Résumé Cours UML

Diagramme de Cas d'Utilisation

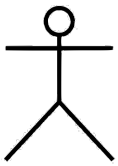
Le diagramme de cas d'utilisation est le premier diagramme du modèle UML. II permet de représenter les interactions fonctionnelles entre les acteurs et le système étudié. Il est particulièrement important pour l'organisation et l'identification des grandes fonctionnalités du système.

A. Éléments des diagrammes de cas d'utilisation

1. Acteur

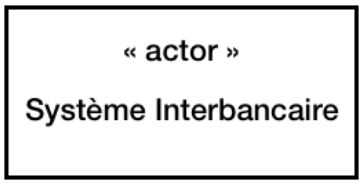
Un acteur est la représentation d'un rôle joué par une personne externe, un processus ou une chose qui interagit avec le système modélisé.

Il se représente par un petit bonhomme avec son nom (c.-à-d. son rôle) inscrit dessous.



Nom

Il est également possible de représenter un acteur sous la forme d'un classeur stéréotypé «actor».

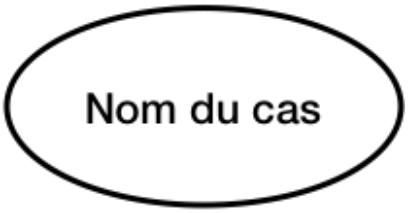


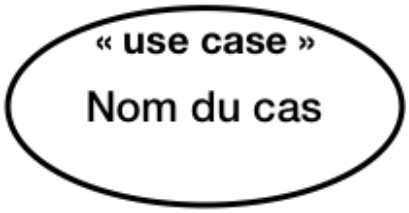
Il est recommandé d'utiliser la représentation rectangulaire pour les systèmes connectés.

2. Cas d'utilisation

Un cas d'utilisation est une unité irréductible représentant une fonctionnalité visible de l'extérieur. Il réalise un service pour l'acteur qui l'initie. Un cas d'utilisation modélise donc un service rendu par le système, sans imposer le mode de réalisation de ce service.

Un cas d'utilisation se représente par une ellipse contenant le nom du cas (un verbe à l'infinitif+complément).





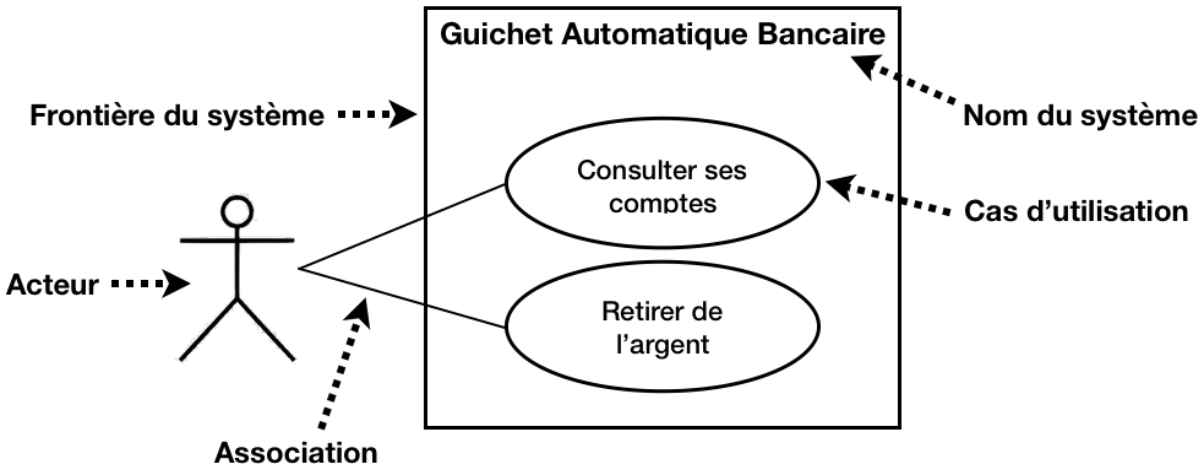
Ou bien ajouter optionnellement, au-dessus du nom, un stéréotype.

• **Cas d'utilisation interne**

Quand un cas n'est pas directement relié à un acteur, il est qualifié de *cas d'utilisation interne*.

3. Représentation d'un diagramme de cas d'utilisation

Comme le montre la figure suivante, la frontière du système est représentée par un cadre. Le nom du système se trouve à l'intérieur du cadre, en haut. Les acteurs sont à l'extérieur et les cas d'utilisation à l'intérieur.

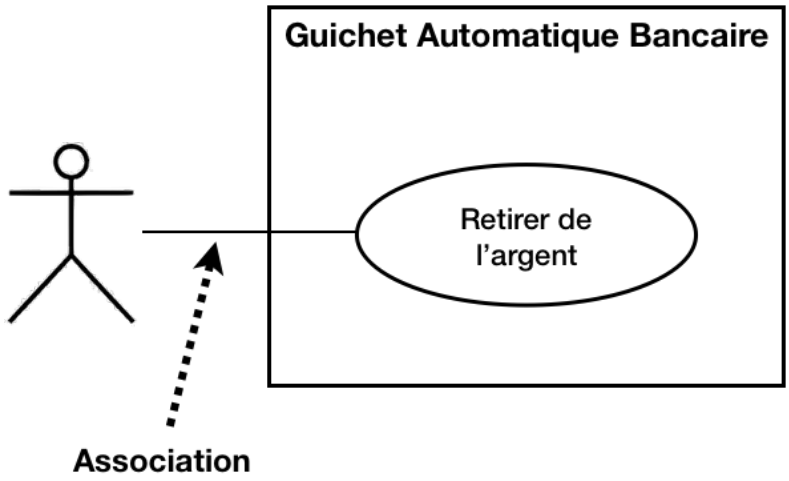


B. Relations dans des diagrammes de cas d'utilisation

1. Relations entre acteurs et cas d'utilisation

• **Relation d'association**

Une relation d'association est un chemin de communication entre un acteur et un cas d'utilisation. Elle matérialise une interaction entre l'acteur et le cas d'utilisation : l'acteur initie le cas d'utilisation ou le cas d'utilisation interagit ou rend service à l'acteur. La relation d'association est représentée par un trait continu.



• Acteurs principaux et secondaires

Un acteur est qualifié de *principal* pour un cas d'utilisation lorsque ce cas rend service à cet acteur. Les autres acteurs sont alors qualifiés de *secondaires*. Un cas d'utilisation possède au plus un acteur principal. Un acteur principal obtient un résultat observable du système tandis qu'un acteur secondaire est sollicité pour des informations complémentaires.

2. Relations entre cas d'utilisation

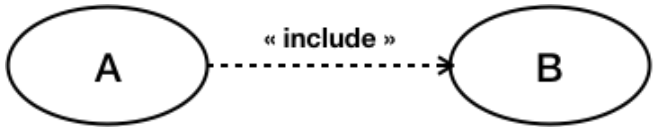
• Types de relations

Il existe principalement deux types de relations :

- les dépendances, qui sont explicitées par un stéréotype (les plus utilisés sont l'inclusion et l'extension),
- et la généralisation/spécialisation.

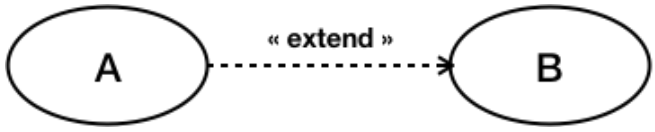
• Relations d'inclusion

Un cas d'utilisation A inclut un cas d'utilisation B si le comportement décrit par le cas A inclut le comportement du cas B : le cas A dépend de B et la flèche est dirigée de A vers B. Lorsque A est sollicité, B l'est obligatoirement, comme une partie de A. Cette dépendance est symbolisée par le stéréotype « include ».



• Relations d'extension

Un cas d'utilisation A étend un cas d'utilisation B lorsque le cas d'utilisation A peut être appelé au cours de l'exécution du cas d'utilisation B. Le cas A dépend de B et la flèche est dirigée de A vers B. Exécuter B peut éventuellement entraîner l'exécution de A : contrairement à l'inclusion, l'extension est optionnelle. Cette dépendance est symbolisée par le stéréotype « extend ».



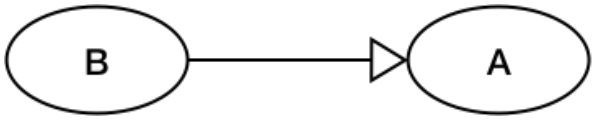
• Point d'extension

L'extension peut intervenir à un point précis du cas étendu. Ce point s'appelle le point d'extension. Il porte un nom, qui figure dans un compartiment du cas étendu sous la rubrique point d'extension, et est éventuellement associé à une contrainte indiquant le moment où

l'extension intervient. Une extension est souvent soumise à condition. Graphiquement, la condition est exprimée sous la forme d'une note.

- **Relations de généralisation**

Un cas A est une généralisation d'un cas B si B est un cas particulier de A. Cette relation de généralisation/spécialisation est présente dans la plupart des diagrammes UML et se traduit par le concept d'héritage dans les langages orientés objet.



3. Relations entre acteurs

La seule relation possible entre deux acteurs est la généralisation : un acteur A est une généralisation d'un acteur B si l'acteur A peut être substitué par l'acteur B. Dans ce cas, tous les cas d'utilisation accessibles à A le sont aussi à B, mais l'inverse n'est pas vrai. Le symbole utilisé pour la généralisation entre acteurs est une flèche avec un trait plein dont la pointe est un triangle fermé désignant l'acteur le plus général (comme nous l'avons déjà vu pour la relation de généralisation entre cas d'utilisation).

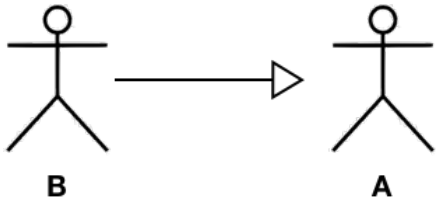


Diagramme de classes

Le diagramme de classes est considéré comme le plus important de la modélisation orientée objet, il est le seul obligatoire lors d'une telle modélisation. Il représente la vue statique du système en modélisant les concepts du domaine, les entités abstraites nécessaires à l'implémentation de l'application et les relations que ces concepts ou entités entretiennent entre eux.

A. Classe

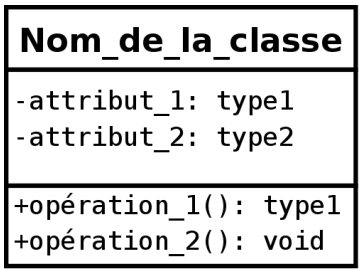
1. Notions de classe, d'objet et d'instance

Une classe est la description formelle d'un ensemble d'objets ayant une sémantique et des caractéristiques communes. Tout système orienté objet est organisé autour des classes.

Un objet est la concrétisation d'une classe. Nous parlerons également d'instance d'une classe.

2. Représentation graphique

Une classe est représentée par un rectangle divisé en trois compartiments. Le premier indique le nom de la classe, le deuxième ses attributs et le troisième ses opérations.



3. Visibilité

La visibilité déclare la possibilité ou l'impossibilité pour un élément d'être accessible depuis une autre classe. Il existe quatre visibilités prédéfinies.

Public ou **+** : tout élément qui peut voir la classe peut également voir l'élément indiqué.

Protected ou # : seul un élément situé dans la classe ou un de ses descendants peut voir l'élément indiqué.

Private ou **-** : seul un élément situé dans la classe peut voir l'élément.

Package ou ~ : seul un élément déclaré dans le même paquetage peut voir l'élément.

Dans une classe, le marqueur de visibilité se situe au niveau de chacune de ses caractéristiques (attributs et opération). Il permet d'indiquer si une autre classe peut y accéder.

Dans la pratique, lorsque des attributs doivent être accessibles de l'extérieur, il est préférable que cet accès ne soit pas direct mais se fasse par l'intermédiaire d'opérations.

4. Syntaxe de la déclaration d'un attribut

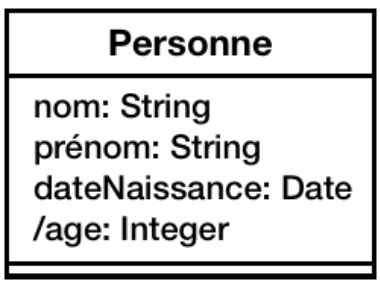
<visibilité> [/] <nom_attribut> : <type> ['['<multiplicité>']' [{<contrainte>}]] [= <valeur_par_défaut>]

Le type de l'attribut (<type>) peut être un nom de classe, un nom d'interface ou un type de donné prédéfini. La multiplicité (<multiplicité>) d'un attribut précise le nombre de valeurs que l'attribut peut contenir. Lorsqu'une multiplicité supérieure à 1 est précisée, il est possible d'ajouter une contrainte ({<contrainte>}) pour préciser si les valeurs sont ordonnées ({ordered}) ou pas ({list}).

5. Attribut dérivé (/)

Les attributs dérivés peuvent être calculés à partir d'autres attributs. Généralement, un attribut dérivé se concrétisera par une opération retournant la valeur calculée.

Les attributs dérivés sont symbolisés par l'ajout d'un « / » devant leur nom.



6. Syntaxe de la déclaration d'une opération

<visibilité> <nom_opération> ([<paramètre_1>, ... , <paramètre_N>]) : [<type_renvoyé>] [{<propriétés>}]

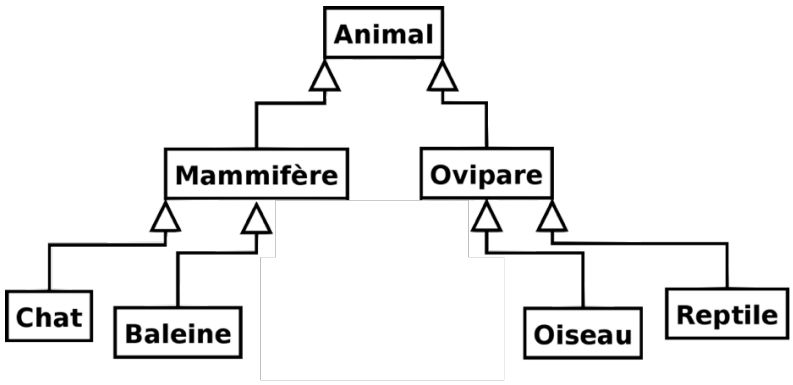
Les propriétés (<propriétés>) correspondent à des contraintes ou à des informations complémentaires comme les exceptions, les préconditions, les postconditions ou encore l'indication qu'une opération est abstraite (mot-clef {abstract}).

B. Relations entre classes

1. Généralisation et Héritage

La généralisation décrit une relation entre une classe générale (classe de base ou classe parent) et une classe spécifique (sous-classe ou classe enfant). La classe spécifique possède (par héritage) toutes les caractéristiques de la classe générale. Chaque instance de la classe spécifique est également une instance de la classe générale.

Le symbole utilisé pour la relation d'héritage ou de généralisation est une flèche avec un trait plein dont la pointe est un triangle fermé désignant le cas le plus général.



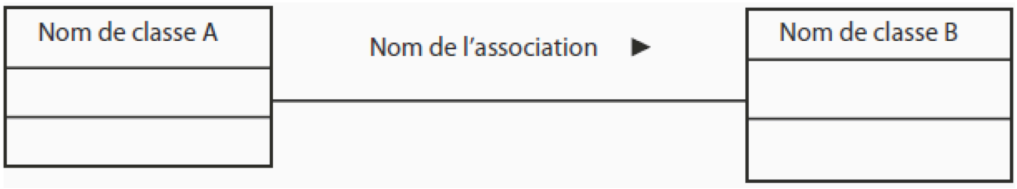
Les propriétés principales de l'héritage sont :

- La classe enfant possède toutes les caractéristiques des ses classes parents, mais elle ne peut accéder aux caractéristiques privées de cette dernière.
- Toutes les associations de la classe parent s'appliquent aux classes dérivées.
- Une instance d'une classe peut être utilisée partout où une instance de sa classe parent est attendue.
- Une classe enfant peut redéfinir (même signature) une ou plusieurs opérations de la classe parent. Sauf indication contraire, un objet utilise les opérations les plus spécialisées dans la hiérarchie des classes mêmes quand l'objet est perçu comme un objet d'une classe parent.

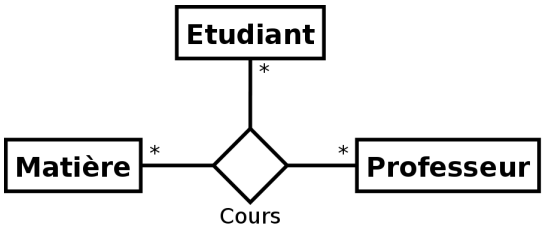
2. Association binaire et n-aire

Une association est une relation structurelle qui indique que les instances d'une classe sont reliées aux instances d'une autre classe. Le fait que deux instances soient ainsi liées permet la navigation d'une instance vers l'autre, et vice versa. Une association qui relie deux classes est qualifiée de binaire tandis qu'une association reliant plus de deux classes est appelée association n-aire.

Une association binaire est matérialisée par un trait plein entre les classes associées. Elle peut être ornée d'un nom, avec éventuellement une précision du sens de lecture (▶ ou ◀).



Une association n-aire lie plus de deux classes. On représente une association n-aire par un grand losange avec un chemin partant vers chaque classe participante. Le nom de l'association, le cas échéant, apparaît à proximité du losange.



3. Multiplicité ou cardinalité

La multiplicité associée à une terminaison d'association, d'agrégation ou de composition déclare le nombre d'objets susceptibles d'occuper la position définie par la terminaison d'association. Voici quelques exemples de multiplicité :

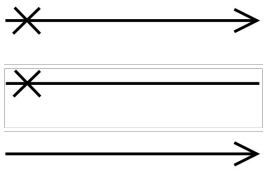
- exactement un : 1 ou 1..1
- plusieurs : * ou 0..*
- au moins un : 1..*
- de un à quatre : 1..4

Dans une association binaire, la multiplicité sur la terminaison cible contraint le nombre d'objets de la classe cible pouvant être associés à un seul objet donné de la classe source (la classe de l'autre terminaison de l'association).

4. Navigabilité

La navigabilité indique s'il est possible de traverser une association. On représente graphiquement la navigabilité par une flèche du côté de la terminaison navigable et on empêche la navigabilité par une croix du côté de la terminaison non navigable. Par défaut, une association est navigable dans les deux sens.

Lorsque l'on représente la navigabilité uniquement sur l'une des extrémités d'une association, il faut remarquer que, implicitement, les trois associations représentées sur la figure suivante ont la même signification : l'association ne peut être traversée que dans un sens.



5. Agrégation et composition

• Agrégation

Une association simple entre deux classes représente une relation structurelle entre pairs, c'est à dire entre deux classes de même niveau conceptuel : aucune des deux n'est plus importante que l'autre. Lorsque l'on souhaite modéliser une relation tout/partie où une classe constitue un élément plus grand (tout) composé d'éléments plus petit (partie), il faut utiliser une agrégation.

Une agrégation est une association qui représente une relation d'inclusion structurelle ou comportementale d'un élément dans un ensemble. Graphiquement, on ajoute un losange vide (◇) du côté de l'agrégat.

• Composition

La composition, également appelée agrégation composite, décrit une contenance structurelle entre instances. Ainsi, la destruction de l'objet composite implique la destruction de ses composants. Une instance de la partie appartient toujours à au plus une instance de l'élément composite : la multiplicité du côté composite ne doit pas être supérieure à 1. Graphiquement, on ajoute un losange plein (◆) du côté de l'agrégat.

6. Classe-association

Parfois, une association doit posséder des propriétés. Par exemple, l'association Emploie entre une société et une personne possède comme propriétés le salaire et la date d'embauche. En effet, ces deux propriétés n'appartiennent ni à la société, qui peut employer plusieurs personnes, ni aux personnes, qui peuvent avoir plusieurs emplois. Il s'agit donc bien de propriétés de l'association Emploie. Les associations ne pouvant posséder de propriété, il faut introduire un nouveau concept pour modéliser cette situation : celui de classe-association.

Une classe-association possède les caractéristiques des associations et des classes : elle se connecte à deux ou plusieurs classes et possède également des attributs et des opérations.

Une classe-association est caractérisée par un trait discontinu entre la classe et l'association qu'elle représente.

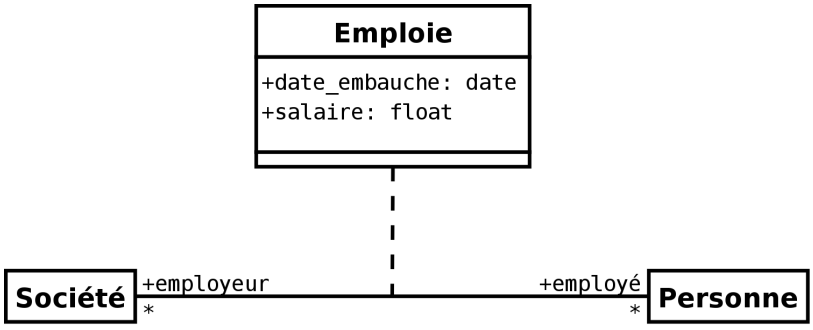


Diagramme d'objets

Un diagramme d'objets représente des objets (c-à-d instances de classes) et leurs liens (c-à-d instances de relations) pour donner une vue figée de l'état d'un système à un instant donné.

Un diagramme d'objets peut être utilisé pour :

- illustrer le modèle de classes en montrant un exemple qui explique le modèle ;
- préciser certains aspects du système en mettant en évidence des détails imperceptibles dans le diagramme de classes ;
- exprimer une exception en modélisant des cas particuliers ou des connaissances non généralisables qui ne sont pas modélisés dans un diagramme de classe ;
- prendre une image (snapshot) d'un système à un moment donné.

Le diagramme de classes modélise les règles et le diagramme d'objets modélise des faits.

Graphiquement, un objet se représente comme une classe. Cependant, le compartiment des opérations n'est pas utile. De plus, le nom de la classe dont l'objet est une instance est précédé d'un « : » et est souligné. Pour différencier les objets d'une même classe, leur identifiant peut être ajouté devant le nom de la classe. Enfin les attributs reçoivent des valeurs. Quand certaines valeurs d'attribut d'un objet ne sont pas renseignées, on dit que l'objet est partiellement défini.

Dans un diagrammes d'objets, les relations du diagramme de classes deviennent des liens. La relation de généralisation ne possède pas d'instance, elle n'est donc jamais représentée dans un diagramme d'objets. Graphiquement, un lien se représente comme une relation, mais, s'il y a un nom, il est souligné. Naturellement, on ne représente pas les multiplicités.

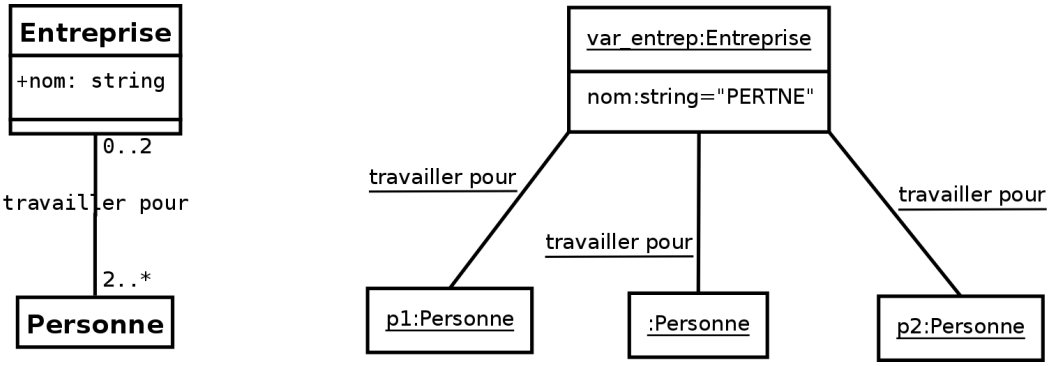
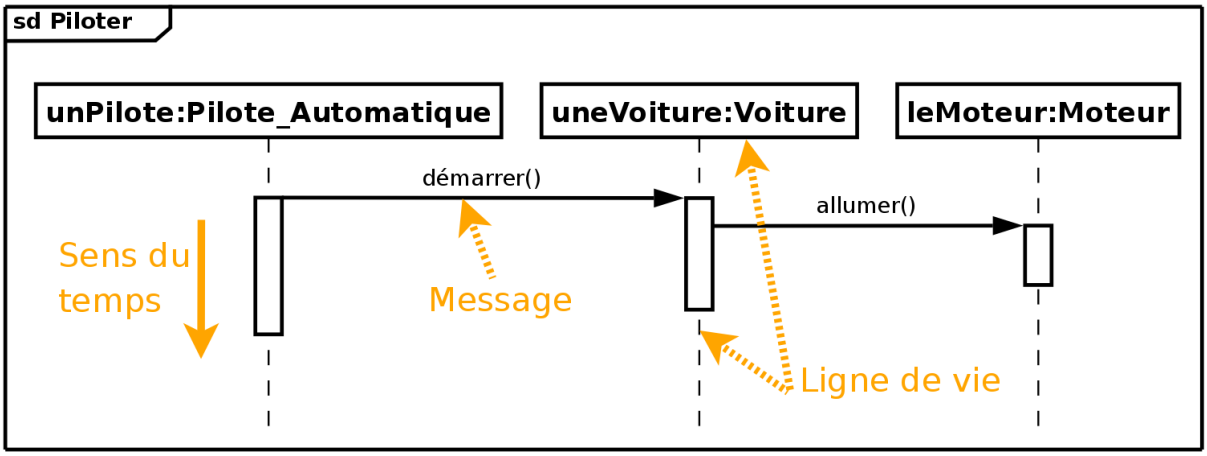


Diagramme de séquence

Les principales informations contenues dans un diagramme de séquence sont les messages échangés entre les lignes de vie, présentés dans un ordre chronologique. Ainsi le temps y est représenté explicitement par une dimension (la dimension verticale) et s'écoule de haut en bas.



1. Représentation des lignes de vie

Une ligne de vie se représente par un rectangle, auquel est accroché une ligne verticale pointillée, contenant une étiquette dont la syntaxe est :

[<nom_du_rôle>] : [<Nom_du_type>]

Au moins un des deux noms doit être spécifié dans l'étiquette, les deux points (:) sont, quand à eux, obligatoire.

2. Représentation des messages

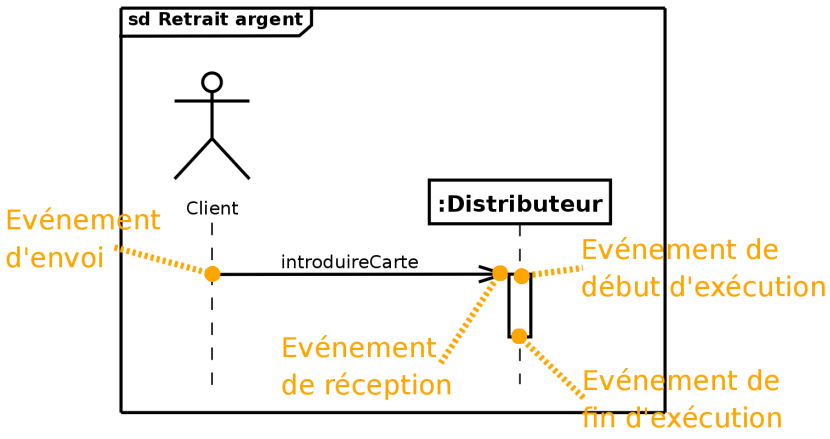
Un message définit une communication particulière entre des lignes de vie. Plusieurs types de messages existent, les plus commun sont :

– l'envoi d'un signal ;

– l'invocation d'une opération ;

– la création ou la destruction d'une instance.

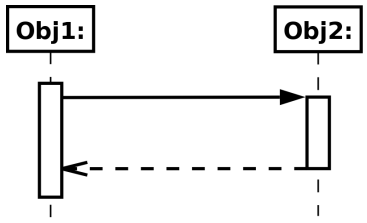
UML permet de séparer clairement l'envoi du message, sa réception, ainsi que le début de l'exécution de la réaction et sa fin.



L'invocation d'une opération est le type de message le plus utilisé en programmation objet. L'invocation peut être asynchrone ou synchrone. Dans la pratique, la plupart des invocations sont synchrones, l'émetteur reste alors bloqué le temps que dure l'invocation de l'opération.

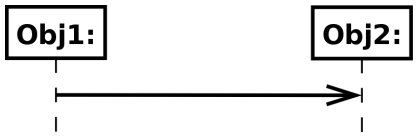
- **Messages synchrones**

Un message synchrone se représente par une flèche en traits pleins et à l'extrémité pleine partant de la ligne de vie d'un objet expéditeur et allant vers celle de l'objet cible. Ce message peut être suivi d'une réponse qui se représente par une flèche en pointillé.



- **Messages asynchrones**

Un message asynchrone se représente par une flèche en traits pleins et à l'extrémité ouverte partant de la ligne de vie d'un objet expéditeur et allant vers celle de l'objet cible.



- **Syntaxe des messages et des réponses**

Dans la plupart des cas, la réception d'un message est suivie de l'exécution d'une méthode d'une classe.Cette méthode peut recevoir des arguments et la syntaxe des messages permet de transmettre ces arguments. La syntaxe de réponse à un message est la suivante :
[<attribut> =] message [: <valeur_de_retour>]
où message représente le message d'envoi.

La figure suivante montre un exemple d'exécution d'une méthode avec une réponse.