1. Introduction

2. Transposer
   What is Transposer?
   Principles
   Implementation

3. Examples
Agenda

1. Introduction
2. Transposer
   - What is Transposer?
   - Principles
   - Implementation
3. Examples
Purpose

Dramatically reducing the effort for the developer to:
- Create transformations
- Incrementally add new rules
- Change the transformation order

Context

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

Injection with Cadence

- Pre-analysis activities
- Pre-scheduling activities
- Pre-executing activities
- Post-execution activities

Transposer workflow

- Analysis (inference)
- Scheduling
- Execution

Transposer

Workflow Element

Executing process
Agenda

1 Introduction
2 Transposer
   What is Transposer?
   Principles
   Implementation
3 Examples
Implementation

**Plugin Structure**

- Mapping
- Mapping Elements
- Mapping Rules

```
Mapping ----┬── Mapping
           ├── Mapping Elements
           └── Mapping Possibilities
```
**Contribution: Mapping**

Extension point: org.polarsys.kitalpha.transposer.rules.handler.mapping

There is only one extension point for all the declaration

### General information

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID (String)</td>
<td>ID of the mapping contribution</td>
<td>Required</td>
</tr>
<tr>
<td>Name (String)</td>
<td>Name of the mapping contribution</td>
<td>Optional</td>
</tr>
</tbody>
</table>

### Mapping information

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Required/Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>mappingPurpose (String)</td>
<td>Purpose</td>
<td>Required</td>
</tr>
<tr>
<td>mappingName (String)</td>
<td>Name</td>
<td>Required</td>
</tr>
<tr>
<td>description (String)</td>
<td>Description</td>
<td>Optional</td>
</tr>
<tr>
<td>Private (Boolean)</td>
<td>The mapping is not showed on the user interface</td>
<td>Optional</td>
</tr>
<tr>
<td>domainHelper (IDomainHelper)</td>
<td>Class which implements IDomainHelper</td>
<td>Required if there is no mapping extension</td>
</tr>
<tr>
<td>Context (Icontext)</td>
<td>Class which implements IContext</td>
<td>Optional</td>
</tr>
<tr>
<td>ExtendedMappingExtensionID (String)</td>
<td>ID of the mapping contribution to extend</td>
<td>Optional</td>
</tr>
</tbody>
</table>
## Mapping Element

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

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

Model example

Source Model

```
package1
  class1
    attribute1
    attribute2
    operation1
    operation2
  class2
    attribute1
    operation1
```

Target Model (with M2)

```
package1
  class1
    operation1
    operation2
  class2
    operation1
```

Mapping example

```
Mapping 1
```

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

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

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

```
Mapping 1 [M1]
  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
Purpose

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

Scheduler
- 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
Component Sample metamodel
**Use Case:**

“Component Sample”-to-UML model transformation and vice versa

**Mapping**

<table>
<thead>
<tr>
<th><strong>Component Sample entity</strong></th>
<th><strong>UML entity</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>ComponentModel</td>
<td>Model</td>
</tr>
<tr>
<td>ComponentPackage</td>
<td>Package</td>
</tr>
<tr>
<td>SoftwareComponent</td>
<td>Component</td>
</tr>
<tr>
<td>HardwareComponent</td>
<td>Class</td>
</tr>
</tbody>
</table>
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: ComponentSample Model to UML Model
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
- CompleteRule: Qualified name to the class that implements a rule interface (IRule)
- Name: Default Component Model Rule
Step 2. Declaration of the Mapping structure 2/2

Domain helper class (DomainHelper)

- Implements IDomainHelper
- GetAllDomainClasses()
  - Returns 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
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
Step 4. Workflow 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.initTransformation
  - Name: Initialization of Transformation 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.InitializeTransformation)

Initialization Activity

- Responsible for preparing the resource where the UML model will be saved
Step 4. Workflow 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.
Open a Kitalpha IDE with on the workspace than contains the example

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

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

Mapping

<table>
<thead>
<tr>
<th>UML entity</th>
<th>component Sample entity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>ComponentModel</td>
</tr>
<tr>
<td>Component</td>
<td>SoftwareComponent</td>
</tr>
<tr>
<td>Package</td>
<td>ComponentPackage</td>
</tr>
<tr>
<td>Class</td>
<td>HardwareComponent</td>
</tr>
</tbody>
</table>
Example – UML to component sample

UI Transposer Launch configuration – Mapping choose

Name of the configuration

Cadence activities

Create a new configuration

Mapping name

Mapping purpose

Create a new configuration
UI Transposer Launch configuration – Cadence activities

Transposer workflow elements

- Pre-analysis activities
- Add a new activity
- Remove an activity
- Change the order of activities
- Edit the parameters of selected activity
UI Transposer Launch configuration – Activity parameters

<table>
<thead>
<tr>
<th>Name of the parameter</th>
<th>Value of the parameter. (Editable field)</th>
<th>Validity information</th>
</tr>
</thead>
<tbody>
<tr>
<td>TraceModelPath</td>
<td>/org.polarsys.kitalpha.transposer.componentsample.uml.example/traces/trace.traces</td>
<td></td>
</tr>
</tbody>
</table>

Description

Validation

OK

Message(s)
Thank You!

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

https://polarsys.org/wiki/Kitalpha

benoit.langlois@thalesgroup.com

#LangloisBenoit