Tutorial : Conception de syntaxes concrètes graphiques

(TP3 + TP4)

Au cours de ce tutorial on va examiner un méta modèle et on va générer une syntaxe concrète graphique en utilisant l'utile « Diagraph ».

Prérequis :

- Eclipse Indigo Modeling SR2
- Le plugin Diagraph
- JDK 1.6 min
- Le plugin SVN

Etape 1 : Compréhension du modèle sémantique fourni (méta modèle)

1. Pour commencer, lancer Eclipse et sélectionner votre workspace.
2. Changer la perspective en SVN (Window -> Open Perspective -> Other -> SVN Repository Exploring).
3. Créer un nouveau repository location (File -> new Ropository Location -> URL : http://aigle-2012.googlecode.com/svn/trunk/aigle.idm.diagraph.tp3/ -> finish).
4. Enregistrer le projet en local (click droit sur le projet -> Check Out)
5. Passer en perspective Java (Window -> Open Perspective -> Java)
6. Déconnecter vous de SVN (click droit sur le projet -> team -> Disconnect)
7. Analyser les fichiers tp3.ecore qui représente la syntaxe abstraite de notre modèle et tp3.ecorediag qui représente la syntaxe concrète **(on est au niveau M2)**. Essayer de comprendre ce méta model

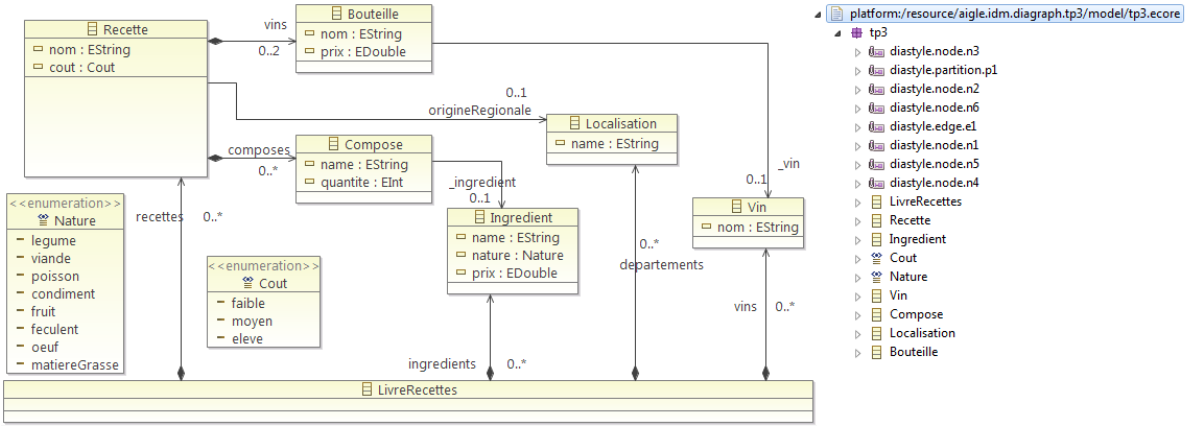

Figure 1 : tp3.ecorediag (gauche) tp3.ecore (droit)

Etape 2 : Génération d'un modèle arborescent, tester la consistance du méta modèle

1. Générer tp3.genmodel (click droit sur le projet -> new -> other -> EMF Generator Model -> ecore -> load tp3.ecore -> finish)
2. Ajouter le préfix a tp3.genmodel (Window -> Show View -> Properties pour avoir la vue des propriétés, puis sur tp3.genmodel sélectionner Tp3 -> dans la vue des propriétés sélectionner Base Package et ajouter « aigle.idm.diagraph »)
3. Maintenant qu'on a le genmodel, on peut tester la consistance de notre modèle en utilisant l'éditeur arborescent offert par EMF.
4. EMF ou Eclipse Modeling Framefork se base sur deux méta modèles : Ecore et Genmodel. Ecore contient les informations des classes définies alors que Genmodel contient des informations pour la génération de code, comme par exemple le path, des informations sur les fichiers…
5. Générer l'éditeur (dans tp3.genmodel click droit sur Tp3 -> Generate All)
6. Du code a été générer dans src ainsi que trois nouveaux projets : edit, editor et tests
7. Lancer l'éditeur (click droit sur editor -> Run As -> Eclipse Aplication)
8. Nouvelle instance d'Eclipse est ouverte. Créer un projet général et click droit sur le projet new -> Other -> Tp3 Model -> --- -> Livre Recettes -> finish
9. Tester la consistance : Créer des Recettes, des ingrédients…

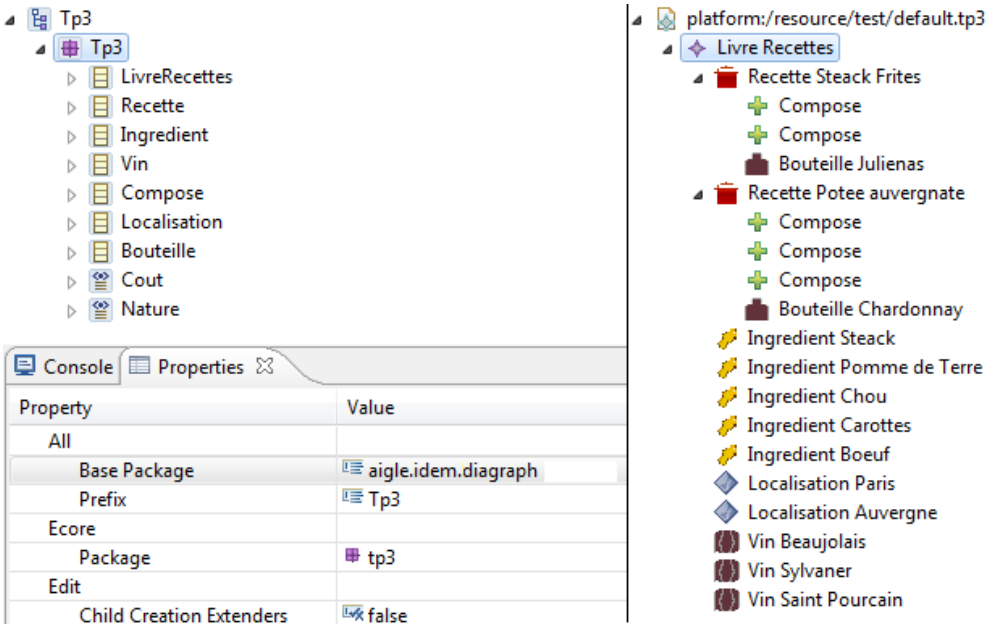

Figure 2 : tp3.genmodel (gauche), l'éditeur arborescent d'EMF (droit)

Etape 3 : Conception de la notation

Maintenant qu'on a testé l'éditeur arborescent, on veut passer à une représentation graphique dotée d'une légende en utilisant Diagraph. C'est beaucoup mieux de lire une représentation graphique que lire une représentation arborescente. Pour réaliser une représentation graphique, il faut d'abord imaginer à quoi ça va ressembler et faire un

premier exemple « papier-crayon ». Ensuite il faut bien commenter le méta model : des couleurs différents pour des classes représentant des choses différents, par exemple le

canvas , les nœuds du canvas, les liens avec une étiquette etc…, des commentaires pour chaque classe du méta modèle…

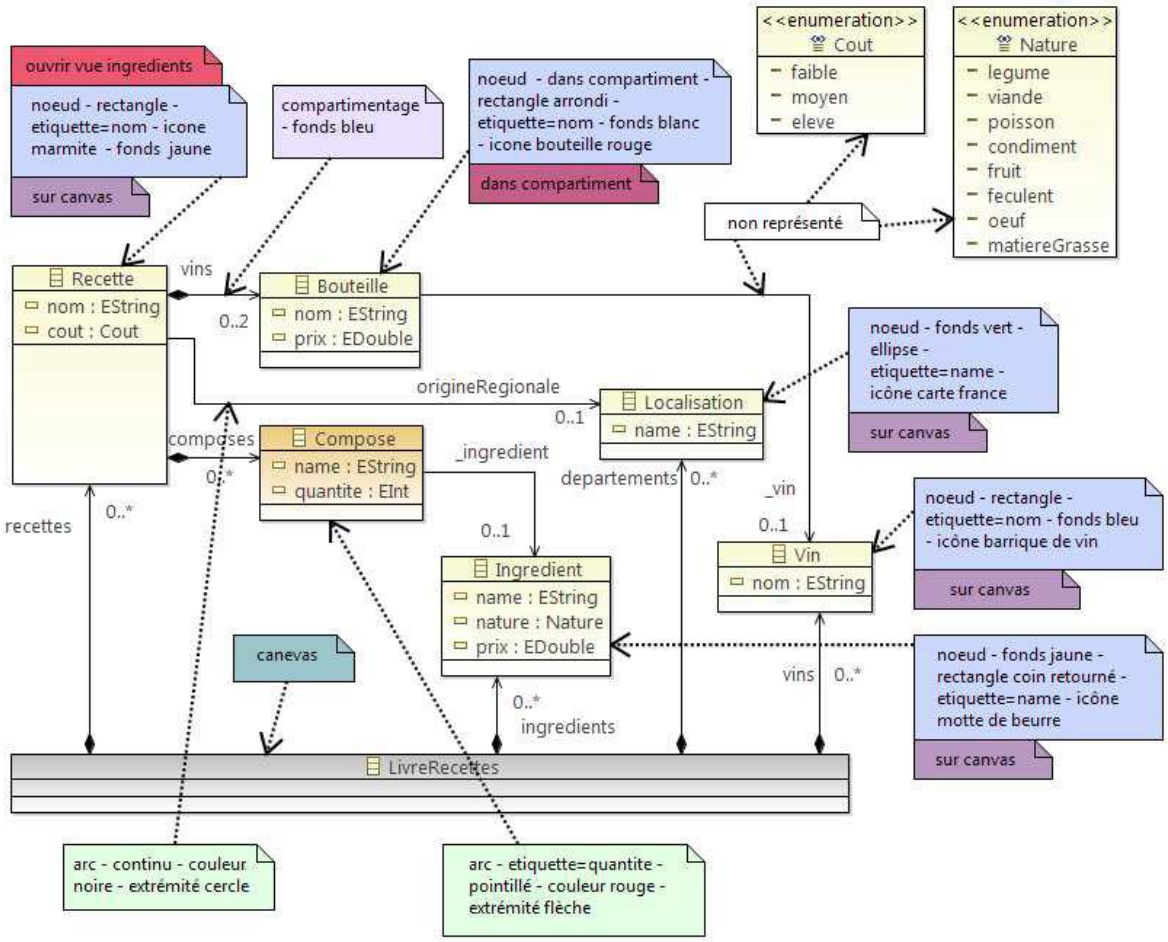

Figure 3 : Commenter le méta modèle

Etape 4 : Implémentation de la notion

Apres avoir fait l'étape 3 qui sert à nous facilite la vie, on peut attaquer à la création de la représentation graphique avec l'utile Diagraph qui est une surcouche de GMF. On va utiliser les annotations pour définir ce qu'on veut représenter. Pour que l'utile Diagraph prenne en compte nos annotations, il faut les nommer soit avec le mot clé « diagraph » pour représenter les entités du modèle, soit avec le mot clé « diastyle » pour définir le style des entités. Il existe quatre types d'annotation diagraphe : pov, node, link et non définie.

1. Pov : Point Of View ou Canvas qui est le containeur, dans notre exemple c'est la classe LivreRecettes qui va être annotée comme pov. (un seul attribut dans cette annotation – pov)

2. Node : Tous les nœuds qu'on veut représenter dans notre modèle. C'est possible d'ajouter plusieurs attributs : label pour l'étiquète du nœud, cref et kref pour imbriqué un nœud dans le nœud curant, lnk pour définir un lien vers d'autre nœud à travers d'un « link », ref pour définir un lien vers d'autre nœud et style pour attache un style prédéfinie au nœud curant.
3. Link : Tous les liens étiquetés qu'on veut représenter. Attributs possible : label, style et une variation de lnk cotée node qui peut être remplacé par cont, lsrc et ltrg coté link
4. Non-définie : sert à définir des propriétés qui vont être partagée par des entités en utilisant l'héritage (à voir la classe ElementEmbarque du tp4)

Il existe trois types de diastyle : node, edg et partition :

1. diastyle.node : sert à définir le style d'un nœud (node)
2. diastyle.edge : sert à définir le style d'un arc (link)
3. diastyle.partition : sert à définir le style de la partie occupée par les crefs ou krefs dans un nœud

La création de la représentation graphique avec l'utile diagraph se fait dans deux phases

1. La première phase consiste à définir le canvas, les nœuds, les liens entre les nœuds. Cette phase est appeler « organisation spatial »
2. La deuxième phase consiste à définir le style des nœuds et des arcs ainsi que changer les icones de la palette.

Phase 1

- Ouvrir le fichier tp3.ecorediag + affiche la vue Outline (Window -> Show View -> Outline). Sur la vue Outline en base on a une représentation arborescente du modèle qui représente notre ecore. C'est sur l'ecore qu'on va créer les annotations !
- Pour créer une annotation, click droit sur l'élément qu'on veut représenter avec l'utile diagraph -> new child -> Eannotation
- Tirer l'annotation d'ecore vers ecorediag (la représentation graphique du modèle).
- Maintenant qu'on a l'annotation sur ecorediag il faut la renommer en diagraph pour qu'elle soit considérée pas l'utile diagraph
- Ajouter des attributs dans l'annotation (node,link,label,ref,kref,cref,…)
- Afficher la vue Diagraph (Window -> Show View -> Other -> Diagraph). La vue Diagraphe contient deux buttons. Layout pour arranger les annotations et Generate Editors pour générer l'éditeur graphique.
- Cliquer d'abord sur Layout, ensuit sur Generate Editors
- Du code a été générer dans src ainsi que quatre nouveaux projets : edit, editor, tests et diagram_default_root

- Tester l'éditeur graphique : Click droit sur diagram_default_root ->Run As -> Eclipse Aplication. Une deuxième instance d'Eclipse est lancée
- Créer nouveau projet, dans le projet : new -> Other -> Tp3_default_root Diagram
  Et vous avez l'éditeur graphique avec une palette à droit (Figure 4)

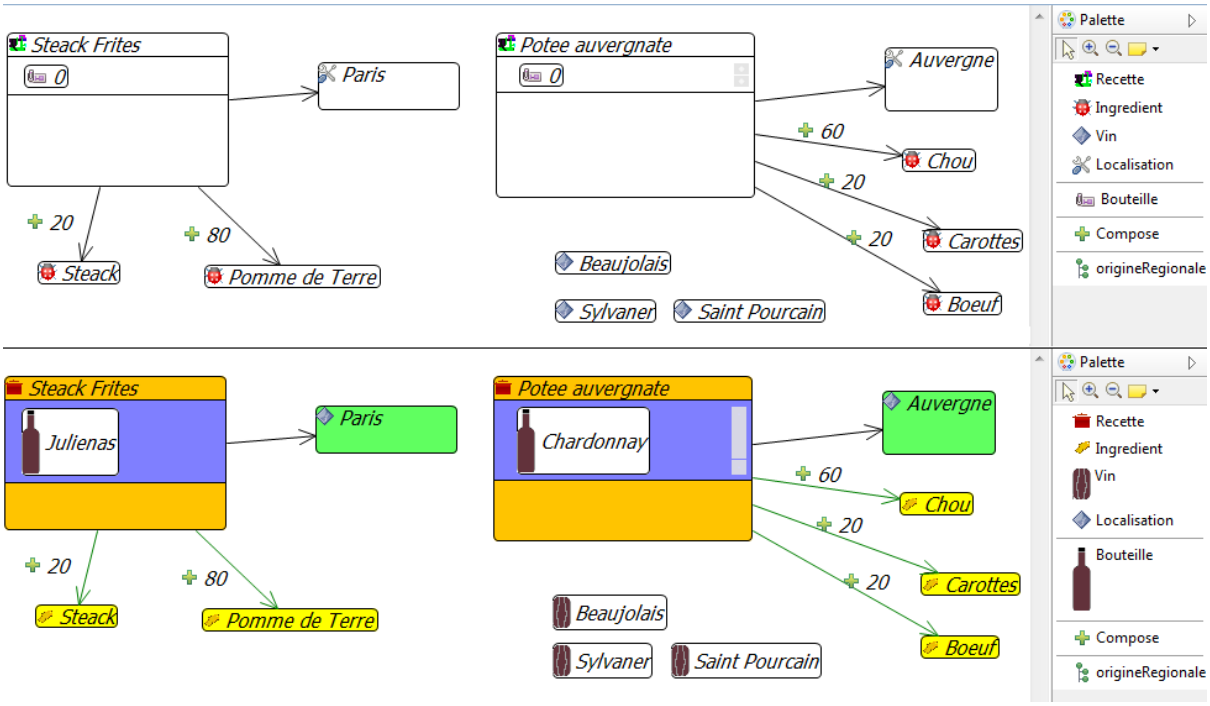

Figure 4 : Editeur Graphique générer pas Diagraph (avec style et icones en bas)

Phase 2

- Créer une annotation de style sur la racine d'ecore (click droit sur tp3 -> new child--> Eannotation). Puis glisser l'annotation vers ecoregiag et changer le nom en diastyle.**type**.name avec type Є {node, partition, edge}. Puis ajouter des attributs
- Pour associer le style a une entité, ajouter l'attribut « style=name » dans l'annotation de l'entité

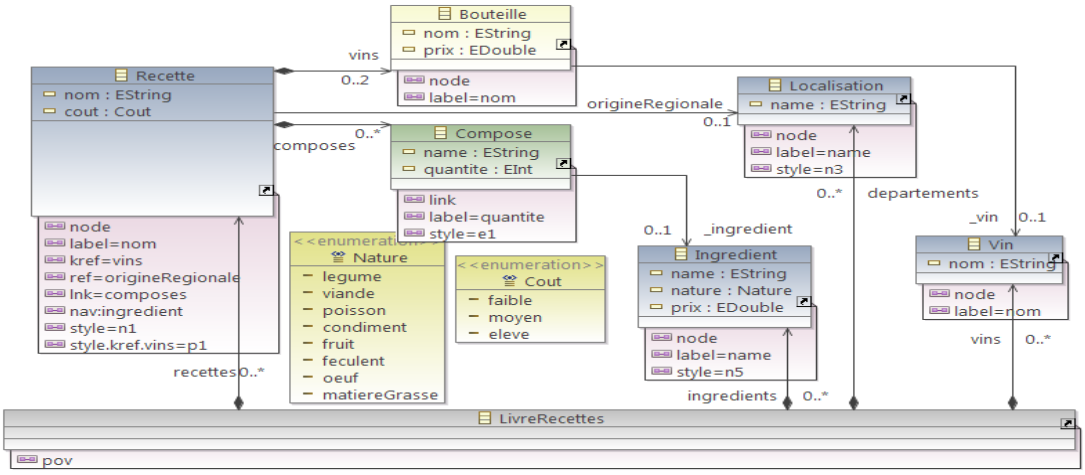

Figure 5 : tp3.ecoregiag

- Pour changer les icones de la palette, il suffit de copier les images que vous désirez dans le fichier icons et icons-backup. Si vous voulez changer l'icône de la classe Bouteille, il faut nommer la nouvelle image « Bouteille.gif » avec une largeur de 16p.
- Au final on a le méta modèle de la Figure 5 qui va nous permettre de créer l'editeur graphique de la figure 4.

Variations

- Ajouter ref=_vin dans l'annotation de la classe Bouteille pour faire un lien entre les vins et les bouteilles
- Toutes les classes possèdent un attribut name, et certains possèdent un attribut prix, alors c'est possible de créer une super classe Named qui va contenir un attribut name et une deuxième super classe ValuedNamed qui va contenir un attribut prix, et va hériter de Named. A voir Figure 6. **En rouge une remarque !**

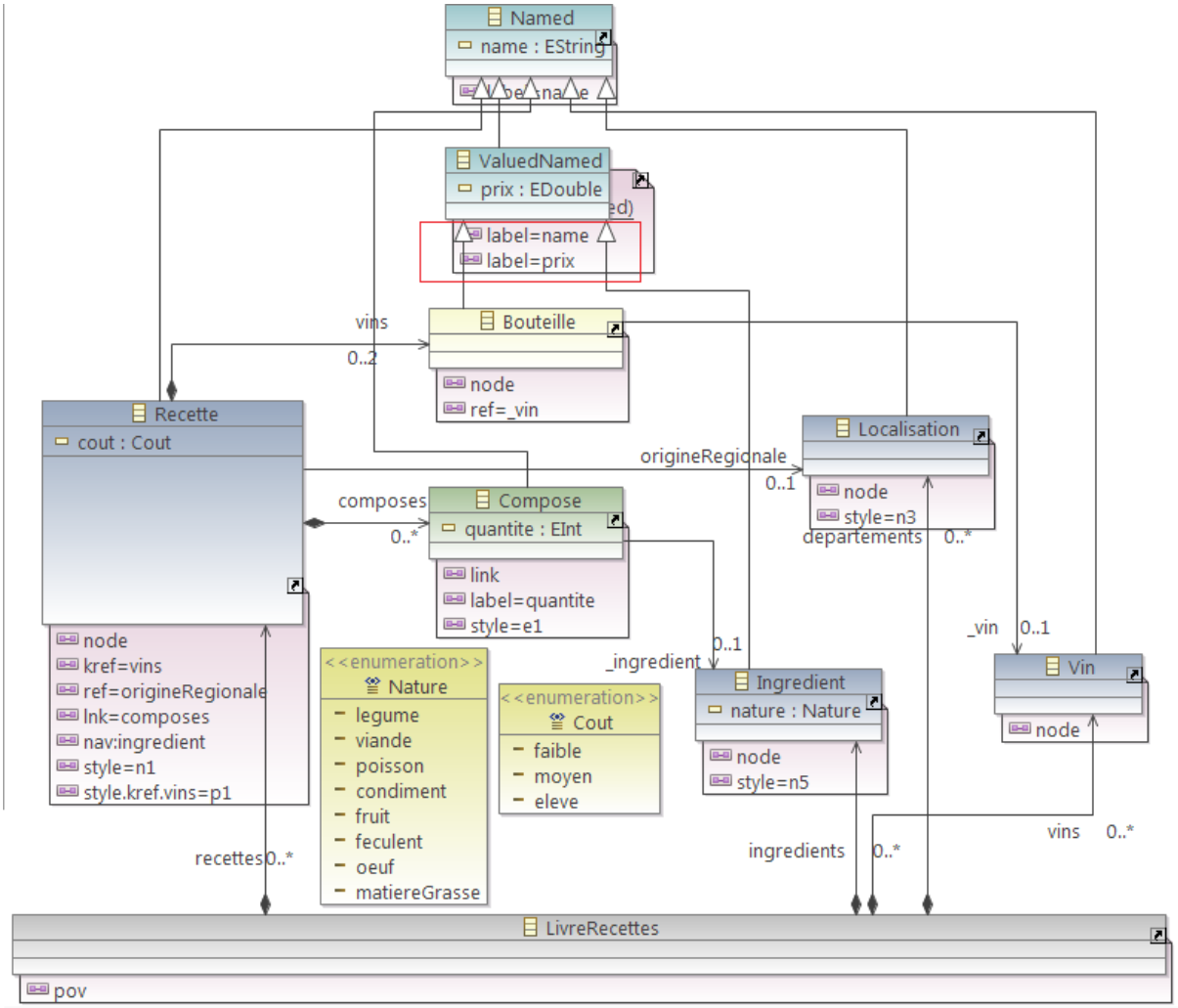

Figure 6 : Une amélioration de la version précédente

Remarques

- Quand on hérite d'un élément on garde bien la propriété label dans le cas quand la classe ne spécifie pas une nouvelle propriété label (Par exemple la classe Vin va

garder le label de Named). Mais si la classe veut rajouter un autre label (Par exemple ValuedNamed ajoute le lable prix), on est obligé de spécifier les labels qu'on veut hériter de la super class ! Dans notre cas on veut bien que ValuedNamed contient deux labels name et prix, et bah on est obligé de réécrire « label=name » qui n'est pas hérité pas default !

- L'attribut Shape n'est pas considéré ! L'annotation de la classe Localisation définie un attribut : style=n3 qui définit la forme du nœud localisation en ellipse

TP 4

Pour réaliser le tp4, il faut faire les 4 étapes décrit en haut. Je vais juste ajouter l'image d'Ecouredіag pour le méta model et une image d'éditeur graphique. Je crois que c'est inutile de vous présenter la première version du tp4 parque je suis déjà à 8 pages, mais si ça vous intéresse, contacter moi je vais vous l'envoyer.

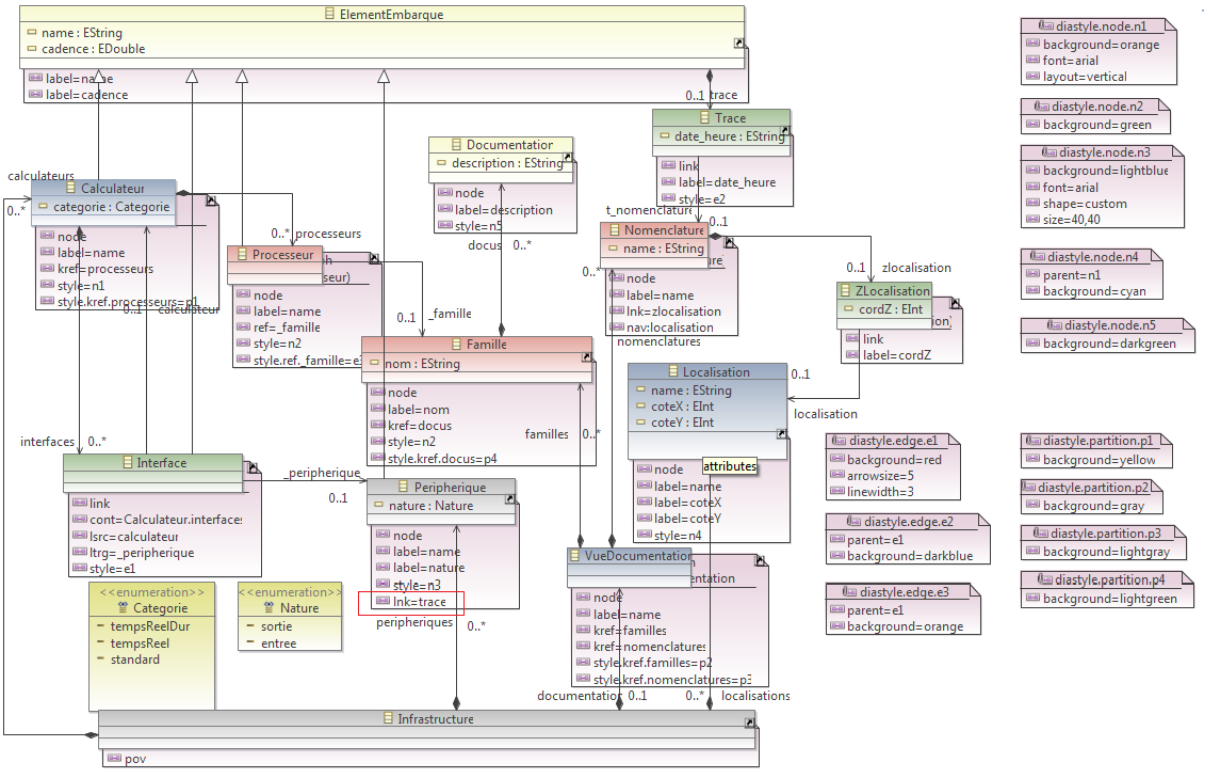

Figure 7 : Meta model de tp 4 version 2 (en rouge une remarque détaillée en bas)

Remarques

- ElementEmbarque est une généralisation qui englobe les attributs name et cadence pour Calculateur, Processeur, Peripherique qui sont des nœuds et Interface qui est un lien. Le système jette une exception « impossible de caster un élément générique en nœud qui est tout à fait normal parc qu'il a trouvé que la classe Trace est un lien vers le nœud Nomenclature. On a la même exception si on essaye d'ajouter lnk=trace dans l'annotation de ElementEmbarque.

- Par contre si on ajoute « lnk=trace » dans l'un des annotations des sous-classes qui sont annotées comme des nœuds, le system accepte ça, et on peut relier Processeur, Peripherique et Calculateur avec Nomenclature, vu qu'il n'y a que l'annotation de Processeur qui contient lnk=trace…
- Solution possible : ajouter des attributs spéciaux dans les annotations des classes génériques qui ne vont concerner que les sous nœuds ou liens. Par exemple lnk dans un élément générique qui ne concerne que les sous éléments nœuds.

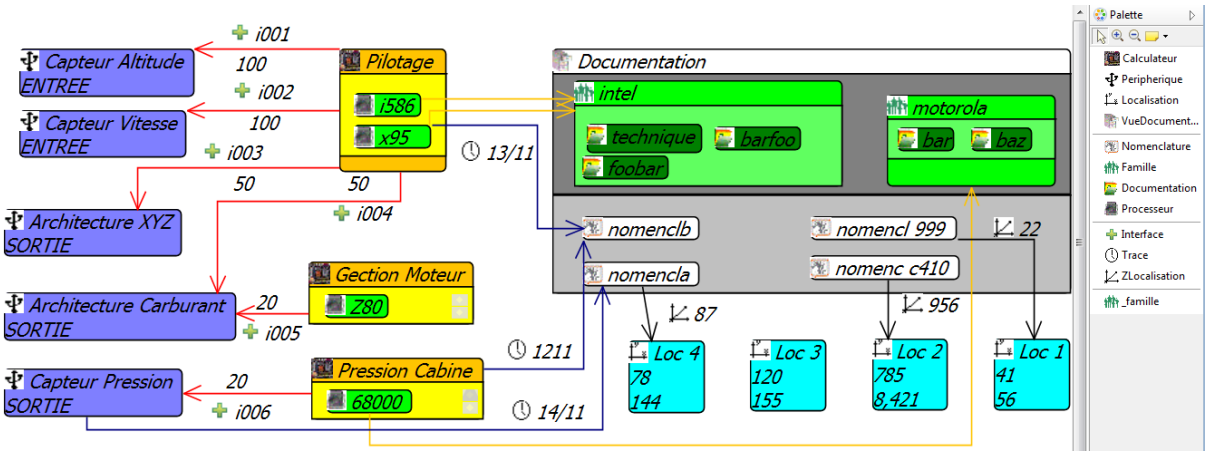

Figure 8 : L'éditeur graphique

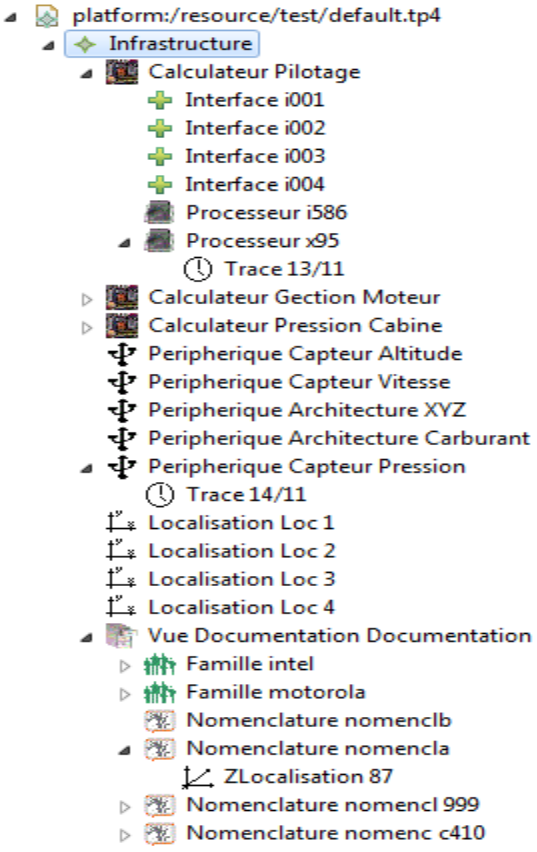

Figure 9 : Le modele en syntaxe abstraite