Télé Enseignement - Cnam des Pays de Loire
Méthodologie B8

Le Langage UML

Présentation Générale

Claude Belleil - Université de Nantes

1. Introduction

UML (Unified Modeling Language, "**langage de modélisation objet unifié**") est né de la fusion des trois méthodes qui ont le plus influencé la modélisation objet au milieu des années 90 : OMT, OOD et OOSE.

L'absence de consensus sur une méthode d'analyse objet avait longtemps freiné l'essor des technologies objet. Prenant conscience de cette difficulté, les grands acteurs du monde informatique ont favorisé l'unification et la normalisation des méthodes objet dominantes. **UML est le fruit de cette démarche**.

Comme OMT, UML n'est pas une méthode à proprement parler. Il s'agit en fait d'un **langage de représentation de processus**. UML ne propose pas de démarche permettant d'organiser l'enchaînement des activités d'une organisation.

UML est essentiellement **un support de communication**, qui facilite la représentation et la compréhension de solutions objet. Sa notation graphique permet d'exprimer visuellement une solution objet, ce qui facilite la comparaison et l'évaluation de solutions. L'aspect formel de sa notation, limite les ambiguïtés et les incompréhensions.
Son indépendance par rapport aux langages de programmation, aux domaines d'application et aux processus, en fait un langage universel. D'autre part, la véritable force d'UML, est du au fait qu'il repose sur un **métamodèle**. Ainsi, il normalise la sémantique des concepts qu'il propose**.**

Fin 1997, UML est devenu une norme **OMG**[1] (Object Management Group).

[1] L'OMG est un organisme, créé en 1989 à l'initiative de grandes sociétés (HP, Sun, Unisys, American Airlines, Philips...). Il fédère tous les grands acteurs du monde informatique. Son rôle est de promouvoir des standards qui garantissent l'interopérabilité entre applications orientées objet,

En permettant d'exprimer et d'élaborer des modèles objet, indépendamment de tout langage de programmation, UML comble une lacune importante des technologies objet. Il a été pensé pour servir de support à une analyse basée sur les concepts objet.

Le **métamodèle** d'UML décrit de manière très précise tous les éléments de modélisation (les concepts véhiculés et manipulés par le langage) et la sémantique de ces éléments (leur définition et le sens de leur utilisation). En d'autres termes : **UML normalise les concepts objet**.
Un métamodèle permet de limiter les ambiguïtés et encourage la construction d'outils. Il permet aussi de classer les différents concepts du langage selon leur niveau d'abstraction ou leur domaine d'application. Il expose ainsi clairement sa structure. Enfin, on peut noter que le métamodèle d'UML est lui-même décrit par un méta-métamodèle de manière standardisée, à l'aide de MOF (Meta Object Facility : norme OMG de description des métamodèles).

2. Naissance du Langage UML

Les premières bases d'UML résultent de la fusion de trois méthodes, en 1995 :

2.1. *OMT (Object Modeling Technique)*

OMT a été développée par James Rumbaugh dans le Centre de Recherche et Développement de la société General Electric à la fin des années 80. Le principal ouvrage existant est celui de l'auteur: "Object-Oriented Modeling and Design", paru en 1991. Cette méthode connaît un certain succès en France et en Europe.

OMT propose une triple perception du système d'information :

1. une dimension statique qui décrit le schéma objet du système,
2. une dimension dynamique qui décrit les changements d'états des objets en fonction d'évènements,
3. une dimension fonctionnelle qui décrit les processus de transformation des informations.

Un modèle spécifique correspond à chacune de ces dimensions. Ce modèle a pour objectif de représenter le maximum de sémantique du monde réel à l'aide d'une panoplie d'outils de représentation de classes, d'associations, de relations d'agrégation et de généralisation.
Ce modèle est basé sur:

1. le modèle Entité-Association employé dans les méthodes "relationnelles",
2. le modèle Agrégation-Généralisation
3. des compléments venus du modèle "objet" (classe, instance, héritage).

Il propose quatre phases de modélisation:

1. Analyse

---

développées sur des réseaux hétérogènes. L'OMG propose notamment l'architecture CORBA (Common Object Request Broker Architecture), un modèle standard pour la construction d'applications à objets distribués (répartis sur un réseau).

2. Conception Système
3. Conception Objet
4. Implantation

2.2. *OOD (Object Oriented Development)*

Pionnier de l 'Orienté-Objet, **Grady Booch** publie en 1981 un article intitulé "Object Oriented Development" qui présente une méthode de modélisation objet. Au départ, elle est conçue pour le développement d'applications en langage Ada à destination du « Department of Défense ». Ensuite, elle sera étendue au C++.

Elle applique le modèle objet pour synthétiser la structure logique, physique, statique et dynamique d'une application. Pour cela, elle utilise une notation expressive et clairement définie permettant de:

- capturer la réalité à l'aide de diagrammes de classes, d'objets, de modules, de processus
- communiquer des décisions de conception
- trancher des points de vue différents

La méthode Booch[2] propose des diagrammes de représentation qui se répartissent en 2 niveaux:

Un niveau Logique
Diagrammes de classes
Diagramme d'instance
Diagramme états/transitions
Un niveau Physique
Diagrammes de modules
Diagramme de processus

2.3. *OOSE (Object Oriented Software Engineering)*

La méthode OOSE[3] de Ivar Jacobson couvre tout le cycle de développement du logiciel. Créée dans un centre de recherche d'Ericsson (Suède) , elle repose essentiellement sur l'analyse des besoins des utilisateurs à l'aide des Diagrammes de **cas d'utilisation** formalisés par Ivar Jacobson dans sa méthode, puis introduits dans UML.
Ils décrivent sous la forme d'actions et de réactions, le comportement d'un système du point de vue de l'utilisateur, définissent les limites du système et les relations entre le système et l'environnement. Ils permettent d'introduire une manière spécifique d'utiliser un système. C'est l'image d'une fonctionnalité du système, déclenchée en réponse à la stimulation d'un acteur externe.

2.4. *Unification et normalisation d'UML*

[2] autre nom de la méthode
[3] souvent appelée **Objectory**

Pendant la période allant de 1995 à 1997, les différentes méthodes composant UML ont été unifiées et normalisées. Un certain nombre d'autres méthodes ont également fournit des idées. La version finale d'UML, version 1.1, a été adoptée par l'OMG (Object Management Group) en novembre 1997.

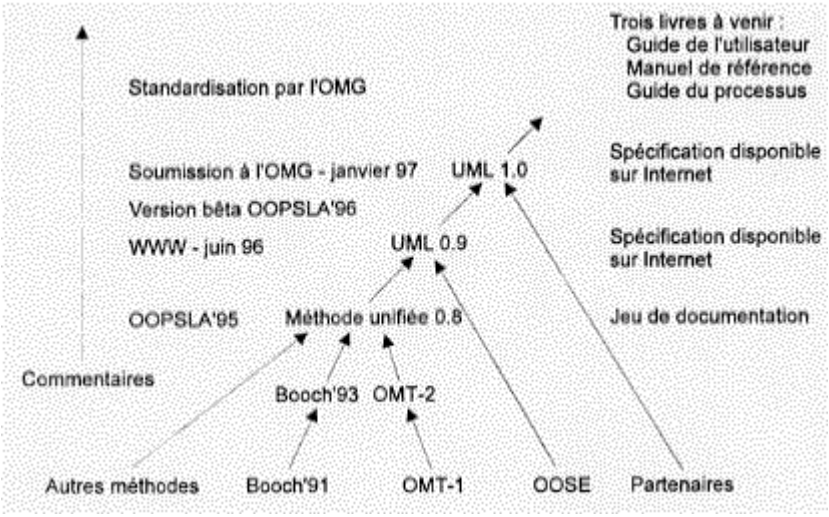


Figure 1: Principales étapes de la définition d'UML

Aujourd'hui, UML atteint sa version 1.4, qui a été examinée début 2000. Un groupe de réflexion travaille déjà sur la version 2.0. L'avancée du travail peut être suivie sur le site de l'OMG (http://www.omg.org/uml).

3. UML: 2+1 types de vues et 9 diagrammes

Le métamodèle UML fournit une panoplie d'outils permettant de représenter l'ensemble des éléments du monde objet (classes, objets, ...) ainsi que les liens qui les relient.
Toutefois, étant donné qu'une seule représentation est trop subjective, UML fournit un moyen astucieux permettant de représenter diverses projections d'une même représentation grâce aux **vues**. Une vue est constituée d'un ou plusieurs **diagrammes**. On distingue deux types de vues:

3.1. *Les vues statiques*

1. diagrammes de cas d'utilisation
2. diagrammes d'objets
3. diagrammes de classes
4. diagrammes de composants
5. diagrammes de déploiement

3.2. *Les vues dynamiques*

6. diagrammes de séquence
7. diagrammes de collaboration

8. diagrammes d'états-transitions
9. diagrammes d'activités

Les diagrammes des cas d'utilisation représentent également une vue fonctionnelle du système.

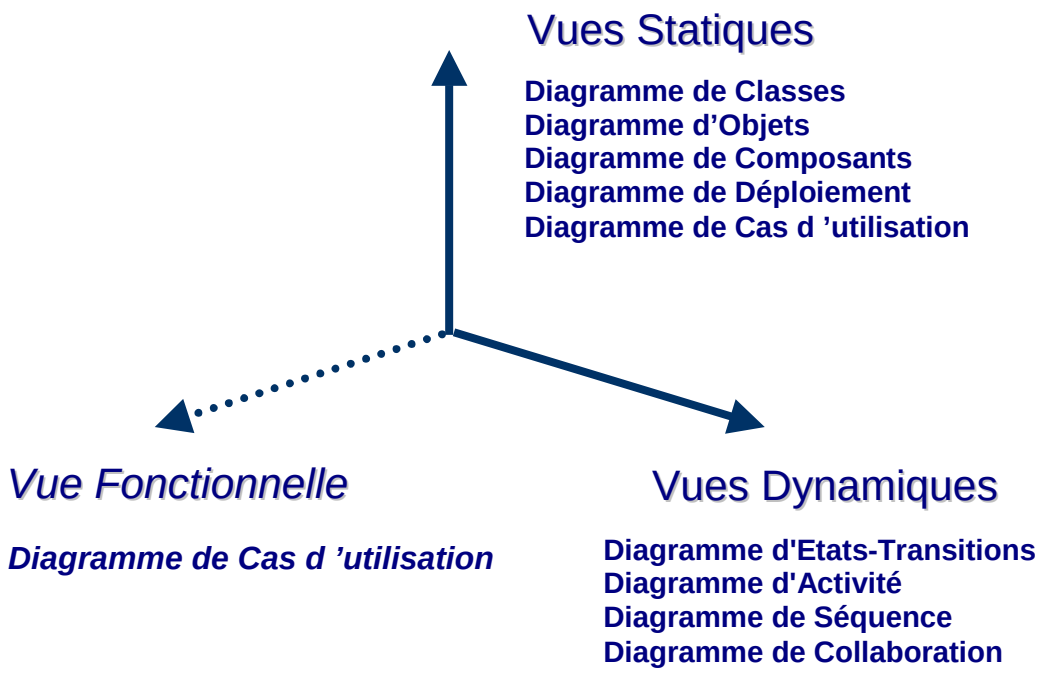


Figure 2: les différentes vues du système

4. La modélisation selon différentes approches

UML offre des éléments de modélisation pour :

4.1. *Les cas d'utilisation*

Les cas d'utilisation partionnent les besoins fonctionnels d'un système, selon le point de vue d'une catégorie d'utilisateurs à la fois. La modélisation par les cas d'utilisation partcipe à une meilleure collaboration entre les différents acteurs d'un projet: les analystes, les utilisateurs et les développeurs.

4.2. *Les classes et les objets*

La modélisation objet est utilisée dans le langage UML pour définir des objets-métier et l'architecture de l'application. Ces objets sont créés en tant qu'instances de classes. Ils interagissent dynamiquement pour offrir le comportement décrit par les cas d'utilisation. La modélisation objet définit le comportement requis par les différentes classes pour assurer la bonne mise en place des cas d'utilisation et des règles de gestion.
Les objets constituent la base de l'architecture des applications. Ils peuvent être réutilisés à travers des domaines d'application ou encore être identifiés et dérivés directement des cas d'utilisation ou du domaine de l'application.

La modélisation des classes capture le détail de la structure des objets. Les classes constituent la base pour la génération de code et pour la génération des schémas de bases de données. Les définitions des classes et de leurs relations sont regroupées dans des paquetages afin de définir l'architecture des applications. Ces paquetages peuvent être emboîtés les uns dans les autres. Les relations entre paquetages définissent les dépendances dans l'application et déterminent la stabilité de l'ensemble de l'architecture.

4.3. *Les composants*

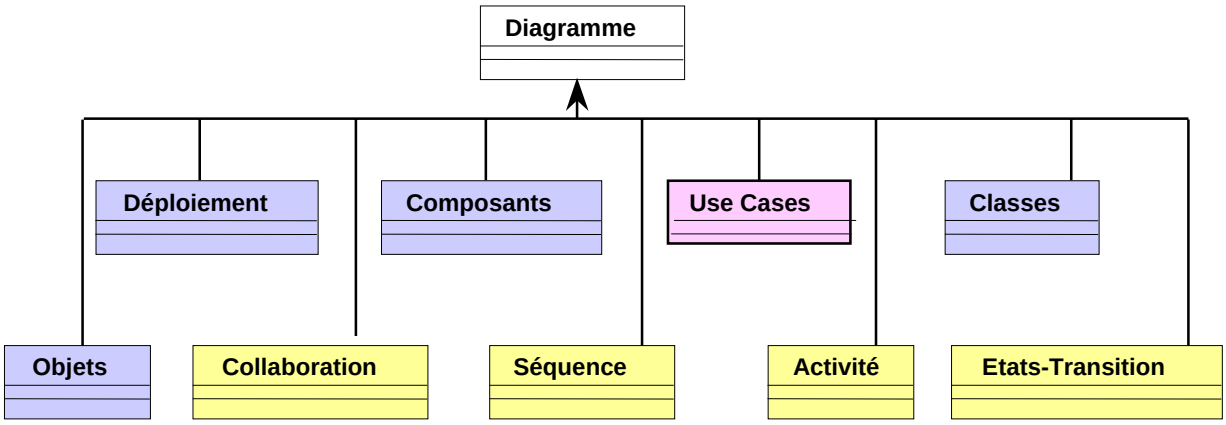
Ce sont les unités physiques de code source et les unités exécutables qui sont assemblées pour former des applications. Les classes sont affectées à des composants fournissant des éléments réutilisables pour la construction des applications. Ces composants formeront la base pour une architecture d'application de type *plug-and-play*.

4.4. *La distribution et le déploiement*

La modélisation de déploiement permet de représenter la façon dont l'application est distribuée dans un réseau. Différentes topologies de réseau peuvent être modélisées. Par exemple les architectures client/serveur, trois-tiers ou Internet/Intranet. UML permet de décrire la topologie des nœuds dans le réseau, la façon dont ces nœuds sont connectés et la manière dont l'application est partitionnée et distribuée sur ces nœuds.

5. Sémantique des neuf diagrammes du langage

UML définit neuf types de diagrammes différents regroupés en six modèles. Un même type de diagramme peut représenter la réalité à des niveaux de granularité différents. Voici une présentation rapide avec pour chacun un exemple graphique.



Nature de la vue	Diagrammes	Sémantique
Statique	cas d'utilisation	décrit les besoins utilisateur
	objet	définit la structure statique
	classes	
	composants	montre les unités de travail
	déploiement	précise la répartition des processus
Dynamique	collaboration	scénarios et flots de messages
	séquence	
	états-transitions	définit le comportement dynamique
	activités	

5.1. *Diagramme de cas d'Utilisation*

Le diagramme de Cas d'utilisation constitue l'apport original de la méthode OOSE à la notation UML. L'approche consiste à regarder le système à construire de l'extérieur, du point de vue de l'utilisateur et des fonctionnalités qu'il en attend. Les cas d'utilisation sont par conséquent très utiles en phase d'analyse des besoins. Ils permettent de modéliser les besoins des clients d'un système. Ils ne doivent pas chercher l'exhaustivité, mais clarifier, filtrer et organiser les besoins. Une fois identifiés et structurés, ces besoins définissent le contour du système à modéliser (ils précisent le but à atteindre), et permettent d'identifier les fonctionnalités principales (critiques) du système.
Les cas d'utilisation ne doivent donc en aucun cas décrire des solutions d'implémentation.

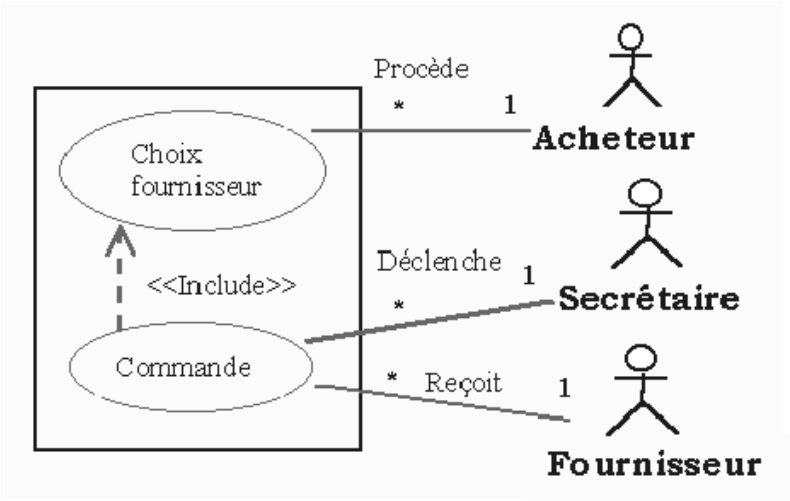


Figure 3: Un exemple de diagramme de cas d'utilisation (Bernard Morand - Caen**)**

5.2. *Diagramme de classes*

Une classe est un type abstrait caractérisé par des propriétés (attributs et méthodes) communes à un ensemble d'objets et permettant de créer des objets ayant ces propriétés.

Classe = attributs + méthodes + instanciation

Un diagramme de classe exprime de manière générale la structure statique du système: les classes et les relations qui existent entre elles. De même qu'une classe décrit un ensemble d'objets, une association entre classes décrit un ensemble de liens entre objets. Un objet est donc une instance de classe, au même titre qu'un lien est une instance d'association. Ce diagramme ne décrit rien en particulier sur les objets, mais exprime les liens potentiels d'un objet vers les autres objets. Chaque extrémité d'une association permet de préciser le rôle joué par chaque classe dans l'association. Il est également possible de préciser pour une association:

- un nom de rôle
- une cardinalité
- la navigabilité
- l'agrégation
- la spécialisation ou la généralisation

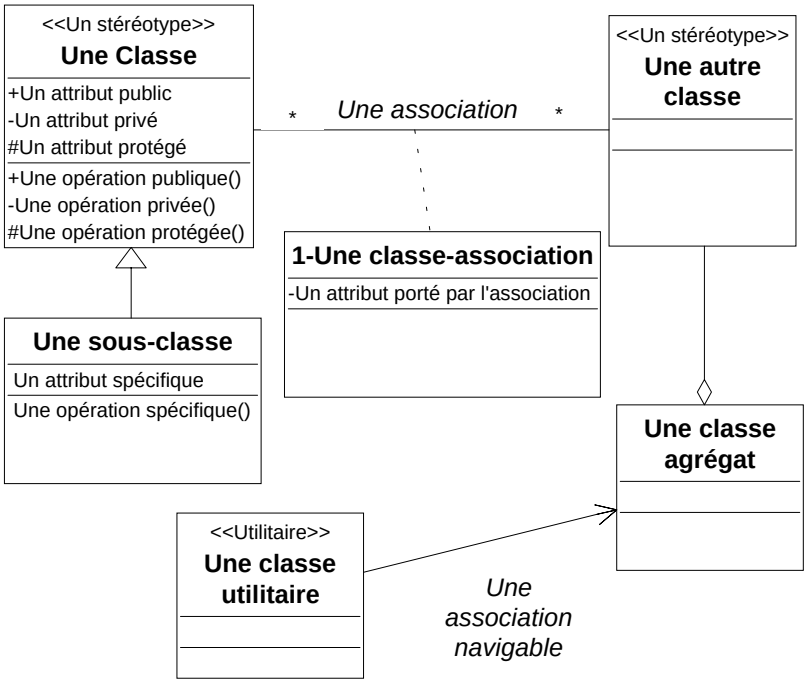


Figure 4: Un exemple de diagramme de classes

5.3. *Diagramme d'objets*

Un objet est une **entité concrète** ou **abstraite** aux frontières précises qui possède une identité (un nom). **L'état** d'un objet est caractérisé par un ensemble d'attributs et de relations. Un ensemble d'opérations (méthodes) en définissent le **comportement**. Un objet est une instance de classe (une occurrence d'un type abstrait).
Une classe est un type de données abstrait, caractérisé par des propriétés (attributs et méthodes) communes à des objets et permettant de créer des objets possédant ces propriétés.

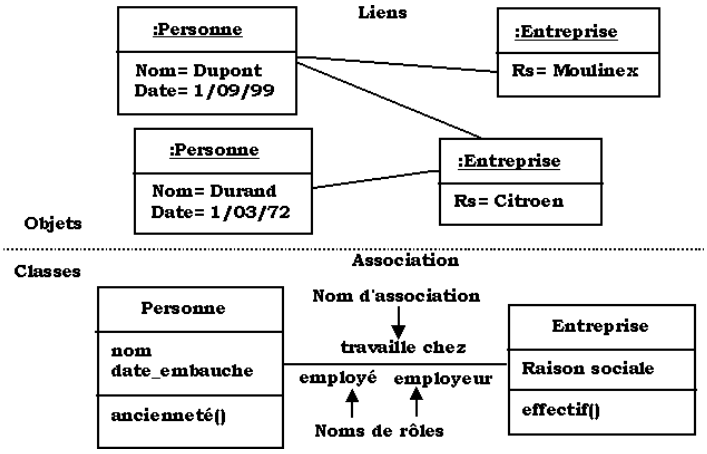


Figure 5: Un exemple de diagramme d'objets et son diagramme de classes (Bernard Morand - Caen)

Un diagramme d'objets montre les objets ainsi que les liens qui existent entre eux. Les objets représentent la structure statique du système modélisé, **à un instant donné**. Ils expriment donc un contexte : ce qui est en mémoire à un moment donné, et quels sont les objets qui peuvent communiquer entre eux.

5.4. *Diagramme de collaboration*

Ces diagrammes montrent les interactions entre les objets. Ils insistent particulièrement sur la structure des liens permettant de mettre les objets en collaboration, en communication. Ils représentent à la fois les objets et les messages que ceux-ci s'envoient. Les diagrammes de collaboration sont une extension des diagrammes d'objets. Un lien sert de support de transmission pour le message. Le message déclenche une action dans l'objet destinataire
Il est également possible de préciser pour un message ses arguments et valeurs de retour

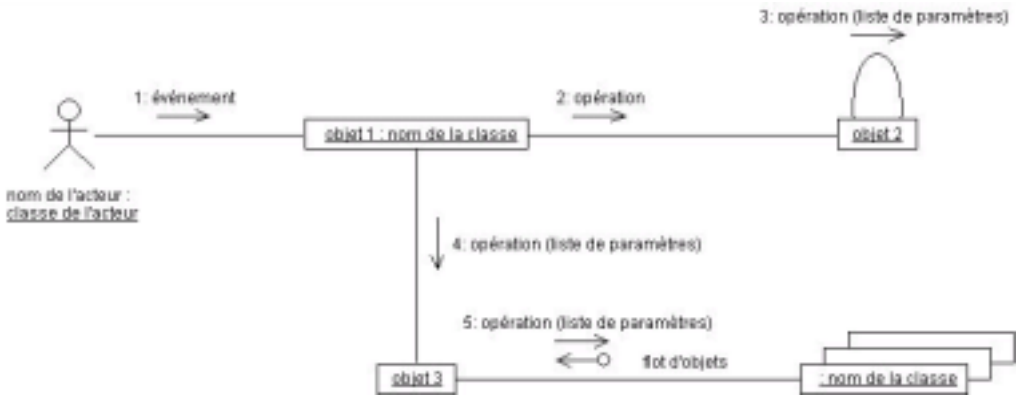


Figure 6: Exemple de diagramme de collaboration entre objets

5.5. *Diagramme de séquences*

Par rapport au diagramme de collaboration Le diagramme de séquence met en évidence l'aspect chronologique de l'envoi des messages. Ce type de diagramme insiste sur l'aspect temporel, alors que le diagramme de collaboration insiste sur l'aspect spatial. C'est l'équivalent du scénario de la méthode OMT. Souvent, le diagramme de séquence permet de compléter le diagramme des cas d'utilisation en mettant en évidence les objets et leur interaction

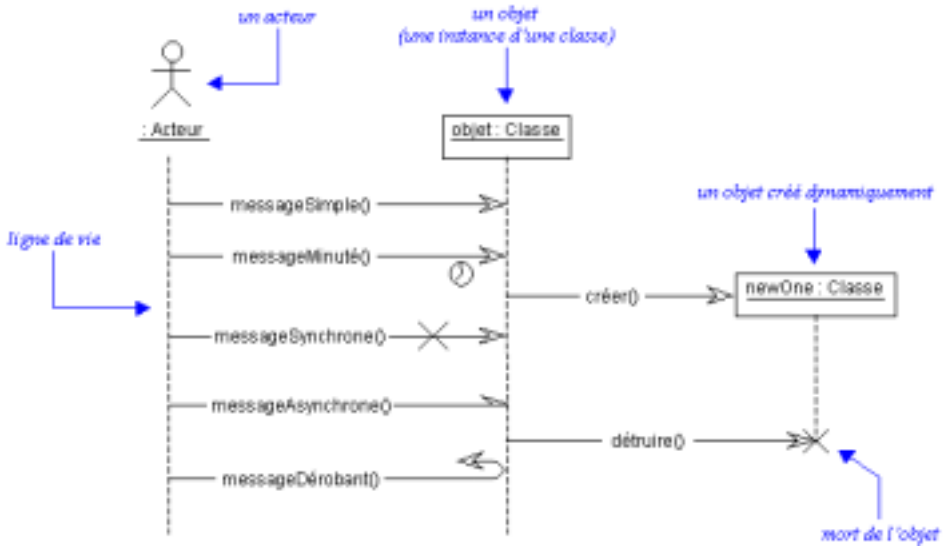


Figure 7: Exemple de diagramme de séquence

5.6. *Diagramme états-transitions*

Le diagramme Etats-Transitions représente la dynamique de chacun des objets (et donc du système entier) au moyen d'automates.
Un automate est composé d'états, modélisant une situation dans laquelle l'objet se situe à un instant précis. L'automate est composé de transitions, précisant comment on passe d'un état à un autre. Il décrit l'évolution au cours du temps d'une instance d'une classe en réponse aux interactions avec d'autres objets
Il est forcément associé à une classe, mais toutes les classes n'en ont pas besoin. C'est un graphe orienté d'états (nœuds) connectés par des transitions (arc orienté)

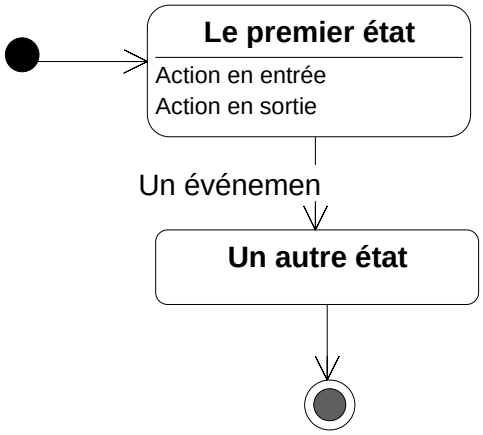
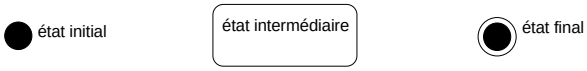


Figure 8: Un exemple de digramme d'Etats- Transitions

Origine: Les Statecharts de David Harel

Chaque objet est à un moment donné dans un état particulier :
Etat Initial : état d'une instance juste après sa création (un seul état initial)
Etat Intermédiaire : un objet est toujours dans un état donné pour un certain temps
Etat Final : état d'une instance juste avant sa destruction (un automate infini peut ne pas avoir d'état final)



5.7. *Diagramme des activités*

Un diagramme d'activité est une variante du diagramme d'états-transitions. Il s'applique à représenter les comportements internes des méthodes et des opérations des objets. C'est en fait un diagramme d'états-transitions qui s'organise par rapport aux actions.

Ils peuvent être utilisés pour:

1. représenter le comportement d'opérations d 'une classe
2. formaliser un processus d'une organisation

Ils peuvent correspondre à des points de vues différents:

1. pour concevoir un objet
2. pour analyser un processus

La notion d'activité peut être considérée vis à vis des éléments suivants:

1. une opération
2. une étape dans une opération
3. une action d'un scénario d'un cas d 'utilisation[4]

[4] On notera que les diagrammes **d'activités** et de **séquences** peuvent être utilisés pour représenter le détail de sénarios modélisés dans des Cas d'Utilisation

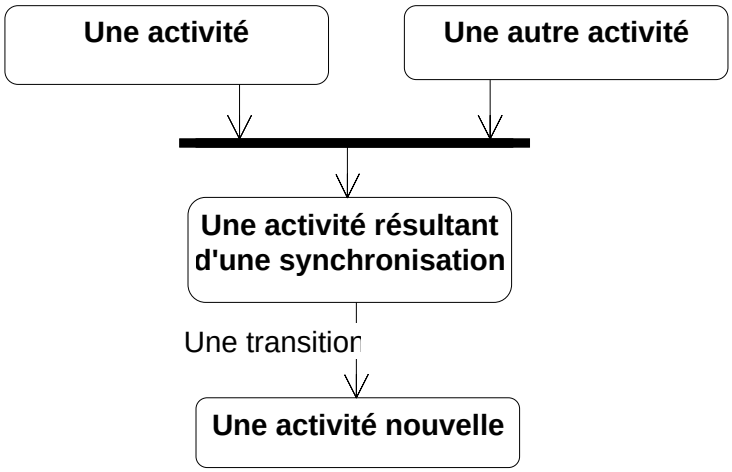


Figure 9: Un exemple de diagramme d'activités

5.8. *Diagrammes de composants*

Ils permettent la description de l'architecture technique, des nœuds et leurs interconnexions.
Les nœuds de l'architecture sont des serveurs, des postes de travail et des périphériques
Les composants sont alloués aux différents nœuds.

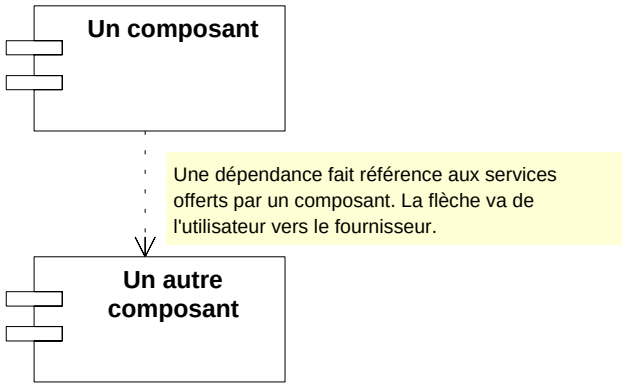
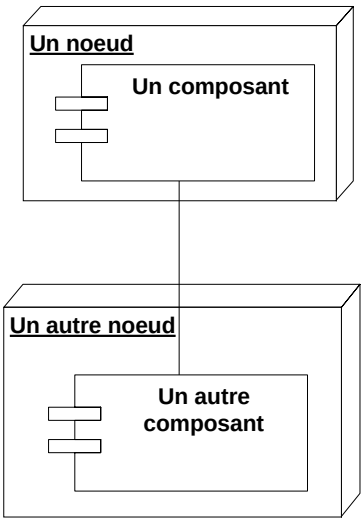


Figure 10: Un exemple de diagramme de composants

5.9. *Diagrammes de déploiement*

Ils sont utilisés pour regrouper des éléments de modélisation : classes, cas d'utilisation ou autres diagrammes.
Ils sont utilisés pour:

- obtenir une vision de plus haut niveau
- mettre en évidence des éléments réutilisables
- répartir le travail entre développeurs
- organiser la gestion de configuration (versionning)



6. UML et les points de vue de modélisation

Le nombre important de diagrammes, la faculté que possèdent la plupart d'entre eux d'être utilisable à n'importe quel niveau d'abstraction amène quelques réflexions sur le « fonctionnement » d'un tel ensemble.

En tant que support d'étude, d'anticipation, de conception et de documentation du système, le modèle doit représenter les points de vue nécessaires aux différents protagonistes du développement. Ces points de vue représentent autant de vitrines d'observation du même problème en mettant en valeur certains contenus et en en masquant d'autres. Les points de vue du modèle d'un système logiciel sont les suivants:

6.1. *Le point de vue de spécification fonctionnelle*

Il concerne l'organisation du modèle des besoins fonctionnels exprimés par les utilisateurs et étudiés par les analystes.

6.2. *Le point de vue structurel*

Il a trait à l'organisation du modèle des besoins élaboré en classes par les analystes.

6.3. *Le point de vue matériel*

Il développe la structure physique des machines et des réseaux sur lesquels repose le système informatique. Il concerne les ingénieurs système et réseau.

6.4. *Le point de vue de déploiement*

Il représente la structure des postes de travail et localise les composants sur le réseau physique. Il concerne les ingénieurs d'exploitation chargés d'installer le logiciel et d'identifier les causes de pannes.

6.5. *Le point de vue d'exploitation*

Il correspond à l'organisation des composants et identifie les fonctions prises en charge par le logiciel installé. Il concerne à la fois les ingénieurs d'exploitation et les concepteurs, les uns pour trouver le composant incriminé par une panne, les autres pour cartographier les dépendances logicielles.

6.6. *Le point de vue de spécification logicielle*

Il concerne les architectes qui décident de répartir par couches les exigences techniques, afin de les dissocier par nature de responsabilités.

6.7. *Le point de vue logique*

Il est relatif à l'organisation du modèle de solution élaboré en classes par les concepteurs.

6.8. *Le point de vue de configuration logicielle*

Il retrace enfin la manière dont sont construits les composants qui servent à l'élaboration d'un sous-système. Il permet de mesurer l'impact d'un changement sur la construction du logiciel et d'établir les configurations et les compatibilités entre plusieurs éléments de versions différentes.

Tous les points de vue n'interviennent ni au même moment, ni au même niveau d'abstraction dans le processus de développement. Il existe par ailleurs des dépendances entre points de vue. Le tableau suivant établit la correspondance entre les diagrammes proposés par UML 1.3 et les points de vue de modélisation.

Diagramme UML	Point de vue							
	Spécification fonctionnelle	Structurel	Matériel	Déploiement	Exploitation	Spécification logicielle	Logique	Configuration logicielle
Classes	O	I				O	I	
Objets		O					O	
Cas d'utilisation	I					I		
Séquence	I					O	I	
Collaboration	I					O	I	
États-transitions		I					I	
Activités	I					I	O	
Composants				I	I			I
Déploiement			I	I				

O : optionnel I : indispensable

7. Conclusion: UML, un standard de fait

L'unification des méthodes de modélisation objet a été rendue possible parce que l'expérience avait permis de faire le tri entre les différents concepts proposés par les différentes méthodes, et ainsi de savoir ce qu'il fallait conserver et ce qu'il fallait laisser de côté.
Partant de la constatation que les différences entre les méthodes s'amenuisaient et que la guerre des méthodes ne faisait plus progresser la technologie des objets, Jim Rumbaugh et Grady Booch décident fin 94 d'unifier leurs travaux au sein d'une méthode unique : la méthode unifiée. Une année plus tard, ils sont rejoints par Ivar Jacobson, le créateur des cas d'utilisation, une technique très efficace pour la détermination des besoins. Les auteurs de la méthode unifiée atteignent très rapidement un consensus sur les concepts fondamentaux de l'objet. La convergence sur les éléments de notation est par contre plus difficile à obtenir, et la représentation graphique retenue pour les différents éléments de modélisation connaîtra plusieurs modifications. Hormis ces problèmes de forme la méthode unifiée a pour objectif de définir un langage commun pour la modélisation objet. Ce langage est conçu comme un langage généraliste, utilisable par tous les types de projets, toutes les équipes de développement, et indépendamment du processus de développement retenu.

7.1. *UML, les apports incontestables*

- Il est le résultat d'un large consensus (industriels, méthodologistes...).
- Il est le fruit d'un travail d'experts reconnus.
- Il couvre toutes les phases d'un cycle de développement.
- Il est indépendant du domaine d'application et des langages d'implémentation.
- Après l'unification et la standardisation, bientôt l'industrialisation d'UML : les outils qui supportent UML se multiplient (GDPro, ObjectTeam, Objecteering, OpenTool, Rational Rose, Rhapsody, STP, Visio, Visual Modeler, WithClass...).
- XMI (format d'échange standard de modèles UML).
- Il évolue mais reste très stable
- L'OMG RTF (nombreux acteurs industriels) centralise et normalise les évolutions d'UML au niveau international.
- Les groupes d'utilisateurs UML favorisent le partage des expériences.
- Il inclut des mécanismes standards d'auto-extension.
- La description de son Métamodèle est standardisée (OMG-MOF).

7.2. *UML n'est pas une méthode de développement*

On parle souvent de méthode objet pour UML par **abus de langage**. Une méthode propose non seulement des outils de représentation du réel perçu, mais également une démarche méthodologique de développement permettant de piloter de façon rationnelle l'analyse et la réalisation concrète d'une l'application informatique. Or, UML a été défini pour permettre de modéliser les processus d'une entreprise, pas pour les organiser!

7.3. *UML est fondé sur un métamodèle*

Ce métamodèle définit les éléments de modélisation, c'est à dire les concepts manipulés par le langage lui-même. Il précise également la sémantique de ces éléments, c'est à dire leur définition et le sens de leur utilisation.
Un métamodèle est une description très formelle de tous les concepts d'un langage. Il limite les ambiguïtés et encourage la construction d'outils. Le métamodèle d'UML permet un classement des concepts du langage selon leur niveau d'abstraction ou leur domaine d'application. Le métamodèle UML est lui-même décrit par un méta-métamodèle (OMG-MOF).

8. Références et Bibliographie

8.1. *Sites*

http://uml.free.fr/
Le site français de référence sur UML

http://www.cetus-links.org/
Plus de 15 000 références de sites sur l'OO ! Exhaustif et bien organisé.

http://www.stm.tj/objet/ (La boîte à objets)
Site francophone qui aborde les méthodes/notations de modélisation,

la description des concepts du paradigme objet (encapsulation, héritage, agrégation...) les notations UML, OMT…

http://www.uml.crespim.uha.fr/ (site de l'ESSAIM)
http://www.uml.crespim.uha.fr/contributions/contributions.html (Les contributions)
Le site web de l'ESSAIM (Ecole supérieure des sciences appliquées pour l'ingénieur - Mulhouse), propose de nombreuses informations sur UML.

http://www.isg.de/people/marc/UmlDocCollection/UMLDocCollection.html
Collection de liens (annotés), vers des documents traitant d'UML.

http://www.krumbach.de/home/jeckle/unified.htm (Jeckle UML publications available online)

Liens, articles, livres, AGL...
http://www.objectnews.com/
Un grand classique. Nombreux liens. Revues de livres.

http://www.rational.com/uml

Les pages UML de Rational Software. Propose notamment quelques articles intéressants sur UML.

http://iamwww.unibe.ch/~scg/OOinfo/ (Object-Oriented Information Sources)
Liens web intéressants. Propose aussi un moteur de recherche.

http://www.laas.fr/~delatour/Igloo/index_image_fr.html (page LIGLOO)
Liens sur le Génie Logiciel Orienté Objet

8.2. *Livres en français*[5]

UML
Le tout en poche
Martin Fowler
Campus Press
Isbn : 2-7440-1090-1
Prix: 65F

Le guide de l'utilisateur UML,
Grady Booch, James Rumbaugh, Ivar Jacobson,
Éditions Eyrolles, février 2000

Merise, vers OMT et UML,
Joseph Gabay,
InterÉditions, 3e édition, février 1998

De Merise à UML,
Nasser Kettani, Dominique Mignet, Pascal Paré, Camille Rosenthal-Sabroux,
Éditions Eyrolles, novembre 1999

Penser objet avec UML et Java,
Michel Lai,

[5] Seul l'achat du premier ouvrage est recommandé (voir avec le secrétariat du Télé Enseignement)

InterÉditions, juin 1999

Intégrer UML dans vos projets,
Nathalie Lopez, Jorge Migueis, Emmanuel Pichon,
coédition Eyrolles, Informatiques Magazine, octobre 1997

Modélisation objet avec UML
Pierre-Alain Muller
Eyrolles (1999) 2-212-08966-X

Modélisation objet avec UML,
Pierre-Alain Muller, Nathalie Gaertner,
Éditions Eyrolles, 2e édition, mars 2000

De Merise à UML
Nasser Kettani & al.
Eyrolles (1998) 2-212-08997-X

UML La notation unifiée de modélisation orientée objet
Michel Lai
InterEditions (1997) 2-7296-0659-9

Vers OMT & UML
Masson (1998) 2-225-83003-7

8.3. *Livres en anglais*

The Unified Modeling Language User Guide
Grady Booch, James Rumbaugh and Ivar Jacobson
Addison Wesley Longman (1999) 0-201-57168-4

The Unified Modeling Language Reference Manual
James Rumbaugh, Ivar Jacobson and Grady Booch
Addison Wesley Longman (1999) 0-201-30998-X

The Unified Software Development Process
Ivar Jacobson, Grady Booch and James Rumbaugh
Addison Wesley Longman (1999) 0-201-57169-2

Visual Modeling with Rational Rose and UML
Terry Quatrani
Addison Wesley Longman (1998) 0-201-31016-3

UML Distilled
Martin Fowler with Kendall Scott
Addison Wesley Longman (1998) 0-201-32563-2

Use Case Driven Modeling with UML
Doug Rosenberg with Kendall Scott
Addison Wesley Longman (1999) 0-201-43289-7

Index du texte:

Index des illustrations