# Chapitre 2 : Diagramme de cas d'utilisation (*Use Case Diagram*)

## 2.1 Introduction

Bien souvent, la maîtrise d'ouvrage et les utilisateurs ne sont pas des informaticiens. Il leur faut donc un moyen simple d'exprimer leurs besoins. C'est précisément le rôle des diagrammes de cas d'utilisation qui permettent de recueillir, d'analyser et d'organiser les besoins, et de recenser les grandes fonctionnalités d'un système. Il s'agit donc de la première étape UML d'analyse d'un système.

Un diagramme de cas d'utilisation capture le comportement d'un système, d'un sous-système, d'une classe ou d'un composant tel qu'un utilisateur extérieur le voit. Il scinde la fonctionnalité du système en unités cohérentes, les cas d'utilisation, ayant un sens pour les acteurs. Les cas d'utilisation permettent d'exprimer le besoin des utilisateurs d'un système, ils sont donc une vision orientée utilisateur de ce besoin au contraire d'une vision informatique.

Il ne faut pas négliger cette première étape pour produire un logiciel conforme aux attentes des utilisateurs. Pour élaborer les cas d'utilisation, il faut se fonder sur des entretiens avec les utilisateurs.

## 2.2 Éléments des diagrammes de cas d'utilisation

### 2.2.1 Acteur

Un acteur est l'idéalisation d'un rôle joué par une personne externe, un processus ou une chose qui interagit avec un système.

Il se représente par un petit bonhomme avec son nom (*i.e.* son rôle) inscrit dessous.

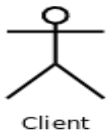


Figure : Exemple de représentation d'un acteur

Il est également possible de représenter un acteur sous la forme d'un classeur << *actor* >>


Figure : Exemple de représentation d'un acteur sous la forme d'un classeur

### 2.2.2 Cas d'utilisation

Un cas d'utilisation est une unité cohérente représentant une fonctionnalité visible de l'extérieur. Il réalise un service de bout en bout, avec un déclenchement, un déroulement et une fin, 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), et optionnellement, au-dessus du nom, un stéréotype

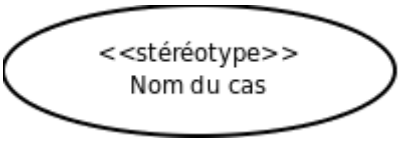


Figure : Exemple de représentation d'un cas d'utilisation

Dans le cas où l'on désire présenter les attributs ou les opérations du cas d'utilisation, il est préférable de le représenter sous la forme d'un classeur stéréotypé << *use case* >>. Nous reviendrons sur les notions d'attributs ou d'opération lorsque nous aborderons les diagrammes de classes et d'objets.

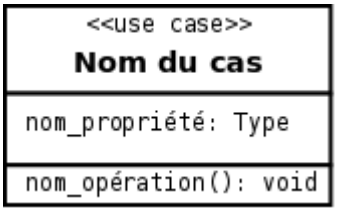


Figure : Exemple de représentation d'un cas d'utilisation sous la forme d'un classeur

### 2.2.3 Représentation d'un diagramme de cas d'utilisation

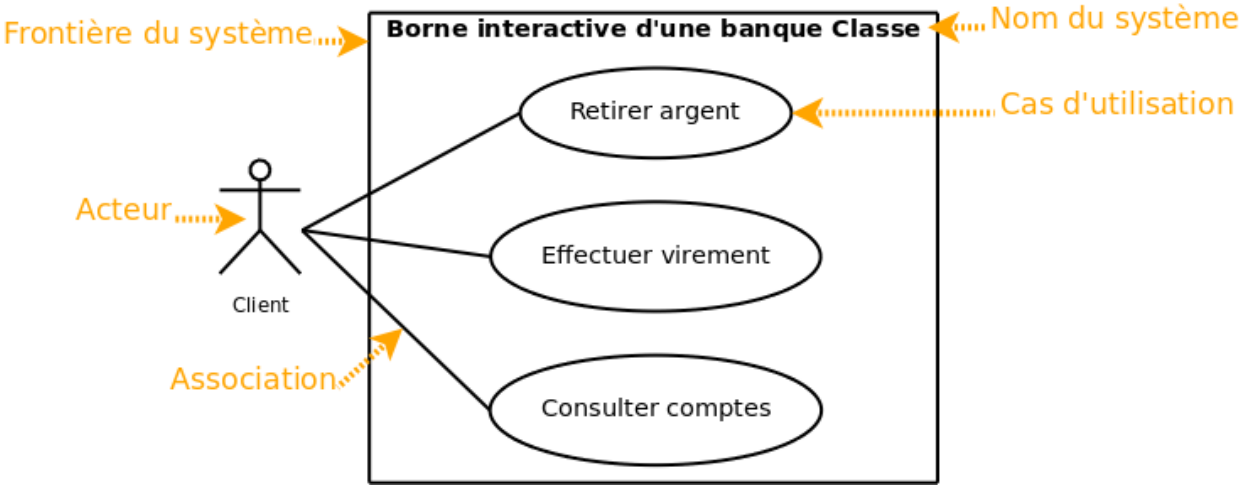


Figure : Exemple simplifié de diagramme de cas d'utilisation modélisant une borne d'accès à une banque.

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

# 2.3 Relations dans les diagrammes de cas d'utilisation

## 2.3.1 Relations entre acteurs et cas d'utilisation

### Relation d'association

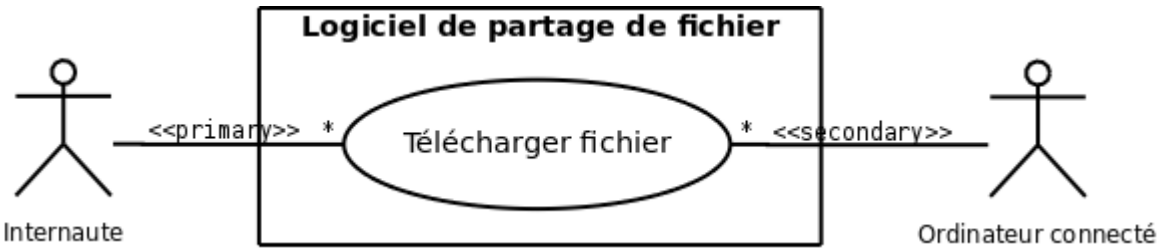


Figure : Diagramme de cas d'utilisation représentant un logiciel de partage de fichiers

Une relation d'association est chemin de communication entre un acteur et un cas d'utilisation et est représenté un trait continu.

### Multiplicité

Lorsqu'un acteur peut interagir plusieurs fois avec un cas d'utilisation, il est possible d'ajouter une multiplicité sur l'association du côté du cas d'utilisation. Le symbole * signifie *plusieurs*, exactement *n* s'écrit tout simplement *n*, *n..m* signifie entre *n* et *m*, etc. Préciser une multiplicité sur une relation n'implique pas nécessairement que les cas sont utilisés en même temps.

La notion de multiplicité n'est pas propre au diagramme de cas d'utilisation. Nous en reparlerons dans le chapitre consacré au diagramme de classes.

### 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 a 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. En général, l'acteur principal initie le cas d'utilisation par ses sollicitations. Le stéréotype << *primary* >> vient orner l'association reliant un cas d'utilisation à son acteur principal, le stéréotype << *secondary* >> est utilisé pour les acteurs secondaires.

**Cas d'utilisation interne**

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

## 2.3.2 Relations entre cas d'utilisation

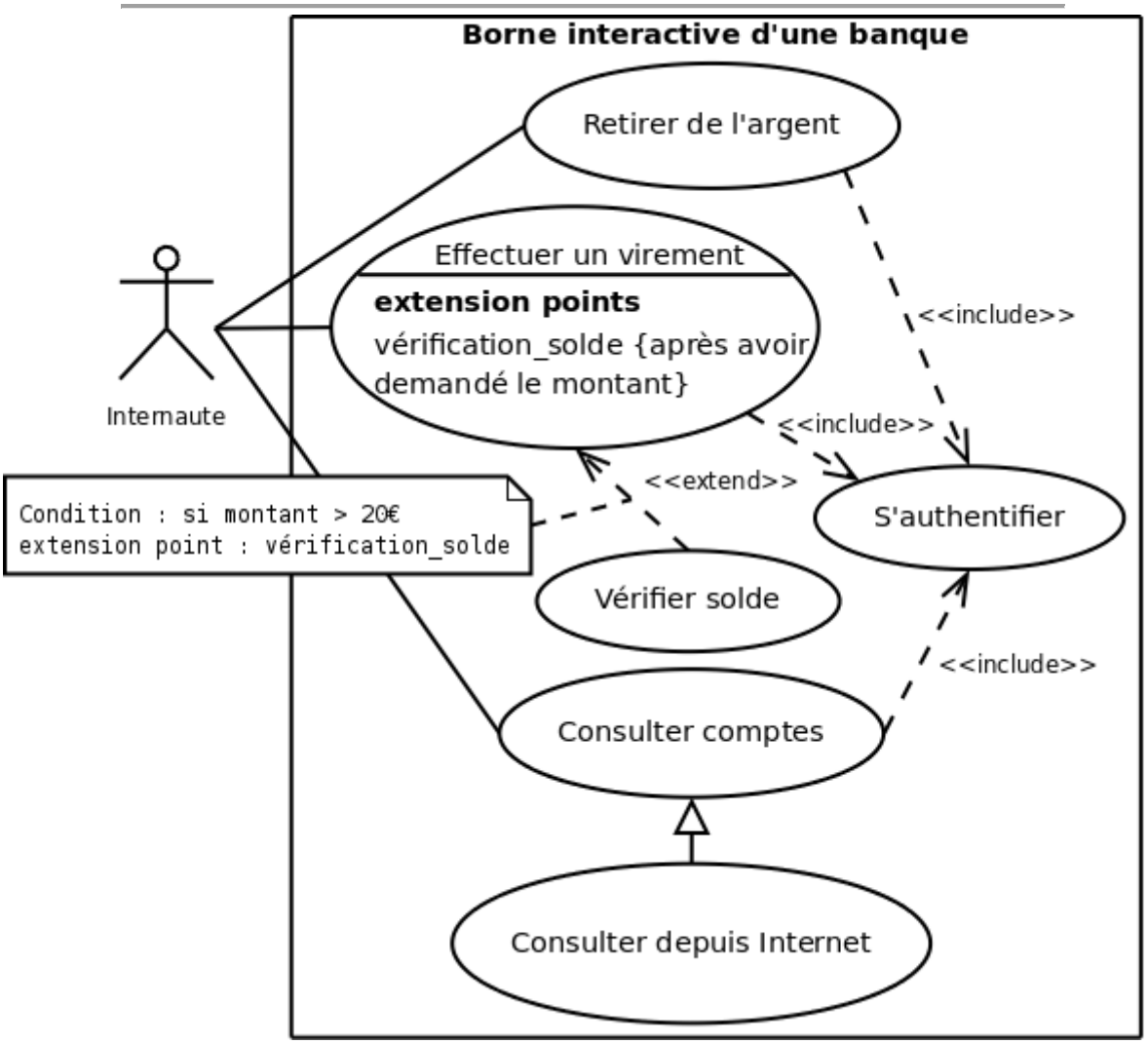


Figure : Exemple de diagramme de cas d'utilisation

**Types et représentations**

Il existe principalement deux types de relations :

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

Une dépendance se représente par une flèche avec un trait pointillé. Si le cas A inclut ou étend le cas B, la flèche est dirigée de A vers B.

Le symbole utilisé pour la généralisation est une flèche avec un trait pleins dont la pointe est un triangle fermé désignant le cas le plus général.

**Relation d'inclusion**

Un cas A inclut un cas B si le comportement décrit par le cas A inclut le comportement du cas B : le cas A dépend de B. Lorsque A est sollicité, B l'est obligatoirement, comme une partie de A. Cette dépendance est symbolisée par le stéréotype << *include* >>. Par exemple, l'accès aux informations d'un compte bancaire inclut nécessairement une phase d'authentification avec un identifiant et un mot de passe.

Les inclusions permettent essentiellement de factoriser une partie de la description d'un cas d'utilisation qui serait commune à d'autres cas d'utilisation.

Les inclusions permettent également de décomposer un cas complexe en sous-cas plus simples. Cependant, il ne faut surtout pas abuser de ce type de décomposition : il faut éviter de réaliser du découpage fonctionnel d'un cas d'utilisation en plusieurs *sous-cas d'utilisation* pour ne pas retomber dans le travers de la décomposition fonctionnelle.

Attention également au fait que, les cas d'utilisation ne s'enchaînent pas, puisqu'il n'y a aucune représentation temporelle dans un diagramme de cas d'utilisation.

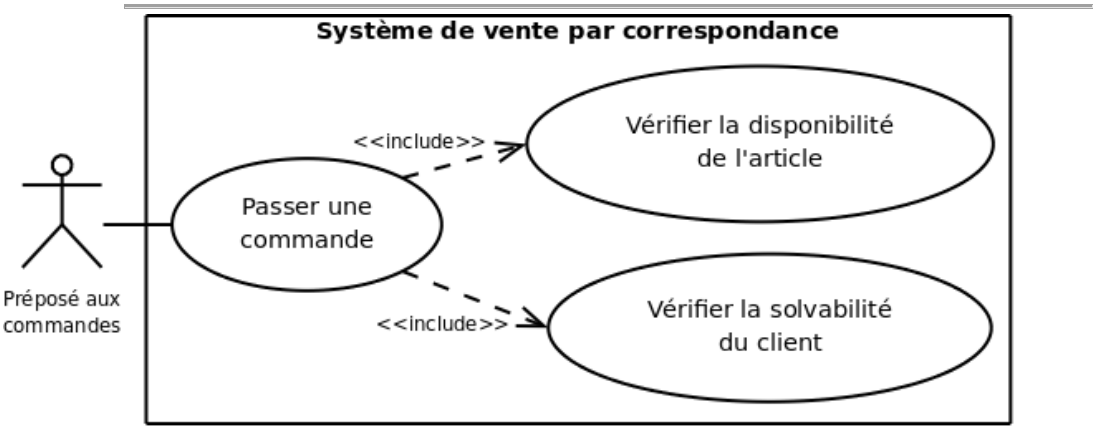


Figure : Relations entre cas pour décomposer un cas complexe

**Relation d'extension**

La relation d'extension est probablement la plus utile car elle a une sémantique qui a un sens du point de vue métier au contraire des deux autres qui sont plus des artifices d'informaticiens.

On dit qu'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. 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* >>.

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. La figure présente l’exemple d’une banque où la vérification du solde du compte n’intervient que si la demande de retrait dépasse 20 euros.

**Relation de généralisation**

Un cas A est une généralisation d’un cas B si B est un cas particulier de A. Dans la figure, la consultation d’un compte *via* Internet est un cas particulier de la consultation. 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.

## 2.3.3 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).

Par exemple, la figure montre que le directeur des ventes est un préposé aux commandes avec un pouvoir supplémentaire : en plus de pouvoir passer et suivre une commande, il peut gérer le stock. Par contre, le préposé aux commandes ne peut pas gérer le stock.

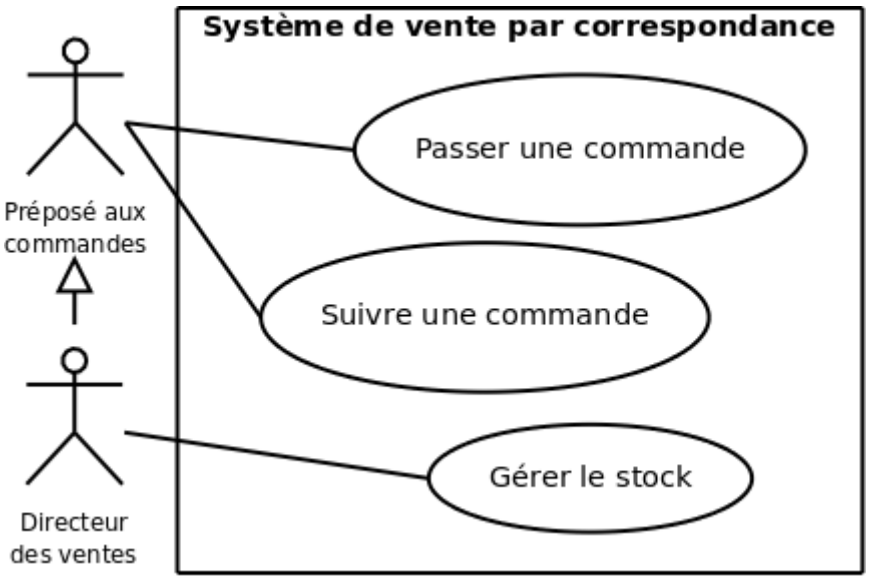


Figure : Relations entre acteurs

# 2.4 Modélisation des besoins avec UML

## 2.4.1 Comment identifier les acteurs ?

UML n'emploie pas le terme *d'utilisateur* mais d'*acteur*. Les acteurs d'un système sont les entités externes à ce système qui interagissent (saisie de données, réception d'information, …) avec lui. Les acteurs sont donc à l'extérieur du système et dialoguent avec lui. Ces acteurs permettent de cerner l'interface que le système va devoir offrir à son environnement. Oublier des acteurs ou en identifier de faux conduit donc nécessairement à se tromper sur l'interface et donc la définition du système à produire.

Il faut faire attention à ne pas confondre acteurs et utilisateurs (utilisateur avec le sens de la personne physique qui va appuyer sur un bouton) d'un système. D'une part parce que les acteurs inclus les utilisateurs humains mais aussi les autres systèmes informatiques ou hardware qui vont communiquer avec le système. D'autre part parce qu'un acteur englobe tout une classe d'utilisateur. Ainsi, plusieurs utilisateurs peuvent avoir le même rôle, et donc correspondre à un même acteur, et une même personne physique peut jouer des rôles différents vis-à-vis du système, et donc correspondre à plusieurs acteurs.

Chaque acteur doit être nommé. Ce nom doit refléter son rôle car un acteur représente un ensemble cohérent de rôles joués vis-à-vis du système.

Pour trouver les acteurs d'un système, il faut identifier quels sont les différents rôles que vont devoir jouer ses utilisateurs (ex : responsable clientèle, responsable d'agence, administrateur, approbateur, …). Il faut également s'intéresser aux autres systèmes avec lesquels le système va devoir communiquer comme :

- Les périphériques manipulés par le système (imprimantes, hardware d'un distributeur de billet, …) ;
- Des logiciels déjà disponibles à intégrer dans le projet ;
- Des systèmes informatiques externes au système mais qui interagissent avec lui, etc.

Pour faciliter la recherche des acteurs, on peut imaginer les frontières du système. Tout ce qui est à l'extérieur et qui interagit avec le système est un acteur, tout ce qui est à l'intérieur est une fonctionnalité à réaliser.

Vérifiez que les acteurs communiquent bien *directement* avec le système par émission ou réception de messages. Une erreur fréquente consiste à répertorier en tant qu'acteur des entités externes qui n'interagissent pas directement avec le système, mais uniquement par le biais d'un des véritables acteurs. Par exemple, l'hôtesse de caisse d'un magasin de grande distribution est un acteur pour la caisse enregistreuse, par contre, les clients du magasin ne correspondent pas à un acteur car ils n'interagissent pas directement avec la caisse.

## 2.4.2 Comment recenser les cas d'utilisation ?

L'ensemble des cas d'utilisation doit décrire exhaustivement les exigences fonctionnelles du système. Chaque cas d'utilisation correspond donc à une fonction métier du système, selon le point de vue d'un de ses acteurs. Aussi, pour identifier les cas d'utilisation, il faut se placer du point de vue de chaque acteur et déterminer comment et surtout pourquoi il

se sert du système. Il faut éviter les redondances et limiter le nombre de cas en se situant à un bon niveau d'abstraction. Trouver le bon niveau de détail pour les cas d'utilisation est un problème difficile qui nécessite de l'expérience.

Nommez les cas d'utilisation avec un verbe à l'infinitif suivi d'un complément en vous plaçant du point de vue de l'acteur et non pas de celui du système. Par exemple, un distributeur de billets aura probablement un cas d'utilisation *Retirer de l'argent* et non pas *Distribuer de l'argent*.

De par la nature fonctionnelle, et non objet, des cas d'utilisation, et en raison de la difficulté de trouver le bon niveau de détail, il faut être très vigilant pour ne pas retomber dans une décomposition fonctionnelle descendante hiérarchique. Un nombre trop important de cas d'utilisation est en général le symptôme de ce type d'erreur.

Dans tous les cas, il faut bien garder à l'esprit qu'il n'y a pas de notion temporelle dans un diagramme de cas d'utilisation.

### 2.4.3 Description textuelle des cas d'utilisation

Le diagramme de cas d'utilisation décrit les grandes fonctions d'un système du point de vue des acteurs, mais n'expose pas de façon détaillée le dialogue entre les acteurs et les cas d'utilisation. Bien que de nombreux diagrammes d'UML permettent de décrire un cas, il est recommandé de rédiger une description textuelle car c'est une forme souple qui convient dans bien des situations.

Une description textuelle couramment utilisée se compose de trois parties.

1. La première partie permet d'identifier le cas, elle doit contenir les informations qui suivent.

   **Nom :**
   Utiliser une tournure à l'infinitif (ex : Réceptionner un colis).
   **Objectif :**
   Une description résumée permettant de comprendre l'intention principale du cas d'utilisation. Cette partie est souvent renseignée au début du projet dans la phase de découverte des cas d'utilisation.
   **Acteurs principaux :**
   Ceux qui vont réaliser le cas d'utilisation (la relation avec le cas d'utilisation est illustrée par le trait liant le cas d'utilisation et l'acteur dans un diagramme de cas d'utilisation)

   **Acteurs secondaires :**
   Ceux qui ne font que recevoir des informations à l'issue de la réalisation du cas d'utilisation
   **Dates :**
   Les dates de créations et de mise à jour de la description courante.
   **Responsable :**
   Le nom des responsables.
   **Version :**
   Le numéro de version.

2. La deuxième partie contient la description du fonctionnement du cas sous la forme d'une séquence de messages échangés entre les acteurs et le système. Elle contient toujours une séquence nominale qui décrit de déroulement normal du cas. À la séquence nominale s'ajoutent fréquemment des séquences alternatives (des embranchements dans la séquence nominale) et des séquences d'exceptions (qui interviennent quand une erreur se produit).

   **Les préconditions :**
   Elles décrivent dans quel état doit être le système (l'application) avant que ce cas d'utilisation puisse être déclenché.
   **Des scénarii :**
   Ces scénarii sont décrits sous la forme d'échanges d'évènements entre l'acteur et le système. On distingue le scénario nominal, qui se déroule quand il n'y a pas d'erreur, des scénarii alternatifs qui sont les variantes du scénario nominal et enfin les scénarii d'exception qui décrivent les cas d'erreurs.
   **Des postconditions :**
   Elles décrivent l'état du système à l'issue des différents scénarii.

3. La troisième partie de la description d'un cas d'utilisation est une rubrique optionnelle. Elle contient généralement des spécifications non fonctionnelles (spécifications techniques, …). Elle peut éventuellement contenir une description des besoins en termes d'interface graphique.

### 2.4.4 Remarques

**Concernant les relations dans les cas d'utilisation**

Il est important de noter que l'utilisation des relations n'est pas primordiale dans la rédaction des cas d'utilisation et donc dans l'expression du besoin. Ces relations peuvent être utiles dans certains cas mais une trop forte focalisation sur leur usage conduit souvent à une perte de temps ou à un usage faussé, pour une valeur ajoutée, au final, relativement faible.

**Concernant les cas d'utilisation**

Unanimement reconnus comme cantonnés à l'ingénierie des besoins, les diagrammes de cas d'utilisation ne peuvent être qualifiés de modélisation à proprement parler. D'ailleur, de nombreux éléments descriptifs sont en langage naturel. De plus, ils ne correspondent pas *stricto sensu* à une approche objet. En effet, capturer les besoins, les découvrir, les réfuter, les consolider, etc., correspond plus à une analyse fonctionnelle classique.

**Les *Use case Realization***

UML ne mentionne que le fait que la réalisation d'un cas d'utilisation est décrite par une suite de collaborations entre éléments de modélisation mais ne parle pas de l'élément de modélisation *use case realization*. Les *use case realization* ne sont pas un formalisme d'UML mais du RUP (*Rational Unified Process*).

Après avoir rédigé les cas d'utilisation, il faut identifier des objets, des classes, des données et des traitements qui vont permettre au système de supporter ces cas d'utilisation. Pour documenter la manière dont sont mis en œuvre les cas d'utilisation du système, on peut

utiliser le mécanisme des *use case realization*. Ils permettent de regrouper un diagramme de classes et des diagrammes d'interaction. On retrouvera dans le diagramme de classes les classes qui mettent en œuvre le cas d'utilisation associé au *use case realization*. On retrouvera dans les différents diagrammes d'interaction une documentation des différents évènements échangés entre les objets afin de réaliser les différents scénarii décrit dans le cas d'utilisation.

Au final on aura un *use case realization* par cas d'utilisation et dans chaque *use case realization* on aura autant de diagrammes d'interaction que nécessaire pour illustrer les scénarii décrits dans le cas d'utilisation (scénario nominal, scénarii alternatifs et scénarii d'exception).