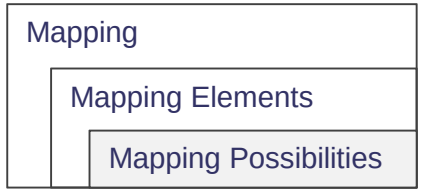


Mapping Element

Field	Description	Required/Optional
domainMetaClass (Class)	Source domain metaclass	Required
reuseExtendedElementDefaultPossibility (Boolean)	Indication if the current mapping element reuses the default possibility of the extended mapping element (= “super-mapping element”)	Required
reuseExtendedElementPossibility (Boolean)	Indication if the extended possibilities are evaluated after the current one	Required
name (String)	Name of the mapping element	Optional

Plugin declaration 3/3

Plugin Structure



Mapping possibilities

Mapping possibilities = Mapping rules in the previous slides

Structure: one default mapping possibility, a set of mapping possibility

Execution of mapping possibility**:**

among all the mapping possibilities, just one is executed: the first IRule with a verified guard

Default mapping possibility:

When the current mapping does not declare any mapping possibility, the default one from the extended mapping is reused

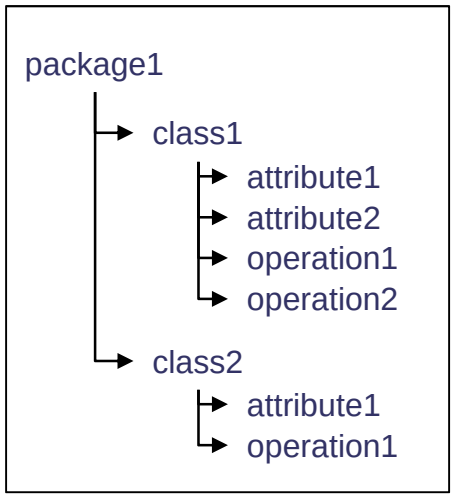
Mapping possibility

Field	Description	Required/Optional
completeRule (IRule)	Java class which implements IRule. It implements the premises and the rule to apply to perform the transformation	Required
enabled (Boolean)	Specify if this rule is enabled (= executable or not)	Required
incompleteRule (IRule)	Java class which implements IRule. It is used when cycles are detected during the transformation.	Optional
context (IContext)	Java class which implements IContext. It used as local context of the rule	Optional
name (String)	Name of the mapping possibility	Optional

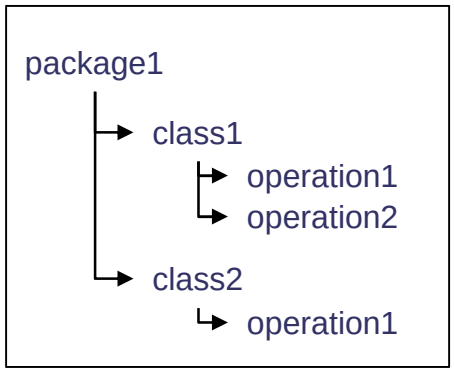
Example

Model example

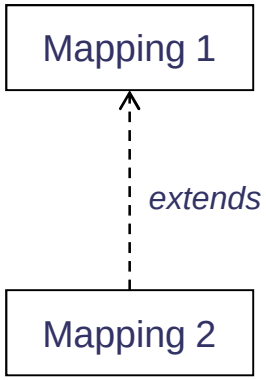
Source Model



Target Model (with M2)



Mapping example



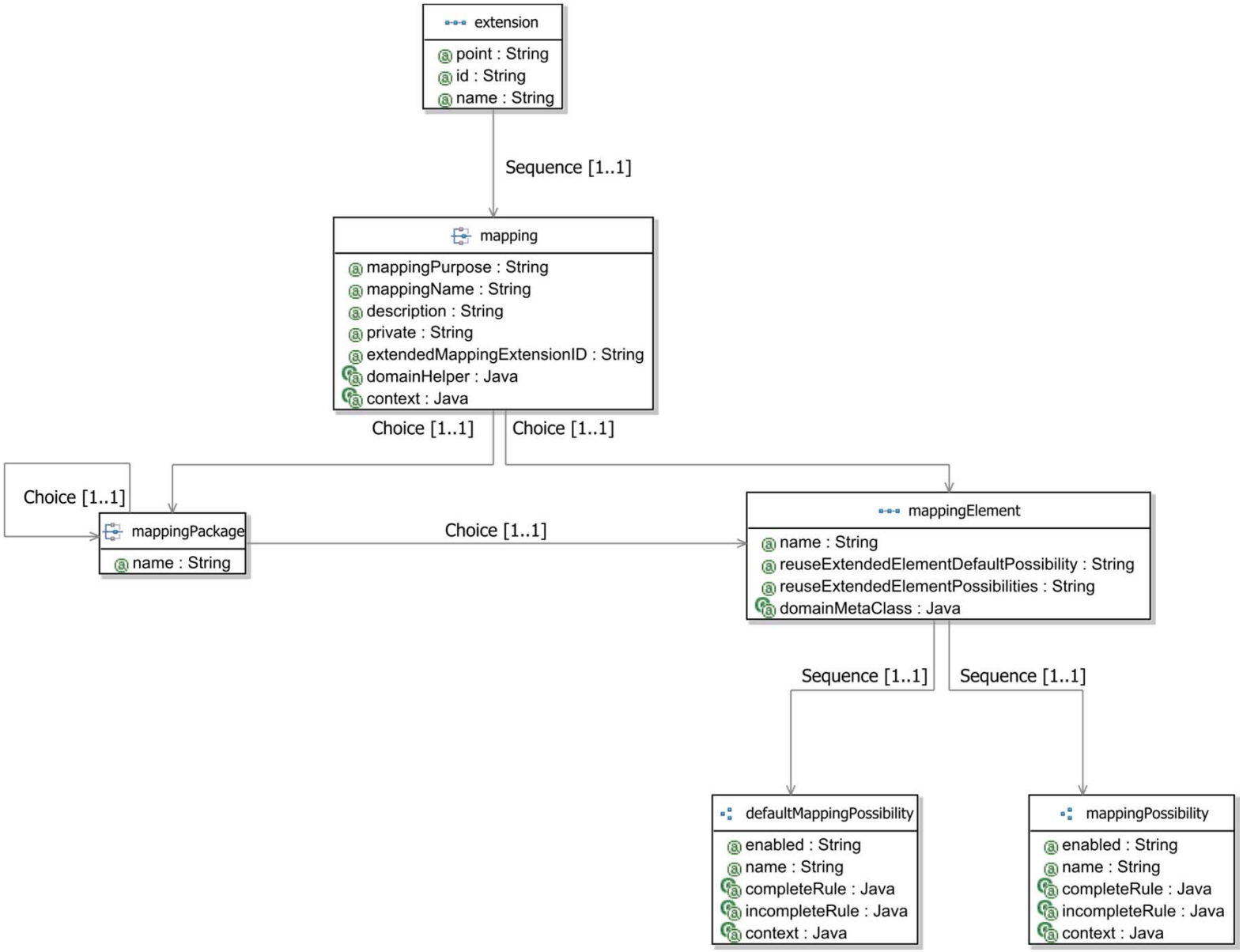
Mapping 1 [M1]
- Mapping Element Class [ME-Class]
  - Mapping Possibilities [MP]
- Mapping Element Attribute [ME-Attribute]
  - Mapping Possibilities [MP]

Mapping 2 [M2]
- Mapping Element Class [ME-Class]
  - Mapping Possibilities [MP]
- Mapping Element Operation [ME-Operation]
  - Mapping Possibilities [MP]

Explanation

- With Mapping 1, it is possible to map Class and Attribute
- With Mapping 2, it is possible to map Class and Operation
- Mapping 2 extends Mapping 1
- When executing Mapping 2:
  - if (M2.ME-Class.MP has a default Possibility and no other Possibility is not verified) then the default Possibility of M2.ME-Class.MP is applied ⇒ In this case, only Classes and Operations are mapped
  - if (M2.ME-Class.MP has no default Possibility and no other Possibility is not verified and reuseExtendedElementDefaultPossibility == true) then the default Possibility of M1.ME-Class.MP is applied ⇒ In this case, only Classes and Attributes are mapped

Extension Point Structure



Transposer API

Purpose

- Providing utilities to facilitate EMF Transformation
- Providing transformation traceability
- Managing complete and incomplete rule in one rule (cycles)
- Providing helpers to perform transformations

Transposer API

Define rule

Rule extends AbstractTransformationRule<T>

- Create(T element_p, IContext context_p): create the object
- Update(T element_p, IContext context_p): Update the object created

Define DomainHelper

DomainHelper extends EmfDomainHelper

Traceability

To allow traceability, set the mapping context as GenericTransformationContext class.

Cadence activities:

- EMF Based Trace loader: Pre-analysis workflow element activity to load trace model and initializing the GenericTransformationContext context
- EMF Based Soft Trance cleaner: Post-execution workflow element activity to clean trace model or update it
- EMF Based Trace Saver: Post-execution workflow element activity to save trace model

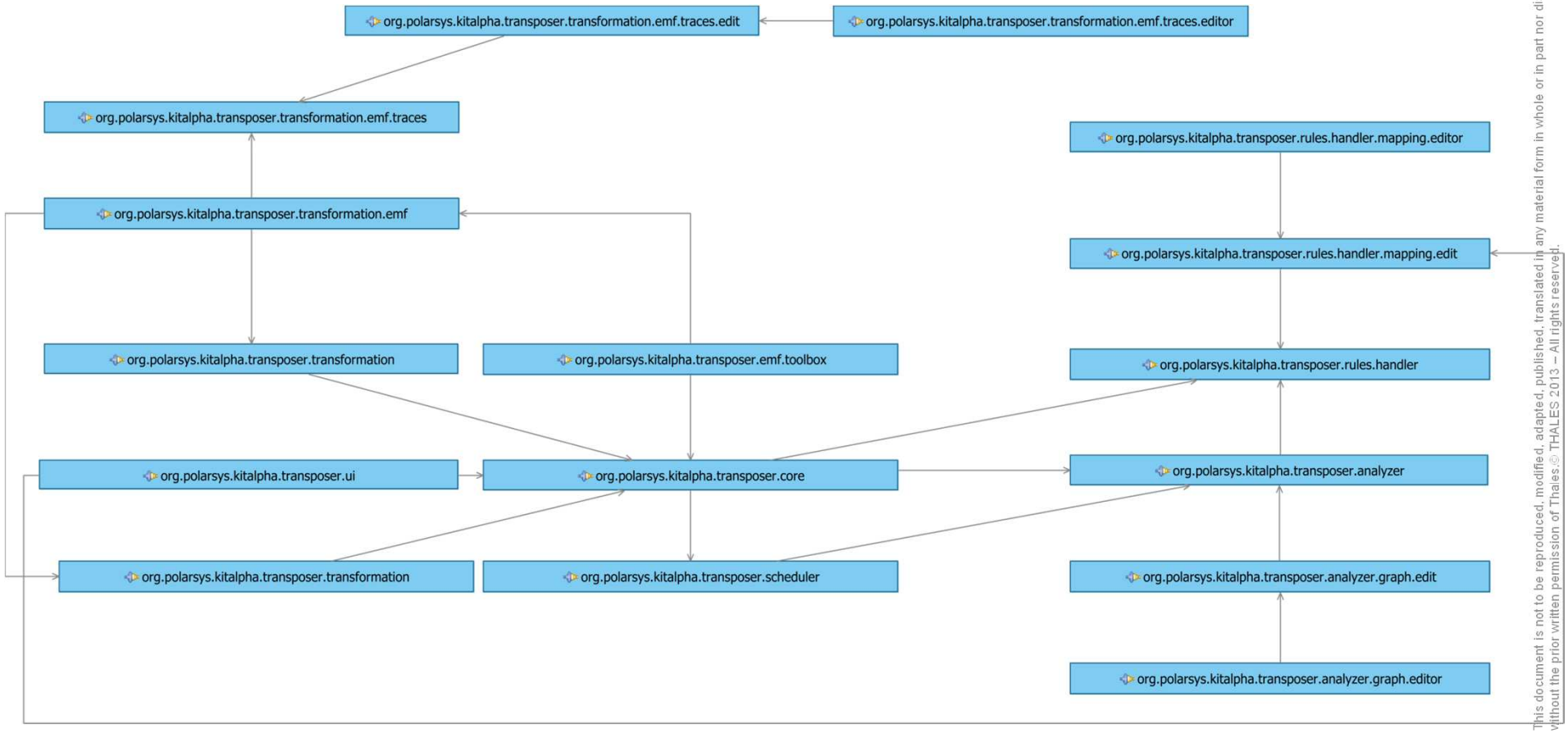
ContextHelper

Get/add transformed object from/in the context with a specific role (for traceability)

Transposer is composed of five main components

Transposer UI	Provides UI for Transposer (Launch Configurations, Menu, etc…)
Analyser	Building a dependency graph from computed premises
Sheduler	Scheduling the rule application thanks to the dependency graph
Rule Handler	Responsible for rule discovery and rule inference
Transposer Core	Managing the preceding ones

Internal Structure



Agenda

1 **Introduction**

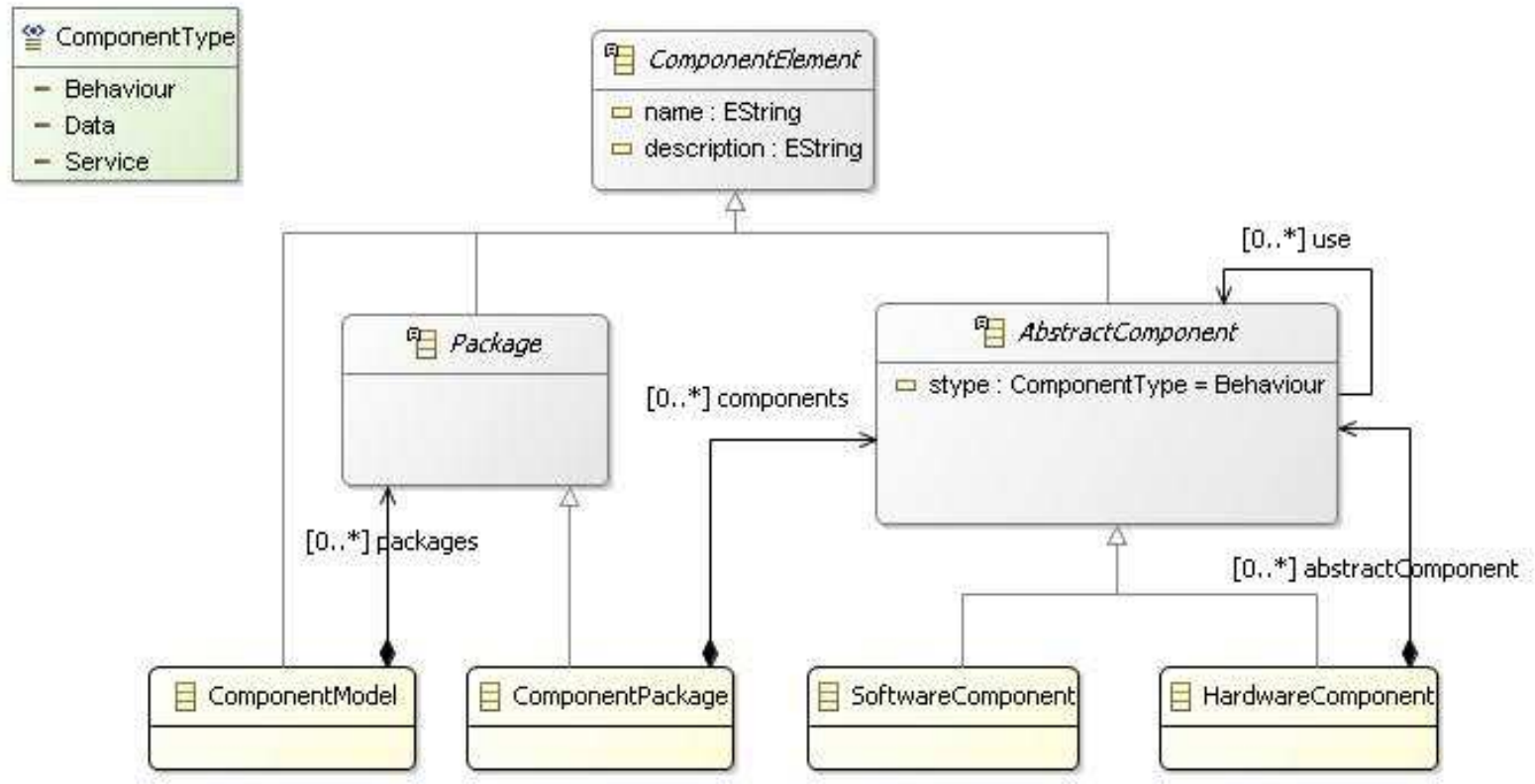
2 **Transposer**

What is Transposer?
Principles
Implementation

3 **Examples**

Example

Component Sample metamodel



Example – Simple mapping

Use Case:

"Component Sample"-to-UML model transformation and vice versa

Mapping

	Component Sample entity		UML entity
	ComponentModel		Model
	ComponentPackage		Package
	SoftwareComponent		Component
	HardwareComponent		Class

Example – Simple mapping

Step 1. Project Definition

Create a plugin

- org.polarsys.kitalpha.transposer.m2m.componentsample.to.uml

Add plugin dependencies

- org.eclipse.ui
- org.eclipse.core.runtime
- org.polarsys.kitalpha.transposer.core
- org.polarsys.kitalpha.transposer.transformation.emf
- org.polarsys.kitalpha.vp.componentsample.model

Add extension to

- org.polarsys.kitalpha.transposer.rules.handler.mapping
  - ID: org.polarsys.kitalpha.transposer.m2m.componentsample.to.uml.mapping
  - Name: ComonentSample Model to UML Model

Example – Simple mapping

Step 2. Declaration of the Mapping structure 1/2

Add new Mapping

- MappingPurpose: org.polarsys.kitalpha.transposer.m2m.componentsample.to.uml.purpose
- MappingName: ComponentSample Model to UML Model
- Description: Default ComponentSample Model to UML Model Transformation
- Private: false
- DomainHelper: Qualified name to the class that implements the domain helper interface

Add new PackageMapping

- Name: Give a name, example: default

Add a new Mapping Element

- DomainMetaClass: Qualified name to domain meta-class on which the rule works (example: ComponentModel)
- Name: Component Model Case

Add a new Mapping Rules

- CompletRule: Qualified name to the class that implements a rule interface (IRule)
- Name: Default Component Model Rule

Example – Simple mapping

Step 2. Declaration of the Mapping structure 2/2

Domain helper class (DomainHelper)

- Implements IDomainHelper
- GetAllDomainClasses()
  - Return all classes of the simplecomponent Ecore
- getAnalysisSources(Collection<?> source)
  - Returns the collection of classes to be transformed from models given in source parameter.
- getDomainMetaclass(String name)
  - Returns the meta-class of name class given in the parameter
- getDomainMetaclass(Object o)
  - Returns the meta-class of the object given in the parameter
- getName(Object o)
  - Returns the name of the object
- isDomainFor(Object o)
  - Indicates to transposer whether the launch menu will be proposed
- isHotSpot(Object o)
  - Indicates whether the object is the entry point for the graph scheduling

Example – Simple mapping

Step 3. Declaration of Rule

Rule class (ComponentSampleClassRule)

- Implements IRule

- apply(Object o, IContext c)
  - The execution of the rule
- getPremises(Object o)
  - Returns the premises of the rule
- isApplicableOn(Object o)
  - Returns if the rule is applicable on the object given in the parameter

Example – Simple mapping

Step 4. Worflow contribution 1/2

Initialization activity to pre-analysis workflow element

- Add extension to:
  - org.polarsys.kitalpha.candence.core.activity.declaration
  - Fill the fields:
    - Identifier: org.polarsys.kitalpha..transposer.m2m.componentsample.to.uml.initTranformation
    - Name: Initialization of Tranformation ComponentSample To UML
    - WorkflowIdentifier: org.polarsys.kitalpha.transposer.workflow
    - WorkflowElementIdentifier: org.polarsys.kitalpha.transposer.before.analysis
    - ActivityClass: full qualified name to the class that implement IActivity interface (here: org.polarsys.kitalpha.transposer.m2m.componentsample.to.uml.activities.InitializeTranformation)

Initialization Activity

- Responsible for preparing the resource where the UML model will be saved

Example – Simple mapping

Step 4. Worflow contribution 2/2

Finalization activity to post-execution workflow element

- Add extension to:
  - org.polarsys.kitalpha.candence.core.activity.declaration
  - Fill the fields:
    - Identifier: org.polarsys.kitalpha.transposer.m2m.componentSample.to.uml.PostTransformation
    - Name: Finalization of Transformation ComponentSample To UML
    - WorkflowIdentifier: org.polarsys.kitalpha.transposer.workflow
    - WorkflowElementIdentifier: org.polarsys.kitalpha.transposer.after.rule.execution
    - ActivityClass: full qualified name to the class that implement IActivity interface (here: org.polarsys.kitalpha.transposer.m2m.componentsample.to.uml.activities.FinalizeTransformation)

Initialization Activity

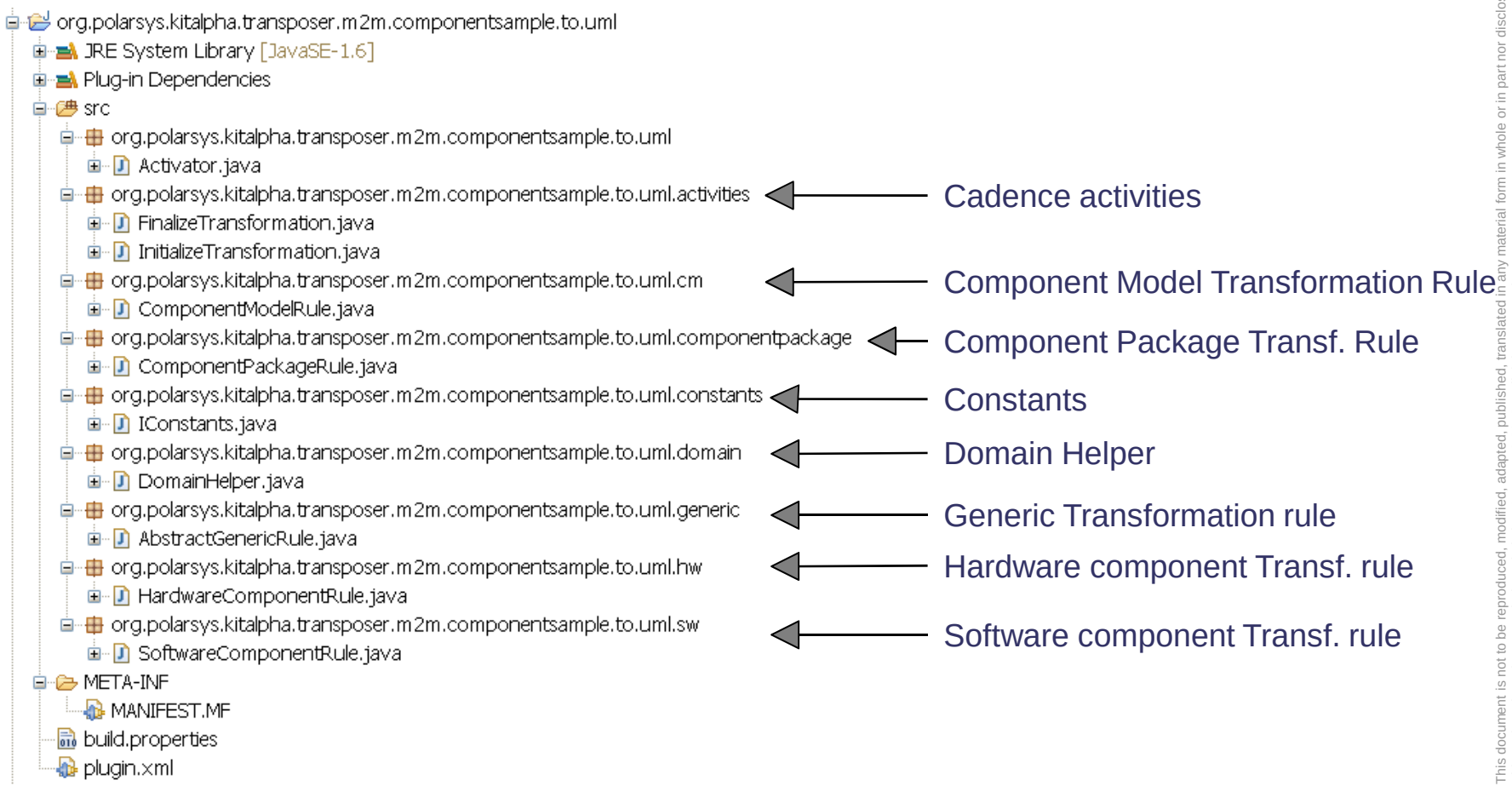
- Retrieving a resource prepared by the initialization activity from Transposer context and save it.

Example – Simple mapping

Hands on! 1/2

Open a Kitalpha IDE with on the workspace than contains the example

- org.polarsys.kitalpha.transposer.m2m.componentsample.to.uml (Example source)

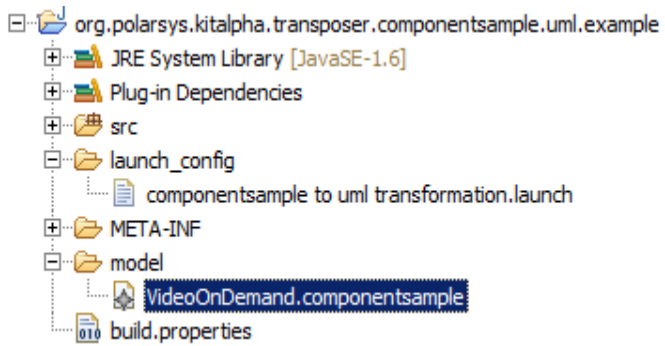


Example – Simple mapping

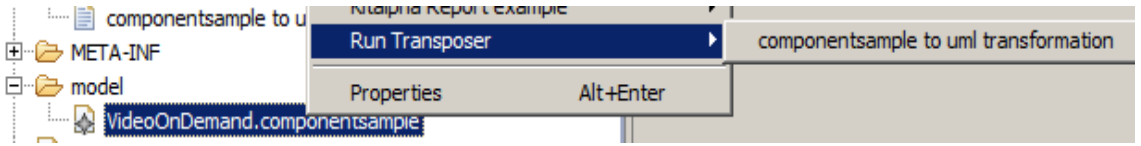
Hands on! 2/2

Launch a new instance of Kitalpha (1/5)

- Create a new project or plugin
- Create a model folder
- Create a new ComponentSample model or import one.
- Create a launch_configurations
- Copy ComponentSample to UML (default).launch configuration in launch_configurations folder
- Right click on ComponentSample Model → Run Transposer → simpleComponent to UML (default)



1. Example project hierarchy



2. Launch Transposer

3. The result

Example – UML to component sample

Use Case:

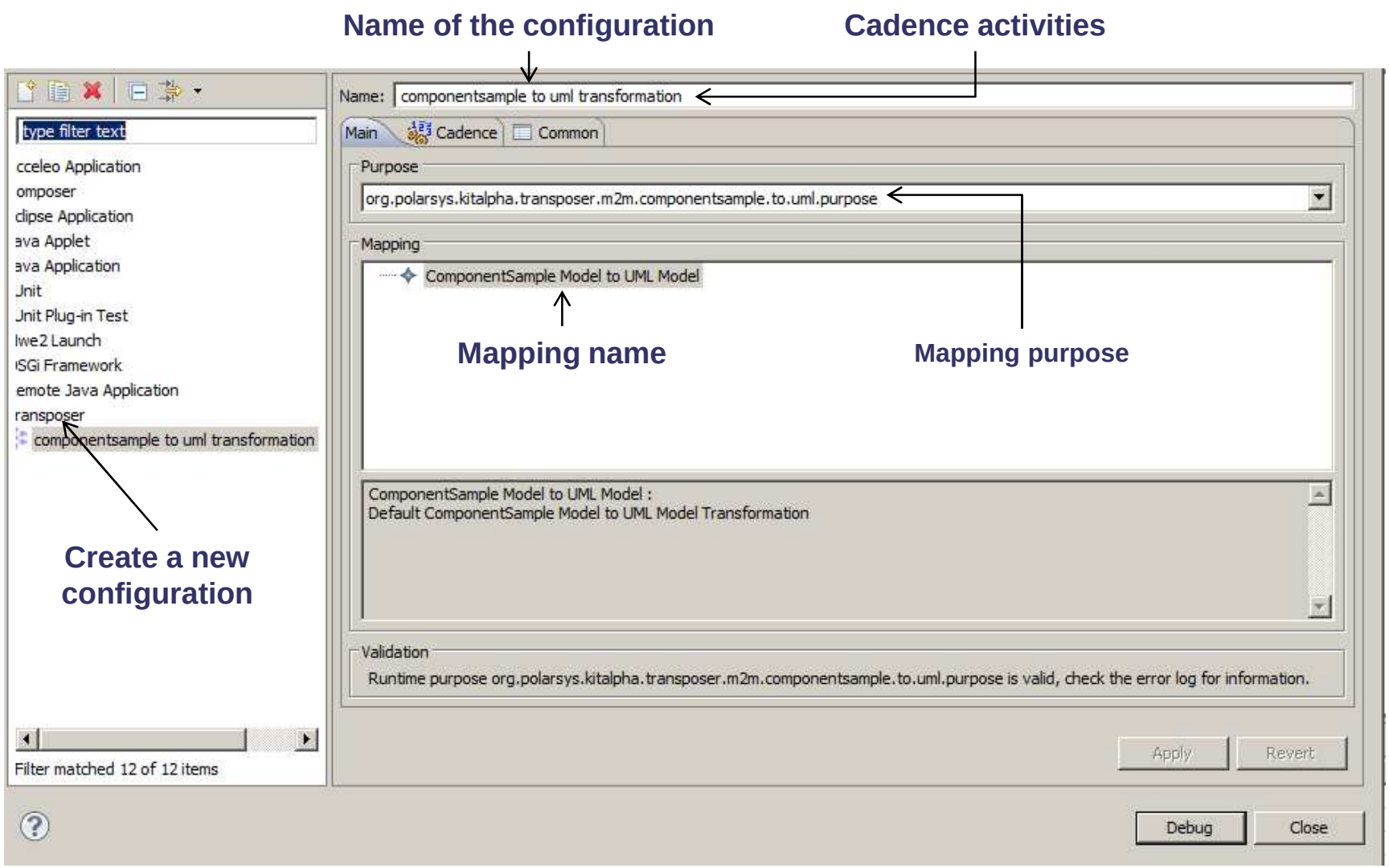
- UML-to-“Sample component” model transformation
- Use the Transposer API:
  - Initialization activity: Prepare a componentsimple resource instead UML one
  - Finalization activity: Save a simpleComponent resource.
  - DomainHelper: Read the UML model and it implements EmfDomainHelper
  - Context: GenericTransformationContext for traceability
  - Rules implementation: Classes implement AbstractTransformationRule

Mapping

	UML entity		**component Sample entity**
	Model		ComponentModel
	Component		SoftwareComponent
	Package		ComponentPackage
	Class		HardwareComponent

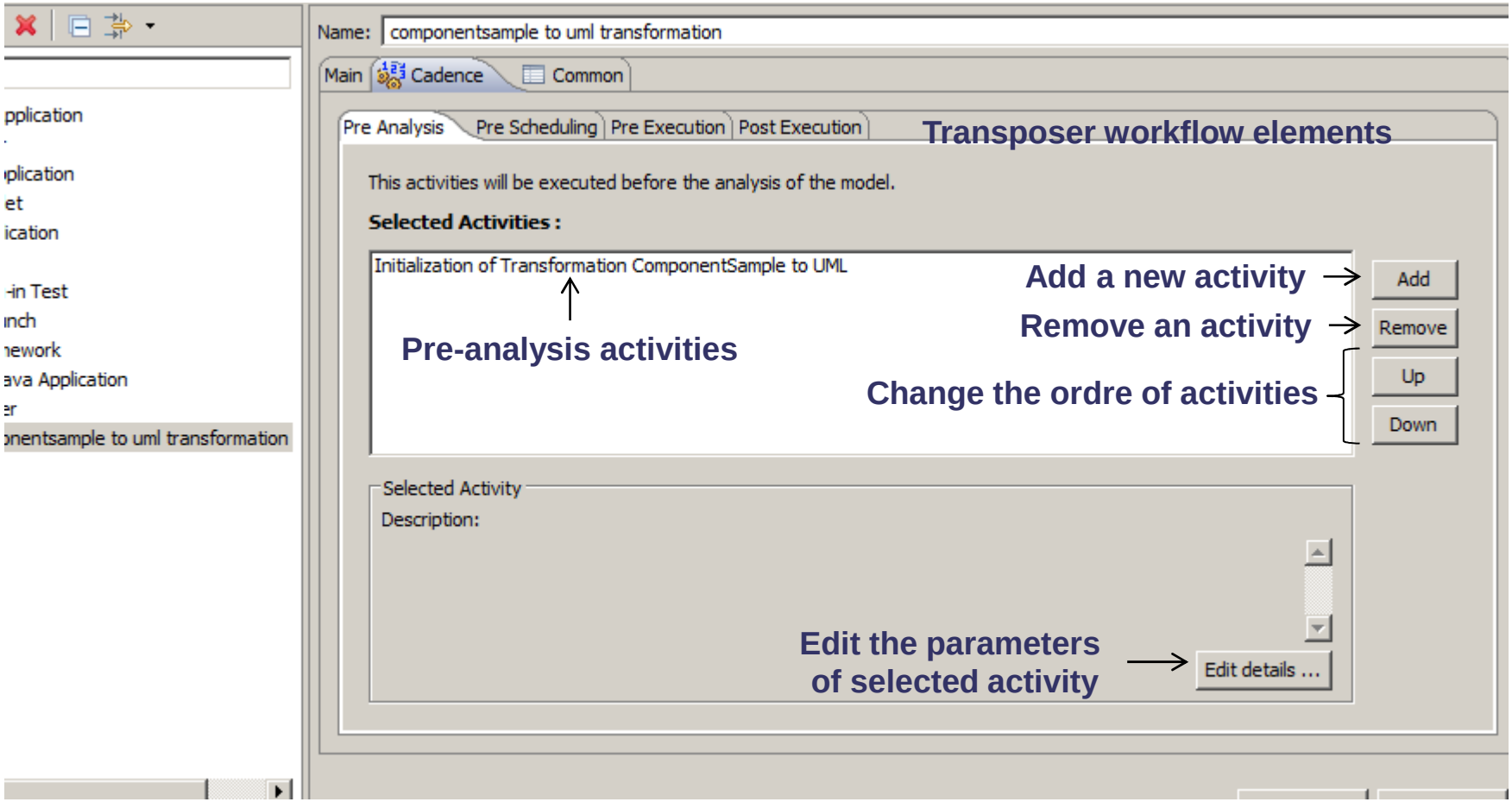
Example – UML to component sample

UI Transposer Launch configuration – Mapping choose



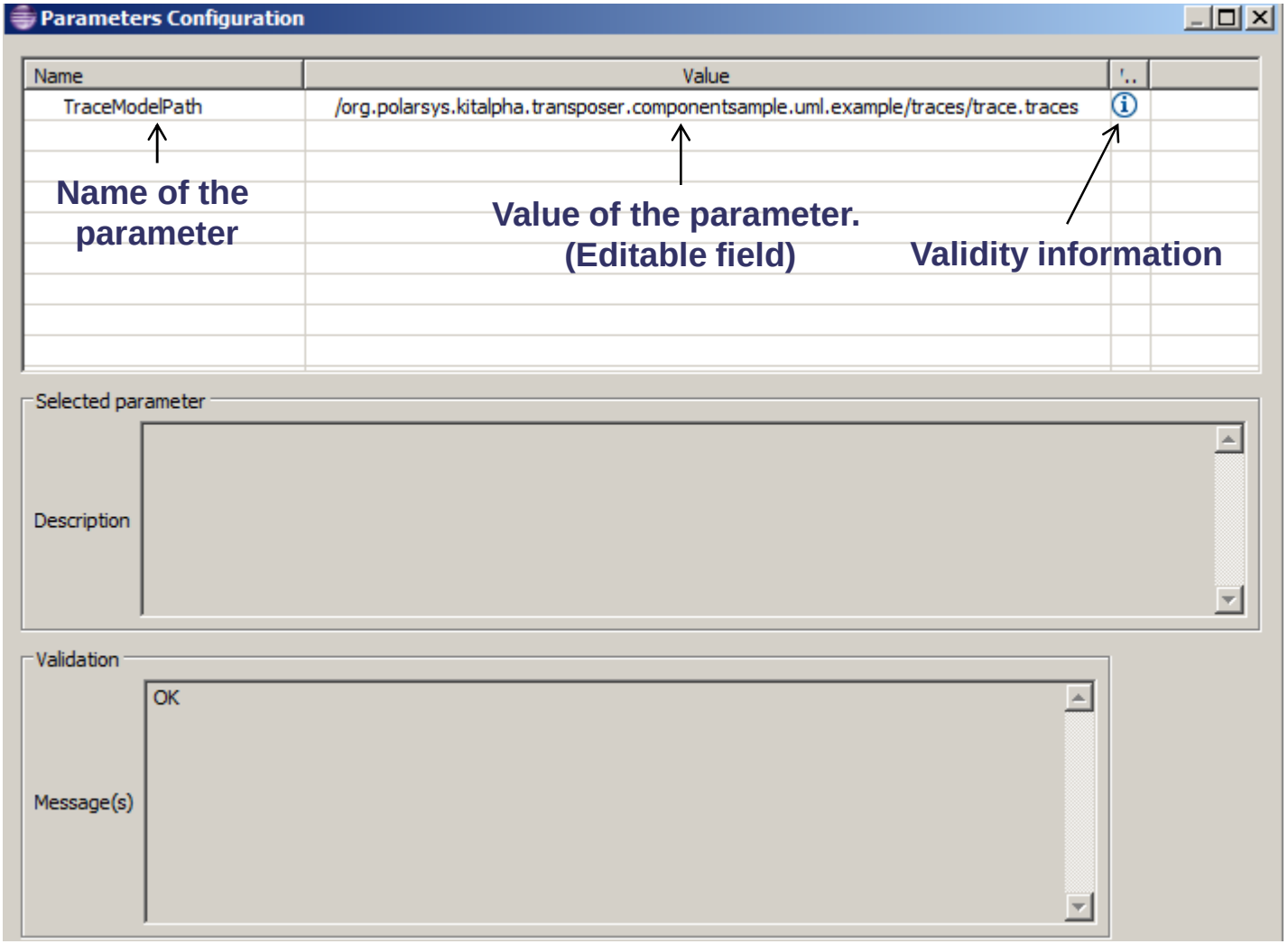
Example – UML to component sample

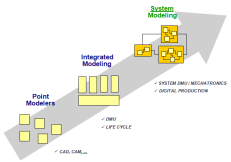
UI Transposer Launch configuration – Cadence activities



Example – UML to component sample

UI Transposer Launch configuration – Activity parameters





Kitalpha is supported by **Sys2Soft** and **Crystal**, respectively French and European projects

Thαnk You!

https://www.polarsys.org/projects/polarsys.kitalpha

https://polarsys.org/wiki/Kitalpha

benoit.langlois@thalesgroup.com

#LangloisBenoit