
IDM: Mini Projet

Rapport

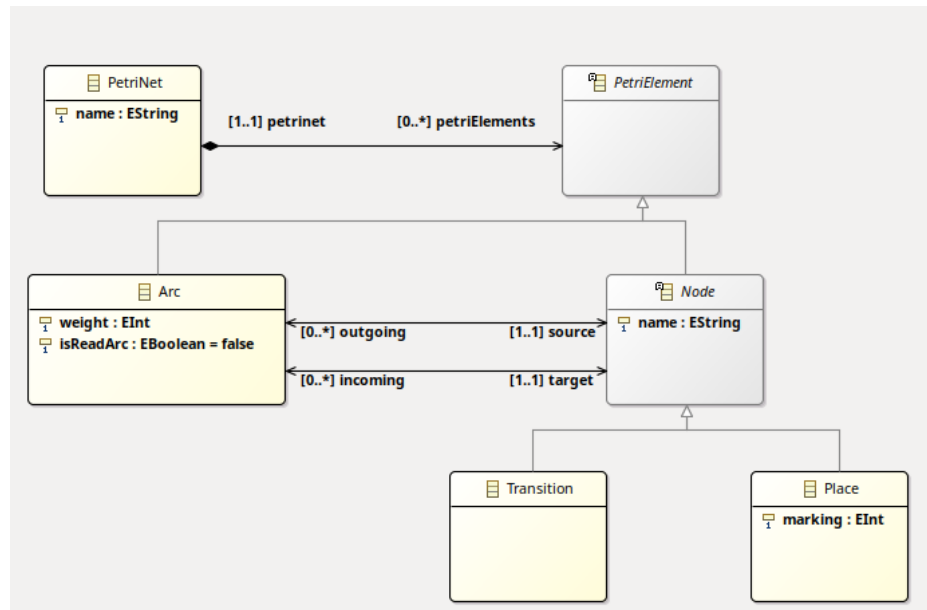
1/ Introduction	3
2/ Les Métamodèles	4
2.1/ SimplePDL	4
2.2/ PetriNet	4
3/ Les contraintes OCL	6
3.1/ Définition des contraintes	6
3.2/ Violation des contraintes	7
4/ Transformation Modèle à modèle (SimplePDL ver PetriNet)	8
4.1/ ATL	8
4.3/ Validation de la transformation SimplePDL vers PetriNet sur plusieurs exemples	9
5/ Transformation modèle à texte avec Acceleo	11
5.1/ ToTina	11
5.2/ ToLTL	13
6/ Syntaxe graphique avec Sirius	15
7/ Syntaxe textuelle avec xtext	17
8/ Conclusion	19
Annexe 0 - Glossaire des technologies utilisées	20

1/ Introduction

L'objectif du Mini-projet est de finaliser une chaîne de transformation et de validation commencé en TP, en ajoutant la notion de Ressource au méta modèle SimplePDL. Pour ce faire on a utilisé l'application Eclipse et les différents outils (ATL , Acceleo, etc.) vu en cours et en TP.

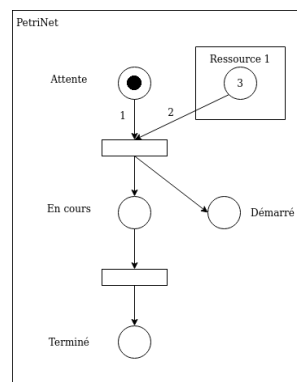
La chaîne de transformation consiste à définir les deux métamodèles SimplePDL et PetriNet et ses contraintes OCL. Par la suite, définir une transformation modèle à modèle utilisant ATL et une transformation modèle à texte de Petrinet vers Tina. En dernier, engendrer les propriétés LTL afin de valider la transformation modèle à modèle.

On a également utilisé les outils qui permettent de décrire un syntaxe graphique et texte.



Métamodèle SimplePDL

Cela correspond également au modèle étudié en cours. Une différence réside en la classe *Node* qui vient généraliser les branches d'un Arc en une seule classe. Comme étudié en cours, une ressource peut être interprété comme une *Place* dont l'attribut *marking* correspond au nombre de ressource disponible à un instant *t*. On retrouve alors une place par ressources. Le besoin est traduit par une transition. Le nombre de ressource nécessaire correspond au poids/*weight* de l'*Arc*.



Utilisation de ressource en PetriNet

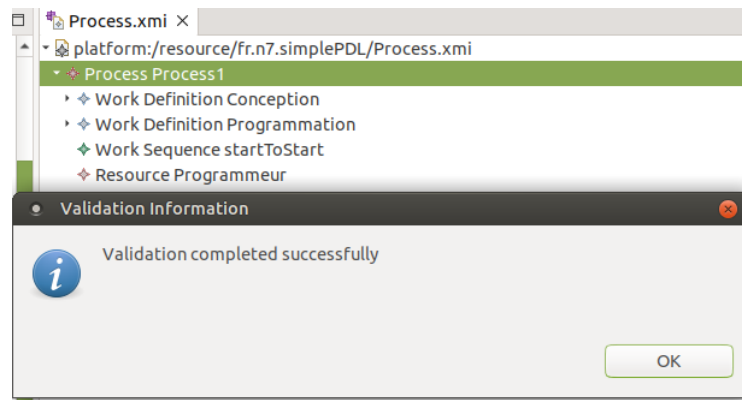
3/ Les contraintes OCL

OCL est un outil permettant l'écriture de contraintes sur modèle. En parcourant les éléments d'un modèle et en accédant à leurs propriétés, nous pouvons spécifier des invariants complexes qui garantissent la cohérence du modèle.

3.1/ Définition des contraintes

Nous appliquons ce principe sur les métamodèles SimplePDL et PetriNet. Les contraintes sont énumérés ci-dessous:

SimplePDL	PetriNet
Process <ul style="list-style-type: none"> - Nom valide 	PetriNet <ul style="list-style-type: none"> - Nom valide
WorkSequence <ul style="list-style-type: none"> - <i>Predecessor</i> et <i>successor</i> appartiennent à au même <i>Process</i>. - <i>Predecessor</i> et <i>successor</i> sont différents. 	Arc <ul style="list-style-type: none"> - Associe bien une <i>Transition</i> et une <i>Place</i>, ou inversement. - La <i>Transition</i> et la <i>Place</i> appartiennent au même <i>PetriNet</i>. - Unicité de l'arc. - <i>weight</i> strictement positif - Si <i>incoming</i> et une <i>Transition</i>, alors <i>isReadArc</i> est à <i>false</i>.
WorkDefinition <ul style="list-style-type: none"> - Nom unique - Nom valide 	Place <ul style="list-style-type: none"> - <i>marking</i> strictement positif.
Resource <ul style="list-style-type: none"> - Nom unique - Nom valide - Quantité totale strictement positif 	Node <ul style="list-style-type: none"> - Nom unique - Nom valide
Need <ul style="list-style-type: none"> - Quantité attendu strictement positive et inférieur au nombre de ressource totale - La <i>WorkDefinition</i> et la <i>Resource</i> associés appartiennent au même <i>Process</i>. - Unicité pour chaque association <i>WorkDefinition/Resource</i> 	



En appliquant les contraintes à un modèle valide, cela fonctionne.

3.2/ Violation des contraintes

Nous avons généré des modèles *.xmi* qui violent volontairement les contraintes OCL. Ils sont disponibles dans les projets *fr.n7.simplePDL.example* et *fr.n7.petriNet*.

SimplePDL	PetriNet																				
<p>developpement_KO_ressources Nombre de ressources disponibles nul.</p> <table> <tr> <th>Property</th><th>Value</th></tr> <tr> <td>Name</td><td>Machine</td></tr> <tr> <td>Nb Available Resources</td><td>0</td></tr> <tr> <td>Needs</td><td>Need 5</td></tr> </table> <p>Résultat:</p> <p>⚠ Resource property "nbAvailableResources" should be greater or equal to 1</p>	Property	Value	Name	Machine	Nb Available Resources	0	Needs	Need 5	<p>petrinet_KO_arc_position</p> <table> <tr> <th>Property</th><th>Value</th></tr> <tr> <td>Is Read Arc</td><td>false</td></tr> <tr> <td>Petrinet</td><td>Petri Net KO</td></tr> <tr> <td>Source</td><td>Place P1</td></tr> <tr> <td>Target</td><td>Place P2</td></tr> <tr> <td>Weight</td><td>1</td></tr> </table> <p>Résultat:</p> <p>⚠ All arcs should be between Place and Transition</p>	Property	Value	Is Read Arc	false	Petrinet	Petri Net KO	Source	Place P1	Target	Place P2	Weight	1
Property	Value																				
Name	Machine																				
Nb Available Resources	0																				
Needs	Need 5																				
Property	Value																				
Is Read Arc	false																				
Petrinet	Petri Net KO																				
Source	Place P1																				
Target	Place P2																				
Weight	1																				
<p>developpement_KO_ressources <i>Predecessor</i> identique au <i>Successor</i></p> <table> <tr> <th>Property</th><th>Value</th></tr> <tr> <td>Link Type</td><td>finishToFinish</td></tr> <tr> <td>Predecessor</td><td>Work Definition Programming</td></tr> <tr> <td>Successor</td><td>Work Definition Programming</td></tr> </table> <p>Résultat:</p> <p>⚠ Successor (Programming) and predecessor (Programming) are the same</p>	Property	Value	Link Type	finishToFinish	Predecessor	Work Definition Programming	Successor	Work Definition Programming													
Property	Value																				
Link Type	finishToFinish																				
Predecessor	Work Definition Programming																				
Successor	Work Definition Programming																				

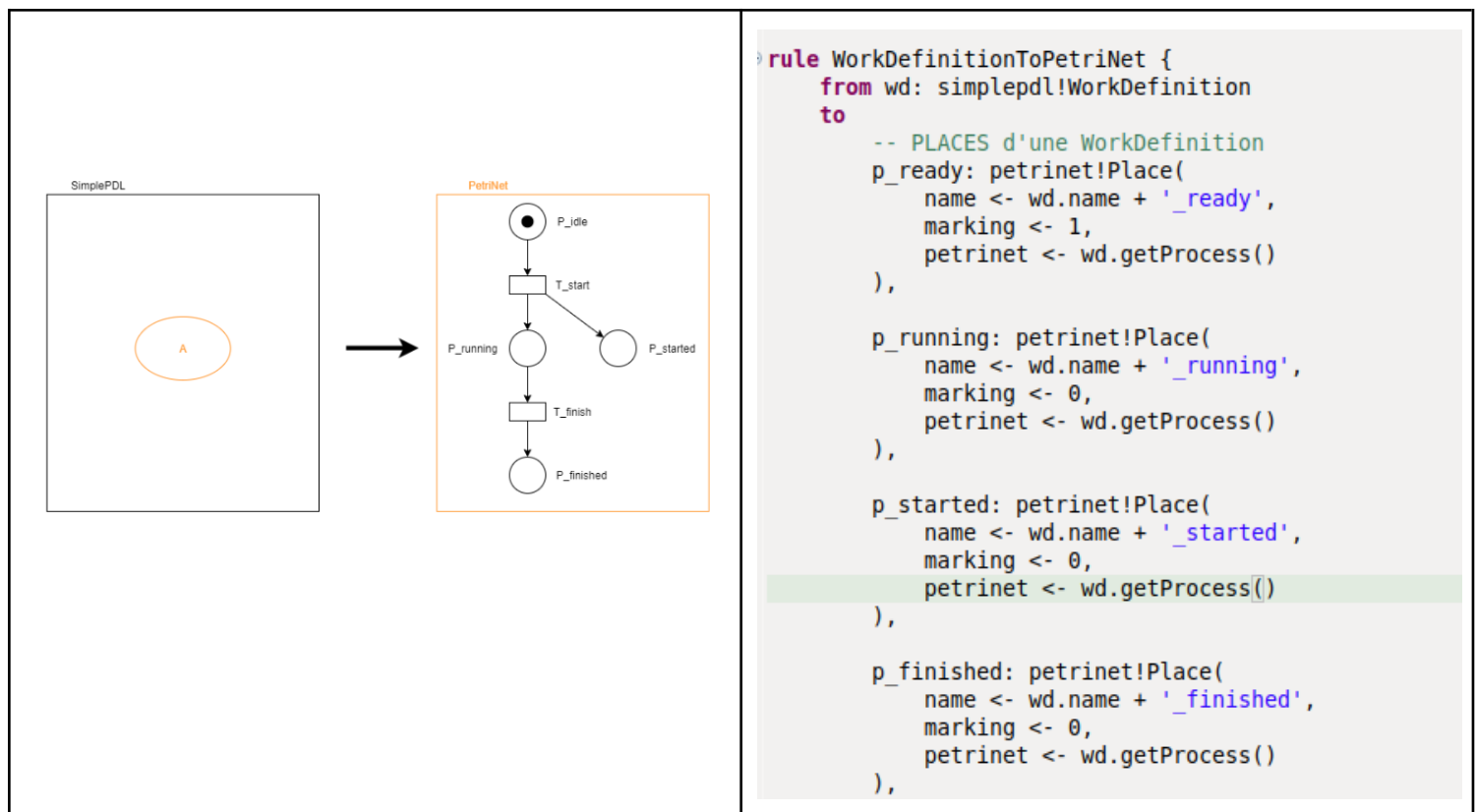
4/ Transformation Modèle à modèle (SimplePDL ver PetriNet)

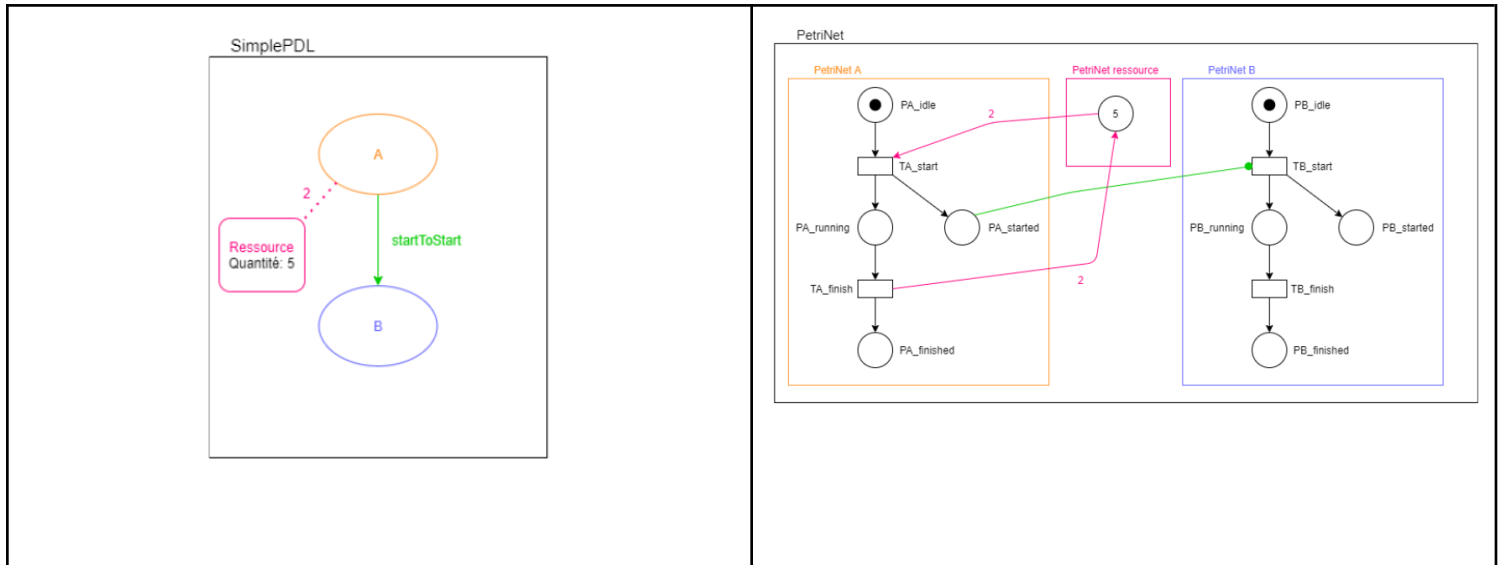
Cette partie s'attarde sur les transformations de modèle à modèle. Passer d'un modèle à l'autre peut être pratique pour transformer les données d'un formalisme à un autre. Ce changement de formalisme permet d'exploiter les données différemment, d'utiliser d'autres outils ou faciliter certaines modifications. Une utilisation pratique consistera à extraire une vue à partir du nouveau modèle.

Pour se faire, différents outils existent. EMF est fourni par défaut avec eclipse et se base sur java pour passer d'un élément à l'autre. ATL quant à lui est un outil à la syntaxe spécialisé pour permettre la transformation en un minimum de ligne.

4.1/ ATL

Il s'agit donc de fournir à chaque élément du modèle *SimplePDL* sa contrepartie *PetriNet*. Prenons l'exemple d'une *WorkDefinition*:





On remarque que le code ATL décuple une *WorkDefinition* en plusieurs état/*Place* possible sur la *PetriNet* en fonction de si elle est en attente, lancée ou terminée. Au départ, une *WorkDefinition* est en attente, c'est pourquoi seule la *Place* *_ready* a un *marking* à 1.

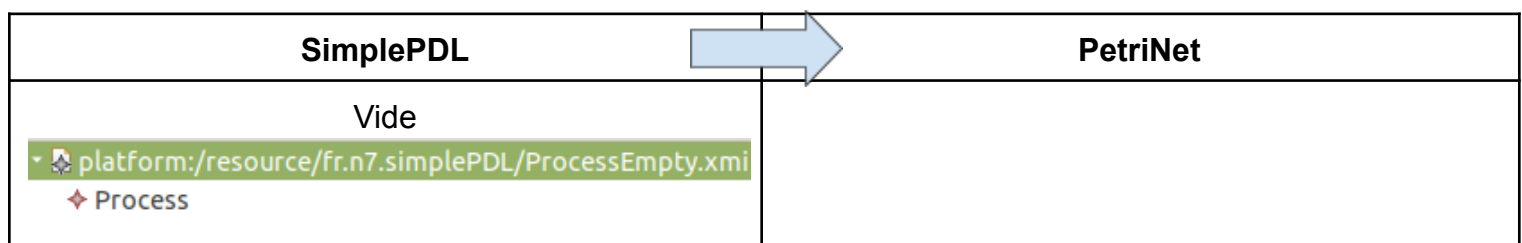
```
-- Traduction d'une Ressource dans le réseau de Petri
rule ResourceToPetriNet {
  from rs: simplepdl!Resource
  to prs: petrinet!Place(
    name <- rs.name + 'resource',
    marking <- rs.nbAvailableResources,
    petrinet <- rs.getProcess()
  )
}
```

Traduction de Need et Ressource

Ici, on convertit une *Resource* en *Place* comme mentionné en 2.1.

4.3/ Validation de la transformation SimplePDL vers PetriNet sur plusieurs exemples

Il est possible de créer une instance dynamique de simplePDL, de lui ajouter à la main des éléments et de générer le petriNet associé.



	<ul style="list-style-type: none"> platform:/resource/fr.n7.petriNet/PetriNetEmpty.xmi Petri Net
<p>Avec resources</p> <ul style="list-style-type: none"> platform:/resource/fr.n7.simplePDL/ProcessLearning.xmi Process Process <ul style="list-style-type: none"> Work Definition Teach Work Definition Learn <ul style="list-style-type: none"> Work Sequence startToStart Resource Teacher Resource Student 3 Work Definition Sleep Work Sequence finishToStart 	<ul style="list-style-type: none"> platform:/resource/fr.n7.petriNet/PetriNetLearning.xmi Petri Net Process <ul style="list-style-type: none"> Place Teach_ready Place Teach_running Place Teach_started Place Teach_finished Transition Teach_start Transition Teach_finish Arc 1 Arc 1 Arc 1 Arc 1 Arc 1 <p>.... voir fichier</p>
<p>Exemple Original</p> <ul style="list-style-type: none"> platform:/resource/fr.n7.simplePDL/ProcessProjet.xmi Process <ul style="list-style-type: none"> Work Definition Conception <ul style="list-style-type: none"> Need 2 Need 2 Work Definition Programmation Work Definition RedactionTests Work Definition RedactionDoc <ul style="list-style-type: none"> Work Sequence startToStart Work Sequence startToStart Work Sequence finishToFinish Work Sequence finishToFinish Work Sequence finishToStart Resource concepteur Resource developpeur Resource machine Resource redacteur Resource testeur 	<ul style="list-style-type: none"> platform:/resource/fr.n7.petriNet/PetriNetProjet.xmi Petri Net <ul style="list-style-type: none"> Place Conception_ready Place Conception_running Place Conception_started Place Conception_finished Transition Conception_start Transition Conception_finish Arc 1 Arc 1 Arc 1 Arc 1 Arc 1 Place Programmation_ready Place Programmation_running Place Programmation_started Place Programmation_finished Transition Programmation_start Transition Programmation_finish <p>.... voir fichier</p>

Nous pouvons également lancer les contraintes *OCL* associées au *PetriNet* pour vérifier que les modèles générés les respectent. Tous les exemples présentés les respectent.

5/ Transformation modèle à texte avec Acceleo

5.1/ ToTina

```
[comment encoding = UTF-8 /]
[module toTina('http://petrinet')]

[template public petriNetToTina(aPetriNet : PetriNet)]
[comment @main/]
[file (aPetriNet.name.concat('.net'), false, 'UTF-8')]
[let places : OrderedSet(Place) = aPetriNet.getPlace() ]
[for (place : Place | places)]
pl [place.name/] ([place.marking/])
[/for]
[/let]

[let transitions: OrderedSet(Transition) = aPetriNet.getTransition() ]
[for (transition : Transition | transitions)]
[transition.transitionToTina()/]
[/for]
[/let]
[/file]
[/template]

[query public getTransition(p: PetriNet) : OrderedSet(Transition) =
    p.petriElements->select(e | e.ocIsTypeOf(Transition))
    ->collect(e | e.ocAsType(Transition))
    ->asOrderedSet()
/]

[query public getPlace(p: PetriNet) : OrderedSet(Place) =
    p.petriElements->select(e | e.ocIsTypeOf(Place))
    ->collect(e | e.ocAsType(Place))
    ->asOrderedSet()
/]

[template public transitionToTina(tr : Transition) post (trim()) ]
tr [tr.name /] [for (arc: Arc | tr.incoming)
    before ( ' ' ) separator ( ' ' ) after ( ' ' )
][arc.source.name /][if (arc.isReadArc)]?[arc.weight/][elseif (arc.weight > 1)]*[arc.weight/][if][for]->
    before ( ' ' ) separator ( ' ' ) after ( ' ' )
][arc.target.name /][if (arc.isReadArc)]?[arc.weight/][elseif (arc.weight > 1)]*[arc.weight/][if][for]
[/template]
```

Pour la syntaxe de Tina, il faut afficher chaque place avec le mot clé “pl” et en parenthèse le marking. Par la suite, il faut transformer les transitions en texte et pour cela on a créé une fonction avec le nom transitionToTina. Celle-ci prend en compte les poids sur les transition et si les arcs sont des ReadArc ou pas.

Après le déploiement des greffons, en cliquant sur une instance dynamique (.xmi) on a pu générer le fichier PetriNet.net :

Après import du fichier .net dans Tina et la création du .ndr on est capable de visualiser le graphe du réseau de petri :

2021/2022

5.2/ ToLTL

```
[template public processToLTL(aProcess : Process)]
[comment @main/]
[file (aProcess.name + '.ltl', false, 'UTF-8')]
[let workdefinitions : OrderedSet(WorkDefinition) = aProcess.getWorkDefinitions() ]
[if (workdefinitions->size() > 0)]
[comment Toutes les activités finissent (T10) /]
[for (wd: WorkDefinition | workdefinitions)
before ('<> (') separator (' /\ ' ) after(');')
][wd.name /]_finished[/for]

[comment Cohérence des états, un seul état par activité (T11) /]
[for (wd : WorkDefinition | workdefinitions)]
['[]' /]([wd.name /]_ready + [wd.name /]_running + [wd.name /]_finished = 1);
[/for]

[comment Une activité démarrée reste démarrée pour toujours (T11) /]
[for (wd: WorkDefinition | workdefinitions)
][[]' /]([wd.name /]_started => [[]' /] ([wd.name/]_started));
[/for]
[/if]
[/let]
[/file]
[/template]

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

Après la transformation modèle à modèle avec ATL et la transformation modèle à texte vers Tina avec Acceleo, on aimerait vérifier avec le model-checker “selt” de la boîte à outil Tina, que certaines propriétés sont valides. sur le réseau de pétri. On a choisi de générer les propriété LTL à partir de SimplePDL, car c’était plus facile qu’à partir du modèle PetriNet.

Nos propriétés LTL vérifient qu’un processus se termine un jour, c’est-à-dire que toutes les activités arrivent à l’état “finished”. De plus, on vérifie qu’une activité ne se trouve pas dans plusieurs états à un instant donné et qu’un processus ayant démarré reste démarré.

Avec le code Acceleo on obtient:

```
1 <> (Conception_finished /\ Programmation_finished /\ RedactionTest_finished);
2
3 [] (Conception_ready + Conception_running + Conception_finished = 1);
4 [] (Programmation_ready + Programmation_running + Programmation_finished = 1);
5 [] (RedactionTest_ready + RedactionTest_running + RedactionTest_finished = 1);
6
7 [] (Conception_started => [] (Conception_started));
8 [] (Programmation_started => [] (Programmation_started));
9 [] (RedactionTest_started => [] (RedactionTest_started));
```

Avec l’outil Selt on a pu vérifier les propriétés :

```
n7student@n7app01:~/iidm$ tina Process1.net Process1.ktz

# net {}, 14 places, 6 transitions, 21 arcs                                     #
# bounded, not live, not reversible                                           #
# abstraction      count      props      psets      dead      live #
#   states         15         14         15         1         1 #
# transitions      24          6          6          0          0 #
n7student@n7app01:~/iidm$ selt -p -S Process1.scn Process1.ktz -prelude Process1
.ltl
Selt version 3.7.0 -- 01/19/22 -- LAAS/CNRS
ktz loaded, 15 states, 24 transitions
0.003s

- source Process1.ltl;
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
TRUE
0.006s

- 
```

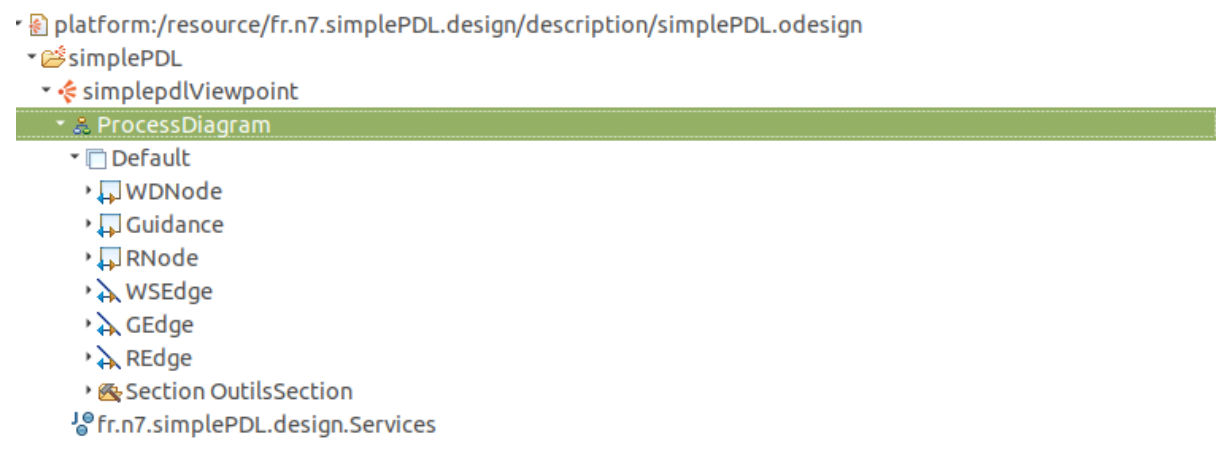
6/ Syntaxe graphique avec Sirius

le modèle Sirius (fichier o.design)

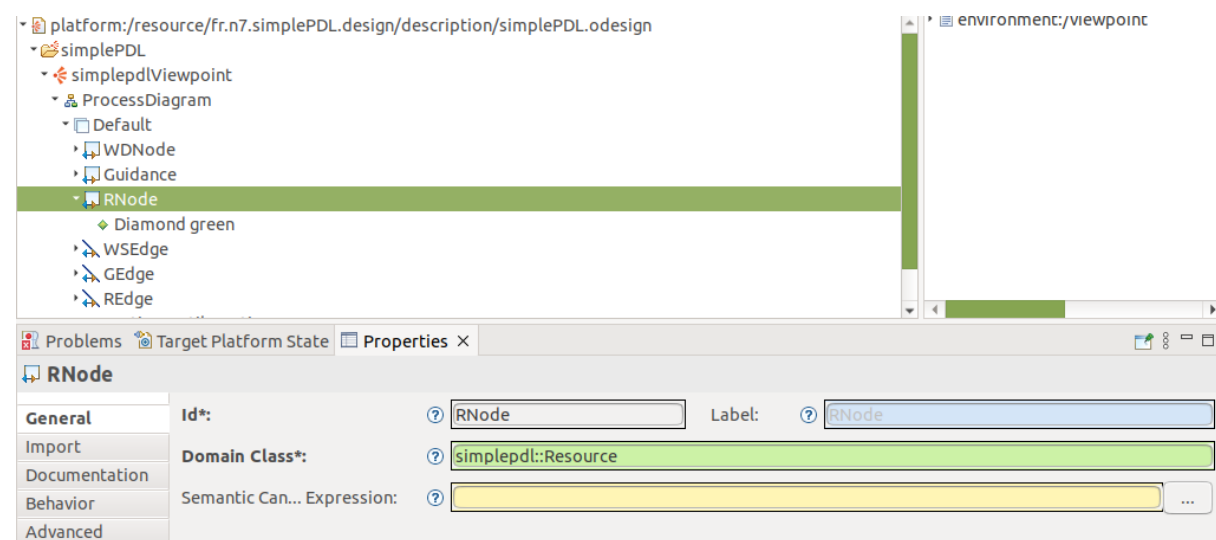
Dans ce fichier, on ajoute la représentation graphique des différents éléments d'un Processus. On y retrouve des *Nodes* qui représentent des *Workdefinitions*, *Guidance* et ressources.

On a des éléments qui mettent en relation des éléments comme la *Worksequence* et la classe *Need* qui met en relation une *Ressource* et une *Workdefinition* avec le nombre utilisé..

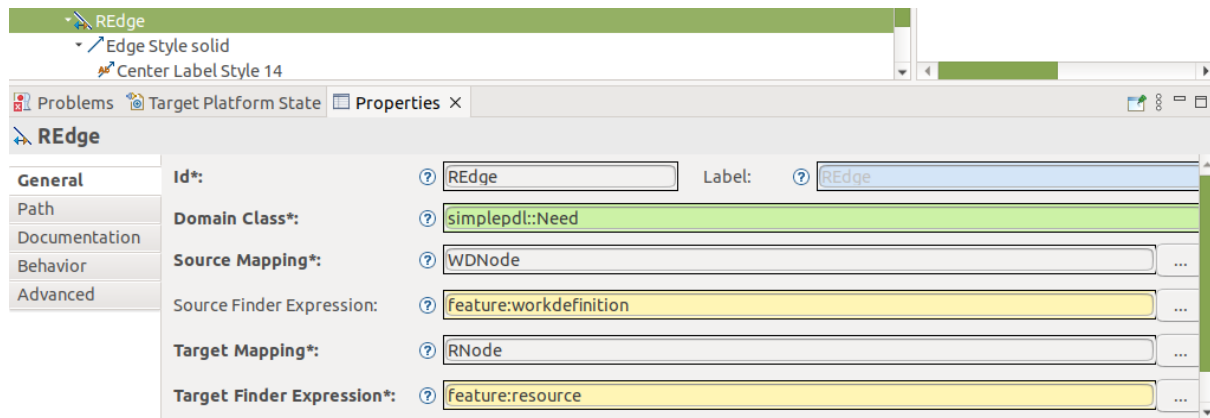
On a une relation qui met en relation une guidance avec les *WorkDefinitions*.



Ajout du Node Ressources



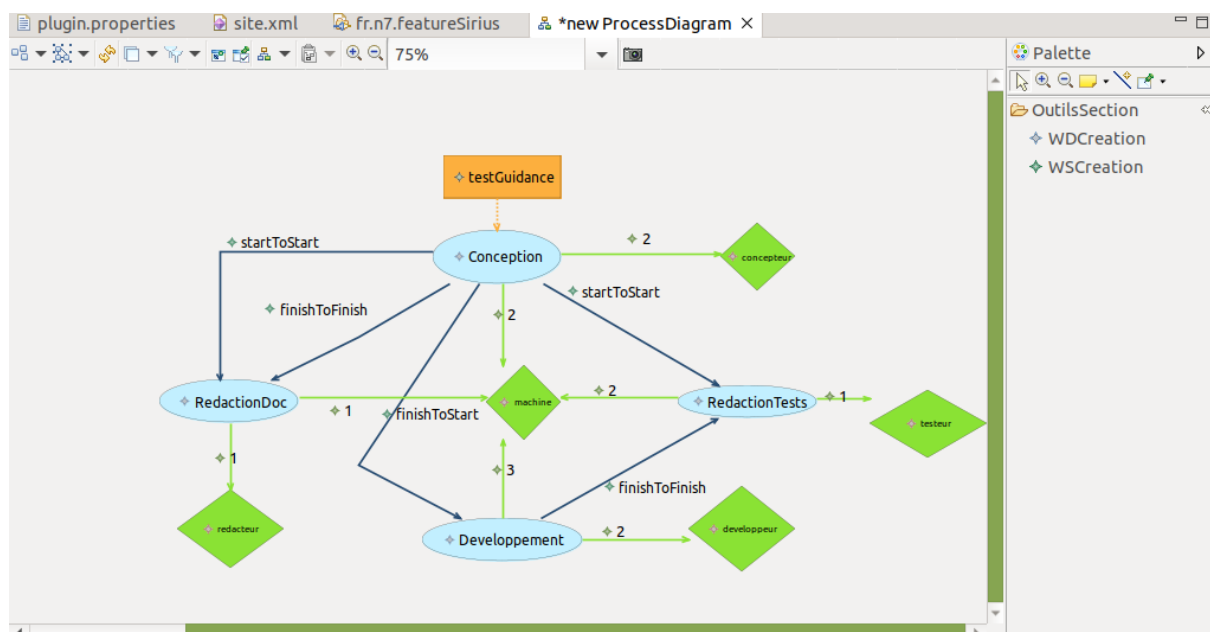
Ajout du lien entre Workdefinition et Ressources (classe Need):



Ajout de la palette, afin de pouvoir créer des éléments du modèle simplePDL graphiquement la palette:

- ▼ Section OutilsSection
 - ▼ Node Creation WDCreation
 - Node Creation Variable simplepdl::Process
 - Container View Variable ProcessDiagram
 - Begin
 - ▼ Edge Creation WSCreation
 - Source Edge Creation Variable simplepdl::WorkDefinition
 - Target Edge Creation Variable simplepdl::WorkDefinition
 - Source Edge View Creation Variable WDNode
 - Target Edge View Creation Variable WDNode
 - Begin

On a créé une *section* avec le nom *OutilsSection* qui permet de créer une *WorkDefinition* et une *WorkSequence* (pas fonctionnel).



7/ Syntaxe textuelle avec xtext

Nous avons défini une syntaxe textuelle qui permet de décrire notre méta modèle simpleddl

```
1 grammar fr.n7.PDL1 with org.eclipse.xtext.common.Terminals
2
3 generate pDL1 "http://www.n7.fr/PDL1"
4
5
6 Process : 'process' name=ID '{'
7     processElements+=ProcessElement*
8 '}' ;
9
10 ProcessElement : WorkDefinition | WorkSequence | Guidance | Resource;
11
12 enum WorkSequenceType : startToStart = 'startToStart'
13     | finishToStart = 'finishToStart'
14     | startToFinish = 'startToFinish'
15     | finishToFinish = 'finishToFinish';
16
17 WorkDefinition : 'task' name=ID '{'
18     (need+=Need)?
19 '}' ;
20
21 Guidance : 'note' text=STRING ('for' elements+=[ProcessElement] ( "," elements+=[ProcessElement])* );
22
23 WorkSequence : 'dep'
24     linkType=WorkSequenceType
25     'from' predecessor=[WorkDefinition]
26     'to' successor=[WorkDefinition];
27
28 Need : 'need' nbResources=INT 'of' resource=[Resource];
29
30 Resource : 'create' nbAvailableResources=INT 'of' name=ID;
31
```

Pour comprendre la syntaxe qu'on a choisi :

Process -> process {} qui contient 0 ou plus :

WorkDefinition -> task ID {}

WorkSequence -> dep WorkSequenceType from WD to WD

Resource -> create int of ID

Une WorkDefinition qui contient 0 ou plusieurs besoin:
need INT of Resource

le fichier de test :

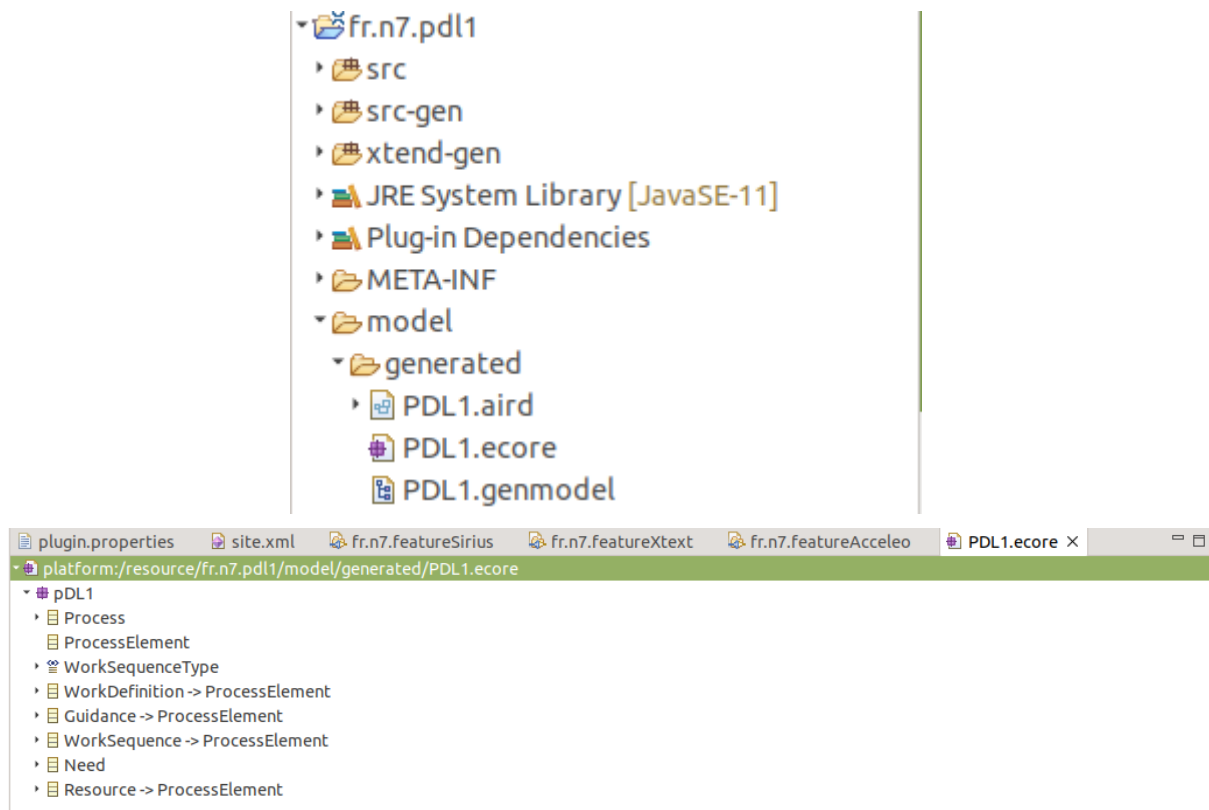
```
process Restaurant {
    create 4 of chef
    create 7 of serveur
    task cuisiner {
        need 3 of chef
    }

    task service {
        need 5 of serveur
    }

    dep finishToStart from cuisiner to service

    note "comment cuisiner un ragout" for cuisiner
    note "gérer un restaurant" for cuisiner,service
}
```

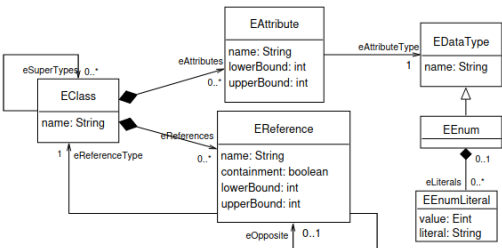
le modèle générer avec xtext :



8/ Conclusion

Ce projet aura nécessité de nombreuses batailles avec eclipse qui resteront dans les annales. Mais c'est dans la difficulté que nous avons cherché à repousser les limites de nos connaissances. Nous sommes fières d'avoir pu effectuer la majorité des transformations requises et d'avoir au moins compris les autres.

Annexe 0 - Glossaire des technologies utilisées

Nom	Description/Utilité	Exemples
Tina	Syntaxe concrète de représentation de modèle	<pre> pl Hiver (1) tr h2p Hiver -> Printemps tr p2e Printemps -> Ete tr e2a Ete -> Automne tr a2h Automne -> Hiver </pre> <p>PetriNET Saison en syntaxe Tina</p>
Ecore	<p>Langage de méta-modélisation Méta-modèle des méta-modèle</p> <p>EMF est inclus dans Eclipse et permet d'engendrer :</p> <ul style="list-style-type: none"> - Le modèle Java correspondant : - Un schéma XML correspondant et les opérations de sérialisation/désérialisation associées. - Un éditeur arborescent pour saisir un modèle. 	
XText	<p>Définir une syntaxe concrète textuelle pour un méta-modèle Intérêt: utiliser tous les outils classiques pour les textes (Ctrl+F, replace ...)</p>	<p>Description de la syntaxe concrète:</p> <p>Un exemple de syntaxe concrète textuelle :</p> <pre> process ExampleProcessus { wd RedactionDoc wd Conception wd Programmation wd RedactionTests ws Conception f2f RedactionDoc ws Conception s2s RedactionDoc ws Conception f2s Programmation ws Conception s2s RedactionTests ws Programmation f2f RedactionTests } grammar fr.n7.PDL1 with org.eclipse.xtext.common.Terminals generate pDL1 "http://www.n7.fr/PDL1" Process : 'process' name=ID '{' processElements+=ProcessElement* '}' ; ProcessElement : WorkDefinition WorkSequence Guidance ; WorkDefinition : 'wd' name=ID ; WorkSequence : 'ws' linkType=WorkSequenceType 'from' predecessor=[WorkDefinition] 'to' successor=[WorkDefinition] ; enum WorkSequenceType : start2start = 's2s' finish2start = 'f2s' start2finish = 's2f' finish2finish = 'f2f' ; Guidance : 'note' texte=STRING ; </pre>

<p>ATL</p>	<p>Modèle à modèle Intérêt:</p> <ul style="list-style-type: none"> - Transformer des données d'un formalisme à un autre - Modifier un modèle - Extraire une vue d'un modèle 	<pre> module SimplePDL2PetriNet; create OUT: PetriNet from IN: SimplePDL; -- Obtenir le processus qui contient ce process element. -- Remarque: Ce helper ne serait pas utile si une référence opposite -- avait été placée entre Process et ProcessElement helper context SimplePDL!ProcessElement def: getProcess(): SimplePDL!Process = SimplePDL!Process.allInstances() -> select(p p.processElements-> includes(self)) -> asSequence()-> first(); -- Traduire un Process en un PetriNet de même nom rule Process2PetriNet { from p: SimplePDL!Process to pn: PetriNet!PetriNet (name <- p.name) } -- Traduire une WorkDefinition en un motif sur le réseau de Petri rule WorkDefinition2PetriNet { from wd: SimplePDL!WorkDefinition to -- PLACES d'une WorkDefinition p_ready: PetriNet!Place (name <- wd.name + '_ready', marking <- 1, net <- wd.getProcess()) -- TRANSITIONS du RdP d'une WorkDefinition t_start: PetriNet!Transition (name <- wd.name + '_start', net <- wd. getProcess()) -- ARCS du RdP d'une WorkDefinition a_nsed2s: PetriNet!Arc (kind <- #normal , weight <- 1 , source <- p_ready , target <- t_start , net <- wd.getProcess()) } </pre>
<p>Acceleo</p>	<p>Modèle à texte</p> <p>Engendrer un texte à partir d'un modèle</p> <p>Intérêt:</p> <ul style="list-style-type: none"> - Engendrer du code - Engendrer de la documentation 	<p>Sérialisation d'un modèle SimplePDL</p> <pre> [comment encoding = UTF-8 //] [module toPDL('http://simplepdl') //] [comment Generation de la syntaxe PDL1 à partir d'un modèle de processus/] [template public toPDL(proc : Process)] [comment @main/] [file (proc.name.concat('.pdl1'), false, 'UTF-8')] process [proc.name] { for (wd : WorkDefinition proc.processElements->getWDs()) wd [wd.name/] [/for] for (ws : WorkSequence proc.processElements->getWSs()) ws [ws.predecessor.name/] [ws.getWSType()/] [ws.successor.name/] [/for] [/file] [/template] [query public getWDs(elements : OrderedSet(ProcessElement)) : OrderedSet(WorkDefinition) = elements->select(e e.oclsTypeOf(WorkDefinition)) ->collect(e e.oclsType(WorkDefinition)) ->asOrderedSet() /] [query public getWSs(elements : OrderedSet(ProcessElement)) : OrderedSet(WorkSequence) = elements->select(e e.oclsTypeOf(WorkSequence)) ->collect(e e.oclsType(WorkSequence)) ->asOrderedSet() /] [template public getWSType(ws : WorkSequence)] [if (ws.linkType = WorkSequenceType::startToStart)] s2s[elseif (ws.linkType = WorkSequenceType::startToFinish)] s2f[elseif (ws.linkType = WorkSequenceType::finishToStart)] f2s[elseif (ws.linkType = WorkSequenceType::finishToFinish)] f2f[/if] [/template] </pre>

		<pre> comment encoding = UTF-8 /] [module toHTML('http://simplepdl')] [template public processToHTML(aProcess : Process)] [comment @main/] [file (aProcess.name + '.html', false, 'UTF-8')] <head><title>[aProcess.name/]</title></head> <body> <h1>Process "[aProcess.name/]"</h1> <h2>Work definitions</h2> [let wds : OrderedSet(WorkDefinition) = aProcess.getWds()] [if (wds->size() > 0)] [for (wd : WorkDefinition wds)] [wd.toHTML()/] [/for] [else] None. [/if] [/let] </body> [/file] [/template] </pre>
Sirius	<p>syntaxe graphique</p> <p>Sirius, on écrit le méta modèle et on fait une association graphique des éléments</p>	