Le 30 / 10 / 2012.

UML (Unified Modeling Language).
(Résumé de Cours)
Licence 3 ACAD (2012/2013)

1. Introduction

Le génie logiciel et la méthodologie s'efforcent de couvrir tous les aspects de la vie du logiciel. Issus de l'expérience des développeurs, concepteurs et chefs de projets, ils sont en constante évolution, parallèlement à l'évolution des techniques informatiques et du savoir-faire des équipes. Comme toutes les tentatives de mise à plat d'une expérience et d'un savoir-faire, les méthodologies ont parfois souffert d'une formalisation excessive, imposant aux développeurs des contraintes parfois contre-productives sur leur façon de travailler. Avec la mise en commun de l'expérience et la maturation des savoir-faire, on voit se développer à présent des méthodes de travail à la fois plus proches de la pratique réelle des experts et moins contraignantes.

UML (Unified Modeling Language), qui se veut un instrument de capitalisation des savoir-faire puisqu'il propose un langage qui soit commun à tous les experts du logiciel, va dans le sens de cet assouplissement des contraintes méthodologiques.

2. Historique des méthodes de conception

À chacune des différentes phases de la conception d'un logiciel correspondent des problèmes ou des contraintes différentes. Naturellement, ces niveaux ont fait l'objet de recherches méthodologiques considérables depuis les années 80. Il en résulte que de nombreuses méthodes de développement ou d'analyse de logiciel ont vu le jour, chacune plus ou moins spécialisée ou adaptée à une démarche particulière, voire à un secteur industriel particulier (bases de données, matériel embarqué, ...). Celles-ci ayant été développées indépendamment les unes des autres, elles sont souvent partiellement redondantes ou incompatibles entre elles lorsqu'elles font appel à des notations ou des terminologies différentes, voire à des faux amis. De plus, à chaque méthode correspond un ou plusieurs moyens (plus ou moins formel) de représentation des résultats. Celui-ci peut être graphique (diagramme synoptique, plan physique d'un réseau, organigramme) ou textuel (expression d'un besoin en langage naturel, jusqu'au listing du code source). Dans les années 90, un certain nombre de méthodes orientées objets ont émergé, en particulier les méthodes :

- OMT de James RUMBAUGH,
- BOOCH de Grady BOOCH,
- OOSE (Object Oriented Software Engineering) de Ivar JACOBSON.
-

En 1994, on recensait plus de 50 méthodologies orientées objets. C'est dans le but de remédier à cette dispersion que les « poids-lourds » de la méthodologie orientée objets ont entrepris de se regrouper autour d'un standard. En octobre 1994, Grady Booch et James Rumbaugh se sont réunis au sein de la société RATIONAL dans le but de travailler à l'élaboration d'une méthode commune qui intègre les avantages de l'ensemble des méthodes reconnues, en corrigeant les défauts et en comblant les déficits. Lors de OOPSLA'95 (Object Oriented Programming Systems, Languages and Applications, la grande conférence de la programmation orientée objets), ils présentent UNIFIED METHOD V0.8. En 1996, Ivar Jacobson les rejoint. Leurs travaux ne visent plus à constituer une méthodologie, mais un langage. Leur initiative a été soutenue par de nombreuses sociétés, que ce soit des sociétés de développement (dont Microsoft, Oracle, Hewlet-Packard, IBM – qui a apporté son langage de contraintes OCL –, ...) ou des sociétés de conception d'ateliers logiciels. Un projet a été déposé en janvier 1997 à l'OMG en vue de la normalisation d'un langage de modélisation. Après amendement, celui-ci a été accepté en novembre 97 par l'OMG sous la référence UML-1.1. La version UML-2.0 est annoncée pour la fin 2004. UML est donc non seulement un outil intéressant mais une norme qui s'impose en technologie à objets et à laquelle se sont rangés tous les grands acteurs du domaine, acteurs qui ont d'ailleurs contribué à son élaboration.

UML (Unified Modeling Language, que l'on peut traduire par "langage de modélisation unifié) est une notation permettant de modéliser un problème de façon standard. Ce langage est né de la fusion de plusieurs méthodes existant auparavant, et est devenu désormais la référence en

terme de modélisation objet, à un tel point que sa connaissance est souvent nécessaire pour obtenir un poste de développeur objet.

UML est l'Unified Modeling Language standardisé par l'OMG (Object Management Group : http://www.omg.org). Ce n'est pas une méthode, il ne donne pas de solution pour la mise en œuvre d'un projet. C'est avant tout un **formalisme graphique** issu de notations employées dans différentes méthodes objets. **UML** sert à :

- Décomposer le processus de développement,
- Mettre en relation les experts métiers et les analystes,
- Coordonner les équipes d'analyse et de conception,
- Séparer l'analyse de la réalisation,
- Prendre en compte l'évolution de l'analyse et du développement,
- Migrer facilement vers une architecture objet d'un point de vue statique et dynamique.

3. UML en œuvre

UML n'est pas une méthode (*i.e.* une description normative des étapes de la modélisation) : ses auteurs ont en effet estimé qu'il n'était pas opportun de définir une méthode en raison de la diversité des cas particuliers. Ils ont préféré se borner à définir un langage graphique qui permet de représenter et de communiquer les divers aspects d'un système d'information. Aux graphiques sont bien sûr associés des textes qui expliquent leur contenu. UML est donc un métalangage car il fournit les éléments permettant de construire le modèle qui, lui, sera le langage du projet.

Il est impossible de donner une représentation graphique complète d'un logiciel, ou de tout autre système complexe, de même qu'il est impossible de représenter entièrement une statue (à trois dimensions) par des photographies (à deux dimensions). Mais il est possible de donner sur un tel système des *vues* partielles, analogues chacune à une photographie d'une statue, et dont la conjonction donnera une idée utilisable en pratique sans risque d'erreur grave.

4. Diagrammes UML

UML 2.0 comporte ainsi treize types de diagrammes représentant autant de *vues* distinctes pour représenter des concepts particuliers du système d'information. Ils se répartissent en deux grands groupes :

Diagrammes structurels ou diagrammes statiques (*UML Structure*)
- Diagramme de classes (*Class diagram*)
- Diagramme d'objets (*Object diagram*)
- Diagramme de composants (*Component diagram*)
- Diagramme de déploiement (*Deployment diagram*)
- Diagramme de paquetages (*Package diagram*)
- Diagramme de structures composites (*Composite structure diagram*)

Diagrammes comportementaux ou diagrammes dynamiques (*UML Behavior*)
- Diagramme de cas d'utilisation (*Use case diagram*)
- Diagramme d'activités (*Activity diagram*)
- Diagramme d'états-transitions (*State machine diagram*)
- **Diagrammes d'interaction (*Interaction diagram*)**
  - Diagramme de séquence (Sequence diagram)
  - Diagramme de communication (Communication diagram)
  - Diagramme global d'interaction (Interaction overview diagram)
  - Diagramme de temps (Timing diagram)

Ces diagrammes, d'une utilité variable selon les cas, ne sont pas nécessairement tous produits à l'occasion d'une modélisation. Les plus utiles pour la maîtrise d'ouvrage sont les diagrammes d'activités, de cas d'utilisation, de classes, d'objets, de séquence et d'états-transitions. Les diagrammes de composants, de déploiement et de communication sont surtout utiles pour la maîtrise d'œuvre à qui ils permettent de formaliser les contraintes de la réalisation et la solution technique.
Voici la description des quelques diagrammes :

Diagramme de cas d'utilisation (*Use Case Diagram*)

1. Introduction

Le rôle des diagrammes de cas d'utilisation 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.

2. Objectifs

1. Définir les besoins fonctionnels du système. Les cas d'utilisation ont pour principal objectif la capture des fonctionnalités couvertes par le système.
2. Définir le périmètre fonctionnel du système. Les cas d'utilisation permettent de définir les frontières du système avec son environnement.
3. Définir le dialogue entre l'utilisateur et le système. Les cas d'utilisation recensent comment l'utilisateur interagit avec le système.
4. Etablir les scénarios fonctionnels qui seront utilisés pour la recette du système. Les cas d'utilisation recensent et décrivent les principales fonctionnalités attendues du système.
5. Servir de support de référence tout au long des phases de développement du système. Les cas d'utilisation seront consultés et référencés tout au long du processus de développement du système

3. Comment déterminer les cas d'utilisation

Se poser les questions suivantes :

1. Quelles sont les grandes fonctionnalités attendues du système ?
2. Le système doit-il informer une personne ou un dispositif extérieur lorsque son état interne est modifié ?
3. Le système doit-il être informé d'événements extérieurs se produisant dans son entourage ?
4. Le système stocke-t-il des informations ? Comment sont-elles stockées, mises à jour, détruites ?

4. Éléments des diagrammes de cas d'utilisation

Pour élaborer les cas d'utilisation, il faut se fonder sur des entretiens avec les utilisateurs.

4.1. 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.

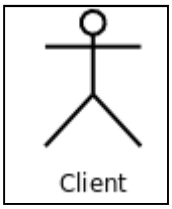


Figure1. Exemple de représentation d'un acteur.

- L'acteur est à l'origine des événements initiateurs reçus par le système.
- L'acteur dialogue par la suite avec le cas d'utilisation dont il est l'initiateur.
- L'acteur possède un nom : celui du rôle qu'il joue lors de son interaction avec le système.
- L'acteur n'est pas forcément humain. Il peut s'agir :
  - D'un autre système.
  - D'un équipement.

Exemple :

- Mouhamed utilise le système pour gérer son agenda.
- Amine utilise aussi le système pour gérer son agenda. Mais Amine est aussi autorisé à administrer le système.
- Mouhamed n'est pas un acteur du système, Amine n'est pas un acteur du système.
- Le rôle « Utilisateur » est un acteur du système.
- Le rôle « Administrateur » est un acteur du système.

Ne pas confondre personne physique et rôle. Une personne peut très bien assumer plusieurs rôles et réciproquement.

Un acteur est représenté par un petit personnage

- Le nom de l'acteur apparaît sous le petit personnage.
- On peut définir des catégories d'acteurs plus générales ou au contraire spécialiser un type d'acteur.

Comment déterminer les acteurs

Se poser les questions suivantes : Qui installe le système ?, Qui utilise le système ?, Qui démarre le système ?, Qui maintient le système?, Quels sont les autres systèmes qui utilisent le système ?, Qui fournit de l'information au système ?, Qui récupère de l'information à partir du système ?...

4.2. 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.

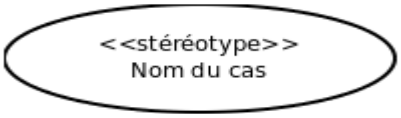


Figure2. Exemple de représentation d'un cas d'utilisation.

Le diagramme de cas d'utilisation est une représentation contextuelle de haut niveau du système modélisé.

Permet de définir de manière précise les frontières du système à modéliser :

- Montre les interactions entre le système et son environnement extérieur
- Montre les dépendances existant entre les cas d'utilisation

Le diagramme de cas d'utilisation met en scène :

- Les acteurs.
- Les cas d'utilisation.
- Les interactions entre acteurs et cas d'utilisation.
- Les dépendances entre cas d'utilisation.

4.3. Interaction entre acteur et cas d'utilisation :

- Elle est représentée par une association sous la forme d'un lien éventuellement orienté dans le sens de l'interaction.
- Une seule association est utilisée pour représenter l'ensemble des événements échangés.
- L'association peut comporter des cardinalités.

Représentation d'un diagramme de cas d'utilisation

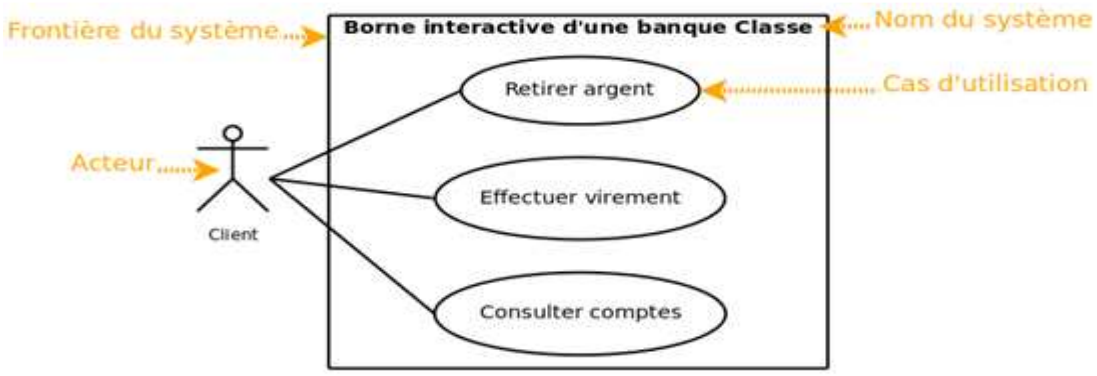


Figure3. Exemple simplifié de diagramme de cas d'utilisation modélisant une borne d'accès à une banque.

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.

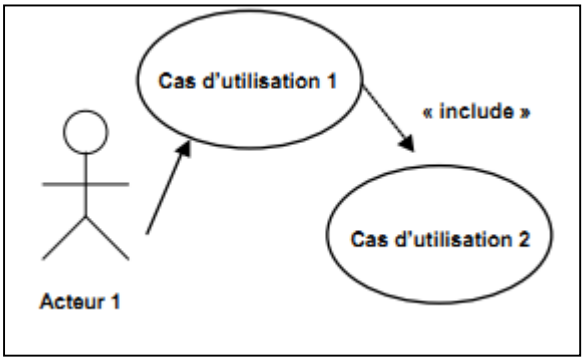
5. Dépendances entre cas d'utilisation

Il existe 3 types de dépendances entre use cases :

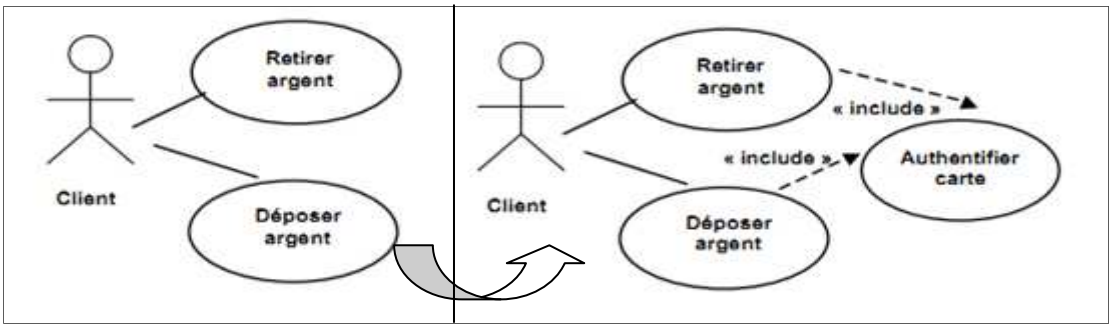
5.1. Les dépendances d'utilisation :

Mise en facteur de séquences d'événement communes.

- Indique qu'un cas d'utilisation utilise systématiquement et intégralement une séquence d'activités décrite dans un autre cas d'utilisation.
- Est représentée par une flèche pointillée étiquetée « ***include*** », pointant vers le cas d'utilisation utilisé.



- Permet de décomposer un cas d'utilisation complexe en cas d'utilisation plus simples.
- Permet de factoriser des comportements utiles à plusieurs cas d'utilisation.

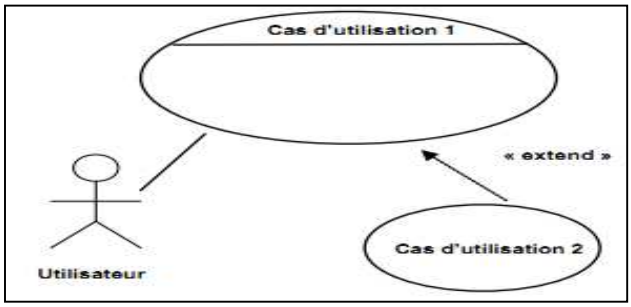


Un cas A inclut un cas B si le comportement décrit par le cas A inclut le comportement du cas B : le cas A dépend de B. Lorsque A est sollicité, B l'est obligatoirement, comme une partie de A.

5.2. Les dépendances d'extension

Externalisation de séquences d'événement exceptionnelles.

- Indique qu'un cas d'utilisation utilise facultativement ou sous certaines conditions un séquence d'activités décrite dans un autre cas d'utilisation.
- Est représentée par une flèche pointillée étiquetée « ***extend*** », pointant vers le cas d'utilisation étendu.

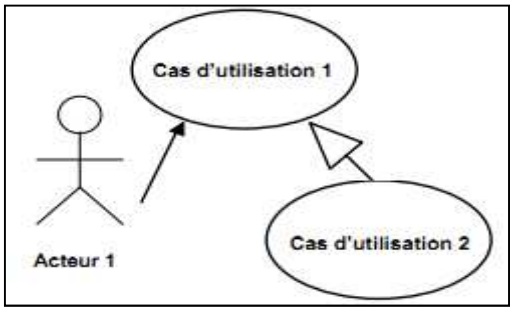


On dit qu'un cas d'utilisation A étend un cas d'utilisation B lorsque le cas d'utilisation A peut être appelé au cours de l'exécution du cas d'utilisation B. Exécuter B peut éventuellement entraîner l'exécution de A.

5.3. Les dépendances de généralisation

Généralisation / spécialisation de cas d'utilisation.

- Indique qu'un cas d'utilisation est une spécialisation d'un autre cas d'utilisation
- Est représentée par une flèche « d'héritage » pointant du cas d'utilisation spécialisé vers le cas d'utilisation le plus général.



- Permet de factoriser un comportement commun à un ensemble de cas d'utilisation proches
- Le cas d'utilisation le plus général est dit abstrait si seuls les cas d'utilisation spécialisés sont exécutables.