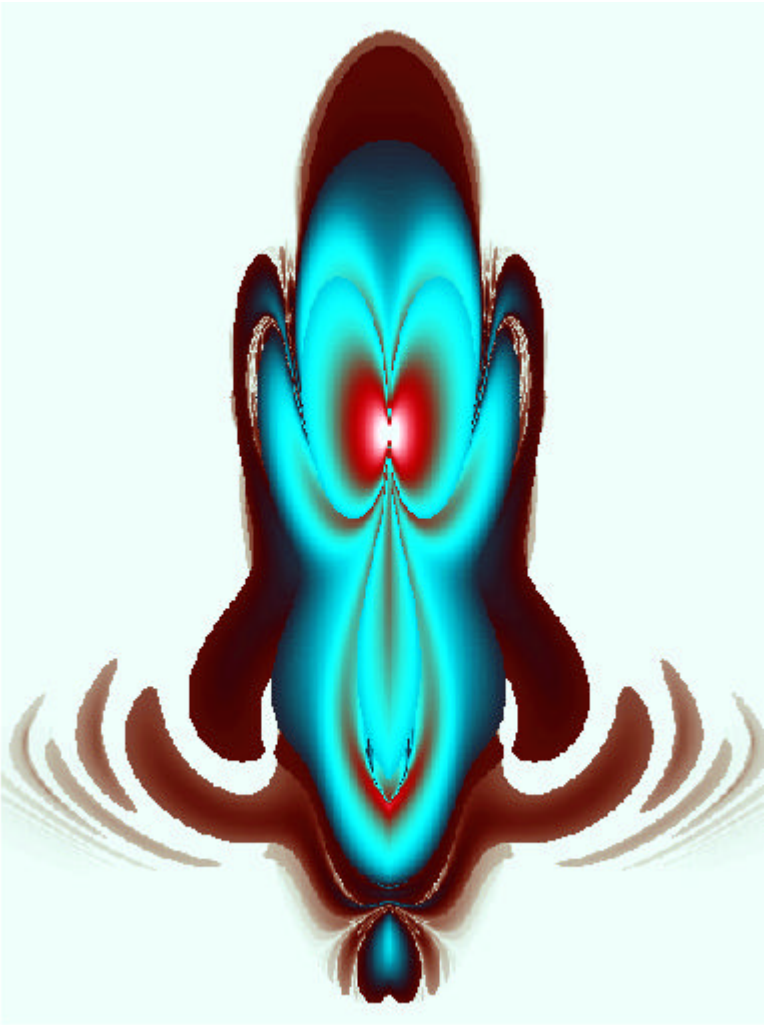
CHAPITRE 8

UML, les cas d'utilisation



8.1 *Les cas d'utilisation (Use Case)*

Les cas d'utilisation représentent un élément essentiel de la modélisation orientée objets : ils interviennent très tôt dans la conception, et doivent en principe permettre de concevoir, et de construire un système adapté aux besoins de l'utilisateur (***Build the right system***). Ils doivent également servir de fil rouge tout au long du développement, lors de la phase de conception, d'implémentation et de tests. Ils servent donc aussi bien à définir le produit à développer, à modéliser le produit, qu'à tester le produit réalisé.

8.1.1 *Anecdote pour montrer les buts des cas d'utilisation*

Au début des années 90, Ivar Jacobson (inventeur de OOSE, une des méthodes fondatrices d'UML) a été nommé chef d'un énorme projet informatique chez Ericsson. Sans méthodologie de travail particulière, ce projet devint rapidement ingérable. Personne ne savait vraiment quelles étaient les fonctionnalités du produit, ni comment elles étaient assurées, ni comment les faire évoluer. Le produit avait été défini à coups de promesses de vente, sans aucun souci de systématique quelconque.

Pour éviter de foncer droit dans un mur et mener à bien ce projet critique pour Ericsson, Jacobson a eu l'idée de redéfinir le projet en termes de besoins des utilisateurs, et d'essayer d'identifier, parmi ces besoins, ceux qui étaient réellement critiques, nécessaires à la viabilité du projet. Ces besoins critiques, une fois identifiés et structurés, devaient permettre enfin de cerner "ce qui est important pour la réussite du projet".

Le bénéfice de cette démarche simplificatrice est double. D'une part, tous les acteurs du projet ont une meilleure compréhension du système à développer, d'autre part, les besoins des utilisateurs, une fois clarifiés, serviront de fil rouge, tout au long du cycle de développement. A chaque itération de la phase d'analyse, on clarifie, affine et valide les besoins des utilisateurs ; à chaque itération de la phase de conception et de réalisation, on veille à la prise en compte des besoins des utilisateurs et à chaque itération de la phase de test, on vérifie que les besoins des utilisateurs sont satisfaits.

La détermination et la compréhension des besoins sont souvent difficiles car les intervenants sont noyés sous de trop grandes quantités d'informations. Or, comment mener à bien un projet si l'on ne sait pas où l'on va ? La démarche d'Ivar Jacobsson est un exemple de démarche centrée sur les cas d'utilisation.

Pour la petite histoire, signalons que le projet fut une réussite.

Conclusion :

Il faut clarifier et organiser les besoins des clients (les modéliser).

Jacobson identifie les caractéristiques suivantes pour les modèles :

- Un modèle est une simplification de la réalité.

- Il permet de mieux comprendre le système qu'on doit développer.
- Les meilleurs modèles sont proches de la réalité.

Les *use cases* permettent de modéliser les besoins des clients d'un système et doivent aussi posséder ces caractéristiques. Ils ne doivent pas chercher l'exhaustivité, mais clarifier, filtrer et organiser les besoins ! Une fois identifiés et structurés, ces besoins :

- définissent le contour du système à modéliser (ils précisent le but à atteindre),
- permettent d'identifier les fonctionnalités principales (critiques) du système.

Les *use cases* ne doivent donc en aucun cas décrire des solutions d'implémentation. Leur but est justement d'éviter de tomber dans la dérive d'une approche fonctionnelle, où l'on liste une litanie de fonctions que le système doit réaliser.

Bien entendu, rien n'interdit de gérer à l'aide d'outils (Doors, Requisite Pro, etc...) les exigences systèmes à un niveau plus fin et d'en assurer la traçabilité, bien au contraire. Mais un modèle conceptuel qui identifie les besoins avec un plus grand niveau d'abstraction reste indispensable. Avec des systèmes complexes, filtrer l'information, la simplifier et mieux l'organiser, c'est rendre l'information exploitable.

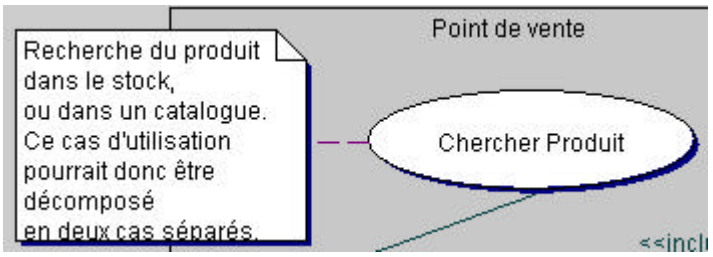
8.1.2 *Elements des cas d'utilisation*

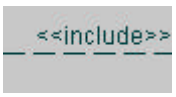
Les cas d'utilisation énumèrent toutes les interactions possibles entre de système et son environnement extérieur. A cet effet, on définit des **acteurs**, censés être initiateurs d'actions, et le cas d'utilisation lui-même. L'acteur est en principe extérieur au système, délimité par ses bornes. L'acteur a un nom, qui le définit, ou qui précise son rôle dans la transaction décrite.

Acteur 

Le **cas d'utilisation** est représenté par une ellipse, dont le contenu est un texte décrivant le cas d'utilisation. Au besoin, on pourra préciser le cas d'utilisation par une boîte de commentaires.

Une boîte de commentaires est reliée au cas d'utilisation par une ligne traitillée. Nous allons retrouver cette syntaxe tout au long des divers diagrammes de UML.

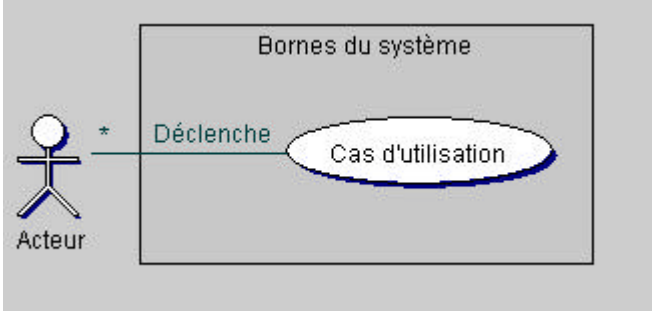




Les acteurs et les cas d'utilisation, ainsi que les cas d'utilisation entre eux, sont reliés par des liaisons indiquant leurs dépendances mutuelles. Une liaison sera générale-

ment commentée. Le commentaire est indiqué entre << et >>, et indique le type de dépendance de manière informelle (inclut, déclenche, entraîne, etc...).

FIGURE 8.1 Cas d'utilisation

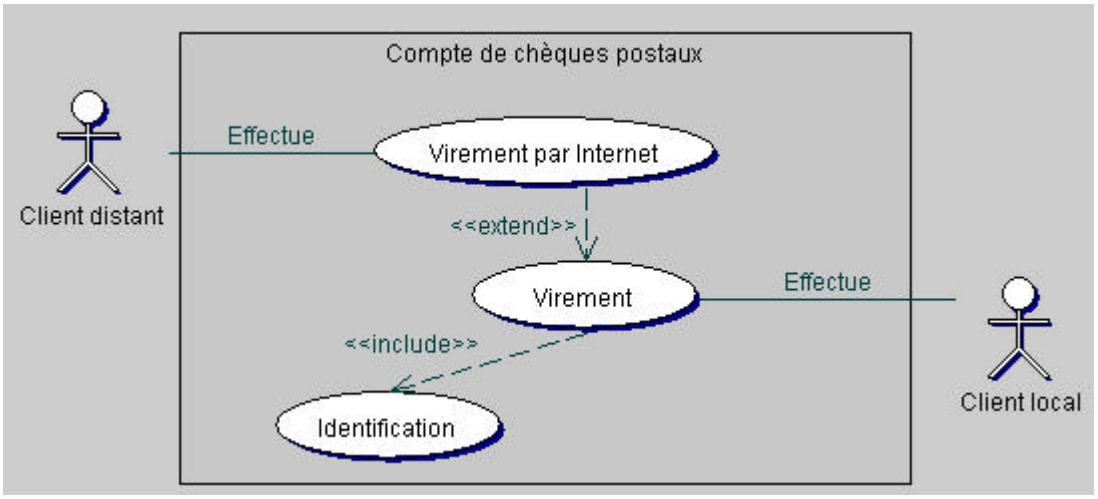


Les cas d'utilisation interviennent à tous les niveaux de la conception. Proprement énumérés, ils permettront de définir les fonctionnalités indispensables, et ainsi d'éviter l'introduction de fonctions inutiles, voire inappropriées. Ils serviront ainsi de base à la spécification, puis de guide pour la définition des sous-systèmes et des classes composant les sous-systèmes. Plus tard dans la conception, ils permettront de vérifier si les classes prévues remplissent effectivement les cas d'utilisation que l'on a défini; tard dans le projet, ils serviront de base aux tests, en indiquant quelles sont les fonctions à tester.

8.1.3 *Propriétés des cas d'utilisation*

Les cas d'utilisation ne génèrent pas de code, contrairement aux diagrammes de classes. En revanche, les cas d'utilisation peuvent avoir des relations entre eux.

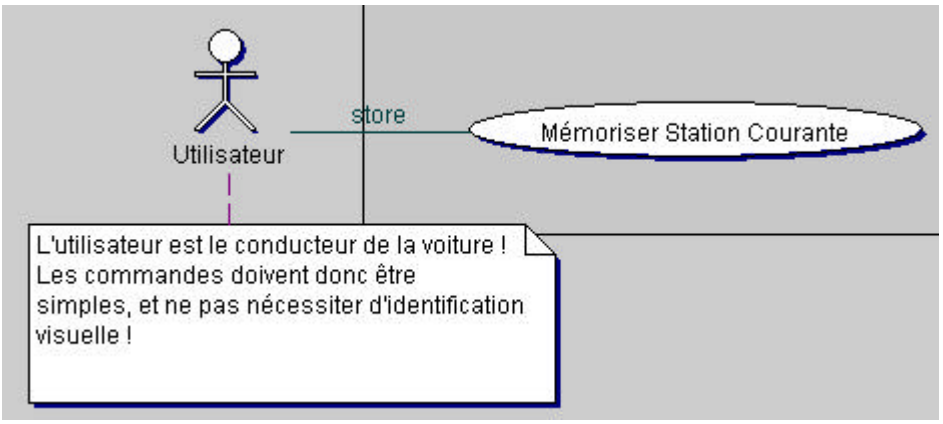
FIGURE 8.2 Relations entre cas d'utilisation



Il faut insister sur le fait que, en dépit de son caractère apparemment trivial, le diagramme des cas d'utilisation est fondamental pour tous les niveaux de la modélisation. Le bref aperçu que nous en donnons ici ne peut pas donner une idée valable de l'importance de ce diagramme : il faut avoir participé à un projet d'envergure (pour la définition duquel le temps et la place nous manquent ici) pour mesurer l'importance et la caractère fondamental de cette étape de la modélisation.

En règle générale, on essaiera de décrire des cas d'utilisation simples, ne faisant pas appel à un nombre trop élevé d'acteurs (1 est le chiffre idéal). Les cas d'utilisation seront documentés aussi précisément que possible, au besoin par des fiches de commentaire. Au mieux le cas d'utilisation sera décrit, et plus ce cas d'utilisation apportera d'aide lors de la modélisation du système. La figure8.3, page89 montre le cas d'utilisation d'un conducteur d'automobile sélectionnant un poste sur son autoradio et le mémorisant.

FIGURE 8.3 Commentaires sur un cas d'utilisation



Il faut insister sur l'importance **primordiale** des cas d'utilisation dans tout le processus de développement. Lors de la phase de spécification, ils aident à clarifier les diverses circonstances d'utilisation du produit futur, et permettent de détecter des utilisations non prévues par la spécification courante. Plus important, ils permettent d'éliminer des cas d'utilisation peu vraisemblables et donc peu utiles. Il est d'ailleurs plus que vraisemblable que l'inflation actuelle de certains programmes (traitement de texte, tableurs, etc...) serait grandement freinée par une utilisation judicieuse des cas d'utilisation : à quoi bon intégrer du code que pratiquement personne n'utilise, et qui ne fait que freiner l'application finale ?

Au lieu de cela, une bonne détection des cas d'utilisation normaux permet de définir un logiciel plus stable, mieux adapté, et de laisser les "spécialités" à des logiciels "spécialisés". On diminue ainsi le temps de développement du logiciel incriminé (moins de tests, en particulier), on augmente souvent sa fiabilité, et son coût s'en trouve baissé, ce qui permet corollairement d'augmenter les bénéfices potentiels. Pourquoi se priver ?

On a tendance, dans une modélisation objets, à faire la partie belle au diagramme de classes, sous prétecte que les outils de modélisation peuvent générer du code à partir de ce diagramme, mais pas à partir des cas d'utilisation. C'est une erreur : le diagramme de classes est souvent complémentaire, mais toujours dépendant du diagramme des cas d'utilisation. De plus, la portée des diagrammes d'utilisation va de la spécification aux tests d'intégration, alors que le diagramme de classes est utile lors de l'implémentation, essentiellement.

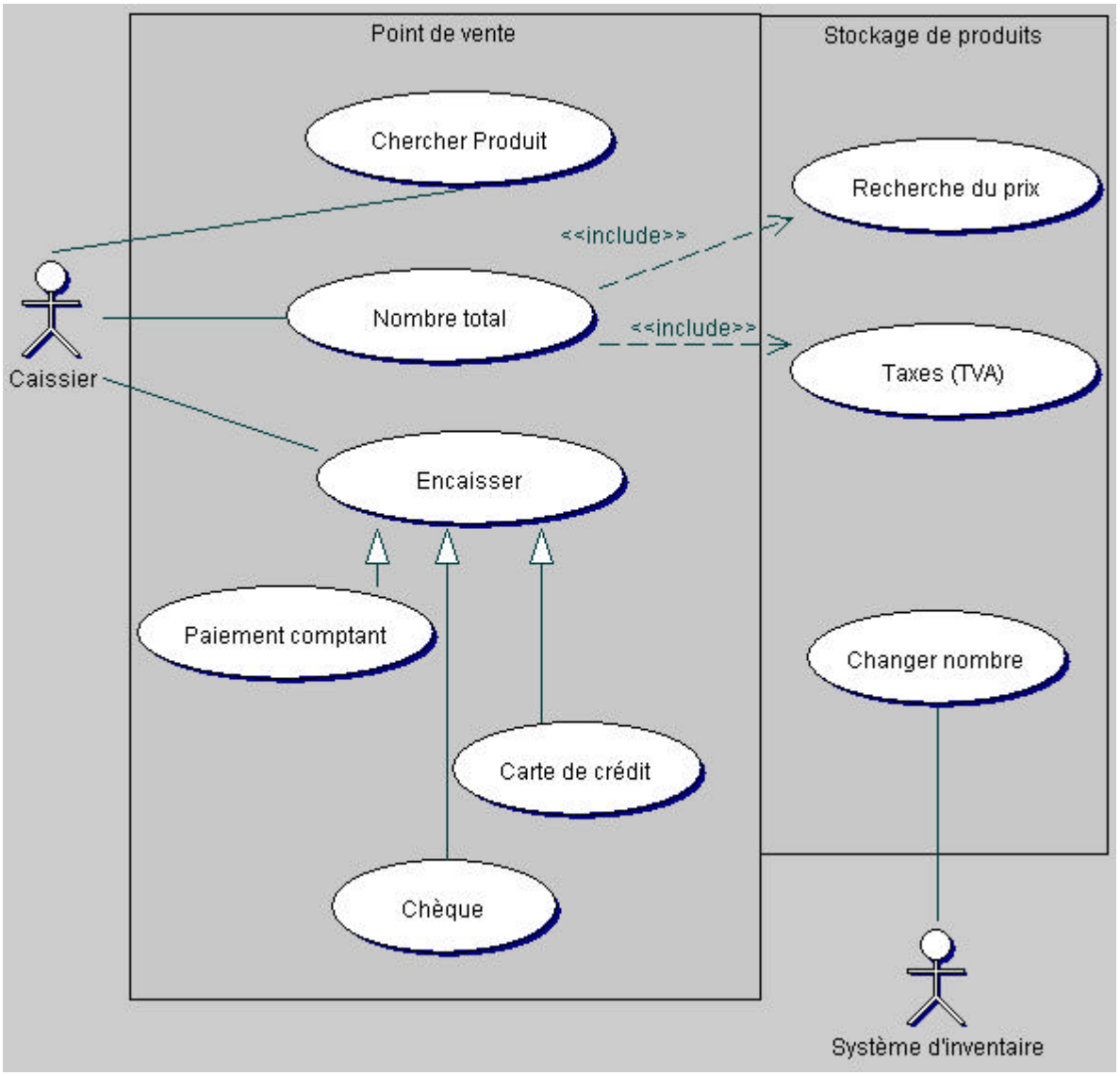
8.1.4 *Un exemple un peu plus fouillé*

La figure8.4, page91 montre un cas d'utilisation un peu plus complexe, décrivant le problème d'un caissier devant facturer le prix d'un produit à un client. Cet exemple est tiré de Peter Coad (Togethersoft).

Le schéma comprend deux acteurs : le caissier lui-même et le système d'inventaire. Ces acteurs sont tous deux externes au système. De manière similaire, il y a deux systèmes : le point de vente (*Point of Sale system*) et le système de stockage de produits (*Product System*).

Comme on peut le voir sur le schéma (figure8.4, page91), il y a deux acteurs et deux systèmes qui interagissent dans ces cas d'utilisation. On notera que plusieurs cas d'utilisations individuels pourraient être précisés : ainsi le paiement par carte de crédit peut-il être un peu différent selon le type de carte; de manière similaire, les cartes peuvent aussi être de débit, ce que l'on n'a pas documenté, bien que la procédure soit assez différente dans les deux cas. Ces précisions doivent être apportées tôt ou tard, mais il n'est pas nécessaire de les apporter immédiatement : elles peuvent l'être par la suite. Un modèle n'est jamais tout à fait complet : ceci est aussi vrai pour les cas d'utilisation.

FIGURE 8.4 Le cas d'utilisation d'une vente de produits



8.2 *Pour résumer*

Souvent, le développeur ne voit que l'utilité des diagrammes de classes, car ces derniers permettent une écriture immédiate du code associé; les autres diagrammes de UML paraissent peu utiles du fait qu'ils ne produisent pas de résultat tangible. En apparence, du moins ! Car tous ces diagrammes permettent d'acquérir une meilleure compréhension du système à construire; parmi ces diagrammes "inutiles", le plus important est le diagramme de cas d'utilisation.

Chaque fois que le développeur se trouve confronté à un dilemme, à quelque stade du développement que ce soit, il devrait penser aux diagrammes des cas d'utilisation. Plus simplement, il s'agit de se poser la question

Comment ferait l'utilisateur dans telle ou telle situation ?

Dit de cette manière, cela peut paraître trivial; trop souvent, on constate que le développeur a choisi de construire le système qui techniquement lui plaisait, quitte à introduire des fonctionnalités que même pas 1% des utilisateurs vont mettre à profit. On pense ainsi que les applications de bureautique habituelles comportent bien plus de 50% de code virtuellement "mort" (jamais exécuté). Ce code coûte cher, et complique inutilement le projet à tous les stades du développement.

Pour illustrer cette attitude regrettable, prenons un exemple indépendent du développement proprement dit, mais caractéristique d'une certaine attitude vis-à-vis d'un problème. Les théoriciens de l'informatique déplorent souvent la mauvaise qualité technique des solutions proposées; ainsi, un tel regrette l'engouement pour C/C++ alors que des langages comme Eiffel ou Ada lui paraissent techniquement plus avancés. Cette attitude méprise l'utilisateur qui a choisi C ou C++ : ce n'est pas le rôle de l'ingénieur de décider de ce qui est bon pour l'utilisateur; c'est au contraire l'utilisateur qui doit garder à tout moment le pouvoir de décider de ce qu'il trouve adapté ou non, même si sa majesté l'ingénieur n'est pas d'accord avec ce choix. Ainsi, si une partie importante d'utilisateurs a décidé que C était leur langage de prédilection, c'est qu'ils ont raison. Si une autre partie décide que c'est Visual Basic qui est le meilleur langage, ils ont également raison. Heureusement, ce genre de querelle est généralement facilement réglée par les faits; bien que la période de transition puisse être difficile.

L'utilisateur doit en tous temps rester roi, et si un système a été conçu avec ce dogme comme *leitmotiv,* alors ce système sera forcément adéquat (sous réserve de la qualité de finition du produit, bien évidemment). Un système adéquat fera forcément un utilisateur satisfait, et l'utilisateur étant souvent également un client, un client satisfait est une denrée suffisamment précieuse pour que l'on essaye par tous les moyens de la conserver en l'état. C'est le but vers lequel les cas d'utilisation cherchent à tendre. Les cas d'utilisation ne constituent pas à proprement parler une technique de programmation : il s'agit d'une attitude vis-à-vis d'un problème. Cette attitude est indépendente des outils qui permettent la réalisation : elle est applicable aussi bien pour des projets de microélectronique, de thermique, que d'informatique.

Ceci n'entraîne pas une attitude rétrograde vis-à-vis d'une solution : au contraire ! Mettre en avant les besoins de l'utilisateur conduit plus sûrement à imaginer des solutions originales qu'une attitude purement technique qui tend à réutiliser encore et encore les techniques éprouvées sous le seul prétexte que ces dernières fonctionnent.

Ce plaidoyer pour la technique des *Use Case* ne constitue pas une manifestation de fanatisme gratuit. La méthode a été éprouvée par des milliers de développeurs dans le cadre de projets regroupant des centaines d'ingénieurs pendant plusieurs années. Récemment, le téléphone portable est né d'un tel développement, et son succès est indéniable. Les *Use Case* ne sont pourtant pas un garant de succès : mais il s'agit à coup sûr d'un formidable outil pour améliorer les chances de succès d'un projet d'envergure.