

## FACTORY COMPONENT

---

Author: Benoît Langlois – benoit.langlois@thalesgroup.com

Version: 1.0

### DEFINITION

A factory component is a composite activity. Structurally, it contains a set of viewpoints and a production plan. Behaviorally, it delegates its activities to other factory components or tasks.

### OBJECTIVES



The objectives of a factory component are to:

- Provide reusable generation service.

The interests are to:

- Assemble and reuse generation activities.

### CONCERNS

 <b>Designer</b>	<ul style="list-style-type: none"><li>• The designer functionally assembles factory components and tasks.</li><li>• The designer executes factory components.</li></ul>
 <b>Developer</b>	<ul style="list-style-type: none"><li>• The developer develops the generation aspects which require code (e.g., patterns).</li></ul>

## STRUCTURE

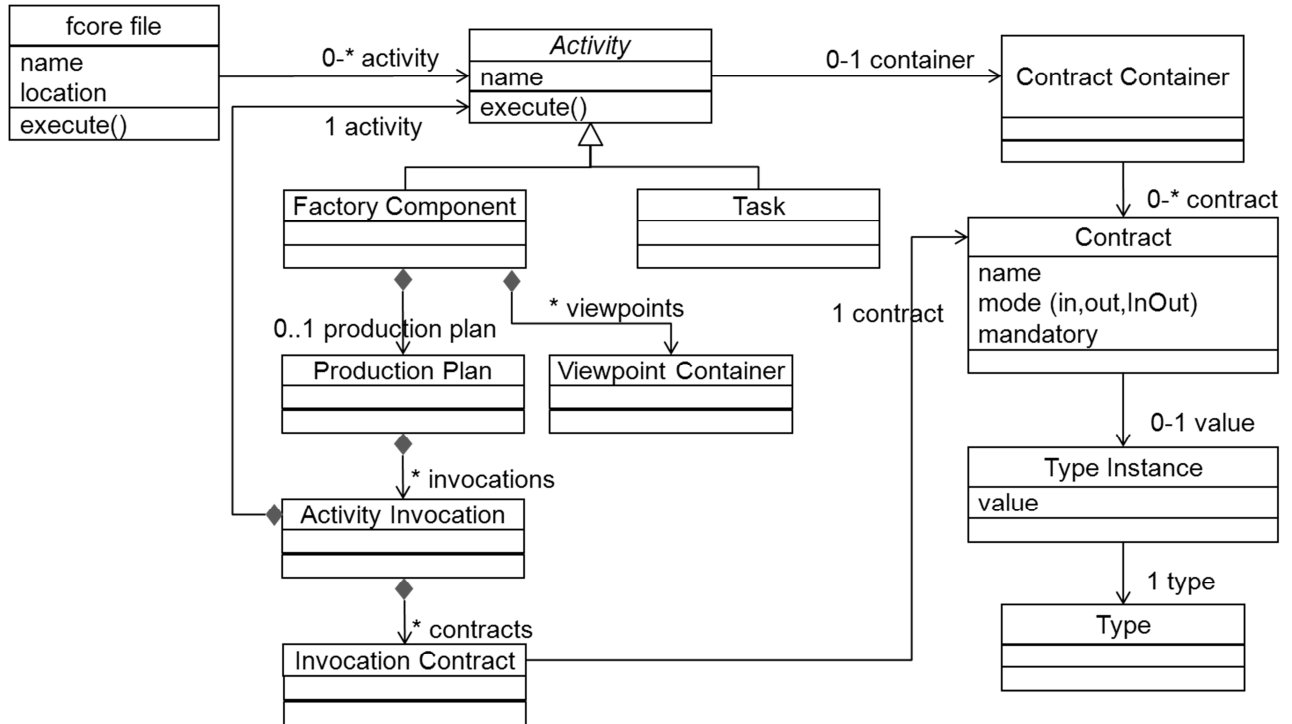


Figure 1. Factory component at the metamodel level

### Comprehension:

- A Factory Component is an Activity with:
  - A production plan as a workflow to assemble and invoke activities (i.e., Factory Components or Tasks).
  - A set of Viewpoints to describe aspects, such as domain declaration or patterns, used in the production plan.
- A Production Plan invokes a suite of Activities where each Activity Invocation provides a value to an identified Contract (e.g., parameter) of the invoked Activity. Then, a Factory Component is called with valued Contracts which can be reused later in Activity Invocations. The principle is the same for a Task which provides a valued Contract to a Task implementation. With the In/Out/InOut Contract mode, a parameter can have its value read or changed several times along the Production Plan.

## EXAMPLES

The following figure shows a Pricer which computes an amount from a price and quantity, displays the computed amount, and returns the amount. The computation and display steps are each time delegated to a specific task.

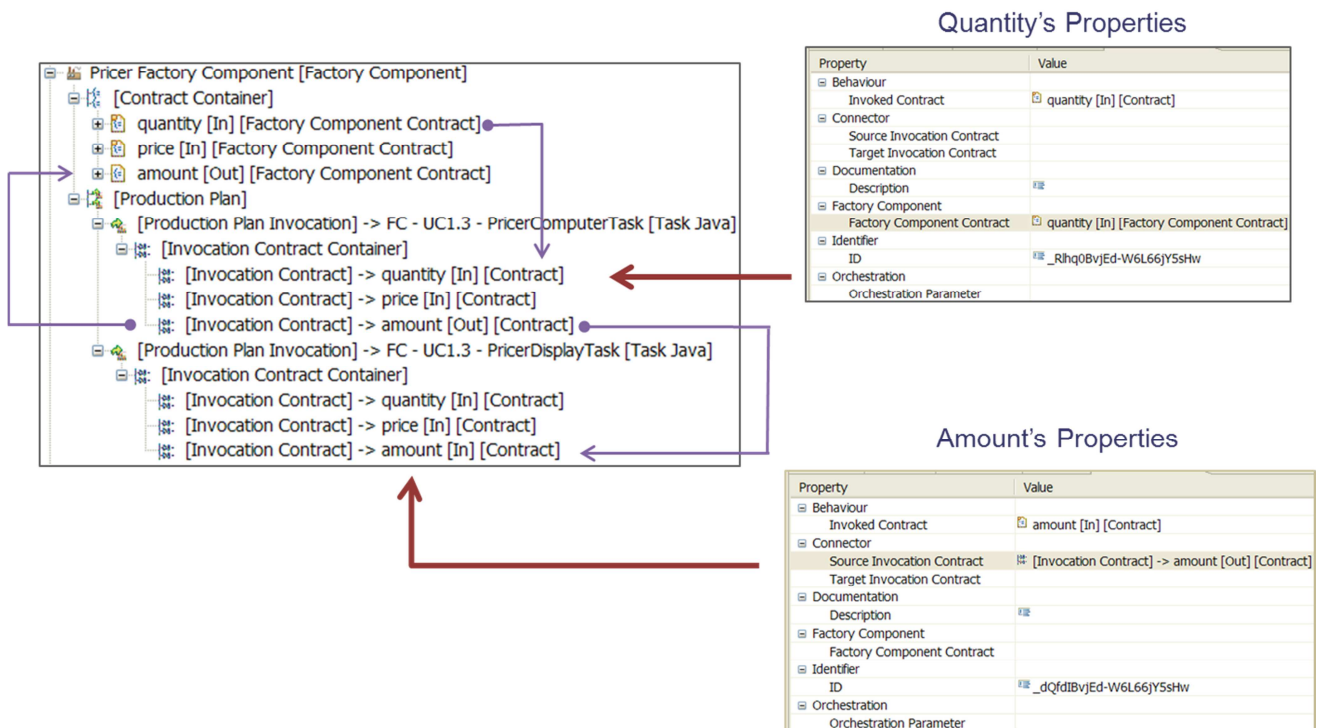


Figure 2. Example of Pricer with a Factory Component

The second example displays in a console the classes and attributes of the EGF Fcore model. The Factory Component contains a Viewpoint Container which declares the Fcore.ecore model as a Domain Viewpoint and Patterns in a Pattern Viewpoint. The Production Plan delegates the job to a "Domain Driven Pattern Strategy" Task which navigates over the Fcore model, applies the two patterns (i.e., classPattern and attributePattern), and reports the result to the console. The result is shown in the next figure.

Those examples are available in the Basic EGF Examples distributed with EGF.

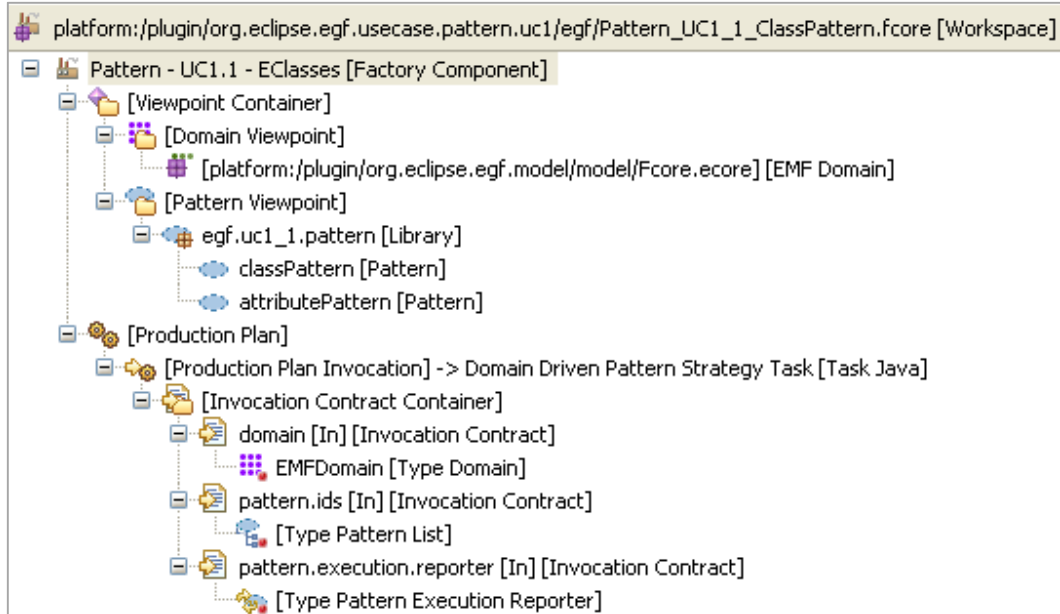


Figure 3. Example of model output with Patterns

```

-----
Result of pattern:

- Hello ModelElement Class
  - iD attribute
  - description attribute
- Hello NamedModelElement Class
  - name attribute
- Hello Activity Class
- Hello Contract Class
  - mandatory attribute
  - mode attribute
- Hello FactoryComponent Class
- Hello ContractContainer Class
- Hello FactoryComponentContract Class
- Hello ViewpointContainer Class
- Hello Viewpoint Class
- Hello Orchestration Class
- Hello OrchestrationParameterContainer Class
- Hello OrchestrationParameter Class
- Hello Invocation Class
- Hello InvocationContractContainer Class
- Hello InvocationContract Class
  
```

Figure 4. Result in the console

### DOMAIN TYPES

In the Domain Viewpoint are declared a set of domains (aka resources), instances of domain types. The following table details the standard domain types supported by EGF. A domain is potentially any kind of structured data representation.

<i>Domain Name</i>	<i>Description</i>
EMF Domain	EMF Ecore model
File System Domain	File and directories
Workspace Domain	Eclipse IFiles and IContainers

*Table 1. Standard Domains provided by EGF*

## PROCESS

The section presents the process dimension from the designer and developer viewpoints.

### Designer viewpoint

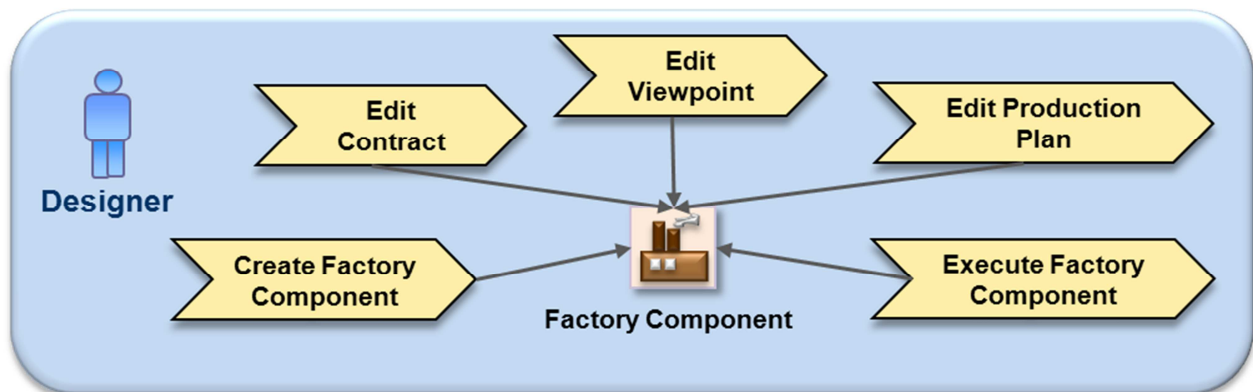


Figure 5. Process – Designer Viewpoint

<i>Create Factory Component</i>	The Designer creates a Factory Component in an fcore file.
<i>Edit Contract</i>	The Designer defines the Contracts (aka parameters) of the Factory Component.
<i>Edit Viewpoint</i>	The Designer edits a specific Viewpoint with its own formalism.
<i>Edit Production Plan</i>	The Designer edits the Production Plan in order to define the workflow of Contract values between the Factory Component Contracts and different Activity Invocations.
<i>Execute Factory Component</i>	The Designer executes the Factory Component.

Table 2. Designer activities

The Designer has to take care of two points:

- Factory Component naming: like in Java, a namespace in the Factory Component name may reflect a classification to organize Factory Components.
- Factory Component Contracts: they define the interface of the Factory Component.

**Developer viewpoint**

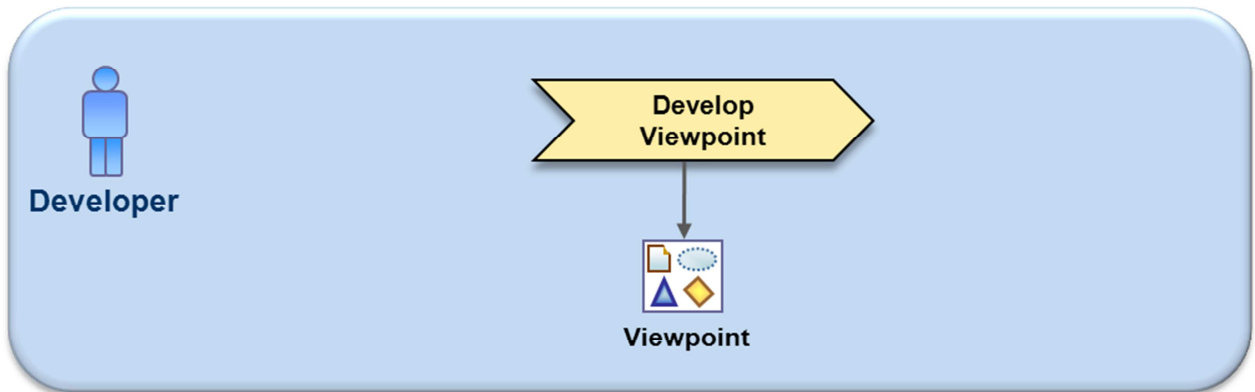


Figure 6. *Process – Developer Viewpoint*

<i>Develop Viewpoint</i>	The Developer develops all the viewpoint artifacts such as Patterns.
--------------------------	--

Table 2. *Developer activities*