Chaîne de Vérification de Modèles de Processus

Rapport Mini-Projet

2ème Année, Systèmes Logiciels

Mehdi Sensali
Younes Saoudi

2020 - 2021

Contents

Introduction

Ce mini-projet consiste à produire une chaîne de vérification de modèles de processus SimplePDL dans le but de vérifier leur cohérence, en particulier pour savoir si le processus décrit peut se terminer ou non. Pour répondre à cette question, nous utilisons les outils de model-checking définis sur les réseaux de Petri au travers de la boîte à outils Tina. Il nous faudra donc traduire un modèle de processus en un réseau de Petri.

Tâches Réalisées

2.1 Métamodèle SimplePDL

2.1.1 Métamodèle Ecore

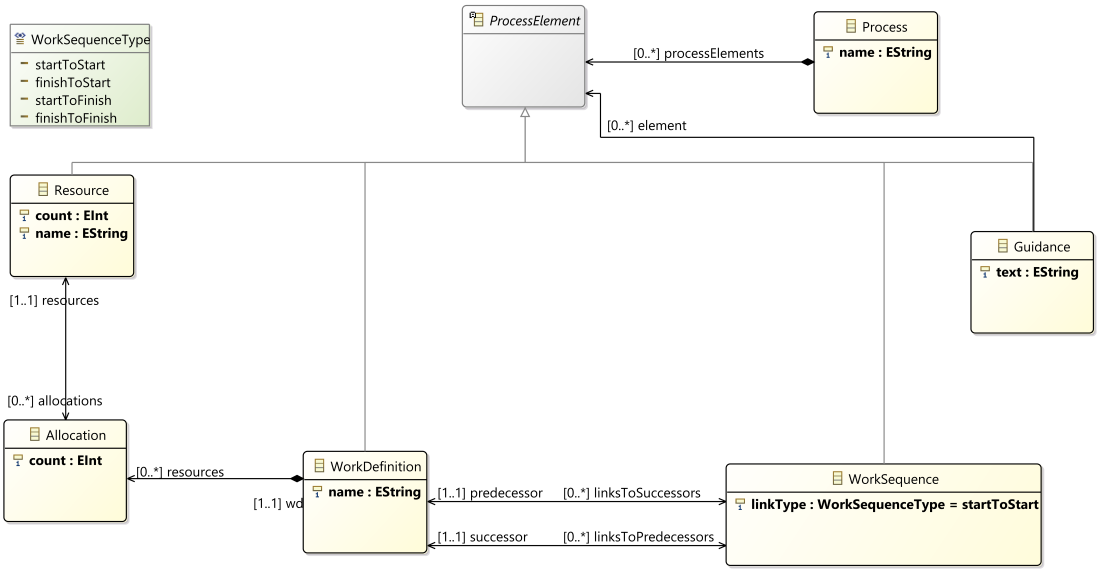

Figure 2.1: Métamodèle SimplePDL

Chaque `WorkDefinition` peut allouer un nombre `count` d'une ressource `Resource`. Nous remarquons que plusieurs propriétés obligatoires ne sont pas capturées par le métamodèle, ce qui nécessite une complétion par une **sémantique statique.**

2.1.2 Contraintes OCL

Les contraintes qu'il faut que chaque métamodèle SimplePDL vérifie sont les suivantes:

- Chaque `Process`, `WorkDefinition` et `Resource` doivent avoir un nom valide
- Les `Successor` et `Predecessor` d'une même `WorkSequence` doivent être distincts et appartenir au même `Process`
- Chaque WorkDefinition doit avoir un nom unique
- Une `Allocation` ne peut pas allouer un nombre supérieur au nombre de ressources disponibles.

- La somme des allocations doit être inférieure ou égale au nombre total de ressources.

2.2 Métamodèle PetriNet

2.2.1 Métamodèle Ecore

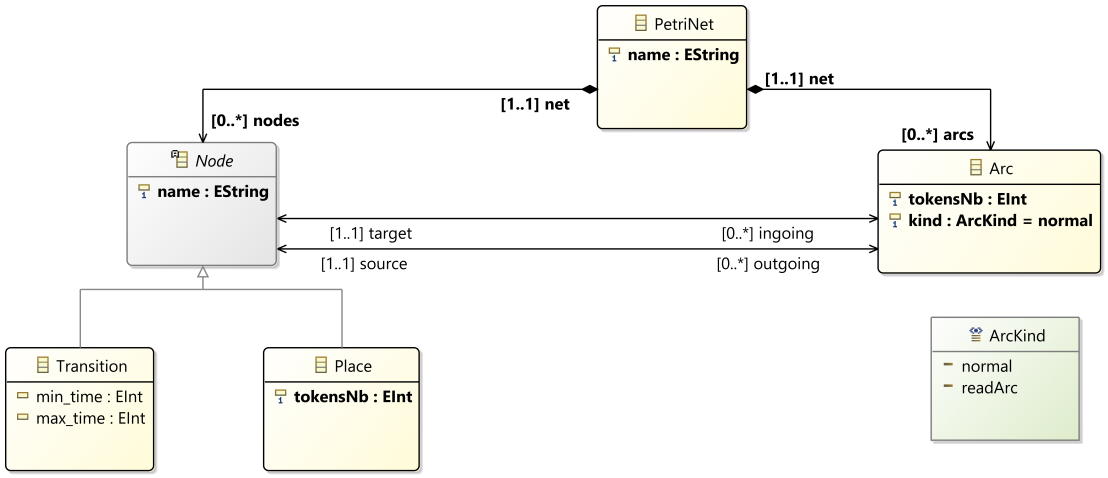

Figure 2.2: Métamodèle PetriNet

2.2.2 Contraintes OCL

Les contraintes qu'il faut que chaque métamodèle PetriNet vérifie sont les suivantes:

- Chaque `PetriNet` et `Place` doivent avoir un nom valide
- Les `Source` et `Target` d'un même `Arc` doivent appartenir au même `PetriNet`
- Chaque `Node` doit avoir un nom unique.
- Une `Place` doit avoir un nombre de jetons positif ou nul.
- Un Arc doit avoir un nombre de jetons supérieur à `1`.
- Les `Read Arcs` doivent avoir pour source une `Place` et pour cible une `Transition`
- Si la `Source` d'un `Arc` est de type `Place` alors `Target` est de type `Transition` et vice-versa.

2.3 Syntaxe Textuelle de SimplePDL : `XText`

Nous avons implanté une syntaxe concrète textuelle de SimplePDL avec XText. Prenons l'exemple du métamodèle suivant:

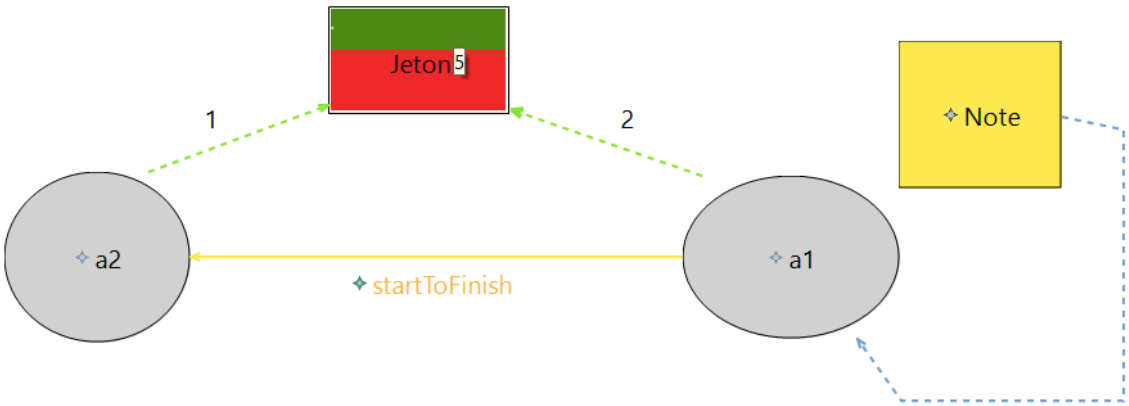

Figure 2.3: Exemple Métamodèle créé avec Sirius

Avec la syntaxe que nous avons créée, ce métamodèle est équivalent à :

```
process exemple {
   rs Jeton counting 5
   wd a1 uses (allocation of 2 Jeton)
   wd a2 uses (allocation of 1 Jeton)
   ws s2f from a1 to a2
   guidance Note for (a1)
}
```

Voir script `XPDL.xtext`

2.4 Développement d'un Editeur Graphique SimplePDL

Comme vous l'aviez remarqué sur la figure 2.3, nous avons créé un éditeur graphique simplePDL pour la saisi graphique de processus ainsi que de ressources.

Pour l'exemple du modèle de procédé donné sur le sujet du mini-projet, cela donne les résultats suivants qui contiennent l'addition de `Guidance` et de `Resource` (9 Jetons):

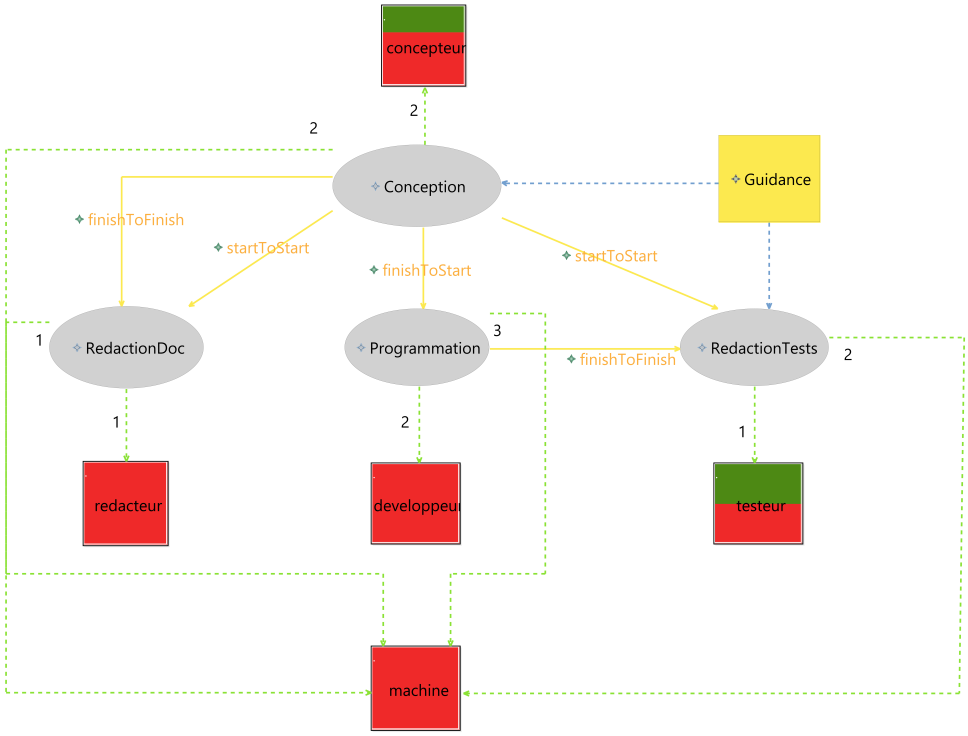

Figure 2.4: `developpement.simplepdl` créé avec Sirius

Remarque: En utilisant la syntaxe textuelle précédente, le modèle est le suivant:

```
process developpement {
    rs concepteur counting 3
    rs developpeur counting 2
    rs machine counting 4
    rs redacteur counting 1
    rs testeur counting 2
    wd Conception uses (allocation of 2 concepteur, allocation of 2 machine)
    wd RedactionDoc uses (allocation of 1 machine, allocation of 1 redacteur)
    wd Programmation uses (allocation of 2 developpeur, allocation of 3 machine)
    wd RedactionTests uses (allocation of 2 machine, allocation of 1 testeur)
    ws f2f from Conception to RedactionDoc
    ws s2s from Conception to RedactionDoc
    ws f2s from Conception to Programmation
    ws s2s from Conception to RedactionTests
    ws f2f from RedactionTests to Programmation
    guidance Guidance for (Conception, RedactionTests)
}
```

L'editeur contient des palettes d'outils pour la création de `WorkDefinitions`, `WorkSequences`, `Guidances` et `Resources`:

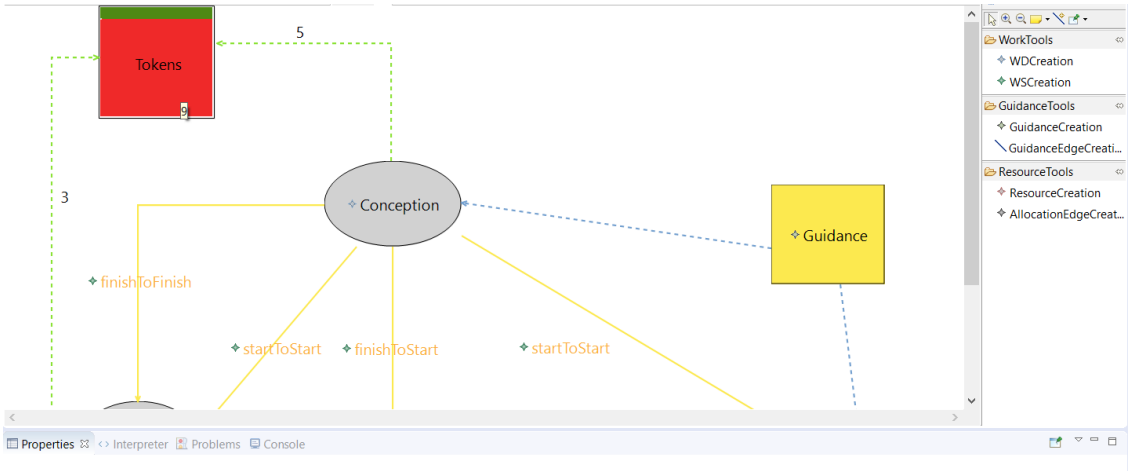

Figure 2.5: L'Editeur Graphique SimplePDL

2.5 Transformation SimplePDL vers PetriNet

Le principe de la transformation est simple:

- `Process` devient `PetriNet` et leurs attributs `name` sont égaux.
- `WorkDefinition` devient soit une `Place notStarted, started, inProgress` ou `Finished`, soit une `Transition start` ou `finish`, entre lesquelles il peut y avoir un `Arc` de type `ArcKind::normal`
- `WorkSequence` devient un `Arc` de type `ArcKind::readArc` entre les `Places` et `Transitions` correspondantes.

Par exemple: `ws s2s from a1 to a2` signifie qu'il y aura un `Read Arc` de la place `a1_started` à la transition `a2_start`

- `Resource` devient une `Place` avec des `Arcs` de type `ArcKind::normal` sortant vers les `Transitions` correspondant aux `WorkDefinitions` qui allouent cette ressource. Le nombre de ressources alloué correspond au **tokensNb** de ces `Arcs`

2.5.1 EMF/Java

En s'inspirant des fichiers `SimplePDLCreator.java` et `SimplePDLManipulator.java` du TP4, nous avons implanté une transformation de SimplePDL vers PetriNet avec Java.

Voir script `SimplePDL2PetriNet.java`

2.5.2 ATL

En s'inspirant du Listing 1 du TP 8 `SimplePDL2PetriNet.atl`, nous avons implanté une transformation de SimplePDL vers PetriNet avec ATL.

Voir script `SimplePDL2PetriNet.atl`

2.6 Validation de la Transformation avec des tests

Nous avons effectués des tests en transformant des modèles SimplePDL, comme `developpement` qui est présent dans les de TP ainsi que celui du mini-projet, en Petri Network en utilisant **`SimplePDL2PetriNet.atl`** et **`SimplePDL2PetriNet.java`**.

Voir dossier `exemples`

2.7 Transformation PetriNet vers Tina : `Acceleo`

Afin de convertir les métamodèles PetriNet vers Tina avec `Acceleo`, nous nous sommes inspirés des exercices 1 à 3 du TP 5 et avons obtenu les résultats suivants, que ça soit avec un modèle PetriNet normal ou transformé depuis SimplePDL par Acceleo/EMF-JAVA:

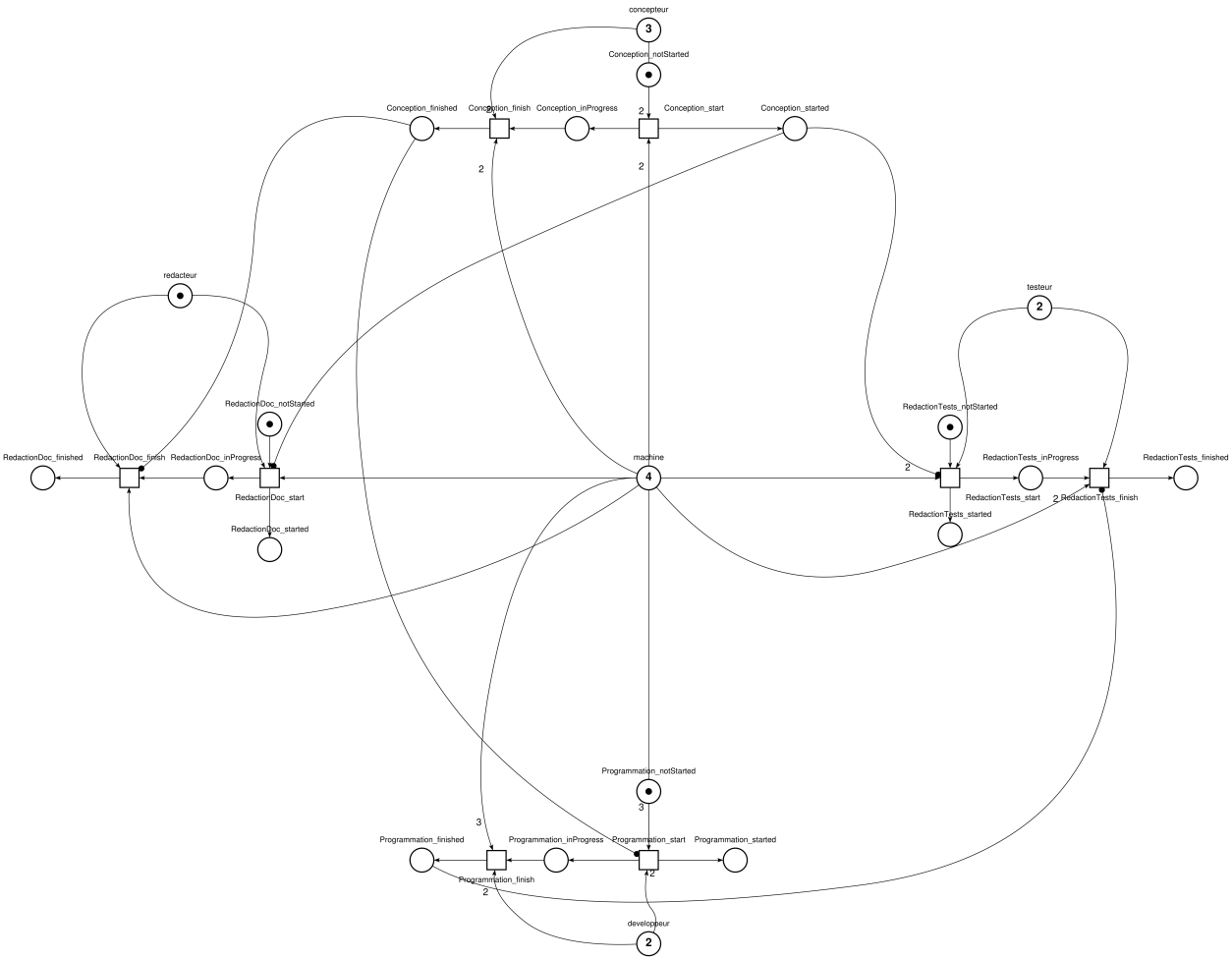

Figure 2.6: Syntaxe Graphique de `developpement.petrinet` Transformé vers TINA

```
net developpement {
     pl Conception_notStarted (1)
     pl Conception_started (0)
     pl Conception_inProgress (0)
     pl Conception_finished (0)
     pl RedactionDoc_notStarted (1)
     pl RedactionDoc_started (0)
     pl RedactionDoc_inProgress (0)
     pl RedactionDoc_finished (0)
     pl Programmation_notStarted (1)
     pl Programmation_started (0)
     pl Programmation_inProgress (0)
     pl Programmation_finished (0)
     pl RedactionTests_notStarted (1)
     pl RedactionTests_started (0)
     pl RedactionTests_inProgress (0)
     pl RedactionTests_finished (0)
     pl concepteur (3)
     pl developpeur (2)
     pl machine (4)
     pl redacteur (1)
     pl testeur (2)
     tr Conception_start Conception_notStarted concepteur*2 machine*2 -> Conception_started Conception_inProgress
     tr Conception_finish Conception_inProgress concepteur*2 machine*2 -> Conception_finished
     tr RedactionDoc_start RedactionDoc_notStarted Conception_started?1 machine redacteur -> RedactionDoc_started
RedactionDoc_inProgress
     tr RedactionDoc_finish RedactionDoc_inProgress Conception_finished?1 machine redacteur -> RedactionDoc_finished
     tr Programmation_start Programmation_notStarted Conception_finished?1 developpeur*2 machine*3 -> Programmation_started
Programmation_inProgress
     tr Programmation_finish Programmation_inProgress developpeur*2 machine*3 -> Programmation_finished
     tr RedactionTests_start RedactionTests_notStarted Conception_started?1 machine*2 testeur -> RedactionTests_started
RedactionTests_inProgress
     tr RedactionTests_finish RedactionTests_inProgress Programmation_finished?1 machine*2 testeur -> RedactionTests_finished
}
```

Figure 2.6: Syntaxe Textuelle de `developpement.petrinet` Transformé vers TINA

Voir script `toTINA.mtl`

2.8 Propriétés LTL

Nous avons créé un plugin Acceleo `fr.n7.simplepdl.toLTL` qui permet de générer le fichier `LTL` des conditions à vérifier pour chaque modèle simplePDL.

Voir script `toLTL.mtl`

2.8.1 Terminaison d'un processus

La terminaison d'un processus est assurée par les expressions:

- □ (finished ⇒ dead)
- □ ◊ dead
- □ (dead ⇒ finished)
- ¬ ◊ finished

:

2.8.2 Validation de la transformation : Invariants SimplePDL

Les invariant à vérifier sont:

- Une activité finit toujours par cesser d'évoluer:

$$\Box \Diamond \text{dead}$$

- Une activité n'évoluant plus est terminée:

$$\Box (\text{dead} \Rightarrow \text{finished})$$

- Une activité commencée ne peut pas redémarrer:

$$\Box (\text{started} \Rightarrow \neg \Diamond \text{notStarted})$$

- Une activité finie ne peut pas être en cours:

$$\Box (\text{finished} \Rightarrow \neg \Diamond \text{inProgress})$$

- Une activité finie ne peut pas redémarrer:

$$\Box (\text{finished} \Rightarrow \neg \Diamond \text{notStarted})$$

- Une activité terminée signifie qu'elle a été d'abord commencée:

$$\Box (\text{finished} \Rightarrow \text{started})$$

- Chaque activité est soit non commencée soit en cours soit terminée:

```
    [for(wd :  WorkDefinition | aProcess.processElements->getWDs())] ['[ ]'/]
([wd.name/]_notStarted + [wd.name/]_inProgress + [wd.name/]_finished = 1);[/for]
```

Un exemple d'un "invariant qui n'en est pas un":

- Une activité n'évoluant plus est forcément terminée:

$$\Box (\text{dead} \Rightarrow \text{finished})$$

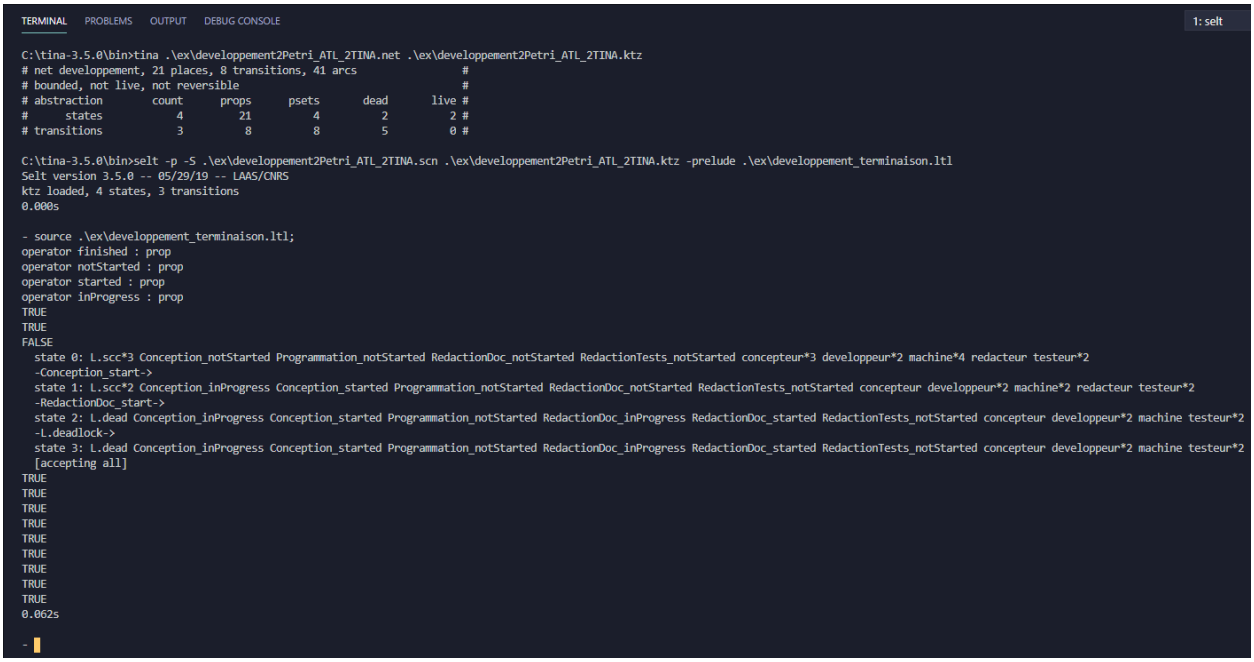

Figure 2.7: Résultats de l'utilisation de l'outil `selt` sur `developpement.net` Transformé vers TINA depuis PetriNet

Nous remarquons que tous les invariants qui sont réels passent (le message TRUE s'affiche en console) sauf pour la propriété "Une activité n'évoluant plus est forcément terminée" qui n'est clairement pas vraie pour tous les modèles de processus, `developpement.net` en est la preuve.
De plus, le fichier .aut nous montre un contre-exemple de cet invariant:

```
des(0,3,4)
(0, "Conception_start", 1)
(1, "RedactionDoc_start", 2)
(1, "RedactionTests_start", 3)
```

Conclusion

En guise de conclusion, ce projet nous a permis de manipuler les méta-modèles et de nous familiariser avec les outils de développement employés. En s'inspirant du méta-modèle `ECORE PetriNet`. Ensuite nous avons pu exprimer des contraintes sur ces derniers par le biais du langage `OCL`. Par l'outil `Xtext`, nous avons généré et défini des syntaxes concrètes textuelles permettant de définir des modèles de processus `SimplePDL`. Grâce aux outils `EMF/Java` et `ATL`, nous avons effectué des transformations modèle à modèle de `SimplePDL` vers `PetriNet`. Par l'outil `accéléo`, nous avons réussi à définir une transformation `Petrinet` vers `Tina` ainsi que d'engendrer des propriétés `LTL` propres aux processus.

Annexe

4.1 XPDL.xtext

```
grammar fr.n7.XPDL with org.eclipse.xtext.common.Terminals
generate xPDL1 "http://www.n7.fr/XPDL"

Process :
  'process' name=ID '{'
      processElements+=ProcessElement*
  '}' ;

ProcessElement returns ProcessElement:
  WorkDefinition | WorkSequence | Guidance | Resource;

WorkDefinition :
  'wd' name=ID ( 'uses' '(' resources+=Allocation ( ',' resources+=Allocation)* ')' )?;

WorkSequence :
  'ws' linkType=WorkSequenceType
    'from' predecessor=[WorkDefinition]
    'to' successor=[WorkDefinition] ;

Guidance :
  'guidance' text=ID  ( 'for' '(' element+=[ProcessElement|ID] ( "," element+=[
    ProcessElement|ID])* ')' )?;

Resource :
    'rs' name=ID
    'counting' count=INT
    ;

Allocation returns Allocation :
    'allocation' 'of' count=INT
    resources=[Resource]
  ;
enum WorkSequenceType returns WorkSequenceType:
        startToStart = 's2s' | finishToStart = 'f2s' | startToFinish = 's2f' |
    finishToFinish = 'f2f';
```

4.2 SimplePDL2PetriNet.java

```java
package simplepdl.util;

import java.util.*;
import java.util.Collections;
import org.eclipse.emf.common.util.URI;
import org.eclipse.emf.ecore.resource.Resource;
import org.eclipse.emf.ecore.resource.ResourceSet;
import org.eclipse.emf.ecore.resource.impl.ResourceSetImpl;
import org.eclipse.emf.ecore.xmi.impl.XMIResourceFactoryImpl;
import petriNet.*;
import simplepdl.*;

public class SimplePDL2PetriNet {

    static PetriNet PetriNetwork;
    static PetriNetFactory PetriNetFactory;

    static Map<String, Place> resources = new HashMap<String, Place>();
    static Map<String, Place> startedPlaces = new HashMap<String, Place>();
    static Map<String, Place> finishedPlaces = new HashMap<String, Place>();
    static Map<String, Transition> startTransitions = new HashMap<String, Transition>();
    static Map<String, Transition> finishTransitions = new HashMap<String, Transition>();


    public static void convertWorkDefinition(WorkDefinition wd)
    {
      //PLACES
        Place p_notStarted = PetriNetFactory.createPlace();
        p_notStarted.setName(wd.getName() + "_notStarted");
        p_notStarted.setTokensNb(1);
        p_notStarted.setNet(PetriNetwork);
        PetriNetwork.getNodes().add(p_notStarted);

        Place p_started = PetriNetFactory.createPlace();
        p_started.setName(wd.getName() + "_started");
        p_started.setTokensNb(0);
        p_started.setNet(PetriNetwork);
        PetriNetwork.getNodes().add(p_started);

        Place p_inProgress = PetriNetFactory.createPlace();
        p_inProgress.setName(wd.getName() + "_inProgress");
        p_inProgress.setTokensNb(0);
        p_inProgress.setNet(PetriNetwork);
        PetriNetwork.getNodes().add(p_inProgress);

        Place p_finished = PetriNetFactory.createPlace();
        p_finished.setName(wd.getName() + "_finished");
        p_finished.setTokensNb(0);
        p_finished.setNet(PetriNetwork);
        PetriNetwork.getNodes().add(p_finished);

        //TRANSITIONS
        Transition t_start = PetriNetFactory.createTransition();
        t_start.setName(wd.getName() + "_start");
        t_start.setNet(PetriNetwork);
        PetriNetwork.getNodes().add(t_start);

        Transition t_finish = PetriNetFactory.createTransition();
        t_finish.setName(wd.getName() + "_finish");
        t_finish.setNet(PetriNetwork);
        PetriNetwork.getNodes().add(t_finish);

```

```
63          //ARCS
64          Arc arc_notStarted2start = PetriNetFactory.createArc();
65          arc_notStarted2start.setSource(p_notStarted);
66          arc_notStarted2start.setTarget(t_start);
67          arc_notStarted2start.setTokensNb(1);
68          arc_notStarted2start.setNet(PetriNetwork);
69          PetriNetwork.getArcs().add(arc_notStarted2start);
70
71          Arc arc_start2started = PetriNetFactory.createArc();
72          arc_start2started.setSource(t_start);
73          arc_start2started.setTarget(p_started);
74          arc_start2started.setTokensNb(1);
75          arc_start2started.setNet(PetriNetwork);
76          PetriNetwork.getArcs().add(arc_start2started);
77
78          Arc arc_start2inProgress = PetriNetFactory.createArc();
79          arc_start2inProgress.setSource(t_start);
80          arc_start2inProgress.setTarget(p_inProgress);
81          arc_start2inProgress.setTokensNb(1);
82          arc_start2inProgress.setNet(PetriNetwork);
83          PetriNetwork.getArcs().add(arc_start2inProgress);
84
85          Arc arc_inProgress2finish = PetriNetFactory.createArc();
86          arc_inProgress2finish.setSource(p_inProgress);
87          arc_inProgress2finish.setTarget(t_finish);
88          arc_inProgress2finish.setTokensNb(1);
89          arc_inProgress2finish.setNet(PetriNetwork);
90          PetriNetwork.getArcs().add(arc_inProgress2finish);
91
92          Arc arc_finish2finished = PetriNetFactory.createArc();
93          arc_finish2finished.setSource(t_finish);
94          arc_finish2finished.setTarget(p_finished);
95          arc_finish2finished.setTokensNb(1);
96          arc_finish2finished.setNet(PetriNetwork);
97          PetriNetwork.getArcs().add(arc_finish2finished);
98
99          startedPlaces.put(wd.getName(), p_started);
100         finishedPlaces.put(wd.getName(), p_finished);
101         startTransitions.put(wd.getName(), t_start);
102         finishTransitions.put(wd.getName(), t_finish);
103     }
104
105     public static void convertWorkDefinitions(simplepdl.Process process)
106     {
107         for (Object obj : process.getProcessElements())
108         {
109             if (obj instanceof WorkDefinition)
110             {
111                 convertWorkDefinition((WorkDefinition) obj);
112             }
113         }
114     }
115
116     public static void convertWorkSequence(WorkSequence ws)
117     {
118         Arc arc = PetriNetFactory.createArc();
119
120         if (ws.getLinkType() == WorkSequenceType.FINISH_TO_START || ws.getLinkType() ==
    WorkSequenceType.FINISH_TO_FINISH)
121         {
122             arc.setSource(finishedPlaces.get(ws.getPredecessor().getName()));
123         } else {
124           arc.setSource(startedPlaces.get(ws.getPredecessor().getName()));
125         }
126
```

```
        if(ws.getLinkType() == WorkSequenceType.FINISH_TO_START || ws.getLinkType() ==
    WorkSequenceType.START_TO_START)
        {
            arc.setTarget(startTransitions.get(ws.getSuccessor().getName()));
        }
        else
        {
            arc.setTarget(finishTransitions.get(ws.getSuccessor().getName()));
        }
        arc.setKind(ArcKind.READ_ARC);
        arc.setNet(PetriNetwork);
        arc.setTokensNb(1);
        PetriNetwork.getArcs().add(arc);
    }

    public static void convertWorkSequences(simplepdl.Process process)
    {
        for (Object obj : process.getProcessElements())
        {
            if (obj instanceof WorkSequence)
            {
                convertWorkSequence((WorkSequence) obj);
            }
        }
    }

    public static void convertAllocation(Allocation allocation)
    {
        Place p_resource = resources.get(allocation.getResources().getName() + "_resource");
        Transition t_start = startTransitions.get(allocation.getWd().getName());
        Transition t_finish = finishTransitions.get(allocation.getWd().getName());

      Arc arc_allocateResource = PetriNetFactory.createArc();
      arc_allocateResource.setSource(p_resource);
      arc_allocateResource.setTarget(t_start);
      arc_allocateResource.setTokensNb(allocation.getCount());
      arc_allocateResource.setNet(PetriNetwork);
        PetriNetwork.getArcs().add(arc_allocateResource);

      Arc arc_deallocateResource = PetriNetFactory.createArc();
      arc_deallocateResource.setSource(p_resource);
      arc_deallocateResource.setTarget(t_finish);
      arc_deallocateResource.setTokensNb(allocation.getCount());
      arc_deallocateResource.setNet(PetriNetwork);
        PetriNetwork.getArcs().add(arc_deallocateResource);
    }

    public static void convertResource(simplepdl.Resource resource)
    {

        Place p_resource = PetriNetFactory.createPlace();
        p_resource.setName(resource.getName() + "_resource");
        p_resource.setTokensNb(resource.getCount());
        p_resource.setNet(PetriNetwork);
        PetriNetwork.getNodes().add(p_resource);

        resources.put(resource.getName() + "_resource", p_resource);
        for (Allocation allocation :  resource.getAllocations())
        {
            convertAllocation(allocation);
        }
    }

    public static void convertResources(simplepdl.Process process)
    {
```

```
        for (Object obj : process.getProcessElements())
        {
            if (obj instanceof simplepdl.Resource)
            {
                convertResource((simplepdl.Resource) obj);
            }
        }
    }

    public static void main(String[] args)
    {

      String simplePdlModel = args.length >=1 ? args[0] : "developpement";

      SimplepdlPackage simplePdlPackageInstance = SimplepdlPackage.eINSTANCE;
      PetriNetPackage petriNetworkPackageInstance = PetriNetPackage.eINSTANCE;


        Resource.Factory.Registry reg = Resource.Factory.Registry.INSTANCE;
        Map<String, Object> extensionToFactoryMap = reg.getExtensionToFactoryMap();
        extensionToFactoryMap.put("xmi", new XMIResourceFactoryImpl());

        ResourceSet resSet = new ResourceSetImpl();

        URI modelURI = URI.createURI("models/" + simplePdlModel + ".xmi");
        Resource resource = resSet.getResource(modelURI, true);

        simplepdl.Process process = (simplepdl.Process)resource.getContents().get(0);

        URI petriNetworkURI = URI.createURI("models/" + simplePdlModel + "2Petri_JAVA.xmi");
        Resource petriNetwork = resSet.createResource(petriNetworkURI);

        PetriNetFactory = petriNet.PetriNetFactory.eINSTANCE;
        PetriNetwork = PetriNetFactory.createPetriNet();
        PetriNetwork.setName(process.getName());
        petriNetwork.getContents().add(PetriNetwork);

        convertWorkDefinitions(process);
        convertWorkSequences(process);
        convertResources(process);

        try {
          petriNetwork.save(Collections.EMPTY_MAP);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

}
```

4.3 SimplePDL2PetriNet.atl

```
module SimplePDL2PetriNet;
create OUT: PetriNet from  IN: SimplePDL;

helper context SimplePDL!ProcessElement
def: getProcess(): SimplePDL!Process =
  SimplePDL!Process.allInstances()
    ->select(p | p.processElements->includes(self))
    ->asSequence()->first();

rule Process2PetriNet {
    from p: SimplePDL!Process
    to pn: PetriNet!PetriNet (name <- p.name)
}


rule WorkDefinition2PetriNet {
    from wd: SimplePDL!WorkDefinition
    to
    --PLACES
        p_notStarted: PetriNet!Place(
            name <- wd.name + '_notStarted',
            tokensNb <- 1,
            net <- wd.getProcess()
    ),

    p_started: PetriNet!Place(
            name <- wd.name + '_started',
            tokensNb <- 0,
            net <- wd.getProcess()
    ),

        p_inProgress: PetriNet!Place(
            name <- wd.name + '_inProgress',
            tokensNb <- 0,
            net <- wd.getProcess()
    ),

        p_finished: PetriNet!Place(
            name <- wd.name + '_finished',
            tokensNb <- 0,
            net <- wd.getProcess()
    ),

    --TRANSITIONS
        t_start: PetriNet!Transition(
            name <- wd.name + '_start',
            net <- wd.getProcess()
    ),

        t_finish: PetriNet!Transition(
            name <- wd.name + '_finish',
            net <- wd.getProcess()
    ),

    --ARCS
        a_notStarted2start: PetriNet!Arc(
            tokensNb <- 1,
            net <- wd.getProcess(),
      source <- p_notStarted,
            target <- t_start
    ),

```

```
        a_start2started: PetriNet!Arc(
            tokensNb <- 1,
            net <- wd.getProcess(),
      source <- t_start,
            target <- p_started
    ),

        a_start2inProgress: PetriNet!Arc(
            tokensNb <- 1,
            net <- wd.getProcess(),
          source <- t_start,
            target <- p_inProgress
    ),

        a_inProgress2finish: PetriNet!Arc(
            tokensNb <- 1,
            net <- wd.getProcess(),
          source <- p_inProgress,
            target <- t_finish
    ),

        a_finish2finished: PetriNet!Arc(
            tokensNb <- 1,
            net <- wd.getProcess(),
          source <- t_finish,
            target <- p_finished
    )
}

rule WorkSequence2PetriNet {
    from ws: SimplePDL!WorkSequence

    to
        a_ws: PetriNet!Arc(
          tokensNb <- 1,
            kind <- #readArc,
            net <- ws.successor.getProcess(),
            source <-   thisModule.resolveTemp(ws.predecessor,
                  if((ws.linkType = #finishToStart) or (ws.linkType = #finishToFinish))
                    then 'p_finished'
            else 'p_started'
            endif),
            target <-   thisModule.resolveTemp(ws.successor,
                  if((ws.linkType = #finishToStart) or (ws.linkType = #startToStart))
                    then 't_start'
            else 't_finish'
                  endif)
    )
}

rule Allocation2PetriNet {
  from allocation: SimplePDL!Allocation
  to
    a_allocateResource: PetriNet!Arc(
      tokensNb <- allocation.count,
      net <- allocation.resources.getProcess(),
      source <- allocation.resources,
      target <- thisModule.resolveTemp(allocation.wd, 't_start')
    ),
    a_deallocateResource: PetriNet!Arc(
      tokensNb <- allocation.count,
      net <- allocation.resources.getProcess(),
      source <- allocation.resources,
      target <- thisModule.resolveTemp(allocation.wd, 't_finish')
    )
```

```
}

rule Resource2PetriNet {
  from resource: SimplePDL!Resource
  to
    p_resource: PetriNet!Place(
      name <- resource.name,
      tokensNb <- resource.count,
      net <- resource.getProcess()
    )
}
```

4.4 toTINA.mtl

```
[comment encoding = UTF-8 /]
[module toTINA('http://www.example.org/petriNet')]


[template public petriToTINA(aPetriNet : PetriNet)]
[comment @main/]
[file (aPetriNet.name.concat('2TINA.net'), false, 'UTF-8')]
net [aPetriNet.name/]
[for (pl : Place | aPetriNet.nodes->getPlaces())]
  pl [pl.name/] ([pl.tokensNb/])
[/for]
[for (t : Transition | aPetriNet.nodes->getTransitions())]
  tr [t.name/] [t.getIngoingTransitions()/] -> [t.getOutgoingTransitions()/]
[/for]
[/file]
[/template]

[query public getPlaces(elements : OrderedSet(Node)) : OrderedSet(Place) =
  elements
    ->select(e | e.oclIsTypeOf(Place))
    ->collect(e | e.oclAsType(Place))
    ->asOrderedSet()
/]

[query public getTransitions(elements : OrderedSet(Node)) : OrderedSet(Transition) =
  elements->select(e | e.oclIsTypeOf(Transition))
    ->collect(e | e.oclAsType(Transition))
    ->asOrderedSet()
/]

[template public getIngoingTransitions(t : Transition)]
  [for (arc : Arc | t.ingoing)][arc.getTargetsAndTokens()/][/for]
[/template]

[template public getOutgoingTransitions(t : Transition)]
  [for (arc : Arc | t.outgoing)][arc.getSourcesAndTokens()/][/for]
[/template]

[template public getTargetsAndTokens(arc : Arc)]
  [arc.source.name/][if (arc.kind = ArcKind::readArc)]?[arc.tokensNb/][elseif (arc.tokensNb
    > 1)]*[arc.tokensNb/][/if]
[/template]

[template public getSourcesAndTokens(arc : Arc)]
  [arc.target.name/][if (arc.tokensNb > 1)]*[arc.tokensNb/][/if]
[/template]
```

4.5 toLTL.mtl

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


[template public processtoLTL(aProcess : Process)]
[comment @main/]
[file (aProcess.name.concat('_VerificationsInvariants.ltl'), false, 'UTF-8')]
#-----------------------------------------
#                                        |
# Process [aProcess.name/]
#----------------------------------------|
#

# Verification terminaison processus
[for (wd : WorkDefinition | aProcess.processElements->getWDs()) before ('op finished = ')
    separator (' /\\ ') after (';')][wd.name/]_finished[/for]
[for (wd : WorkDefinition | aProcess.processElements->getWDs()) before ('op notStarted = ')
    separator (' /\\ ') after (';')][wd.name/]_notStarted[/for]
[for (wd : WorkDefinition | aProcess.processElements->getWDs()) before ('op started = ')
    separator (' /\\ ') after (';')][wd.name/]_started[/for]
[for (wd : WorkDefinition | aProcess.processElements->getWDs()) before ('op inProgress = ')
    separator (' /\\ ') after (';')][wd.name/]_inProgress[/for]

# Une activite terminee n'evolue plus
['[]'/] (finished => dead);
# Une activite finit toujours pas cesser d'evoluer
['[] <>'/] dead;
# Une activite n'evoluant plus est terminee
['[]'/] (dead => finished);
# Une activite ne termine jamais
['- <>'/] finished;

# Une activite commencee ne peut pas redemarrer
['[]'/] (started => ['- <>'/] notStarted);
# Une activite finie ne peut pas etre en cours
['[]'/] (finished => ['- <>'/] inProgress);
# Une activite finie ne peut pas redemarrer
['[]'/] (finished => ['- <>'/] notStarted);
# Une activite terminee signifie qu'elle a ete d'abord commencee
['[]'/] (finished => started);

# Chaque activite est soit non commencee soit en cours soit terminee
[for(wd : WorkDefinition | aProcess.processElements->getWDs())] ['[]'/] ([wd.name/]
    _notStarted + [wd.name/]_inProgress + [wd.name/]_finished = 1);
[/for]
[/file]
[/template]

[query public getWDs(elements : OrderedSet(ProcessElement)) : OrderedSet(WorkDefinition) =
  elements->select( e | e.oclIsTypeOf(WorkDefinition) )
    ->collect( e | e.oclAsType(WorkDefinition) )
    ->asOrderedSet()
/]

[query public getWSs(elements : OrderedSet(ProcessElement)) : OrderedSet(WorkSequence) =
  elements->select( e | e.oclIsTypeOf(WorkSequence) )
    ->collect( e | e.oclAsType(WorkSequence) )
    ->asOrderedSet()
/]

[template public getLinkType(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]
```