# Diagrammes de cas d'utilisation

## Objectifs

- Présenter les concepts UML relatifs à la vue fonctionnelle (diagramme de cas d'utilisation)
- Présenter la notation graphique du diagramme de cas d'utilisation
- Expliquer la sémantique des cas d'utilisation UML en précisant le lien avec les interactions UML

## Vue fonctionnelle du modèle UML

La partie fonctionnelle du modèle UML d'une application permet de spécifier les fonctionnalités offertes par l'application sans pour autant spécifier la façon dont ces fonctionnalités sont réalisées par les objets de l'application.

### *Fonctionnalités d'une application orientée objet*

Le principe fondateur du paradigme objet est de réunir en une seule et même entité, l'objet, des données et des traitements. À l'époque de la création de ce paradigme, ce principe était considéré comme révolutionnaire, car il allait à rebours des paradigmes existants (fonctionnel et données), qui visaient à séparer les données et les traitements dans des entités différentes.

L'avantage le plus important qu'apporte ce principe fondateur est de permettre l'identification, la décomposition et la réutilisation d'entités capables de réaliser des traitements uniquement grâce aux données qu'elles possèdent. Avec le paradigme objet, il est possible de considérer une application comme un ensemble d'objets indépendants mais cohérents, chacun réalisant la tâche pour laquelle il a été conçu. Ainsi, si une tâche réalisée par un objet est nécessaire dans une autre application, il est possible de réutiliser l'objet.

L'inconvénient de ce principe fondateur est qu'il masque les fonctionnalités offertes par des groupes d'objets. En effet, leur spécification n'est pas explicite et est répartie dans les traitements et dans les interactions réalisés par chacun des objets participant au groupe. De ce fait, il est très difficile de spécifier les besoins fonctionnels que nous avons sur une application à l'aide d'objets. Ces besoins fonctionnels, qui s'expriment naturellement à l'aide de fonctions, ne peuvent donc être exprimés simplement sous forme de groupes d'objets.

Par exemple, si nous voulions spécifier à l'aide d'objets les besoins fonctionnels d'une application de gestion de prêts bancaires tels que la création d'un prêt ou le calcul d'un taux d'intérêt, il faudrait spécifier l'ensemble des objets participant à la réalisation de l'application et spécifier chacun des traitements associés à ces objets. Si cela reste faisable, ce n'est guère naturel pour nous développeurs, plutôt habitués à utiliser le découpage fonctionnel.

Pour résoudre ce problème et réconcilier le paradigme objet avec la possibilité d'exprimer les besoins d'une application sous forme de fonctions, UML définit le concept de cas d'utilisation.

## Concepts élémentaires

Cette section présente les concepts élémentaires de la vue fonctionnelle d'un modèle UML. Dans notre contexte, ces concepts sont suffisants pour exprimer les fonctionnalités d'une application.

### Cas d'utilisation

*Sémantique*

Un cas d'utilisation spécifie une fonction offerte par l'application à son environnement. Un cas d'utilisation est spécifié uniquement par un intitulé.

Nous recommandons que l'intitulé du cas d'utilisation respecte le pattern « verbe + compléments ». Le verbe de l'intitulé permet de spécifier la nature de la fonctionnalité offerte par l'application, tandis que les compléments permettent de spécifier les données d'entrée ou de sortie de la fonctionnalité.

Par exemple, le cas d'utilisation « calculer taux d'intérêt du prêt » permet de comprendre d'une certaine manière que l'application permet à ses utilisateurs de calculer le taux d'intérêt d'un prêt.

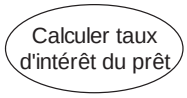
Le concept de cas d'utilisation offre une vue fonctionnelle sur l'application. La façon dont sera réalisé concrètement un cas d'utilisation n'apparaît pas dans la définition du cas d'utilisation. Elle sera précisée par la suite, lors de l'établissement des liens de cohérence avec les autres parties du modèle.

*Graphique*

Un cas d'utilisation se représente par une ellipse contenant l'intitulé du cas d'utilisation.

La figure 9.1 représente le cas d'utilisation que nous avons introduit à la section précédente.

**Figure 9.1**
*Représentation graphique d'un cas d'utilisation*




### Acteur

*Sémantique*

Un acteur représente une entité appartenant à l'environnement de l'application qui interagit avec l'application.

Le concept d'acteur permet de classifier les entités externes à l'application. Un acteur est identifié par un nom.

*Graphique*

Un acteur se représente par un petit bonhomme et un nom (nom du rôle).

La figure 9.2 représente l'acteur `Client`.

**Figure 9.2**
*Représentation graphique d'un acteur*


### Système

*Sémantique*

Un système représente une application dans le modèle UML. Il est identifié par un nom et regroupe un ensemble de cas d'utilisation qui correspondent aux fonctionnalités offertes par l'application à son environnement.

L'environnement est spécifié sous forme d'acteurs liés aux cas d'utilisation.

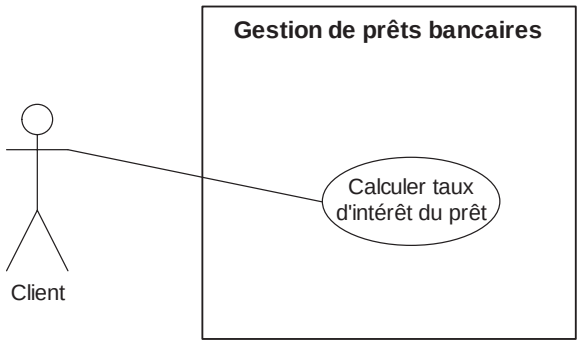
*Graphique*

Un système se représente par un rectangle contenant le nom du système et les cas d'utilisation de l'application.

Les acteurs, extérieurs au système, sont représentés et reliés aux cas d'utilisation qui les concernent. L'ensemble correspond à un diagramme de cas d'utilisation.

La figure 9.3 représente le diagramme de cas d'utilisation d'une application de gestion de prêts bancaires avec son unique cas d'utilisation offert à l'acteur qui représente le client.

**Figure 9.3**
*Diagramme de cas d'utilisation*




## Liens avec les autres parties du modèle

Nous venons de voir que la partie fonctionnelle du modèle UML permettait de spécifier les fonctionnalités d'une application mais aussi de préciser à quelles entités externes ces fonctionnalités sont offertes.

Il est possible d'élaborer plusieurs diagrammes de cas d'utilisation à chaque niveau d'abstraction permettant de regrouper les fonctionnalités de l'application en différents sous-systèmes. Cependant, nous considérons dans ce cours qu'il suffit d'élaborer un unique diagramme de cas d'utilisation par niveau d'abstraction. Ce diagramme représente les fonctionnalités principales de l'application à un niveau d'abstraction donné et les principaux bénéficiaires de ces fonctionnalités. Il correspond en quelque sorte à l'emballage commercial des applications vendues en grande surface, sur lequel sont écrites les fonctionnalités offertes par l'application à ses utilisateurs.

Nous savons que les fonctionnalités d'une application orientée objet sont réalisées par les objets qui composent l'application. Ces objets sont spécifiés à l'aide des classes présentes dans la partie structurelle du modèle de l'application, alors que les fonctionnalités sont spécifiées dans la partie fonctionnelle du modèle. Il est donc nécessaire de faire apparaître un lien de cohérence entre les parties structurelle et fonctionnelle du modèle UML afin de savoir quels sont les objets réalisant telle ou telle fonctionnalité.

Pour établir ce lien entre les parties structurelle et fonctionnelle du modèle UML, nous préconisons d'utiliser les interactions présentes dans la partie comportementale du

modèle de l'application. L'idée sous-jacente est de faire correspondre à chaque cas d'utilisation une ou plusieurs interactions spécifiant un exemple de réalisation de la fonctionnalité. Ainsi, les cas d'utilisation sont en cohérence avec les interactions, lesquelles sont en cohérence avec les classes, puisque les objets qui apparaissent dans les interactions sont typés par des classes spécifiées dans la partie structurelle du modèle.

Plus précisément, nous préconisons de réaliser pour chaque cas d'utilisation :

- au moins une interaction spécifiant l'exécution normale de l'application ;
- une interaction spécifiant les exécutions soulevant des erreurs de l'application.

La figure 9.4 illustre les relations de cohérence entre les parties fonctionnelle, comportementale et structurelle du modèle d'une application. Nous constatons que les cas d'utilisation du diagramme de cas d'utilisation sont reliés à des diagrammes de séquence et que les objets de ces diagrammes de séquence sont reliés à des classes.

Par rapport à notre vision schématique du modèle UML d'une application, ces liens entre les vues fonctionnelle, comportementale et structurelle existent à chacun des niveaux d'abstraction du modèle. Nous verrons au chapitre 10 de ce cours comment mettre en œuvre les relations de cohérence entre les différents niveaux d'abstraction.

Dans chacune de ces interactions, nous préconisons de faire apparaître les objets correspondant aux acteurs qui utilisent la fonctionnalité. Il est d'ailleurs possible de faire en sorte que le type d'un objet participant à une interaction soit un acteur (et non une classe).

Afin d'améliorer la visibilité des diagrammes de séquence représentant ces interactions, nous préconisons de faire apparaître les objets représentant des acteurs à gauche du diagramme. De plus, nous préconisons de réutiliser, autant que possible, les mêmes objets dans toutes les interactions spécifiées. Cela donne une meilleure visibilité au modèle en ne multipliant pas inutilement le nombre des objets.

Notons qu'il est possible d'exploiter ces interactions afin de définir des interactions de test *(voir le chapitre 7)*, ce que nous ne ferons pas dans le cadre de ce cours.

## Concepts avancés

Les concepts avancés que nous présentons dans cette section permettent de spécifier plus finement les fonctionnalités et l'environnement d'une application. Ces concepts sont toutefois si délicats à employer que nous déconseillons fortement leur utilisation. Nous ne les présentons qu'afin de compléter notre présentation du diagramme de cas d'utilisation.

Soulignons en outre que ces concepts n'ont quasiment pas évolué entre les versions UML 1.4 et UML 2.1.

### *Concepts avancés relatifs aux cas d'utilisation*

Cette section présente les concepts avancés applicables aux cas d'utilisation. Ces concepts sont essentiellement des relations entre cas.

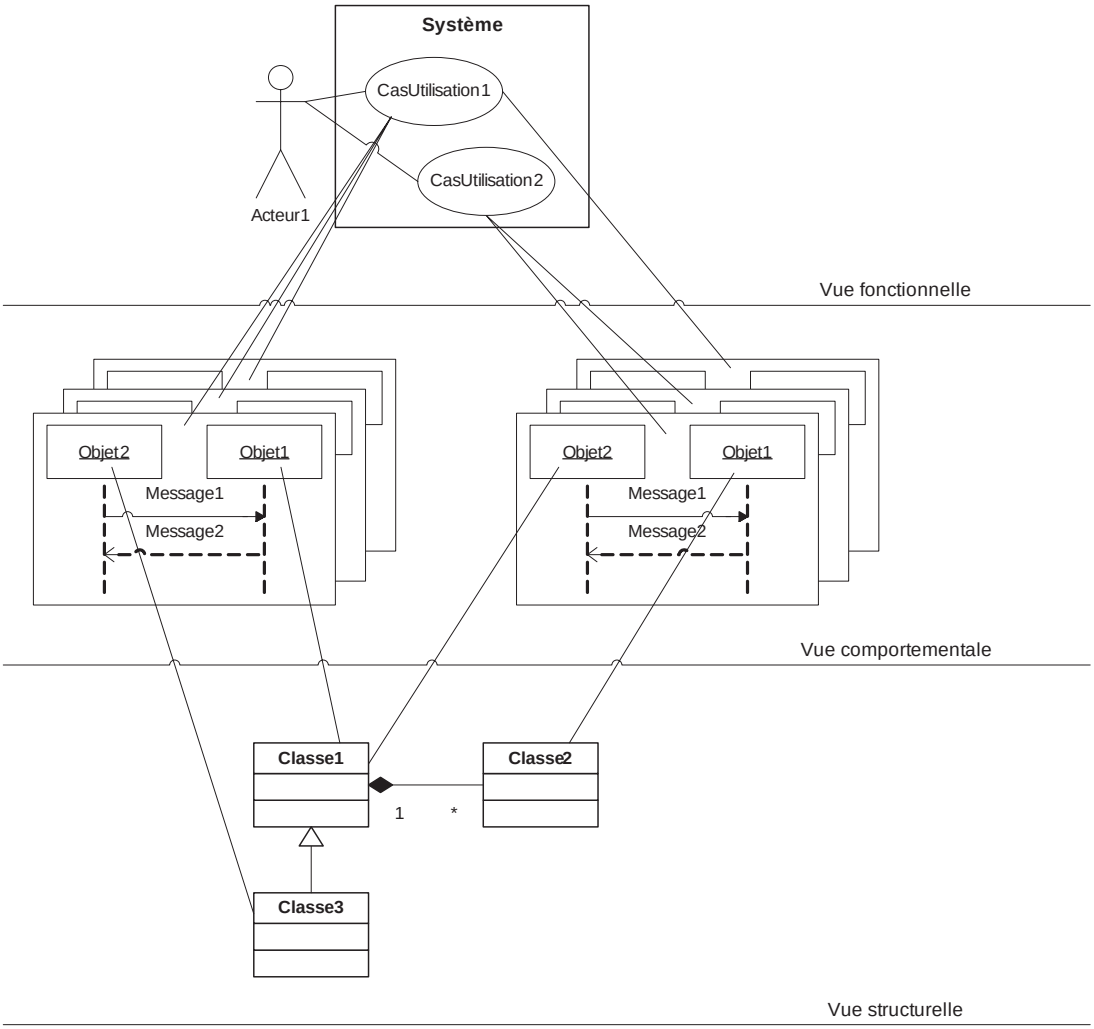



**Figure 9.4**
*Liens de cohérence entre les vues d'un modèle*

### include

*Sémantique*

Il est possible de spécifier qu'un cas d'utilisation inclut un autre cas d'utilisation.

Étant donné que les cas d'utilisation correspondent à des fonctions, la relation d'inclusion entre deux cas d'utilisation peut être comparée à une relation d'inclusion de fonctions. En d'autres termes, si un cas d'utilisation C2 hérite d'un cas d'utilisation C1, l'exécution de C1 fait appel à C2.

La relation d'inclusion entre cas d'utilisation doit cependant être employée avec parcimonie. L'idéal est de n'y recourir que lorsqu'un cas d'utilisation est inclus dans au moins trois ou quatre autres cas, car cela permet de factoriser le cas inclus.

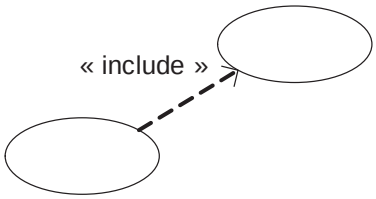
Soulignons le fait que la relation d'inclusion ne doit pas être utilisée pour exprimer une décomposition fonctionnelle entre plusieurs cas. En effet, la relation d'inclusion ne permet pas de préciser une quelconque relation d'ordre ou de priorité d'appel entre les cas inclus.

*Graphique*

La relation d'inclusion entre cas d'utilisation se représente graphiquement à l'aide d'une flèche pointillée sur laquelle nous faisons apparaître la chaîne de caractères « `include` ». La flèche va du cas qui inclut vers le cas inclus.

La figure 9.5 représente une relation d'inclusion entre deux cas d'utilisation.

**Figure 9.5**
*Relation d'inclusion entre cas d'utilisation*




### extend

*Sémantique*

Il est possible de spécifier qu'un cas d'utilisation étend un autre cas d'utilisation.

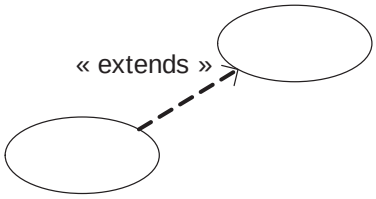
Cela signifie en quelque sorte qu'un comportement qui n'était pas spécifié est ajouté au cas étendu. La relation d'extension est souvent utilisée pour préciser des fonctionnalités optionnelles qui sont ajoutées aux fonctionnalités de base.

*Graphique*

La relation d'extension entre cas d'utilisation se représente graphiquement à l'aide d'une flèche pointillée sur laquelle nous faisons apparaître la chaîne de caractères « `extend` ». La flèche va du cas qui étend vers le cas étendu.

La figure 9.6 représente une relation d'extension entre deux cas d'utilisation.

**Figure 9.6**
*Relation d'extension entre cas d'utilisation*



#### Héritage

*Sémantique*

Il est possible de spécifier une relation d'héritage entre cas d'utilisation.

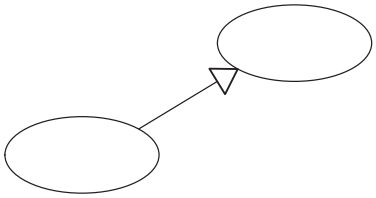
Si un cas d'utilisation C1 hérite d'un cas d'utilisation C2, C2 peut être substitué à C1 pour un acteur qui souhaite bénéficier de C1. Cette sémantique reste toutefois ambiguë, car les conditions de substitution ne sont pas spécifiées. C'est pourquoi nous déconseillons vivement l'utilisation de cette relation.

*Graphique*

Comme l'héritage entre classes, la relation d'héritage entre cas d'utilisation se représente graphiquement par une flèche allant du cas d'utilisation qui hérite vers le cas d'utilisation hérité.

La figure 9.7 représente une relation d'héritage entre deux cas d'utilisation.

**Figure 9.7**
*Relation d'héritage entre cas d'utilisation*



## Concept avancé relatif aux acteurs

Cette section présente un concept avancé applicable aux acteurs. Nous choisissons ici de ne présenter que la relation d'héritage, car cette relation est employée dans certains diagrammes.

#### Héritage

*Sémantique*

Il est possible d'exprimer une relation d'héritage entre deux acteurs.

La signification de cette relation d'héritage est sensiblement la même que celle de la relation d'héritage entre classes *(voir le chapitre 2).* Si un acteur A1 hérite d'un acteur A2, cela signifie que toutes les entités externes correspondant à A1 correspondent aussi à A2.

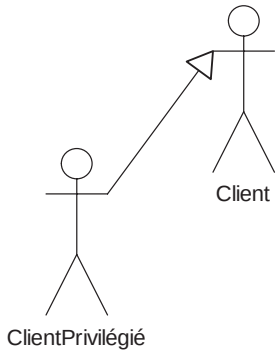
Il est important de souligner que cette relation est ensembliste (tous les A1 sont des A2). De ce fait, elle ne doit pas être employée si nous voulons exprimer qu'une entité externe peut correspondre à deux acteurs différents. Par exemple, si les acteurs « `Client` » et « `Caissier` » ont été définis et que nous voulions exprimer qu'un caissier peut être un client, il ne faut surtout pas utiliser la relation d'héritage.

*Graphique*

Comme la relation d'héritage entre classes, la relation d'héritage entre acteurs se représente par une flèche allant de l'acteur qui hérite vers l'acteur hérité.

La figure 9.8 représente une relation d'héritage entre l'acteur `ClientPrivilégié` et l'acteur `Client`.

**Figure 9.8**
*Relation d'héritage entre acteurs*




# Synthèse

Nous avons détaillé dans ce chapitre la partie fonctionnelle d'un modèle UML. Cette partie permet de représenter les différentes fonctionnalités offertes par l'application à son environnement. Il s'agit de la dernière des trois parties du modèle UML que nous voulions introduire dans ce cours.

Il est important de rappeler que si UML propose d'autres parties, nous avons choisi de ne pas les présenter afin de nous concentrer sur les parties indispensables à la mise en œuvre d'un cycle de développement avec UML tel que nous le concevons.

Nous avons ensuite présenté les liens de cohérence qui existent entre les parties fonctionnelle, comportementale et structurelle. Ces liens de cohérence ajoutés aux opérations de synchronisation avec le code permettent d'obtenir à la fois les gains des opérations de modélisation à tous les niveaux d'abstraction et les gains des opérations réalisables sur le code. C'est ce que nous cherchions à obtenir dès le début de ce cours.

Pour finir, nous avons introduit les concepts avancés de la partie fonctionnelle d'un modèle UML, en insistant bien sur le fait que ces concepts étant très délicats à employer, il ne fallait y recourir qu'en cas de réelle nécessité.

Le prochain et dernier chapitre de ce cours s'intéresse à un moyen méthodologique permettant de comprendre un peu mieux comment mettre en place un cycle de développement UML.

Travaux dirigés

# TD9. Diagrammes de cas d'utilisation

Le diagramme de cas d'utilisation de la figure 9.9 représente les fonctionnalités d'une agence de voyage classique.

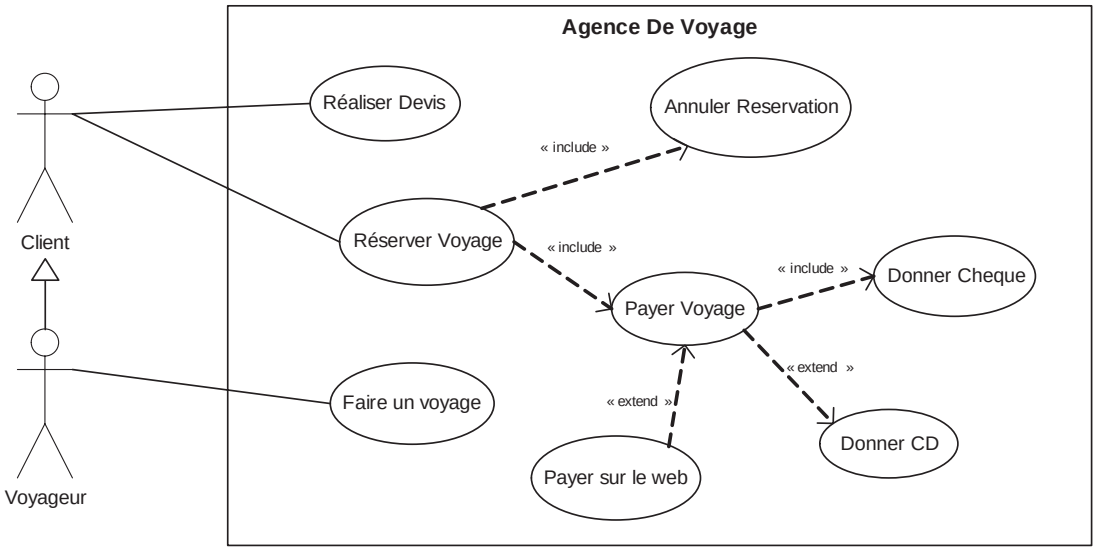



**Figure 9.9**
*Diagramme de cas d'utilisation de l'agence de voyage*

**Question 77.** *Commentez les acteurs du diagramme de cas d'utilisation.*

**Question 78.** *Commentez les cas d'utilisation du diagramme de cas d'utilisation.*

*Nous souhaitons réaliser le diagramme de cas d'utilisation du championnat d'échecs présenté au TD6.*

**Question 79.** *Donnez la liste des acteurs du système.*

**Question 80.** *Donnez la liste des cas d'utilisation du système en les liant aux acteurs.*

**Question 81.** *Donnez le diagramme de cas d'utilisation du système.*

**Question 82.** *Reprenez les diagrammes de séquence réalisés au TD6 pour l'application de championnat d'échecs, et expliquez comment les relier au diagramme de cas d'utilisation obtenu à la question précédente.*

Ce TD aura atteint son objectif pédagogique si et seulement si :

- Vous arrivez à différencier les fonctionnalités de base d'une application et son organisation fonctionnelle (différence entre les niveaux besoin et conceptuel).

- Vous savez établir un diagramme de cas d'utilisation d'une application à partir de la description textuelle de cette dernière.
- Vous savez faire le lien entre un diagramme de cas d'utilisation et les diagrammes de séquence d'une application au niveau besoin.