Configuration Interface for MEssage ROuting

Conception détaillée

Date : 10/04/07
Version : 0.2
Statut : initial

Auteurs : **BAGNARD Natacha**
FOROT Julien

Table des révisions

Version	*Date*	*Modifications*
0.1	10/04/07	Création du document
0.2	09/07/07	Modifications suite au développement de l'application

Table des matières

1.Introduction

1.1.Objectifs du document

Ce document présente la conception détaillée du projet CIMERO Version 2.

Il suit la phase de conception globale et a pour but d'exposer l'organisation du développement de CIMERO Version 2.

1.2.Portée du document

Ce document servira de base au développement du produit à livrer.

Ce document est destiné:

- au MOAd : Jérôme Camilleri
- à la consultante : Martine Tasset
- au jury du Master 2 Pro GI pour l'évaluation du stage
- à l'équipe projet : Natacha Bagnard et Julien Forot

1.2.Documentation de référence

Ce document fait référence au Cahier des Charges du projet (« CahierDesCharges.odt ») et au dossier de conception globale (« ConceptionGlobale.odt »). Il est rédigé en fonction des clauses qualités définies dans le PAQL (« PlanAssuranceQualiteLogicielle.odt »).

1.3.Glossaire

--> voir section 1.3, Définition, Acronymes et Abréviations, du Cahier des Charges (« CahierDesCharges.odt »).

2.Framework GMF

Le modèle de données, représentation et mapping sont générés grâce à GMF. Cette section présente les modèles et le mapping utilisés pour générer le code[1]. Celui-ci sera ensuite présenté dans la section suivante.

2.1.Modèle des données, EMF

Le modèle de données ci-dessous correspond au contenu du fichier ecore_diagram.

2.1.1.Définition du modèle de données

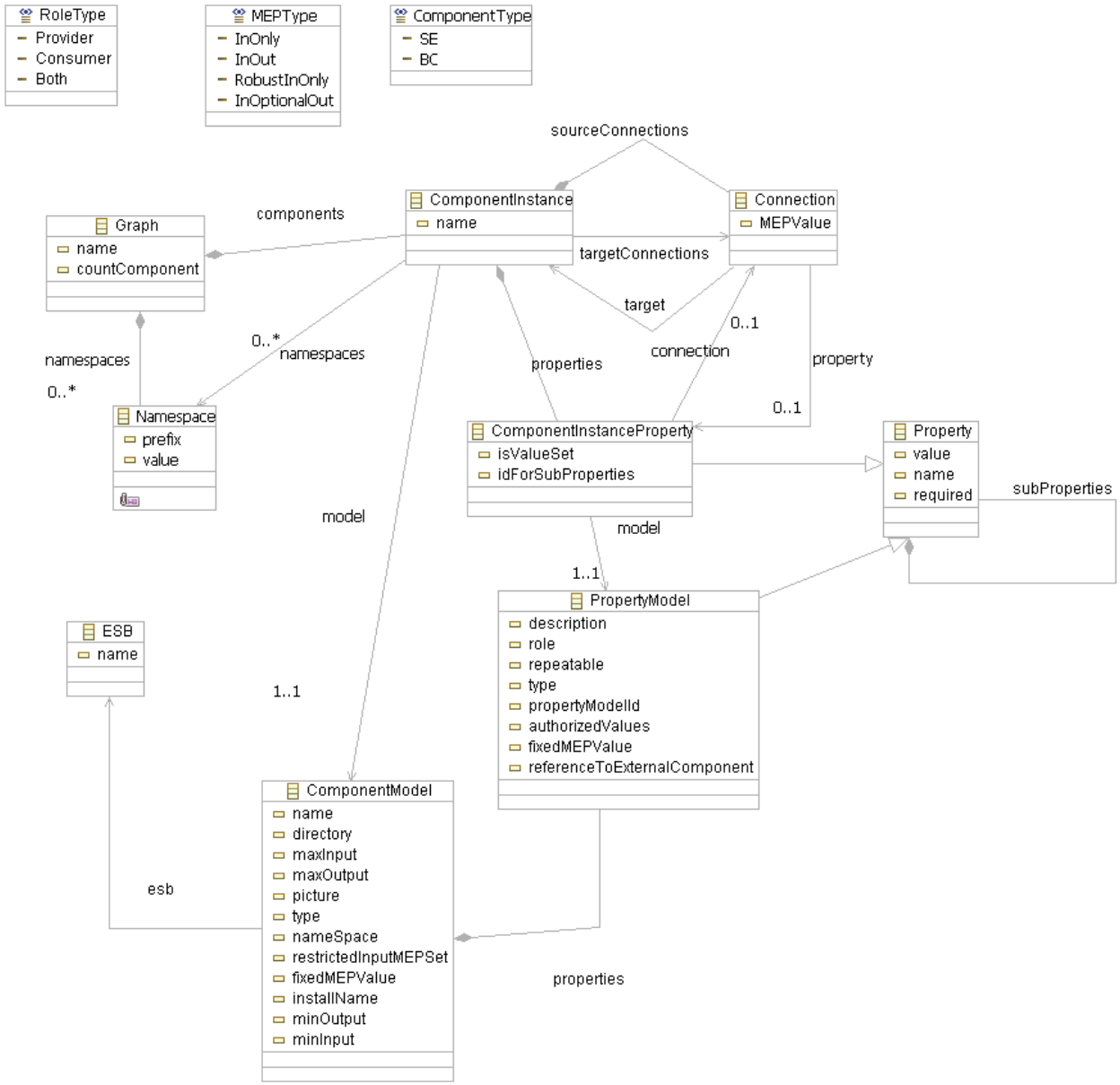

Illustration 1: modèle de données de l'éditeur

1 Voir section 3 GMF de ConceptionGlobale.odt pour plus de précisions

2.2.Présentation, GEF

Les modèles graphiques ci-dessous correspondent au contenu du fichier gmfgraph pour le graphe et dans gmftool pour les outils.

2.2.1.Définition du modèle graphique

- **Diagramme**

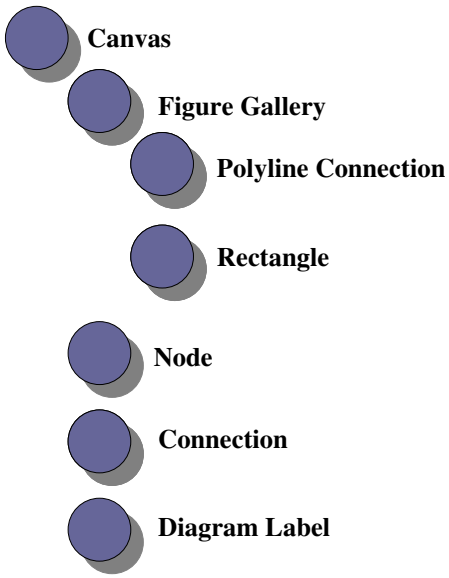

Illustration 2: Modèle graphique

- **Outils**

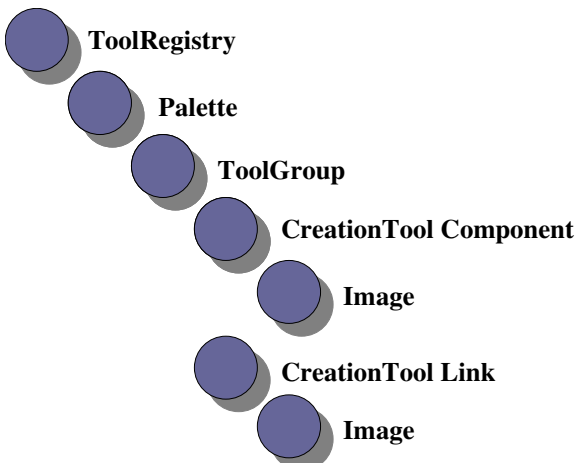

Illustration 3: Modèle des outils

2.3.Mapping

Le mapping ci-dessous est celui défini dans fichier gmfmap. Il associe les éléments du modèle aux éléments graphiques. Les contraintes sont également présentées.

2.3.1.Définition du mapping

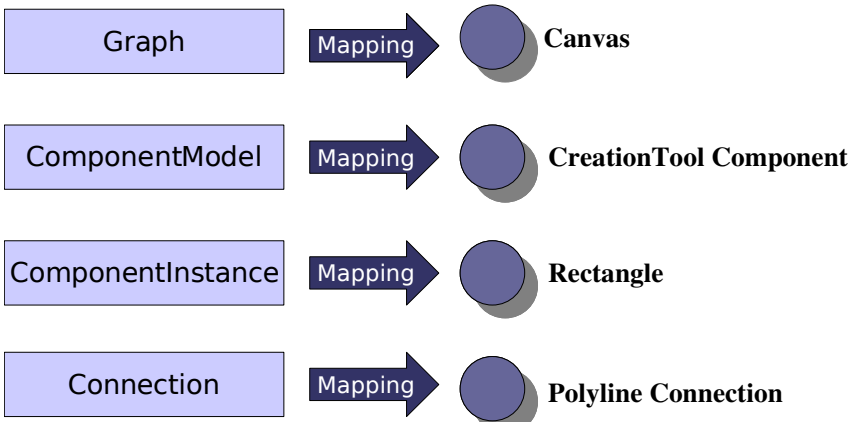

Illustration 4: Mapping entre la définition graphique et le modèle

2.3.2.Définition des contraintes

Les restrictions liée à la conception des graphes de flux sont exprimées grâce à des contraintes OCL. Ces contraintes ne peuvent être violées lors de la manipulation des graphes. Si une contrainte est violée, l'utilisateur est prévenu par un message d'erreur et le diagramme revient dans l'état précédent la modification. Les contraintes sont de 2 types. Ce sont soient des « Link Constraint », soit des « Audit Rule ». Une Link Constraint s'applique aux connections et une audit rule au model de données.

3.Description détaillée des modules

3.1.Modèle de données

➢ **Description**

Le code correspondant à ce module est celui du plugin CimeroEditor généré par GMF. Ce plugin possède les fabriques des différents objets du modèle, il permet de d'instancier et de manipuler ces objets. Le diagramme de classe correspond au modèle de données EMF présenté dans la section précédente. Chaque classe possède une interface et une implémentation. Les factories permettent de créer et d'accéder aux objets du modèle.

➢ **Modifications apportées**

- **Activator**

La classe CimeroEditorPlugin est la première classe appelée lors du démarrage de l'application.

Elle permet d'initialiser la liste des ESB supportés, de créer le répertoire de configuration personnelle de l'application (.cimero2) si celui-ci n'existe pas et d'y copier les fichiers XML représentant les modèles de composants et les différents icônes des composants ajoutés par l'utilisateur. Elle appelle ensuite le parser.

- **Parser**

Les modèles des composants sont stockées sous forme de fichiers XML. Chaque fichier décrit un modèle qui doit être instancié en tant qu'objet de type ComponentModel.

Lors du lancement de Cimero, le parser XML parse les fichiers contenant la « représentation XML » de chacun des composants. Il instancie le modèle du composant générique avec les données spécifiques au composant du fichier parsé : Ceci crée une « représentation Java » du composant.

La classe Parser permet de réaliser cette opération de la manière suivante :

- Parcours du répertoire contenant les fichiers XML
- Vérification de la validité du fichier grâce à un XML Schéma[1]
- Création d'une instance de modèle de composant pour chaque fichier grâce à JDOM

3.2.Présentation

Le framework GMF organise ce module en 2 plugins distincts : CimeroEditor.diagram et CimeroEditor.edit.

1 Voir annexes

3.2.1.CimeroEditor.diagram

- **Description**

Ce plugin permet de gérer principalement la représentation graphique du diagramme :

- La représentation graphique des instances du modèle de données
- Les outils de la palette

Il dispose également de listeners sur les éléments du diagramme permettant de faire remonter et de traiter les interactions de l'utilisateur avec le diagramme.

Il est également en charge de l'initialisation des nouveaux projet : Création du fichier représentant le diagramme et de celui représentant le modèle. Ces 2 fichiers sont étroitement liés.

- **Modifications apportées**

  - **Palette**

La palette est créée dynamiquement en fonction des différents modèles de composants instanciés et de l'ESB associé au diagramme. Pour chacun d'eux, un outil représenté par un icône spécifique est disponible dans la palette qui permet de créer des instances[1] de ce modèle dans le diagramme.

  - **Elements du diagramme**

Les instances des composants sont représentées par l'image associée au modèle de ce composant, présente dans la palette sous forme d'icône.

Les connexions entre les composants ont un aspect différent en fonction de la valeur du MEP du composant source. La modification de cette valeur entraîne une mise à jour de l'aspect de la connexion.

Une contrainte sur les connexions permet de limiter le nombre de connexions entrantes et sortantes en fonction du nombre de connexions autorisées pour chaque composant. Cette contrainte est écrite en OCL et le code correspondant est généré par GMF.

  - **Menu contextuel**

Les classes suivantes ont été crées afin de pouvoir faire appel aux classes de génération du plugin Générateur de package depuis le menu contextuel d'un graphe.

- CimeroEditorGenerateJBIPackageAction pour la génération d'un package JBI correspondant au graphe
- CimeroEditorGenerateAntTaskAction pour la génération de l'aborescence des répertoires et de la tâche Ant pour le graphe

1 Objet ComponentInstance du modèle de données

- CimeroEditorAddNamespaceAction pour l'ajout d'un namespace au graphe.
- CimeroEditorValidateAction pour la validation du graphe.

3.2.2.CimeroEditor.edit

- **Description**

Ce plugin est utilisé pour la génération des éléments de la properties view : les PropertyDescriptors.

- **Modifications apportées**

Les propriétés d'un composants sont représentées sous la forme d'une classe dans le modèle de données. Elles ne sont donc pas accessibles directement depuis une instance d'un composant. La properties view générées ne permet pas d'afficher d'autres éléments que ceux directement liés à l'objet sélectionné.

Une nouvelle properties view dédiée aux instances de composants est mise en place. Elle permet de présenter les attributs de l'instance du composant ainsi que de ceux de ses propriétés.Les propriétés sont représentées comme des arbres avec sous chaque propriété complexe l'ensemble de ses fils.

De plus, un icône présent devant le champ de saisie est utilisé pour apporté plus d'informations à l'utilisateur : il peut ainsi visualiser le type attendu, si la propriété est requise ou non et si la valeur de la propriété est fixée. Des listes permettant de proposer des choix restreint de valeurs pour certaines propriétés sont également mises en place.

Chaque valeur entrée est soumise à une validation directe et n'est conservée que si la validation n'a pas soulevé d'erreurs. Cette validation est effectuée grâce aux Audit Rules présentées dans la partie Validateur[1].

3.3.Générateur de modèle

3.3.1.Description technique

Le générateur de modèle contient 2 classes permettant de gérer les modèles des composants disponibles dans l'éditeur : un générateur de fichier pour écrire les fichier XML correspondant aux modèles des composants créer via l'interface graphique et un parser utilisé pour l'édition.

1 Voir section 3.4

3.3.2.Diagramme de classes

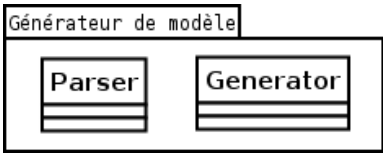

Illustration 5: Diagramme de classe du générateur de modèle

3.3.3.Algorithme de calcul

La génération de modèle est divisé en 2 parties :

- Le génération de fichiers XML pour la représentation de nouveaux composants grâce à JET[1].
- L'édition de fichiers XML représentant des composants de CIMERO.

• Génération

Le générateur de fichier XML crée un fichier XML en fonction des données fournit par l'utilisateur. Ce fichier représente un composant, il doit être conforme au schéma XML des composants. Si les données fournies sont incomplètes ou de type incompatibles avec celui attendu, la génération est stoppée.

• Edition

L'édition des fichiers XML représentant les composants permet de modifier les valeurs des attributs ou d'en ajouter de nouveau. Les étapes nécessaires à l'édition sont les suivantes :

1. Le fichier est d'abord parsé afin de récupérer une représentation Java du composant
2. Les modifications apportées par l'utilisateur sont reportées sur cette représentation
3. Le fichier XML correspondant au composant édité est regénéré.

3.4.Validateur

3.4.1.Description technique

Le validateur est la classe nécessaire à la validation des graphes, afin de permettre le bon fonctionnement des packages générés sur l'ESB . Les contraintes sont suffisantes pour s'assurer que les composants sont corrects mais il peut subsister des erreurs au niveau du graphe même si toutes les règles précisées par les contraintes OCL sont respectées. La validateur permet d'effectuer une dernière validation permettant de s'assurer que le graphe est correct.

1 Voir section 4.1 de ConceptionGlobale.odt

3.4.2.Algorithme de calcul

La validation consiste à s'assurer que le flux créé est valide, c'est-à-dire qu'il n'y a pas d'incohérence comme un composant isolé qui ne peut pas communiquer avec les autres. Les règles suivantes doivent être respectées :

➢ **Link Constraint**

Les contraintes permettant de contrôler la création de connections valides sont les suivantes :

- Un composant ne doit pas avoir plus de connections sortantes, ni entrantes que sa limite
- Un composant ne peut pas être lié à lui-même
- Si le composant est un BC et qu'il possède déjà des connections entrantes, alors il ne peut pas posséder de connections sortantes et inversement

Ces contraintes sont exprimées sous la forme de 2 contraintes OCL. Une contrainte pour la source des connections et une pour la cible.

- *Source End Constraint*

```
(self.sourceConnections->size() < self.maxOutput) and (self <> oppositeEnd) and
(self.type=ComponentType::BC implies self.targetConnections->size()=0)
```

- *Target End Constraint*

```
(self.targetConnections->size() < self.maxInput) and (self.type=ComponentType::BC
implies self.sourceConnections->size()=0)
```

➢ **Audit Rules**

Les contraintes permettant de s'assurer que les éléments du modèle de données sont corrects sont les suivantes :

- Un composant doit avoir au minimum minOutput connections sortantes

```
Constraint : self.sourceConnections->size() >= self.model.minOutput
Domain Element Target : ComponentInstanceProperty
```

- Un composant doit avoir au minimum minInput connections entrantes

```
Constraint : self.targetConnections->size() >= self.model.minInput
Domain Element Target : ComponentInstanceProperty
```

- Un composant doit avoir au moins une connection entrante ou sortante

```
Constraint : self.targetConnections->size() > 0 or self.sourceConnections->size()
> 0
Domain Element Target : ComponentInstanceProperty
```

- Toutes les propriétés « requises » doivent avoir une valeur associée

```
Constraint : self.required implies self.isValueSet
Domain Element Target : ComponentInstanceProperty
```

- Un diagramme doit posséder au moins un BC

```
Constraint : self.components->select(c | c.type = ComponentType::BC)->size()>=1
Domain Element Target : Graph
```

- La valeur d'une propriété de type URL doit comencer par « http:// »

```
Constraint : (self.isValueSet and self.model.type='URL') implies
(self.value.oclIsTypeOf(String) and self.value.size()>7 and
self.value.substring(1,7)='http://')
Domain Element Target : ComponentInstanceProperty
```

- La valeur d'une propriété de type Booléen ne peut avoir que 'true' et 'false' comme valeur

```
Constraint : (self.isValueSet and self.model.type='Boolean') implies
(self.value.oclIsTypeOf(String) and (self.value='true' or self.value='false'))
Domain Element Target : ComponentInstanceProperty
```

3.5.Générateur de package

3.5.1.Description technique

Le générateur de package comprend l'ensemble des classes utilisées pour la génération et packages JBI, de tâches Ant ai,nsi que pour la migration des graphe de la version précédente vers la nouvelle version.

3.5.2.Algorithme de calcul

La génération de package est divisée en 3 parties :

- La génération d'une tâche Ant
- La génération d'un package JBI
- L'importation d'un package JBI

• Génération d'une tâche Ant

La génération d'une tâche Ant consiste à créer l'arborescence des répertoires correspondants aux SUs et aux SAs et un fichier XML permettant de construire ces différents SUs et ou les SAs, ainsi que de déployer le ou les SAs générés. Les arborescences des différents SU sont générés.

1. L'arborescence des SUs est générée

2. Les descripteurs des SUs sont générés
3. Si d'autres fichiers que le descripteur sont associés au SU, ceux-ci sont ajoutés
4. Chaque SU est compressé dans une archive Zip
5. L'arborescence du SA unique et des SAs disctinct est généré
6. Les descripteurs des SA sont générés
7. Le fichier XML correspondant à la tâche Ant est généré

Si une erreur est détectée dans une des étapes, le processus de génération de la tâche Ant est stoppée, les arborescences qui auront déjà été générées sont supprimées pour conserver un environnement propre.

- **Génération d'un package JBI**

La génération d'un package JBI consiste à générer tout les fichiers nécessaires et à les compresser dans une archive Zip. Les étapes sont les suivantes :

1. Le graphe est soumis au validateur. La validation doit être positive
2. L'arborescence est générées ainsi que la tâche Ant[1]
3. La génération du package JBI est lancée via la cible Ant correspondant à la génération d'un SA unique

A chaque étapes, si une erreur survient le processus de génération est stoppé, tout ce qui aura déjà été généré est supprimé. Les arborescences sont conservées si il n'y a pas eu d'erreur lors de la procédure de génération.

3.6.Coordinateur

3.6.1.Description technique

Le contrôleur est la classe utilisée pour ordonnancer les communications entre les modules.

3.6.2.Algorithme de calcul

Les algorithmes de calcul sont basés sur les diagrammes de séquence[2]. Le contrôleur met en place la suite des appels au différents composants des modules pour réaliser chacune des tâches utilisateur.

1 Voir paragraphe précédent : Génération d'une tâche Ant

2 Voir section 4 de ConceptionGlobale.odt

3.7.Importateur

3.7.1.Description technique

L'importateur est une classe utilisée pour la migration des projets fait grâce à Cimero vers Cimero 2.

3.7.2.Algorithme de calcul

La migration consiste à générer le modèle de données correspondant au graphe de la version précédente puis à l'afficher. Les étapes suivies sont les suivantes :

1. Création d'un fichier xml représentant le graphe de la version précédente car les fichiers .cimero de la version précédente stocke le graphe sous d'objet Java sérialisés
2. Récupération des informations concernant les composants formant le graphe. Instanciation des différents composants du flux. Les composants qui ne sont pas parmi ceux disponibles dans la nouvelle version de CIMERO sont remplacés par l'instanciation d'un composant « default » sans propriétés.
3. Génération des fichiers .cimero2_diagram et .cimero2 correspondant
4. Affichage du graphe

3.8.ServiceMix

3.8.1.Description technique

ServiceMix est représenté par le plugin CIMERO ServiceMix et il est séparé du plugin CIMERO Editor. En effet, ces deux plugins doivent pouvoir être utilisés séparément.

Ce plugin fournit à l'utilisateur la possibilité d'ajouter un serveur ServiceMix sous Eclipse, de le démarrer ou de l'arrêter et de fournir des méthodes permettant de déployer (et de retirer) facilement des packages JBI.

Cette partie du plugin est déjà implémentée et ne devrait pas être modifiée. Nous ne détaillerons pas son fonctionnement, pour plus d'informations, se référer aux documents produits lors du développement de CIMERO Version 1.

Annexes

A)Schéma XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!--  Schema de definition d'un composant  -->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  targetNamespace="http://java.sun.com/xml/ns/component"
  xmlns:this="http://java.sun.com/xml/ns/component">

  <xs:annotation>
    <xs:documentation>
      Definition schema for a generic component
    </xs:documentation>
  </xs:annotation>

  <!-- Component -->
  <xs:element name="component">
    <xs:complexType>
      <xs:sequence>
        <!-- Component name of the component which has to be installed to allow
deployement -->
        <xs:element ref="this:install-component-name" />
        <!-- Component name in the editor -->
        <xs:element ref="this:component-name" />
        <!-- Component picture -->
        <xs:element ref="this:picture" />
        <!-- Targeted ESB -->
        <xs:element ref="this:esb" />
        <!-- Component Namespace -->
        <xs:element ref="this:component-namespace" />
        <!-- BC/SE -->
        <xs:element ref="this:component-type" />
        <!-- The component model archive name -->
        <!-- For example : -->
        <!-- petals-engine-clock-1.1-M2.zip -->
        <!-- servicemix-quartz-3.1-incubating-installer.zip -->
        <xs:element ref="this:install-component-directory" />
        <!-- Maximum input connections -->
        <xs:element ref="this:max-input" />
        <!-- Maximum OutPut connections -->
        <xs:element ref="this:max-output" />
        <!-- Maximum input connections -->
        <xs:element ref="this:min-input" />
        <!-- Maximum OutPut connections -->
        <xs:element ref="this:min-output" />
        <!-- MEP for the component, if there is a fixed one -->
        <xs:element minOccurs="0" maxOccurs="1"
          ref="this:MEP" />
```

```
      <!-- Input set of MEPs authorized  for the component,
       if there is no fixed MEP and the user can choose the MEP value -->
      <xs:element minOccurs="0" maxOccurs="1"
        ref="this:restrictedInputMEPSet" />
      <!-- Properties (between 0 and infinity) of the component -->
      <xs:element minOccurs="0" maxOccurs="unbounded"
        ref="this:property" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

<!-- Type definition -->
<!-- Specific use : Provider, Consumer, or both -->
<xs:simpleType name="usability">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Both" />
    <xs:enumeration value="Provider" />
    <xs:enumeration value="Consumer" />
  </xs:restriction>
</xs:simpleType>

<!-- MEP : InOnly, InOut, RobustInOut, InOptionalOut -->
<xs:simpleType name="MEPType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="InOnly" />
    <xs:enumeration value="InOut" />
    <xs:enumeration value="RobustInOnly" />
    <xs:enumeration value="InOptionalOut" />
  </xs:restriction>
</xs:simpleType>

<!-- Basics types -->
<xs:simpleType name="types">
  <xs:restriction base="xs:token">
  <!-- The types of the properties -->
    <xs:enumeration value="String" />
    <xs:enumeration value="Boolean" />
    <xs:enumeration value="Int" />
    <xs:enumeration value="QName" />
    <xs:enumeration value="URI" />
    <xs:enumeration value="URL" />
    <xs:enumeration value="XOR" />
    <xs:enumeration value="Unknown" />
    <xs:enumeration value="Unsettable" />
    <xs:enumeration value="Class" />
    <xs:enumeration value="MEP" />
    <xs:enumeration value="Role" />
    <xs:enumeration value="File" />
    <xs:enumeration value="Service" />
    <xs:enumeration value="EndPoint" />
    <xs:enumeration value="Interface" />
  </xs:restriction>
</xs:simpleType>

<!-- Component "type" : BC or SE -->
```

```
<xs:simpleType name="component-type-choice">
  <xs:restriction base="xs:string">
    <xs:enumeration value="BC" />
    <xs:enumeration value="SE" />
  </xs:restriction>
</xs:simpleType>

<!-- Definition of elements types for the component -->
<xs:element name="esb" type="xs:string" />
<xs:element name="component-type" type="this:component-type-choice" />
<xs:element name="install-component-directory" type="xs:anyURI" />
 <xs:element name="install-component-name" type="xs:string" />
<xs:element name="component-name" type="xs:string" />
<xs:element name="component-namespace" type="xs:string" />
<xs:element name="picture" type="xs:string" />
<xs:element name="max-input" type="xs:int" />
<xs:element name="max-output" type="xs:int" />
<xs:element name="min-input" type="xs:int" />
<xs:element name="min-output" type="xs:int" />
<xs:element name="MEP" type="this:MEPType" />

<!-- Definition of elements types for the property -->
<xs:element name="attribute-name" type="xs:string" />
<xs:element name="attribute-description" type="xs:string" />
<xs:element name="default-value" type="xs:string" />
<xs:element name="role" type="this:usability" default="Both" />
<xs:element name="required-attribute" type="xs:boolean" />
<!-- Warning, a repeatable property should not have brothers on the model -->
<xs:element name="repeatable-attribute" type="xs:boolean" />
<xs:element name="referenceToExternalComponent" type="xs:boolean" />
<xs:element name="attribute-type" type="this:types" />

<!-- Definition of elements types for the values -->
<xs:element name="value" type="xs:string" />
<xs:element name="value-pattern" type="xs:string" />

<!-- MEPSet allowed for input -->
<xs:element name="restrictedInputMEPSet">
  <xs:complexType>
    <xs:sequence>
      <!-- MEP set for the component -->
      <xs:element minOccurs="0" maxOccurs="4" ref="this:MEP" />
    </xs:sequence>
  </xs:complexType>
</xs:element>


<!-- Definition of a property -->
<xs:element name="property">
  <xs:complexType>
    <xs:sequence>
      <!-- Property name -->
      <xs:element ref="this:attribute-name" />
      <!-- Property description -->
      <xs:element ref="this:attribute-description" />
```

```
        <!-- Use case of the property : provider or consumer or Both -->
        <xs:element ref="this:role" />
        <!-- If the property is required -->
        <xs:element ref="this:required-attribute" />
        <!-- If the property is repeatable -->
        <xs:element minOccurs="0" maxOccurs="1"
          ref="this:repeatable-attribute" />
        <!-- Value description of the property : Values set -->
        <xs:element minOccurs="0" maxOccurs="1"
          ref="this:value-description" />
        <!-- Default value of the property -->
        <xs:element minOccurs="0" maxOccurs="1"
          ref="this:default-value" />
        <!-- Property type -->
        <xs:element ref="this:attribute-type" />
        <!-- MEP type if the property needs one -->
        <xs:element minOccurs="0" maxOccurs="1" ref="this:MEP"/>
        <!-- if the property is a reference to an external service -->
        <xs:element minOccurs="0" maxOccurs="1"
ref="this:referenceToExternalComponent"/>
        <!-- SubProperties -->
        <xs:element minOccurs="0" maxOccurs="unbounded"
          ref="this:property" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <!-- valueDescription : Precise la forme des valeurs possible d'une propriete
-->
  <xs:element name="value-description">
    <xs:complexType>
      <xs:choice>
        <!-- Liste de valeurs -->
        <xs:element ref="this:values-list" />
        <!-- Pattern -->
        <xs:element ref="this:value-pattern" />
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <!-- values-list -->
  <xs:element name="values-list">
    <xs:complexType>
      <xs:sequence>
        <!-- Values -->
        <xs:element minOccurs="1" maxOccurs="unbounded"
          ref="this:value" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```