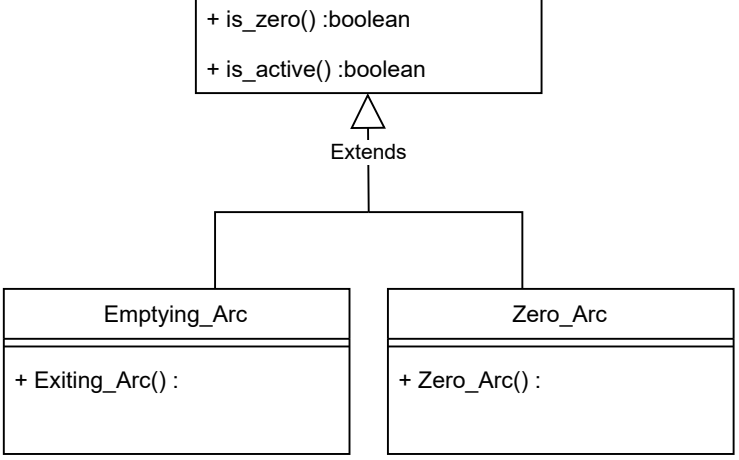

PetriNet
- transitionsList : LinkedList<Transition>
- placesList : LinkedList<Place>
- arcsList : LinkedList<Arc>
+ PetriNet() :
+ firable_transitions() : LinkedList<Transition>
+ fire(Transition) : void
+ add_transition(exiting_ArcsList, entering_ArcsList) : void
+ remove_transition(Transition) : void
+ add_enteringArc(int, Place, Transition) : void
+ add_exitingArc(int, Place, Transition) : void
+ add_zeroArc(int, Place, Transition) : void
+ add_emptyingArc(int, Place, Transition) : void
+ remove_arc(Arc) : void
+ set_arc_weight(Arc, int) : void
+ add_place(int, arcsList) : void
+ remove_place(Place) : void
+ add_tokens(Place, int) : void
+ remove_tokens(Place, int) : void
+ get_TransitionsList() : LinkedList<Transition>
+ get_ArcsList() : LinkedList<Arc>
+ get_PlacesList() : LinkedList<Place>
1..1
1..1
fire>
add/remove_transition>
1..1
add/remove_place>
set_tokens_number>
0..*
1..1
add/remove_arc>
0..*
1..1
set_arc_weight>
1..n
0..*
0..*
Place
- tokens_number : int
- arcsList : LinkedList<Arc>
+ Place()
+ get_tokens_nb() : int
+ set_tokens_nb(int) : void
+ is_empty() : boolean
+ get_arcList() : LinkedList<Arc>
+ add_tokens(int) : void
+ remove_tokens(int) : void
Arc
- weight : int
- place : Place
- transition : Transition
+ Arc() :
+ set_weight(int) : void
+ get_transition() : Transition
+ get_place() : Place
+ get_weight() : int
+ is_exiting() : boolean
+ is_entering() : boolean
+ execute (): void
Transition
- exiting_ArcsList : LinkedList<Arc>
- entering_ArcsList : LinkedList<Arc>
+ Transition() :
+ is_firable() : boolean
+ fire() : void
+ get_exitingList() : LinkedList<Arc>
+ get_enteringList() : LinkedList<Arc>
1..1
-Arc
1..n
- Place
-Arc
1..1
2..n
-Transition
Extends
Entering_Arc
+ Entering_Arc() :
+ is_empty() :boolean
Exiting_Arc
+ Exiting_Arc() :


+ is_zero() :boolean
+ is_active() :boolean
Extends
Emptying_Arc
+ Exiting_Arc() :
Zero_Arc
+ Zero_Arc() :

Commentaires :

Diagramme de classe :

Les classes :

1. “PetriNet”:

- La classe “PetriNet” représente le réseau de Petri global et gère les places, transitions et arcs.
- Elle offre des méthodes pour ajouter, supprimer et sélectionner des éléments dans le réseau.

2. “Place” :

- La classe “Place” représente une place du réseau de Petri avec un attribut “tokens_number” pour les jetons et “arcsList” pour la liste des arcs liés à cette place .
- Elle permet d'ajouter et de retirer des jetons de la place.

3. “Transition” :

- La classe “Transition” représente une transition dans le réseau de Petri.
- Elle offre une méthode “fire()” pour tirer la transition.

4. “Arc” :

- La classe “Arc” est la classe de base pour tous les arcs, avec un attribut “weight” pour le poids de l'arc, “place” pour la place liée à l’arc et “transition” pour la transition liée à ce dernier.
- Elle permet de définir le poids de l'arc via la méthode “set_Weight()”.

5. “Exiting_Arc” :

- `”Exiting_Arc” est une sous-classe de “Arc” représentant les arcs sortants des places vers les transitions.

6. ”Entering_Arc” :

- ”Entering_Arc” est une sous-classe de “Arc” représentant les arcs entrants des transitions vers les places.
-Elle offre une méthode “is_active()” pour vérifier l'activation de l’arc.

7. “Zero_Arc” :

- “Zero_Arc” est une sous-classe de ”Entering_Arc” spécifique aux arcs "zéro".

8. `Emptying_Arc` :

- `Emptying_Arc` est une sous-classe de ”Entering_Arc” spécifique aux arcs "videurs".

Relations entre classes :

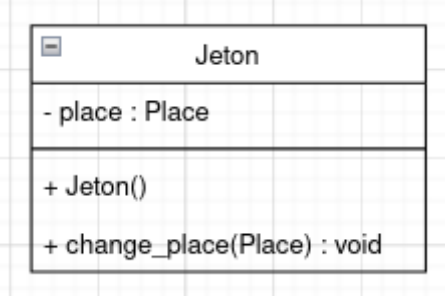
1. Relation entre `PetriNet` et `Transition` :
    - Cette relation permet à un réseau de Petri (`PetriNet`) de contenir un ou plusieurs transitions (`Transition`).
    - Les actions possibles sont les suivantes :
        - "fire" : Un réseau de Petri peut déclencher une transition en appelant la méthode `fire()` de la transition.
        - "add/remove transition" : Un réseau de Petri peut ajouter ou supprimer des transitions en utilisant les méthodes `add_Transition()` et `remove_Transition()`.
2. Relation entre `PetriNet` et `Place` :

- Cette relation permet à un réseau de Petri (`PetriNet`) de contenir un ou plusieurs places (`Place`).
- Les actions possibles sont les suivantes :
  - "set tokens" : Un réseau de Petri peut définir le nombre de jetons dans une place en utilisant la méthode `add_Tokens()` ou `remove_Tokens()` de la place.
  - "add/remove place" : Un réseau de Petri peut ajouter ou supprimer des places en utilisant les méthodes `add_Place()` et `remove_Place()`.

3. Relation entre `PetriNet` et `Arc` :
   - Cette relation permet à un réseau de Petri (`PetriNet`) de contenir un ou plusieurs arcs (`Arc`).
   - Les actions possibles sont les suivantes :
     - "setweight" : Un réseau de Petri peut définir le poids d'un arc en utilisant la méthode `set_Weight()` de l'arc.
     - "add/remove arc" : Un réseau de Petri peut ajouter ou supprimer des arcs en utilisant les méthodes `add_Arc()` et `remove_Arc()`.
4. Relation entre `Arc` et `Place` :
   - La relation entre `Arc` et `Place` est une association bidirectionnelle, cette relation permet à un arc (`Arc`) de se connecter à une place (`Place`) et vice versa, ce qui signifie que chaque arc est associé à une place source ou destination, et chaque place peut avoir plusieurs arcs entrants ou sortants.
   - Cette relation reflète comment les arcs sont utilisés pour relier les places et les transitions dans le réseau de Petri, et comment ils affectent le nombre de jetons dans une place ou l'activation d'une transition.
5. Relation entre `Arc` et `Transition` :
   - La relation entre `Arc` et `Transition` est une association bidirectionnelle, cette relation permet à un arc (`Arc`) de se connecter à une transition (`Transition`) et vice versa, ce qui signifie que chaque arc est associé à une transition et chaque transition peut avoir plusieurs arcs entrants ou sortants.

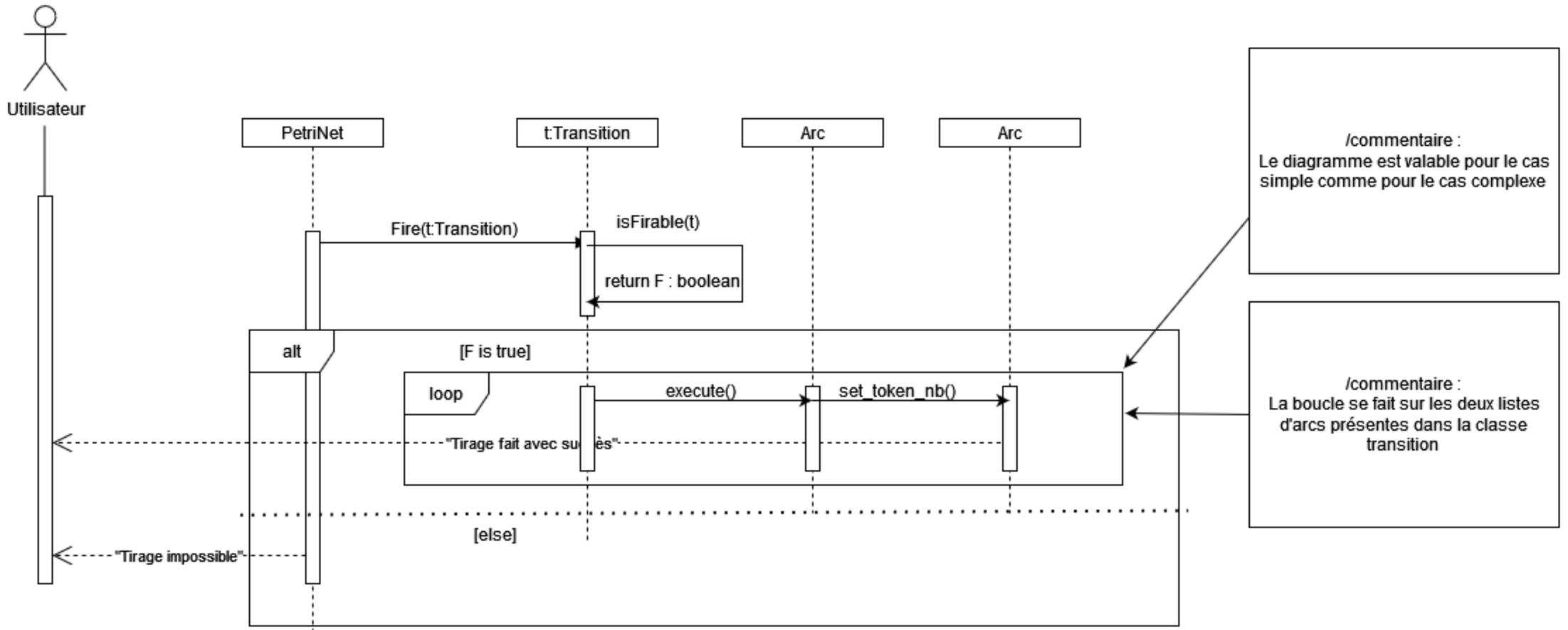
Variante :

Dans un premier temps, nous avons envisagé de créer une classe "Jeton" pour établir un lien plus étroit entre la classe "Place" et les jetons qu'elle contient :



Cependant, cette approche peut sembler inefficace car elle introduit de la complexité et une charge supplémentaire dans le modèle de réseau de Petri. Chaque jeton entraîne une utilisation de la mémoire, ce qui peut accroître la consommation de ressources, et la gestion individuelle des jetons peut devenir fastidieuse. Normalement, cette conception est plus pertinente lorsque les jetons possèdent des propriétés ou des comportements particulièrement spécifiques à prendre en compte.

Fire



get_ArcsList