EGF Tutorial
EMF Generation Patterns

Benoît Langlois – Thales/EPM

Agenda

- Introduction
- Process
- Benefits of the Pattern Technique
- Best Practices for the EMF Generation

THALES

What is EGF?

- EGF (Eclipse Generation Factories) is an Eclipse open source project under the **EMFT project**

- **Purpose**: providing a **model-based generation framework**
- **Operational objectives**:
  - Supporting complex, large-scale and customizable generations
  - Promoting the constitution of generation portfolios in order to capitalize on generation solutions
  - Providing an extensible generation structure

THALES

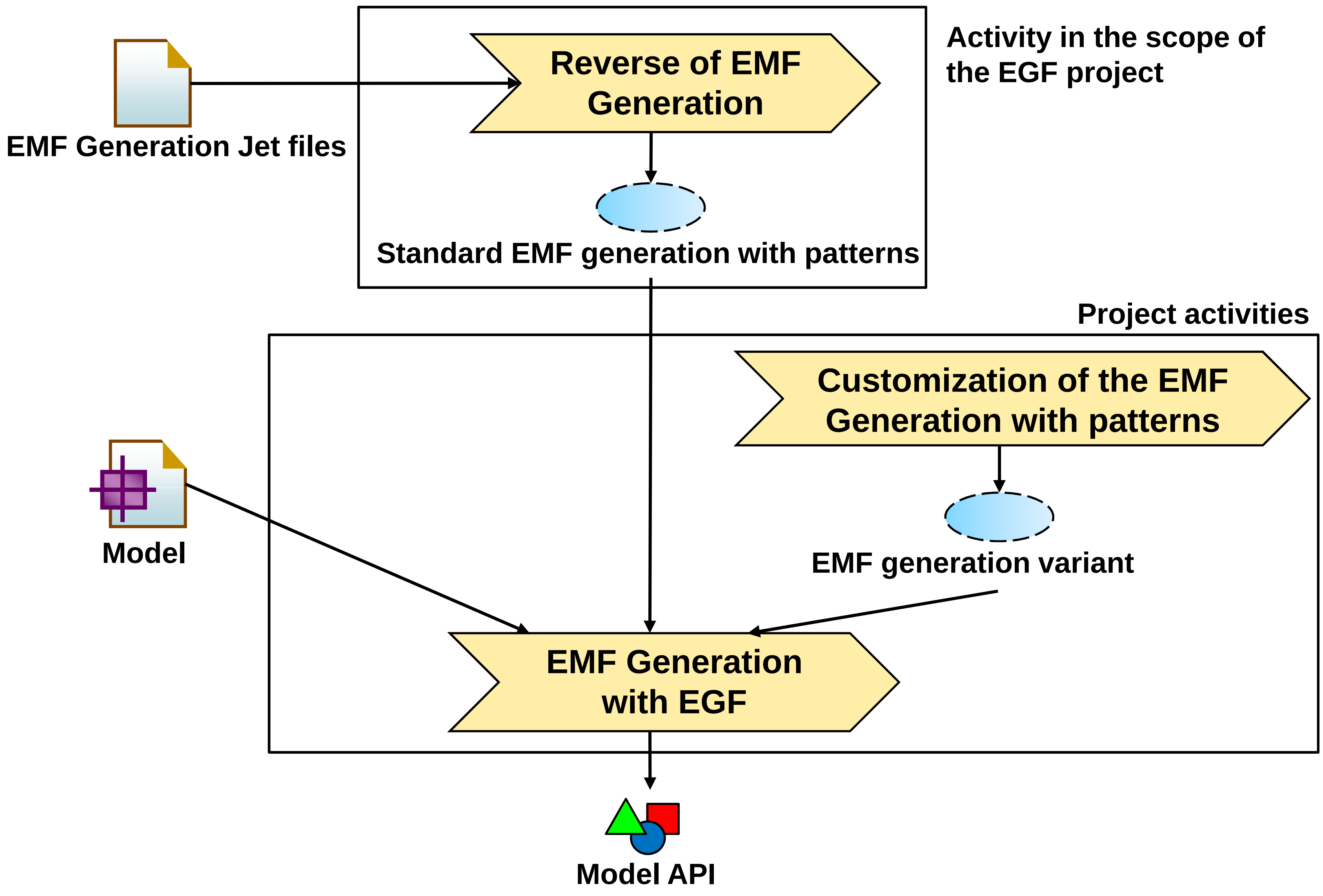
EMF Generation and EGF

- **The EMF generation is central for model-based developments,**
- **But limits exist to the EMF generation today, e.g.:**
  - Some Jet files are monolithic, problem of readability due to the generation complexity
  - Reuse / customization: problem of capitalization and scalability for large-scale applications with common and specific needs
- **Work realized with EGF:**
  - Transformation of the Jet files for the EMF generation into patterns
- **Added-value:**
  - Clarification of the EMF generation
  - Taking profit from the pattern technique

Agenda

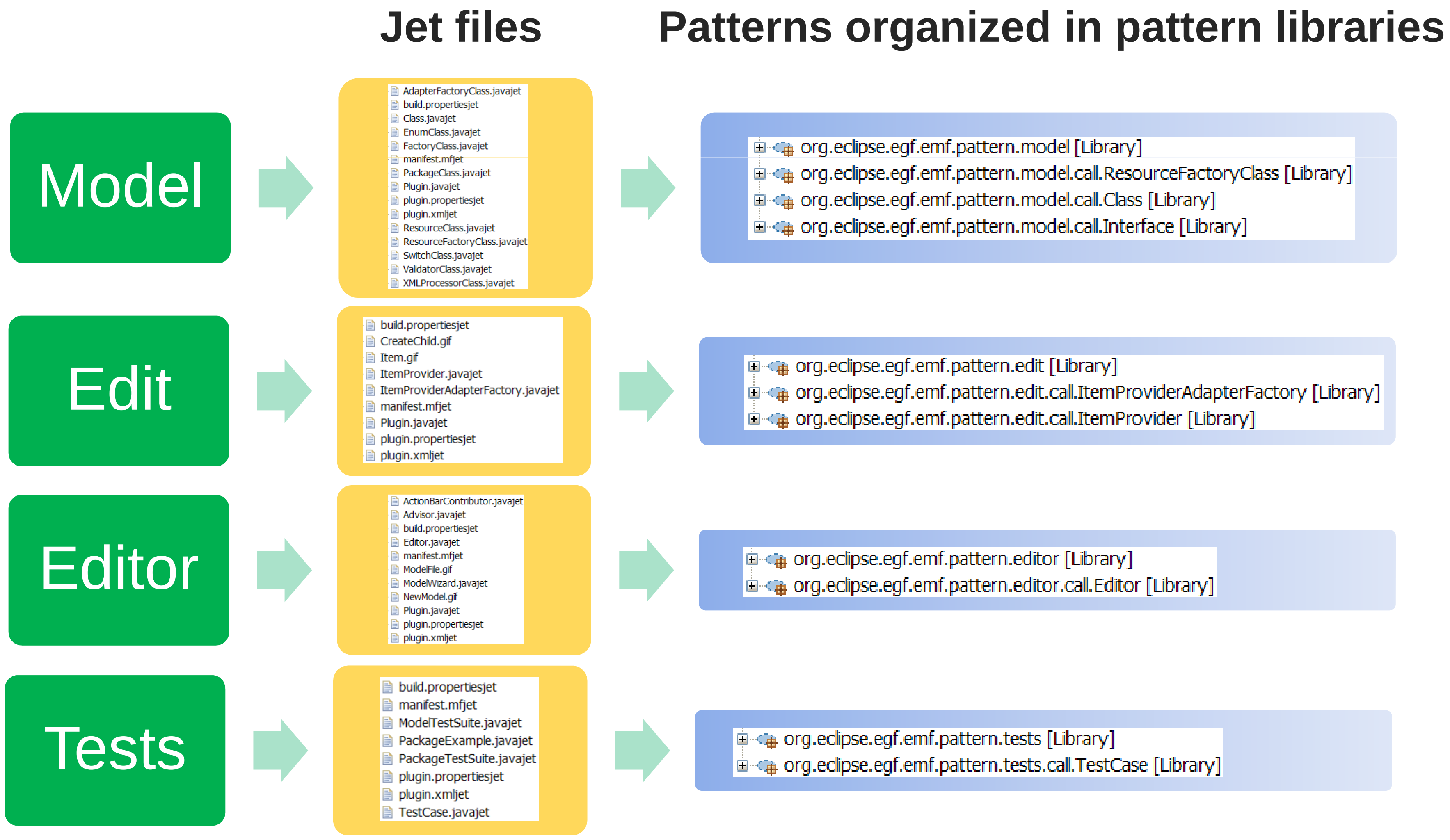
THALES

Process Overview



Activity in the scope of the EGF project
EMF Generation Jet files
Reverse of EMF Generation
Standard EMF generation with patterns
Project activities
Customization of the EMF Generation with patterns
EMF generation variant
Model
EMF Generation with EGF
Model API

Reverse of the EMF Generation into Patterns



Model – Jet files:

- AdapterFactoryClass.javajet
- build.propertiesjet
- Class.javajet
- EnumClass.javajet
- FactoryClass.javajet
- manifest.mfjet
- PackageClass.javajet
- Plugin.javajet
- plugin.propertiesjet
- plugin.xmljet
- ResourceClass.javajet
- ResourceFactoryClass.javajet
- SwitchClass.javajet
- ValidatorClass.javajet
- XMLProcessorClass.javajet

Model – Pattern libraries:

- org.eclipse.egf.emf.pattern.model [Library]
- org.eclipse.egf.emf.pattern.model.call.ResourceFactoryClass [Library]
- org.eclipse.egf.emf.pattern.model.call.Class [Library]
- org.eclipse.egf.emf.pattern.model.call.Interface [Library]

Edit – Jet files:

- build.propertiesjet
- CreateChild.gif
- Item.gif
- ItemProvider.javajet
- ItemProviderAdapterFactory.javajet
- manifest.mfjet
- Plugin.javajet
- plugin.propertiesjet
- plugin.xmljet

Edit – Pattern libraries:

- org.eclipse.egf.emf.pattern.edit [Library]
- org.eclipse.egf.emf.pattern.edit.call.ItemProviderAdapterFactory [Library]
- org.eclipse.egf.emf.pattern.edit.call.ItemProvider [Library]

Editor – Jet files:

- ActionBarContributor.javajet
- Advisor.javajet
- build.propertiesjet
- Editor.javajet
- manifest.mfjet
- ModelFile.gif
- ModelWizard.javajet
- NewModel.gif
- Plugin.javajet
- plugin.propertiesjet
- plugin.xmljet

Editor – Pattern libraries:

- org.eclipse.egf.emf.pattern.editor [Library]
- org.eclipse.egf.emf.pattern.editor.call.Editor [Library]

Tests – Jet files:

- build.propertiesjet
- manifest.mfjet
- ModelTestSuite.javajet
- PackageExample.javajet
- PackageTestSuite.javajet
- plugin.propertiesjet
- plugin.xmljet
- TestCase.javajet

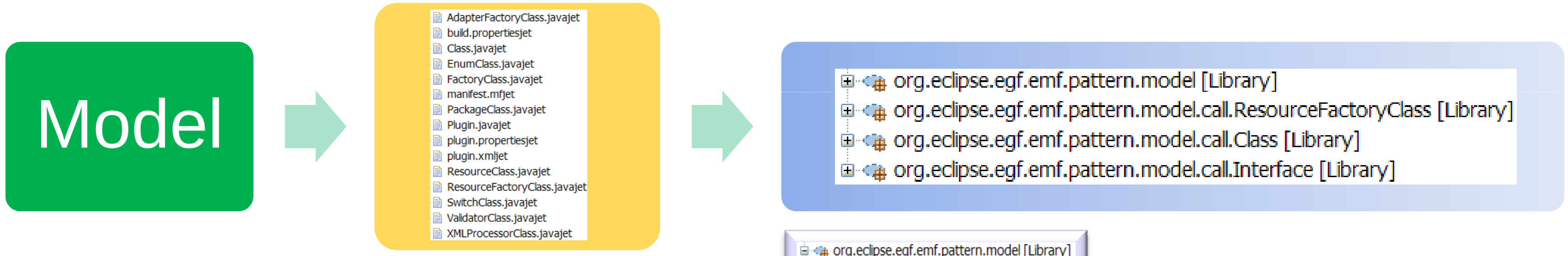
Tests – Pattern libraries:

- org.eclipse.egf.emf.pattern.tests [Library]
- org.eclipse.egf.emf.pattern.tests.call.TestCase [Library]

THALES

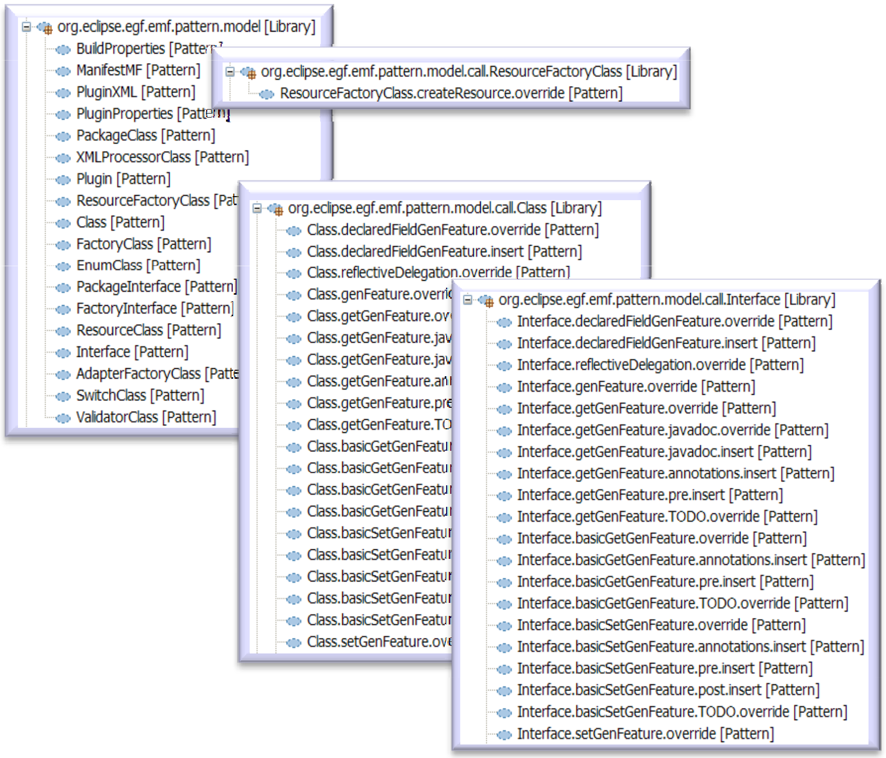
Reverse of EMF Generation

Jet files **Patterns organized in pattern libraries**



AdapterFactoryClass.javajet
build.propertiesjet
Class.javajet
EnumClass.javajet
FactoryClass.javajet
manifest.mfjet
PackageClass.javajet
Plugin.javajet
plugin.propertiesjet
plugin.xmljet
ResourceClass.javajet
ResourceFactoryClass.javajet
SwitchClass.javajet
ValidatorClass.javajet
XMLProcessorClass.javajet

org.eclipse.egf.emf.pattern.model [Library]
org.eclipse.egf.emf.pattern.model.call.ResourceFactoryClass [Library]
org.eclipse.egf.emf.pattern.model.call.Class [Library]
org.eclipse.egf.emf.pattern.model.call.Interface [Library]



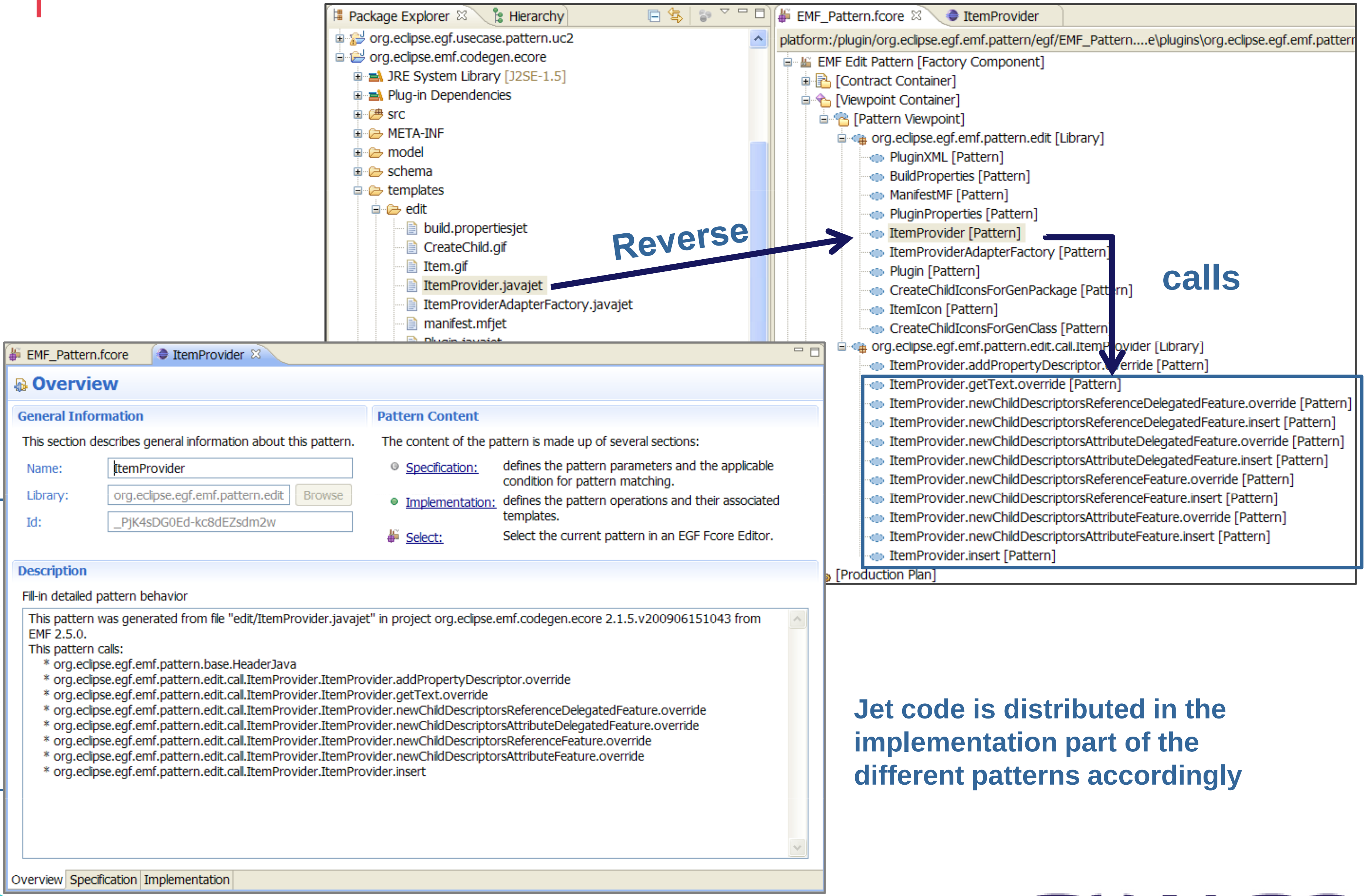
org.eclipse.egf.emf.pattern.model [Library]
- BuildProperties [Patter
- ManifestMF [Pattern]
- PluginXML [Pattern]
- PluginProperties [Pattern]
- PackageClass [Pattern]
- XMLProcessorClass [Pattern]
- Plugin [Pattern]
- ResourceFactoryClass [Pat
- Class [Pattern]
- FactoryClass [Pattern]
- EnumClass [Pattern]
- PackageInterface [Pattern]
- FactoryInterface [Pattern]
- ResourceClass [Pattern]
- Interface [Pattern]
- AdapterFactoryClass [Patte
- SwitchClass [Pattern]
- ValidatorClass [Pattern]

org.eclipse.egf.emf.pattern.model.call.ResourceFactoryClass [Library]
- ResourceFactoryClass.createResource.override [Pattern]

org.eclipse.egf.emf.pattern.model.call.Class [Library]
- Class.declaredFieldGenFeature.override [Pattern]
- Class.declaredFieldGenFeature.insert [Pattern]
- Class.reflectiveDelegation.override [Pattern]
- Class.genFeature.overric
- Class.getGenFeature.ov
- Class.getGenFeature.jav
- Class.getGenFeature.jav
- Class.getGenFeature.an
- Class.getGenFeature.pre
- Class.getGenFeature.TO
- Class.basicGetGenFeatu
- Class.basicGetGenFeatu
- Class.basicGetGenFeatu
- Class.basicGetGenFeatu
- Class.basicSetGenFeatu
- Class.basicSetGenFeatu
- Class.basicSetGenFeatu
- Class.basicSetGenFeatu
- Class.basicSetGenFeatu
- Class.setGenFeature.ove

org.eclipse.egf.emf.pattern.model.call.Interface [Library]
- Interface.declaredFieldGenFeature.override [Pattern]
- Interface.declaredFieldGenFeature.insert [Pattern]
- Interface.reflectiveDelegation.override [Pattern]
- Interface.genFeature.override [Pattern]
- Interface.getGenFeature.override [Pattern]
- Interface.getGenFeature.javadoc.override [Pattern]
- Interface.getGenFeature.javadoc.insert [Pattern]
- Interface.getGenFeature.annotations.insert [Pattern]
- Interface.getGenFeature.pre.insert [Pattern]
- Interface.getGenFeature.TODO.override [Pattern]
- Interface.basicGetGenFeature.override [Pattern]
- Interface.basicGetGenFeature.annotations.insert [Pattern]
- Interface.basicGetGenFeature.pre.insert [Pattern]
- Interface.basicGetGenFeature.TODO.override [Pattern]
- Interface.basicSetGenFeature.override [Pattern]
- Interface.basicSetGenFeature.annotations.insert [Pattern]
- Interface.basicSetGenFeature.pre.insert [Pattern]
- Interface.basicSetGenFeature.post.insert [Pattern]
- Interface.basicSetGenFeature.TODO.override [Pattern]
- Interface.setGenFeature.override [Pattern]

Example of Decomposition



```
⊞ org.eclipse.egf.usecase.pattern.uc2
⊟ org.eclipse.emf.codegen.ecore
   ⊞ JRE System Library [J2SE-1.5]
   ⊞ Plug-in Dependencies
   ⊞ src
   ⊞ META-INF
   ⊞ model
   ⊞ schema
   ⊟ templates
      ⊟ edit
         build.propertiesjet
         CreateChild.gif
         Item.gif
         ItemProvider.javajet
         ItemProviderAdapterFactory.javajet
         manifest.mfjet
```

```
platform:/plugin/org.eclipse.egf.emf.pattern/egf/EMF_Pattern....e\plugins\org.eclipse.egf.emf.patter
⊟ EMF Edit Pattern [Factory Component]
   ⊞ [Contract Container]
   ⊟ [Viewpoint Container]
      ⊟ [Pattern Viewpoint]
         ⊟ org.eclipse.egf.emf.pattern.edit [Library]
            PluginXML [Pattern]
            BuildProperties [Pattern]
            ManifestMF [Pattern]
            PluginProperties [Pattern]
            ItemProvider [Pattern]
            ItemProviderAdapterFactory [Pattern]
            Plugin [Pattern]
            CreateChildIconsForGenPackage [Pattern]
            ItemIcon [Pattern]
            CreateChildIconsForGenClass [Pattern]
         ⊟ org.eclipse.egf.emf.pattern.edit.call.ItemProvider [Library]
            ItemProvider.addPropertyDescriptor.override [Pattern]
            ItemProvider.getText.override [Pattern]
            ItemProvider.newChildDescriptorsReferenceDelegatedFeature.override [Pattern]
            ItemProvider.newChildDescriptorsReferenceDelegatedFeature.insert [Pattern]
            ItemProvider.newChildDescriptorsAttributeDelegatedFeature.override [Pattern]
            ItemProvider.newChildDescriptorsAttributeDelegatedFeature.insert [Pattern]
            ItemProvider.newChildDescriptorsReferenceFeature.override [Pattern]
            ItemProvider.newChildDescriptorsReferenceFeature.insert [Pattern]
            ItemProvider.newChildDescriptorsAttributeFeature.override [Pattern]
            ItemProvider.newChildDescriptorsAttributeFeature.insert [Pattern]
            ItemProvider.insert [Pattern]
   [Production Plan]
```

Overview

General Information

This section describes general information about this pattern.

Name:	ItemProvider
Library:	org.eclipse.egf.emf.pattern.edit
Id:	_PjK4sDG0Ed-kc8dEZsdm2w

Pattern Content

The content of the pattern is made up of several sections:

Specification:	defines the pattern parameters and the applicable condition for pattern matching.
Implementation:	defines the pattern operations and their associated templates.
Select:	Select the current pattern in an EGF Fcore Editor.

Description

Fill-in detailed pattern behavior

```
This pattern was generated from file "edit/ItemProvider.javajet" in project org.eclipse.emf.codegen.ecore 2.1.5.v200906151043 from
EMF 2.5.0.
This pattern calls:
   * org.eclipse.egf.emf.pattern.base.HeaderJava
   * org.eclipse.egf.emf.pattern.edit.call.ItemProvider.ItemProvider.addPropertyDescriptor.override
   * org.eclipse.egf.emf.pattern.edit.call.ItemProvider.ItemProvider.getText.override
   * org.eclipse.egf.emf.pattern.edit.call.ItemProvider.ItemProvider.newChildDescriptorsReferenceDelegatedFeature.override
   * org.eclipse.egf.emf.pattern.edit.call.ItemProvider.ItemProvider.newChildDescriptorsAttributeDelegatedFeature.override
   * org.eclipse.egf.emf.pattern.edit.call.ItemProvider.ItemProvider.newChildDescriptorsReferenceFeature.override
   * org.eclipse.egf.emf.pattern.edit.call.ItemProvider.ItemProvider.newChildDescriptorsAttributeFeature.override
   * org.eclipse.egf.emf.pattern.edit.call.ItemProvider.ItemProvider.insert
```

Overview | Specification | Implementation

Jet code is distributed in the implementation part of the different patterns accordingly

Agenda

● Introduction

● Process

● Benefits of the Pattern Technique

● Best Practices for the EMF Generation

Pattern – Definition

- **Definition:**
  - Definition #1 – Rationale: A pattern is a solution to a recurrent problem
  - Definition #2 – Structural: A pattern is a formalism to express systematic behavior

- **Key points:**
  - A pattern conforms to a language and is executable
  - The pattern specification reflects the external view (e.g., parameters), while pattern implementation reflects the internal view (e.g., methods)

- **Introduction to patterns:**
  - Tutorials: “EGF Tutorial”, “Reuse and Customization”
  - Examples: Pattern Use Cases 1 and 2, EMF generation use cases

THALES

Benefits for the EMF Generation

- **Pattern = generation unit**
  - Interest of the problem decomposition by pattern
- **Extensibility**
  - Interest of the pattern inheritance and pattern call mechanisms
  - Ability to change / extend patterns by a substitution mechanism
  - Ability to combine patterns written in different languages
- **Team management**
  - Possibility to have several contributors
- **Toward product lines**
  - Autonomy to create generation variants

THALES

Agenda

- Introduction
- Process
- Benefits of the Pattern Technique
- Best Practices for the EMF Generation

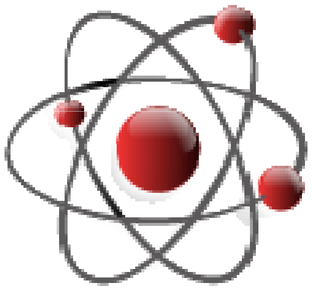
THALES

Best Practices with Patterns

- **Next slides present best practices that can be introduced in the EMF generation**

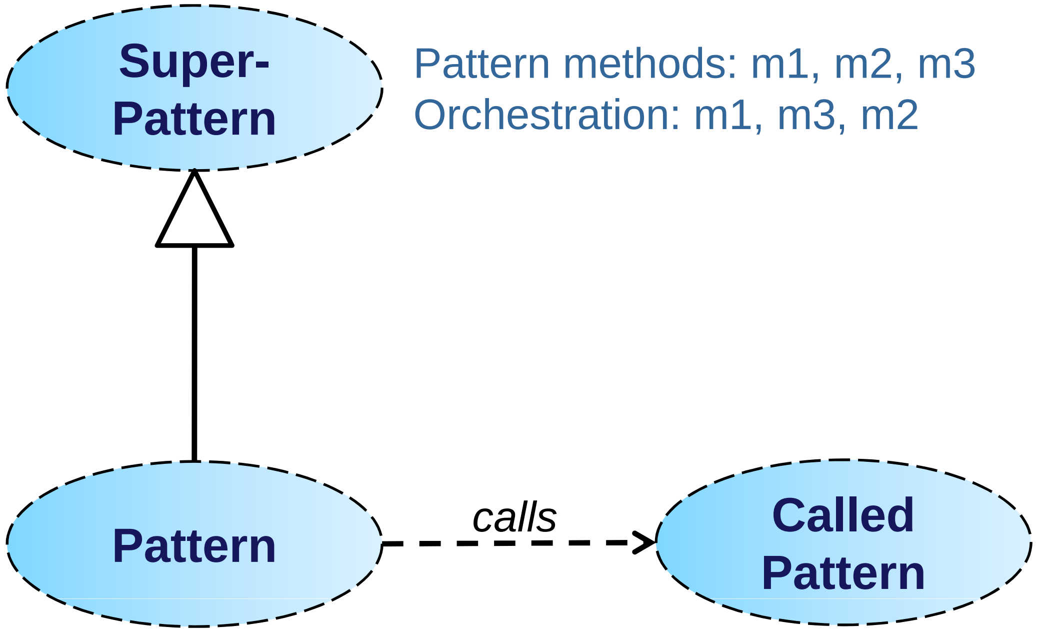
- **Level of confidence:**
  - ★☆☆ Tested
  - ★★☆ Experimented
  - ★★★ Operational

THALES



Pattern Best Practices

★★★ Pattern Adaptation



Pattern methods: m1, m2, m3
Orchestration: m1, m3, m2

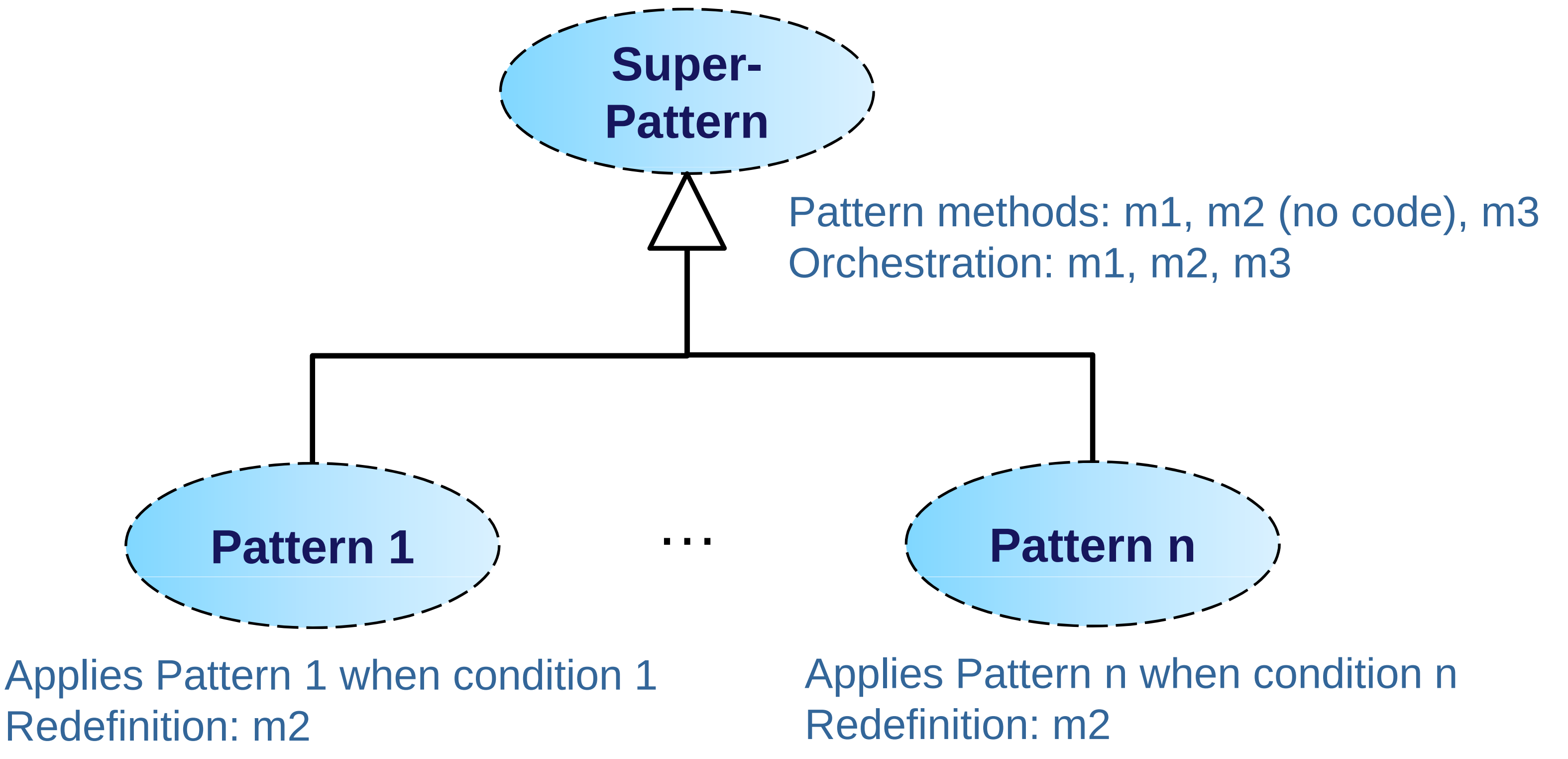
Pattern methods: m1, m4
Orchestration: applies super-pattern orchestration, called pattern, m4

Pattern Adaptation: Reusing orchestration of the parent pattern and adding orchestration specificities (e.g., method polymorphism, pattern call)

Example: Redefinition of the getText pattern

THALES

Pattern Alternative

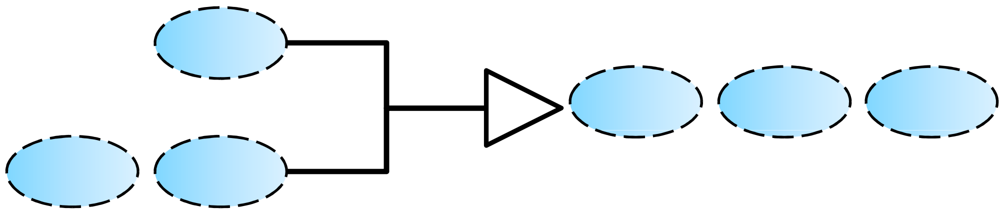


Super-Pattern
Pattern methods: m1, m2 (no code), m3
Orchestration: m1, m2, m3
Pattern 1
…
Pattern n
Applies Pattern 1 when condition 1
Redefinition: m2
Applies Pattern n when condition n
Redefinition: m2

Pattern Alternative: A super-pattern defines a prototype; a sub-pattern is applied when its condition is satisfied; the prototype is redefined. Possibility of exclusive / inclusive alternatives.

Example: Method contents depends on metamodel conditions

★★★ Pattern Merge



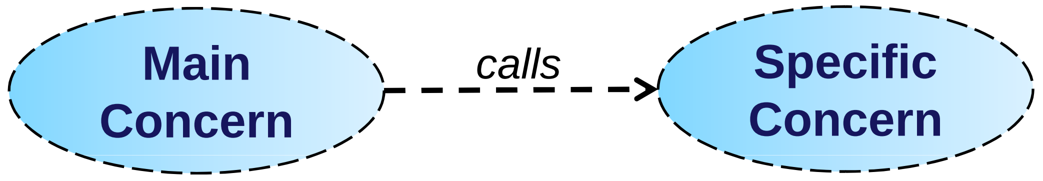
Merge operation

Pattern Merge: Merge of two pattern lists

Example: Combining standard and customized generations

THALES

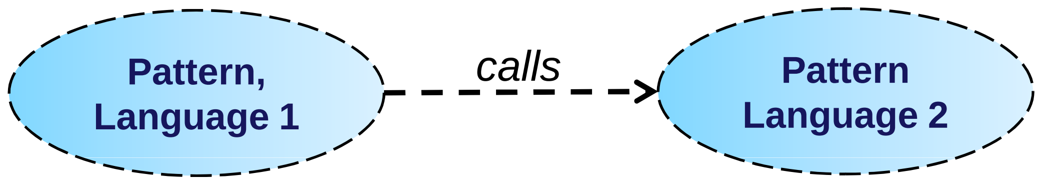
★☆☆ Separation of Concerns



Separation of Concerns: A standard generation delegates processing for a specific concern

Example: During an EMF generation, invocation of pattern for a text-to-text transformation based on an AST analysis in order to modify method content

★☆☆ Bridge of Language



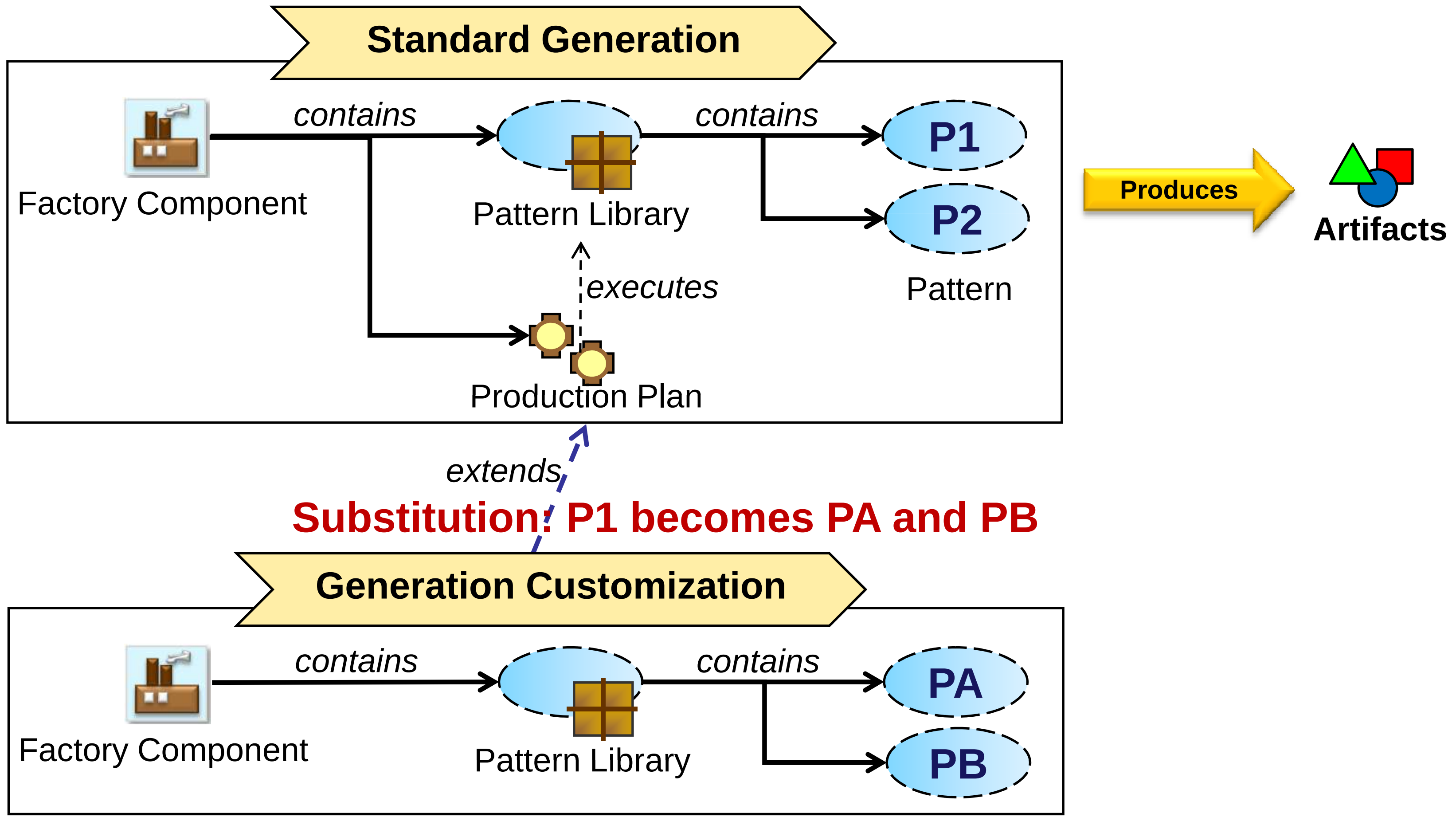
Bridge of Language: A pattern written in a language calls a pattern written in another language

Example: a Jet pattern calls a pattern in Java / in another model-to-text language

THALES

Portfolio Best Practices

★★★ Customization by Substitution

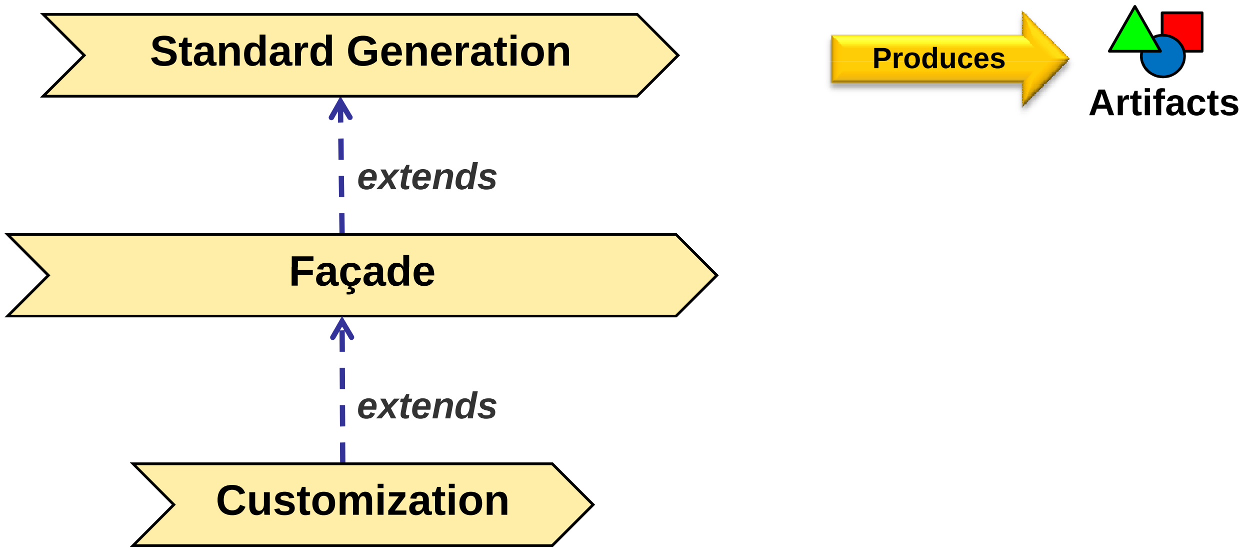


Customization by Substitution: Extension of a pattern-based standard generation with patterns for customization

Example: Redefinition of the insert/override patterns

THALES

★★☆ Generation Façade

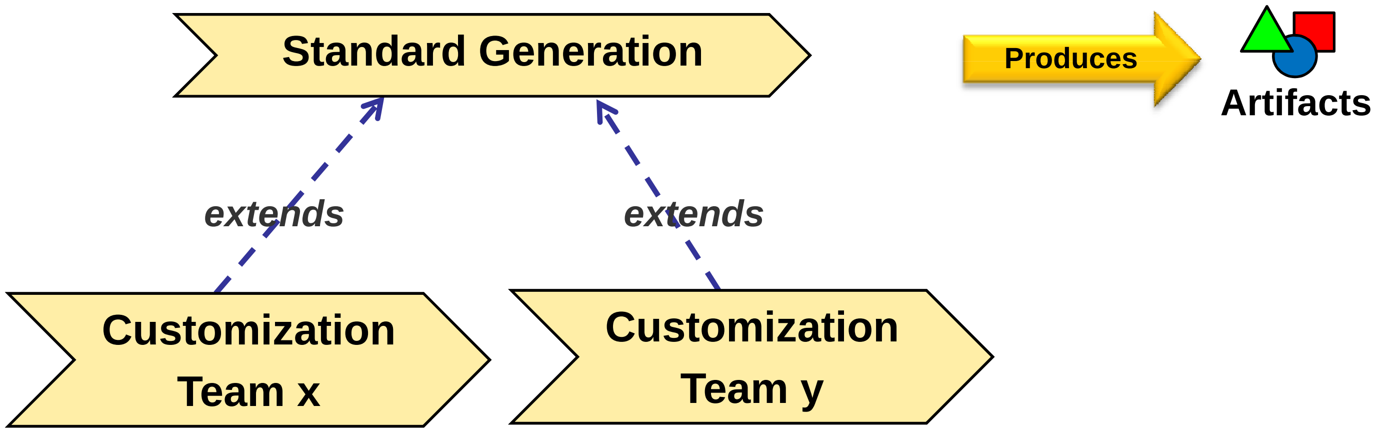


Generation Façade: A façade hides a standard generation and customization in the façade, and takes into account provided customizations

Example: Creation of a standard EMF generation for a company / department

Organizational Best Practices

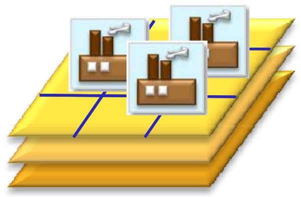
★★☆ Generation Variation



Generation Variation: Teams isolate and apply different generations based on the same standard generation

Example: Two teams extends differently the EMF generation

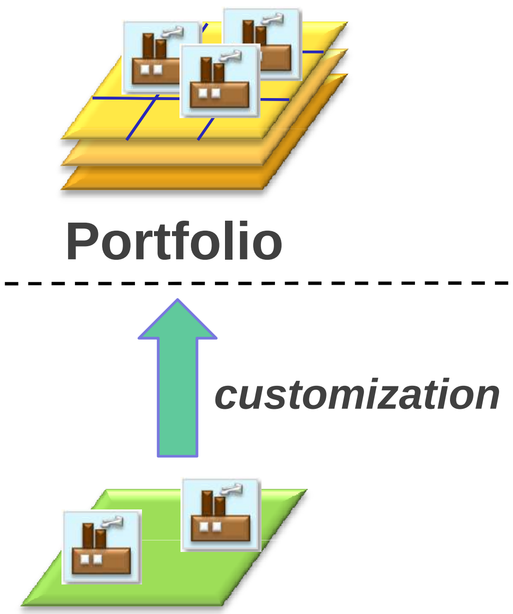
★★☆ Scenario of Customization in series



Portfolio

EMF Generation

★★☆ Scenario of Customization in series

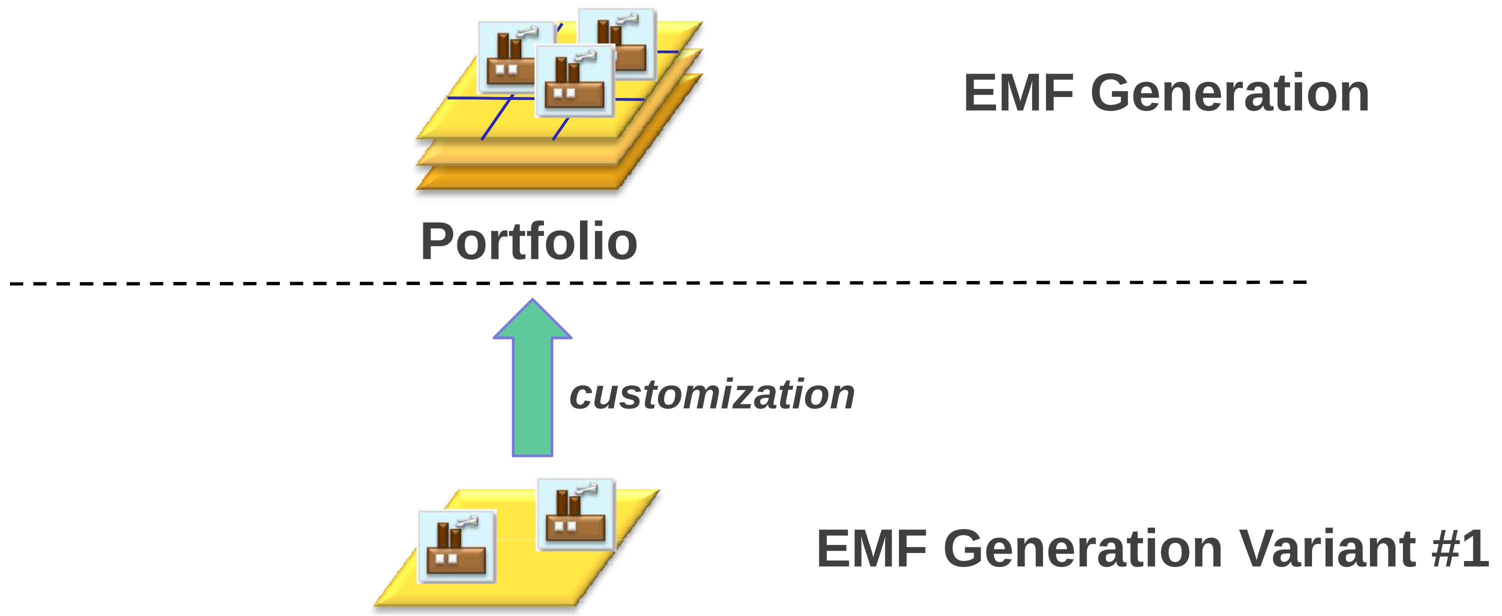


EMF Generation

Portfolio Adaptation
Team #1

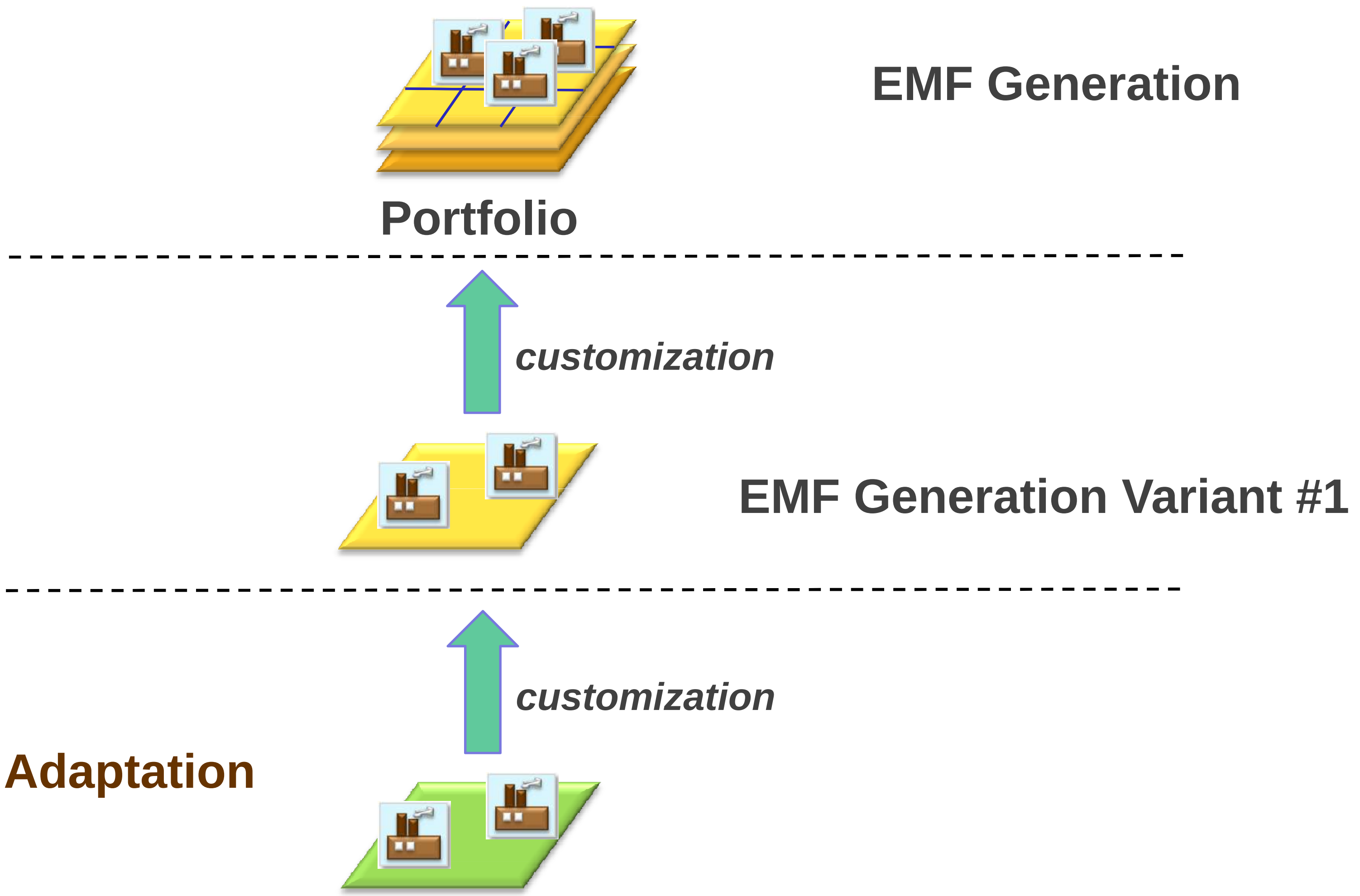
THALES

★★☆ Scenario of Customization in series



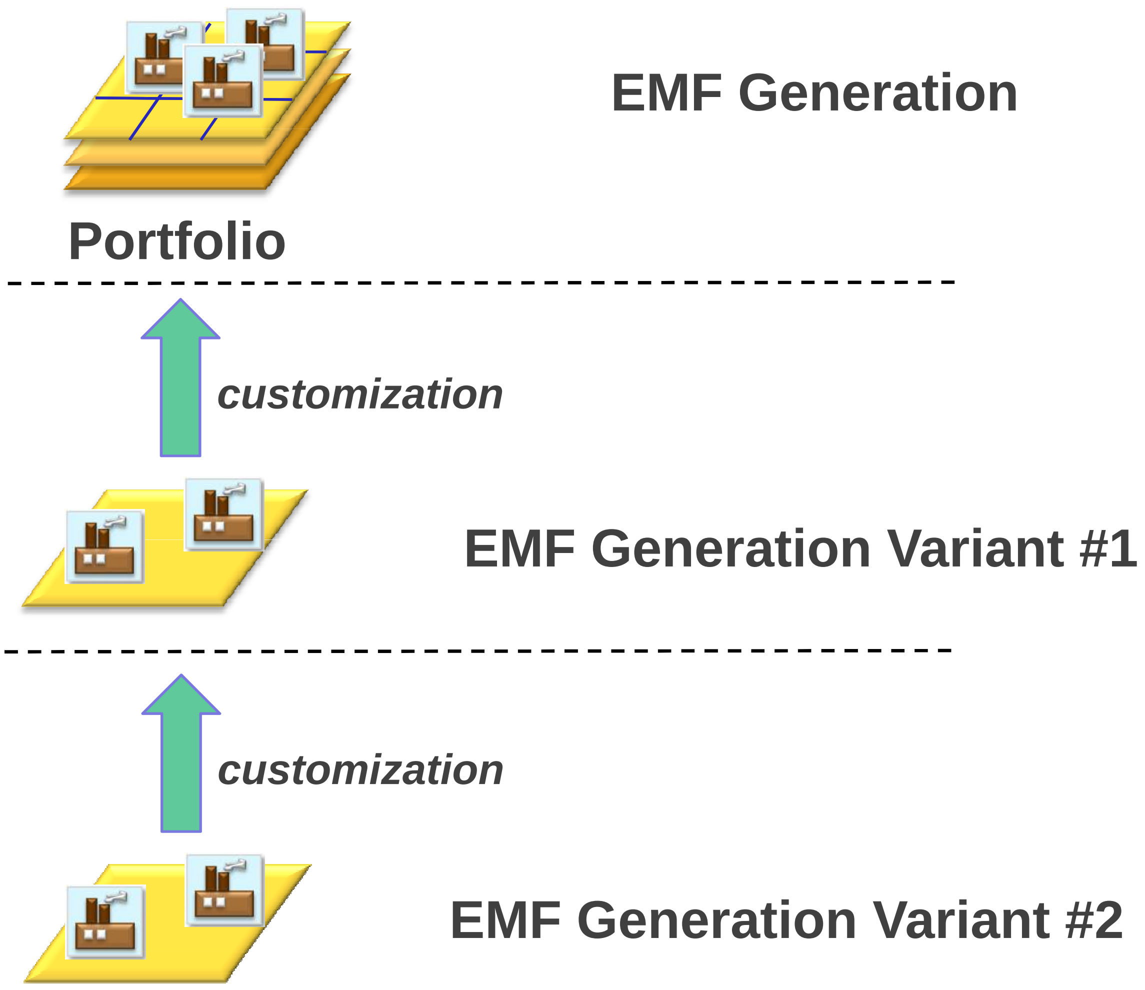
THALES

★★☆ Scenario of Customization in series



THALES

★★☆ Scenario of Customization in series



THALES