Introduction générale à UML

Origines d'UML

Différentes méthodes de modélisation basée sur les objets ont été développées durant les années 80. Il y en avait plus de 50 au début des années 90. Chacune avait des avantages et des inconvénients, aucune ne faisait l'unanimité. Parmi les trois plus populaires, on pouvait trouver :

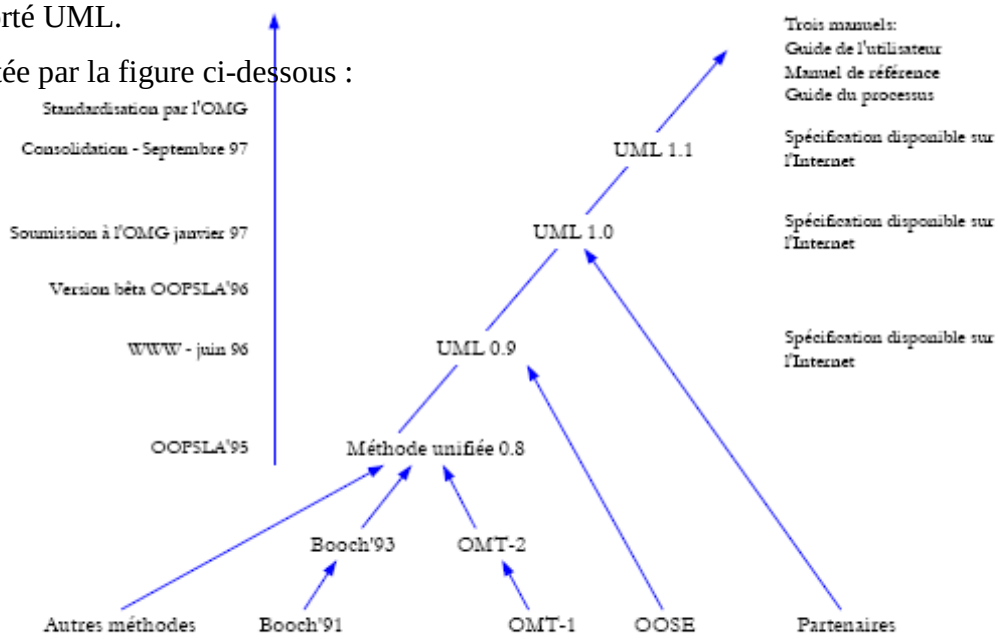
- la méthode de Grady Booch, appelée parfois OOD
- OMT de James Rumbaugh,
- OOSE et Objectory de Ivar Jacobson.

Chacune de ces méthodes a été décrite dans un livre paru au début des années 90, et chacune est défendue par un individu dont les compétences dans le domaine du développement orienté objet sont reconnues.

On aurait pu s'attendre à une guerre des méthodes. En fait, les choses se sont déroulées très différemment. Booch était chez Rational Software (les développeurs de Rose). En 1994, Rumbaugh a rejoint Rational également. Booch et Rumbaugh ont an- noncé leur intention de fusionner leurs méthodes. Les industriels n'étaient pas trop contents de voir une entreprise prendre le marché de cette manière (« à la Microsoft !»). Mais toute résistance était inutile : en 1995, Rational achète l'entreprise de Jacobson, Objectory.

Un premier brouillon de la méthode unifiée est mis à disposition sur l'Internet et largement discuté dans comp.object, puis soumis à approbation de l'OMG (Object Management Group). Plus tard, l'objectif a été réajusté pour développer un langage de modélisation plutôt qu'une méthode, ce qui nous a apporté UML.

L'évolution d'UML est relatée par la figure ci-dessous :



Objectifs d'UML

L'objectif d'UML est de standardiser les artefacts du développement : les modèles,les notations et les diagrammes. Il est également de ne pas standardiser le processus de développement. Toutefois, le processus sera dirigé par les cas d'utilisation, centré sur l'architecture du système, itératif et incrémental.

La validité d'UML est garantie par l'existence d'un méta-modèle : le langage est modélisé en utilisant le formalisme UML. La sémantique des concepts est définie dans le méta-modèle. Ce méta-modèle n'est utilisé que dans les phases de définition du langage pour garantir son intégrité; il est très austère et vraiment peu pédagogique. En revanche son existence garantit la cohérence d'UML : les règles du langage UML se décrivent à l'aide d'UML lui-même.

En résumé, UML est :

- une notation, pas une méthode;
- un langage de modélisation objet;
- le fruit de l'expérience et des besoins de la communauté des utilisateurs
- adapté à toutes les méthodes objet.

La modélisation avec UML ?

UML est un langage pour décrire des modèles, mais il ne définit pas de processus d'élaboration d'un modèle. Il s'agit purement et simplement d'un langage. Il n'induit aucune méthodologie.

Toutefois, pour obtenir une modélisation d'une application informatique, les auteurs d'UML (Booch, Jacobson, Rumbaugh) et Rational Software préconisent une démarche :

- itérative et incrémentale,
- centrée sur les besoins de l'utilisateur,
- centrée sur l'architecture logicielle.

1 Une démarche itérative et incrémentale

Pour établir un modèle d'un système complexe, il vaut mieux s'y prendre en plusieurs passes. Chaque passe consistera à raffiner le modèle obtenu précédemment en le détaillant un peu plus.

Mais cette démarche peut s'appliquer également au processus de développement. Dans le contexte actuel ou les besoins évoluent très rapidement, il est plus intéressant de favoriser l'élaboration rapide de prototypes.

L'objectif de cette démarche est de mieux maîtriser le risque. Par opposition à l'approche fonctionnelle qui préconisait un développement en une passe, le développement incrémental et itératif rendu possible par les technologies objet permet de beaucoup mieux gérer le risque.

2 Une démarche centrée sur les besoins de l'utilisateur

Les cas d'utilisation sont le point d'entrée dans la conception d'un modèle. Il est important de se focaliser sur les besoins des utilisateurs, les cas d'utilisation permettent de les retranscrire formellement. Ce sont les utilisateurs qui définissent ce que doit être le système, et le système est assuré de répondre aux besoins des utilisateurs.

De plus, les besoins des utilisateurs servent de fil conducteur tout au long du développement du système. En effet, le caractère itératif et incrémental du développement permet de soumettre rapidement et fréquemment des prototypes aux utilisateurs :

- lors de la phase d'analyse de chaque itération, on clarifie, affine et valide les besoins des utilisateurs,
- lors de la phase de conception et de réalisation de chaque itération, on réalise la prise en compte des besoins des utilisateurs,
- lors de la phase de test de chaque itération, on vérifie que les besoins des utilisateurs sont satisfaits, éventuellement en leur soumettant un prototype.

3 **Une démarche centrée sur l'architecture logicielle**

Une architecture logicielle adaptée est la clé de voute du succès d'un développement. C'est sur l'architecture choisie que reposeront les qualités du logiciel en termes d'adaptabilité, de performances, de fiabilité, de maintenance, etc.

L'architecture prend en compte tous les aspects, aussi bien logiques au niveau des abstractions, que pratiques en termes d'implémentation et d'exécution. Un système complexe peut être constitués d'un grand nombre de classes, qui seront assemblées en plusieurs modules, ces modules devront coopérer, éventuellement au travers d'un réseau, et pourront être exécutés sur plusieurs machines.

Définition

- **UML** est un langage de modélisation fondé sur des concepts orientés objets et composé d'éléments de relations de diagrammes conçus pour visualiser spécifier construire et documenter un système logiciel.

- **UML** ne décrit pas de méthode ou de processus de développement spécifique.
- Encourage un processus itératif et incrémental piloté par le cas d'utilisation et centré sur l'architecture.

UML permet de définir et de visualiser un modèle, à l'aide de diagrammes. 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).

- **Visualiser =** Symbole graphique ou description textuelles
- **Spécifier =** Modéliser sans ambiguïté
- **Construire =** Traduire directement un modèle UML en langage de programmation
- **Documenter =** Documenter la globalité de l'étude et de la réalisation d'un logiciel.

Structure du langage UML

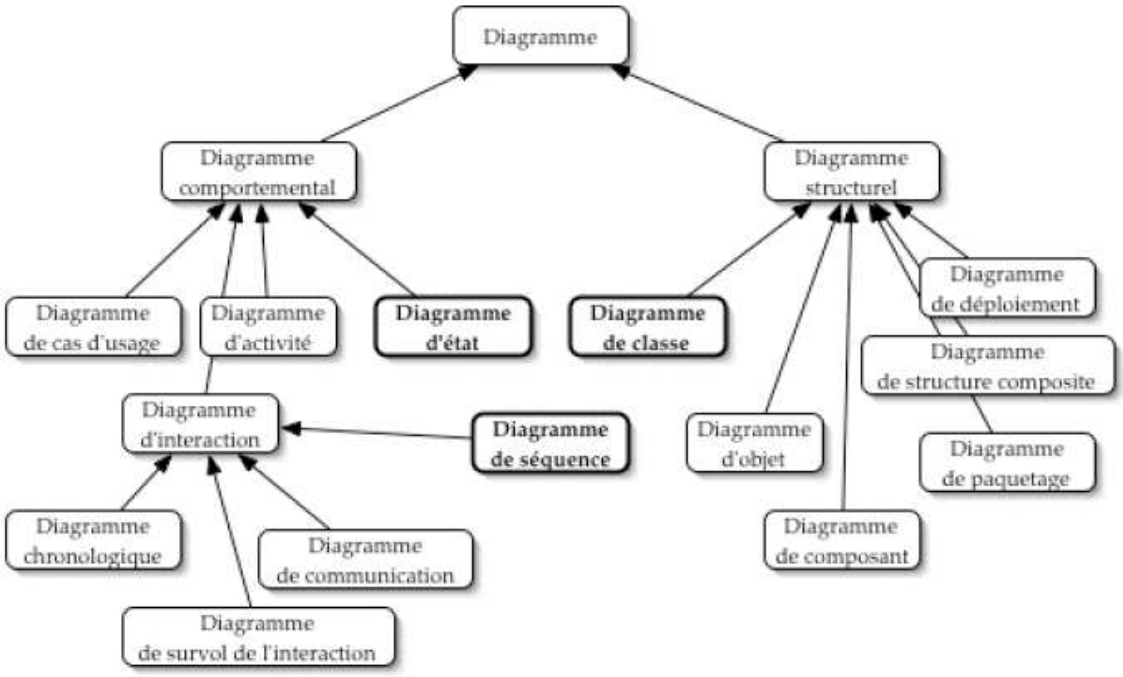
UML propose toute une panoplie de diagrammes qui sont des représentations graphiques des divers aspects d'un logiciel. Voici une classification des différents types de diagrammes UML définis par le groupe de normalisation (Object Management Group ou OMG).

La terminologie fondamentale

- **Les éléments / Composants**: abstraction essentielles d'un modèle
- **Les relations** : permets la relation entre différents éléments.
- **Les diagrammes** : représentation graphique d'un ensemble d'élément du système.

composants	structuraux	classe interface collaboration cas d'utilisation classe active composant noeud
	comportementaux	interaction machine à états
	regroupement	package
	annotation	note
relations		dépendance association généralisation réalisation
diagrammes	vues statiques	diagrammes de cas d'utilisation diagrammes d'objets diagrammes de classes diagrammes de composants diagrammes de déploiement
	vues dynamiques	diagrammes de collaboration diagrammes de séquence diagrammes d'états-transitions diagrammes d'activités

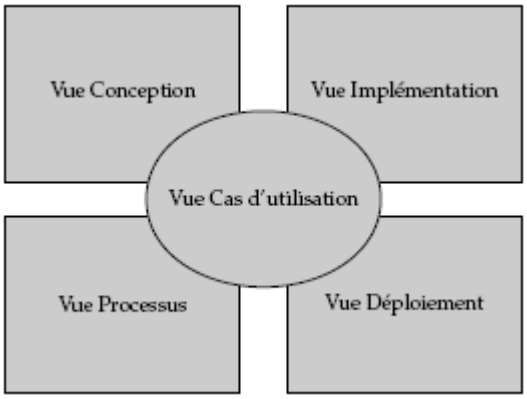
Structure du langage UML



Les diagrammes d'UML

Vue d'ensemble sur UML

Où se place UML dans la conception et la réalisation d'un système ?



Ces cinq vues sont interdépendantes dans l'analyse d'un système, chacune est une projection de l'organisation et de la structure du système selon un axe particulier :

1. **la vue des cas d'utilisation** décrit le comportement du système comme il est vu par les utilisateurs; cette vue ne spécifie pas directement l'organisation du logiciel mais va guider les quatre autres vues :
    - concevoir l'architecture d'un système informatique n'est pas suffisant en soi, il faut qu'elle soit justifiée par les besoins des utilisateurs,
    - il est important voir vital que les besoins des utilisateurs soient au centre de la conception du système,
    - Les scénarios d'utilisation, regroupés en cas d'utilisation, vont conduire à définir une architecture pertinente et cohérente,
    - cette vue est la colle qui va unifier les quatre autres vues;
2. **la vue logique** se concentre sur l'abstraction et l'encapsulation :

- elle modélise les éléments et les mécanismes principaux du système,
- elle identifie les éléments du domaine ainsi que les relations qui les lient : les classes, les interfaces et les collaborations qui font le vocabulaire de la solution apportée au problème et qui sont liés au métier de l'entreprise
- elle organise également (mais purement logiquement) les éléments du do- maine en catégories pour répartir les tâches dans les équipes, pour regrouper ce qui est générique, pour isoler ce qui est propre à une version donnée, etc.

3. **la vue des processus**, très importante pour les systèmes multi-tâches ou répartis, elle montre

- la décomposition du système en terme de processus (ou de threads),
- les interactions entre ces processus,

- la synchronisation des activités qui se déroulent en parallèle;

4. **la vue d'implémentation** montre

- l'allocation des éléments de modélisation dans des modules (fichiers sources, bibliothèques dynamiques, programmes exécutables, bases de données, pages web, etc.),
- comment les classes de la vue logique sont réalisées physiquement,
- l'organisation des composants, leurs dépendances, la distribution du code en gestion de configuration (si plusieurs cibles sont prévues),
- les contraintes de développement (comme des bibliothèques externes),
- également, l'organisation du système en sous-systèmes, avec leurs interfaces et leurs dépendances;

5. **la vue de déploiement**, très importante dans les environnements distribués, décrit les ressources matérielles et la répartition du logiciel dans ces ressources, en fait la topologie matérielle du système :

- la disposition et nature physique des matériels, ainsi que leurs performances,
- l'implantation des modules principaux sur les noeuds du réseau
- les exigences en terme de performances (temps de réponse, tolérance aux fautes et pannes...).