Définition de syntaxes concrètes graphiques

Attention : Version d'Eclipse à utiliser : `/mnt/n7fs/ens/tp_dupont/modelling-2023-09/eclipse/eclipse` ou `/mnt/n7fs/ens/tp_cregut/eclipse-gls/eclipse`

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).

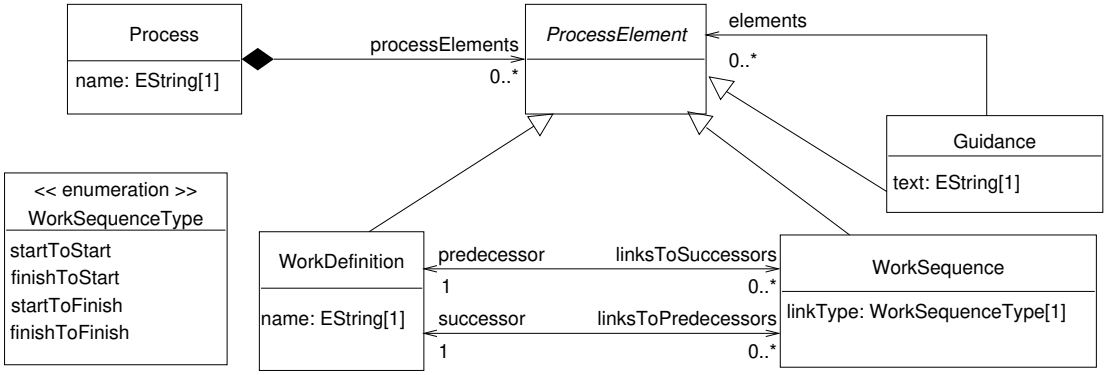

FIGURE 1 – Nouveau métamodèle de SimplePDL

Exercice 1 : Engendrer 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.n7.simplepdl` contenant le méta-modèle de SimplePDL (`SimplePDL.ecore`).
2. Créer le fichier `SimplePDL.genmodel` (s'il n'est pas présent) et faire *Generate All*.
3. Lancer l'Eclipse de deploiement depuis le premier Eclipse : Clic droit sur le projet, puis choisir *Run As* > *Eclipse Application*.

Tout le reste se fera maintenant en utilisant l'eclipse de déploiement.

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

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.).

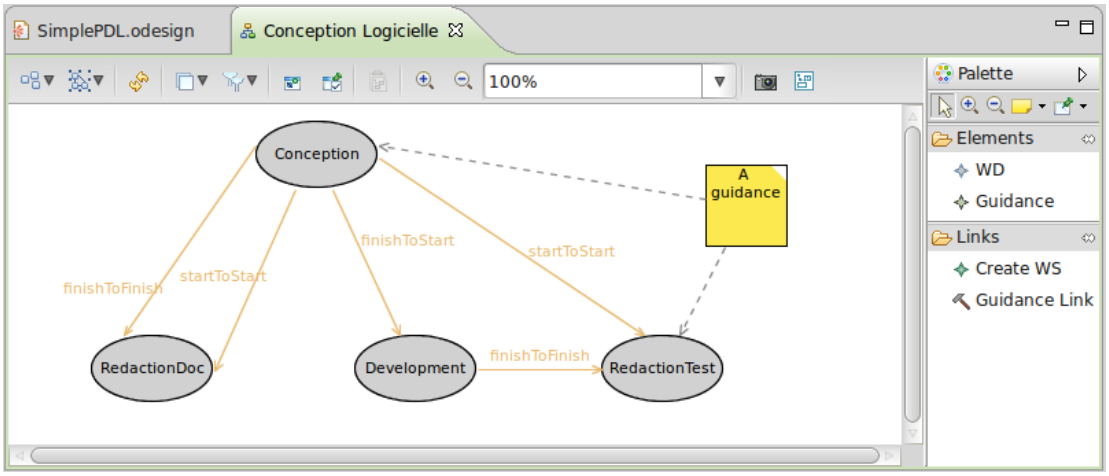

FIGURE 2 – Exemple d'éditeur possible pour SimplePDL

Définir une syntaxe graphique c'est donc :

1. définir comment représenter les éléments d'un modèle. Deux catégories d'éléments sont disponibles : les nœuds (rectangle, ellipse, losange, note, etc.) et des liens (ou flèches) au tracé et extrémités variés. 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 (dits éléments *sémantiques*) et les éléments graphiques. Cette correspondance doit être bidirectionnelle pour permettre de visualiser un modèle (il faut savoir à quels éléments graphiques correspondent les éléments du modèle) 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 propose les outils pour créer les différents éléments graphiques.

Sirius propose un unique métamodèle pour ces trois aspects. 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 processus

Commençons par créer un projet qui contiendra 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.n7.simplepdl.samples`. On peut aussi utiliser le projet existant qui contient déjà les exemples. Il faut cependant le convertir en un projet *Modeling*. Ceci peut être fait en faisant un clic droit sur le projet puis *Configure / Convert to Modeling Project*.

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

2.4. Vérifier que le modèle est bien chargé en le dépliant et qu'il est valide.

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.n7.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. Quand on déplie le modèle, on trouve un élément `simplepdlViewpoint`. Il s'agit d'un point de vue (*Viewpoint*) qui décrit une vue graphique d'un modèle. Modifier ses propriétés pour donner la valeur « simplepdlViewpoint » pour *Id* et « simplepdl » pour *Model File Extension* (extension).

3.3. Les points de vue permettent de décrire différents types de vues graphiques d'un même modèle. Définir une nouvelle vue graphique avec un clic 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. Sélectionner l'onglet *Metamodels*, faire un ajout depuis les greffons enregistrés du méta-modèle `http://simplepdl` (*Add from registry*).

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.n7.simplepdl.samples`.

1. Faire un clic droit sur la racine du projet, sélectionner *Viewpoint Selection*, et choisir le Viewpoint précédemment créé.
2. Déplier `developpement.simplepdl` dans le *Model Explorer* pour faire apparaître l'élément *Process developpement*. Sur cet élément, faire 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 représentation 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 pallier 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 régulièrement la cohérence du .odesign avec un clic droit sur la racine du fichier puis *Validate*.
4.1. Un calque *Default* est présent dans le diagramme *ProcessDiagram*.

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

— *Node* : Représentation graphique par un nœud d'une métaclasse [4] du modèle métier. Par exemple, une *WorkDefinition* sera représentée par un nœud de style *ellipse*.

— *Element Based Edge* : Représentation graphique par un lien d'une métaclasse du modèle métier. Un lien relie les représentations graphiques [5] de deux objets du modèle métier. Par exemple, une (instance de) *WorkSequence* sera représentée par un lien (une flèche) entre la représentation de sa *WorkDefinition* précédente (*predecessor*) et celle de sa *WorkDefinition* suivante (*successor*) [6].

— *Relation Based Edge* : Représentation par un lien d'une référence d'une métaclasse du modèle métier (et donc un lien entre la représentation d'un objet de cette classe et la représentation d'un objet du type de la référence). C'est par exemple le cas d'un lien entre une *Guidance* et un *ProcessElement* qui correspond à la référence *elements* de *Guidance*.

— *Container* : Représentation d'une métaclasse du modèle métier pouvant accueillir la représentation des objets qu'elle contient (accessibles par une relation de composition).

— *Decoration* : Décoration ajoutée sur une représentation graphique : 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* (`WDNode` par exemple). Il faut ensuite remplir le champ *Domain Class* avec le nom de la métaclasse représentée par ce nœud (ici `simplepdl::WorkDefinition`).

Afin d'afficher ce nœud nous devons définir son style graphique. Faire un clique droit sur le nouveau *Node* puis *New Style / Ellipse*.

4. Comme souvent, nous faisons abus de langage. Une formulation plus juste (mais plus longue et lourde) serait : une instance de nœud est la représentation graphique d'une instance d'une métaclasse du modèle métier.

5. Les représentations graphiques peuvent être des nœuds ou des liens. Par exemple, la représentation d'une *Guidance* peut être liée aux représentations de *WorkDefinition* (nœud), *WorkSequence* (lien), etc.

6. Par abus de langage, on dira souvent : « Une *WorkSequence* sera représentée par un lien entre sa *WorkDefinition* précédente et sa *WorkDefinition* suivante ».

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 sur simplepdl.odesign, créer un *Diagram element / Element based edge* dans le calque *Default*. Cela crée une représentation graphique sous forme de lien dans le diagramme. Remplir les champs *Id* (avec `WSEdge`) et *Domain Class* (avec `simplepdl::WorkSequence`).

Les champs *Source Mapping* et *Target Mapping* permettent d'indiquer les éléments graphiques qui seront la source et la cible du lien représentant une *WorkSequence*. Pour les deux choisir l'élément *WDNode* qui est la représentation des *WorkDefinition*.

Nous devons également définir comment identifier les *WDNodes* qui seront source et cible d'un *WSEdge*. Ceci se fait grâce aux champs *Source Finder Expression* et *Target Finder Expression*. On y renseigne les expressions qui, partant d'une *WorkSequence* identifieront les *WorkDefinitions* dont les représentations (*WDNode*) seront respectivement la source et la cible du *WSEdge* associé à cette *WorkSequence*. Les valeurs à donner ici sont : `feature:predecessor` et `feature:successor`.

Constater l'effet sur la vue graphique.

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

Constater l'effet sur la vue graphique.

4.6. Créer les éléments nécessaires pour afficher les nœuds *Guidance* ainsi que les liens les reliant à leurs *ProcessElement* (*Relation Based Edge*). S'inspirer de la représentation donnée à 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. Ces outils sont regroupés en sections (à l'image de ce que propose l'éditeur graphique Ecore avec les sections *Classifier*, *Relation*, etc.). Ici, nous ne définirons qu'une seule section.

5.1. *Créer une section.* Dans le calque, créer une section pour héberger les outils : Faire un clic droit puis sélectionner *New Tool... / Section*. Définir son *Id* (`OutilsSection`).

5.2. *Ajouter un outil pour créer une* WorkDefinition. Une *WorkDefinition* est représentée comme un nœud (*WDNode*). Pour créer un tel objet, le mode opératoire consiste à sélectionner dans la palette l'outil de création, puis à cliquer sur le container qui doit accueillir ce nouvel objet (ici *ProcessDiagram*), le *WDNode* est créé ainsi que la *WorkDefinition* associée. Cette *WorkDefinition* doit être ajoutée dans la référence `processElements` de *Process*. Voyons comment ceci se traduit avec Sirius.

5.2.1. *Créer l'outil de création de WorkDefinition.* Dans la section, créer un outil de création d'éléments : faire un clic droit pour sélectionenr *New Element Creation... / Node Creation*. Saisir son *Id* (`WDCreation`) et le *Node Mappings* (la représentation graphique de l'élément que l'on veut

créer : *WDNode*).

L'outil de création d'un nœud (*WDCreation* ici) possède deux sous-élements qui définissent chacun une variable (container et containerView) dont il ne faut pas changer le nom :

— *Container View Variable containerView* : Variable désignant la représentation graphique sur lequel l'utilisateur doit cliquer lors de l'utilisation de l'outil (*ProcessDiagram* ici).

— *Node Creation Variable container* : Variable désignant l'élément sémantique associé à la représentation graphique désignée par containerView. Ici ce sera l'élément de type `simplepdl::Process` (car *WDNode* est défini dans *ProcessDiagram* associé à *Process*).

5.2.2. *Définition des actions.* L'élément *Begin* de *Node Creation* permet de définir les actions à exécuter quand l'outil est utilisé. Ici, il s'agit de créer une nouvelle *WorkDefinition*.

La première action consiste à changer de contexte [7] (clic droit puis *New operation... / Change Context*). Saisir l'expression [container/]. Le contexte des expressions sera maintenant l'objet désigné par la variable *container* (ici *Process*).

Sur *Change Context*, faire un clic droit puis *New Operation... / Create Instance* ce qui ajoute l'action de creation d'une instance. Il faut préciser son type (*Type Name*), `simplepdl::WorkDefinition` et le nom de la référence où la ranger depuis le container (*Reference Name*) : `processElements` (c'est là que sont stockées les *WorkDefinition* dans un *Process*).

Variable Name est initialisé avec *instance*. Ce nom de variable servira à faire référence à cet objet si besoin dans la suite de la définition de l'outil.

5.2.3. *Utiliser l'outil.* Constater sur la vue graphique que la section créée est apparue dans la palette à droite avec un outil à l'intérieur. On pourra changer les *Label* de la *Section* et du *Node Creation* pour avoir de meilleurs noms.

On peut cliquer sur l'outil, puis sur le diagramme et ainsi créer une nouvelle *WorkDefinition* sur le diagramme. Il est bien ajouté à l'élément *Process*. On peut le constater en dépliant le contenu du modèle dans l'éditeur arborescent.

5.3. *Créer un outil pour définir une* WorkSequence. Une *WorkSequence* est représentée par un lien (*WSEdge*) reliant un *WDNode* source à un *WDNode* cible. Le mode opératoire consiste à cliquer sur l'outil, puis sur le *WDNode* source puis sur le *WDNode* cible. Le lien est alors créé (*WSEdge*) ainsi que la *WorkSequence* associée. Cette *WorkSequence* doit être enregistrée dans la référence `processElements` de *Process* et ses références `predecessor` et `successor` doivent être initialisées (avec les *WorkDefinitions* associées aux *WDNodes* source et cible). Voici comment le faire avec Sirius.

5.3.1. Faire un clic droit sur la section des outils puis sélectionner *New Element Creation... / Edge Creation*. L'élément *Edge Creation Tool* apparaît avec quatre sous-éléments :

— *Source Edge Creation Variable source* : Variable désignant le premier élément sémantique cliqué lors de l'utilisation de l'outil : `simplepdl::WorkDefinition`.

— *Target Edge Creation Variable target* : Variable désignant le deuxième élément sémantique cliqué lors de l'utilisation de l'outil : *simplepdl.WorkDefinition*.

7. Cette étape n'est pas nécessaire pour les *Node*. Sur l'opération *Create Instance* on aura accès à `processElements` même sans changer de contexte.

— *Source Edge View Creation Variable sourceView* : Variable désignant l'élément graphique représentant la source (*source*) du lien.

— *Target Edge View Creation Variable targetView* : Variable désignant l'élément graphique représentant la cible (*target*) du lien.

5.3.2. *Définition des actions.*

Créer une opération *Change Context...* dans l'élément *Begin* et renseigner *Browse Expression* avec `[self.eContainer()/]` [8]. Le contexte est maintenant le *Process*.

Créer une opération *Create Instance* pour créer un élément de type `simplepdl::WorkSequence` associé à la référence `processElements`.

Lui ajouter deux opérations *Set* : faire un clic droit puis *New Operation... / Set*. La première permet d'initialiser la référence `predecessor` (Feature Name) avec la source `[source/]`. La seconde initiale la référence `successor` avec la cible `[target/]`.

5.3.3. *Utiliser l'outil.* Passer sur la vue graphique et utiliser l'outil pour créer une dépendance et constater que la *WorkSequence* est bien ajoutée sur le modèle (via l'éditeur arborescent).

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

6.3. Organiser les outils de la palette en plusieurs sections en s'inspirant de la figure 2.

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

8. La fonction eContainer() permet de retrouver le conteneur d'un objet (relation de composition).