1. Introduction

2. Composer
   What is Composer?
   Principles

3. Example
1. Introduction

2. Composer
   - What is Composer?
   - Principles

3. Example
Code generation technologies are:
- Not flexible
- No control on the organization of generated files
- No way to adapt the generated code to specific libraries
- Not very customizable
- Mix of templates and code (i.e. templates are integrated in code of generation algorithms and vice versa)

Purpose

- Design a tool to overcome the limitations of code generation technologies by assembling, adapting and extending code generators
- The tool is not a code generator itself but based on existing one
- Easily customizable
- Separation of templates and algorithms
1 Introduction
2 Composer
   What is Composer?
   Principles
3 Example
Features

Composer is an Eclipse component that allows to:
• Organize the generated files
• Be independent of generation technology
• Separate templates and algorithms
1. Introduction
2. Composer
   What is Composer?
   Principles
3. Example
Big picture: Software architecture with Composer
Composer workflow

- **Generation strategies**: definition of one or more strategies to code generation
- **Generation refineries**: definition for each strategy a refinery that computes strategy properties
- **Generators**: register the generators and launch them
- **Additional Cadence activities** may be executed at Composer workflow element
Strategy concepts

Allocation Concept

- Generic and extensible meta-model to define a “generation plan”
- Defines the files and their structure
- Each file of allocation model is bound to one or more elements of the semantic model (types)
- All entities of allocation metamodel are abstract

The file that will be generated

The type to be included in the file

Reference to the semantic model element
Binding Concept

- Links the allocation meta-model to the semantic model
- Declaration of binding
  - Contribute to extension point: org.polarsys.kitalpha.cgm.allocation.binding
  - Provide:
    - Name: binding name [Required]
    - Id: binding identifier [Required]
    - NsUri: NSURI of the allocation metamodel [Required]
    - Description: binding description [Required]
  - Bind all the NsUri of the metamodels which the business model is conform to the allocation metamodel
    - NsUri: The NsUri of the metamodel to bind [required]
Strategy

• Creates the allocation model from a semantic model or a set of model elements
• Returns the Root of the allocation
• Implements IStrategy contract

Strategy declaration

o Contribute to extension point: org.polarsys.kitalpha.cgm.allocation.strategies
o Provide:
  ▪ Name : strategy name [Required]
  ▪ NsUri : NSURI of the allocation metamodel [Required]
  ▪ Id : strategy identifier [Required]
  ▪ Class : class that implements IStrategy [Required]
  ▪ Description : description of the strategy [Optional]
IStrategy contract
• Contract used to specify how the allocation model is created

The contract
  o Strategy parameters:
    ▪ Are used to customize, to adapt the strategy by the user
    ▪ getParameters(): Creates a map of parameters and return it.

  o Parameter validation:
    ▪ To validate Strategy parameters
    ▪ validateParameters(Map of parameters): Validate the parameters of the strategy. Returns the Map of the invalid parameters

  o Allocation model:
    ▪ Creation of the allocation model
    ▪ allocateModelElements(SemanticModelRoot, Map of parameters): returns the root of allocation model created from the semantic model root)
    ▪ allocateModelElements(SemanticModelRoot, Map of parameters, List of semantic model elements): return the root of allocation model created from the list of semantic model elements
Refinery concepts

Allocation Concept

- Works on allocation model and semantic model
- Satisfies the constraints of the generation
  - Reorder the elements in one file
  - Identify the dependencies (imports, includes declarations)
- Computes and fills the specific properties of allocation model from the allocation model and the semantic model
- Implements IRefinery contract

Whereas we can define several strategies, we must define only one refinery per generated language
Refinery Declaration

- Contribute to extension point: org.polarsys.kitalpha.cgm.allocation.rafineries
- Provide:
  - Name: the name of the refinery [Required]
  - NsUri: NSURI of the allocation metamodel [Required]
  - Id: refinery identifier [Required]
  - Class: class that implements IRefinery [Required]
  - Description: description of the refinery [Optional]
IRefinery contract
• Contract used to fills the specific properties of allocation model

The contract
  o Refinery parameters:
    ▪ Used to to customize, to adapt the refinery by the user
    ▪ getParameters(): Returns a map of the Refinery parameters
  o Parameter validation:
    ▪ To validate Refinery parameters
    ▪ validateParameters(Map of parameters): Validate the parameters of the Refinery. Returns the Map of the invalid parameters.
  o Refinery job
    ▪ To compute all specific properties of allocation model.
    ▪ refineModelElements(Allocation model root, Refinery parameters): Fill the allocation model with the specific target language and returns the root of allocation
Generation concepts

**Generator**
- Delegates the generation to any other code generation technologies
- Launch the generator
- The generation is based on information contained in the allocation model
- Implements the IGenerator contract

**Generator declaration**
- Contribute to extension point: `org.polarsys.kitalpha.cgm.cots.generators`
- Provide:
  - Name : generator name            [Required]
  - NsUri : NSURI of the allocation metamodel [Required]
  - Id : unique identifier of the generator [Required]
  - Class : class that launches the generation [Required]
  - Description : description of the generator [Optional]
IGenerator contract
- Used to launch the code generator

The contract
- Generator parameters
  - Used to customize, to adapt the generator by the user
  - getParameters(): Returns a map of the generator parameters
- Parameter validation
  - To validate Generator parameters
  - validateParameters(Map of parameters): Validate the parameters of the strategy. Returns the Map of the invalid parameters
- Generator
  - The start of the generator
  - generateCode(Allocation root model, generator parameters, output folder): Generates the allocation model in the output folder
Composer Activities

- Composer workflow is based on cadence
  - Declares four workflow elements:
    - **Before strategy**: Activities can be executed before strategy process (e.g., Validation of semantic model, adapt a semantic model)
    - **Before refinery**: Activities can be executed before the refinery process (e.g., Validation of allocation model)
    - **Before generation**: Activities can be executed before the generation task (e.g., clean the output directory)
    - **After generation**: Activities can be executed after the generation task (e.g., format of the code, Commit the code on SVN, compile the code)
Libraries

- Are allocation models
- Used to resolve the dependencies of the generated code if this one uses external code (e.g., includes)
- Implements ISearchAlgorithm contract
  - To get the path of an EObject in the allocation model
  - public String getIncludeFromAllocationModel(Root root, EObject object)
    - The method to implement
    - Computes the path of the object in the generated file and returns it
    - Returns null if the object is not found
Composer launch API – semantic model

- Composer can be launched on:
  A part of semantic model: list of elements from a semantic model
  One semantic model: resource contains the semantic model
  Many semantic models: resource set contains the semantic models

- Method
  ```java
  public void launch(
      final IStrategy strategy, //the strategy used for the generation
      final Map<String, Parameter> strategy_p, //the strategy parameters
      final IRefinery refinery, //the refinery used for the generation
      final Map<String, Parameter> refinery_p, //the refinery parameters
      final IGenerator generator, //the generator used for the generation
      final IPath path, //the output folder
      final Map<String, Parameter> generator_p, //the generator parameters
      final IGenerator generator, //the semantic model
      final Map<String, Parameter> generator_p, //the semantic model
      Boolean save //for saving the allocation model
  );
  ```

- The semantic model
  List<EObject> partOfModel | Resource model | ResourceSet models
Composer registry API

- The registry stores
  - Strategies
  - Refineries
  - Generators

- Queries on the registry
  - Get a strategy with its name or its ID
  - Get a refinery with its name or its ID
  - Get a generator with its name or its ID
### Path variables

- **$modelDir**
  - Refer the directory where the semantic model is located
  - Can be used to generate the generation plan in the model directory (or in sub-directory: $modelDir/my_generation)

- **$projectDir**
  - Refer the project where the semantic model is located
  - Can be used to generate the generation plan in the same directory project than the semantic model (ex: $projectDir/my_generation)

- **Contribute with a new variable**
  - Define the variable by implementing IComposerVariable contract
    - `getName()` method: returns the name of the variable (ex: `modelDir`)
    - `execute(Object)` method: Operation on the path when the variable is found and returns the path
  - Add the variable in the registry
    - `ComposerVariableInterpreter.INSTANCE.addNewVariable(VariableImpl);`
1. Introduction
2. Composer
   What is Composer?
   Principles
3. Example
Use Case #1

- **Generate Multi Files HTML Documentation**
  - Strategy one (Multi File Strategy)
    - For each component (Hardware, Software) create one HTML Page
    - Create index page which contains links to the component

Use Case #2

- **Generate One File HTML Documentation**
  - Strategy two (One File Strategy)
    - Create one HTML file which contains all documentation of all components
• **Step 1: Project Definition Action**
  
  o Create a new plugin: org.polarsys.kitalpha.m2t.componentsample.to.html
  
  o Add the dependencies:
    
    - org.eclipse.ui
    - org.eclipse.core.runtime
    - org.eclipse.emf.ecore
    - org.polarsys.kitalpha.composer.core
    - org.polarsys.kitalpha.composer.metamodel.allocation.base.model
    - org.polarsys.kitalpha.cadence.core
    - org.polarsys.kitalpha.vp.componentsample.model
• Step 2: Extend allocation base metamodel action
  o Create a new folder named model
  o Create a new.ecore model with nsuri: http://www.polarsys.org/kitalpha/componentsampleallocation/1.0.0
  o Load the allocation base metamodel
  o Create the extension as showed on the diagram
  o Generate the Java model for the extended base allocation metamodel
• **Step 3 – Composer Contribution Action (1/6)**

  o Bind the extended allocation metamodel to the business model
    • Contribute to org.polarsys.kitalpha.composer.allocation.binding
      • Name: Component Sample to HTML Documentation
      • Id: org.polarsys.kitalpha.m2t.componentsample.to.html.binding
      • NsUri: http://www.polarsys.org/kitalpha/componentsampleallocation/1.0.0
      • Description: Binding between Component Sample Ecore and Component Sample Allocation Ecore
      • Add new business metamodel nsuri declaration, and specify the nsuri: http://www.polarsys.org/kitalpha/ComponentSample

  o Multi files strategy contribution (Use Case 1)
    • Contribute to org.polarsys.kitalpha.composer.allocation.strategies
      • Name: HTML Component Sample Generation Multi Files
      • NsUri: http://www.polarsys.org/kitalpha/componentsampleallocation/1.0.0
      • Id: org.polarsys.kitalpha.m2t.componentsample.to.html.multi.files.strategy
      • Class: org.polarsys.kitalpha.m2t.componentsample.to.html.strategies.MultiFilesStrategy
      • Description: Multi Files Strategy
• Step 3 – Composer Contribution Action (2/6)
  o Strategy Multi Files Generation Code Example

```java
@override
public Root allocateModelElements(EObject modelRoot_p,
  Map<String, Parameter> strategyParams_p){

  ComponentSampleRoot root =
      ComponentSampleAllocationFactory.eINSTANCE.createComponentSampleRoot();

  //Create componentModelType
  File modelFile = createFile(modelRoot_p);
  root.getOwnedFiles().add(modelFile);

  Iterator<EObject> it = modelRoot_p.eAllContents();

  while (it.hasNext()){
    EObject currentChild = it.next();
    //Create file for each Software/Hardware component
    File file = createFile(currentChild);
    if (file != null)
      root.getOwnedFiles().add(file);
  }

  return root;
}
```
• Step 3 – Composer Contribution Action (3/6)

  o One File Generation contribution (Use Case 2)
    § Add a new strategy to strategies extension point
    • Right click on Strategies extension point defined before
    • New Strategy
    § Fill the fields:
    • Name: HTML Component Sample Generation One File
    • NsUri: http://www.polarsys.org/kitalpha/componentsampleallocation/1.0.0
    • Id: org.polarsys.kitalpha.m2t.componentsample.to.html.one.file.strategy
    • Class:
      org.polarsys.kitalpha.m2t.componentsample.to.html.strategies.OneFileStrategy
    • Description: One File Strategy

  o Strategy One Files Generation Code Example

```java
@Override
public root allocateModelElements(Object modelRoot_p,
                                    Map<String, Parameter> strategyParams_p)
{
    ComponentSampleRoot root = ComponentSampleAllocationFactory.eINSTANCE.createComponentSampleRoot();

    ComponentSampleFile file = ComponentSampleAllocationFactory.eINSTANCE.createComponentSampleFile();
    file.setName("index.html");
    file.setPath("/");
    Iterator<Object> it = modelRoot_p.eAllContents();
    while (it.hasNext())
    {
        Object next = it.next();
        if (next instanceof SoftwareComponent)
        {
            ComponentSampleSoftware software = ComponentSampleAllocationFactory.eINSTANCE.createComponentSampleSoftware();
            software.setReferencedElement(next);
            file.getOwnedTypes().add(software);
        }
        if (next instanceof HardwareComponent)
        {
            ComponentSampleHardware hardware = ComponentSampleAllocationFactory.eINSTANCE.createComponentSampleHardware();
            hardware.setReferencedElement(next);
            file.getOwnedTypes().add(hardware);
        }
    }
    root.getOwnedFiles().add(file);
    return root;
}
```
Step 3 – Composer Contribution Action (4/6)

- Refinery contribution
  - Contribute to: org.polarsys.kitalpha.composer.allocation.refineries
    - Name: Component Sample Refinery
    - NsUri: http://www.polarsys.org/kitalpha/componentsampleallocation/1.0.0
    - Id: org.polarsys.kitalpha.m2t.componentsample.to.html.refinery
    - Class: org.polarsys.kitalpha.m2t.componentsample.to.html.refineries.ComponentSampleRefinery
  - Description: Component Sample Refinery

- Refinery Code Example

```java
@Override
public Root refineModelElements(Root allocRoot_p,
    Map<String, Parameter> refineryParams_p)
{
    Set<ComponentSampleSoftware> softwares = new HashSet<ComponentSampleSoftware>();
    Set<ComponentSampleHardware> hardwares = new HashSet<ComponentSampleHardware>;
    ComponentModelType modelType = null;

    ComponentSampleRoot root = (ComponentSampleRoot)allocRoot_p;
    ComponentSampleAllocVisitor v = new ComponentSampleAllocVisitor();

    List<EObject> allContents = getAllContents(root);

    for (EObject eObject : allContents) {
        if (eObject instanceof ComponentModelType)
            modelType = (ComponentModelType)eObject;

        ComponentType type = v.doSwitch(eObject);

        if (type instanceof ComponentSampleHardware)
            hardwares.add((ComponentSampleHardware)type);
        else
            softwares.add((ComponentSampleSoftware)type);
    }

    addComponentTypesToModelType(softwares, hardwares, modelType);
    return root;
}
```
• Step 3 – Composer Contribution Action (5/6)

  o Generator contribution
    ▪ Contribute to: org.polarsys.kitalpha.cots.generators
      • Name: Component Sample Generator
      • NsUri: http://www.polarsys.org/kitalpha/componentsampleallocation/1.0.0
      • Id: org.polarsys.kitalpha.m2t.componentsample.to.html.cots
      • Class: org.polarsys.kitalpha.m2t.componentsample.to.html.sysout.generator.ComponentSampleGenerator
      • Description: Component Sample to Html Documentation generator

  o Generator Code Example
    ▪ For each file, launch the generator

```java
@Override
public void generateCode(Root allocRoot_p,
    Map<String, Parameter> generatorParams_p, IPath target_f) {

    HtmlDocGenerator genDoc = new HtmlDocGenerator();
    for (File file : allocRoot_p.getOwnedFiles()){
        StringBuffer page = new StringBuffer();

        initHtmlHeader(file, page);
        genDoc.generate(((ComponentSampleFile)file), target_f, page);
        createResource(file.getName(), target_f, page);
        setHtmlFooter(page);
    }
}
```
• **Step 3 – Composer Contribution Action (6/6)**
  
  o **Cadence activity – Cleaner of the folder generation**
    
    - Before the file generation, it is good to clean the folder where files are generated
    - Contribute with cadence activity to before generation Composer workflow element
      
      - Contribute to: org.polarsys.kitalpha.cadence.core.activity.declaration
      - Identifier: org.polarsys.kitalpha.m2t.componentsample.to.html.cleanActivity
      - Name: Folder Cleaner
      - WorkflowIdentifier: org.polarsys.kitalpha.composer.core.workflow
      - WorkflowElementIdentifier: org.polarsys.kitalpha.composer.core.workflow.beforegeneration
      - ActivityClass: org.polarsys.kitalpha.m2t.componentsample.to.html.activities.CleanGenFolder
      - Multiple: false
  
  o **Cadence activity code example**

```java
private void cleanFolder(final IPath target_f, final IProgressMonitor monitor) {
    final IFolder folder = ResourcesPlugin.getWorkspace().getRoot().getFolder(target_f);
    try {
        folder.accept(new IResourceVisitor()
            @Override
            public boolean visit(IResource resource) throws CoreException {
                if (!folder.getAbsolutePath().toString().equals(resource.getAbsolutePath().toString())) {
                    resource.delete(true, monitor);
                    return true;
                }
            });
        folder.getProject().refreshLocal(IResource.DEPTH_INFINITE, new NullProgressMonitor());
    } catch (CoreException e) { e.printStackTrace();
```
Create a new plugin – hierarchy code below

- Cadence activities
- Refinery activities
- Strategies package
- Small html generator
- EMF API
- Allocation metamodel
Composer extension points contributions

- Binding between Allocation and Component sample meta-model
- Strategies contribution
- Refinery contribution
- Generator contribution
- Cadence activity contribution
Play the example

- Launch a new instance of Kitalpha
- Create a new project or plugin
- Create a model folder
- Create a new ComponentSample model or import one.
- Create a Composer launch configurations
  - Choose a generation type
  - Choose a Strategy (One or Mutli files strategy)
  - Choose the Refinery
  - Choose the generator
  - Choose the folder where files will be generated
  - Add folder clean activity to “Before Generated” workflow element
  - Specify folder generation as parameter of the activity
- Right click on ComponentSample Model, then, Run Composer, then, componentSample to HTML Documentation
• Launch a new instance of Kitalpha
Composer configuration

- Name of the configuration
- To manage Cadence activities
- Select the strategy
- Strategy, refinery and generator parameter configurations
- Select the Refinery
- Select the Generator
- Select the output folder
- Get Help to use Composer variable

New composer configuration

Example
Thank You!

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

https://polarsys.org/wiki/Kitalpha

benoit.langlois@thalesgroup.com

#LangloisBenoit