Modélisation objet avec UML

Le développement des systèmes est une tâche d'une grande envergure et un investissement important pour toute entreprise. La modélisation des systèmes déjà existants ou d'un système à construire est une étape importante du cycle de développement des systèmes- La modélisation permet de visualiser, souvent d'une manière graphique, un système tel qu'il est ou comment nous souhaitons qu'il va être.
La modélisation d'une manière générale, aide à l'élaboration et à la structuration des idées et permet de faciliter la communication entre humains.

Un modèle est une abstraction de la réalité. Modéliser consiste à identifier les caractéristiques intéressantes ou pertinentes d'un système dans le but de pouvoir l'étudier du point de vue de ses caractéristiques.
Avant de construire une voiture, on conçoit des plans, des tests, des essais de moteurs… c'est la même chose avant de construire une maison, un plan est conçu, les tests par rapport à la résistance des matériaux sont réalisés etc…. Les météorologues utilisent des modèles pour prévoir la météo. On utilise les modèles dans tous les domaines scientifiques et de la réingénérie.
Un bon modèle doit posséder deux caractéristiques essentielles.

- Il doit faciliter la compréhension du phénomène (système) étudié, il réduit la complexité
- Il doit permettre de simuler le phénomène (système) étudié, il reproduit ses comportements.

1. Modèle de système informatique :

Un modèle d'un système informatique aide mieux à percevoir les relations et les interactions à l'intérieur de celui-ci. Il doit permettre de visualiser les conséquences de modifications apportées au système, il doit également permettre de visualiser les raisons du comportement du système par rapport à une situation donnée. C'est donc un guide pour construire un système fiable et stable. Le modèle doit également aider à documenter le système construit.
L'expérience du passé dans la construction de modèles suggère quatre principes de bases :

- Le choix du modèle initial a une grande influence sur la manière dont le modèle est attaqué et une solution ébauchée.
- Tout modèle peut être exprimé à divers niveaux de précision.

Sources UML 2 par la pratique Étude de cas et exercices corrigés. Pascal Roques Editions Eyrolles
Le processus unifié de développement logiciel Par : I. Jacobson, G. Booch et J. Rumbaugh, 2000

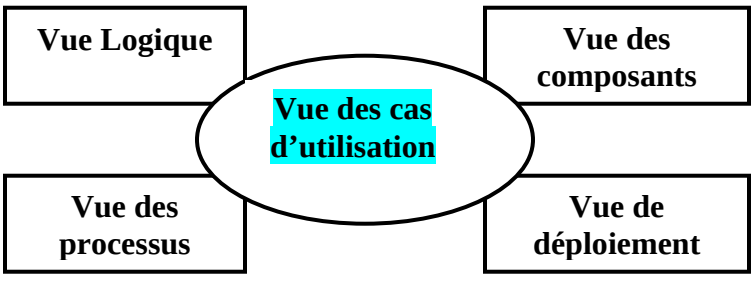
- Les meilleurs modèles sont ceux qui sont proches de la réalité.
- Aucun modèle ne peut prétendre résoudre à lui seul un problème complexe. Tout système complexe (non trivial) est approché à l'aide d'un petit nombre de modèles pratiquement indépendants.

Pourquoi UML pour la modélisation Objet?

UML opte pour **l'élaboration des modèles**, plutôt que pour une approche qui impose une barrière stricte entre analyse et conception. Les modèles d'analyse et de conception ne diffèrent que par leur niveau de détail, il n'y a pas de différence dans les concepts utilisés. UML n'introduit pas d'éléments de modélisation propres à une activité (analyse, conception...) ; le langage reste le même à tous les niveaux d'abstraction.
Cette approche simplificatrice facilite le passage entre les niveaux d'abstraction. L'élaboration encourage une approche non linéaire, les "retours en arrière" entre niveaux d'abstraction différents sont facilités et la traçabilité entre modèles de niveaux différents est assurée par l'unicité du langage.

- UML **favorise donc le prototypage**, et c'est là une de ses forces. En effet, modéliser une application n'est pas une activité linéaire. Il s'agit d'une tâche très complexe, qui nécessite une approche itérative, car il est plus efficace de construire et valider par étapes, ce qui est difficile à cerner et maîtriser.
- UML permet donc non seulement de représenter et de manipuler les concepts objet, il sous-entend une démarche d'analyse qui permet de concevoir une solution objet de manière itérative, grâce aux diagrammes, qui supportent l'abstraction.

Pour exprimer les diverses perspectives de l'architecture d'un système, les auteurs d'UML proposent **4+1 vues**, selon le schéma suivant :



Cette approche propose que plusieurs perspectives concourent à l'expression de l'architecture d'un système et explique qu'il est nécessaire de garantir la séparation et

Sources UML 2 par la pratique Étude de cas et exercices corrigés. Pascal Roques Editions Eyrolles
Le processus unifié de développement logiciel Par : I. Jacobson, G. Booch et J. Rumbaugh, 2000

l'indépendance des différentes perspectives; l'évolution de l'une des perspectives ne doit pas avoir impact (sinon limitée) sur les autres.

1. *La vue logique* : cette vue exprime la perspective abstraite de la solution en termes de classes, d'objets, de relations, de machine à états de transition etc.. de manière indépendante des langages de programmation et des environnements de développement utilisés pour les mettre en œuvre. Il s'agit d'exprimer le problème de façon abstraite. Les concepts utilisés incluent les concepts de la majorité des méthodes orientés objets existantes. Cette vue concerne *l'intégrité de conception*

2. *La vue des composants :* cette vue exprime la perspective physique de l'organisation du code en terme de modules, des composants et surtout des concepts du langage et de l'environnement d'implémentation. Cette perspective dépend du choix du langage utilisé. Dans cette perspective, le concepteur est surtout intéressé par des aspects de gestion de codes, d'ordre de compilation, de réutilisation.
   UML offre des concepts adaptés comme les modules, l'interface, les composants, les relations de dépendance ect… Malheureusement la plus part des concepteurs ignorent cette vue. Cette vue concerne *l'intégrité de gestion*

3. *La vue des processus* : cette vue exprime la perspective sur les activités concurrentes et parallèles (tâches et processus) du système. On y trouve les activités parallèles, et leur communication et synchronisation. Cette vue concerne *l'intégrité d'exécution*.

4. L*a vue de déploiement* : elle exprime la répartition du système à travers un réseau de calculateurs et de nœuds logiques de traitement. Cette vue est utile pour décrire la répartition du système réparti, elle concerne *l'intégrité de performance*

5. *La vue des cas d'utilisation* : cette vue guide et justifie les autres en ce sens que la modélisation fondée sur les scénarios (cas d'utilisation) constitue ce que l'on fait de mieux aujourd'hui. Cette approche constitue l'unique moyen de guider la modélisation, de trouver le bon modèle,

Sources UML 2 par la pratique Étude de cas et exercices corrigés. Pascal Roques Editions Eyrolles
Le processus unifié de développement logiciel Par : I. Jacobson, G. Booch et J. Rumbaugh, 2000

UML propose 13 diagrammes pour modéliser un système. Selon la vue que l'on veut décrire, statique ou dynamique, ces diagrammes sont :

Représentation statique du système :

- Le diagramme de cas d'utilisation
- Le diagramme Objets
- Le diagramme de classes
- Le diagramme des composants
- Le diagramme de déploiement
- Diagramme de packages
- Diagramme de structure composite

Représentation dynamique du système

- Le diagramme de collaboration
- Le diagramme de séquences
- Le diagramme d'état de transition
- Le diagramme d'activités
- Diagramme de communication

Sources UML 2 par la pratique Étude de cas et exercices corrigés. Pascal Roques Editions Eyrolles
Le processus unifié de développement logiciel Par : I. Jacobson, G. Booch et J. Rumbaugh, 2000

2. Le diagramme de cas d'utilisation : (Use case)

1) Définitions

Les cas d'utilisation décrivent le comportement du système du point de vue de l'utilisateur. Ils permettent de définir les limites du système et les relations entre le système et son environnement. Un cas d'utilisation est une manière spécifique d'utiliser le système. C'est l'image d'une fonctionnalité en réponse à la stimulation d'un acteur externe

2) Objectifs des cas d'utilisation

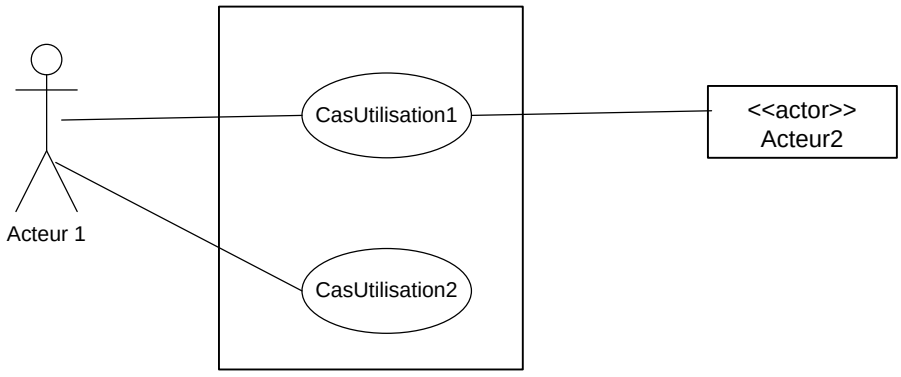
- Permettent de structurer les besoins des utilisateurs et les objectifs correspondants d'un système.
- Ils centrent l'expression des exigences du système sur ses utilisateurs Ils se limitent aux préoccupations "réelles" des utilisateurs ; ils ne présentent pas de solutions d'implémentation et ne forment pas un inventaire fonctionnel du système.
- Ils identifient les utilisateurs du système et leur interaction avec celui-ci

Un cas d'utilisation est un ensemble de séquences d'actions qui sont réalisées par le système et qui produisent un résultat observable pour un acteur particulier du système. Il permet de décrire ce que le futur système devra faire sans spécifier comment il le fera.

Acteur : entité externe qui agit sur le système. Un acteur peut consulter et /ou modifier l'état du système en émettant et/ou en recevant des messages susceptibles d'être porteurs de données.

En réponse à l'action d'un acteur, le système fournit un service qui correspond à son besoin. Un acteur peut être une personne (utilisateur humain) ou les autres systèmes connexes qui interagissent directement avec le système.

3) Représentation



Sources UML 2 par la pratique Étude de cas et exercices corrigés. Pascal Roques Editions Eyrolles
Le processus unifié de développement logiciel Par : I. Jacobson, G. Booch et J. Rumbaugh, 2000

Les noms des cas d'utilisation doit être un verbe à l'infinitif suivi d'un complément du point de vue de l'utilisateur

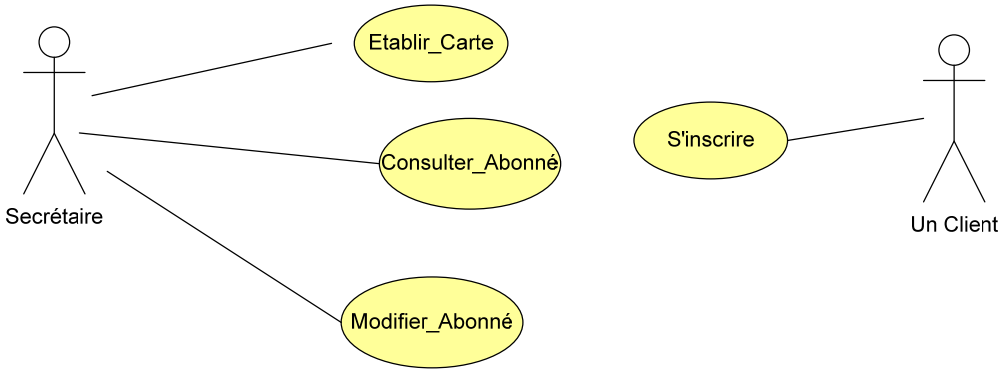
Comment identifier les acteurs?

- Les utilisateurs humains (opérateur, administrateur, l'utilisateur, le formateur …)
- Les autres systèmes connexes qui interagissent directement avec le système.

Comment identifier les cas d'utilisation?

- Chaque cas d'utilisation doit décrire les exigences fonctionnelles du système.
- Chaque cas d'utilisation correspond à une fonction métier du système (besoins des utilisateurs et possibilités du système).
- Il convient donc de rechercher pour chaque acteur
  - Les différentes intentions métier avec lesquelles il utilise le système
  - Déterminer les services fonctionnels attendus du système.

  Exemple



4) Description textuelle des cas d'utilisation

Les cas d'utilisation ont besoin d'être décrits soit textuellement, soit en utilisant un autre diagramme. Deux parties sont importantes lors de la description d'un cas d'utilisation.

1) Le sommaire d'identification
   a. Le titre
   b. Résumé
   c. Acteurs
   d. Date de création
   e. Version

Sources UML 2 par la pratique Étude de cas et exercices corrigés. Pascal Roques Editions Eyrolles
Le processus unifié de développement logiciel Par : I. Jacobson, G. Booch et J. Rumbaugh, 2000

f. Date de mise à jour
g. Responsable

2) Le scénario nominal.
   a. Préconditions
   b. Scenario nominal (enchaînement des opérations dans le cas où le cas d'utilisation se déroule normalement.)
   c. Scénario alternatif
   d. Enchaînement des erreurs
   e. postconditions

Exemple. Caisse enregistreuse

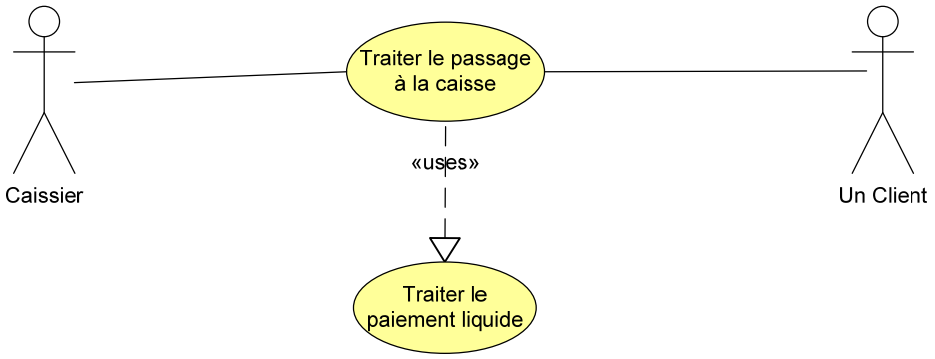
Le déroulement normal des opérations d'une caisse enregistreuse (point de vente) est le suivant :

- Un client arrive à la caisse avec des articles à payer
- Le caissier enregistre le n° d'identification de chaque article, ainsi que la quantité si elle est supérieure à 1
- La caisse affiche le prix de chaque article et son libellé.
- Lorsque tous les achats sont enregistrés, le caissier signale la fin de la vente.
- La caisse affiche le total des achats.
- Le client choisit son mode de paiement.
  1. Liquide: le caissier encaisse l'argent reçu, la caisse indique la monnaie à rendre au client.
  2. Carte de débit
  3. **Carte de crédit: un terminal bancaire fait partie de la caisse. Il transmet une demande d'autorisation en fonction du type de la carte.**
- La caisse enregistre la vente et imprime un ticket.
- Le caissier donne le ticket de caisse au client.

Lorsque le paiement est terminé, la caisse transmet les informations sur le nombre d'articles vendus au système de gestion des stocks.
Tous les matins, le responsable du magasin initialise les caisses pour la journée.

Voici le diagramme des cas d'utilisation (non complété) de la caisse enregistreuse.

Sources UML 2 par la pratique Étude de cas et exercices corrigés. Pascal Roques Editions Eyrolles
Le processus unifié de développement logiciel Par : I. Jacobson, G. Booch et J. Rumbaugh, 2000



Description des cas d'utilisation

Titre: Traiter le passage à la caisse
Brève description: Un client arrive à la caisse avec des articles qu'il souhaite acheter.
Acteurs: Caissier (principal), Le client (secondaire)

Préconditions:
- Le terminal de point de vente est ouvert
- Un caissier y est connecté
- La base de données des produits est disponible

Postcondition.
La vente est enregistrée dans le terminal de vente.

Enchaînement des opérations (scénario nominal ou déroulement normal des opérations)

Acteurs	Système
1. Le cas d'utilisation débute lorsqu'un client arrive à la caisse avec des articles à acheter.	
2. Le caissier enregistre chaque article. Le caissier indique également la quantité pour chaque article s'il y a lieu.	3. le terminal de vente valide le numéro d'identification de chaque article. Le terminal de vente affiche la description et le prix de chaque article.
4. après avoir enregistré tous les articles, le caissier indique que la vente est terminée.	5. Le terminal de vente calcule et affiche le montant de la vente.
6. le caissier annonce le prix au client.	
7. Le client choisi de payer en liquide. Exécuter le cas «paiement en espèces	
	8. Le terminal de vente enregistre la vente et imprime un ticket.
9. le caissier remet le ticket de caisse	
10. Le client s'en va	

Sources UML 2 par la pratique Étude de cas et exercices corrigés. Pascal Roques Editions Eyrolles
Le processus unifié de développement logiciel Par : I. Jacobson, G. Booch et J. Rumbaugh, 2000

Scénario alternatif

A1:Numéro d'identification du produit inconnu.

L'enchaînement de A1 démarre au point 3 du scénario nominal

3- le terminal de vente indique que le numéro du produit est inconnu. L'article ne peut être prix en vente. Le scénario reprend au point 2 s'il y a d'autres produits.

Le scénario nominal reprend au point 2, s'il y a d'autres articles ou au point 4 sinon

A2: le client demande l'annulation de l'article (article trop cher)

L'enchaînement de A2 démarre au point 2du scénario nominal

2- le caissier demande l'annulation de l'article

3- le terminal de vente supprime l'article de la vente.

Le scénario nominal reprend au point 2, s'il y a d'autres articles ou au point 4 sinon.

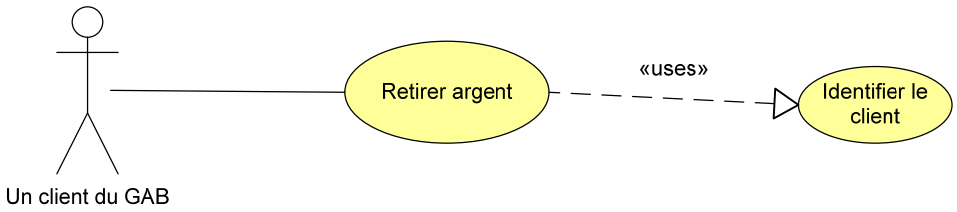
Questions : décrire le cas d'utilisation : Traiter le paiement liquide. Continuer le diagramme des cas d'utilisation de la caisse enregistreuse.

5) Relation entre les cas d'utilisation

Relation d'inclusion:«include» ou «utilise» ou «use»

Un cas d'utilisation de base (Emprunter un livre) incorpore explicitement un autre cas d'utilisation (s'identifier) de **façon obligatoire** à un endroit précis de son enchaînement. Le but est d'éviter de décrire plusieurs fois le même cas d'utilisation

Exemple:

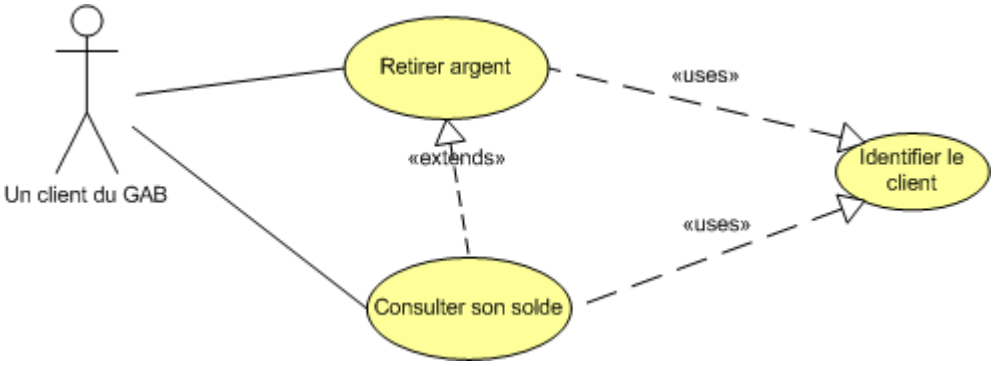


Relation d'extension: B est la suite logique de A

Un cas d'utilisation A (consulter la liste des abonnés) peut étendre son action, pas obligatoire, sur un autre cas d'utilisation B (imprimer la liste des abonnés). **Les deux cas peuvent s'exécuter de manière indépendante.**

Sources UML 2 par la pratique Étude de cas et exercices corrigés. Pascal Roques Editions Eyrolles
Le processus unifié de développement logiciel Par : I. Jacobson, G. Booch et J. Rumbaugh, 2000

Exemple:



Les cas d'utilisation *Retirer argent* et *Consulter son solde* sont deux cas indépenadants, en ce sens qu'un client du GAB peut consulter son solde sans retirer de largent ou qu'il peut retirer de l'argent sans effectuer d'interrogation de solde.
Cependant le cas *Consulter son solde* peut etendre son action au cas *Retirer de l'argent*. Un client après avoir consulter son solde peut décider de retirer de l'argent.
Quelque soit le cas d'utilisation à exécuter (*Retirer argent* ou *Consulter son solde*), ils doivent faire appel (utilise) au d'utilisation *Identifier le client* (carte débit et nip). On ne peut retirer de l'argent si nous n'avons pas de carte bancaire valide et un nip valide

Sources UML 2 par la pratique Étude de cas et exercices corrigés. Pascal Roques Editions Eyrolles
Le processus unifié de développement logiciel Par : I. Jacobson, G. Booch et J. Rumbaugh, 2000