Définition de syntaxes concrètes graphiques

La version d'Eclipse à utiliser est la suivante :

```
/mnt/n7fs/ens/tp_babin/gls/eclipse-gls/bin/eclipse-gls
```

À l'instar d'une syntaxe concrète textuelle, une syntaxe concrète graphique fournit un moyen de visualiser et/ou éditer plus agréablement et efficacement un modèle. Nous allons utiliser l'outil Eclipse Sirius [1] développé par les sociétés Obeo et Thales, et basé sur les technologies Eclipse Modeling comme EMF et GMF [2]. Il permet de définir une syntaxe graphique pour un langage de modélisation décrit en Ecore et d'engendrer un éditeur graphique intégré à Eclipse.

1 Préparation : métamodèle Ecore et genmodel

Le point de départ est la définition de la syntaxe abstraite du DSML [3]. Elle est définie par un modèle Ecore (le métamodèle) éventuellement complété par des contraintes (exprimées en OCL par exemple). Nous allons considérer le métamodèle de SimplePDL (figure 1).

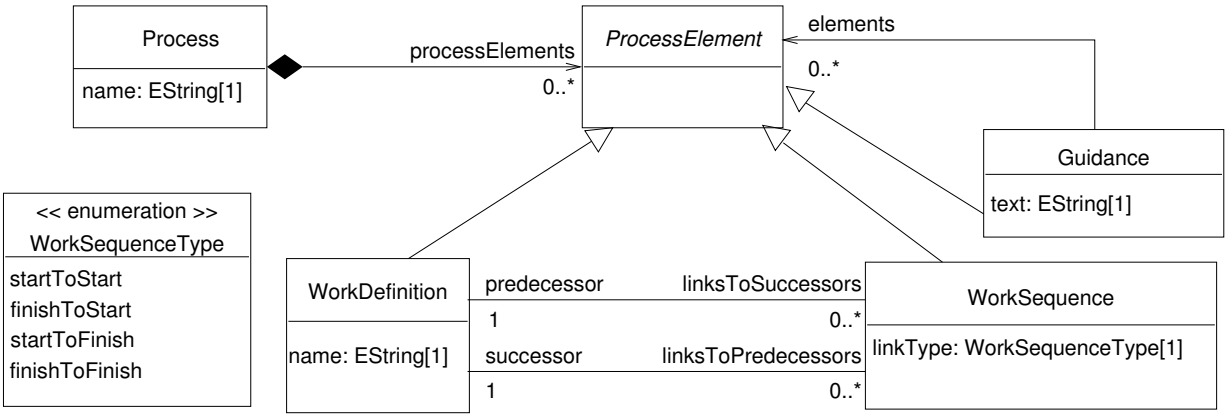

FIGURE 1 – Nouveau métamodèle de SimplePDL

Exercice 1 : Engendrer et déployer l'infrastructure EMF
Si cela n'a pas été fait dans lors des séances précédentes, les étapes à réaliser sont les suivantes :

1. Se placer dans le projet `fr.enseeiht.simplepdl` contenant le méta-modèle de SimplePDL (`SimplePDL.ecore`).

1. `https://www.eclipse.org/sirius/`
2. Eclipse Graphical Modeling Framework : `https://www.eclipse.org/gmf-tooling/`
3. domain-specific modeling language

2. Créer le fichier `SimplePDL.genmodel` (s'il n'est pas présent) et faire *Generate All*.
3. Déployer les greffons `fr.enseeiht.simplepdl`, `fr.enseeiht.simplepdl.edit`, `fr.enseeiht.simplepdl.editor`.

Tout le reste se fera maintenant en utilisant les greffons déployés.

2 Définir une syntaxe graphique avec Sirius

Voyons comment définir une syntaxe concrète graphique souhaitée pour un métamodèle. Par exemple, on peut souhaiter la syntaxe proposée à la figure 2. Sirius permet de représenter des modèles graphiquement sous la forme d'un graphe composé de nœuds et d'arcs. Il est possible de choisir la forme d'un nœud (rectangle, ellipse, etc.) et la forme d'un arc (décoration à l'extrémité, tracé du trait, etc.).

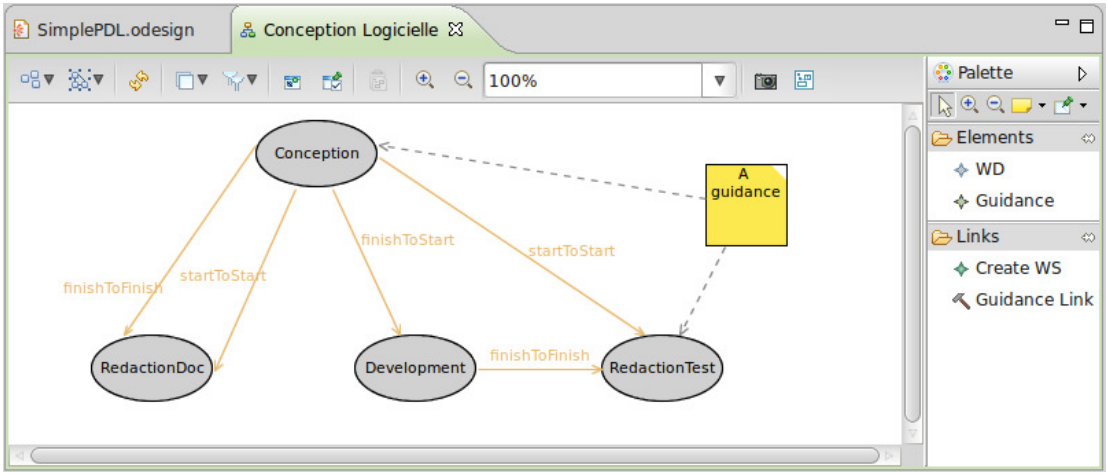

FIGURE 2 – Exemple d'éditeur possible pour SimplePDL

Définir une syntaxe graphique c'est donc :

1. définir comment représenter les éléments d'un modèle. Dans le cas de SimplePDL, on décide de représenter :
   — une activité (`WorkDefinition`) par une ellipse de fond gris clair,
   — une dépendance (`WorkSequence`) par un arc orienté entre deux activités.
   — une aide (`Guidance`) par une note avec un fond jaune,
   — l'aide sera attachée aux éléments qu'elle décrit par une flèche en traits interrompus à l'instar des annotations en UML.
2. définir la correspondance entre les éléments du modèle et les éléments graphiques. Cette correspondance doit être bidirectionnelle pour permettre de visualiser un modèle (il faut savoir à quels élément graphiques correspondent les éléments du métamodèle) et l'éditer (il faut connaître l'impact de l'ajout, la modification ou la suppression d'un élément graphique sur le modèle correspondant.

3. si on veut définir un éditeur, il faut avoir une palette qui permet de créer les différents éléments graphiques de la syntaxe graphique.

Sirius propose un unique métamodèle pour ces trois types d'éléments. Lorsque le modèle correspondant est défini, il est possible de générer une vue graphique pour tout modèle conforme au métamodèle pour lequel on a défini l'éditeur.

Un point fort de Sirius est qu'il permet de modifier l'éditeur graphique tout en visualisant le résultat sur un modèle.

Dans la suite, nous allons voir comment construire un éditeur (donc définir une syntaxe concrète graphique) pour SimplePDL. À travers quelques extensions précises, nous entrapercevrons aussi les possibilités offertes par Sirius.

Exercice 2 : Création d'un projet contenant un modèle de test

Commençons par créer un projet avec un modèle conforme à SimplePDL qui servira à tester la syntaxe graphique.

2.1 Se placer en perspective Sirius.

2.2 Créer un *Sirius / Modeling Project* nommé `fr.enseeiht.simplePDL.sample`.

2.3 Importer le fichier `developpement.simplepdl` dans ce projet.

2.4 Vérifier que le modèle est bien chargé en le dépliant.

Exercice 3 : Mise en place du modèle de description de la syntaxe graphique

Ici, nous allons initier le modèle de description de la syntaxe graphique souhaitée et l'utiliser avec le modèle de test. Nous la complèterons dans les exercices suivants.

3.1 Créer un project *Sirius / Viewpoint Specification Project* nommé `fr.enseeiht.simplePDL.design`, avec comme nom de modèle simplePDL.odesign (sur le deuxième écran). Il s'agit du modèle de description de l'interface de Sirius.

3.2 Dans Sirius, une vue graphique d'un modèle est appelée un *Point de vue* (Viewpoint). Nous allons en créer un en cliquant droit sur le dernier élément du modèle .odesign puis *New / Viewpoint*. Dans les propriétés de ce *Viewpoint*, donner comme Id « simplePDLViewpoint », et comme extension (*Model File Extension*) « simplepdl ».

3.3 Les points de vue permettent de décrire différents types de vues graphiques d'un même modèle. Définir une nouvelle vue graphique en cliquant droit sur le point de vue nouvellement créé puis *New Representation / Diagram Description*.

Définir son identifiant (*Id*), ici « ProcessDiagram », ainsi que l'élément de la syntaxe abstraite qu'il représente (*Domain Class*), ici « simplepdl.Process ». Le champ *Domain Class* dispose d'un fond vert car il attend le nom d'un élément de la syntaxe abstraite (le métamodèle). Nous reviendrons sur les champs à fond jaune un peu plus tard.

3.4 Nous disposons maintenant de la définition d'un diagramme pour les éléments de type *simplepdl.Process*. Pour la tester, se placer dans le projet `fr.enseeiht.simplePDL.sample`.

1. Faire un clic droit sur la racine du projet, sélectionner *Viewpoint Selection*, et choisir le Viewpoint précédemment créé.
2. Faire ensuite un clic droit sur l'élément Process à la racine du modèle puis *New Representation / new ProcessDiagram*.

Une vue graphique s'ouvre alors : l'éditeur de diagrammes de Processus. Pour l'instant cette vue est vide car nous n'avons créé aucune représentation graphique pour les éléments du modèle.
Remarque : L'extension *.aird* est l'acronyme pour Advanced Interactive Representation Data.

Exercice 4 : Définition de la partie graphique de l'éditeur
Afin d'afficher sur l'éditeur les différents éléments de nos modèles, nous devons définir leur style graphique. Lors de la définition d'un éditeur graphique, il arrive parfois que le nombre d'éléments à afficher sur une vue est tel qu'il devient difficile de comprendre le modèle affiché. Sirius permet de limiter ce problème par l'utilisation de calques. Un calque affiche des éléments graphiques et peut être activé ou désactivé.
Attention : Il faut enregistrer les modifications sur le .odesign pour qu'elles soient prises en compte sur le diagramme.
Conseil : Vérifier la cohérence du .odesign en cliquant droit sur la racine du fichier puis *Validate*.

4.1 Un nouveau calque *Default* a été ajouté au diagramme.

Dans ce calque, nous pouvons désormais définir les propriétés graphiques de chaque élément. Les éléments que nous pouvons créer sont les suivants :

- *Node* : Un nœud. Représente une métaclasse du modèle métier.
- *Relation Based Edge* : Un lien entre des nœuds. Ce lien est la représentation graphique d'une relation entre deux métaclasses du modèle métier.
- *Element Based Edge* : Un lien entre deux nœuds. Ce lien est la représentation graphique d'une métaclasse du modèle métier.
- *Container* : Un conteneur permettant d'afficher une métaclasse du modèle métier ainsi que des éléments référencés par un lien de composition à partir de cette métaclasse.
- *Decoration* : Permet de décorer un élément graphique avec un texte, une image, ...

4.2 *Représentation graphique des éléments WorkDefinition.* Créer un nœud (*New Diagram Element / Node*). Ce nœud sera la représentation des éléments *WorkDefinition*. Lui donner un *id*. Il faut ensuite remplir le champ *Domain Class* avec le nom de la métaclasse représentée par ce nœud (ici *WorkDefinition*).

Afin d'afficher ce nœud nous devons définir son style graphique. Faire un clique droit sur le nouveau *Node* puis *New Style / Ellipse*. Parcourir les options pour changer le style de l'ellipse et de son libellé.

Enregistrer et constater l'apparition des ellipses dans la vue graphique.

4.3 *Libellé des éléments WorkDefinition.* Le champ *Label Expression* est une expression qui détermine le label apposé sur les éléments WorkDefinition. Le fond jaune de ce champ signifie que son contenu sera évalué. Par défaut, il est initialisé à *feature:name*, c'est-à-dire, l'attribut *name* de la méta-classe.

Nous aurions aussi pu utiliser une expression Acceleo 3 pour afficher le nom : [name /].

4.4 *Représentation graphique des éléments WorkSequence.* De retour dans le fichier de design, créer un *Diagram element* : *Element based edge* dans le calque. Cela va créer une représentation graphique pour une flèche dans le diagramme. Remplir les champs *Id* et *Domain Class*.

Il nous faut maintenant définir les éléments qui sont la source et la cible de chaque *WorkSequence*. Dans les champs *Source Mapping* et *Target Mapping* choisir l'élément graphique correspondant (ici nous sélectionnerons bien évidement le nœud représentant une *WorkDefinition*).

Nous devons définir sur quel attribut de l'élément sémantique pointent la source et la cible de la flèche. Remplir les champs *Source Finder Expression* et *Target Finder Expression* respectivement avec feature:predecessor et feature:successor.

4.5 Afficher la valeur de l'attribut *linkType* comme libellé sur les flèches *WorkSequence* (utiliser *Label Expresssion* dans *Center Label Style 8* de *Edge Style Solid*).

4.6 Créer les éléments nécessaires pour afficher les nœuds *Guidance* ainsi que les flèches les reliant à leurs *ProcessElement* de rattachement (*Relation Based Edge*). S'inspirer de la représentation donnée dans la figure 2.

Exercice 5 : Définition de la palette

Disposer d'une vue graphique sur un modèle est pratique mais il nous faut aussi des outils pour manipuler le modèle au travers des objets graphiques de cette vue.

5.1 Dans le calque, créer une section pour héberger les outils : *Cliquer droit / New Tool... / Section*. Définir son *Id*.

5.2 *Outil de création de WorkDefinition.*. Dans la section, créer un outil de création d'éléments : *Cliquer droit > New Element Creation... / Node Creation*. Saisir son *Id* et le *Node Mappings* (qui doit pointer sur la vue graphique de l'élément que l'on veut créer).

Lors de l'initialisation d'un outil de création d'un nœud, deux variables sont créées (ne pas modifier leur noms : container et containerView) :

— *Node Creation Variable container* : Une variable pointant sur l'élément sémantique cliqué lors de l'utilisation de l'outil. Ici ce sera l'élément de type *simplepdl.Process*.

— *Container View Variable containerView* : Une variable pointant sur la représentation graphique de l'élément sémantique cliqué lors de l'utilisation de l'outil. C'est un élément de type *DSemanticDiagramSpec*.

Un élément *Begin* définira les actions à effectuer lors de l'utilisation de l'outil.

5.3 Nous devons décrire dans l'outil *Node Creation* sous l'élément *Begin*, les actions à exécuter pour effectuer l'action de l'outil. Ajouter une nouvelle opération de navigation : cliquer doit sur *Begin* puis *New operation... / Change Context*. Saisir l'expression [container/]. A partir de ce point, le contexte d'exécution des expressions se situera dans l'objet pointé par la variable *container*.

5.4 Maintenant que nous avons défini le contexte de l'action, nous devons créer un élément de type *simplepdl.WorkDefinition*. Cliquer droit sur l'opération *Change Context* puis *New Operation... / Create Instance*. Définir les propriétés de la nouvelle action :

— *Reference Name = processElements* (référence où sont stockés les éléments de type *simplepdl.WorkDefinition* dans le container, l'élément parent *simplepdl.Process*).

— *Type Name = simplepdl.WorkDefinition* (on veut créer une WorkDefinition).

— *Variable Name = instance*. Ce nom de variable servira à faire référence à cette variable si besoin dans la suite de la définition de l'outil.

5.5 Afin de définir un outil pour la création de flèches (edge), Cliquer droit sur la section des outils de création puis *New Element Creation... / Edge Creation*. Lors de la définition d'un *Edge Creation Tool*, et en plus de l'élément *Begin*, quatre variables sont créées :

— *Source Edge Creation Variable source* : Une variable pointant sur l'élément sémantique cliqué en premier lors de l'utilisation de l'outil : *simplepdl.WorkDefinition*.
— *Target Edge Creation Variable target* : Une variable pointant sur l'élément sémantique cliqué en deuxième lors de l'utilisation de l'outil : *simplepdl.WorkDefinition*.
— *Source Edge View Creation Variable sourceView* : Une variable pointant sur l'élément graphique représentant *source*.
— *Target Edge View Creation Variable targetView* : Une variable pointant sur l'élément graphique représentant *target*.

5.6 Créer une opération *Change Context...* dans l'élément *Begin*. Afin de créer une *WorkSequence*, nous devons en créer une nouvelle instance dans la liste de *ProcessElement* du process parent. Nous pouvons modifier le contexte en navigant vers le *Process* via l'expression : [source.eContainer()/]. Créer ensuite une opération *Create Instance* pour créer un élément de type *WorkSequence*.

5.7 Une fois cet élément créé, lui ajouter deux opérations *Set* : Clique droit puis *New Operation... / Set*. L'une de ces opérations permettra de définir la référence (*Feature Name*) *predecessor* à la valeur [source/], l'autre de définir la référence *successor* à la valeur [target/].

5.8 Définir les outils de création pour les éléments de type *simplepdl.Guidance* ainsi que la création des liens entre *simplepdl.Guidance* et *simplepdl.ProcessElement* (la référence *elements* de l'élément *simplepdl.Guidance*.

Exercice 6 : Quelques améliorations sur l'éditeur

Modifions maintenant quelques éléments de la représentation graphique.

6.1 Changer la représentation des éléments *WorkDefinition* afin de les dessiner à l'aide d'un losange.

6.2 Faire en sorte que le style graphique d'une *WorkSequence* soit fonction de son type (s2s, s2f, f2f, f2s).

6.3 Créer un nouveau calque dans la représentation. Déplacer l'affichage des éléments *Guidance* et de leurs liens dans l'affichage du nouveau calque.