



**RAPPORT DE MINI-PROJET IDM :
CHAINE DE VERIFICATION DES MODELES DE PROCESSUS**

GROUPE : L34-04

MEMBRES:

TENE FOGANG ZACHARIE IGOR
OUDARD VERON
MEKKAOUI OSSAMA MOUSSA

TABLE DE MATIERE

Introduction.....	4
1. Les métamodèles SimplePDL et PetriNet.....	4
1.1 Métamodèle SimplePDL.....	4
1.2 Métamodèle PetriNet.....	7
3. Définition de syntaxes concrètes graphiques avec Sirius.....	8
4. Définition des contraintes java sur les modèles.....	10
4.1 Contraintes sur SimplePDL.....	10
4.2 Contraintes sur le modèle PetriNet.....	13
5. Définition d'une transformation de modèle à modèle (M2M).....	15
5.1 Définir une transformation SimplePDL vers PetriNet en utilisant EMF/Java.....	15
5.2 Définir une transformation SimplePDL vers PetriNet en utilisant ATL..	17
6. Définition de transformations modèle à texte (M2T) avec Acceleo.....	18
6.1 Transformation d'un modèle de Process en Tina.....	18
6.2 Transformation d'un modèle de Process en Ltl.....	19
7. Définition d'une grammaire avec XText pour le métamodèle SimplePDL..	22
7.1 Syntaxe de la grammaire.....	22
7.2 Transformation ATL de PDL1 vers SimplePDL.....	24
Conclusion.....	24

TABLE DE FIGURES

<i>Figure 1 : Métamodèle SimplePDL sans ressource</i>	<i>5</i>
<i>Figure 2 : Métamodèle SimplePDL avec ressources</i>	<i>6</i>
<i>Figure 3: Modèle de processus pdl-sujet.xmi.....</i>	<i>7</i>
<i>Figure 4 : Métamodèle PetriNet.....</i>	<i>8</i>
<i>Figure 5 : Editeur graphique sirius.....</i>	<i>9</i>
<i>Figure 6 : Vue arborescente de l'éditeur graphique</i>	<i>9</i>
<i>Figure 7 : résultat du test de validation du modèle ko-pdl-process1.xmi.....</i>	<i>10</i>
<i>Figure 8 : résultats du test de validation du modèle ko-pdl-process2.xmi</i>	<i>11</i>
<i>Figure 9 : résultats du test de validation du modèle ko-pdl-process_reflexive.xmi</i>	<i>11</i>
<i>Figure 10 : résultat du test de validation du modèle ko-pdl-wsduplicate.xmi.....</i>	<i>11</i>
<i>Figure 11 : résultat du test validation du modèle ko-pdl- processRessourceNonValide.xmi.....</i>	<i>12</i>
<i>Figure 12 : résultats du test de validation du modèle ko-pdl- guidanceNonConforme.xmi</i>	<i>13</i>
<i>Figure 13 : résultats du test de validation du modèle ko-net-violationArc.xmi.....</i>	<i>13</i>
<i>Figure 14: résultats du test de validation du modèle ko-net-violateTokenPositif.xmi. 13</i>	
<i>Figure 15 : résultats du test de validation du modèle ko-net-petrinet.xmi</i>	<i>14</i>
<i>Figure 16 : résultats du test de validation du modèle ko-net-violationPoids.xmi</i>	<i>14</i>
<i>Figure 17: chemin source et destination à configurer pour la transformation</i>	<i>16</i>
<i>Figure 18: net-pdl-sujet-java (résultat de la transformation).....</i>	<i>17</i>
<i>Figure 19 : résultat de la transformation net-pdl-sujet en pdl-sujet.net.....</i>	<i>18</i>
<i>Figure 20: visualisation graphique sur Tina de pdl-sujet.net (resultant de la transformation M2T)</i>	<i>19</i>
<i>Figure 21 : SimplePDL-finish.mtl</i>	<i>20</i>
<i>Figure 22 : Vérification de la terminaison avec du réseau avec selt.....</i>	<i>21</i>
<i>Figure 23 : SimplePDL-invariants.mtl.....</i>	<i>22</i>
<i>Figure 24 : Exemple d'utilisation de la syntaxe.....</i>	<i>23</i>

Introduction

Ce projet a pour objectif de concevoir une chaîne de validation de modèles de processus créés avec SimplePDL. À l'aide des outils découverts en TP, nous déterminerons si les processus peuvent atteindre une fin, en nous basant principalement sur la boîte à outils Tina. Pour optimiser l'utilisation de Tina, il a été nécessaire de mettre en œuvre une conversion d'un modèle Simplepdl vers un réseau de Petri. Cette transformation a été réalisée grâce à un ensemble d'outils de modélisation disponibles dans l'environnement Eclipse. Notre approche s'est appuyée sur plusieurs technologies complémentaires : l'infrastructure EMF et le méta-modèle Ecore pour la base de la modélisation, Xtext pour la syntaxe textuelle, Sirius pour la représentation graphique, Java pour les contraintes, ainsi que ATL et Aceleo pour les transformations de modèles.

1. Les métamodèles SimplePDL et PetriNet

La mise en place de notre chaîne de vérification de modèle de processus nécessite une étape fondamentale de métamodélisation. Cette phase implique la définition de deux métamodèles essentiels : celui de Simplepdl pour les processus, et celui de PetriNet pour la représentation en réseaux de Petri. Pour les réaliser, nous nous sommes appuyés sur l'éditeur graphique Ecore d'Eclipse EMF(Eclipse Modeling Framework).

1.1 Métamodèle SimplePDL

SimplePDL est un langage simplifié de modélisation de processus de développement sur lequel nous allons nous appuyer par la suite. Son métamodèle décrit plusieurs activités (WorkDefinition) ainsi que des dépendances (WorkSequence). De plus, pour qu'une activité puisse être réalisée, elle devra généralement disposer de ressources afin d'en garantir l'achèvement. Une ressource se compose d'occurrences, et une activité alloue ces occurrences au début de son exécution. Ces occurrences restent alors exclusivement affectées à cette activité jusqu'à son achèvement.

On a donc réalisé le métamodèle en deux étapes : sans ressources et avec ressources. Voir Figures 1 et 2.

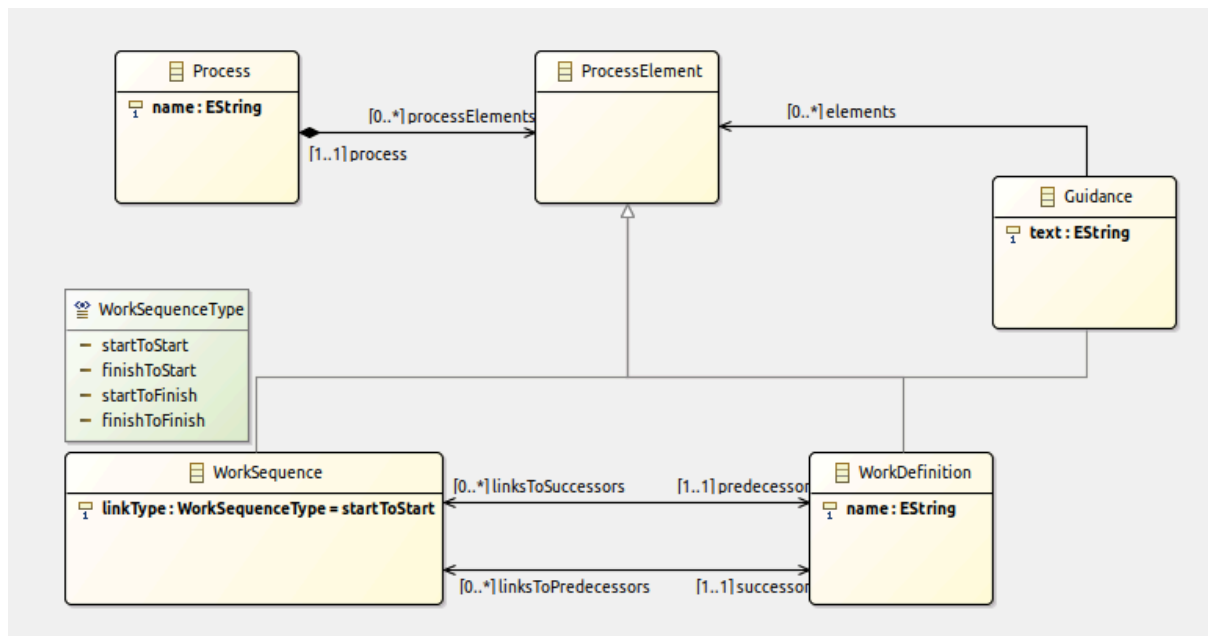


Figure 1 : Métamodèle SimplePDL sans ressource

Dans le cadre de l'évolution de notre projet, nous avons intégré la gestion des ressources au processus. Cette implémentation s'articule autour de deux composants principaux :

1. Une Eclass "Ressource" qui englobe :
 - La définition de la ressource elle-même
 - Un compteur indiquant la quantité disponible
2. Un gestionnaire d'allocation ("RessourceAllocator") qui détermine :
 - Le volume de ressources requis pour chaque tâche
 - L'association entre les ressources et les activités

Dans notre modèle, nous avons établi que :

- Les ressources font partie intégrante du flux de processus
- Une activité peut fonctionner avec un nombre variable de gestionnaires d'allocation (de zéro à plusieurs)
- Chaque activité peut nécessiter différents types et quantités de ressources

Cette approche permet une gestion flexible des ressources tout en maintenant une structure claire et organisée.

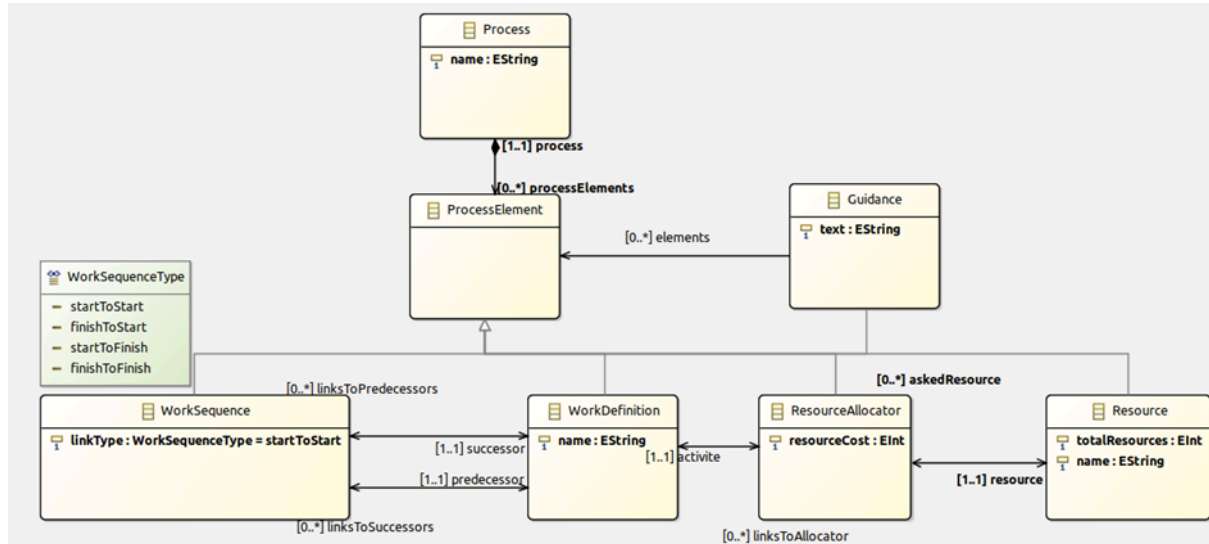


Figure 2 : Métamodèle SimplePDL avec ressources

Un modèle (developpement.xmi) nous a été fourni dans le cadre du mini-projet et nous l'avons complété en utilisant notre métamodèle afin de rajouter les ressources:

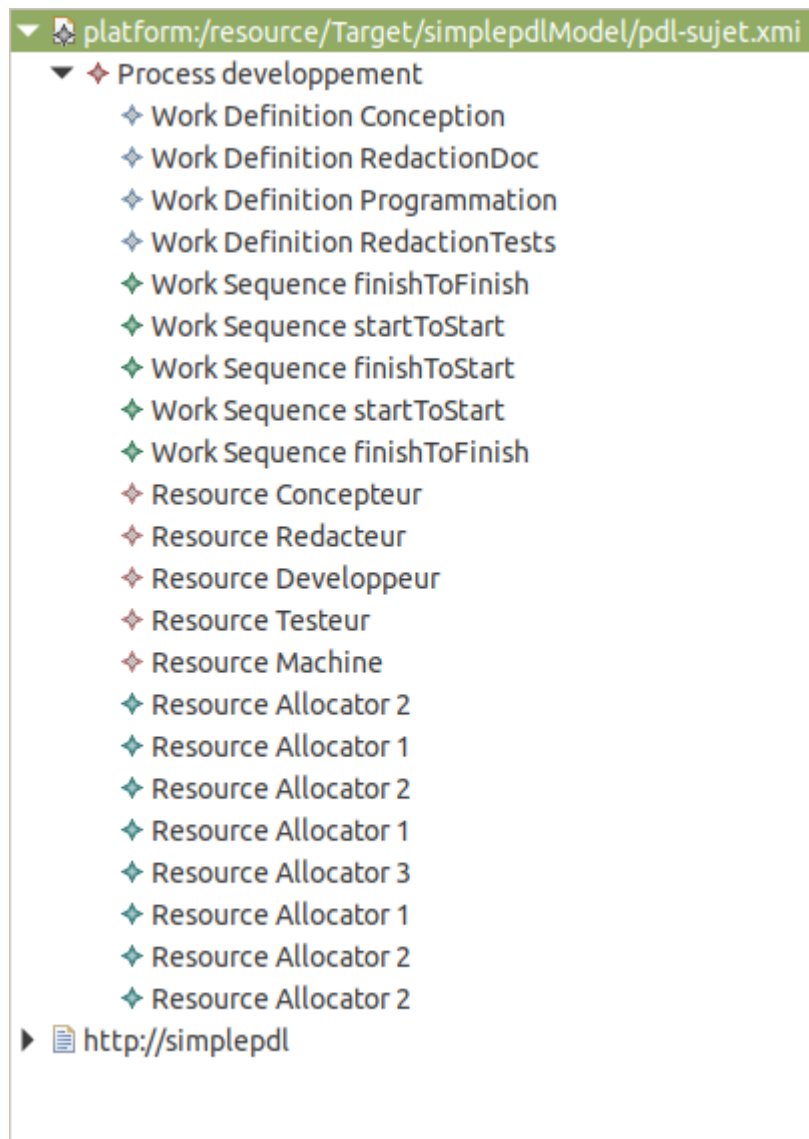


Figure 3 : Modèle de processus pdl-sujet.xmi

1.2 Métamodèle PetriNet

Un Réseau de Petri est une structure graphique comportant un ensemble de places et de transitions, reliées par des arcs orientés, éventuellement porteurs de poids. Ces arcs sont des liens entre place et transition ou entre transition et place exclusivement. Donc en se basant sur ce que l'on a vu pour le modèle SimplePDL, on peut introduire le méta modèle suivant:

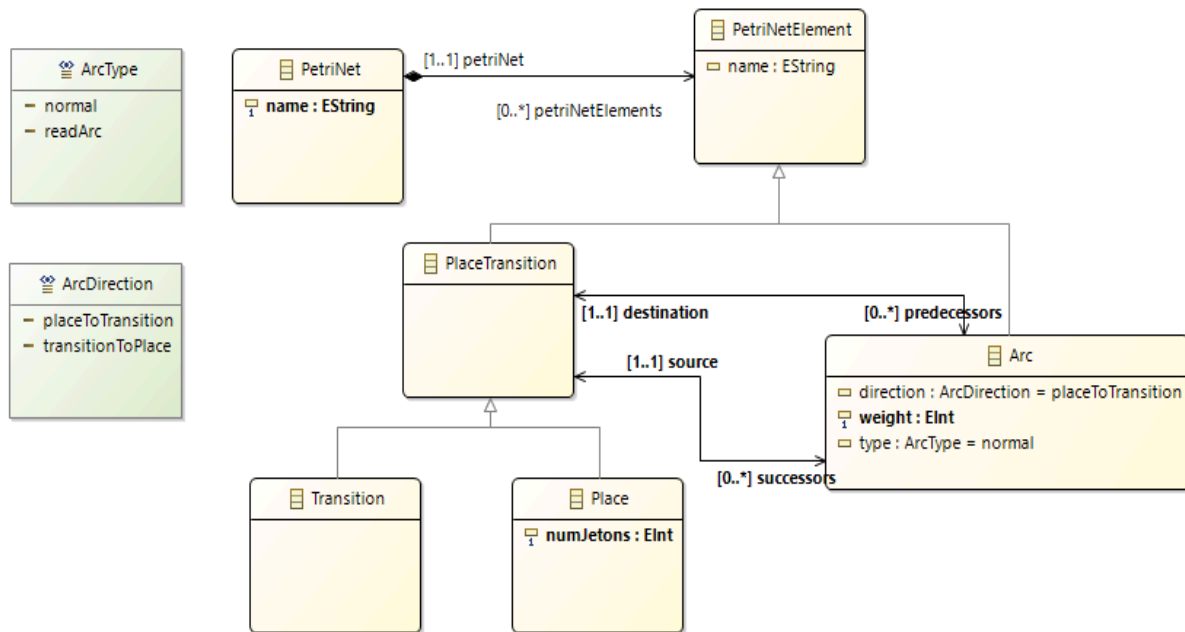


Figure 4 : Métamodèle PetriNet

3. Définition de syntaxes concrètes graphiques avec Sirius

La définition de syntaxes concrètes graphiques est essentielle pour rendre la création et la modification des modèles plus accessibles. A l'aide de l'outil Sirius manipulé en TP, on peut créer un éditeur graphique intégré à Eclipse pour des langages de modélisation définis en Ecore.

Donc dans le cadre de notre projet, nous avons conçu un éditeur graphique pour le métamodèle SimplePDL, où les éléments du modèle : WorkDefinitions, WorkSequences et les ressources, sont représentés visuellement. On peut ajouter, déplacer, nommer/renommer et supprimer ces éléments.

La figure figure 4 montre un exemple de représentation d'un modèle simplePDL avec les outils permettant de créer ce modèle sur la droite de la figure tandis que la figure 5 montre une arborescence qui définit les outils et les éléments du modèle simplePDL.

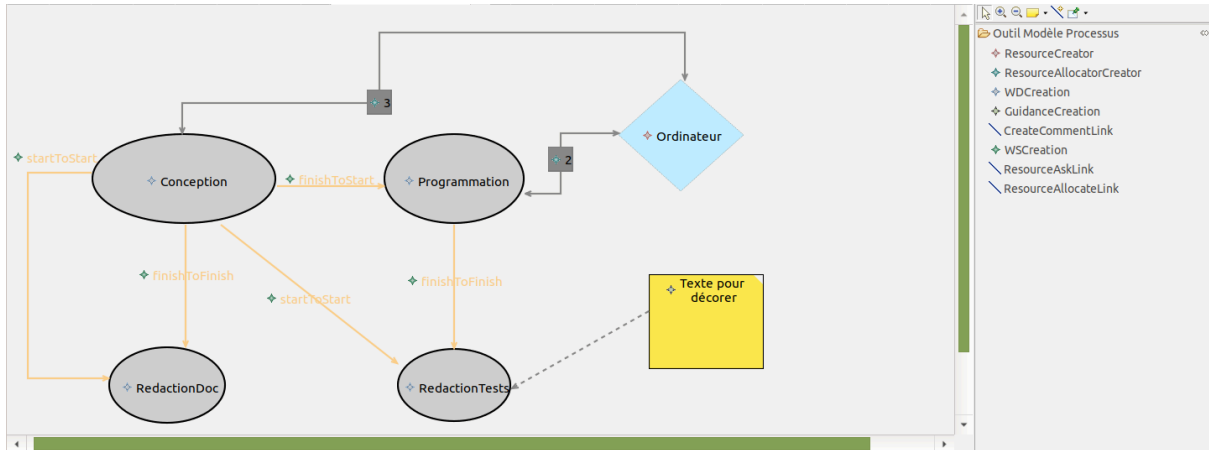


Figure 5 : Editeur graphique sirius

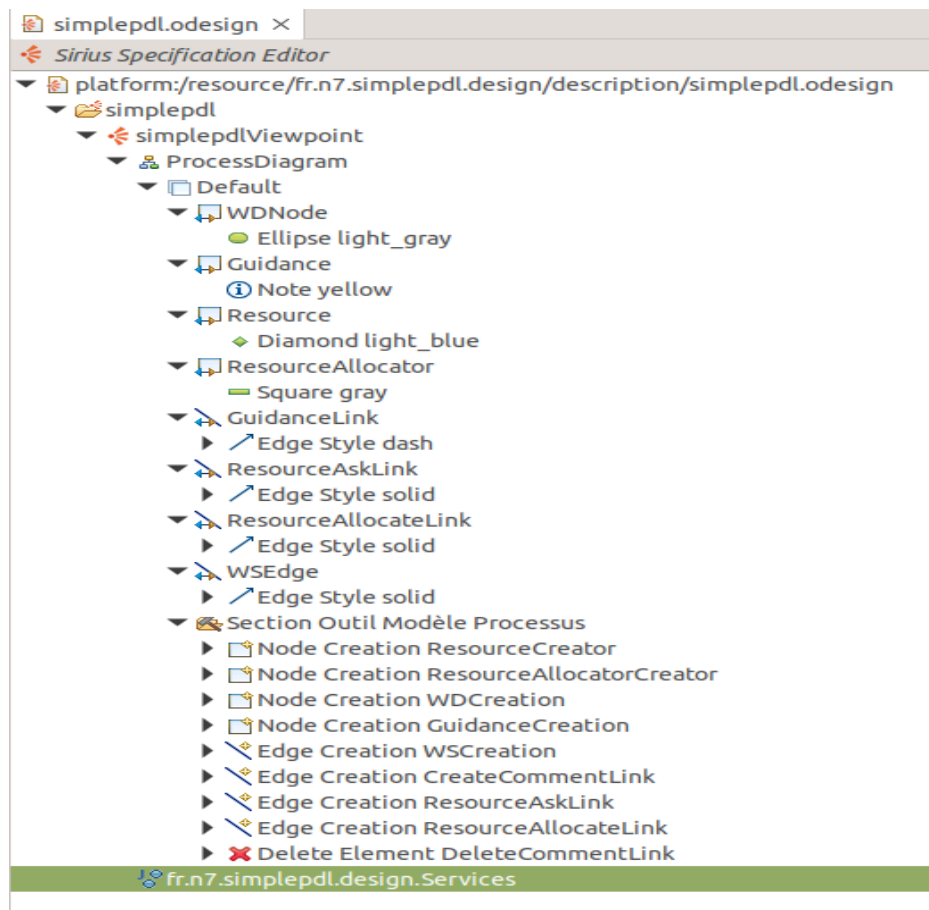


Figure 6 : Vue arborescente de l'éditeur graphique

4. Définition des contraintes java sur les modèles

A l'aide de Java, on va définir ce que normalement on ne peut pas avec le langage de méta-modélisation Ecore, ce sont les contraintes que doivent respecter les modèles de processus. Autrement dit, les règles qu'on crée servent à définir ce qui est permis ou non dans un modèle. En mettant ces limites en place, il est plus simple de vérifier que tout fonctionne, de garder une trace de ce qui a été fait, et d'être sûr que le système est bien construit. Ainsi cela assure un travail plus organisé et fiable.

4.1 Contraintes sur SimplePDL

Les règles développées pour simplepdl sont :

- Les noms (processus, activités et ressources) doivent commencer par une lettre ou un souligné et être composé d'au moins deux caractères alphanumériques:

```
Résultat de validation pour file:/E:/C2/eclipse-workspace/fr.n7.simplePDL/model/ko-pdl-process1.xmi:
- Process: OK
- WorkDefinition: 4 erreurs trouvées
=> Erreur dans @m [simplepdl.impl.WorkDefinitionImpl@20ce78ec (name: @m)]: Le nom de l'activité ne respecte pas les conventions Java ou ne contient pas au moins 02 caracteres
=> Erreur dans 8djk [simplepdl.impl.WorkDefinitionImpl@6973b51b (name: 8djk)]: Le nom de l'activité ne respecte pas les conventions Java ou ne contient pas au moins 02 caract
=> Erreur dans wdRessource [simplepdl.impl.WorkDefinitionImpl@1ab3a8c8 (name: wdRessource)]: Le nom 'wdRessource' est déjà utilisé par une autre ressource ou activité dans ce
=> Erreur dans q [simplepdl.impl.WorkDefinitionImpl@43195e57 (name: q)]: Le nom de l'activité ne respecte pas les conventions Java ou ne contient pas au moins 02 caracteres a
- WorkSequence: OK
- Guidance: OK
- Resource: 4 erreurs trouvées
=> Erreur dans doublon [simplepdl.impl.ResourceImpl@333291e3 (totalResources: 0, name: doublon)]: Le nombre d'occurrences d'une ressource doit être supérieur à 1
=> Erreur dans doublon [simplepdl.impl.ResourceImpl@479d31f3 (totalResources: 0, name: doublon)]: Le nom 'doublon' est déjà utilisé par une autre ressource ou activité dans c
=> Erreur dans doublon [simplepdl.impl.ResourceImpl@479d31f3 (totalResources: 0, name: doublon)]: Le nombre d'occurrences d'une ressource doit être supérieur à 1
=> Erreur dans wdRessource [simplepdl.impl.ResourceImpl@40ef3420 (totalResources: 0, name: wdRessource)]: Le nombre d'occurrences d'une ressource doit être supérieur à 1
- ResourceAllocator: OK
Fini.
```

Figure 7 : résultat du test de validation du modèle ko-pdl-process1.xmi

-
- Les ressources, les activités doivent avoir des noms différents. Ces noms doivent être unique:

```
Résultat de validation pour file:/home/ztg4419/Téléchargements/IDM/PROJECT/eclipse-workspace/Target/simplepdlContrainte/ko-pdl-process2.xmi:
- Process: OK
- WorkDefinition: 3 erreurs trouvées
=> Erreur dans A1 [simplepdl.impl.WorkDefinitionImpl@17046283 (name: A1)]: Le nom de l'activité (A1) n'est pas unique
=> Erreur dans A1 [simplepdl.impl.WorkDefinitionImpl@10e41621 (name: A1)]: Le nom 'A1' est déjà utilisé par une autre ressource ou activité dans ce processus
=> Erreur dans A1 [simplepdl.impl.WorkDefinitionImpl@10e41621 (name: A1)]: Le nom de l'activité (A1) n'est pas unique
- WorkSequence: 1 erreurs trouvées
=> Erreur dans simplepdl.impl.WorkSequenceImpl@353d0772 (linkType: startToStart): La dépendance relie l'activité A2 à elle-même
- Guidance: OK
- Resource: OK
- ResourceAllocator: OK
Fini.
```

Figure 8 : résultats du test de validation du modèle ko-pdl-process2.xmi

-
- Une dépendance ne doit pas relier une activité à elle-même (une activité ne doit pas dépendre d'elle-même) :
-

```
<terminated> ValidateSimplePdl [Java Application] /usr/lib/jvm/java-17-openjdk-amd64/bin/java (24 oct. 2024, 11:05:38 - 11:05:39) [pid:1163618]
Résultat de validation pour file:/home/ztg4419/Téléchargements/IDM/PROJECT/eclipse-workspace/Target/simplePdlContrainte/ko-pdl-process_reflexive.xmi:
- Process: OK
- WorkDefinition: OK
- WorkSequence: 1 erreurs trouvées
=> Erreur dans simplePdl.impl.WorkSequenceImpl@6e4784bc (linkType: finishToFinish): La dépendance relie l'activité programmation à elle-même
- Guidance: OK
- Resource: OK
- ResourceAllocator: OK
Fini.
```

Figure 9 : résultats du test de validation du modèle ko-pdl-process_reflexive.xmi

-
- Deux dépendances différentes de même type entre deux mêmes activités ne peuvent coexister:
-

```
<terminated> ValidateSimplePdl [Java Application] /usr/lib/jvm/java-17-openjdk-amd64/bin/java (24 oct. 2024, 11:02:48 - 11:02:49) [pid:1162717]
Résultat de validation pour file:/home/ztg4419/Téléchargements/IDM/PROJECT/eclipse-workspace/Target/simplePdlContrainte/ko-pdl-wsduplicate.xmi:
- Process: OK
- WorkDefinition: OK
- WorkSequence: 2 erreurs trouvées
=> Erreur dans simplePdl.impl.WorkSequenceImpl@2667f029 (linkType: finishToStart): Il existe déjà une dépendance de même type entre 'Task1' et 'Task2'
=> Erreur dans simplePdl.impl.WorkSequenceImpl@57c758ac (linkType: finishToStart): Il existe déjà une dépendance de même type entre 'Task1' et 'Task2'
- Guidance: OK
- Resource: OK
- ResourceAllocator: OK
Fini.
```

Figure 10 : résultat du test de validation du modèle ko-pdl-wsduplicate.xmi

- La quantité en besoin d'une ressource et le nombre d'occurrence d'une ressource doivent être des entiers naturels. La quantité en besoin ne peut pas être nulle et le nombre d'occurrences(total) d'une ressource inférieure à 1:

```
Résultat de validation pour file:/home/ztg4419/Téléchargements/IDM/PROJECT/eclipse-workspace/Target/simplepdlContrainte/ko-pdl-processRessourceNonValide.xmi:
- Process: OK
- WorkDefinition: OK
- WorkSequence: OK
- Guidance: OK
- Resource: 1 erreurs trouvées
=> Erreur dans InvalidResource [simplepdl.impl.ResourceImpl@19bb07ed (totalResources: 0, name: InvalidResource)]: Le nombre d'occurrences d'une ressource doit être supérieur à 1
- ResourceAllocator: 1 erreurs trouvées
=> Erreur dans simplepdl.impl.ResourceAllocatorImpl@9225652 (resourceCost: -2): La quantité en besoin d'une ressource doit être strictement supérieure à 0
Fini.
```

Figure 11: résultat du test validation du modèle
ko-pdl-processRessourceNonValide.xmi

- Une guidance ne doit pas avoir un texte vide ou ne contenir que des espaces:

```
Résultat de validation pour file:/home/ztg4419/Téléchargements/IDM/PROJECT/eclipse-workspace/Target/simplepdlContrainte/ko-pdl-guidanceNonConforme.xmi:
- Process: OK
- WorkDefinition: OK
- WorkSequence: OK
- Guidance: 2 erreurs trouvées
=> Erreur dans simplepdl.impl.GuidanceImpl@be64738 (text: ): La guidance doit avoir un texte non vide et ne peut pas contenir que des espaces
=> Erreur dans simplepdl.impl.GuidanceImpl@61230f6a (text: ): La guidance doit avoir un texte non vide et ne peut pas contenir que des espaces
- Resource: OK
- ResourceAllocator: OK
Fini.
```

Figure 12: résultats du test de validation du modèle
ko-pdl-guidanceNonConforme.xmi

4.2 Contraintes sur le modèle PetriNet

Pour les modèles Petrinet, les règles que nous avons définies sont:

- Un arc ne doit pas relier une transition à une transition ou une place à une place:

```
Résultat de validation pour file:/home/ztg4419/Téléchargements/IDM/PROJECT/eclipse-workspace/Target/petrinetContrainte/ko-net-violationArc.xmi:
- Petrinet: OK
- Arc: 3 erreurs trouvées
=> Erreur dans petrinet.impl.ArcImpl@3cc2931c (name: ArcInvalide) (direction: placeToTransition, weight: 1, type: normal):
Un arc doit toujours relier une transition ou une place et non deux places/transitions'
=> Erreur dans petrinet.impl.ArcImpl@5c909414 (name: ArcInvalide2) (direction: placeToTransition, weight: 0, type: normal):
Un arc doit toujours relier une transition ou une place et non deux places/transitions'
=> Erreur dans petrinet.impl.ArcImpl@5c909414 (name: ArcInvalide2) (direction: placeToTransition, weight: 0, type: normal): Le poids d'un arc doit etre supérieur ou égal à 1
- Place: OK
- Transition: OK
Fini.
```

Figure 13: résultats du test de validation du modèle *ko-net-violationArc.xmi*

-
- Le nombre de jeton d'une place est un entier positif (Toute place doit avoir avoir un nombre entier naturel de jetons):
-

```
Résultat de validation pour file:/home/ztg4419/Téléchargements/IDM/PROJECT/eclipse-workspace/Target/petrinetContrainte/ko-net-violateTokenPositif.xmi:
- Petrinet: OK
- Arc: OK
- Place: 1 erreurs trouvées
=> Erreur dans petrinet.impl.PlaceImpl@75329a49 (name: ProduitAControler) (numJetons: -1): Le nombre de jeton d'une place ou d'une transition doit etre un entier naturel
- Transition: OK
Fini.
```

Figure 14: résultats du test de validation du modèle *ko-net-violateTokenPositif.xmi*

- Les noms doivent commencer par une lettre ou soulignés et au moins deux caractères alphanumériques. Le nom d'une place ou d'une transition doit être unique:

```
Résultat de validation pour file:/home/ztg4419/Téléchargements/IDM/PROJECT/eclipse-workspace/Target/petrinetContrainte/ko-net-petrinet.xmi:
- Petrinet: 1 erreurs trouvées
=> Erreur dans 1InvalidName [petrinet.impl.PetriNetImpl@1e802ef9 (name: 1InvalidName)]: Le nom du petrinet ne respecte pas les conventions Java
- Arc: 1 erreurs trouvées
=> Erreur dans petrinet.impl.ArcImpl@5ccdd20 (name: InvalidWeightArc) (direction: placeToTransition, weight: 0, type: normal): Le poids d'un arc doit etre supérieur ou égal à 1
- Place: 6 erreurs trouvées
=> Erreur dans petrinet.impl.PlaceImpl@1ed1993a (name: P1) (numJetons: 1): Le nom de la place (P1) n'est pas unique
=> Erreur dans petrinet.impl.PlaceImpl@1f3f4916 (name: P1) (numJetons: 2): Le nom 'P1' est déjà utilisé par une autre place ou transition
=> Erreur dans petrinet.impl.PlaceImpl@1f3f4916 (name: P1) (numJetons: 2): Le nom de la place (P1) n'est pas unique
=> Erreur dans petrinet.impl.PlaceImpl@794cb805 (name: InvalidPlace) (numJetons: -1): Le nombre de jeton d'une place ou d'une transition doit etre un entier naturel
=> Erreur dans petrinet.impl.PlaceImpl@4b5a5ed1 (name: 1) (numJetons: 1): Le nom de la place ne respecte pas les conventions Java
=> Erreur dans petrinet.impl.PlaceImpl@4b5a5ed1 (name: 1) (numJetons: 1): Le nom de la place (1) n'est pas unique
- Transition: 2 erreurs trouvées
=> Erreur dans petrinet.impl.TransitionImpl@59d016c9 (name: T1): Le nom 'T1' est déjà utilisé par une autre place ou transition
=> Erreur dans petrinet.impl.TransitionImpl@3cc2931c (name: CommonName): Le nom 'CommonName' est déjà utilisé par une autre place ou transition
Fini.
```

Figure 15: résultats du test de validation du modèle *ko-net-petrinet.xmi*

-
- Le poids d'un arc doit être supérieure à 1:

```
Résultat de validation pour file:/home/ztg4419/Téléchargements/IDM/PROJECT/eclipse-workspace/Target/petrinetContrainte/ko-net-violationPoids.xmi:
- Petrinet: OK
- Arc: 1 erreurs trouvées
=> Erreur dans petrinet.impl.ArcImpl@1e802ef9 (name: ArcInvalide) (direction: placeToTransition, weight: 0, type: normal): Le poids d'un arc doit etre supérieur ou égal à 1
- Place: OK
- Transition: OK
Fini.
```

Figure 16: résultats du test de validation du modèle *ko-net-violationPoids.xmi*

5. Définition d'une transformation de modèle à modèle (M2M)

5.1 Définir une transformation SimplePDL vers PetriNet en utilisant EMF/Java

La transformation d'un modèle de processus SimplePDL en un modèle de réseau de Pétri passe par plusieurs étapes, facilitées par l'infrastructure EMF.

Donc dans le code java, en chargeant les packages SimplePDL et PetriNet dans le registre d'Eclipse, on se base sur la stratégie de transformation suivante (vue en TD):

Une WorkDefinition est équivalente à 4 places: name_ready, name_started, name_running et name_finished, 2 transitions: name_start, name_finish et 5 arcs:

- *ready -> start -> running*
- *running -> finish -> finished*
- *start -> started*

Une WorkSequence est un arc (read_arc) reliant un nœud de l'activité source et une transition de l'activité transition. Selon la nature de la liaison, on distingue:

- *FINISH_TO_START: place_finished -> transition_start*
- *FINISH_TO_FINISH: place_finished -> transition_finish*
- *START_TO_START: place_started -> transition_start*
- *START_TO_FINISH: place_started -> transition_finish*

Les Ressources de leur part sont transformées en places avec comme marquage initial (jetons) la quantité disponible des ressources et les ResourceAllocators(des arcs) qui contrôlent l'accès à ces ressources sont transformées en arcs. Chaque Resource est ainsi transformée en une Place nommée "name_resource" avec:

- *Nombre de jetons = totalResources de la ressource*
- *Arcs pour l'allocation: - Arc de la place ressource vers transition start*
- *Arc de la transition finish vers la place ressource (coût = resourceCost)*

Grâce à cette approche, nous avons implémenté un code Java (SimplePDL2PetriNet.java) qui réalise automatiquement cette transformation.

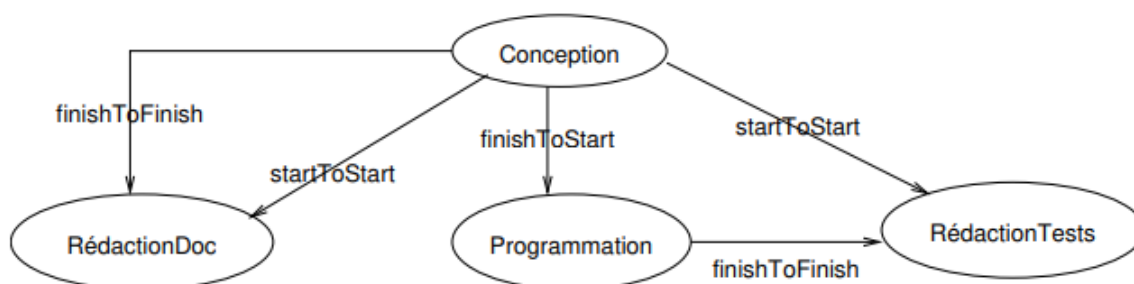
Pour effectuer la transformation, il suffit de spécifier le chemin du fichier .xmi représentant le modèle et le répertoire de sortie où sera généré le modèle petrinet ,résultat de la transformation.

```
//charger le modele simplePDL
URI simplepdlURI = URI.createURI("models/pdl-sujet.xmi");
Resource simplepdlResource = resSet.getResource(simplepdlURI, true);

// Créer un ResourceSet pour le modèle PetriNet
ResourceSet resSetPetriNet = new ResourceSetImpl();
//definir le modele
URI petriNetURI = URI.createURI("models/net-pdl-sujet-java.xmi");
Resource petriNetResource = resSetPetriNet.createResource(petriNetURI);
```

Figure 17: chemin source et destination à configurer pour la transformation

On applique donc notre transformation sur pdl-sujet.xmi (contenant les ressources) modélisant :



▼ ♦ Petri Net developpement	♦ Arc Conception_Acquire_Machine	♦ Arc Programmation_Finish_OUT
♦ Place Concepteur_resource	♦ Arc Conception_Release_Machine	♦ Arc Programmation_Started
♦ Place Redacteur_resource	♦ Arc RedactionDoc_Start_IN	♦ Place Programmation_ready
♦ Place Developpeur_resource	♦ Arc RedactionDoc_Start_OUT	♦ Place Programmation_finished
♦ Place Testeur_resource	♦ Arc RedactionDoc_Finish_IN	♦ Place Programmation_started
♦ Place Machine_resource	♦ Arc RedactionDoc_Finish_OUT	♦ Place Programmation_running
♦ Arc Conception_Start_IN	♦ Arc RedactionDoc_Started	♦ Transition Programmation_start
♦ Arc Conception_Start_OUT	♦ Place RedactionDoc_ready	♦ Transition Programmation_finish
♦ Arc Conception_Finish_IN	♦ Place RedactionDoc_finished	♦ Arc Programmation_Acquire_Developpeur
♦ Arc Conception_Finish_OUT	♦ Place RedactionDoc_started	♦ Arc Programmation_Release_Developpeur
♦ Arc Conception_Started	♦ Place RedactionDoc_running	♦ Arc Programmation_Acquire_Machine
♦ Place Conception_ready	♦ Transition RedactionDoc_start	♦ Arc Programmation_Release_Machine
♦ Place Conception_finished	♦ Transition RedactionDoc_finish	♦ Arc RedactionTests_Start_IN
♦ Place Conception_started	♦ Arc RedactionDoc_Acquire_Redacteur	♦ Arc RedactionTests_Start_OUT
♦ Place Conception_running	♦ Arc RedactionDoc_Release_Redacteur	♦ Arc RedactionTests_Finish_IN
♦ Transition Conception_start	♦ Arc RedactionDoc_Acquire_Machine	♦ Arc RedactionTests_Finish_OUT
♦ Transition Conception_finish	♦ Arc RedactionDoc_Release_Machine	♦ Arc RedactionTests_Started
♦ Arc Conception_Acquire_Concepteur	♦ Arc Programmation_Start_IN	♦ Place RedactionTests_ready
♦ Arc Conception_Release_Concepteur	♦ Arc Programmation_Start_OUT	♦ Place RedactionTests_finished
	♦ Arc Programmation_Finish_IN	♦ Place RedactionTests_started

Figure 18: *net-pdl-sujet-java (résultat de la transformation)*

5.2 Définir une transformation SimplePDL vers PetriNet en utilisant ATL

ATL est un langage de transformation de modèles qui permet de définir des transformations entre différents modèles conformément à des métamodèles.

Ainsi en se basant sur la même stratégie que celle qui nous a permis de transformer un modèle SimplePDL en modèle PetriNet avec Java, nous avons pu écrire un code qui réalise cette transformation sur ATL. Ainsi il suffit de configurer le module ATL en lui spécifiant le chemin d'un modèle SimplePDL en entrée et le chemin en sortie pour qu'il réalise la transformation.

6. Définition de transformations modèle à texte (M2T) avec Acceleo

Comme on a pu faire, à l'aide de ATL ou de JAVA, une transformation modèle à modèle (M2M), on peut également faire une transformation modèle à texte (M2T) avec Acceleo qui permet s'appuyer sur des gabarits (templates) des fichiers à engendrer.

6.1 Transformation d'un modèle de Process en Tina

```
1 net developpement
2 pl Concepteur_resource (3)
3 pl Redacteur_resource (1)
4 pl Developpeur_resource (2)
5 pl Testeur_resource (2)
6 pl Machine_resource (4)
7 pl Conception_ready (1)
8 pl Conception_finished (0)
9 pl Conception_started (0)
10 pl Conception_running (0)
11 pl RedactionDoc_ready (1)
12 pl RedactionDoc_finished (0)
13 pl RedactionDoc_started (0)
14 pl RedactionDoc_running (0)
15 pl Programmation_ready (1)
16 pl Programmation_finished (0)
17 pl Programmation_started (0)
18 pl Programmation_running (0)
19 pl RedactionTests_ready (1)
20 pl RedactionTests_finished (0)
21 pl RedactionTests_started (0)
22 pl RedactionTests_running (0)
23 tr Conception_start Conception_ready Concepteur_resource*2 Machine_resource*2 -> Conception_running Conception_started
24 tr Conception_finish Conception_running -> Conception_finished Concepteur_resource*2 Machine_resource*2
25 tr RedactionDoc_start RedactionDoc_ready Redacteur_resource Machine_resource Conception_started?1 -> RedactionDoc_running RedactionDoc_started
26 tr RedactionDoc_finish RedactionDoc_running Conception_finished?1 -> RedactionDoc_finished Redacteur_resource Machine_resource
27 tr Programmation_start Programmation_ready Developpeur_resource*2 Machine_resource*3 Conception_finished?1 -> Programmation_running Programmation_
28 tr Programmation_finish Programmation_running -> Programmation_finished Developpeur_resource*2 Machine_resource*3
29 tr RedactionTests_start RedactionTests_ready Testeur_resource Machine_resource*2 Conception_started?1 -> RedactionTests_running RedactionTests_sta
30 tr RedactionTests_finish RedactionTests_running Programmation_finished?1 -> RedactionTests_finished Testeur_resource Machine_resource*2
31
```

Figure 19: résultat de la transformation net-pdl-sujet en pdl-sujet.net

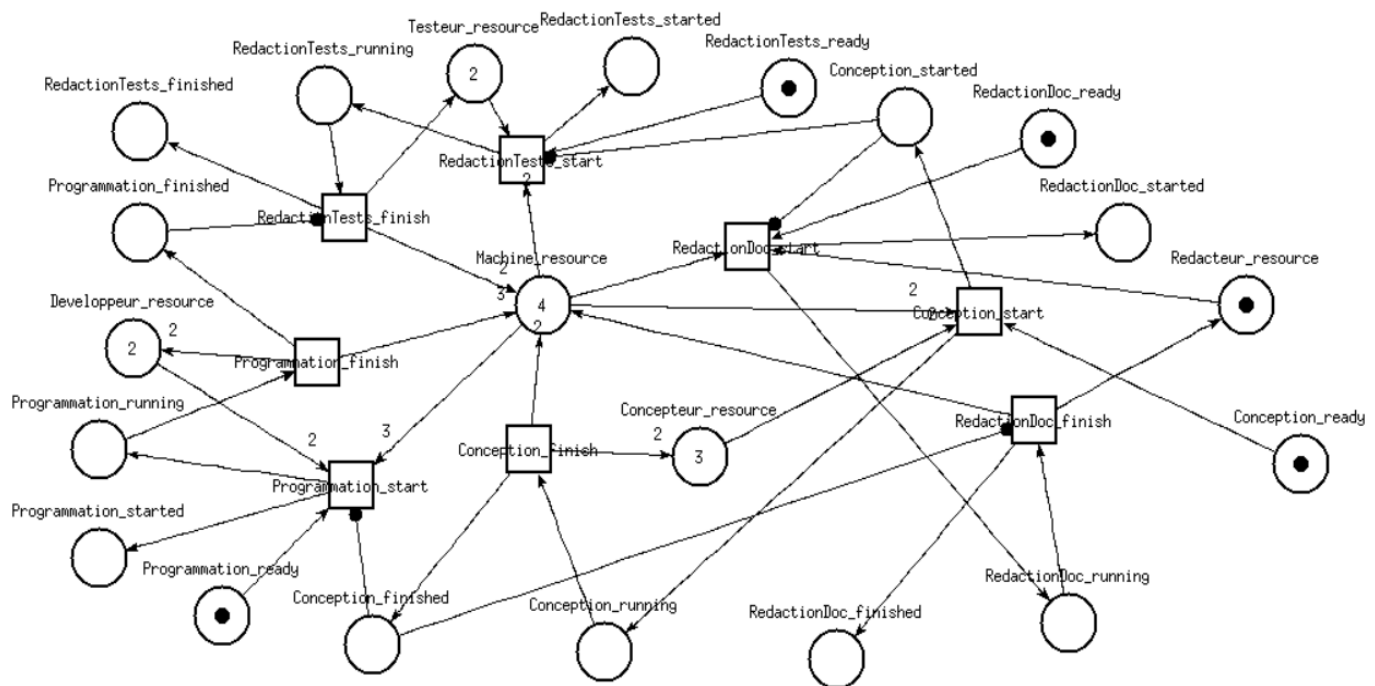


Figure 20: visualisation graphique sur Tina de *pdl-sujet.net* (resultant de la transformation M2T)

6.2 Transformation d'un modèle de Process en Ltl

LTL (Logique Temporelle Linéaire) est un outil qu'on va utiliser pour définir des propriétés que l'on souhaite vérifier sur un réseau de Petri

Les formules LTL peuvent présenter des "variables", qui sont des références à des places.

Pareil à la partie toTina, on a créé deux projets *acceleio* qui prennent un *model* de process *simplpdl*, et génèrent deux fichiers avec extension (.ltl) qui vont contenir les propriétés de terminaison du petrinet les invariants qu'on a définis.

Voici les figures qui montrent le code .mtl :

```

1 [comment encoding = UTF-8 /]
2 [module toLTL('http://simplepdl')]
3
4 [template public processToLtl(aProcess : Process)]
5 [comment @main/]
6 [file (aProcess.name + '.ltl', false, 'UTF-8')]
7 [let wds : OrderedSet(WorkDefinition) = aProcess.getWDs() ]
8 op ready = [for (wd : WorkDefinition | wds)] [if (not(wds->first() = wd))] /\ [/if][wd.name/]_ready[/for];
9 op started = [for (wd : WorkDefinition | wds)] [if (not(wds->first() = wd))] /\ [/if][wd.name/]_started[/for];
10 op running = [for (wd : WorkDefinition | wds)] [if (not(wds->first() = wd))] /\ [/if][wd.name/]_running[/for];
11 op finished = [for (wd : WorkDefinition | wds)] [if (not(wds->first() = wd))] /\ [/if][wd.name/]_finished[/for][[/let];
12 ['[]'/] (finished => dead);
13 ['[]'/] (dead => finished);
14 - <> finished;
15 ['[]'/] <> dead;
16
17
18
19 [/file]
20 [/template]
21
22 [query public getWDs(p: Process) : OrderedSet(WorkDefinition) =
23     p.processElements->select( e | e.ocIsTypeOf(WorkDefinition) )
24     ->collect( e | e.ocLAsType(WorkDefinition) )
25     ->asOrderedSet()
26 /]

```

Figure 21: *SimplePDL-finish.mtl*

Explication:

- À l'aide des variables et d'opérateur de conjonction on définit les états des workdefinitions (exemple : un état finished est lorsqu'il y a un jeton dans tout les places correspondantes aux activités finished).
- On définit une règle qui vérifie si un état finished est bloquant et vice versa
- On utilise la négation pour vérifier s'il y aura un état où tout les activités sont finished
- De même on vérifie s'il y aura un état bloquant

Donc on peut vérifier avec l'outil selt (vu en Tp1), sur notre psl-sujet.net :

ktz loaded, 22 states, 33	FALSE
0.001s	state 0: Concepteur_
operator ready : prop	ctionDoc_ready Redacti
0.000s	-Conception_start ..
operator started : prop	state 8: L.dead Conc
0.000s	finished Programmation
operator running : prop	rted Testeur_resource*
0.000s	-L.deadlock ... (pre
operator finished : prop	state 9: L.dead Conc
0.000s	finished Programmation
TRUE	rted Testeur_resource*
0.000s	[accepting all]
FALSE	0.000s
state 0: Concepteur_res	TRUE
ctionDoc_ready RedactionT	0.000s
-Conception_start ... (

Figure 22 : Vérification de la terminaison avec du réseau avec selt

Donc l'outil selt vérifie que :

- un état finished est bloquant en renvoyant TRUE
- un état bloquant n'est pas forcément un état finished en renvoyant un contre exemple, et on constate que c'est en effet un état finished avec des un jeton dans chaque état started, on peut donc ajouter ce cas dans le variable finished pour qu'il affiche bien TRUE, mais c'est à titre d'exemple)
- le réseau arrive bien à un état finished, selt renvoie FALSE et un contre exemple
- le réseau arrive bien à un état bloquant

De même pour les invariants, on se contente de montrer le code SimplePDL-invariants.mtl, voir la figure suivante:

```

[comment encoding = UTF-8 /]
[module toLTL_invariants('http://simplepdl')]

[template public simplepdlToLTL_invariants(aProcess : Process)]

[comment @main/]
[file (aProcess.name.concat('_invariants.ltl'), false, 'UTF-8')]

[let wds : OrderedSet(WorkDefinition) = aProcess.getWDs() ]
op ready = [for (wd : WorkDefinition | wds)] [if (not(wds->first() = wd))] /\ [if][wd.name/]_ready[/for];
op started = [for (wd : WorkDefinition | wds)] [if (not(wds->first() = wd))] /\ [if][wd.name/]_started[/for];
op running = [for (wd : WorkDefinition | wds)] [if (not(wds->first() = wd))] /\ [if][wd.name/]_running[/for];
op finished = [for (wd : WorkDefinition | wds)] [if (not(wds->first() = wd))] /\ [if][wd.name/]_finished[/for][let];

['[]'/] (running => ready /\ started /\ - <> finished);
['[]'/] (finished => - <> ready /\ started /\ - <> running);
['[]'/] (- <> started => - <> running /\ - <> finished);

[/file]
[/template]

[query public getWDs(p: Process) : OrderedSet(WorkDefinition) =
    p.processElements->select( e | e.ocIsTypeOf(WorkDefinition) )
    ->collect( e | e.ocAsType(WorkDefinition) )
    ->asOrderedSet()
/]

```

Figure 23: *SimplePDL-invariants.mtl*

7. Définition d'une grammaire avec XText pour le métamodèle SimplePDL

Xtext, un composant du Textual Modeling Framework (TMF), il permet de définir des syntaxes concrètes textuelles pour les DSML, facilitant ainsi la saisie et la modification de modèles. Contrairement à l'éditeur arborescent d'EMF ou aux classes Java, peu pratiques pour manipuler directement des modèles conformes à un métamodèle, Xtext offre un environnement intégré à Eclipse avec un éditeur syntaxique qui comprend la coloration, la complétion et la détection d'erreurs. En définissant une grammaire textuelle dans un fichier **.xtext**, comme pour SimplePDL, il est possible de générer des métamodèles adaptés.

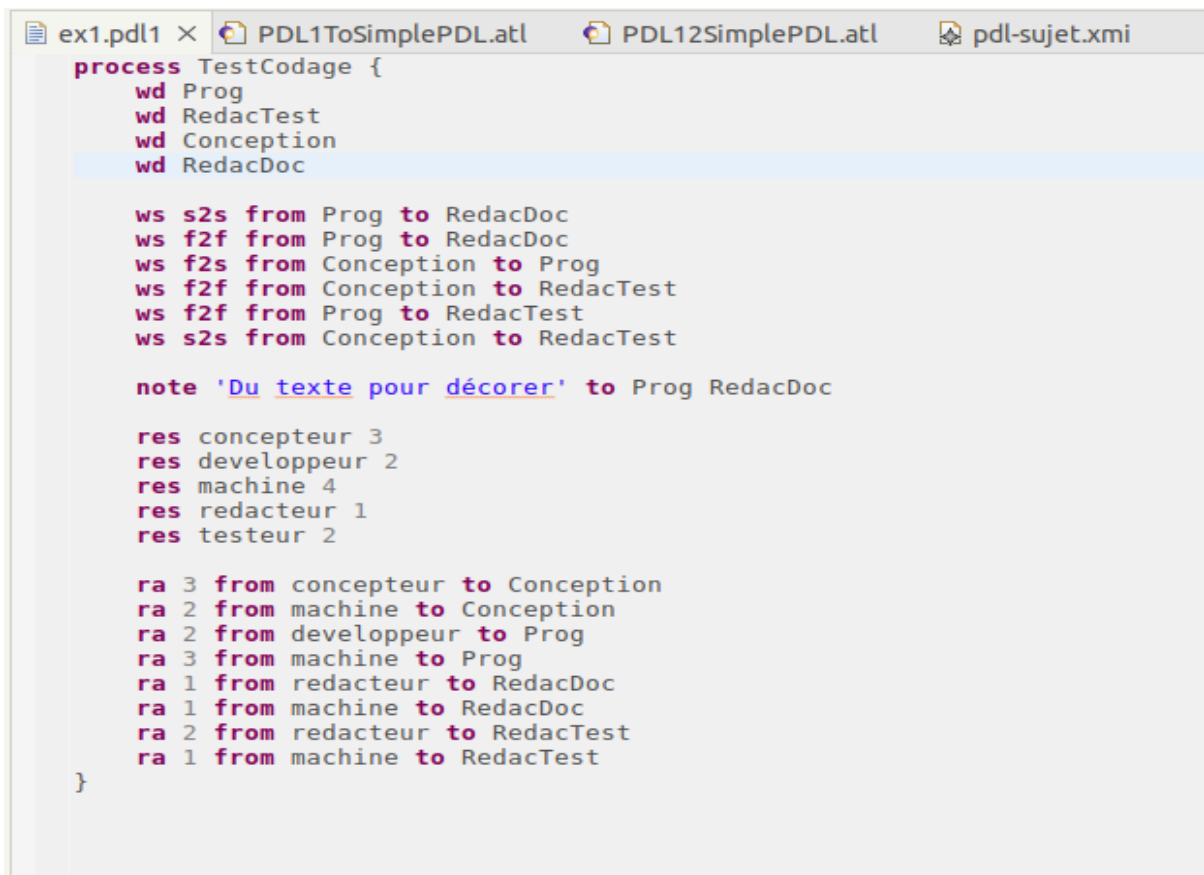
7.1 Syntaxe de la grammaire

Tous le défi de réaliser une syntaxe pour un métamodèle réside dans le fait que ladite syntaxe ne doit pas être ambiguë. Or notre métamodèle SimplePDL est composé de classes qui se référence les unes les autres. Ainsi, il est impossible de créer une syntaxe permettant de créer des modèles exactement conformes au métamodèle.

En effet, si une telle syntaxe est créée, alors pour définir une WorkDefinition par exemple, il faudrait lui donner en argument les WorkSequence pour permettre de définir les attributs linksToPredecessors et linksToSuccessors. Cependant, pour créer une WorkSequence il faudrait également donner les arguments permettant de définir les attributs predecessors et successors qui sont des WorkDefinition.

Par conséquent, il est nécessaire de faire un choix et de retirer des attributs pour éviter une grammaire ambiguë. Dans notre cas, le plus simple était de retirer les attributs de WorkDefinition car il y en a à la fois pour définir les WorkSequence mais également pour définir les ResourceAllocator et ces attributs sont définis sous forme de liste(ce qui est plus complexe à implémenter dans une grammaire). De même, dans la définition de la syntaxe, on retire l'attribut askedResources de la classe Resource.

Finalement, on peut voir une utilisation de la grammaire à la figure 24(le code de la grammaire étant parmi les fichiers rendu sur gitlab) et on voit bien que pour définir les workdefinitions et les ressources, il n'est pas nécessaire de préciser certains de leurs attributs.



```
process TestCodage {
  wd Prog
  wd RedacTest
  wd Conception
  wd RedacDoc

  ws s2s from Prog to RedacDoc
  ws f2f from Prog to RedacDoc
  ws f2s from Conception to Prog
  ws f2f from Conception to RedacTest
  ws f2f from Prog to RedacTest
  ws s2s from Conception to RedacTest

  note 'Du texte pour décorer' to Prog RedacDoc

  res concepteur 3
  res developpeur 2
  res machine 4
  res redacteur 1
  res testeur 2

  ra 3 from concepteur to Conception
  ra 2 from machine to Conception
  ra 2 from developpeur to Prog
  ra 3 from machine to Prog
  ra 1 from redacteur to RedacDoc
  ra 1 from machine to RedacDoc
  ra 2 from redacteur to RedacTest
  ra 1 from machine to RedacTest
}
```

Figure 24: Exemple d'utilisation de la syntaxe

7.2 Transformation ATL de PDL1 vers SimplePDL

Le métamodèle qui découle de la grammaire définit précédemment n'est pas tout à fait conforme au métamodèle SimplePDL que nous avons défini au début. En effet, nous avons pu constater qu'il a fallu supprimer certains attributs pour que le langage ne soit pas ambiguë. Par conséquent, si l'on veut utiliser la syntaxe, il faut réaliser une transformation du métamodèle PDL1 que nous venons de définir vers le modèle SimplePDL. Par ailleurs, comme les deux métamodèles sont très proches l'un de l'autre, il n'y a aucune difficulté à réaliser une transformation ATL de PDL1 vers SimplePDL et le code de cette transformation est dans le rendu gitlab.

Conclusion

En définitive, travailler sur ce mini-projet en Ingénierie des Modèles (IDM) a été une expérience véritablement enrichissante. Nous avons progressivement compris à quel point les méthodes et les outils d'automatisation peuvent être pertinents et avantageux, notamment en matière de gain de temps. Cependant, il est important de noter que l'utilisation d'Eclipse a posé certains défis, notamment en raison de bugs et dysfonctionnements récurrents qui ont parfois ralenti notre progression.