Cours d'Environnement de Développement

Arnaud Sangnier

Partie 3

Quelques mots sur la programmation orientée objets

- L'approche objet est devenue incontournable dans le développement de logiciel
- Exemple de langages de programmation : Java, C++, C#, Ruby, etc.
- La programmation orientée objets moins intuitive que la programmation fonctionnelle
  - Plus naturel de décomposer les problèmes informatiques en termes de fonctions qu'en termes d'ensembles d'objets en interaction
- L'approche objet requiert de **modéliser** avant de concevoir
- La modélisation :
  - apporte une grande rigueur
  - offre une meilleure compréhension des logiciels
  - facilite la comparaison des solutions de conception

Qu'est-ce-qu'UML ?

- UML : Unified Modeling Langage
- UML est une notation graphique conçue pour :
  - représenter
  - spécifier
  - construire
  - documenter

  les systèmes logiciels
- Principaux objectifs d'UML :
  1. Modélisation de systèmes utilisant les techniques orientées objet
  2. Création d'un langage abstrait compréhensible par l'homme et interprétable par la machine
- UML s'adresse à toutes les personnes intervenant dans le processus de création de logiciels (analystes, développeurs, chefs de projets, architectes, etc.)

Historique d'UML

- Fin des années 80 : utilisation de plus en plus omniprésente des langages de programmation orientée objets (C++, Objective C, Eiffel, Smaltalk)
- Plusieurs équipes proposent des méthodes de modélisation (OMT, OOSE, Booch, Coad, Odel, CASE,etc.) pour "penser" objet
- Octobre 1995 (Conférence OOPSLA), Booch et Rumbaugh présentent la version 0.8 de leur méthode unifiée (Unified Method 0.8)
- 1995 : Jacobson rejoint le duo formé par Booch et Rumbaugh
- 1995 : Version 0.9 du langage UML
- 1997 : l'entreprise Rational publie la documentation de UML 1.0
- 1997 : l'OMG (Object Modelling Group) propose UML 1.1 qui devient un standard

Comment modéliser ?

- Un modèle est représentation, souvent simplifiée, d'une réalité
- Un modèle permet de capturer des aspects pertinents pour répondre à un objectif défini
- "Le modèle s'exprime sous une forme simple et pratique pour le travail" [Rumbaugh 2004]
- Quand le modèle devient compliqué, décomposition en plusieurs modèles simples et manipulables (idée principale d'UML)
- L'expression d'un modèle se fait dans un langage compatible avec le système modélisé

Pourquoi modéliser ?

Les modèles ont différents usages :

- Comprendre les mécanismes intervenant dans des systèmes complexes
- Optimiser l'organisation des systèmes
- Permettre de se focaliser sur des aspects spécifiques d'un système
- Décrire avec précision et complétude des besoins sans forcément connaître les détails du système
- Faciliter la conception d'un système
- Tester plusieurs solutions à moindre coût et sélectionner la meilleure

Plusieurs modèles

UML permet de construire plusieurs modèles d'un système

Ces modèles montrent différents aspects du système :

- Point de vue des utilisateurs
- Structure interne du système
- Vision globale
- Vision détaillée

Les modèles se complètent et peuvent être assemblés.

Bibliographie et outil

Cette partie du cours s'inspire largement de l'ouvrage suivant :

- **UML 2 - Pratique de la modélisation** de Benoît Charroux, Aomar Osmani et Yann Thierry-Mieg, Éditions PEARSON Education

Pour faire le dessin des diagrammes :

- **Umbrello** installé dans les salles de TP

Diagramme des cas d'utilisation

Use Case Diagram

Bien recueillir les besoins

Commnunication entre :

- Celui qui commande le logiciel : **Maître d'ouvrage**
- Celui qui réalise le logiciel : **Maître d'oeuvre**

Rôle du maître d'ouvrage :

1. Définir et exprimer les besoins
2. Valider les solutions proposées par le maître d'oeuvre
3. Valider le produit livré

Bien souvent le maître d'ouvrage et les utilisateurs ne sont pas informaticiens

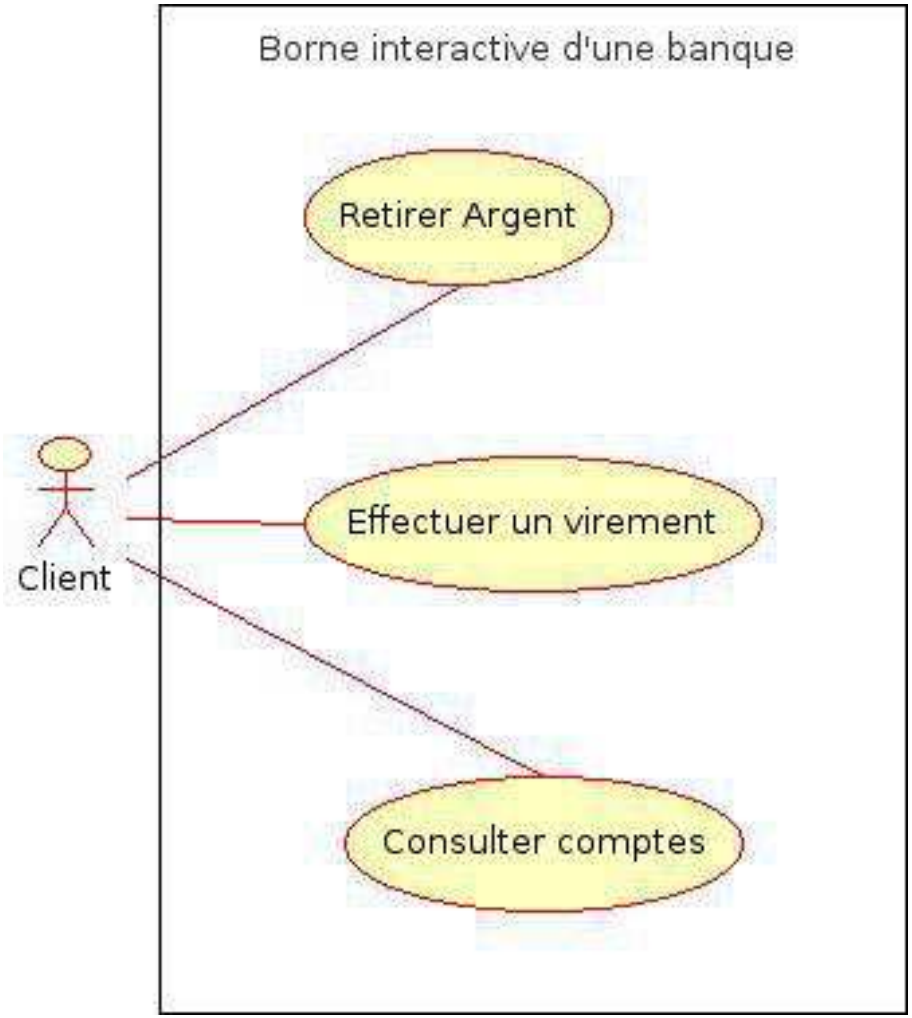
UML va être utilisé pour formaliser les besoins sous forme graphique suffisamment simple pour être compréhensible par toutes les personnes impliquées dans le projet.

Cas d'utilisation

Définition

- Un cas d'utilisation est une manière spécifique d'utiliser un système.
- Les acteurs sont à l'extérieur du système, ils modélisent tout ce qui interagit avec lui.
- Un cas d'utilisation réalise un service de bout en bout , avec un déclenchement, un déroulement et une fin, pour l'acteur qui l'initie.

Exemple


Borne interactive d'une banque
Retirer Argent
Effectuer un virement
Consulter comptes
Client

Relation entre acteurs et cas d'utilisation

Un acteur peut utiliser plusieurs fois le même cas d'utilisation

- Le symbole * qui signifie plusieurs peut être ajouté à l'extrémité de l'association liée au cas
- On peut aussi préciser n fois (en écrivant n)
- Ou encore entre m et n (en écrivant m..n)

Remarque : Préciser une multiplicité sur une relation n'implique pas nécessairement que les cas sont utilisés en même temps

Relation entre cas d'utilisation

UML permet d'établir des relations entre cas d'utilisation

Exemple de relations entre cas d'utilisation :

- **Relation d'inclusion.** Un cas A est inclus dans un cas B si le comportement décrit par le cas A est inclus dans le comportement du cas B
- **Relation d'extension.** Si le comportement de B peut être étendu par le comportement de A. Une extension est souvent soumise à condition
- **Relation de généralisation.** Un cas A est une généralisation d'un cas B si B est un cas particulier de A

A propos des différents cas d'utilisation

Définition

Un cas d'utilisation est dit "interne" s'il n'est pas relié directement à un acteur.

Remarque : Attention à l'orientation des flèches ; si le cas A inclut le cas B, on trace la fléche de A vers B, mais si B étend A la flèche est dirigée de B vers A.

Relation entre acteurs

Une seule relation entre acteurs :

- **Relation de généralisation**

Exemple : Un directeur de ventes est un préposé aux commandes avec un pouvoir supplémentaire. La flèche de généralisation pointe donc du directeur de ventes vers le préposé aux commandes (le directeur de ventes est un cas particulier de préposé aux commandes)

Qui sont les acteurs ?

- Les principaux acteurs sont les utilisateurs du système
- Chaque acteur doit être nommé (par son rôle)
- En plus des utilisateurs, les acteurs peuvent être :
  1. des périphériques manipulés par le système (imprimantes, robots, etc.)
  2. des logiciels déjà disponibles à intégrer dans le projet
  3. des systèmes informatiques externes au système mais qui interagissent avec lui

Qui sont les acteurs ?

Définition

- L'acteur est dit "principal" pour un cas d'utilisation lorsque le cas d'utilisation rend service à cet acteur
- Les autres acteurs sont dits "secondaires"
- Un cas d'utilisation a au plus un acteur principal et un ensemble d'acteurs secondaires
- Un acteur principal obtient un résultat observable du système tandis qu'un acteur secondaire est sollicité pour des informations complémentaires

Encore quelques mots sur le diagramme de cas d'utilisation

- Se placer du point de vue de chaque acteur et déterminez comment il se sert du système
- Éviter les redondances et limiter le nombre de cas en se situant au bon niveau d'abstraction
- L'ensemble des cas d'utilisation doit couvrir exhaustivement tous les besoins fonctionnelles
- Avec les diagrammes des cas d'utilisation, on se place du côté des utilisateurs
- Il n'y a pas de notion temporelle dans un diagramme de cas d'utilisation
- Les diagrammes d'utilisation pourront aussi servir pour :
  1. Établir les tests de vérification du bon fonctionnement du système
  2. Orienter les travaux de rédaction de documentation à l'usage des utilisateurs

Description d'un cas d'utilisation

Contenu de la description textuelle d'un cas d'utilisation :

1. Une première partie pour identifier le cas :
   - Le nom du cas
   - Un résumé de son objectif
   - Les acteurs impliqués (principaux et secondaires)
   - Les dates de création et de mise à jour de la description
   - Le nom des responsables
   - Un numéro de version
2. Une deuxième partie décrivant le fonctionnement :
   - Pré-conditions
   - Enchaînement des messages
   - Post-conditions
3. Une rubrique optionnelle avec par exemple :
   - Spécifications non fonctionelles
   - Contraintes liées aux IHM

Exemple de description d'un cas d'utilisation

Partie Identification

- **Identification**
  - Nom du cas : Retrait d'espèces en euros
  - But : détaille les étapes permettant à un guichetier d'effectuer l'opération de retrait d'euros demandé par un client
  - Acteur principal : Guichetier
  - Acteur secondaire : Système central
  - Date : le 19/03/2011
  - Responsable : M. Dupont
  - Version : 1.0

Exemple de description d'un cas d'utilisation

Partie Fonctionnement

- **Séquencement**
  - Le cas d'utilisation commence lorsqu'un client demande le retrait d'espèce en euros
  - Pré-conditions : Le client possède un compte (donne son numéro de compte)
  - Enchaînement nominal :
    1. Le guichetier saisit le numéro de compte du client
    2. L'application valide le compte auprès du système central
    3. L'application demande le type d'opérations au guichetier
    4. Le guichetier sélectionne un retrait d'espèces de 200 euros
    5. L'application demande au système central de débiter le compte
    6. Le système notifie au guichetier qu'il peut délivrer le montant demandé
  - Post-conditions : Le guichetier ferme le compte, le client récupère l'argent

Exemple de description d'un cas d'utilisation

Partie Optionnelle

- **Rubriques optionnelles**
  - Contraintes non fonctionnelles :
    - Fiabilité : les accès doivent être extrêmement sûrs et sécurisés
    - Confidentialité : les informations concernant le client ne doivent pas être divulguées
  - Contraintes liées à l'IHM
    - Donner la possibilité d'accéder aux autres comptes du client
    - Toujours demander la validation des opérations de retrait

Diagramme de classes

Class Diagram

Présentation générale du diagramme de classes

- Diagramme le plus important de la modélisation orientée objet
- Montre la structure interne du système
- Contient des classes avec des attributs et des opérations
- Description purement statique d'un système

Classe

Définition

Une classe est une description d'un ensemble d'objets ayant une sémantique, des attributs, des méthodes et des relations en commun. Un objet est une instance d'une classe.

Encapsulation

Protéger le coeur d'un système des accès venant de l'extérieur

Niveaux d'encapsulation proposées par UML :

1. *public* (noté +) : propriété ou classe visible partout
2. *protected* (noté #) : propriété ou classe visible dans la classe et par tous ses descendants
3. *private* (noté -) : propriété ou classe visible uniquement dans la classe
4. Aucun mot clef : propriété ou classe visible uniquement dans le paquetage où la classe est définie

Relations entre classes

- Une application nécessite la modélisation de plusieurs classes
- Il faut relier les classes entre elles
- Les relations entre classes expriment des liens sémantiques ou structurelles
- Les relations les plus utilisées sont :
  - l'association
  - l'agrégation
  - la composition
  - la dépendance
  - l'héritage

Propriétés de l'héritage

- La classe enfant possède toutes les de la classe parent, mais elle n'a pas accès aux propriétés privées de celle-ci
- Une classe enfant peut redéfinir (même signature) des méthodes de la classe parent
- 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 (principe de substitution)
- Une classe peut avoir plusieurs parents (héritage multiple), possible en C++ mais pas en Java - **fortement déconseillé**

Bien construire un diagramme de classes

- Éliminez les classes redondantes
- Retardez l'ajout de classes faisant intervenir les choix de conception et d'implémentation
- N'ajoutez pas les tableaux ou les listes d'objets dans les classes
- Quand une classe dispose de beaucoup de responsabilités, trouvez un moyen de la simplifier
- Evitez de mentionner les opérations d'accès aux attributs (getter et setter)