Ingénierie Dirigée par les Modèles

Mini-Projet IDM

Groupe L34-02

Élèves :

THEVENET Louis

SABLAYROLLES Guillaume

Octobre 2024

Contents

1. Méta-Modèles 3
   1.1. SimplePDL 3
   1.2. Exemples 3
   1.3. PetriNet 3
2. Transformation Modèle à Modèle 4
   2.1. SimplePDL vers PetriNet 4
3. Transformation Modèle à Texte 7
   3.1. `SimplePDL` vers Dot 7
   3.2. `PetriNet` vers Tina 8
   3.3. `PetriNet` vers Dot 8
4. Transformation Texte à Modèle 9
5. Vérification de terminaison et invariants 9

1. Méta-Modèles

1.1. SimplePDL

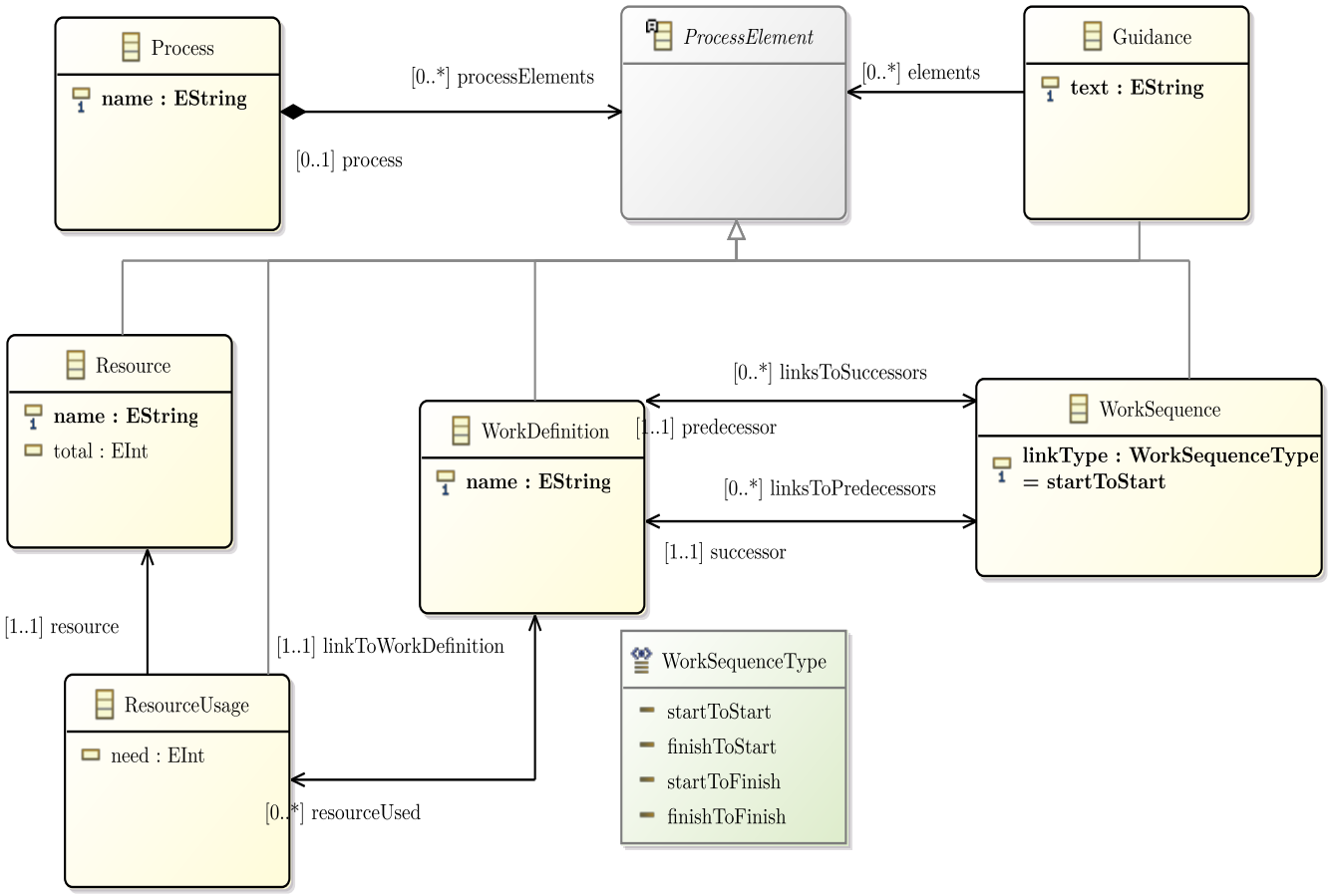

Figure 1: Méta-Modèle de SimplePDL

A partir du méta-modèle fourni, nous avons rajouté l'utilisation des ressources. Une ressource est défini comme une EClass `Resource` ayant :

- un nom
- et un nombre total d'éléments disponible (`total`)

Une `WorkDefinition` utilise une `Resource` en ajoutant une référence à une `ResourceUsage` qui contient :

- référence à la `Resource` en question
- la quantité demandée (`need`)

1.2. Exemples

Ici mettre des exemples

1.3. PetriNet

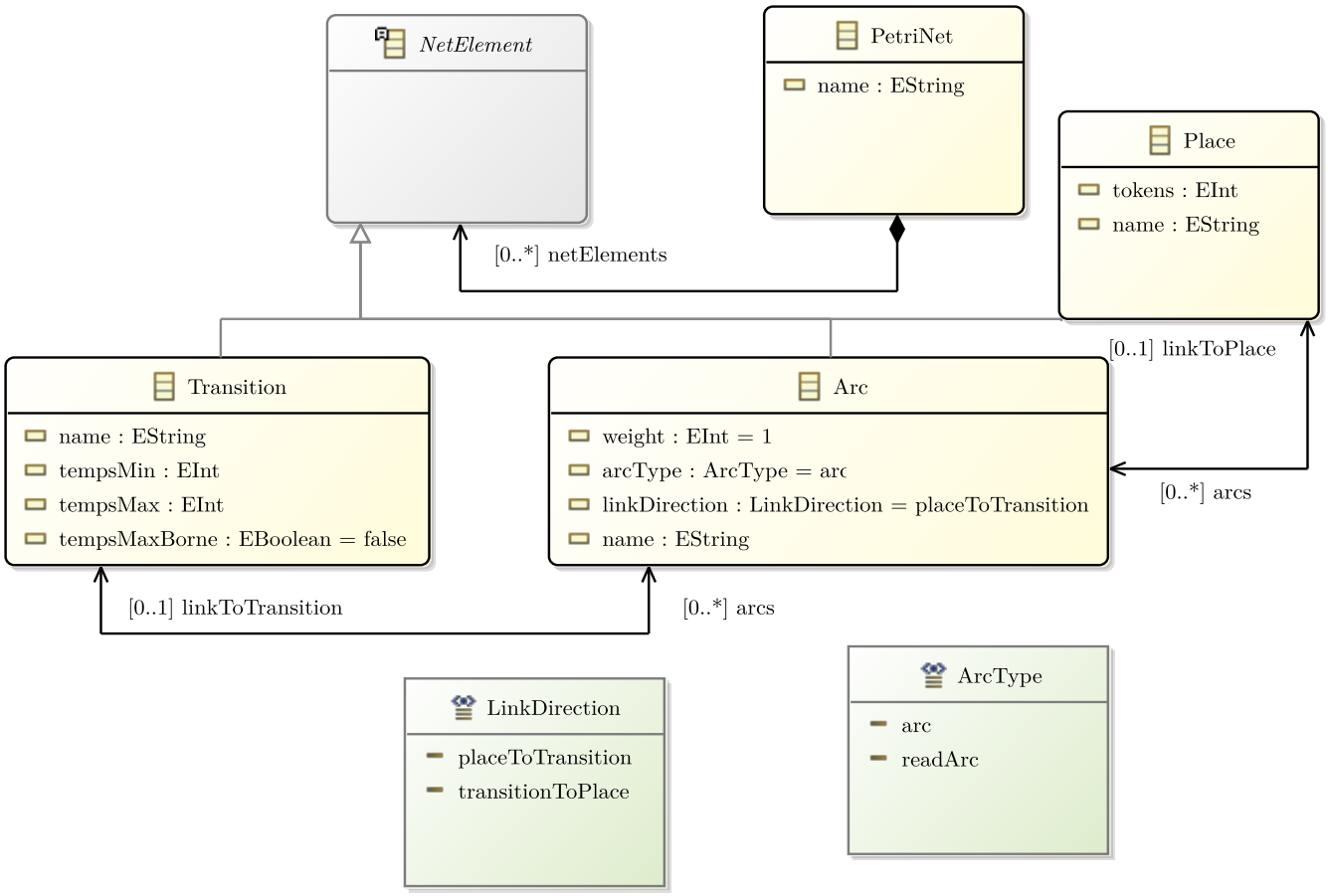

Figure 2: Méta-Modèle de PetriNet

Un `PetriNet` est constitué de `NetElement`. Ces éléments sont les `EClass` `Place`, `Transition` et `Arc`.

Une `Place` est définie par un nom et un nombre de jetons (`tokens`).

Une `Transition` est définie par un nom.

Un `Arc` contient :

- un nom
- un poids
- une référence vers une `Place`
- une référence vers une `Transition`
- Une `LinkDirection` (soit `placeToTransition`, soit `transitionToPlace`)
- Un `arcType` (soit `arc`, soit `readArc`).

Nous avons également ajouté des attributs `tempsMin`, `tempsMax` et `tempsMaxBorne` pour ajouter la notion de temps aux `Transition`.

Ce méta-modèle permet de s'assurer que les `Arc` ne relient jamais deux `Transition` ou deux `Place`.

2. Transformation Modèle à Modèle

2.1. SimplePDL vers PetriNet

Principe de la transformation d'un modèle de processus en réseau de Petri:

- Un élément `Process` devient un élément `PetriNet`

- Une `WorkDefinition` devient 4 places `ready` (avec 1 jeton), `started`, `running` et `finished` et deux transitions `start` et `finish` reliées par des arcs
- Une `WorkSequence` devient un read-arc entre une place de l'activité précédente (`started` ou `finished`) et une transition de l'activité cible (`start` ou `finish`)

Transformation des ressources :

- Une `Resource` devient une place dont le nombre de jetons est égal au nombre de ressources initialement disponibles
- Une `ResourceUsage` devient deux arcs avec pour poids le nombre de ressources demandé :
  - De la place représentant la `Resource` utilisée à la transition `start` de la `WorkDefinition`
  - De la transition `finish` de la `WorkDefinition` à la place représentant la `Resource` utilisée

2.1.1. Fichier d'entrée

Pour illustrer les transformations, nous utiliserons l'exemple de modèle de processus suivant.

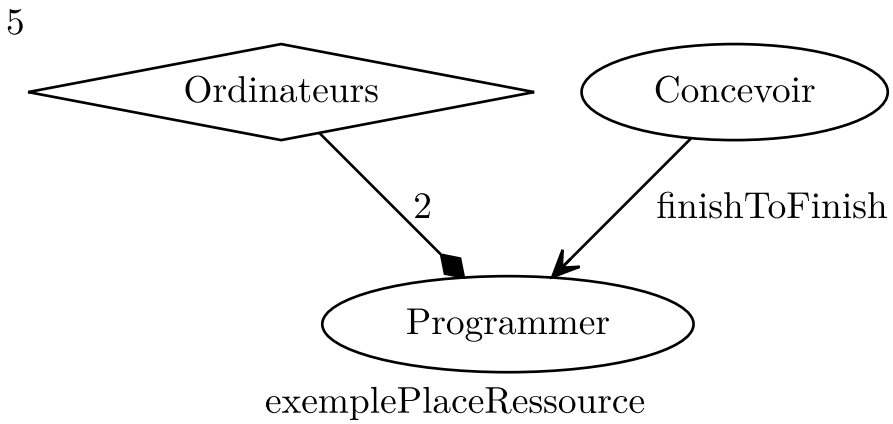

Figure 3: Modèle de processus simple avec une ressource

2.1.2. Java/EMF

On réalise un premier programme de transformation en Java (voir `SimplePDL2PetriNet.java`)

Lors de cette transformation, on traite les `ProcessElement` dans cet ordre :

1. `Resource`
2. `WorkDefinition`
   - `ResourceUsage` (on traite les `ResourceUsage` attachés à la `WorkDefinition` courante)
3. `WorkSequence`

La Figure 4 représente le réseau de Petri en sortie du programme.

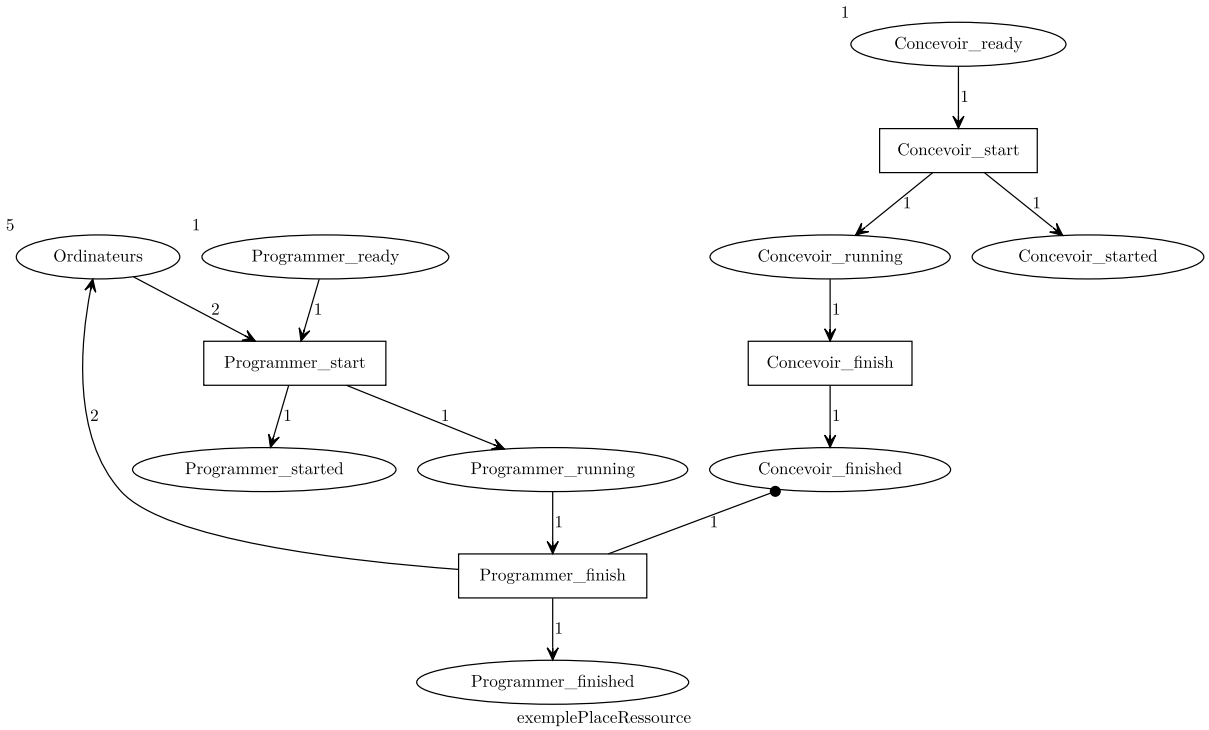

Figure 4: Réseau de Petri résultant de la transformation par Java

On distingue aisément les différents sous-réseaux de Petri associés aux `WorkDefinition` ainsi que la `Ressource` et les arcs qui la relient au sous-réseau associé à *Programmer*

2.1.3. ATL

On réaliste également la même transformation à l'aide d'ATL (voir `SimplePDL2PetriNet.atl`)

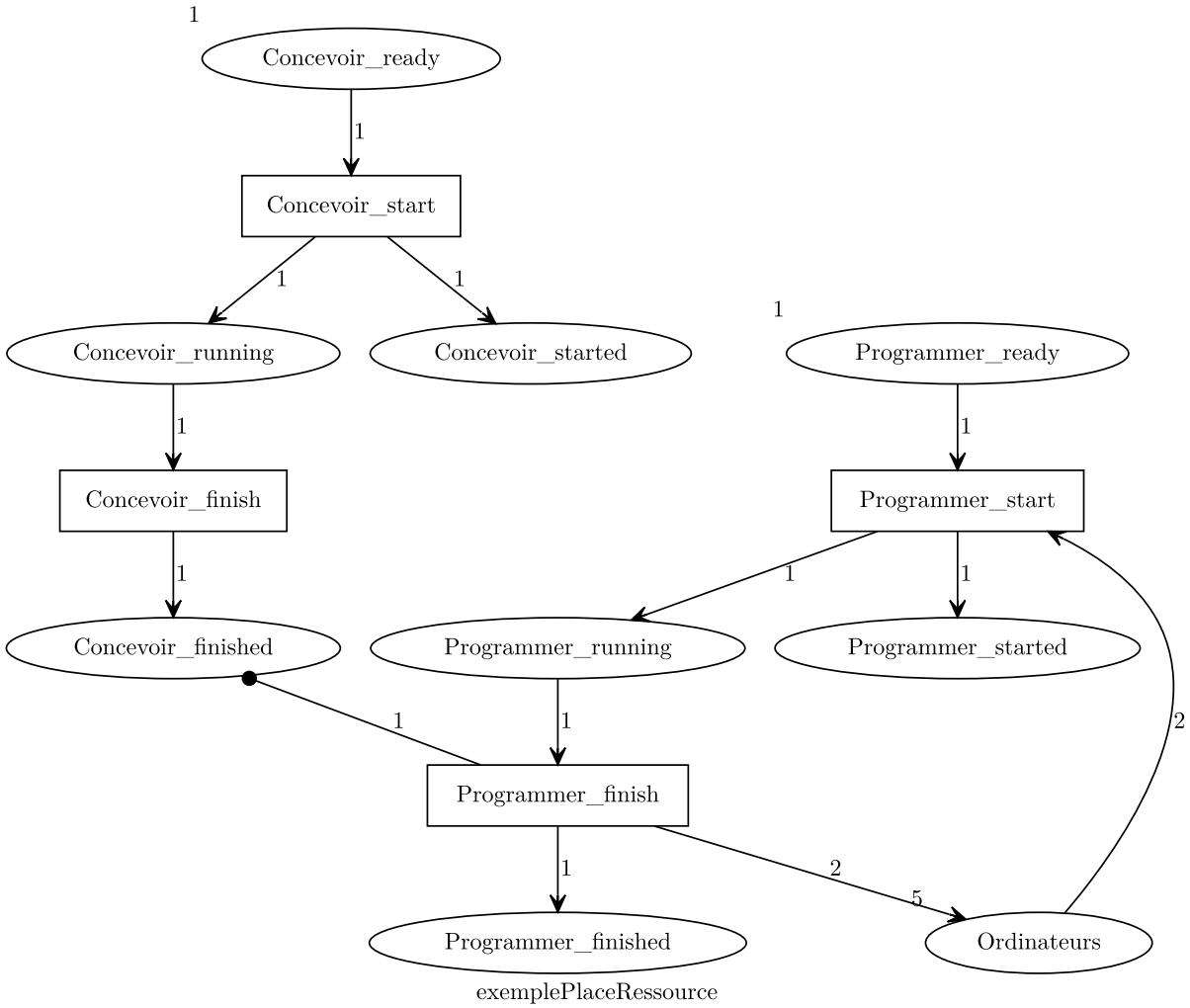

Figure 5: Réseau de Petri résultant de la transformation par ATL

L'emplacement des noeuds n'est plus exactement le même mais les graphes sont bien identiques.

3. Transformation Modèle à Texte

Nous avons réalisé plusieurs transformations modèle à texte :

- `SimplePDL2Dot`
- `PetriNet2Tina`
- `PetriNet2Dot`

Les images précédentes ont été réalisées à partir des transformation vers le format DOT.

3.1. `SimplePDL` vers Dot

Pour chaque `Resource`, on déclare un `node` avec la forme `diamond`, le même nom et le nombre total de ressources.

Pour chaque `WorkSequence` on déclare un arc entre le prédecesseur et le successseur (les `node` associés aux `WorkDefinition` seront générés automatiquement)

Pour chaque `ResourceUsage` on déclare un arc entre la ressource et la `WorkDefinition` et une tête avec la forme `diamond`.

Voir Figure 3 pour un exemple.

3.2. `PetriNet` vers Tina

Le format NET est une traduction presque directe du méta-modèle `PetriNet`, ce qui rend la transformation très simple. (voir `PetriNet2Tina.mtl`)

On parcourt d'abord les `Place` pour les déclarer, puis on déclare chaque transition en ajoutant si besoin les contraintes temporelles et en traitant le cas des read-arcs.

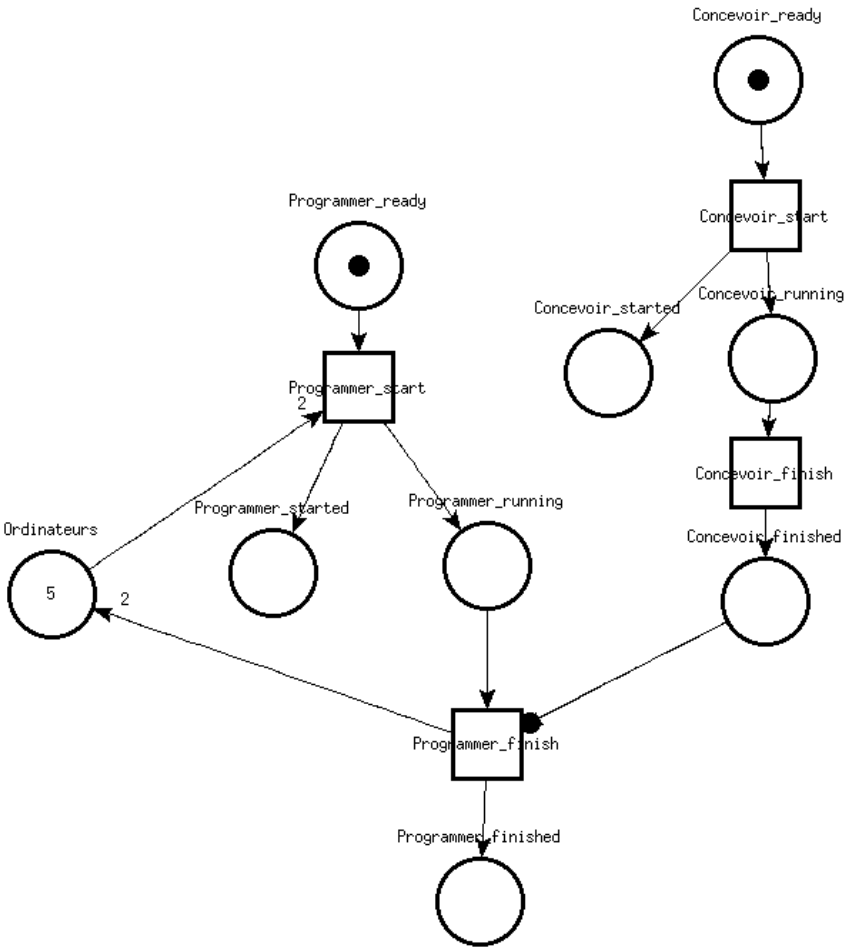

Figure 6: Capture d'écran de Tina affichant le fichier NET résultat

3.3. `PetriNet` vers Dot

Pour chaque `Place`, on déclare un `node` avec le même nom et le nombre de jetons associés.

Pour chaque `Transition`, on déclare un `node` avec le même nom et les éventuelles contraintes temporelles.

On déclare ensuite les arcs en traitant les read-arcs.

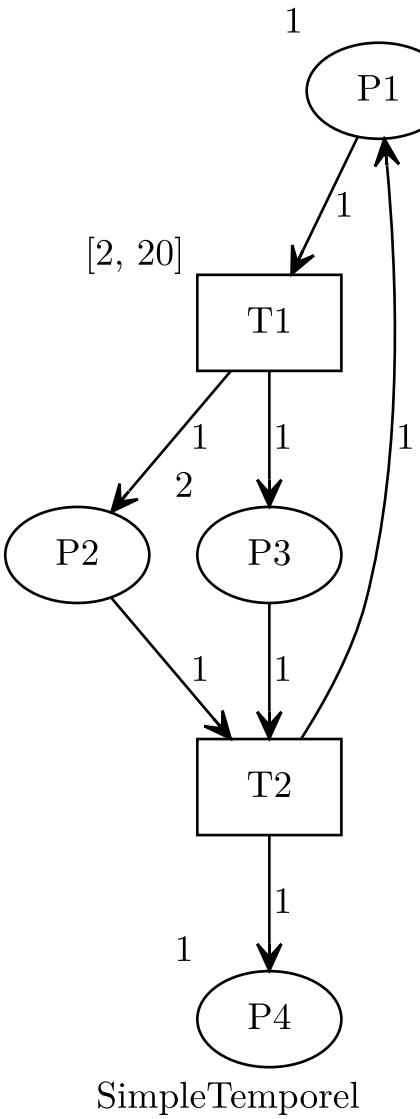

Figure 7: Sortie : réseau de Petri résultant de la transformation en DOT

4. Transformation Texte à Modèle

5. Vérification de terminaison et invariants