Chapitre 2

Modélisation avec UML

1. Introduction

1.1 Rappels : Concepts importants de l'approche objet

L'approche objet rapproche les données et leurs traitements. Mais cette approche ne fait pas que ça, d'autres concepts importants sont spécifiques à cette approche et participent à la qualité du logiciel.

Notion de classe

Tout d'abord, introduisons la notion de classe. Une classe est un type de données abstrait qui précise des caractéristiques (attributs et méthodes) communes à toute une famille d'objets et qui permet de créer (instancier) des objets possédant ces caractéristiques. Les autres concepts importants qu'il nous faut maintenant introduire sont l'encapsulation, l'héritage et l'agrégation.

Encapsulation

L'encapsulation consiste à masquer les détails d'implémentation d'un objet, en définissant une interface. L'interface est la vue externe d'un objet, elle définit les services accessibles (offerts) aux utilisateurs de l'objet.

L'encapsulation facilite l'évolution d'une application car elle stabilise l'utilisation des objets : on peut modifier l'implémentation des attributs d'un objet sans modifier son interface, et donc la façon dont l'objet est utilisé.

L'encapsulation garantit l'intégrité des données, car elle permet d'interdire, ou de restreindre, l'accès direct aux attributs des objets.

Héritage, Spécialisation, Généralisation et Polymorphisme

L'héritage est un mécanisme de transmission des caractéristiques d'une classe (ses attributs et méthodes) vers une sous-classe. Une classe peut être spécialisée en d'autres classes, afin d'y ajouter des caractéristiques spécifiques ou d'en adapter certaines. Plusieurs classes peuvent être généralisées en une classe qui les factorise, afin de regrouper les caractéristiques communes d'un ensemble de classes.

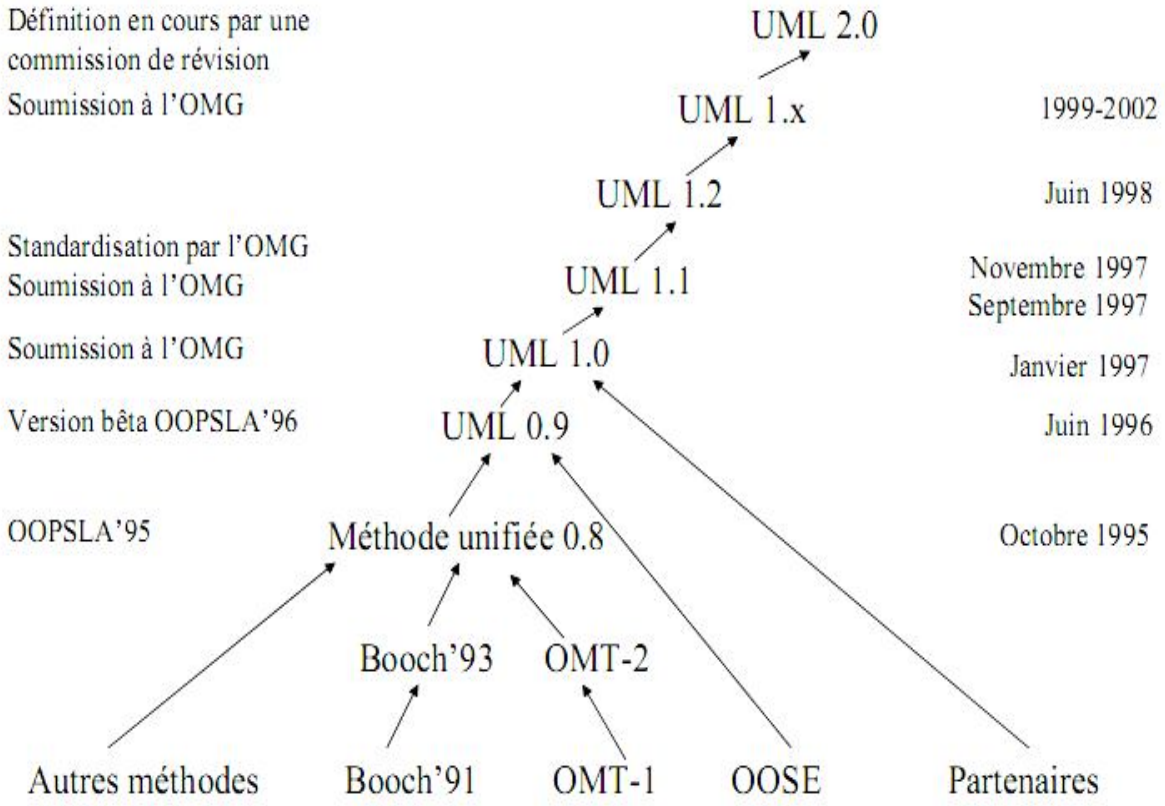
Ainsi, la spécialisation et la généralisation permettent de construire des hiérarchies de classes. L'héritage peut être simple ou multiple. L'héritage évite la duplication et encourage la réutilisation.

Le polymorphisme représente la faculté d'une méthode à pouvoir s'appliquer à des objets de classes différentes. Le polymorphisme augmente la généricité, et donc la qualité, du code.

Agrégation

Il s'agit d'une relation entre deux classes, spécifiant que les objets d'une classe sont des composants de l'autre classe. Une relation d'agrégation permet donc de définir des objets composés d'autres objets. L'agrégation permet donc d'assembler des objets de base, afin de construire des objets plus complexes.

1.2 Histoire des modélisations par objets



Les méthodes utilisées dans les années 1980 pour organiser la programmation impérative (notamment Merise) étaient fondées sur la modélisation séparée des données et des traitements. Lorsque la programmation par objets prend de l'importance au début des années 1990, la nécessité d'une méthode qui lui soit adaptée devient évidente. Plus de cinquante méthodes apparaissent entre 1990 et 1995 (Booch, Classe-Relation, Fusion, HOOD, OMT, OOA, OOD, OOM, OOSE, etc.) mais aucune ne parvient à s'imposer. En 1994, le consensus se fait autour de trois méthodes :

- OMT de James Rumbaugh (*General Electric*) fournit une représentation graphique des aspects statique, dynamique et fonctionnel d'un système ;
- OOD de Grady Booch, définie pour le *Department of Defense*, introduit le concept de paquetage (*package*) ;
- OOSE d'Ivar Jacobson (Ericsson) fonde l'analyse sur la description des besoins des utilisateurs (cas d'utilisation, ou *use cases*).

Chaque méthode avait ses avantages et ses partisans. Le nombre de méthodes en compétition s'était réduit, mais le risque d'un éclatement subsistait : la profession pouvait se diviser entre

ces trois méthodes, créant autant de continents intellectuels qui auraient du mal à communiquer.

Événement considérable et presque miraculeux, les trois gourous qui régnaient chacun sur l'une des trois méthodes se mirent d'accord pour définir une méthode commune qui fédérerait leurs apports respectifs (on les surnomme depuis « the Amigos »). UML (*Unified Modeling Language*) est né de cet effort de convergence. L'adjectif *unified* est là pour marquer qu'UML unifie, et donc remplace.

En fait, et comme son nom l'indique, UML n'a pas l'ambition d'être exactement une méthode : c'est un langage.

L'unification a progressé par étapes. En 1995, Booch et Rumbaugh (et quelques autres) se sont mis d'accord pour construire une méthode unifiée, *Unified Method 0.8* ; en 1996, Jacobson les a rejoints pour produire UML 0.9 (notez le remplacement du mot *méthode* par le mot *langage*, plus modeste). Les acteurs les plus importants dans le monde du logiciel s'associent alors à l'effort (IBM, Microsoft, Oracle, DEC, HP, Rational, Unisys etc.) et UML 1.0 est soumis à l'OMG. L'OMG adopte en novembre 1997 UML 1.1 comme langage de modélisation des systèmes d'information à objets. La version d'UML en cours en 2008 est UML 2.1.1 et les travaux d'amélioration se poursuivent.

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.

1.3 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.

UML 2.0 comporte ainsi treize types de diagrammes représentant autant de *vues* distinctes pour représenter des concepts particuliers du système d'information.

- Un diagramme UML est une représentation graphique, qui s'intéresse à un aspect précis du modèle. C'est une perspective du modèle, pas "le modèle".

- Chaque type de diagramme UML possède une structure (les types des éléments de modélisation qui le composent sont prédéfinis).

- Un type de diagramme UML véhicule une sémantique précise (un type de diagramme offre toujours la même vue d'un système).

- Combinés, les différents types de diagrammes UML offrent une vue complète des aspects statiques et dynamiques d'un système.

- Par extension et abus de langage, un diagramme UML est aussi un modèle (un diagramme modélise un aspect du modèle global).

L'objectif de notre cours étant une première acquisition des concepts de modélisation à travers l'utilisation d'un langage unifié, nous nous limitons à la présentation et la compréhension de la version 11 d'UML. Les 9 diagrammes de celle-ci se répartissent en deux grands groupes :

Diagrammes structurels ou diagrammes statiques (*UML Structure*)

- diagramme de cas d'utilisation (*Use case diagram*)
- diagramme de classes (*Class diagram*)
- diagramme d'objets (*Object diagram*)
- diagramme de composants (*Component diagram*)
- diagramme de déploiement (*Deployment diagram*)

Diagrammes comportementaux ou diagrammes dynamiques (*UML Behavior*)

- 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*)

Ces diagrammes, d'une utilité variable selon les cas, ne sont pas nécessairement tous produits à l'occasion d'une modélisation.

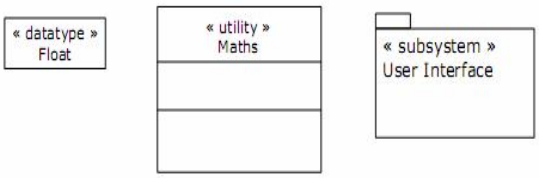
2. Eléments et mécanismes généraux

- Les **Éléments** sont les briques de base du langage
  - Éléments de modélisation: abstractions du système à modéliser.
  - Éléments de visualisation: projections textuelles ou graphiques permettant la manipulation des éléments de modélisation.
  - Exemple: acteur: élément de modélisation; :élément de visualisation

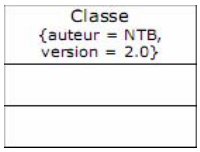
- Des **Mécanismes** sont définis:
  - Stéréotypes (pour extension d'UML)
  - Étiquettes ou valeurs marquées (pour extension d'UML)
  - Notes

- Contraintes (pour extension d'UML)

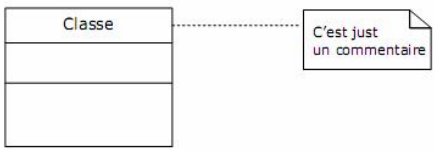
- **Stéréotypes**
  - Un stéréotype est une annotation s'appliquant sur un élément de modèle. Il n'a pas de définition formelle, mais permet de mieux caractériser des variétés d'un même concept. Il permet d'adapter le langage à des situations particulières.
  - Elargir le vocabulaire d'UML
  - Sont employés pour créer de nouveaux types d'éléments en UML qui dérivent des sortes existantes mais qui sont adaptés à un problème donné.
  - Il existe des stéréotypes prédéfinis dans UML.
  - Notation: « nom du stéréotype »
  - Exemple:



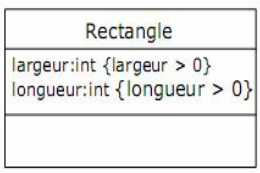
- **Valeurs marquées**
  - Principalement utilisées pour ajouter des informations sur les éléments UML
  - Paire {nom= valeur, version=2.0}
  - Exemple:



- **Les notes**:
  - Commentaires attachés à un ou plusieurs éléments de modélisation.
  - Fournissent l'information supplémentaire sur les éléments de modélisation.



- **Contraintes**
  - Restrictions qui limitent l'utilisation d'un élément.
  - utilisées pour exprimer des relations, les stéréotypes et les valeurs marquées.
  - Les contraintes servent à exprimer la sémantique du profil (Stéréotypes;…valeurs marquées;…Contraintes).
  - Les contraintes en UML sont souvent exprimées en OCL (Object Constraint Language).
  - Exemple :



3. Diagrammes UML

Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation représente la structure des grandes fonctionnalités nécessaires aux utilisateurs du système. C'est le premier diagramme du modèle UML, celui où s'assure la relation entre l'utilisateur et les objets que le système met en œuvre.

Diagramme de classes

Le diagramme de classes est généralement considéré comme le plus important dans un développement orienté objet. Il représente l'architecture conceptuelle du système : il décrit les classes que le système utilise, ainsi que leurs liens, que ceux-ci représentent un emboîtage conceptuel (héritage) ou une relation organique (agrégation).

Diagramme d'objets

Le diagramme d'objets permet d'éclairer un diagramme de classes en l'illustrant par des exemples. Il est, par exemple, utilisé pour vérifier l'adéquation d'un diagramme de classes à différents cas possibles.

Les diagrammes de composants

Les composants sont des codes (sources, exécutables), des scripts, des fichiers, des tables, etc.

Les diagrammes de déploiement

Ces diagrammes illustrent la disposition physique des différents matériels (ou noeuds) qui entrent dans la composition du système, la répartition des composants au sein des nœuds et les supports de communication entre noeuds.

Diagramme d'états-transitions

Le diagramme d'états-transitions représente la façon dont évoluent (*i.e.* cycle de vie) les objets appartenant à une même classe. La modélisation du cycle de vie est essentielle pour représenter et mettre en forme la dynamique du système.

Diagramme d'activités

Le diagramme d'activités n'est autre que la transcription dans UML de la représentation du processus telle qu'elle a été élaborée lors du travail qui a préparé la modélisation : il montre l'enchaînement des activités qui concourent au processus.

Diagramme de séquence et de collaboration

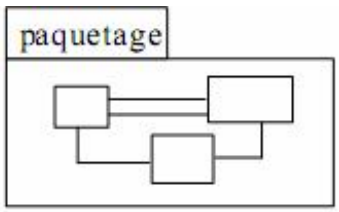
Le diagramme de séquence représente la succession chronologique des opérations réalisées par un acteur. Il indique les objets que l'acteur va manipuler et les opérations qui font passer d'un objet à l'autre. On peut représenter les mêmes opérations par un diagramme de collaboration, graphe dont les nœuds sont des objets et les arcs (numérotés selon la chronologie) les échanges entre objets. En fait, diagramme de séquence et diagramme de collaboration sont deux vues différentes mais logiquement équivalentes (on peut construire l'une à partir de l'autre) d'une même chronologie. Ce sont des diagrammes d'interaction.

4. Paquetages (Packages)

Un **Paquetage** (sous modèle) : c'est une notion qui peut apparaître dans beaucoup de diagrammes pour spécifier le regroupement d'éléments au sein d'un sous système (cas, classes, objets, composants, autres paquetages, ...) et aussi pour encapsuler ces éléments.

Caractéristiques:

- sert d'espace de désignation
- peut inclure d'autres paquetages
- peut importer d'autres paquetages
- L'héritage entre paquetages est possible
- Représentation :



Les éléments contenus dans un paquetage doivent représenter un ensemble fortement cohérent et sont généralement de même nature et de même niveau sémantique.

Généralement, il existe un seul paquetage racine qui détient la totalité du modèle d'un système.