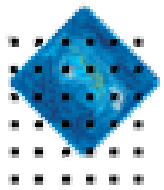
Manuel d'Utilisation E-NetObject : Un Editeur de Réseaux de Petri à Objets

Frédéric Raclot, David Andreu, Thérèse Libourel Rouge, Robin Passama



Manuel d'utilisation
E-NetObject

Un éditeur de Réseaux de Petri à Objets

F. Raclot[1], D. Andreu[2], T. Libourel[3], R. Passama[2,3]

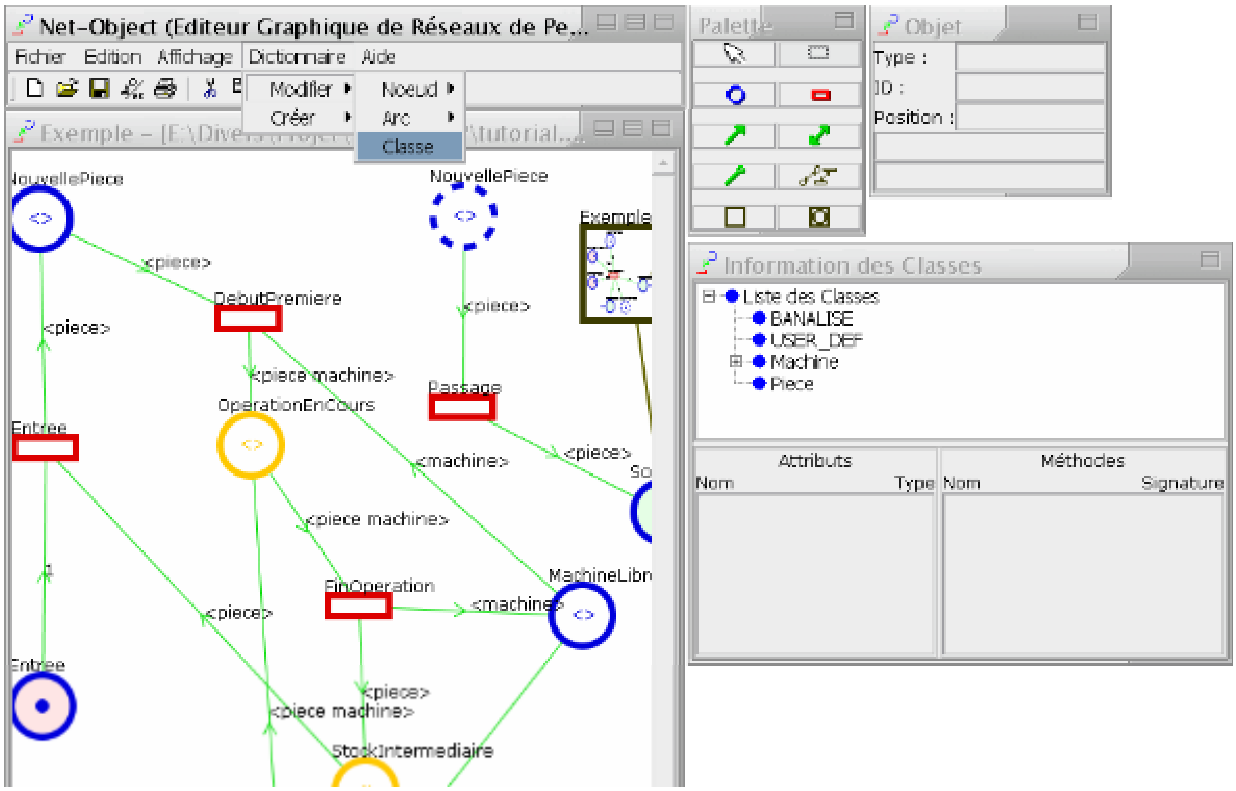
Rapport LIRMM n° 02181

[1] DESS TNI, UMII
[2] LIRMM, département Robotique
[3] LIRMM, département Acquisition et Représentation des Connaissances

1. Présentation

E-NetObject est un éditeur graphique de réseaux de Petri à objets écrit en Java. Il fait partie d'une suite d'applications. Il vous permettra de créer et d'éditer les réseaux de Petri les plus complexes. Voici une liste non-exhaustive de ses caractéristiques :

- Environnement d'édition des plus complets : Création de la plupart des éléments d'un RdP classique (Place, Transition, Arc, Arc Inhibiteur, Arc de Fusion, Sous-Réseau ...), des jetons-objets d'un réseau de Petri à Objets (Classes, Objets, Arborescence des Classes, réutilisation, etc.), l'association de fonctions au modèle ou aux objets (méthodes), la duplication de place (duplication au sens graphique)...
- Gestion des modèles génériques.
- Génération XML d'un réseau (utilisée par les autres applications de E-NetObject).
- Librairies de Classes.
- Et toutes les caractéristiques d'un éditeur d'objets graphique: Couper/Copier/Coller, Supprimer, Annuler/Rétablir, Imprimer (sous forme graphique ou textuelle), Zoom, etc.



Cette aide, accessible directement au sein de l'éditeur, se compose de plusieurs parties :

- L'Editeur : décrit et explique les fonctionnalités de E-NetObject.
- Les Boîtes de Dialogue : vous apprend à utiliser les fenêtres de l'éditeur.
- Les Objets : explique en détail la création/modification d'un objet du réseau.
- Tutorial : apprentissage de la création de deux réseaux de Petri, l'un à Objets et l'autre banalisé.

Note : Ce rapport est un manuel d'utilisation de l'éditeur E-NetObject. Le formalisme du modèle édité, la sémantique des éléments du modèle sont présentés dans le rapport [RR02180]. Le noyau d'exécution de ce modèle est présenté dans le rapport [RR02182].

2. L'éditeur

2.1. La barre de menu

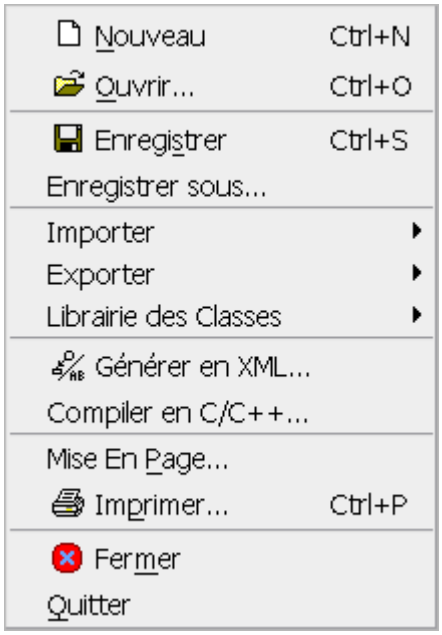
Cette barre de menu réunit les principales fonctions de l'Editeur :

Figure 1

- Fichier : contient les fonctions de manipulation de fichier et l'éditeur.
- Edition : contient les fonctions d'édition de l'éditeur.
- Affichage : permet d'afficher/cacher la fenêtre d'information ou la palette graphique ainsi que les noms des Objets.
- Dictionnaire : contient les fonctions de modification/création de tous les objets du Réseau.
- Aide : permet d'ouvrir la fenêtre 'A Propos'.

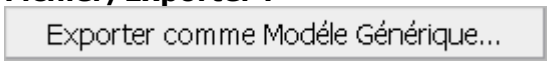
2.1.1. Fichier

Fichier :



Fichier/Importer :

Fichier/Exporter :



Librairie des Classes :

Fichier :

- Nouveau : ouvre un nouveau Réseau de Petri.
- Ouvrir… : ouvre un Réseau de Petri sauvegardé.

- Enregistrer : sauvegarde le Réseau de Petri courant.
- Enregistrer sous… : sauvegarde le Réseau de Petri courant sous un autre nom.

Fichier/Importer :

- Importer un Modèle Générique… : permet de modifier un Modèle Générique.

Fichier/Exporter :

- Exporter comme Modèle Générique… : permet de sauvegarder le réseau courant en tant que Modèle Générique.

Fichier/Librairie :

- Changer de librairie… : change la librairie du réseau courant.
- Importer une classe… : importe une ou plusieurs classes de la librairie dans le réseau courant.
- Supprimer une classe… : supprime une ou plusieurs classes de la librairie.
- Mettre à jour la librairie… : met à jour la librairie de classe à partir du réseau courant.
- Vider la librairie : supprime toutes les classes contenues dans la librairie.

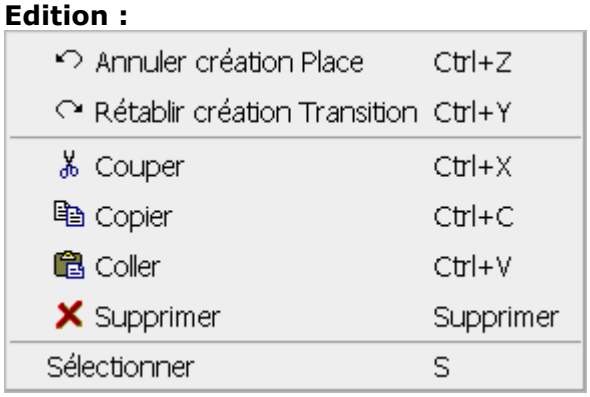
- Générer en XML… : génère le Réseau sous forme XML.
- Compiler en C/C++ : génère une image C/C++ du réseau courant.

- Mise En Page… : permet de mettre en page le Réseau courant.
- Imprimer… : imprime le Réseau courant.

- Fermer : ferme le Réseau courant.
- Quitter : ferme l'éditeur.

2.1.2. Edition

Edition :



Edition :

- Annuler : annule la dernière action.
- Rétablir : rétablit la dernière action annulée.

- Couper : copie et supprime la sélection.
- Copier : copie la sélection.
- Coller : colle le contenu du 'presse-papier'.
- Supprimer : supprime la sélection.

- Sélectionner : sélectionne l'outil de sélection.

2.1.3. Affichage

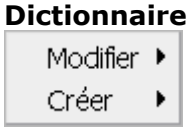
Affichage :

Affichage :

- Palette : cache ou affiche la palette graphique.
- Information : cache ou affiche la fenêtre d'information.
- Classes : cache ou affiche la fenêtre d'information des classes.
- Nom des Objets : active/désactive l'affichage des noms des entités du réseau.
- Réseau Miniature : active/désactive l'affichage des représentations miniatures des sous-réseaux.
- Zoom : permet de choisir une échelle pour le réseau courant allant de 50% à 100% avec un pas de 10pts.

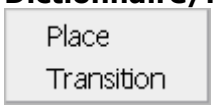
2.1.4. Dictionnaire

Dictionnaire :

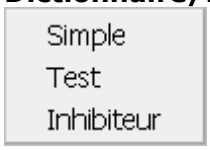


Dictionnaire/Modifier :

Dictionnaire/Modifier/Nœud :



Dictionnaire/Modifier/Arc :



Dictionnaire/Créer :

Dictionnaire :
Dictionnaire/Modifier/Noeud :

- Place : ouvre une fenêtre permettant de sélectionner une Place à modifier.
- Transition : ouvre une fenêtre permettant de sélectionner une Transition à modifier.

Dictionnaire/Modifier/Arc :

- Simple : ouvre une fenêtre permettant de sélectionner un Arc Simple à modifier.
- Test : ouvre une fenêtre permettant de sélectionner un Arc de Test à modifier.
- Inhibiteur : ouvre une fenêtre permettant de sélectionner un Arc Inhibiteur à modifier.

Dictionnaire/Modifier/Classe :

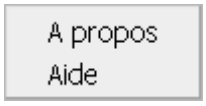
- Classe : ouvre une fenêtre permettant de sélectionner une Classe à modifier.

Dictionnaire/Créer :

- Classe : crée une Classe.
- Verrou : crée un Verrou.

2.1.5. Aide

Aide :



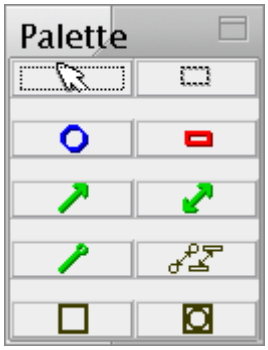
Aide :

- A propos : affiche une fenêtre d'information sur l'Editeur.
- Aide : affiche ce manuel au format HTML.

2.2. La barre d'outils

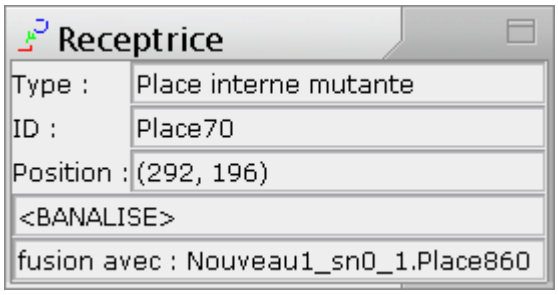
: ouvre un nouveau Réseau de Petri.
: ouvre un Réseau de Petri sauvegardé.
: sauvegarde le Réseau de Petri courant.
: génère le Réseau sous forme textuelle.
: imprime le Réseau courant.
: copie et supprime la sélection (coupe).
: copie la sélection.
: colle le contenu du 'presse-papier'.
: supprime la sélection.
: annule la dernière action.
: rétablit la dernière action annulée.
: ferme le réseau courant.

2.3. La palette graphique



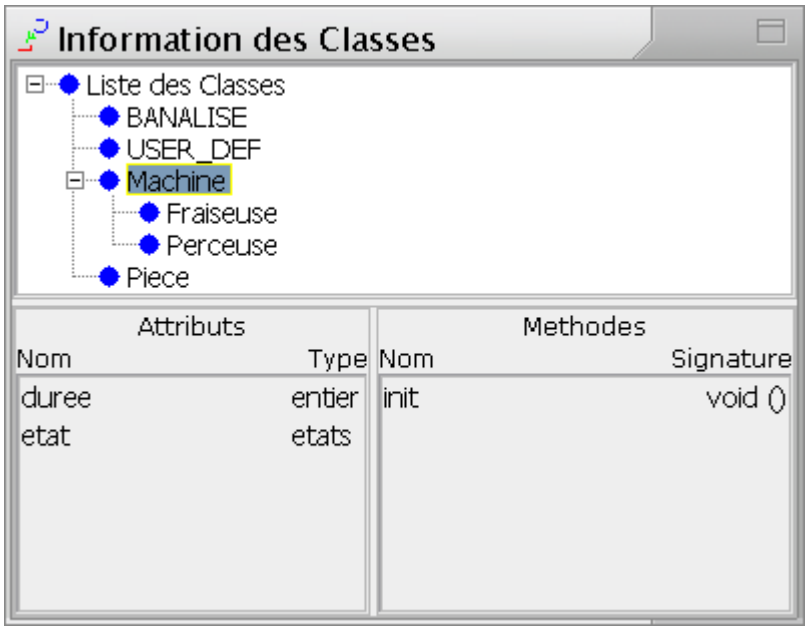
: ne fait rien
: sélectionne l'outil de sélection.
: sélectionne l'outil de création de Place.
: sélectionne l'outil de création de Transition.
: sélectionne l'outil de création d'Arc Simple.
: sélectionne l'outil de création d'Arc de Test.
: sélectionne l'outil de création d'Arc Inhibiteur.
: sélectionne l'outil de création d'Arc de Fusion.
: sélectionne l'outil de création de Sous-Réseau.
: sélectionne l'outil de création de Macro Place (pas encore implémenté).

2.4. La fenêtre d'information



- Barre de titre : affiche le nom de l'objet pointé (si possible).
- Type : affiche le type de l'objet pointé.
- ID : affiche l'identifiant l'objet pointé.
- Position : affiche la position de l'objet pointé.
- Infos supplémentaires 1 : affiche des informations supplémentaires en fonction de l'objet pointé.
- Infos supplémentaires 2 : affiche des informations supplémentaires en fonction de l'objet pointé (en général à propos des réseaux génériques).

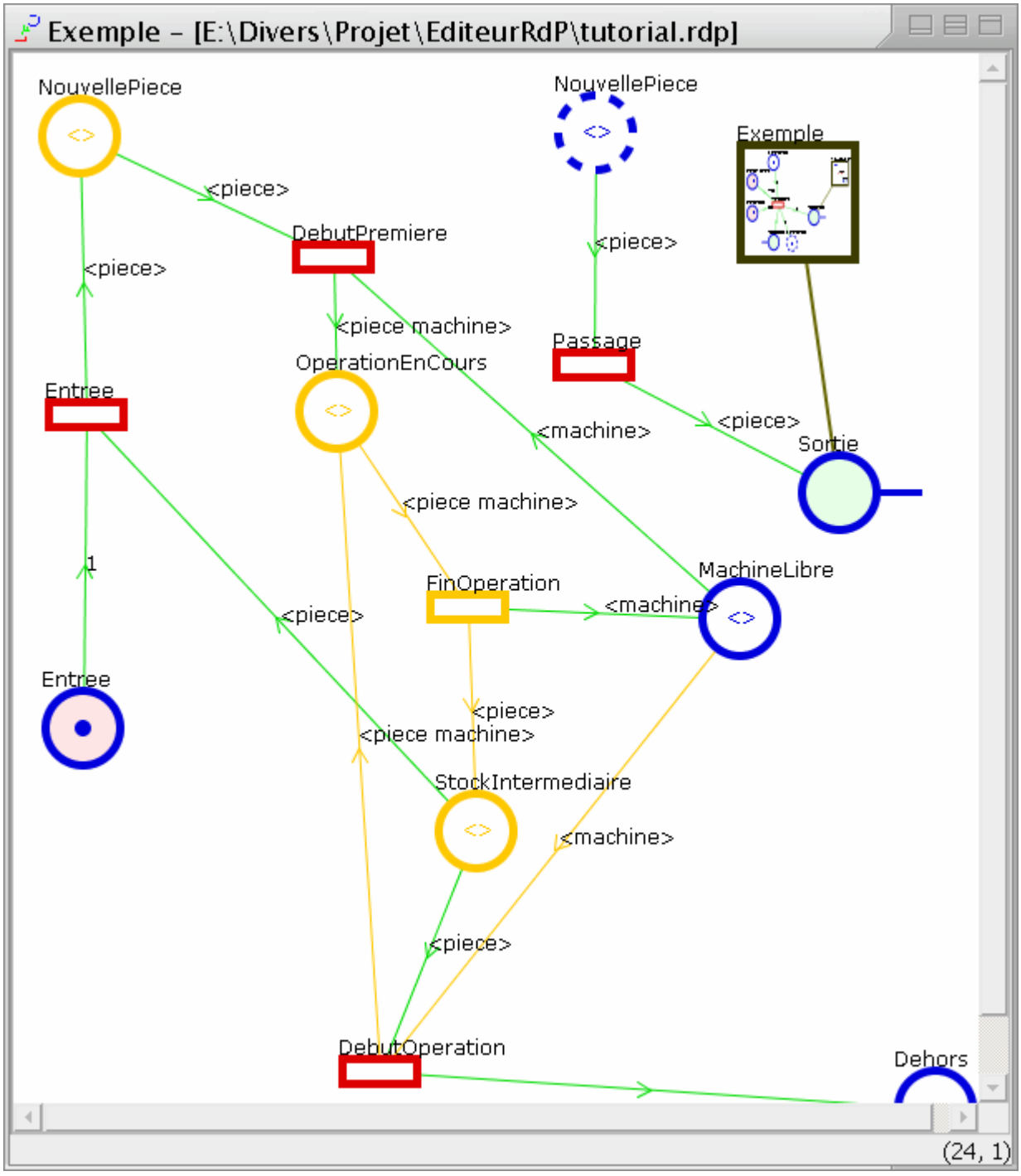
2.5. La fenêtre d'information des classes



En cliquant sur le nom d'une classe de l'arbre les informations concernant les attributs et les méthodes s'afficheront en bas.

En double-cliquant sur le nom d'une classe vous pourrez modifier ses propriétés.

2.6. La fenêtre d'édition



- Barre de Titre : affiche le nom et le chemin de sauvegarde du réseau.
- Barre de statut : à droite, affiche les coordonnées du pointeur; à gauche, affiche certaines informations.

- Description :

Ce réseau est composé de:

- 15 Places distinctes (ronds bleus) (dont 8 dans les Sous-Réseaux).
- 7 Transitions (rectangles rouges) (dont 2 dans les Sous-Réseaux).
- 23 Arcs Simples (flèches vertes) (dont 8 dans les Sous-Réseaux).
- 0 Arcs Inhibiteurs (flèches vertes avec un rond vert au milieu).
- 0 Arcs de Test (doubles flèches vertes)
- 2 Arcs de Fusion (flèches brunes) (dont 2 dans les Sous-Réseaux).
- 2 Sous-Réseaux (grands carrés bruns) (dont 1 dans un Sous-Réseau).

Des barres de défilement apparaissent dans ce cas car la Place "Dehors" est hors du cadre en bas à droite.

- La Place en pointillé représente un "clone" (une duplication graphique) de la Place "NouvellePiece" en haut à gauche. Conceptuellement ces deux Places n'en représentent qu'une. De même un Arc de Fusion est utilisé entre deux Places pour les fusionner, ainsi si 4 Places sont liées par des Arcs de Fusion cet ensemble n'en représentera qu'une. Dans les deux cas (une Place et ses clones, Places liées par des Arcs de Fusion) le nouvel ensemble des Arcs sera l'union de l'ensemble des Arcs des Places fusionnées ou clonées.

- Les Places d'entrée/sortie ont un fond respectivement rouge/vert.

- Les éléments en orange, sont les éléments sélectionnés par l'utilisateur. La création d'un nouvel objet induit automatiquement la sélection de ce nouvel objet (i.e. l'ensemble des objets précédemment sélectionnés ne l'est plus).

3. Les boîtes de dialogue

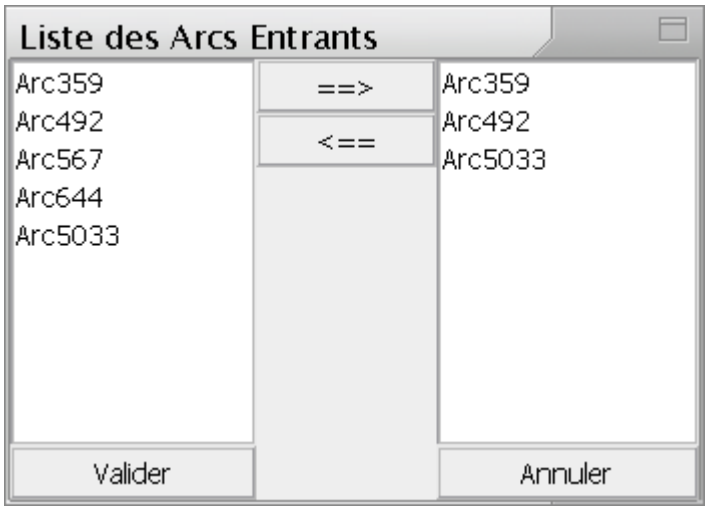
3.1. SelectList

Cette boîte permet de sélectionner un ou plusieurs éléments dans une liste donnée :

Sélectionnez les éléments en les cliquant, pour désélectionner un élément cliquez dessus en appuyant sur la touche 'Ctrl'. Puis validez ou annulez. Dans la plupart des cas, si vous ne sélectionnez aucun élément, vous en créerez un nouveau.

3.2. ModifList

Cette boîte de dialogue permet de modifier les éléments d'une liste en utilisant des éléments d'une autre liste :

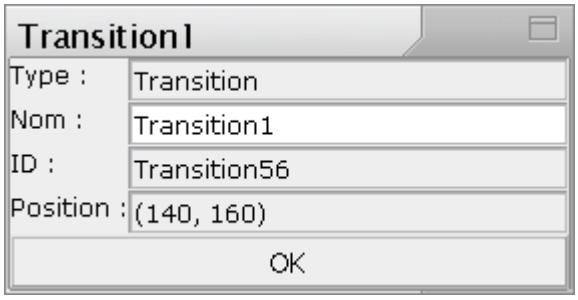


La liste de gauche représente les éléments disponibles, la liste de droite représente la liste à modifier.

==> : rajoute un élément sélectionné à gauche dans la liste de droite.
<== : enlève l'élément sélectionné à droite de la liste de droite.

3.3. PopInformation

Cette boîte de dialogue affiche quelques informations communes à tous les objets, on l'obtient en cliquant droit sur un objet et en cliquant sur 'Information', si on clique droit ailleurs que sur un objet vous pourrez obtenir des informations sur le Réseau.



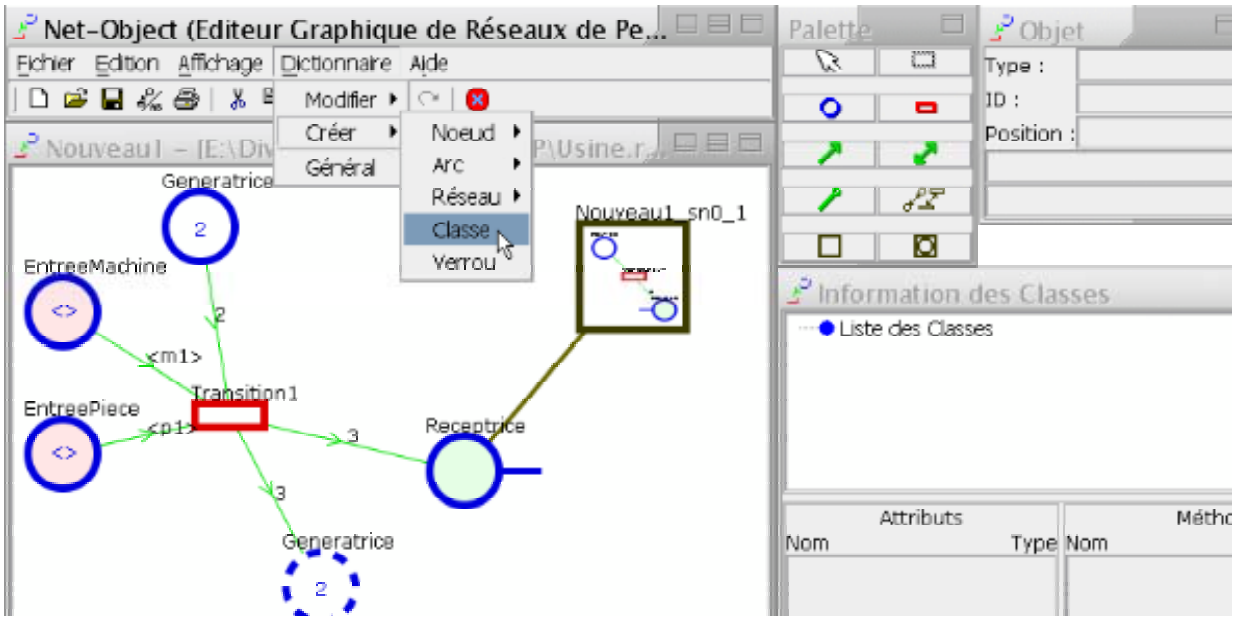
Type : affiche le type de l'objet.
Nom : affiche le nom de l'objet.
ID : affiche l'identifiant l'objet.
Position : affiche la position de l'objet.
OK : valide le changement éventuel du Nom.

4. Les entités

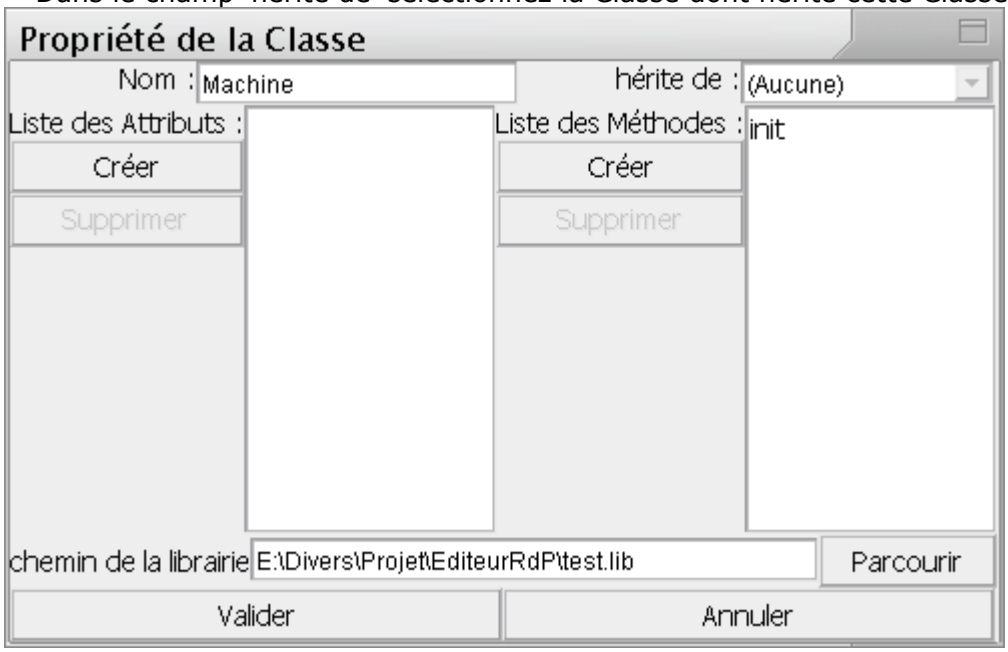
4.1. Classe

4.1.1. Création

Dans le menu 'Dictionnaire', sélectionnez 'Créer/Classe' :

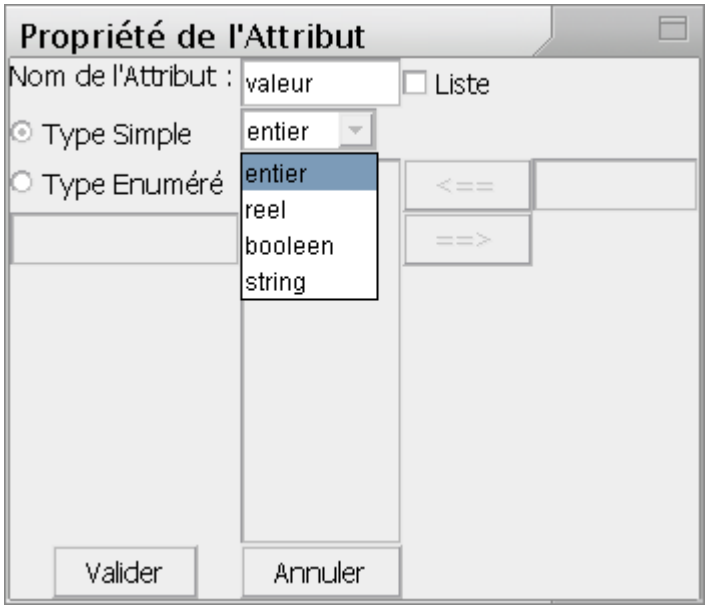


- Dans le champ 'Nom' entrez le nom de la Classe,
- Dans le champ 'hérite de' sélectionnez la Classe dont hérite cette Classe :

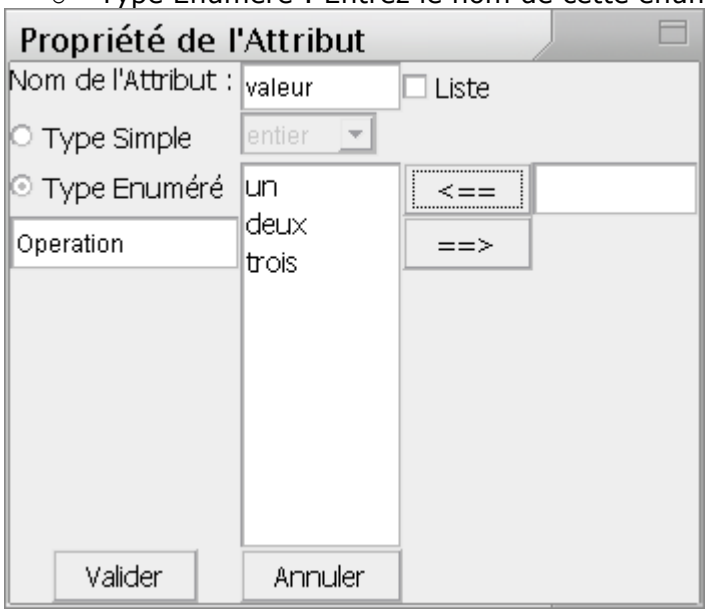


- Dans la zone des attributs (à gauche), cliquez sur 'Créer' et vous obtiendrez la fenêtre ci-dessous, comme pour la Classe, entrez un nom pour l'attribut. Choisissez entre 'Type Simple' et 'Type Enuméré' :

  - Type Simple : choisissez le type de votre attribut, entier, réel, booléen, string ou un type défini précédemment.



- Type Enuméré : Entrez le nom de cette énumération, puis entrez les valeurs.

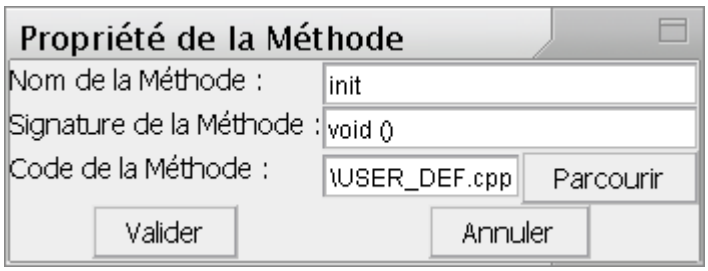


- Pour chaque type d'attribut (simple ou énuméré) vous pouvez spécifier si c'est une liste ou non, en cochant la case en haut à droite. Une liste est un type structuré défini par un multi-ensemble ordonné d'éléments du même type.
- Dans la zone des méthodes (à droite), cliquez sur 'Créer' et vous obtiendrez la fenêtre ci-dessous, comme pour la Classe, entrez un nom pour la méthode.

Entrez la signature, exemples :

```
entier (booleen, liste de string)
liste de operation (reel, liste de string)
```

Appuyer sur le bouton 'Parcourir', pour entrez le chemin du fichier contenant le code de cette méthode.



Recommencez pour créer d'autres attributs/méthodes.

4.1.2. Modification

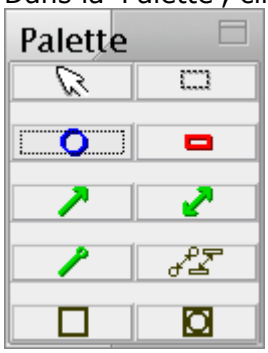
Procédez de la même manière, mais en sélectionnant l'attribut/méthode que vous voulez éditer/supprimer.

Remarque : pour désélectionner un objet de la liste, appuyez sur Ctrl et cliquez sur l'élément.

4.2. Place

4.2.1. Création

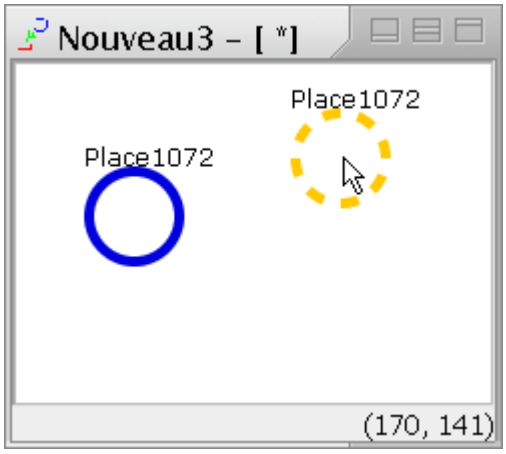
Dans la 'Palette', cliquez sur l'icône de création de Place :



Cliquez sur le Réseau pour créer la Place.

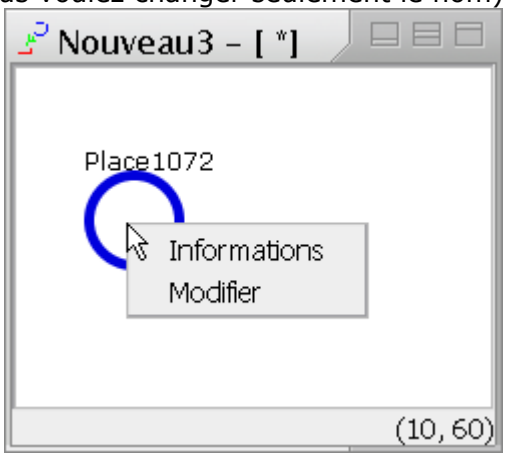
4.2.2. Création d'un Clone

Déplacez la Place voulue avec le bouton droit.



4.2.3. Modification

Cliquez droit sur la Place souhaitée puis cliquez sur 'Modifier' (ou sur 'Information' si vous voulez changer seulement le nom) :

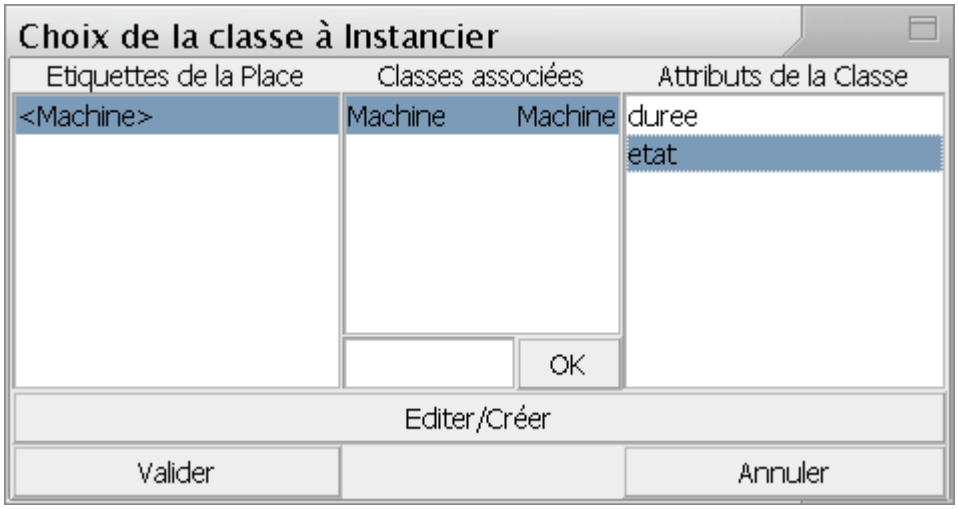


Vous obtiendrez cette fenêtre :

- En appuyant sur l'un des deux premiers boutons, vous pourrez modifier la liste des Arcs Entrants/Sortants de la Place. Attention! Nous vous déconseillons d'ajouter un Arc dans l'une des deux listes, sans en mesurer l'impact sur le modèle édité.

- Si vous cliquez sur le bouton 'Modifier les classes de la Place' vous pourrez définir quels sont les n-uplets de Classes que la Place accepte (voir ModifListe et SelectListe).

- Enfin, le dernier bouton vous permet de modifier le marquage initial de la Place :



A gauche se trouve la liste des classes admises par la Place. Si vous sélectionnez un élément dans la liste de gauche, le n-uplet associé s'affichera dans la liste du milieu, cette liste est séparée en deux colonnes dans celle de gauche, c'est le nom de la Classe, à droite c'est le nom de la marque associée à la Classe. La liste de droite affiche la liste des attributs correspondant à la Classe sélectionnée dans la liste du milieu.

Vous pouvez changer le nom de la marque en sélectionnant l'élément en entrant le nom dans le champ du milieu et cliquez sur OK. Sinon le nom par défaut sera le nom de la Classe en minuscule.

Si vous voulez initialiser les jetons-objets du marquage de la Place, sélectionnez un élément dans chaque colonne puis cliquez sur 'Editer/Créer', une boîte de dialogue spécifique à l'attribut s'ouvrira et vous pourrez choisir la valeur de l'attribut du jeton-objet sélectionné.

Si vous voulez utiliser les valeurs par défaut, sélectionnez un n-uplet à gauche et cliquez sur 'Valider'.

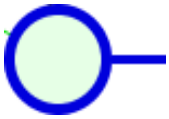
Pour ajouter un jeton sélectionnez <BANALISE> en haut à gauche puis Validez.

4.2.4. Notes

Un rond au centre de la Place apparaît lorsqu'un jeton a été posé dans la Place, si plus d'un jeton a été mis dans la Place le nombre de jetons s'affichera au centre de la Place.

Si la marque n'est pas de type <BANALISE> alors "**<>**" s'affichera au centre de la Place.

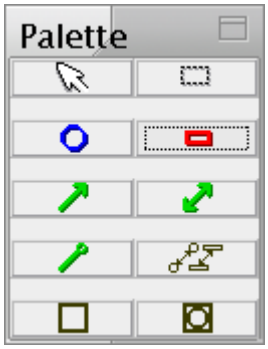
Une courte flèche entrante ou sortante de la Place, signifie que cette Place est liée par un Arc de Fusion à une autre Place.



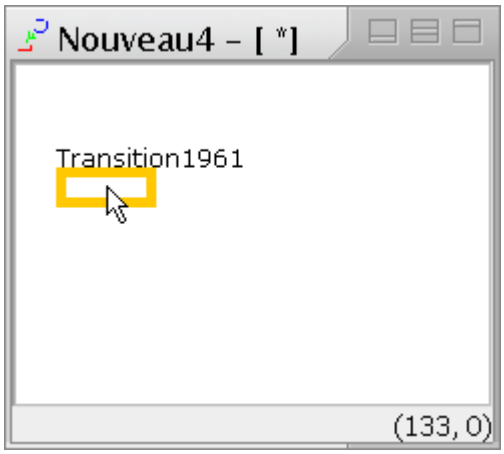
4.3. Transition

4.3.1. Création

Dans la 'Palette', cliquez sur l'icône de création de Transition :

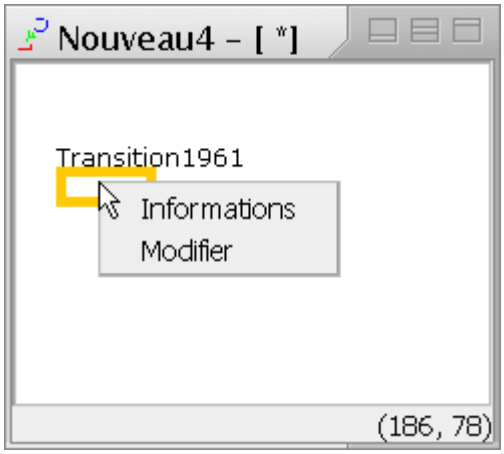


Cliquez sur le Réseau pour créer la Transition.



4.3.2. Modification

Cliquez droit sur la Transition souhaitée puis cliquez sur 'Modifier' (ou sur 'Information' si vous voulez changer seulement le nom) :

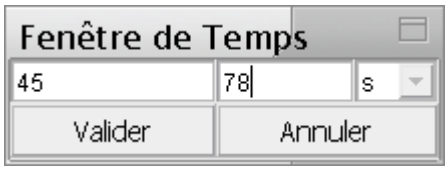


Vous obtiendrez cette fenêtre :

Dans le champ 'Nom' vous pouvez changer le nom de la Transition.
Dans le champ 'Verrou' vous pouvez sélectionner (associer) un verrou pour cette Transition.
Dans le champ 'Priorité' vous pouvez choisir la priorité de cette Transition.

- En appuyant sur l'un des deux premiers boutons, vous pourrez modifier la liste des Arcs Entrants/Sortants de la Transition.

- Si vous cliquez sur le bouton 'Modifier la fenêtre de Temps' vous pourrez définir l'intervalle de la fenêtre temporelle :



- Si vous cliquez sur le bouton 'Modifier les Prédicats' vous pourrez définir les conditions pour cette Transition:

Vous devez remplir les deux champs avec des expressions valides du type :

```
[constante]
[nom_variable].[nom_attribut]
[nom_variable].[nom_methode]()
[nom_variable].[nom_methode]([expression], ...)
```

```
[nom_methode_de_USER_DEF]()
```
Evidemment les types doivent être compatibles. Vous ne pouvez pas entrer une expression de type opération, pour faire par exemple :
```
c.monEntier inf 4+5
```
vous devrez utiliser une méthode :
```
c.monEntier inf PLUS(4, 5)
```
Note : les constantes booléennes sont **true** et **false**. Le symbole décimal pour les réels est le point et non pas la virgule. Les chaînes de caractères sont délimitées par **"**. Les listes sont délimitées par des parenthèses, et les éléments sont séparés par une virgule.

- Si vous cliquez sur le bouton 'Modifier les Actions' vous pourrez définir les actions pour cette Transition:

- Appel à un opérateur :

Le premier champ doit être du type :
```
[nom_variable].[nom_attribut]
```
le ou les autres champ(s) (selon que l'on coche 'Expression' ou non) doivent être du type :
```
[constante]
[nom_variable].[nom_attribut]
[nom_variable].[nom_methode]()
[nom_variable].[nom_methode]([expression], ...)
[nom_methode_de_USER_DEF]()
```
Evidemment les types doivent être compatibles. Donc pour modifier la valeur d'un attribut il faut procéder comme sur la capture d'écran.

- Appel à une méthode :

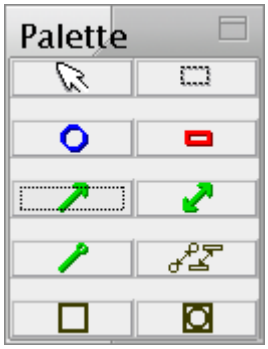
Doit être du type :
```
[nom_variable].[nom_methode]()
[nom_variable].[nom_methode]([expression], ...)
[nom_methode_de_USER_DEF]()
```
Evidemment, dans ce dernier cas, il est conseillé que la méthode soit une méthode de type void, car le résultat sera perdu.

Note : les constantes booléennes sont **true** et **false**. Le symbole décimal pour les réels est le point et non pas la virgule. Les chaînes de caractères sont délimitées par **"**. Les listes sont délimitées par des parenthèses, et les éléments sont séparés par une virgule.

4.4. Arc Simple

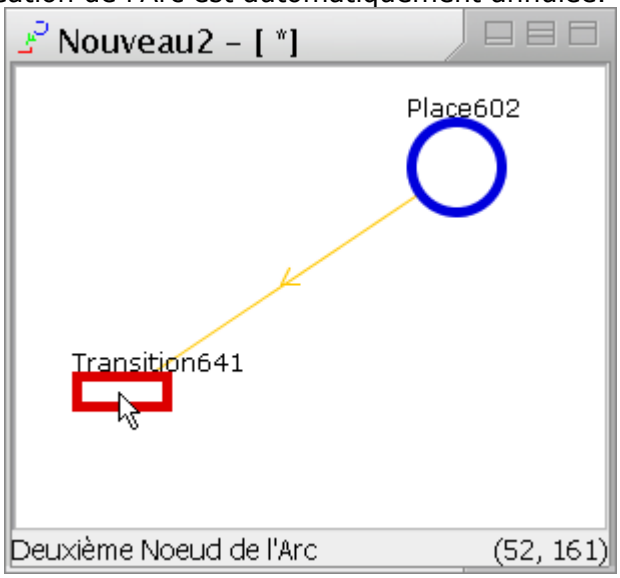
4.4.1. Création

Tout d'abord vous devez avoir créé une Place et une Transition. Puis dans la 'Palette', cliquez sur l'icône de création d'Arc Simple :



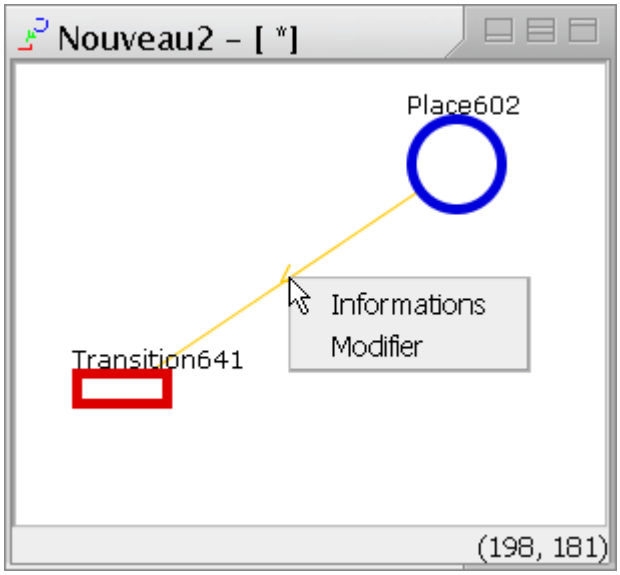
Cliquez sur le premier Noeud (ici une Place).

Puis sur le deuxième (ici une Transition) pour créer l'Arc Simple, en relâchant le bouton entre les deux clics. Si entre-temps vous appuyez sur une portion vierge du réseau, la création de l'Arc est automatiquement annulée.



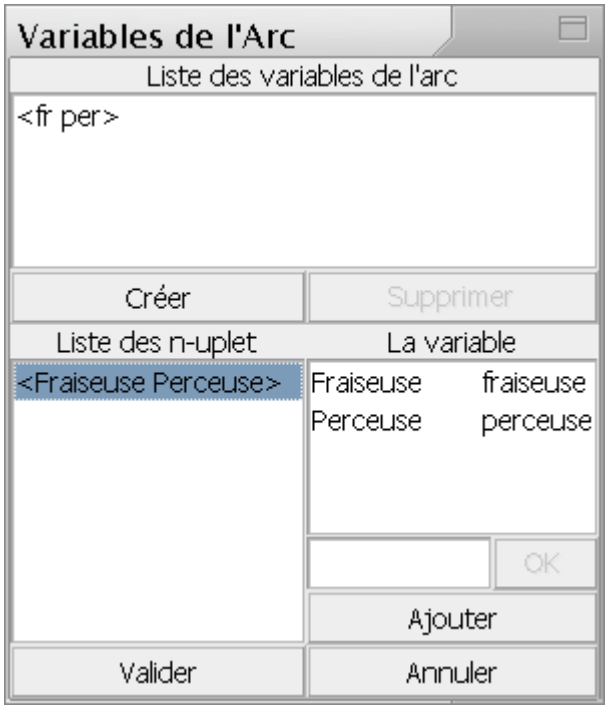
4.4.2. Modification

Cliquez droit sur l'Arc Simple souhaité puis cliquez sur 'Modifier' :



Vous obtiendrez cette fenêtre :

- Si vous cliquez sur le bouton 'Modifier la Liste des Variables' vous pourrez définir la liste des variables de l'Arc :



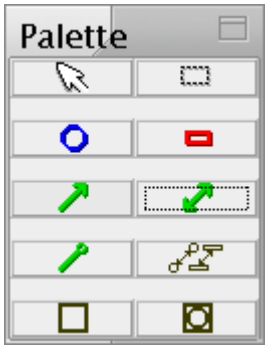
Cliquez sur 'Créer' pour afficher la liste des n-uplets acceptés par la Place amont ou aval. Vous pouvez ensuite sélectionner le n-uplet de votre choix et nommer la variable en

choisissant la Classe et le nom de la variable associée avec le bouton 'OK' et le champ qui est à gauche. Puis cliquez sur 'Ajouter' pour ajouter cette variable à la liste. Pour changer la pondération d'un Arc il suffit d'ajouter ou de supprimer des variables de type <BANALISE>, il n'est pas nécessaire de nommer ces variables.

4.5. Arc de Test

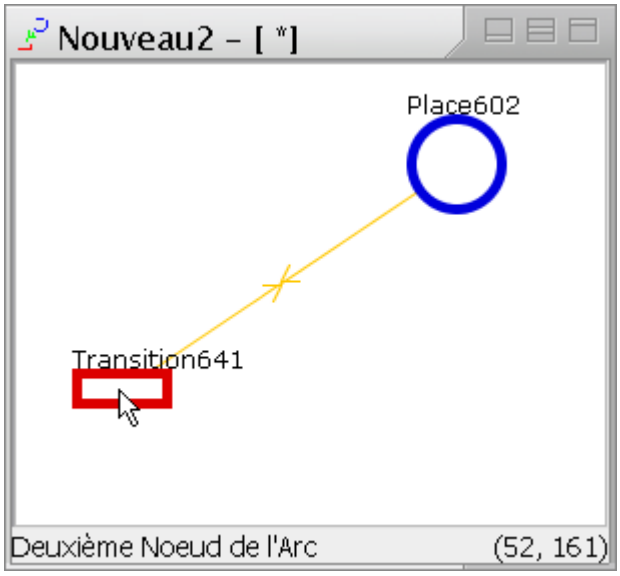
4.5.1. Création

Tout d'abord vous devez avoir créé une Place et une Transition. Puis dans la 'Palette', cliquez sur l'icône de création d'Arc de Test :



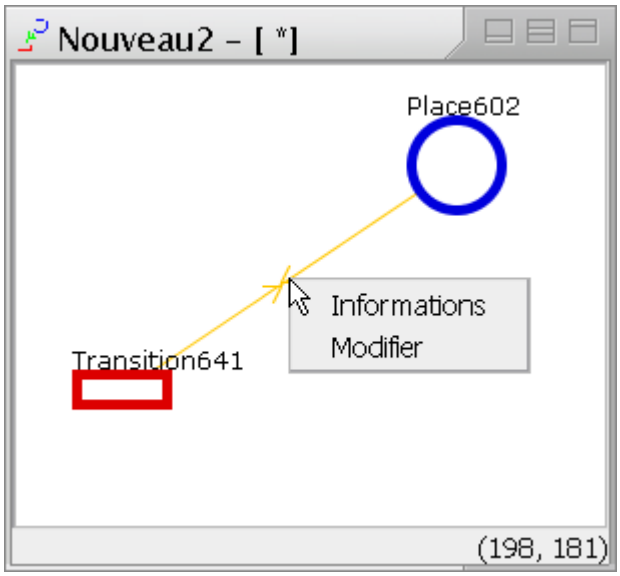
Cliquez sur le premier Noeud (ici une Place).

Puis sur le deuxième (ici une Transition) pour créer l'Arc de Test, en relâchant le bouton entre les deux clics. Si entre-temps vous appuyez sur une portion vierge du réseau, la création de l'Arc est automatiquement annulée.



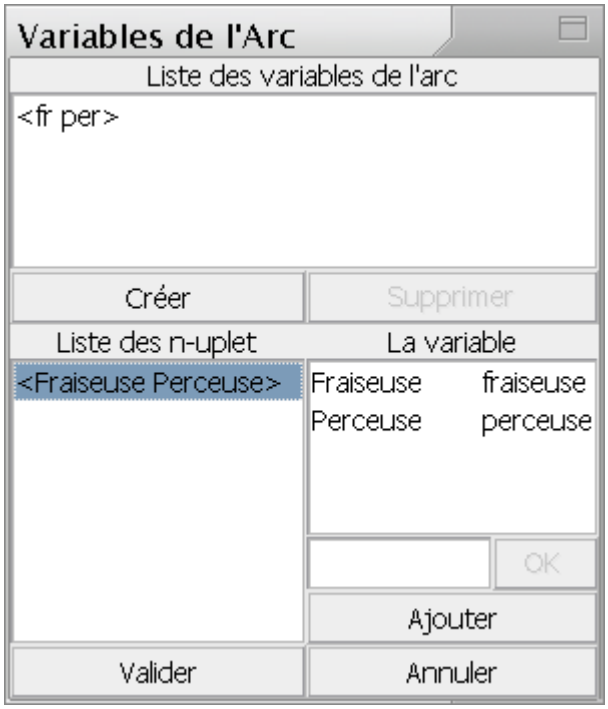
4.5.2. Modification

Cliquez droit sur l'Arc de Test souhaité puis cliquez sur 'Modifier' :



Vous obtiendrez cette fenêtre :

- Si vous cliquez sur le bouton 'Modifier la Liste des Variables' vous pourrez définir la liste des variables de l'Arc :

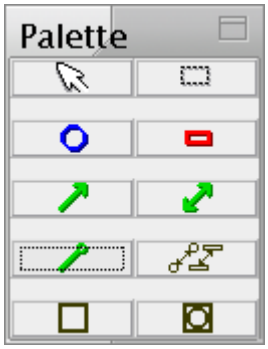


Cliquez sur 'Créer' pour afficher la liste des n-uplets acceptés par la Place amont ou aval. Vous pouvez ensuite sélectionner le n-uplet de votre choix et nommé la variable en choisissant la Classe et le nom de la variable associée avec le bouton 'OK' et le champ qui est à gauche. Puis cliquez sur 'Ajouter' pour ajouter cette variable à la liste. Pour changer la pondération d'un Arc il suffit d'ajouter ou de supprimer des variables de type <BANALISE>, il n'est pas nécessaire de nommer ces variables.

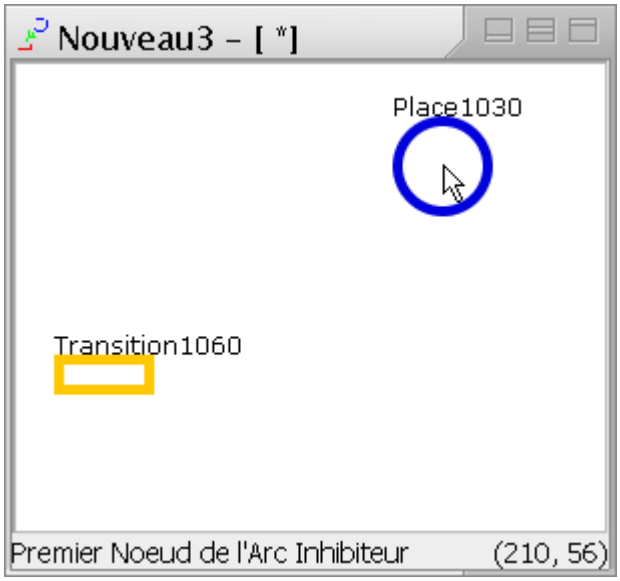
4.6. Arc Inhibiteur

4.6.1. Création

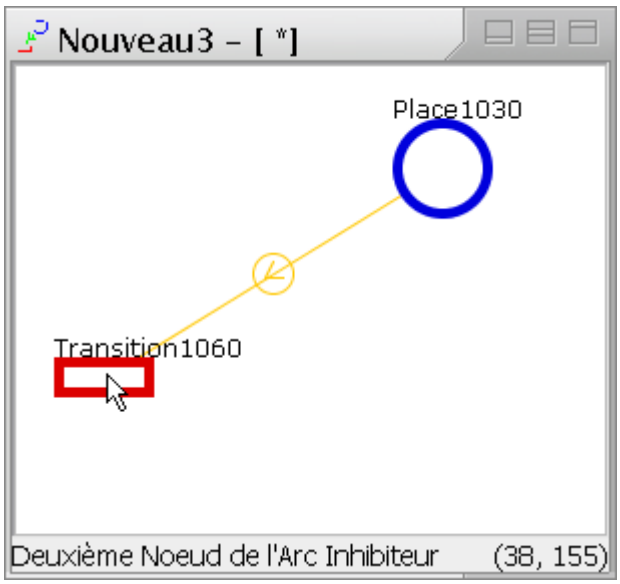
Tout d'abord vous devez avoir créé une Place et une Transition. Puis dans la 'Palette', cliquez sur l'icône de création d'Arc Inhibiteur :



Cliquez sur la Place.



Puis sur la Transition pour créer l'Arc Inhibiteur.

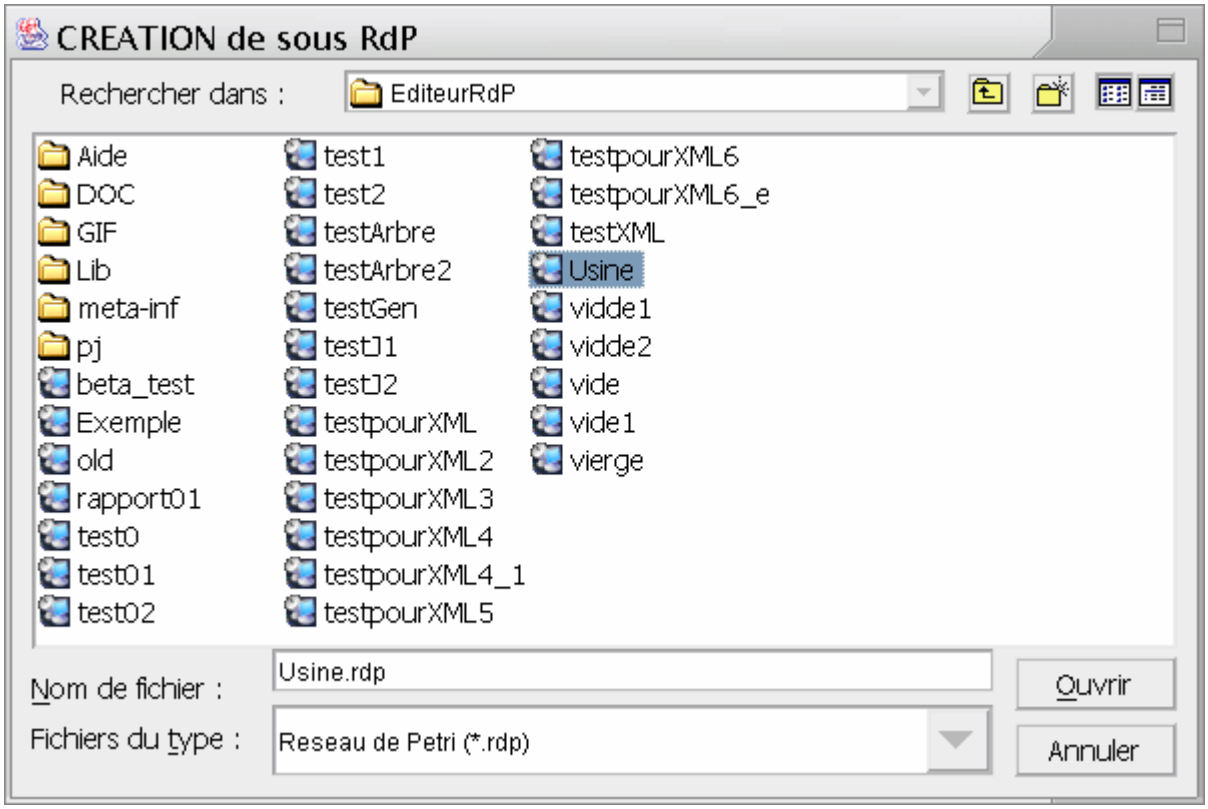


4.7. Sous-Réseau

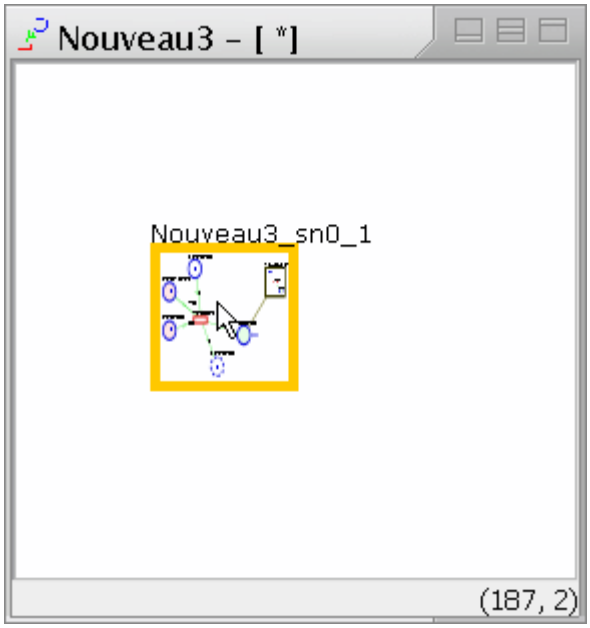
4.7.1. Création

Dans la 'Palette', cliquez sur l'icône de création de Sous-Réseau :

Cliquez sur le Réseau pour ouvrir la boîte de sélection de fichier.

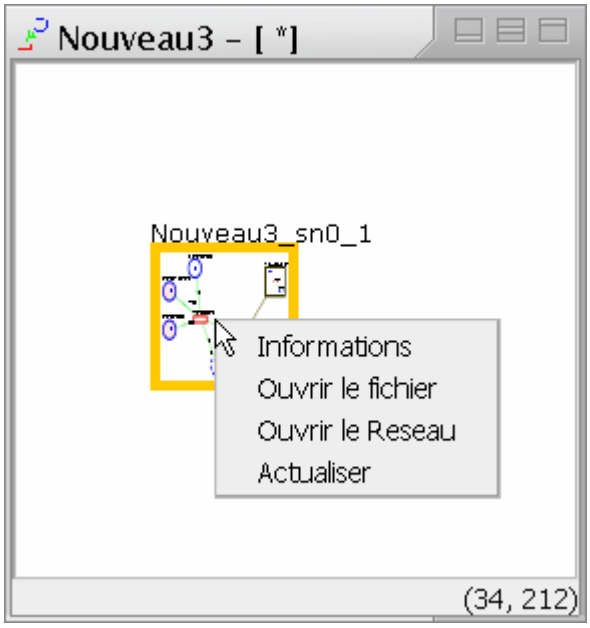


Cliquez sur 'Ouvrir' pour créer le Sous-Réseau.

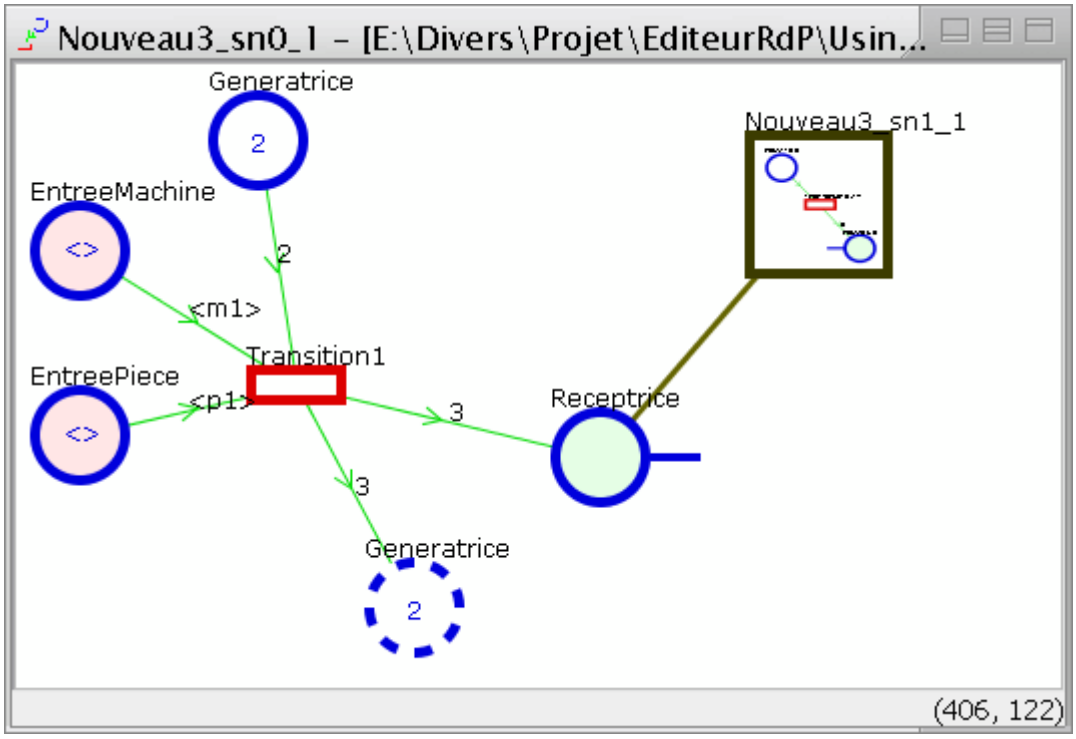


4.7.2. Modification

Cliquez droit sur le Sous-Réseau souhaité puis cliquez sur 'Ouvrir le Réseau' (ou sur 'Information' si vous voulez changer seulement le nom) :



Vous obtiendrez une fenêtre de Réseau semblable à la précédente.

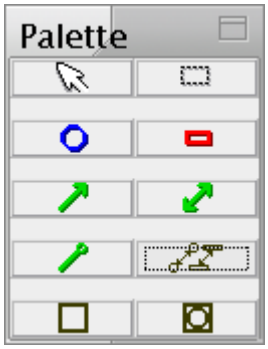


Fermez la fenêtre pour revenir au réseau précédent.

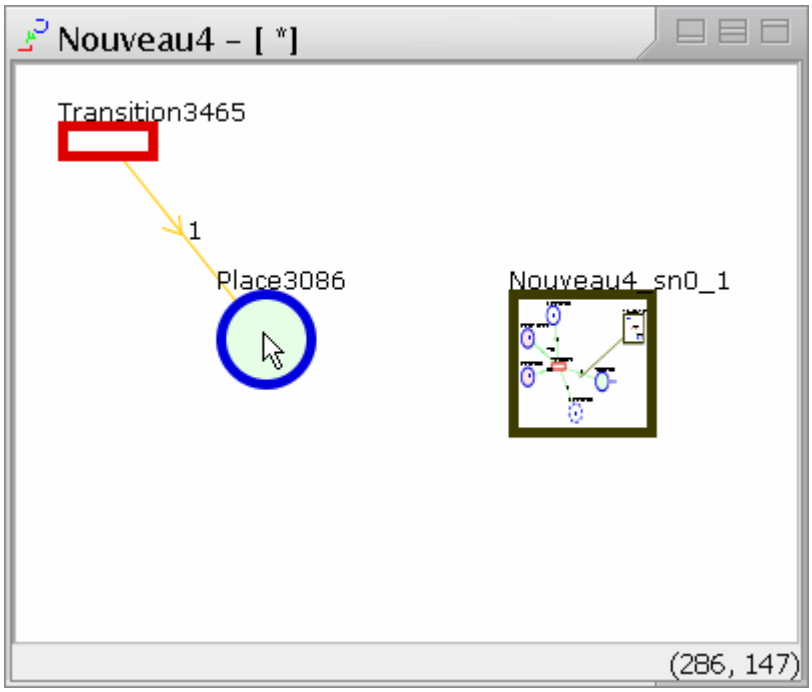
4.8. Arc de Fusion

4.8.1. Création

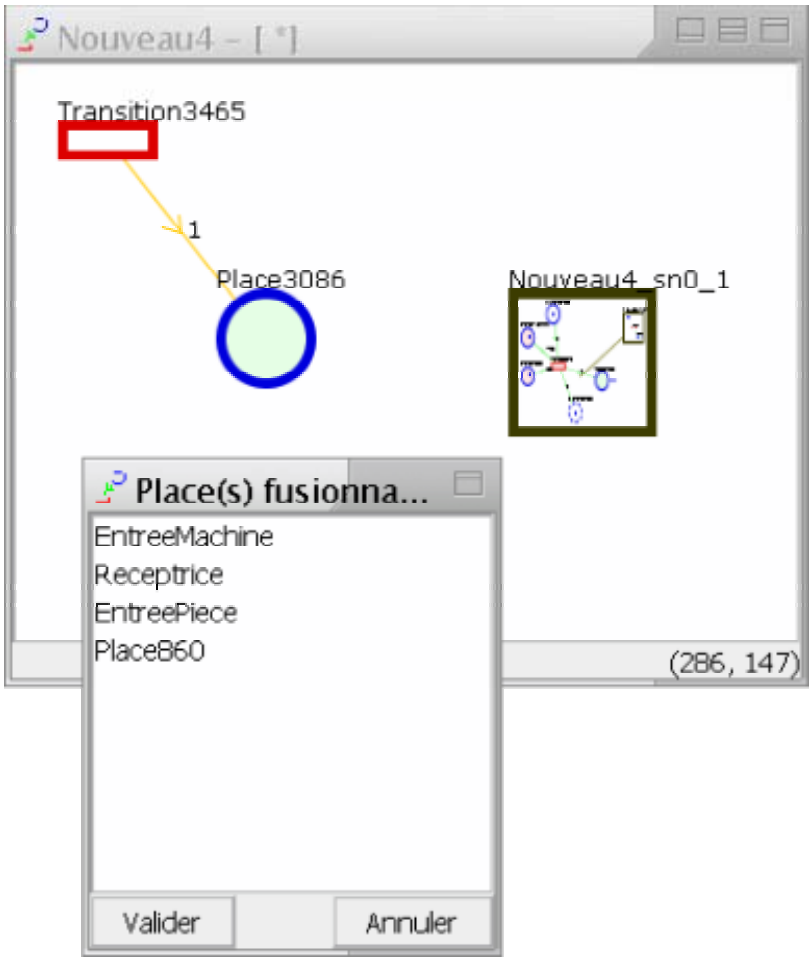
Tout d'abord vous devez avoir créé deux Places ou deux Transitions. Puis dans la 'Palette', cliquez sur l'icône de création d'Arc de Fusion :



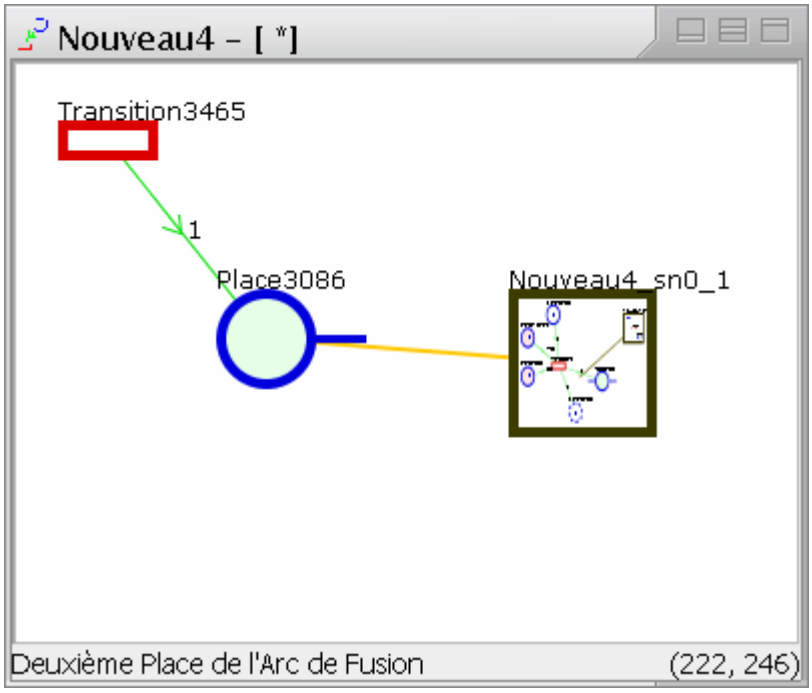
Cliquez sur le premier Noeud (ici une Place).



Puis sur le deuxième (ici un Sous-Réseau) pour ouvrir la boîte de sélection de la Place (du Sous-Réseau) avec laquelle sera faite la fusion.



Sélectionnez la Place voulue puis cliquez sur 'Valider' pour créer l'Arc de Fusion :



Cela se passe de manière symétrique pour les Transitions.

5. Documentation

5.1. Editeur de RdPO

5.1.1. Les entités d'un RdPO

Voici la description des éléments manipulés par l'interface :

a) *Super*

Cette classe est la classe mère de tous les objets manipulés par l'interface.

Attribut :

- `compteur` : Entier qui est incrémenté à chaque qu'une instance de Super (ou d'une de ses sous-classe) est créée.

Méthode :

`quoi()` : renvoie sous forme de chaîne de caractères le nom de la classe de l'objet.

b) *SuperLieu*

Cette classe abstraite est la classe mère de tous les éléments graphiques d'un réseau de Petri. Elle ne contient donc aucun Attribut ou Méthode significatif.

Attributs :

Méthodes :

c) *Lieu*

Cette classe abstraite est la classe mère des classes LieuArc et LieuBoite.

Attributs :

- `xA, yA, xB, yB` : définit les coordonnées des points A et B.

Méthodes :

- `translateDe(x, y)` : déplace le Lieu de `x` sur les abscisses et de `y` sur les ordonnées.
- `translateEn(x, y)` : positionne le point A en (x, y) et positionne le point B en fonction.
- `translateADe(x, y)` : déplace le point A de `x` sur les abscisses et de `y` sur les ordonnées.
- `translateAEn(x, y)` : positionne le point A en (`x`, `y`).
- `translateBDe(x, y)` : déplace le point B de `x` sur les abscisses et de `y` sur les ordonnées.
- `translateBEn(x, y)` : positionne le point B en (`x`, `y`).
- `appartient(x, y)` : détermine si le point (`x`, `y`) se trouve dans la zone définie par les points A et B.
- `appartient(xa, ya, xb, yb)` : détermine si le Lieu défini par les points (`xa`, `ya`) et (`xb`, `yb`) se trouve dans la zone définie par les points A et B.

d) *LieuArc*

Cette classe définit où se trouve un Arc, le point A est le point de départ de l'Arc et le point B est le point d'arrivée de cet Arc.

Attribut :

- `Erreur` : détermine la marge d'erreur lorsque l'on veut déterminer si une entité se trouve dans ce Lieu (voir les méthodes appartient).

Méthodes :

- `translateDe(x, y)` : déplace le Lieu de `x` sur les abscisses et de `y` sur les ordonnées.
- `translateEn(x, y)` : positionne le point A en (`x`, `y`) et positionne le point B en fonction.
- `translateADe(x, y)` : déplace le point A de `x` sur les abscisses et de `y` sur les ordonnées.
- `translateAEn(x, y)` : positionne le point A en (`x`, `y`).
- `translateBDe(x, y)` : déplace le point B de `x` sur les abscisses et de `y` sur les ordonnées.

- `translateBEn(x, y)` : positionne le point B en (`x`, `y`).
- `appartient(x, y)` : détermine si le point (`x`, `y`) se trouve entre les points A et B à Erreur près. Pour cela on calcule la distance AM entre les points A et (`x`, `y`), la distance BM entre les points B et (`x`, `y`) et la distance AB entre les points A et B. Si AM+BM est inférieur à AB+Erreur alors le point est considéré comme appartenant au LieuArc.
- `appartient(xa, ya, xb, yb)` : détermine si le Lieu définit par les points (`xa`, `ya`) et (`xb`, `yb`) se trouve dans la zone définie par les points A et B.

e) *LieuBoite*

Cette classe définit où se trouve un Nœud ou un sous-réseau, le point A définit le coin supérieur haut, et B définit le point inférieur droit, ceci est censé être assuré par l'objet lui-même.

Attributs :

Méthodes :

- `translateDe(x, y)` : déplace le Lieu de `x` sur les abscisses et de `y` sur les ordonnées.
- `translateEn(x, y)` : positionne le point A en (`x`, `y`) et positionne le point B en fonction.
- `translateADe(x, y)` : déplace le point A de `x` sur les abscisses et de `y` sur les ordonnées.
- `translateAEn(x, y)` : positionne le point A en (`x`, `y`).
- `translateBDe(x, y)` : déplace le point B de `x` sur les abscisses et de `y` sur les ordonnées.
- `translateBEn(x, y)` : positionne le point B en (`x`, `y`).
- `appartient(x, y)` : détermine si le point (`x`, `y`) se trouve dans le rectangle défini pas les points A et B.
- `appartient(xa, ya, xb, yb)` : détermine si le Lieu défini par les points (`xa`, `ya`) et (`xb`, `yb`) se trouve dans le rectangle défini par les points A et B.

f) *Arc*

Cette classe définit l'élément Arc du réseau :

Attributs :

- `identifiant` : l'identifiant de l'Arc sous forme de chaîne de caractères.
- `listeEtiquette` : la liste des étiquettes de l'Arc, sous forme de Vecteur d'Etiquette.
- `noeudDepart` : le nœud de départ de l'Arc sous forme de Noeud.
- `noeudArrive` : le nœud d'arrivée de l'Arc sous forme de Noeud.
- `lieu` : l'endroit où se trouve l'Arc sous forme de LieuArc.

Méthodes :

- `getType()` : donne le type de l'Arc sous forme de chaîne de caractères. Renvoie 'tp' si c'est un Arc de type Transition→Place, ou alors 'pt' si c'est un Arc de type Place→Transition.
- `getJetons()` : renvoie un entier donnant le nombre de Jetons banalisés pouvant passer par cet Arc, renvoie -1 si cet Arc ne peut faire passer de Jetons banalisés.
- `translateArriveDe(x, y)` : déplace le point d'arrivée de l'Arc de `x` sur les abscisses et de `y` sur les ordonnées.
- `translateDepartDe(x, y)` : déplace le point de départ de l'Arc de `x` sur les abscisses et de `y` sur les ordonnées.

g) *ArcTest*

Cette classe caractérise les Arcs de Test d'un réseau. C'est une spécialisation de la classe Arc, ainsi tous les attributs et les méthodes proviennent de Arc, à part getType() qui a été redéfinie pour renvoyer 'te'.

Attributs :

Méthode :

`getType()` : renvoie le type de Arc : 'te'.

h) *Inhibiteur*

Cette classe caractérise les Arcs inhibiteurs d'un réseau. C'est une spécialisation de la classe Arc, ainsi tous les attributs et les méthodes proviennent de Arc, à part getType() qui a été redéfinie pour renvoyer 'in'.

Attributs :

Méthode :

- `getType()` : renvoie le type de Arc : 'in'.

i) *ArcFusion*

Ce type d'Arc qui n'a pas du tout le même sens que les Arcs normaux ou inhibiteurs, permet de fusionner deux places ou deux transitions qui ne sont pas dans le même réseau.

Attributs :

- `identifiant` : l'identifiant de l'Arc de fusion sous forme de chaîne de caractères.
- `noeudDepart` : première Place ou Transition de la fusion.
- `noeudArrive` : deuxième Place ou Transition de la fusion.
- `rdpDepart` : le RdPComposite dans lequel se trouve la première Place (respectivement Transition) de la fusion.
- `rdpArrive` : le RdPComposite dans lequel se trouve la deuxième Place (respectivement Transition) de la fusion.
- `lieu` : l'endroit où se trouve l'Arc de fusion sous forme de LieuArc.
- `type` : le type de l'Arc sous forme de chaîne de caractères c'est 'rr' quand les deux places (respectivement deux transitions) sont dans des sous-réseaux différents, 'pr' quand la deuxième Place (respectivement Transition) de la fusion se trouve dans un sous-réseau de la première et 'rp' quand la première Place (respectivement Transition) de la fusion se trouve dans un sous-réseau de la deuxième.

Méthodes :

j) *Methode*

Cette classe implémente les Methodes qui sont utilisées dans une Classe.

Attributs :

- `nom` : le nom de la Methode sous forme de chaîne de caractères.
- `signature` : la signature de la Methode sous forme de Vecteur structuré comme suit :
  1er élément : le type du paramètre sous forme de chaîne de caractère.
  2e élément : un booléen indiquant si ce paramètre est une liste.
  La première paire est le type de retour de la méthode.
  exemple :
  "booleen", false, "entier", true, "reel", false
  dans cet exemple le type de retour de la méthode est un booléen, le premier argument est un tableau d'entiers et le deuxième argument et un réel.
- `chemin` : le chemin du corps de la Methode.

Méthodes :

k) *Classe*

Cette classe implémente les Classes d'un réseau.

Attributs :

- `nom` : le nom de classe.
- `listeAttribut` : la liste des attributs de la Classe sous forme de Vecteur, celui-ci est organisé de la manière suivante :
  1er élément : le nom de l'attribut sous forme de chaîne de caractères.

2e élément : le type de l'attribut, c'est un Vecteur si l'attribut est la définition d'une énumération, ou alors c'est une chaîne de caractères qui peut être 'entier', 'string', 'booleen', 'reel' ou le nom d'une énumération définie précédemment.
3e élément : l'identificateur du type énuméré sous forme de chaîne de caractères (si définition d'une nouvelle énumération).
4e élément : un booléen qui indique si l'attribut est une liste ou non.

- `listeFonction` : la liste des méthodes sous forme d'un Vecteur de Methode.
- `superClasse` : Classe mère, si pas de Classe mère, c'est égal à null.

Méthodes :

- `getListeAttributS()` : renvoie la liste de tous les attributs en ajoutant celles des Classes mères sous forme de Vecteur (voir `listeAttribut` pour voir la structure).
- `getListeFonctionS()` : renvoie la liste de toutes les fonctions en ajoutant celles des Classes mères sous forme de Vecteur de Methode.
- `getEnum()` : renvoie le Vecteur des définitions d'énumération de cette manière :
  1er élément : le Vecteur contenant les valeurs de l'énumération sous forme de chaînes de caractères.
  2e Elément : le nom de l'énumération.
- `getEnum(nomEnum)` : renvoie le Vecteur des valeurs de l'énumération de nom `nomEnum`.
- `getTypesEnum()` : renvoie un Vecteur contenant le nom des toutes les énumérations définies dans cette Classe sous forme de chaîne de caractères.
- `getTypeAttr(nomAttribut)` : renvoie une chaîne de caractères donnant le type de l'attribut ayant pour nom `nomAttribut`.

l) *Etiquette*

Cette classe implémente les Etiquettes qui sont utilisées dans les Places et les Arcs.

Attribut :

- `liste` : un Vecteur définissant l'étiquette, il est structuré de cette manière :
  1er élément : le nom éventuel de la variable, en effet, celui-ci ne sert pas quand l'Etiquette est associée à une Place.
  2e élément : la Classe correspondant à la variable.

Méthodes :

- `get(i)` : renvoie le $i^{ème}$ objet de `liste`.
- `size()` : renvoie la taille de `liste`.
- `equals(o)` : teste l'égalité avec l'Object o.
- `equalsClasse(o)` : teste l'égalité des Classes avec l'Object o.

m) *Noeud*

Classe définissant les Noeuds du réseau (les Places et les Transitions).

Attributs :

- `nom` : le nom du Noeud.
- `identifiant` : l'identifiant du Noeud.
- `listeArcEntrant` : la liste des Arcs entrants dans ce Noeud sous la forme de Vecteur.
- `listeArcSortant` : la liste des Arcs sortants de ce Noeud sous la forme de Vecteur.
- `lieu` : la localisation du Noeud sous forme de LieuBoite.
- `nomRdP` : le nom du réseau de Petri dans lequel se trouve ce Noeud.

Méthodes :

- `getType()` : renvoie le type du Noeud sous forme de chaîne de caractères elle peut être égale à 'interne' si il existe des Arcs entrants et sortants, à 'isole' si il n'existe pas d'Arcs sortants ni d'Arcs entrants, à 'source' si il n'existe que des Arcs sortants ou à 'puit' si il n'existe que des Arcs entrants.
- `ajouteArcEntrant(arc)` : ajoute `arc` à la liste des Arcs entrants.
- `ajouteArcSortant(arc)` : ajoute `arc` à la liste des Arcs sortants.

- `translateDe(x, y)` : déplace le Noeud de `x` sur les abscisses et de `y` sur les ordonnées.

n) *Objet*

Implémente l'instanciation d'une Classe du réseau, utilisé dans Jeton.

Attributs :

- `nom` : le nom de cette instance de Classe.
- `identifiant` : l'identifiant de cet Objet.
- `classe` : la Classe qui est instanciée par cet Objet.
- `listeAttributs` : la liste des attributs de cet Objet sous forme de Vecteur, celui-ci est structuré de cette manière :
  - 1er élément : le nom de l'attribut sous forme de chaîne de caractères.
  - 2e élément : le type de l'attribut, c'est un Vecteur si l'attribut est la définition d'une énumération, ou alors c'est une chaîne de caractères qui peut être 'entier', 'string', 'booleen', 'reel' ou le nom d'une énumération définie précédemment.
  - 3e élément : l'identificateur du type énuméré sous forme de chaîne de caractères (si définition d'une nouvelle énumération).
  - 4e élément : un booléen qui indique si l'attribut est une liste ou non.
  - 5e élément : c'est la valeur de l'attribut, c'est un Vecteur si l'attribut est une liste, sinon c'est une valeur ayant le type approprié.

Méthodes :

- `getNomsAttributs()` : renvoie la liste des noms des attributs sous la forme de Vecteur.
- `getValAttributs()` : renvoie la liste des valeurs des attributs sous la forme de Vecteur.
- `setElementAt(obj, i)` : met l'Object `obj` au rang `i` dans `listeAttributs`.
- `size()` : renvoie la taille de `listeAttributs`.
- `equals(o)` : teste l'égalité de `o` avec cet Objet.
- `update()` : met à jour la liste des attributs en fonction de `classe`.
- `init()` : initialise `listeAttributs` avec des valeurs par défaut.

o) *Jeton*

Cette classe représente les tuples d'Objets.

Attributs :

- `identifiant` : la chaîne de caractère identifiant ce Jeton.
- `listeObjet` : la liste d'Objet de ce Jeton sous forme de Vecteur.

Méthodes :

- `getNom()` : renvoie la liste des noms des Objets contenus dans `listeObjet`.
- `getClasses()` : renvoie la liste des Classe des Objets contenus dans `listeObjet`.
- `getNomsAttributs()` : renvoie un Vecteur contenant la liste des noms des attributs des Objets de `listeObjet`.

p) *Place*

Cette spécialisation de Noeud implémente les Places du réseau.

Attributs :

- `HAUTEUR_DEFAUT` : la hauteur d'une Place sur le réseau.
- `LARGEUR_DEFAUT` : la largeur d'une Place sur le réseau.
- `listeArcFusionEntrant` : la liste des Arcs de fusion entrants dans cette Place.
- `listeArcFusionSortant` : la liste des Arcs de fusion sortants de cette Place.
- `listeEtiquette` : la liste des étiquettes associées à cette Place.
- `listeJeton` : la liste des Jetons de cette Place, c'est le marquage.
- `listeClone` : la liste des clones de cette Place.
- `communication` : booléen qui est égal à vrai si cette Place est d'entrée ou de sortie.

Méthodes :

- `getType()` : retourne le type de la Place sous forme de chaîne de caractère, les valeurs possibles sont :
  1er possibilité : 'normale'
  2e possibilité : 'source'
  3e possibilité : 'entree'
  4e possibilité : 'puit'
  5e possibilité : 'sortie'
  6e possibilité : 'isole'
- `isMutant()` : indique c'est une Place d'entrée ou de sortie fusionnée.
- `isFusion()` : indique si il s'agit d'une Place fusionnée.
- `getJetons()` : retourne le nombre de Jetons banalisés dans la Place, si elle n'en accepte pas retourne -1.
- `estAccepte(etiquette)` : retourne si `etiquette` est acceptée par cette Place.
- `ajoute(etiquette)` : ajoute `etiquette` à `listeEtiquette`.
- `enleve(etiquette)` : enleve `etiquette` de `listeEtiquette`.
- `addJeton(jeton)` : ajoute `jeton` à `listeJeton`.
- `removeJeton(i)` : enlève le Jeton de rang `i` dans `listeJeton`.
- `setJetonAt(jeton, i)` : met le Jeton `jeton` au rang `i` dans `listeJeton`.
- `addClone(placeclone)` : ajoute `placeclone` à `listeClone`.
- `removeClone(i)` : enlève la Place clone de rang `i` dans `listeClone`.
- `removeClone(placeclone)` : enlève la Place clone `placeclone` de `listeClone`.
- `translateDe(x, y)` : déplace la Place de `x` sur les abscisses et de `y` sur les ordonnées.
- `equals(o)` : teste l'égalité entre cette Place et l'Object `o`.

q) *PlaceClone*

Cette spécialisation de Place implémente les Places clones d'un réseau.

Attribut :

- `superPlace` : la Place originelle.

Méthodes :

Toutes les méthodes ont été redéfinies afin de faire appel à la Place d'origine, sauf en ce qui concerne les Arcs, en effet les PlaceClone ont leur propre liste d'Arcs.

r) *Transition*

Spécialisation de Noeud, cette classe implémente les Transitions du réseau.

Attributs :

- `HAUTEUR_DEFAUT` : la hauteur d'une Transition sur le réseau.
- `LARGEUR_DEFAUT` : la largeur d'une Transition sur le réseau.
- `temps` : Vecteur contenant 3 chaînes de caractères décrivant la fenêtre temporelle, la 1ère correspond au premier temps de cette fenêtre, la 2ème correspond au deuxième temps, la 3ème correspond à l'unité ('s' pour secondes ou 'ms' pour millisecondes).
- `nomVerrou` : l'éventuel nom du verrou associé à cette Transition.
- `priorite` : la priorité de cette Transition, par défaut 0.
- `listePredicat` : la liste des conditions de cette Transition sous forme de Vecteur, le 1er élément de ce Vecteur est une chaîne de caractères décrivant la relation entre les termes (non utilisé) ensuite ce sont des Vecteur de taille 3 qui sont de cette forme : [Argument1, Opérateur, Argument2].
- `listeAction` : la liste des actions de cette Transition sous forme de Vecteur, est constituée d'une série de Vecteurs de taille 3 qui sont cette forme : [Argument1, Opérateur, Argument2].

Méthodes :

- `getConflitPotentiel()` : retourne vrai, si la transition est en conflit potentiel avec une ou plusieurs autre transitons.

s) *RdP*

Classe abstraite généralisant les réseaux de Petri.

Attributs :

- `HAUTEUR_DEFAUT` : la hauteur d'un RdP lorsqu'il sous la forme d'un sous-réseau.
- `LARGEUR_DEFAUT` : la largeur d'un RdP lorsqu'il sous la forme d'un sous-réseau.
- `nom` : le nom du RdP.
- `identifiant` : l'identifiant du RdP.
- `comprime` : indique si le RdP est sous forme comprimée (comme un sous-réseau).
- `generique` : indique si le RdP est un modèle générique (cf. [RR02180]).
- `lieu` : la localisation du RdP sous forme de LieuBoite quand c'est un sous-réseau.
- `chemin` : le chemin de sauvegarde du RdP.
- `cheminLib` : le chemin de la librairie de Classes associée à ce RdP.

Méthodes :

- `getRdPSimple()` : retourne le RdPSimple principal du RdP.
- `getListeArc()` : retourne la liste de tous les Arcs du RdP.
- `getListeArcSimple()` : retourne la liste de tous les Arcs simples du RdP.
- `getListeInhibiteur()` : retourne la liste de tous les Arcs inhibiteur du RdP.
- `getListeArcTest()` : retourne la liste de tous les Arcs de Test du RdP.
- `getListeNoeud()` : retourne la liste de tous les Noeuds du RdP.
- `getListePlace()` : retourne la liste de toutes les Places du RdP.
- `getListePlaceClone()` : retourne la liste de toutes les Places clones du RdP.
- `getListeTransition()` : retourne la liste de toutes les Transitions du RdP.
- `getEnum()` : pour chaque Classe de RdP fait appel à `getNum()` et concatène le résultat dans un Vecteur.
- `getEnum(nomEnum)` : pour chaque Classe de RdP fait appel à `getEnum(nomEnum)` et concatène le résultat dans un Vecteur.
- `getTypeAttr(nomClasse, nomAttribut)` : renvoie le type de l'attribut `nomAttribut` de la Classe `nomClasse`.
- `getArc(id)` : renvoie le premier Arc ayant pour identifiant `id` dans les réseaux de RdP.
- `getArc(i)` : renvoie le premier Arc ayant le rang `i` dans la liste des Arcs des réseaux de RdP.
- `getNoeud(nom)` : renvoie le premier Noeud ayant pour nom `nom` dans les réseaux de RdP.
- `getNoeud(i)` : renvoie le premier Noeud ayant le rang `i` dans la liste des Noeuds des réseaux de RdP.
- `getPlace(nom)` : renvoie la première Place ayant pour nom `nom` dans les réseaux de RdP.
- `getPlace(i)` : renvoie la première Place ayant le rang `i` dans la liste des Places des réseaux de RdP.
- `getTransition(nom)` : renvoie la première Transition ayant pour nom `nom` dans les réseaux de RdP.
- `getTransition(i)` : renvoie la première Transition ayant le rang `i` dans la liste des Transitions des réseaux de RdP.
- `getClasse(nom)` : renvoie la Classe ayant pour nom `nom`.
- `getClasse(i)` : renvoie la première Classe ayant le rang `i` dans la liste des Classes des réseaux de RdP.
- `dictionnaire()` : renvoie tous les éléments des réseaux de RdP les uns à la suite des autres, dans cet ordre : les Classes, les Arcs, les Places, les Transitions.
- `dictionnaireAffichage()` : renvoie tous les éléments des réseaux de RdP les uns à la suite des autres, dans cet ordre : les Places, les Transitions, les Arcs.
- `existeNom(type, nom)` : retourne vrai si un élément de type `type` a le nom `nom` dans ce RdP.
- `rienDans(xA, yA, xB, yB)` : vérifie qu'aucun élément de RdP ne se trouve dans le rectangle défini par les points A et B de coordonnées `xA`, `yA`, `xB`, `yB`.

- `rienDans(xA, yA, xB, yB, lui)` : vérifie qu'aucun élément de RdP à part le SuperLieu `lui` ne se trouve dans le rectangle défini par les points A et B de coordonnées `xA`, `yA`, `xB`, `yB`.
- `getIci(x, y)` : renvoie l'élément de RdP dont le lieu recouvre le point de coordonnées (`x`, `y`) dans le cas où il y aurait 2 éléments au même endroit (par exemple, un Arc et un Noeud), c'est le Nœud qui sera renvoyé en priorité.
- `getIci(x, y, lui)` : renvoie l'élément de RdP à part le SuperLieu `lui` dont le lieu recouvre le point de coordonnées (`x`, `y`) dans le cas où il y aurait 2 éléments au même endroit (par exemple, un Arc et un Noeud), c'est le Nœud qui sera renvoyé en priorité.
- `translateDe(x, y)` : déplace le RdP de `x` sur les abscisses et de `y` sur les ordonnées, ceci n'est visible que si RdP est un sous-réseau.

t) *RdPSimple*

Implémentation d'un réseau de Petri sans sous-réseaux.

Attributs :

- `listeArc` : la liste des Arcs du RdPSimple.
- `listeNoeud` : la liste des Noeud du RdPSimple.
- `listeClasse` : la liste des Classes du RdPSimple.
- `listeVerrou` : la liste des verrous du RdPSimple, les verrous sont sous la forme de chaîne de caractère.
- `superRdP` : le RdPSimple père éventuel.

Méthodes :

u) *RdPComposite*

Implémentation des réseaux composés d'un ou plusieurs sous-réseaux.

Attributs :

- `listeRdP` : la liste des RdP de ce RdPComposite (cf. [RR02180]).
- `listeArcFusion` : la liste des Arcs de fusion de ce RdPComposite.
- `listeArcFusionEntrant` : la liste des Arcs de fusion entrants dans ce RdPComposite.
- `listeArcFusionSortant` : la liste des Arcs de fusion sortants de ce RdPComposite.

Méthode :

- `getRdPComposite(nom)` : renvoie le RdPComposite ayant pour nom `nom`.

5.1.2. L'interface de l'éditeur

L'interface graphique a donc été développée en Java avec la librairie Swing. Voici la description (non exhaustives) des principales classes mises en œuvre :

a) *Editeur*

C'est la classe principale de l'éditeur. C'est une spécialisation de JFrame et elle implémente : MouseInputListener, ActionListener, WindowListener , MenuListener, KeyListener. C'est elle qui contient la MenuBar, la ToolBar ainsi que la GraphicBar, voici une description plus détaillée :

Attributs :

- `menuBar` : la barre de menu de menu de l'éditeur.
- `toolBar` : la barre d'outils de l'éditeur.
- `graphicBar` : la palette graphique de l'éditeur.
- `fenetreVerrou` : la fenêtre qui permet de créer un Verrou.
- `listeFenetre` : la liste des Afficheurs qui sont ouverts dans cet Editeur.
- `listeSousReseau` : la liste des sous-réseaux ouverts, il n'est pas possible d'ouvrir plusieurs sous-réseaux de réseaux différents. En d'autres termes, lorsqu'un sous-réseau d'un réseau A est affiché, il n'est pas possible d'afficher le sous-réseau d'un réseau B ouvert dans une autre fenêtre.
- `infoClasses` : la fenêtre d'information des Classes (InfoClasses).

- `propriete` : la fenêtre de Propriete.
- `information` : la fenêtre d'Information.
- `fenetreActive` : l'Afficheur contenant le réseau courant.
- `lastDirectory` : le chemin du dernier répertoire ouvert.
- `pressepapier` : Vecteur contenant la sélection copier ou couper par une Page.

Bien d'autres attributs existent, mais il n'est pas intéressant d'en faire la liste ici.

Méthodes :

- `nouveau()` : crée un Afficheur associé à un nouveau RdP (vide).
- `ouverture(openRdP, chemin, visible)` : ouvre le RdP `openRdP`, dont le chemin de sauvegarde est `chemin` dans un Afficheur qui sera visible selon la valeur de `visible`.
- `ouvertureSousReseau(openRdP, chemin, fichier)` : ouvre le sous-réseau `openRdP`, dont le chemin de sauvegarde est `chemin`. Le paramètre `fichier` spécifie si le sous-réseau a été chargé à partir d'un fichier ou pas.
- `changeLib()` : ouvre une boîte de dialogue permettant de changer la librairie de Classe du réseau courant.
- `importClasse()` : permet d'importer une Classe de la librairie, vers le réseau.
- `supprClasse()` : ouvre une boîte de dialogue qui permet de supprimer une Classe de la librairie.
- `majLibrairie()` : met à jour la librairie de Classe à partir des Classes du réseau.
- `viderLibrairie()` : vide la librairie du réseau courant de toutes ses Classes.

Evidemment cette classe contient bien d'autres méthodes, elle gère certains événements clavier/souris/fenêtre, mais elle initialise la barre de menu etc.

b) *Information*

Cette fenêtre affiche les informations concernant l'élément pointé par la souris. C'est une spécialisation de JDialog.

Attribut :

- `typ2` : JTextField affichant le type de l'élément, par exemple 'Place source', 'Transition' etc.
- `ide2` : JTextField affichant l'identifiant de l'élément pointé.
- `pos2` : JTextField dans lequel s'affiche la position de l'élément pointé.
- `info2` : JTextField qui affiche des infos particulières à certains éléments comme les Jetons pour les Arcs, les Classes acceptées pour les Places etc.
- `info3` : ce JTextField affiche pour Place celle(s) avec qui elle est éventuellement fusionnée.
- `papa` : l'Editeur qui est associée à cette instance.

Cette classe contient d'autres attributs comme des JLabel, ou un gestionnaire de placement.

Méthodes :

- `init(x, y)` : initialise la fenêtre à la position `x y`.
- `affichage()` : met à jour l'affichage des informations.

c) *InfoClasses*

Cette fenêtre affiche des informations relatives au réseau courant. Cette classe est une spécialisation de JDialog et implémente ListSelectionListener, TreeSelectionListener.

Attributs :

- `racine` : la racine de l'arbre sous forme de DefaultMutableTreeNode.
- `arbre` : composant affichant l'arborescence des Classes du réseau courant sous forme de JTree.
- `nomAttr` : liste contenant le nom des attributs de la Classe sélectionnée dans `arbre`, c'est une JList.
- `typeAttr` : liste contenant le type des attributs de la Classe sélectionnée dans `arbre`, c'est une JList.
- `nomMeth` : liste affichant le nom des méthodes de la Classe sélectionnée dans `arbre`, c'est une JList.

- `signMeth` : liste affichant la signature des méthodes de la Classe sélectionnée dans `arbre`, c'est une JList.
- `selection` : la Classe qui est sélectionnée dans l'arborescence des Classes `arbre`.
- `papa` : l'Editeur qui est associé à cette instance.

Méthodes :

- `init(x, y)` : initialise la fenêtre à la position `x y`.
- `modifClasse(r, path)` : méthode appelant la fenêtre de Propriete pour la Classe qui est au bout de `path`, lorsque l'on double-clique sur une Classe de l'arborescence.
- `remplirArbre(liste)` : construit une liste chaînée définissant la hiérarchie des Classes à partir de `liste`, qui est un Vecteur contenant des Classes. Et pour chaque Classe c n'ayant pas de parent, appelle `remplirArbre` (la méthode suivante) avec c, la liste chaînée définissant la structure de l'arbre, la liste des Classes (`liste`) et l'indice de la Classe c dans le Vecteur `liste` (qui est le même que dans la liste chaînée).
- `remplirArbre(noeud, structure, liste, index)` : méthode récursive (appelée par la précédente) ajoutant aux fils de `noeud` (un DefaultMutableTreeNode) un noeud contenant la Classe d'indice `index` dans `liste` puis qui fait appel à elle-même avec la même `structure` et la même `liste`, mais avec le noeud qui vient d'être ajouté et son index dans `liste` et `structure`.
- `maj(liste)` : méthode repeuplant l'arbre avec `liste`, en faisant appel à `remplirArbre`.
- `maj()` : appelle `affichage` avec la `selection` courante.
- `affichage(classe)` : méthode remplissant `nomAttr`, `typeAttr`, `nomMeth` et `signMeth` en fonction de `classe`.

d) *Afficheur*

Comme l'Editeur, cette classe hérite de JFrame et elle implémente KeyListener, MouseInputListener, ActionListener, WindowListener, PopupMenuListener. C'est classe qui contient la Page.

Attributs :

- `scroll` : composant de type JScrollPane qui contient `page` qui elle-même contient le réseau. Il permet de gérer le défilement de la Page.
- `page` : la Page contenant le réseau et qui est chargée de créer et de dessiner celui-ci.
- `status` : l'un des composants (un JLabel) de la barre de statut, c'est celui de gauche affichant des informations relatives au déplacement d'objets ou à leur création etc.
- `statusPos` : l'autre composant de la barre de statut, c'est celui de droite affichant la position du pointeur de la souris, comme `status`, c'est un JLabel.
- `contextHelp` : la fenêtre permettant d'afficher l'aide contextuelle.
- `textAreaHelp` : le composant texte (un JTextArea) qui affiche le texte de l'aide contextuelle.
- `helpPanel` : le composant (un JPanel) contenant `textAreaHelp`, il est lui-même dans `contextHelp`.

Méthodes :

- `initScroll(reseau, opened)` : initialise cet Afficheur avec `reseau` et `opened` qui sont passé en paramètre du constructeur de Page.
- `initContextHelp()` : initialise l'aide contextuelle.
- `afficheAide(message, ligne)` : affiche `message` sur `ligne` lignes dans la fenêtre d'aide.
- `initMenuNorm()` : initialise le menu contextuel.
- `initMenuRdP()` : initialise le menu contextuel dédié aux réseaux.
- `initMenuSousRdP()` : initialise le menu contextuel dédié aux sous réseaux.
- `fermer()` : retourne un entier permettant de déterminer si l'Afficheur peut être fermé par l'Editeur.

e) *Page*

Cette classe hérite de JComponent et implémente Printable.

Attributs :

- `reseau` : le RdP affiché par cette Page.
- `reseauPrecedent` : une pile (Stack) permettant de gérer l'annulation/rétablissement d'une action faîte sur le réseau. Elle contient des RdP.
- `reseauSuivant` : l'autre pile (Stack) permettant de gérer l'annulation/rétablissement d'une action faîte sur le réseau. Elle contient des RdP.
- `actionPrecedent` : une pile (Stack) permettant de gérer le nom des actions faites sur le réseau, sert pour la fonctionnalité d'annulation/rétablissement.
- `actionSuivant` : l'autre pile (Stack) permettant de gérer le nom des actions faites sur le réseau, sert pour la fonctionnalité d'annulation/rétablissement.
- `selection` : un Vecteur de SuperLieu représentant la sélection.

Méthodes :

- `paint(g)` : méthode dessinant dans le Graphics `g` ce qui est affichable sur la Page.
- `print(g,pf, pi)` : méthode permettant de d'imprimer la Page.
- `creerPlace(xA, yA)` : crée une Place dont le centre sera en (`xA`, `yA`).
- `creerPlaceClone(xA, yA)` : crée une Place clone dont le centre sera en (`xA`, `yA`).
- `creerTransition(xA, yA)` : crée une Transition dont le centre sera en (`xA`, `yA`).
- `creerArc(xA, yA, xB, yB)` : crée un Arc dont le départ sera en (`xA`, `yA`) et l'arrivée en (`xB`, `yB`).
- `creerInhibiteur(xA, yA, xB, yB)` : crée un Arc inhibiteur dont le départ sera en (`xA`, `yA`) et l'arrivée en (`xB`, `yB`).
- `creerFusion(xA, yA, xB, yB)` : crée un Arc de fusion dont le départ sera en (`xA`, `yA`) et l'arrivée en (`xB`, `yB`).
- `creerSousReseau(xA, yA)` : crée un sous-réseau dont le centre sera en (`xA`, `yA`). En fait cette méthode ne fait qu'ouvrir une boîte de dialogue permettant de choisir le réseau à ouvrir en tant que sous-réseau, le réseau sera créé effectivement qu'avec la méthode ci-après.
- `actualiserSousReseau(aActualiser)` : actualise le RdPComposite `aActualiser` à partir de son chemin de sauvegarde.
- `creerSousReseau(path, xA, yA)` : méthode créant un sous-réseau qui est situé au chemin `path` dont le centre sera en (`xA`, `yA`).
- `renommageSousReseau(res, n, nomPere)` : renomme récursivement les réseaux de `res` avec `nomPere` comme nom de base, `n` est utilisé pour savoir à quelle profondeur se trouve `res`.
- `fusionneLib(rezo)` : fusionne la librairie de Classe du réseau de la Page avec celle de `rezo`.
- `relieReseau(reseau, rdp)` : encore une méthode récursive, elle permet de relier les sous réseaux à leur père.
- `copier()` : copie le contenu de la sélection dans le presse-papier de l'Editeur.
- `couper()` : coupe le contenu de la sélection dans le presse-papier de l'Editeur.
- `coller()` : colle le contenu du presse-papier de l'Editeur dans la Page.
- `supprimer()` : supprime le contenu de la sélection.
- `annuler()` : annule la dernière action.
- `retablir()` : rétablit la dernière action annulée.
- `stocke(action)` : stocke le réseau courant dans la pile après avoir effectué une action de nom `action`.
- `selection(x, y, ajoute)` : sélectionne l'élément qui se trouve au point (`x`, `y`), `ajoute` qui est un entier défini comme on ajoute cet objet à la sélection.
- `selection(xA, xB, yA, yB, ajoute)` : sélectionne les éléments qui se trouvent dans la zone définie par les points (`xA`, `yA`) et (`xB`, `yB`), `ajoute` qui est un entier défini comme on ajoute ces objets à la sélection.

- `dessineReseauComprime(xA, xB, yA, yB, g, c)` : dessine un sous-réseau de couleur `c`, sur le rectangle défini par (`xA`, `yA`) et (`xB`, `yB`) dans le Graphics `g`.
- `dessineFusion(xA, xB, yA, yB, g, c)` : dessine un Arc de fusion entre les points (`xA`, `yA`) et (`xB`, `yB`) de couleur `c`, dans le Graphics `g`.
- `dessineArc(xA, xB, yA, yB, g, c)` : dessine un Arc entre les points (`xA`, `yA`) et (`xB`, `yB`) de couleur `c`, dans le Graphics `g`.
- `dessineArcTest(xA, xB, yA, yB, g, c)` : dessine un Arc de Test entre les points (`xA`, `yA`) et (`xB`, `yB`) de couleur `c`, dans le Graphics `g`.
- `dessineInhibiteur(xA, xB, yA, yB, g, c)` : dessine un Arc inhibiteur entre les points (`xA`, `yA`) et (`xB`, `yB`) de couleur `c`, dans le Graphics `g`.
- `dessineCercle(x, y, r, g, c)` : dessine un cercle de centre (`x`, `y`), de rayon `r`, de couleur `c` dans le Graphics `g`.
- `dessinePlace(xA, xB, yA, yB, g, c, fond, entrant, sortant, marque)` : dessine une Place à l'intérieur du rectangle défini par les points (`xA`, `yA`) et (`xB`, `yB`) de couleur `c`, dans le Graphics `g`, avec un fond de couleur `fond`, `entrant` Arcs de fusions entrants, `sortant` Arcs de fusions sortants et ayant `marque` Jetons banalisés, si la Place contient des Jetons non banalisés `marque` est égal à -1.
- `dessinePlaceClone(xA, xB, yA, yB, g, c, fond, entrant, sortant, marque)` : dessine une Place clone à l'intérieur du rectangle défini par les points (`xA`, `yA`) et (`xB`, `yB`) de couleur `c`, dans le Graphics `g`, avec un fond de couleur `fond`, `entrant` Arcs de fusions entrants, `sortant` Arcs de fusions sortants et ayant `marque` Jetons banalisés, si la Place contient des Jetons non banalisés `marque` est égal à -1.
- `dessineTransition(xA, xB, yA, yB, g, c)` : dessine une Transition de couleur `c`, sur le rectangle défini par (`xA`, `yA`) et (`xB`, `yB`) dans le Graphics `g`.
- `dessineLabels(g, l)` : dessine les messages contenus dans `l` dans le Graphics `g`. `l` est structuré comme suit :
  1er élément : le label lui-même sous forme de chaîne de caractères.
  2e élément : l'abscisse du point où doit être dessiné le label.
  3e élément : l'ordonnée du point où doit être dessiné le label.
- `dessineReseauMiniature(xA, xB, yA, yB, rezo, g)` : dessine une miniature du réseau rezo dans le rectangle défini par les points (`xA`, `yA`) et (`xB`, `yB`), dans le Graphics `g`.
- `dessineReseau(r, g)` : dessine le réseau `r` dans le Graphics `g`.
- `dejaArcEntre(D, A)` : teste si il existe un Arc entre les Noeud `D` et `A`.
- `dejaArcEntre2(D, A)` : fait le même test que précédemment mais seulement visuellement.
- `dejaArcFusionEntre(D, A)` : teste si un Arc de fusion existe entre les Places (ou Transitions) `D` et `A`.

f) *Propriété*

Cette classe gère toutes les Propriétés des entités des Réseaux.

Attributs :

- `ici` : désigne l'entité du réseau dont les propriétés sont éditées, il est sous la forme d'un Super.
- `papa` : l'éditeur ayant ouvert cette fenêtre de Propriétés.

Méthodes :

- `init()` : initialise cette fenêtre de Propriété en fonction du type de l'entité éditée.

6. Références

[Passama02] R. Passama. *Rapport de Stage de DEA Informatique*. L.I.R.M.M. Montpellier, Juillet 2002.
[Raclot02] F. Raclot. Rapport de Stage de DESS TNI. L.I.R.M.M. Montpellier, Juillet 2002.
[RR02180] F. Raclot, D. Andreu, T. Libourel, R. Passama. *E-NetObject : un éditeur de Réseaux de Petri à Objets*. RR L.I.R.M.M. n° 02180.
[RR02182] R. Passama, D. Andreu, F. Raclot, T. Libourel. *J-NetObject : un noyau d'exécution de Réseaux de Petri à Objets*. RR L.I.R.M.M. n° 02182.