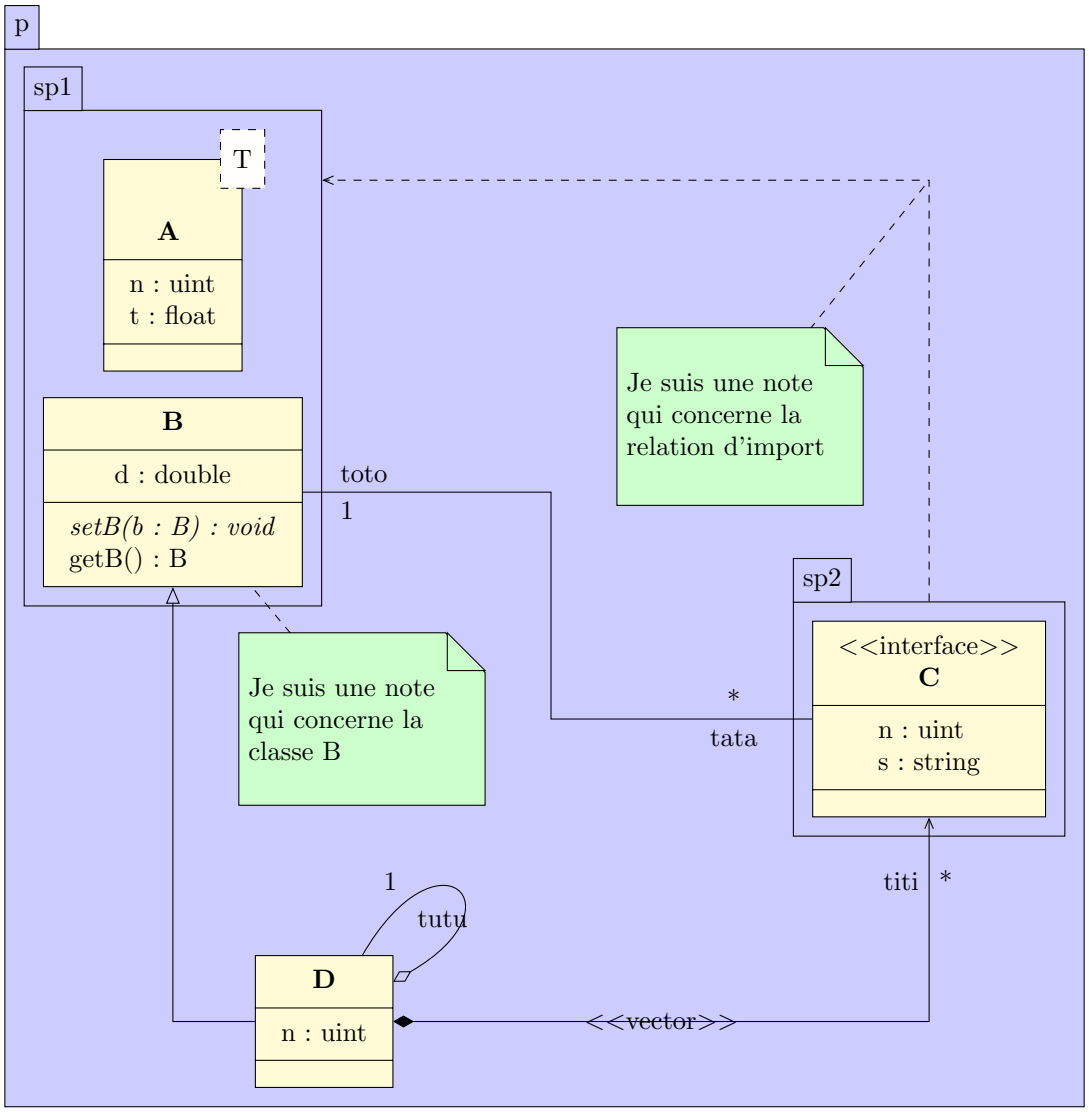
Le package TikZ-uml

Nicolas Kielbasiewicz

22 avril 2011

Devant les immenses possibilités offertes par la librairies PGF/TikZ, et devant l'absence apparente de package dédié aux diagrammes UML, j'ai été amené à développer le package TikZ-uml, proposant un ensemble de commandes et d'environnements spécifiques à ce type de diagrammes. Il s'inscrit dans la logique du package pst-uml développé pour des raisons similaires en PSTricks. Dans son état actuel, la librairie ne permet de définir que des diagrammes de classes complets de manière assez simple, mais il est prévu de proposer des diagrammes de cas d'utilisation, des diagrammes de séquence, voire ensuite d'autres diagrammes.

Voici un exemple de diagramme de classes que l'on peut réaliser :



Nous allons maintenant vous présenter les différentes fonctionnalités offertes par TikZ-uml.

Table des matières

1 Diagrammes de classes **3**
- 1.1 Package, classe, attributs et opérations . . . 3
  - 1.1.1 Définir un package . . . 3
  - 1.1.2 Définir une classe . . . 3
  - 1.1.3 Définir des attributs et des opérations . . . 4
- 1.2 Relations entre classes . . . 4
  - 1.2.1 Commande générale . . . 4
  - 1.2.2 Définir la géométrie de la relation . . . 5
  - 1.2.3 Ajuster la géométrie de la relation . . . 6
  - 1.2.4 Définir des informations sur les attributs associés à une relation . . . 7
  - 1.2.5 Positionner les informations sur les attributs associés à une relation . . . 7
  - 1.2.6 Ajuster l'alignement des informations sur les attributs associés à une relation . . . 8
  - 1.2.7 Définir et positionner le stéréotype d'une relation . . . 8
  - 1.2.8 Modifier les points d'ancrage d'une relation . . . 8
  - 1.2.9 Relation récursive . . . 9
  - 1.2.10 Nom des points de construction d'une relation . . . 10
  - 1.2.11 Tracer un point à une intersection de relations . . . 11
- 1.3 Note de commentaires / contraintes . . . 11
- 1.4 Personnaliser l'apparence . . . 12
- 1.5 Exemples . . . 13
  - 1.5.1 Exemple de l'introduction, pas à pas . . . 13
  - 1.5.2 Définir une spécialisation de classe . . . 17
- 1.6 Règles de priorité des options et bugs identifiés . . . 17

2 Diagrammes de cas d'utilisation **20**
- 2.1 Système . . . 20
- 2.2 Acteur et cas d'utilisation . . . 20
- 2.3 Relations . . . 20
- 2.4 Personnalisation . . . 20

3 Diagrammes d'états-transition **21**

4 Diagrammes de séquence **22**

Chapitre 1

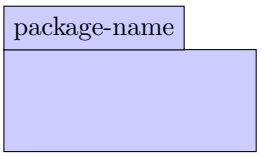
Diagrammes de classes

1.1 Package, classe, attributs et opérations

1.1.1 Définir un package

Un package est défini par l'environnement `umlpackage` :

```
\begin{tikzpicture}
\begin{umlpackage}[x=0,y=0]{package-name}
\end{umlpackage}
\end{tikzpicture}
```



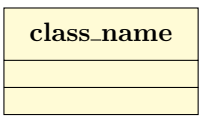
Les options `x` et `y` permettent de définir la position du package dans la figure. Elles valent toutes deux 0 par défaut.

- Quand un package contient des classes ou des sous-packages, sa taille s'ajuste automatiquement à son contenu.
- On peut définir autant de packages que l'on veut dans une figure.
- Il existe un raccourci pour définir un package qui sera vide (on ne définira pas de classes à l'intérieur) : la commande `umlemptypackage` qui prend les mêmes arguments que l'environnement `umlpackage`

1.1.2 Définir une classe

Pour définir une classe, on utilise l'environnement `umlclass`, de la même manière que l'environnement `umlpackage` :

```
\begin{tikzpicture}
\umlclass[x=0,y=0]{class\_name}{}{}
\end{tikzpicture}
```

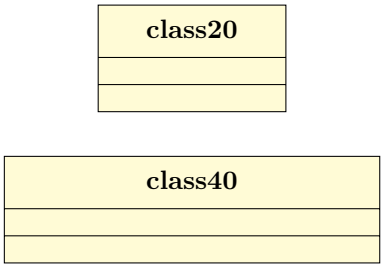


Les options `x` et `y` définissent la position de la classe. 2 possibilités : si la classe est définie dans un package, il s'agit de la position relative de la classe dans le package ; dans le cas contraire, il s'agit de la position dans la figure. Elles valent toutes deux 0 par défaut. Comme pour le package, il existe un raccourci pour définir une classe vide (dont on ne définira pas les attributs et les opérations) : la commande `umlemptyclass`, qui prend les mêmes arguments que l'environnement `umlclass`.

Définir la largeur d'une classe

Par défaut, la largeur d'une classe est de 10ex. On peut la modifier avec l'option `width` :

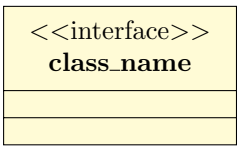
```
\begin{tikzpicture}
\umlemptyclass[width=15ex]{class20}
\umlemptyclass[y=-2, width=30ex]{class40}
\end{tikzpicture}
```



Spécifier le type d'une classe

La classe peut être de différents types : classe, interface, typedef, enum. On utilise l'option `type` :

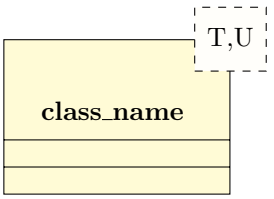
```
\begin{tikzpicture}
\umlemptyclass[type=interface]{class\_
    name}
\end{tikzpicture}
```



Spécifier des paramètres templates

La classe peut être un template. On spécifie la liste des paramètres avec l'option `template` :

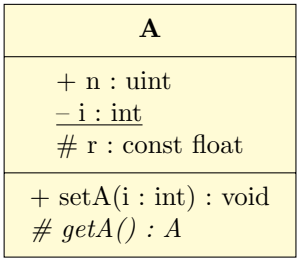
```
\begin{tikzpicture}
\umlemptyclass[template={T,U}]{class\_
    name}
\end{tikzpicture}
```



1.1.3 Définir des attributs et des opérations

On définit les attributs à l'intérieur d'un environnement `umlclass` à l'aide de la commande `umlattr`. On lui passe en argument la liste des arguments en les séparants par \\. On procède de même pour les opérations avec la commande `umlop` :

```
\begin{tikzpicture}
\umlclass{A}{
  + n : uint \\ \umlstatic{-- i : int} \\
      \# r : const float
  }{
  + setA(i : int) : void \\ \umlvirt{\#
      getA() : A}
}
\end{tikzpicture}
```



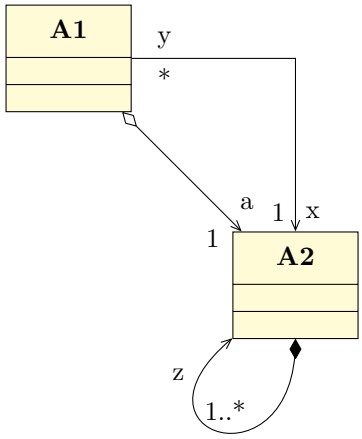
Pour définir un attribut ou une méthode statique, on peut utiliser la commande `umlstatic`. De même, la commande `umlvirt` permet de spécifier des fonctions virtuelles.

1.2 Relations entre classes

1.2.1 Commande générale

Chaque classe et chaque package sont représentés par un nœud ayant le même nom. Pour définir une relation entre 2 classes, on va donc spécifier le nom de la classe de départ, le nom de la classe d'arrivée et un certain nombre d'options propres à la relation.

```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=3,y=-3]{A2}
\umluniaggreg[arg2=a,mult2=1,pos2=0.9]{A
    1}{A2}
\umluniassoc[geometry=-|,arg1=x,mult1=1,
    pos1=1.9,arg2=y,mult2=*,pos2=0.2]{A1}{
    A2}
\umlunicompo[arg=z,mult=1..*,pos=0.8,
    angle1=-90, angle2=-140, loopsize=2cm
    ]{A2}{A2}
\end{tikzpicture}
```



D'un point de vue sémantique, il existe 11 relations différentes, toutes présentes dans TikZ-uml :

La dépendance : utiliser la commande `umldep`

L'association : utiliser la commande `umlassoc`

L'association unidirectionnelle : utiliser la commande `umluniassoc`

L'aggrégation : utiliser la commande `umlaggreg`

L'aggrégation unidirectionnelle : utiliser la commande `umluniaggreg`

La composition : utiliser la commande `umlcompo`

La composition unidirectionnelle : utiliser la commande `umlunicompo`

L'import : utiliser la commande `umlimport`

L'héritage : utiliser la commande `umlinherit`

L'implémentation : utiliser la commande `umlimpl`

La réalisation : utiliser la commande `umlreal`

Ces 11 commandes sont basées sur le même schéma (la commande `umlrelation`) et prennent en théorie exactement le même jeu d'options. En pratique, certaines options concernent certains types de relations.

1.2.2 Définir la géométrie de la relation

Comme vous avez pu le voir dans les exemples précédents, on peut spécifier la forme géométrique de la relation à l'aide de l'option `geometry`. Cette option demande un argument parmi la liste - - (ligne droite), -| (horizontal puis vertical), |- (vertical puis horizontal), -|- (chicane horizontale) ou |-| (chicane verticale), ces arguments étant largement inspirés de la philosophie TikZ.

Il apparaît à l'utilisation que cette option est très souvent utilisée. C'est la raison pour laquelle un alias de la commande `umlrelation` a été défini pour chacune des valeurs possibles de l'option `geometry` :

umlHVrelation : alias de umlrelation avec geometry=-|

umlVHrelation : alias de umlrelation avec geometry=|-

umlHVHrelation : alias de umlrelation avec geometry=-|-

umlVHVrelation : alias de umlrelation avec geometry=|-|

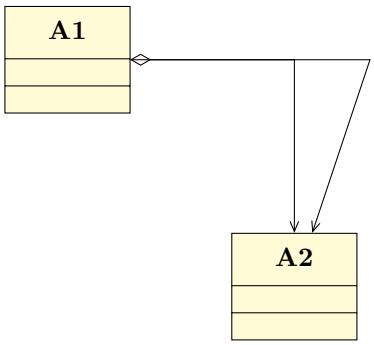
⚠ Pour chacun de ces 4 alias, l'option `geometry` est interdite.

⚠ Il n'y a pas d'alias dans le cas de la valeur - - pour la seule raison qu'il s'agit de la valeur par défaut.

1.2.3 Ajuster la géométrie de la relation

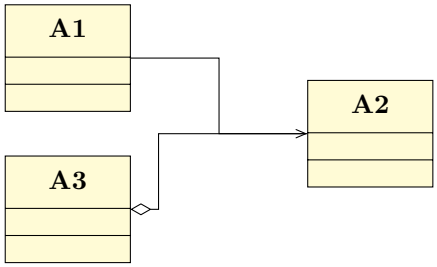
Lorsque la géométrie de la relation contient 2 segments, on peut spécifier les coordonnées du point intermédiaire, ou nœud de contrôle. Plutôt que la commande `umlrelation`, on utilisera `umlCNrelation`, elle aussi déclinée en 11 alias :

```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=3,y=-3]{A2}
\umluniaggreg[geometry=-|]{A1}{A2}
\umlCNuniassoc{A1}{4,0}{A2}
\end{tikzpicture}
```



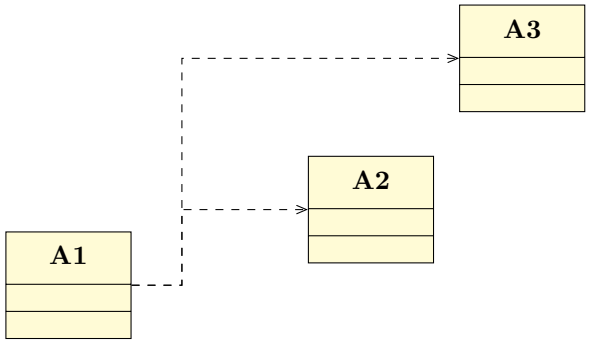
Lorsque la géométrie de la relation contient 3 segments, la position relative du segment central entre les classes est défini comme passant par le milieu entre les classes reliées. On peut ajuster ce paramètre à l'aide de l'option `weight` :

```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=4, y=-1]{A2}
\umlemptyclass[y=-2]{A3}
\umlassoc[geometry=-|-]{A1}{A2}
\umluniaggreg[geometry=-|-, weight=0.3]{A
    3}{A2}
\end{tikzpicture}
```

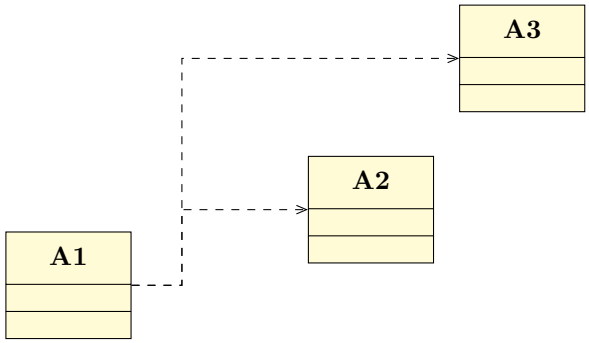


Dans certains cas, cette option est peu pratique, car elle demande de calculer la valeur à passer à l'option. On peut alors procéder autrement en utilisant les options `arm1` et `arm2` qui fixent la taille respectivement du premier et dernier segment. Regardons ici les deux exemples utilisant respectivement l'option `weight` et l'option `arm1` :

```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=4, y=1]{A2}
\umlemptyclass[x=6, y=3]{A3}

\umlHVHdep[weight=0.375]{A1}{A2}
\umlHVHdep[weight=0.25]{A1}{A3}
\end{tikzpicture}
```



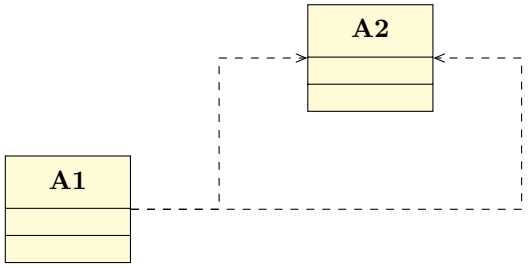

```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=4, y=1]{A2}
\umlemptyclass[x=6, y=3]{A3}

\umlHVHdep[arm1=1.5cm]{A1}{A2}
\umlHVHdep[arm1=1.5cm]{A1}{A3}
\end{tikzpicture}
```



Les options `arm1` et `arm2` prennent aussi des valeurs négatives. Que se passe t'il alors ? Si l'on passe une valeur positive, alors le bras sera orienté dans le sens normal (soit à droite, soit vers le haut). Si l'on

passe une valeur négative, alors l'arc sera orienté dans l'autre sens, ce qui permet de dessiner d'autres types de relations à 3 segments, comme le montre l'exemple ci-dessous :

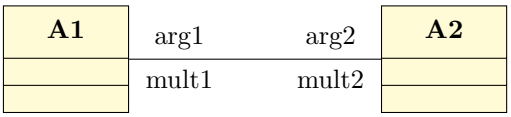
```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=4, y=2]{A2}

\umlHVHdep[arm2=-2cm]{A1}{A2}
\umlHVHdep[arm2=2cm]{A1}{A2}
\end{tikzpicture}
```



1.2.4 Définir des informations sur les attributs associés à une relation

Une relation visualise la dépendance entre deux classes et se traduit généralement sous la forme d'un attribut. On peut spécifier son nom avec l'option `arg1` ou `arg2`, et sa multiplicité avec `mult1` ou `mult2` :

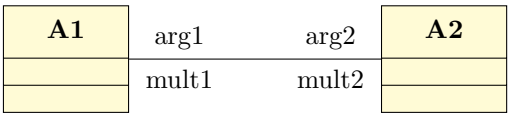
```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=5]{A2}
\umlassoc[arg1=arg1, mult1=mult1, arg2=
    arg2, mult2=mult2]{A1}{A2}
\end{tikzpicture}
```



Pour les relations unidirectionnelles, on ne va être amené à utiliser que les options `arg2` et `mult2`. Comme il paraît dans ce cas peu naturel que ces options finissent par 2, on peut utiliser les options `arg` et `mult` qui jouent le même rôle.

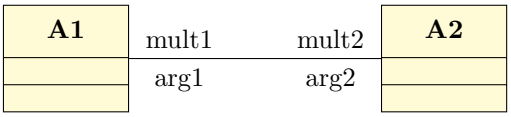
Par ailleurs, lorsqu'on définit à la fois le nom et la multiplicité d'un attribut, on peut le faire sous une forme plus contractée à l'aide des options `attr1`, `attr2` et `attr` :

```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=5]{A2}
\umlassoc[attr1=arg1|mult1, attr2=arg2|
    mult2]{A1}{A2}
\end{tikzpicture}
```



L'avantage de cette forme contractée est de pouvoir s'affranchir de la sémantique entre le nom et la multiplicité et de pouvoir alors inverser les deux :

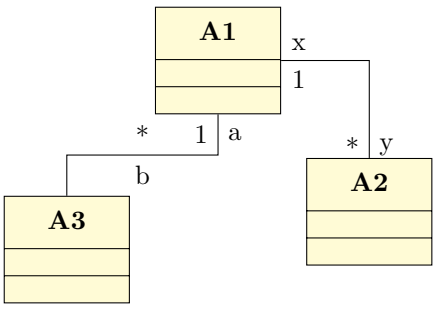
```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=5]{A2}
\umlassoc[attr1=mult1|arg1, attr2=mult2|
    arg2]{A1}{A2}
\end{tikzpicture}
```



1.2.5 Positionner les informations sur les attributs associés à une relation

On peut positionner les informations définies dans la section précédente à l'aide des options `pos1`, `pos2` et `pos`. La commande `umlrelation` détermine elle-même si le nom et la multiplicité doivent être respectivement à gauche et à droite, ou au-dessus et au-dessous, de la flèche, selon sa géométrie et leur position. Pour les initiés à TikZ, elle se base sur les options `auto` et `swap`.

```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=2,y=-2]{A2}
\umlemptyclass[x=-2,y=-2.5]{A3}
\umlassoc[geometry=-|,arg1=x,mult1=1,pos
    1=0.2,arg2=y,mult2=*,pos2=1.9]{A1}{A2}
\umlassoc[geometry=|-|,arg1=a,mult1=1,pos
    1=0.5,arg2=b,mult2=*,pos2=1.5]{A1}{A3}
\end{tikzpicture}
```

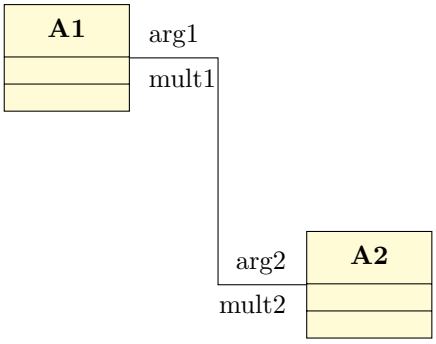


Il est à noter que l'intervalle de valeurs de la position d'un nom d'argument dépend du nombre de segments constituants la flèche de relation. Si la flèche est droite, alors la position doit être comprise entre 0 (classe de départ) et 1 (classe d'arrivée). Si la flèche comporte un seul angle droit, alors la position varie entre 0 et 2 (point d'arrivée), la valeur 1 correspondant à l'angle. Dans les deux autres possibilités, la position varie entre 0 et 3 (point d'arrivée), les valeurs 1 et 2 correspondant respectivement au premier et au deuxième angle.

1.2.6 Ajuster l'alignement des informations sur les attributs associés à une relation

Par défaut, nom et multiplicité de l'argument, quand ils sont affichés l'un au dessus de l'autre, sont centrés. Les options `align1`, `align2` et permettent de les justifier à gauche ou à droite.

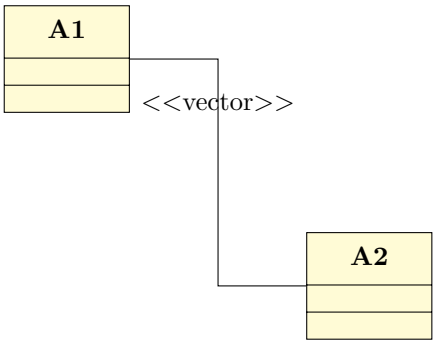
```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=4, y=-3]{A2}
\umlassoc[geometry=-|-, arg1=arg1, mult1=
    mult1, pos1=0.1, align1=left, arg2=arg
    2, mult2=mult2, pos2=2.9, align2=right
    ]{A1}{A2}
\end{tikzpicture}
```



1.2.7 Définir et positionner le stéréotype d'une relation

Le stéréotype d'une relation est un mot clé contenu entre << et >>. On peut le définir à l'aide de l'opération `stereo`

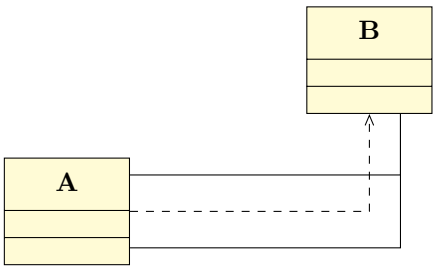
```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=4, y=-3]{A2}
\umlassoc[geometry=-|-,stereo=vector, pos
    stereo=1.2]{A1}{A2}
\end{tikzpicture}
```



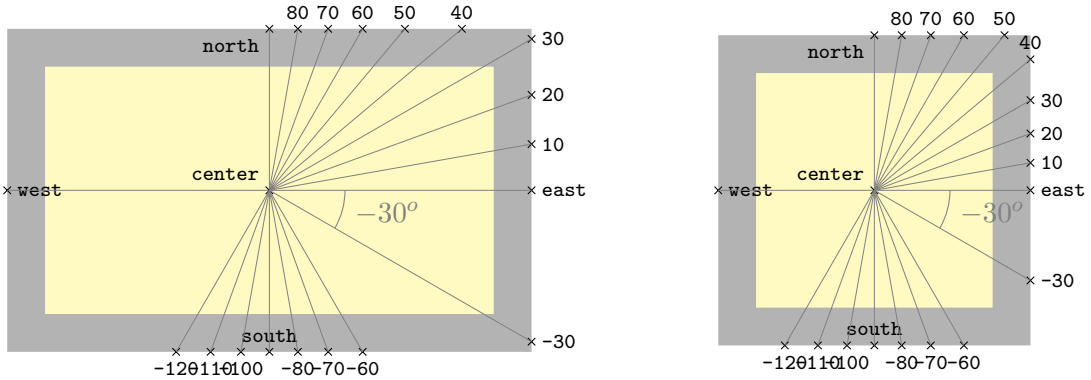
1.2.8 Modifier les points d'ancrage d'une relation

Le paragraphe qui vient concerne les relations dont la géométrie est à base de flèches segmentées. Par défaut, elles partent du centre du nœud de la classe d'origine et vont au centre du nœud de la classe cible. Il est possible d'ajuster ce comportement avec la paire d'options `anchor1`, `anchor2`.

```
\begin{tikzpicture}
\umlemptyclass{A}
\umlemptyclass[x=4,y=2]{B}
\umldep[geometry=-|]{A}{B}
\umlassoc[geometry=-|, anchor1=30, anchor2=300,
    name=assoc 1]{A}{B}
\umlassoc[geometry=-|, anchor1=-30, anchor2=-60,
    name=assoc 2]{A}{B}
\end{tikzpicture}
```

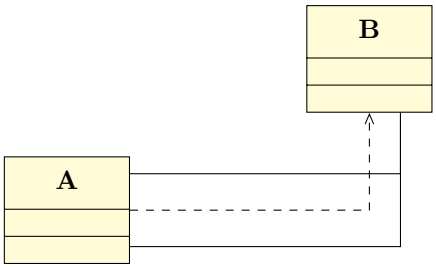


Les arguments que l'on donne sont des angles dont la valeur est en degré et peut être négative. Le mécanisme interne de la librairie TikZ effectue un modulo pour ramener ce nombre dans l'intervalle adéquat. La valeur 0 indique l'est, 90, indique le nord, 180 indique l'ouest, et 270 (ou -90) indique le sud du nœud. La figure ci-dessous illustre cette option et sa signification angulaire, sur 2 exemples de nœud de type rectangle, comme c'est le cas pour les classes. À noter que les points d'ancrage frontières (pour prendre la terminologie TikZ) dépendent bien des dimensions du nœud.



Il arrive finalement très souvent que l'on définisse les deux options `anchor1` et `anchor2` simultanément. On peut donc utiliser une forme contractée : l'option `anchors` qui s'utilise de la manière suivante, en reprenant l'exemple précédent :

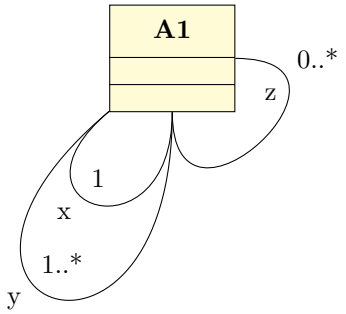
```
\begin{tikzpicture}
\umlemptyclass{A}
\umlemptyclass[x=4,y=2]{B}
\umldep[geometry=-|]{A}{B}
\umlassoc[geometry=-|, anchors=30 and 300, name=
    assoc 1]{A}{B}
\umlassoc[geometry=-|, anchors=-30 and -60, name=
    assoc 2]{A}{B}
\end{tikzpicture}
```



1.2.9 Relation récursive

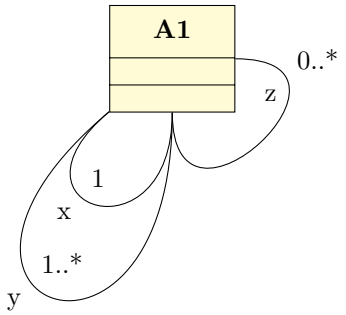
Il est possible de définir une relation récursive, c'est-à-dire une relation d'une classe à elle-même. Dans ce cas, l'option `geometry` doit être ignorée, mais 3 options deviennent très utiles : `angle1` détermine l'angle de départ, `angle2` détermine l'angle d'arrivée, et `loopsize` donne une idée de la taille de la boucle.

```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlassoc[arg=x,mult=1,pos=0.6, angle
    1=-90, angle2=-140, loopsize=2cm]{A1}{
    A1}
\umlassoc[arg=y,mult=1..*,pos=0.6, angle
    1=-90, angle2=-140, loopsize=4cm]{A1}{
    A1}
\umlassoc[arg=z,mult=0..*,pos=0.8, angle
    1=-90, angle2=0, loopsize=2cm]{A1}{A1}
\end{tikzpicture}
```



À l'utilisation, il s'avère que l'on utilise très souvent les 3 options en même temps. C'est la raison pour laquelle il existe une forme contractée, l'option `recursive` qui s'utilise de la manière suivante :

```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlassoc[arg=x, mult=1, pos=0.6,
    recursive=-90|-140|2cm]{A1}{A1}
\umlassoc[arg=y, mult=1..*, pos=0.6,
    recursive=-90|-140|4cm]{A1}{A1}
\umlassoc[arg=z, mult=0..*, pos=0.8,
    recursive=-90|0|2cm]{A1}{A1}
\end{tikzpicture}
```



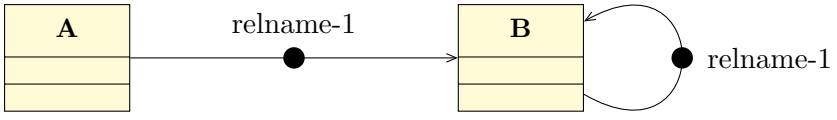
1.2.10 Nom des points de construction d'une relation

Pour voir l'importance de la possibilité de donner un nom à une relation et son utilité, il nous faut d'abord ici répondre à la question technique suivante : comment les flèches sont-elles concrètement définies ?

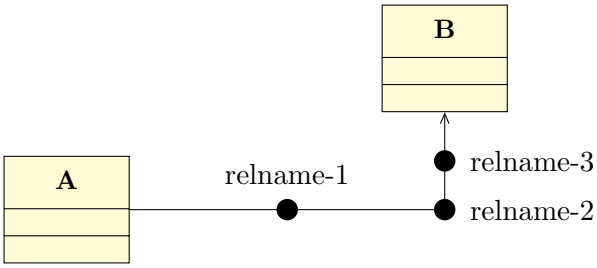
Pour construire une flèche, on a besoin de définir des nœuds de contrôle, auquel il faut donner un nom. Le seul moyen d'identifier de manière unique une relation est de lui donner un identifiant à travers un compteur que l'on va incrémenter. Supposons que notre relation a pour identifiant i. Le nom de la relation que l'on appellera dans ce qui suit *relname* est alors initialisé à : relation-i

Le premier nœud défini est le milieu de la relation, il s'appelle *relname-middle*. Je ne parlerai pas ici du placement adéquat de l'argument et de sa multiplicité dans un souci de simplification. Il y a donc 3 possibilités :

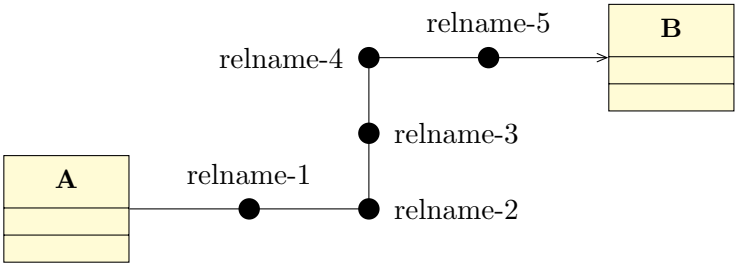
1. Si la flèche est une ligne continue (droite ou récursive), il est renommé en *relname-1*.



2. Si la flèche a un seul angle droit, alors l'unique nœud correspondant à l'angle droit est nommé *relname-2*, ce qui suffit à tracer la flèche. On définit par ailleurs les milieux des 2 arêtes constituant la flèche, nommés respectivement *relname-1* et *relname-3*.



3. Si la flèche à deux angles droits, ceux-ci sont définis de manière unique à l'aide de *relname-middle*, ce qui suffit à tracer la flèche. On nomme les nœuds aux angles droits respectivement *relname-2* et *relname-4*. On définit alors les milieux des 3 arêtes constituant la flèche, nommés respectivement *relname-1*, *relname-3*, et *relname-5*.

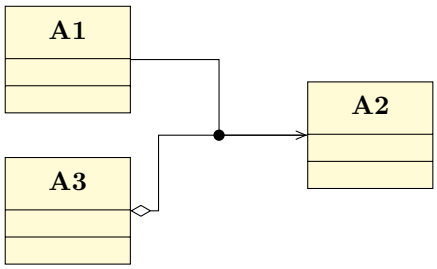


Il faut toutefois reconnaître que la définition par défaut de *relname* est non seulement peu pratique puisqu'on n'a pas vraiment accès à la numérotation, mais aussi fortement sujette à l'ordre dans lequel on définit les relations, donc peu portable en cas de modification de diagrammes. Pour simplifier cela, on va pouvoir définir *relname* à l'aide de l'option `name`. Et la raison pour laquelle cette option existe va être expliquée dans les 2 sections suivantes.

1.2.11 Tracer un point à une intersection de relations

Il arrive que dans un diagramme des relations se croisent et se chevauchent. Prenons 2 flèches qui se croisent. Est-ce que les 2 points de départ peuvent aller aux deux points d'arrivée ? Si oui, alors on va vouloir tracer un point à l'intersection des deux flèches, qui vraisemblablement va être un des nœuds de contrôle définis sur la relation. On utilisera pour cela la commande `umlpoint`.

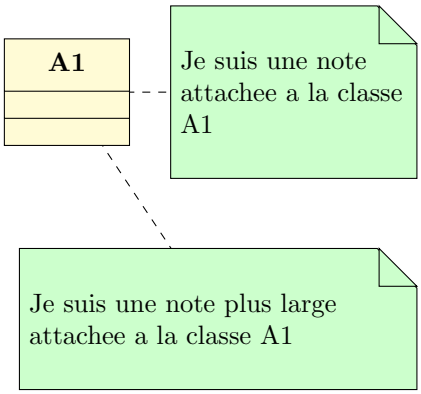
```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlemptyclass[x=4, y=-1]{A2}
\umlemptyclass[y=-2]{A3}
\umlassoc[geometry=-|-, name=assoc]{A1}{A
    2}
\umluniaggreg[geometry=-|-, weight=0.3]{A
    3}{A2}
\umlpoint{assoc-4}
\end{tikzpicture}
```



1.3 Note de commentaires / contraintes

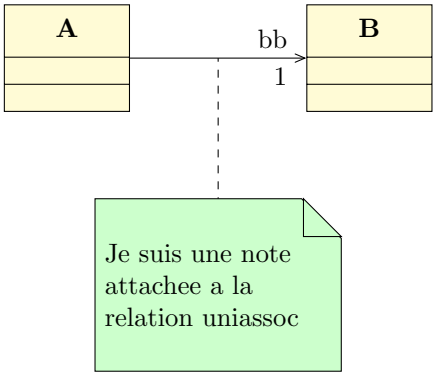
Une note est un commentaire de texte attaché à une classe ou une relation. La commande `umlnote` demande pour cela le nom du nœud auquel il faut se rattacher et le texte du commentaire en argument :

```
\begin{tikzpicture}
\umlemptyclass{A1}
\umlnote[x=3]{A1}{Je suis une note
    attachee a la classe A1}
\umlnote[x=2,y=-3, width=5cm]{A1}{Je suis
     une note plus large attachee a la
    classe A1}
\end{tikzpicture}
```



Là encore, on va pouvoir donner le nom d'un des points de contrôle d'une relation si l'on veut y attacher une note. Il faut donc pour cela pouvoir définir soi-même un nom simple à ces points :

```
\begin{tikzpicture}
\umlemptyclass{A}
\umlemptyclass[x=4]{B}
\umluniassoc[arg=bb, mult=1, pos=0.95, align=
    right, name=uniassoc]{A}{B}
\umlnote[x=2,y=-3]{uniassoc-1}{Je suis une note
    attachee a la relation uniassoc}
\end{tikzpicture}
```



1.4 Personnaliser l'apparence

Grâce à la commande `tikzumlset`, il est possible de modifier l'apparence par défaut des packages, des classes et des notes. Les options que l'on peut personnaliser sont :

text : permet de spécifier la couleur du texte (=black par défaut),

draw : permet de spécifier la couleur des bords (=black par défaut),

fill class : permet de spécifier la couleur de fond des classes (=yellow!20 par défaut),

fill template : permet de spécifier la couleur de fond des boites templates (=yellow!2 par défaut),

fill package : permet de spécifier la couleur de fond des packages (=blue!20 par défaut),

fill subpackage : permet de spécifier la couleur de fond des sous-packages (=blue!20 par défaut),

fill note : permet de spécifier la couleur de fond des notes (=green!20 par défaut),

font : permet de définir le style de fonte du texte contenu dans tous les éléments d'un diagramme (=par défaut).

Par ailleurs, les commandes de relation disposent toutes d'une option `style` prenant en argument un nom de style au sens de TikZ.

Regardons l'exemple de la définition de la commande `umlinherit` :

```
\tikzstyle{tikzuml inherit style}=[color=\tikzumldrawcolor, -open triangle 45]
\newcommand{\umlinherit}[3][]{\umlrelation[style={tikzuml inherit style},
    #1]{#2}{#3}}
```

Vous pouvez donc très facilement définir une commande sur le même modèle en définissant un style particulier.

1.5 Exemples

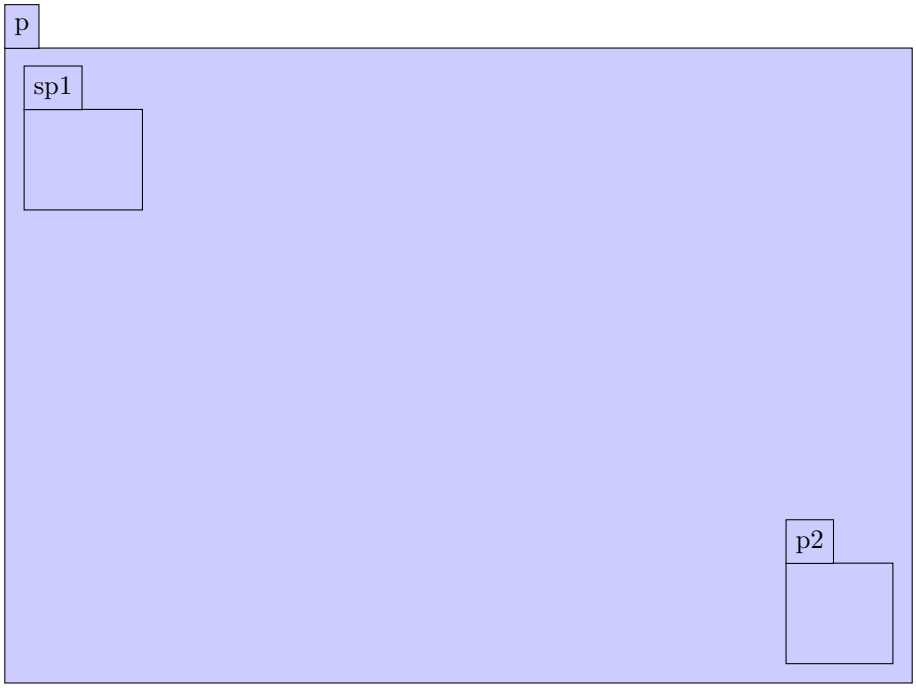
1.5.1 Exemple de l'introduction, pas à pas

On va construire petit à petit l'exemple illustrant la première page de ce document afin de mettre en valeur les différentes commandes utilisées.

Définition des packages p, sp1 et sp2

On laisse le package p aux coordonnées (0,0) (comportement par défaut), et on place le sous-package p1 aux coordonnées (0,0) et le sous-package p2 aux coordonnées (10,-6).

```
\begin{umlpackage}{p}
\begin{umlpackage}{sp1}
\end{umlpackage}
\begin{umlpackage}[x=10,y=-6]{sp2}
\end{umlpackage}
\end{umlpackage}
```



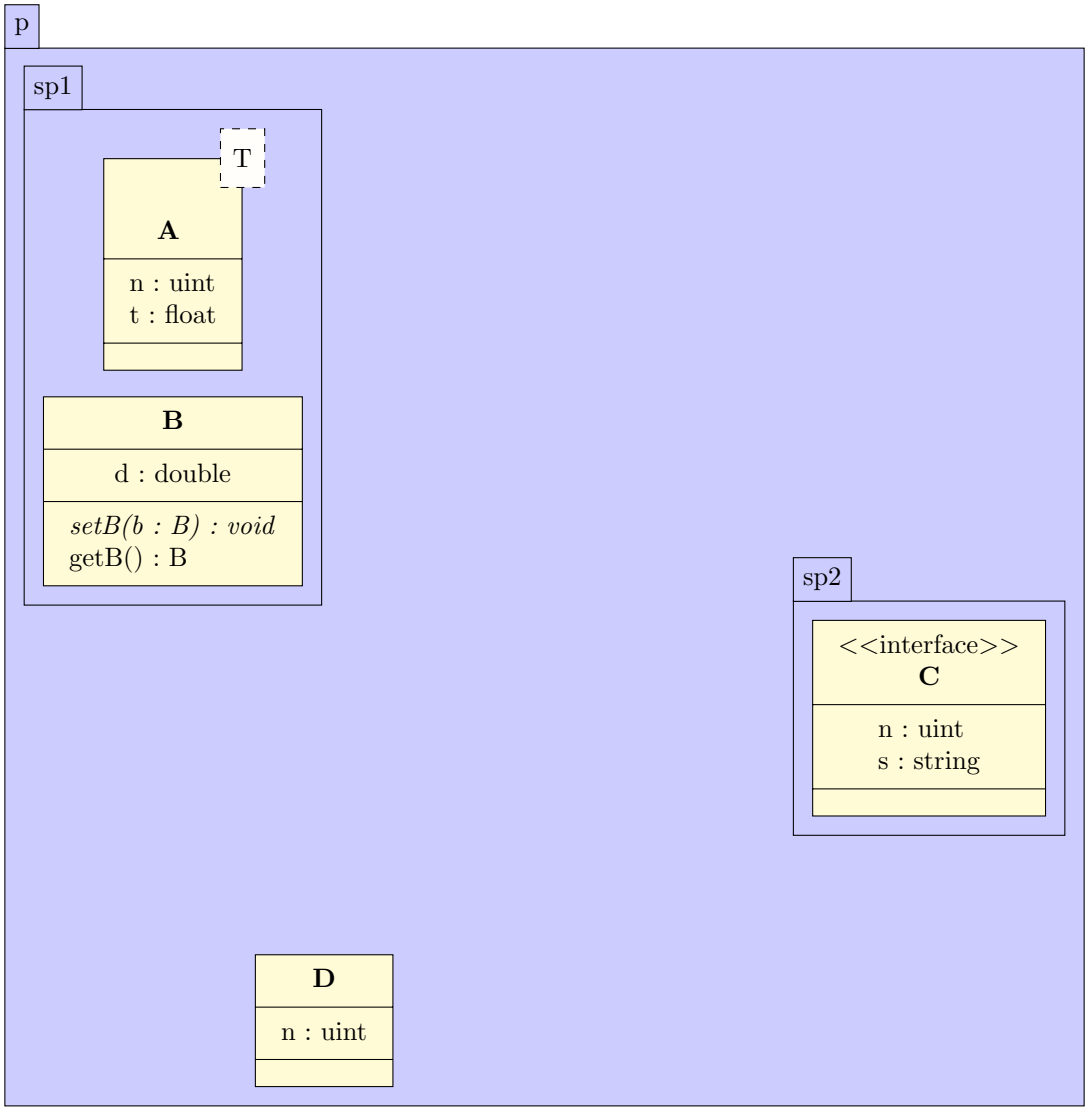
Définition des classes A, B, C, D et de leurs attributs et opérations

La classe A est en (0,0) dans le sous-package sp1 et a un paramètre template : T. La classe B est positionnée 3 unités en dessous de A, toujours dans le sous-package sp1, et possède un attribut statique et une opération virtuelle. La classe C est une interface en (0,0) dans le sous-package sp2. La classe D est placée en (2,-11) dans le package p.

La classe A a deux attributs. La classe B a un attribut et deux opérations dont une virtuelle. La classe C a deux attributs. La classe D a un attribut.

```
\begin{umlpackage}{p}
\begin{umlpackage}{sp1}
\umlclass[template=T]{A}{
  n : uint \\ t : float
}{}
\umlclass[y=-3]{B}{
  d : double
```

```
}{
  \umlvirt{setB(b : B) : void} \\ getB() : B}
\end{umlpackage}
\begin{umlpackage}[x=10,y=-6]{sp2}
\umlinterface{C}{
  n : uint \\ s : string
}{}
\end{umlpackage}
\umlclass[x=2,y=-10]{D}{
  n : uint
  }{}
\end{umlpackage}
```

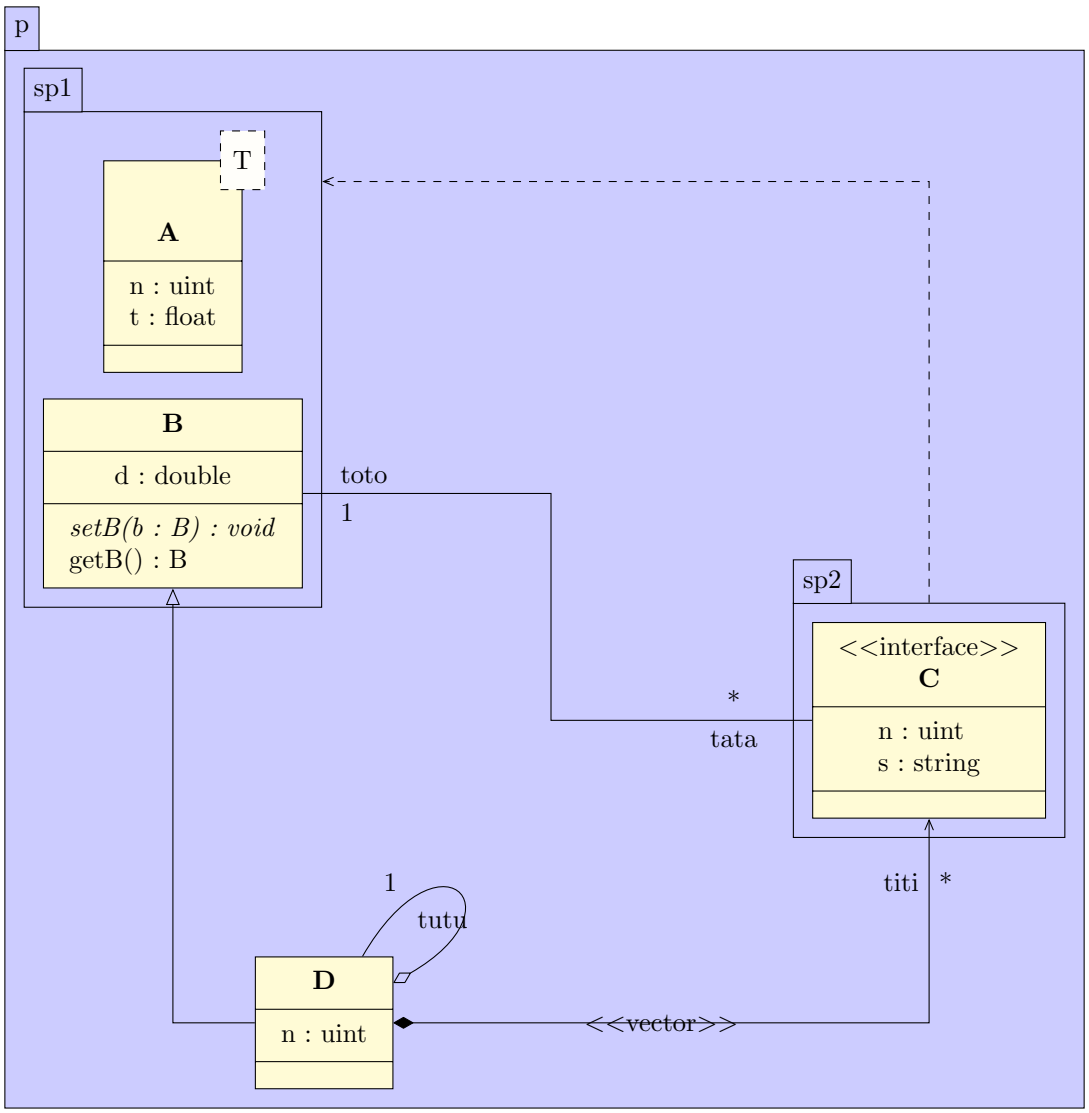


Définition des relations

On définit une association entre les classes C et B, une composition unidirectionnelle entre les classes D et C, une relation d'import nommée "import" entre les sous-packages sp2 et sp1 (avec modification des ancres), une relation d'aggrégation récursive sur la classe D et une relation d'héritage entre les classes D et B. Sur ces relations, on va spécifier des noms d'arguments et leurs multiplicités. Regardez la valeur donnée à la position de ces éléments suivant la géométrie de la flèche.

...

```
\umlassoc[geometry=-|-, arg1=tata, mult1=*, pos1=0.3, arg2=toto, mult2=1, pos
    2=2.9, align2=left]{C}{B}
\umlunicompo[geometry=-|, arg=titi, mult=*, pos=1.7, stereo=vector]{D}{C}
\umlimport[geometry=|-, anchors=90 and 50, name=import]{sp2}{sp1}
\umlaggreg[arg=tutu, mult=1, pos=0.8, angle1=30, angle2=60, loopsize=2cm]{D}{D}
\umlinherit[geometry=-|]{D}{B}
```

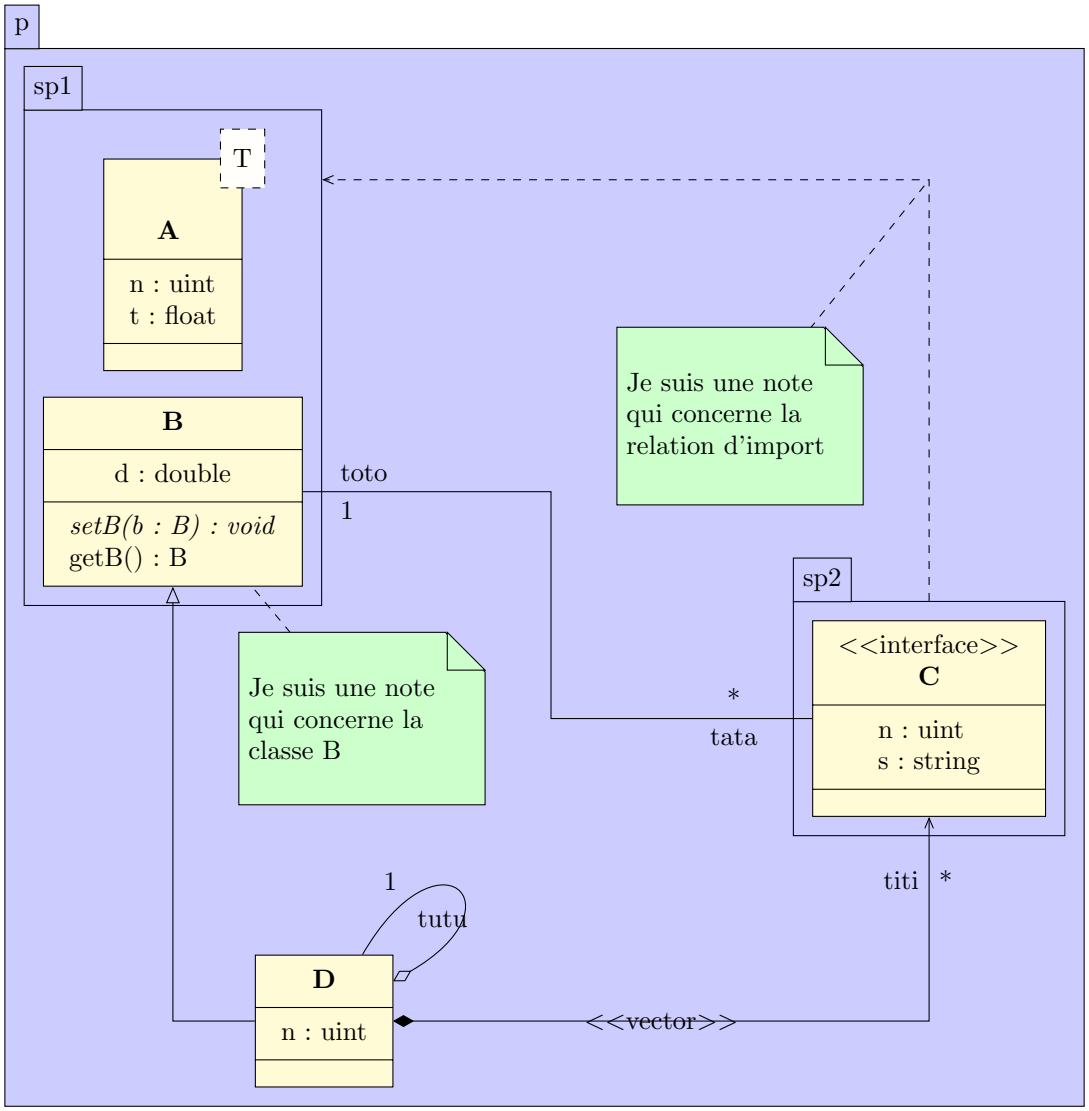


Définition des notes

On rajoute enfin une note attchée à la classe B et une note attachée à la relation d'import affectée du nom import.

...

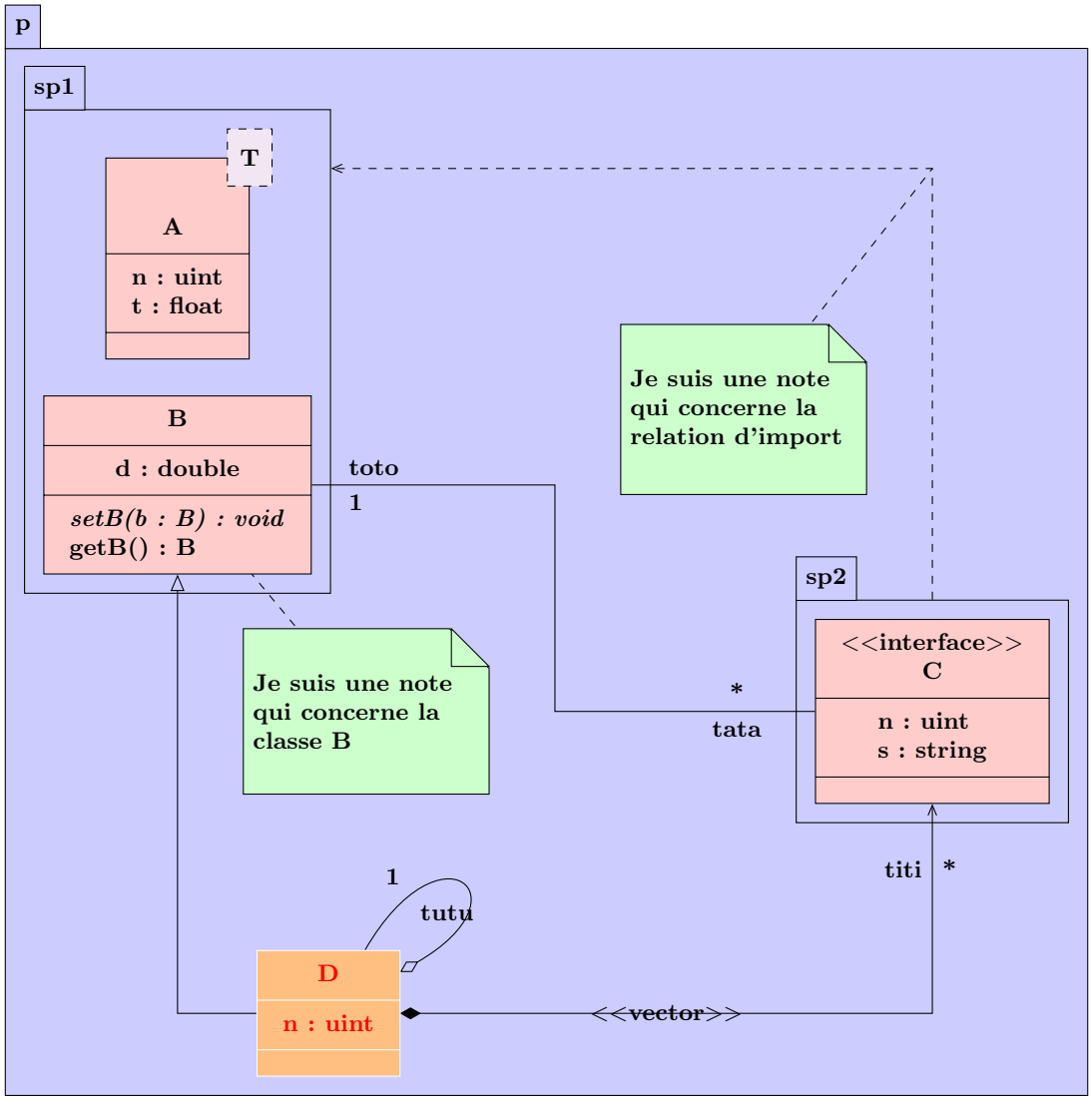
```
\umlnote[x=2.5,y=-6, width=3cm]{B}{Je suis une note qui concerne la classe B}
\umlnote[x=7.5,y=-2]{import-2}{Je suis une note qui concerne la relation d'import}
```



Modification du style

On illustre l'utilisation de la commande `tikzumlset` en changeant les couleurs associées à la classe et le type de font. On peut par ailleurs modifier les couleurs d'une classe donnée avec les options `draw`, `text` et `fill`

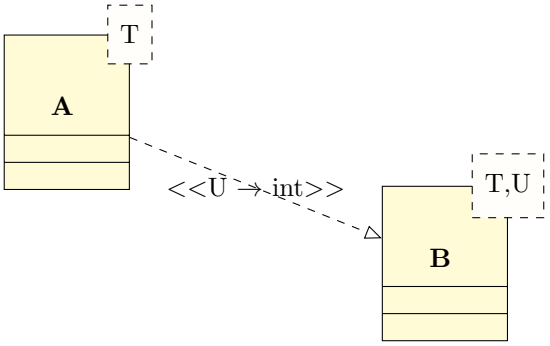
```
\tikzumlset{fill class=red!20, fill template=violet!10, font=\bfseries\
    footnotesize}
\begin{tikzpicture}
...
\umlclass[x=2,y=-11, fill=orange!50, draw=white, text=red]{D}{
  n : uint
  }{}
...
\end{tikzpicture}
```



1.5.2 Définir une spécialisation de classe

Une spécialisation de classe est de l'héritage d'un patron de classe dans lequel l'un des paramètres à son type fixé. Pour définir cette relation, c'est la commande `umlreal` qui va servir ici, ainsi que l'option `stereo` :

```
\begin{tikzpicture}
\umlemptyclass[template=T]{A}
\umlemptyclass[template={T,U}, x=5, y
    =-2]{B}
\umlreal[stereo={U $\rightarrow$ int}]{A
    }{B}
\end{tikzpicture}
```



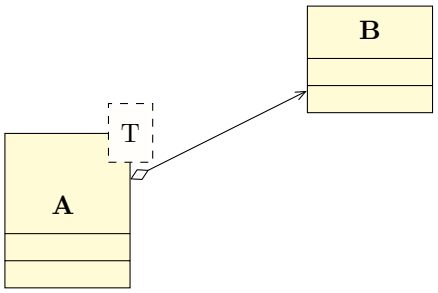
1.6 Règles de priorité des options et bugs identifiés

1. L'option `geometry` prime toujours sur les autres arguments. Cela signifie en particulier que si elle n'a pas sa valeur par défaut (– –), alors les options `angle1`, `angle2` et `loopsize`, paramétrant les

relations récursives, seront ignorées.

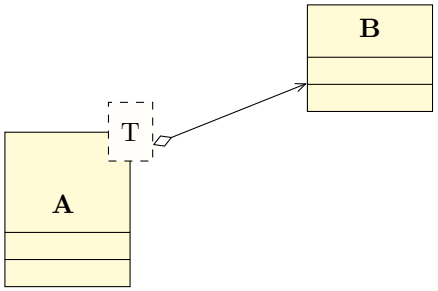
2. Dans le cas d'un patron de classe, il existe des cas où une relation la concernant sera mal définie, comme le montre le dessin ci-dessous, où le losange de la relation d'aggrégation est caché par le paramètre template :

```
\begin{tikzpicture}
\umlemptyclass[template=T]{A}
\umlemptyclass[x=4,y=2]{B}
\umluniaggreg{A}{B}
\end{tikzpicture}
```

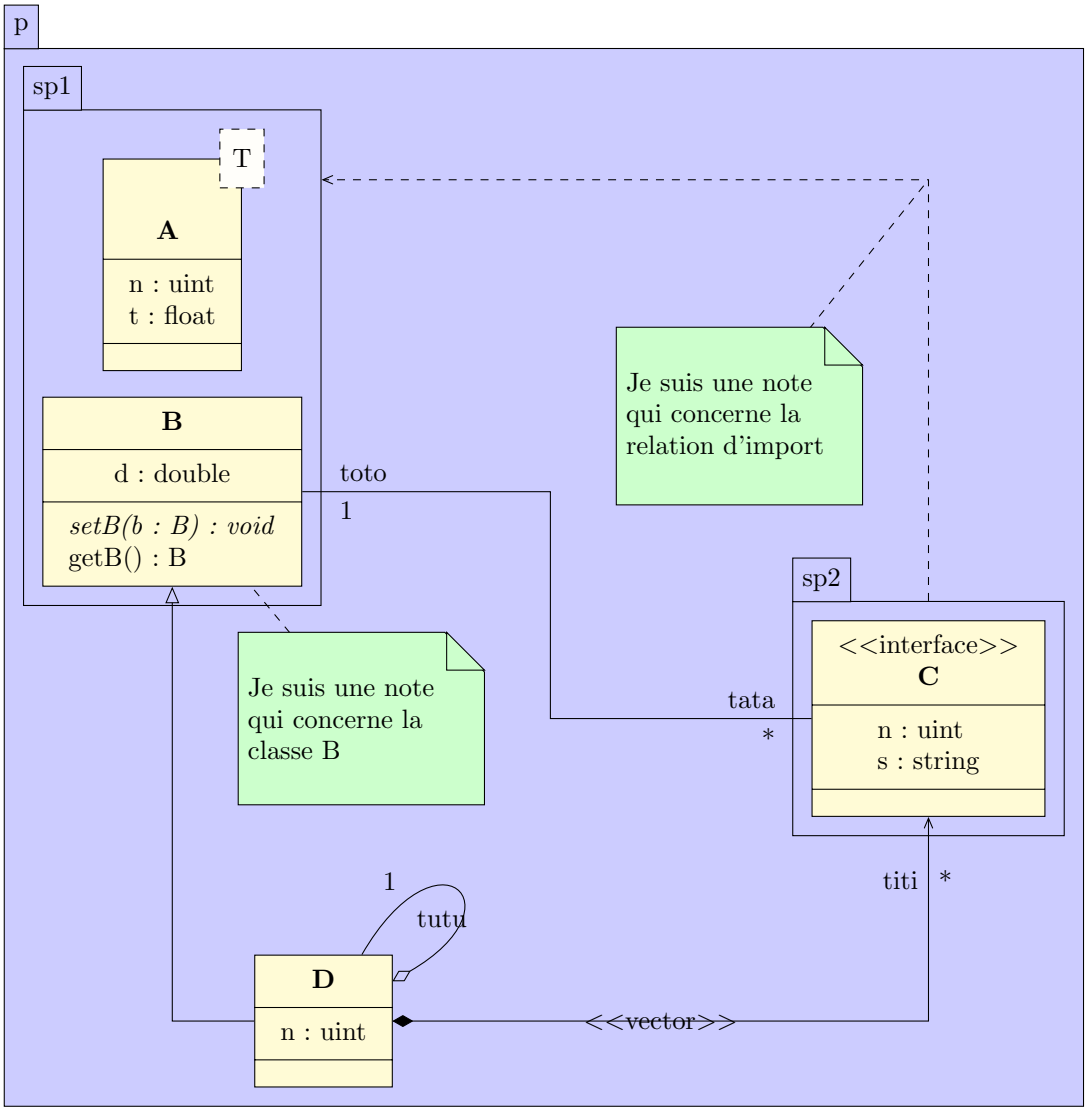


On peut toutefois corriger partiellement ce problème en reliant la flèche directement entre la partie template de la classe A et la classe B en rajoutant le suffixe -template et en ajustant l'ancre de départ (la valeur -30 est assez satisfaisante) :

```
\begin{tikzpicture}
\umlemptyclass[template=T]{A}
\umlemptyclass[x=4,y=2]{B}
\umluniaggreg[anchor1=-30]{A-template}{B}
\end{tikzpicture}
```



3. Le comportement de placement automatique des informations d'un attribut sur une relation peut surprendre quand on veut le court-circuiter. Reprenons l'exemple de l'introduction. Si l'on regarde la rela tion d'association et les attributs *toto* et *tata*. Si *toto* est au-dessus, *tata* est lui en dessous. Demandons maintenant de justifier à droite l'attribut *tata* (Au passage, on met sa position à 0.1). On constate alors que les positions de *tata* et de sa multiplicité s'inversent.


p
sp1
T
A
n : uint
t : float
B
d : double
setB(b : B) : void
getB() : B
toto
1
Je suis une note
qui concerne la
relation d'import
Je suis une note
qui concerne la
classe B
sp2
<<interface>>
C
n : uint
s : string
tata
*
titi
*
1
tutu
D
n : uint
<<vector>>

Chapitre 2

Diagrammes de cas d'utilisation

2.1 Système

2.2 Acteur et cas d'utilisation

2.3 Relations

2.4 Personnalisation

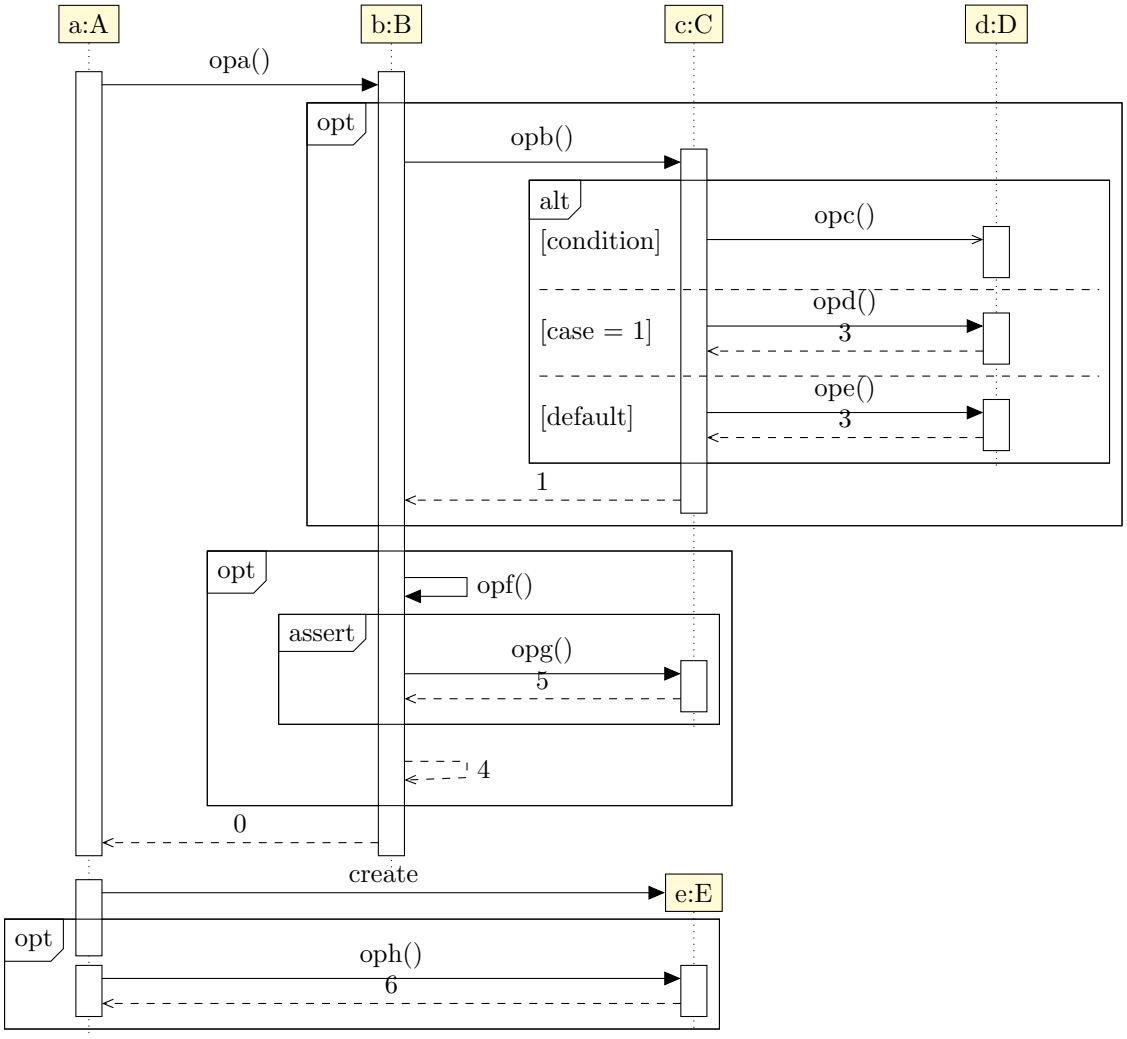
Chapitre 3

Diagrammes d'états-transition

Chapitre 4

Diagrammes de séquence

```
\begin{umlseqdiag}
\umlobj[class=A]{a}
\umlobj[class=B, x=4]{b}
\umlobj[class=C, x=8]{c}
\umlobj[class=D, x=12]{d}
\begin{umlcall}[op=opa(), type=synchron, return=0, dt=1.5]{a}{b}
\begin{umlfragment}
\begin{umlcall}[op=opb(), type=synchron, return=1, dt=0.5]{b}{c}
\begin{umlfragment}[type=alt, label=condition, inner xsep=1.1]
\begin{umlcall}[op=opc(), type=asynchron, dt=0.5]{c}{d}
\end{umlcall}
\umlfpart[case = 1]
\begin{umlcall}[op=opd(), type=synchron, return=3, dt=1]{c}{d}
\end{umlcall}
\umlfpart[default]
\begin{umlcall}[op=ope(), name=toto, type=synchron, return=3, dt=1]{c}{d}
\end{umlcall}
\end{umlfragment}
\end{umlcall}
\end{umlfragment}
\begin{umlfragment}
\begin{umlcallself}[op=opf(), type=synchron, return=4, dt=0.5]{b}
\begin{umlfragment}[type=assert]
\begin{umlcall}[op=opg(), type=synchron, return=5, dt=0.5]{b}{c}
\end{umlcall}
\end{umlfragment}
\end{umlcallself}
\end{umlfragment}
\end{umlcall}
\end{umlseqdiag}
```


a:A
b:B
c:C
d:D
opa()
opt
opb()
alt
[condition]
opc()
[case = 1]
opd()
3
[default]
ope()
3
1
opt
opf()
assert
opg()
5
4
0
create
e:E
opt
oph()
6