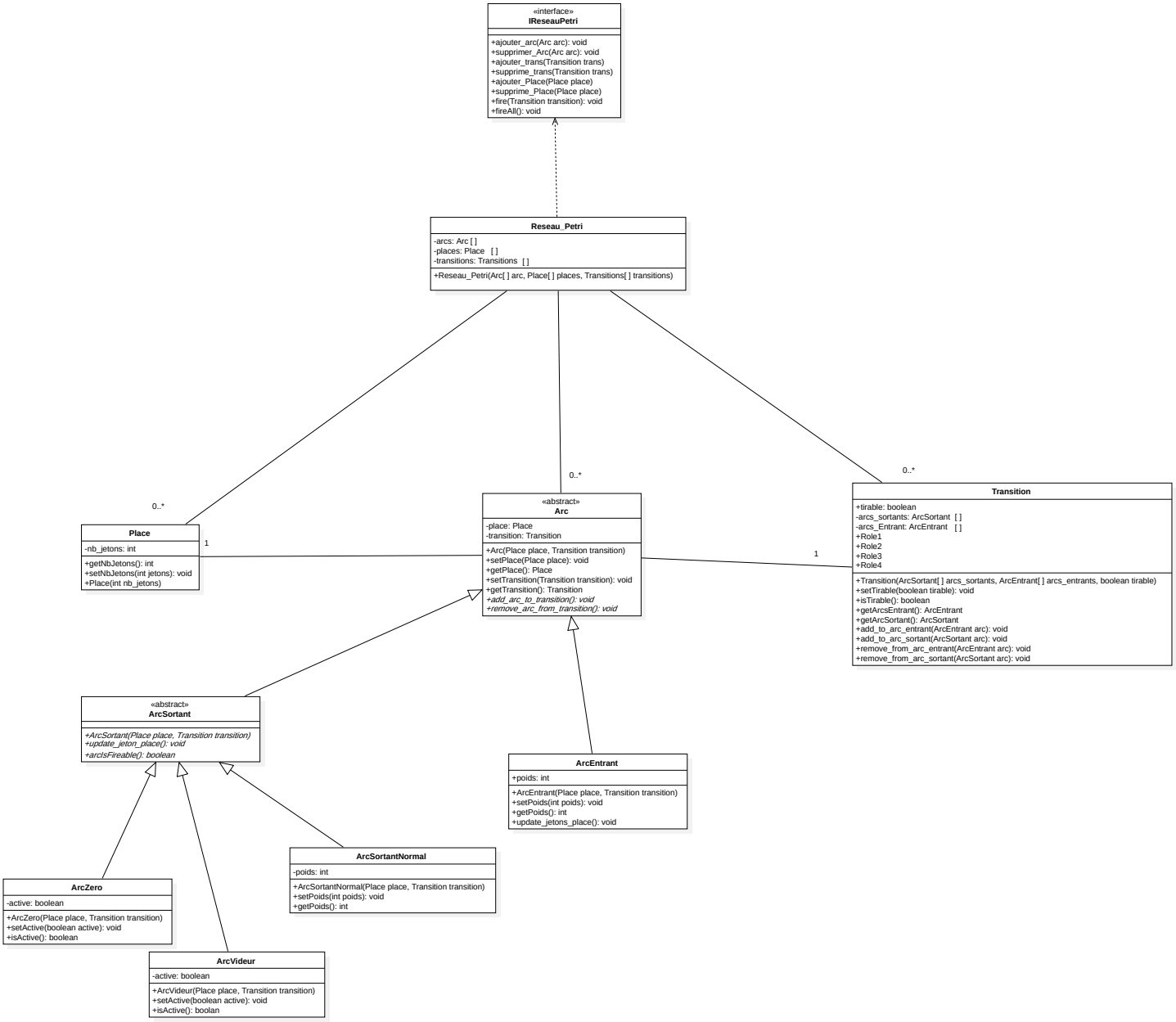
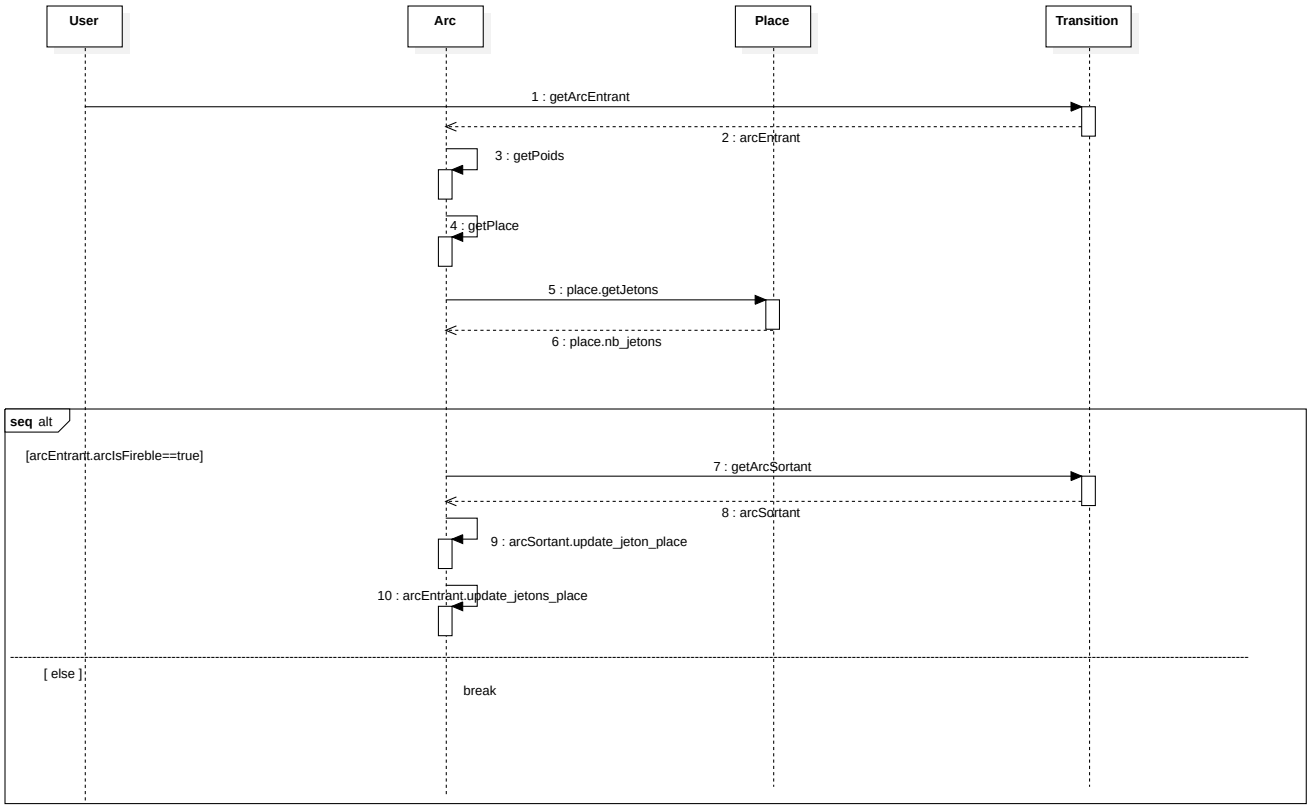
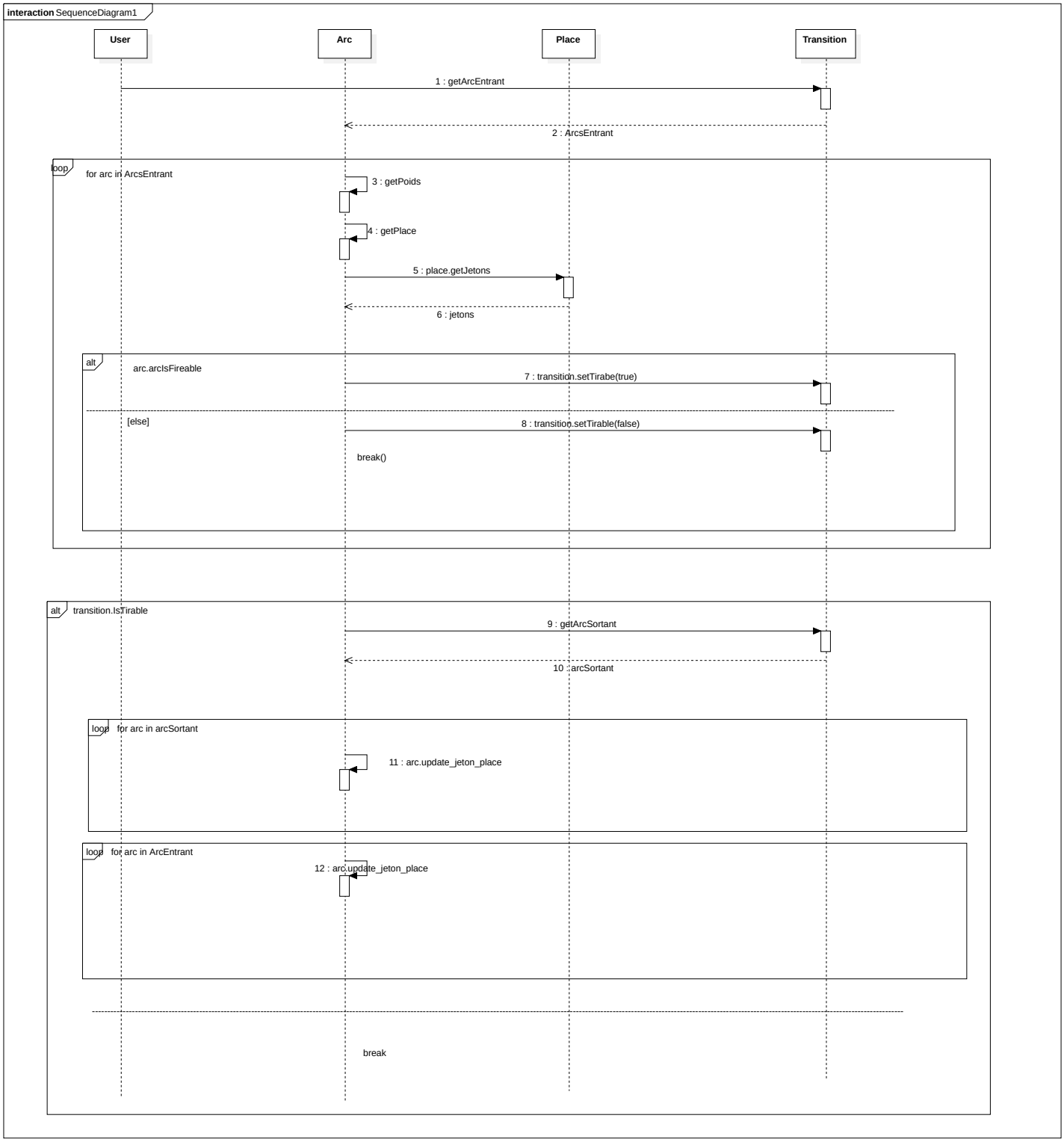
Model::Main


«interface»
IReseauPetri
+ajouter_arc(Arc arc): void
+supprimer_Arc(Arc arc): void
+ajouter_trans(Transition trans)
+supprime_trans(Transition trans)
+ajouter_Place(Place place)
+supprime_Place(Place place)
+fire(Transition transition): void
+fireAll(): void
Reseau_Petri
-arcs: Arc [ ]
-places: Place [ ]
-transitions: Transitions [ ]
+Reseau_Petri(Arc[ ] arc, Place[ ] places, Transitions[ ] transitions)
0..*
0..*
0..*
1
1
Place
-nb_jetons: int
+getNbJetons(): int
+setNbJetons(int jetons): void
+Place(int nb_jetons)
«abstract»
Arc
-place: Place
-transition: Transition
+Arc(Place place, Transition transition)
+setPlace(Place place): void
+getPlace(): Place
+setTransition(Transition transition): void
+getTransition(): Transition
+add_arc_to_transition(): void
+remove_arc_from_transition(): void
Transition
+tirable: boolean
-arcs_sortants: ArcSortant [ ]
-arcs_Entrant: ArcEntrant [ ]
+Role1
+Role2
+Role3
+Role4
+Transition(ArcSortant[ ] arcs_sortants, ArcEntrant[ ] arcs_entrants, boolean tirable)
+setTirable(boolean tirable): void
+isTirable(): boolean
+getArcsEntrant(): ArcEntrant
+getArcSortant(): ArcSortant
+add_to_arc_entrant(ArcEntrant arc): void
+add_to_arc_sortant(ArcSortant arc): void
+remove_from_arc_entrant(ArcEntrant arc): void
+remove_from_arc_sortant(ArcSortant arc): void
«abstract»
ArcSortant
+ArcSortant(Place place, Transition transition)
+update_jeton_place(): void
+arcIsFireable(): boolean
ArcEntrant
+poids: int
+ArcEntrant(Place place, Transition transition)
+setPoids(int poids): void
+getPoids(): int
+update_jetons_place(): void
ArcSortantNormal
-poids: int
+ArcSortantNormal(Place place, Transition transition)
+setPoids(int poids): void
+getPoids(): int
ArcZero
-active: boolean
+ArcZero(Place place, Transition transition)
+setActive(boolean active): void
+isActive(): boolean
ArcVideur
-active: boolean
+ArcVideur(Place place, Transition transition)
+setActive(boolean active): void
+isActive(): boolan

Model::Transition::Interaction1::SequenceDiagram1

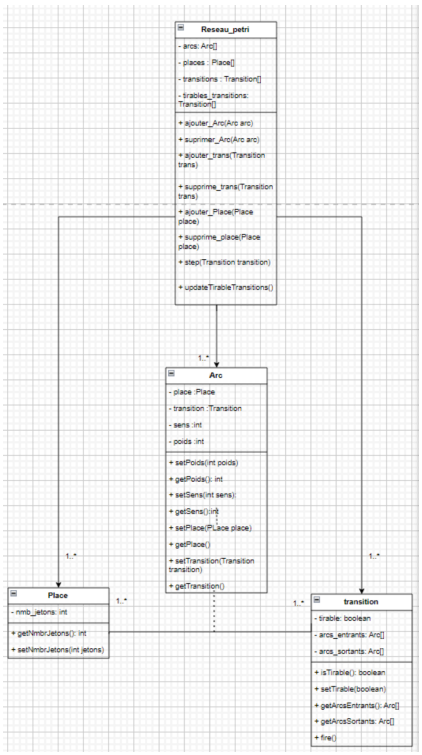


Collaboration1::Interaction1::SequenceDiagram1



Critique de notre première conception et valorisation d'une nouvelle conception

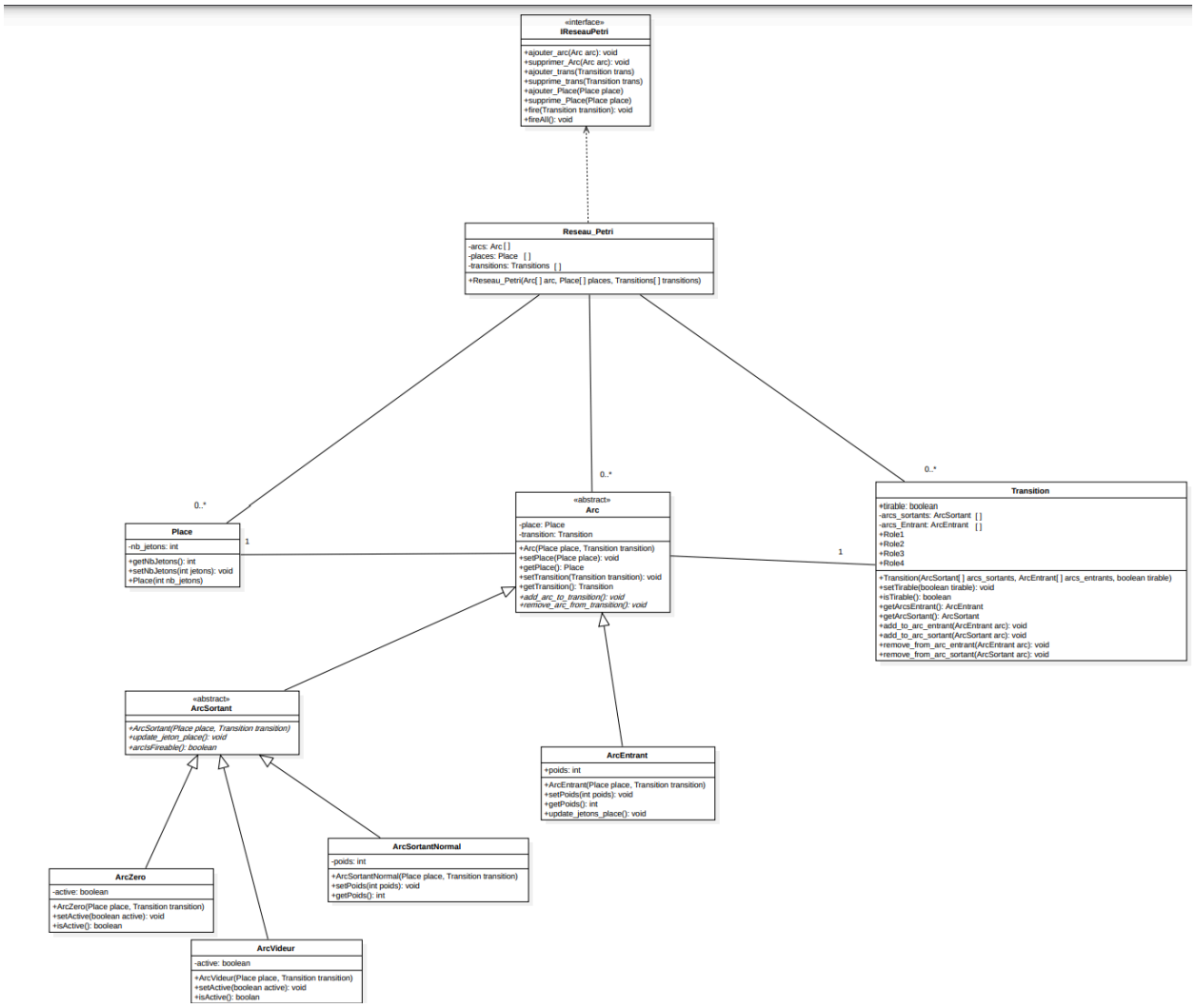
Conception initiale:



Dans notre modélisation initiale, nous avons envisagé une classe centrale appelée **Reseau_de_Petri** qui regrouperait toutes les fonctionnalités décrites dans le cahier des charges, telles que l'ajout et la suppression d'arcs, de transitions, et de places. Cette classe devait servir de point de gestion unique pour les éléments du réseau de Petri. Cependant, pour ce qui concerne la modification du poids des arcs, nous avons décidé de déléguer cette responsabilité aux arcs eux-mêmes.
Notre première idée était de concevoir une classe générique appelée **Arc**, définie par les attributs poids, place, transition, et sens. Nous avions choisi cette approche car un arc n'a aucune raison d'exister s'il n'est pas lié à une place et à une transition. Autrement dit, chaque Place et chaque Transition doit être connectée par au moins un arc. C'est pourquoi nous avons opté pour une association unidirectionnelle, avec chaque arc connecté à une place et une transition spécifiques. L'attribut sens permettait de distinguer les arcs entrants (1) des arcs sortants (-1). Cette modélisation était censée être simple, mais elle a révélé de nombreuses faiblesses. En effet, la classe Arc se contentait d'implémenter des getters et des setters, sans logique métier. Toute la complexité était donc concentrée dans la classe Transition via les méthodes isTirable et fire, ce qui créait un problème de répartition des responsabilités.

La **classe Transition** gérait deux attributs principaux : tirable (indiquant si elle peut être activée) et deux listes d'arcs (entrants et sortants). La méthode isTirable analysait chaque arc entrant et comparait le nombre de jetons de la place associée au poids requis pour déterminer si la transition pouvait se déclencher. Ensuite, fire exécutait la transition en mettant à jour les jetons dans les places connectées. En résumé, toute la logique de validation et d'exécution était centralisée dans cette classe, tandis que la classe Arc restait passive. Cette approche avait pour effet de rendre la classe Transition surchargée, avec une forte dépendance à la structure des arcs, ce qui rendait le code difficilement extensible et maintenable.
À ce stade, nous avons réalisé que notre conception initiale manquait de clarté au niveau de la répartition des responsabilités. La classe Arc était sous-utilisée, car tout le travail était effectué dans les méthodes fire et isTirable de Transition. Cela compliquait la gestion de nouvelles règles ou d'éventuelles modifications dans le système, car tout changement devait être intégré dans ces deux méthodes centrales.

Conception finale:



Nous avons alors entrepris une refonte complète, en tenant compte des retours et en cherchant à exploiter pleinement le principe de **programmation orientée objet**. Nous avons repensé

notre modèle pour mieux décomposer les responsabilités en utilisant le concept d'héritage. La première étape a été de définir une **classe abstraite Arc** qui servirait de base pour les différentes sortes d'arcs, avec deux méthodes abstraites : add_arc_to_transition et remove_arc_from_transition. Ces méthodes, implémentées dans les sous-classes, assureraient une gestion spécifique selon le type d'arc (entrant ou sortant). À ce niveau, nous devions distinguer entre les arcs entrant dans une place et ceux sortant d'une place

Nous avons donc créé une sous-classe abstraite, **ArcSortant**, pour modéliser les arcs sortants des places vers les transitions. Cette classe implémente les deux méthodes de base (add_arc_to_transition et remove_arc_from_transition), et introduit deux nouvelles méthodes abstraites : arcIsFireable, pour vérifier si l'arc permet à la transition d'être tirable, et update_jetons_place, pour ajuster le nombre de jetons dans les places. La classe ArcSortant a ensuite été subdivisée en trois sous-classes concrètes :

- **ArcZero** : Ce type d'arc est caractérisé par l'attribut active et ne permet à la transition de se déclencher que si la place associée contient zéro jeton. Les méthodes arcIsFireable et update_jetons_place sont basées sur cet attribut active.

- **ArcVideur** : Ce type d'arc vide entièrement la place associée lorsqu'il est activé. L'attribut active permet de vérifier si la place doit être vidée, et update_jetons_place met à jour le nombre de jetons à zéro.

- **ArcSortantNormal** : C'est le type d'arc standard qui possède un poids. arcIsFireable compare le nombre de jetons dans la place au poids de l'arc, et update_jetons_place ajuste le nombre de jetons en conséquence.

Pour les **arcs entrants**, nous avons implémenté la classe ArcEntrant, qui gère le transfert de jetons vers une transition, en se basant sur le poids requis. Cette structure permet de distribuer les responsabilités de manière claire entre les arcs et les transitions. La classe Transition devient ainsi plus simple, avec des méthodes isTirable et fire qui délèguent la majorité du travail aux arcs.

Avec cette nouvelle architecture, nous avons mieux organisé la répartition des rôles entre les différentes classes. Désormais, si nous devons ajouter un nouveau type d'arc, il suffit de créer une nouvelle sous-classe de ArcSortant ou de ArcEntrant sans affecter le reste du modèle. Cette solution rend le système plus modulaire, extensible, et plus simple à maintenir, tout en facilitant les modifications futures en cas de changement des règles métier.