

UML en Francais

par [Laurent Piechocki](#)

Date de publication :

Dernière mise à jour : 22/10/2007

En l'espace d'une poignée d'années seulement, UML est devenu un standard incontournable. La presse spécialisée foisonne d'articles exaltés et à en croire certains, utiliser les technologies objet sans UML relève de l'hérésie. Lorsqu'on possède un esprit un tant soit peu critique, on est en droit de s'interroger sur les raisons qui expliquent un engouement si soudain et massif ! UML est-il révolutionnaire ?

I - Introduction

I-A - Modélisation, modèle ?

II - UML, OMG, OMA et c ie.

III - Penser objet avec UML, pour concevoir objet.

IV - Un langage universel et visuel.

V - UML comme cadre d'une analyse objet.

VI - UML : le chemin vers l'unification des processus.

VII - Conclusion.

I - Introduction

UML (Unified Modeling Language, traduisez "**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, Booch et OOSE.

Issu "du terrain" et fruit d'un travail d'experts reconnus, UML est le résultat d'un large consensus. De très nombreux acteurs industriels de renom ont adopté UML et participent à son développement.

En l'espace d'une poignée d'années seulement, UML est devenu un standard incontournable. La presse spécialisée foisonne d'articles exaltés et à en croire certains, utiliser les technologies objet sans UML relève de l'hérésie. Lorsqu'on possède un esprit un tant soit peu critique, on est en droit de s'interroger sur les raisons qui expliquent un engouement si soudain et massif ! UML est-il révolutionnaire ?

L'approche objet est pourtant loin d'être une idée récente. Simula, premier langage de programmation à implémenter le concept de type abstrait à l'aide de classes, date de 1967 ! En 1976 déjà, Smalltalk implémente les concepts fondateurs de l'approche objet : encapsulation, agrégation, héritage. Les premiers compilateurs C++ datent du début des années 80 et de nombreux langages orientés objets "académiques" ont étayés les concepts objets (Eiffel, Objective C, Loops...).

Il y donc déjà longtemps que l'approche objet est devenue une réalité. Les concepts de base de l'approche objet sont stables et largement éprouvés. De nos jours, programmer "objet", c'est bénéficier d'une panoplie d'outils et de langages performants. L'approche objet est une solution technologique incontournable. Ce n'est plus une mode, mais un réflexe quasi-automatique dès lors qu'on cherche à concevoir des logiciels complexes qui doivent "résister" à des évolutions incessantes.

Oui, mais... Tout n'est pas si rose. Beaucoup on cédé aux sirènes de l'orienté objet et leur aveuglement a fait couler bien des projets...

Premier hic : l'approche objet est moins intuitive que l'approche fonctionnelle. Malgré les apparences, il est plus naturel pour l'esprit humain de décomposer un problème informatique sous forme d'une hiérarchie de fonctions atomiques et de données, qu'en terme d'objets et d'interaction entre ces objets.

Or, rien dans les concepts de base de l'approche objet ne dicte comment modéliser la structure objet d'un système de manière pertinente. Quels moyens doit-on alors utiliser pour mener une analyse qui respecte les concepts objet ? Sans un cadre méthodologique approprié, la dérive fonctionnelle de la conception est inévitable... Beaucoup l'ont appris à leurs dépens.

Autre problème critique : **l'application des concepts objet nécessite une très grande rigueur.** Le vocabulaire précis est un facteur d'échec important dans la mise en oeuvre d'une approche objet (risques d'ambiguïtés et d'incompréhensions). Beaucoup de développeurs (même expérimentés) ne pensent souvent objet qu'à travers un langage de programmation. Or les langages orientés objet ne sont que des outils qui proposent une manière particulière d'implémenter certains concepts objet. Ils ne valident en rien l'utilisation de ces moyens techniques pour concevoir un système conforme à la philosophie objet.

Connaître C++ ou Java n'est donc pas une fin en soi, il faut aussi savoir se servir de ces langages à bon escient. La question est donc de savoir "qui va nous guider dans l'utilisation des concepts objet, si ce ne sont pas les langages orientés objet ?".

Pour finir : comment comparer deux solutions de découpe objet d'un système si l'on ne dispose pas d'un moyen de représentation adéquat ? Il est très simple de décrire le résultat d'une analyse fonctionnelle, mais qu'en est-il d'une découpe objet ?

Pour remédier à ces inconvénients majeurs de l'approche objet, il nous faut donc :

- 1 un langage (pour s'exprimer clairement à l'aide des concepts objets), qui doit permettre de
 - représenter des concepts abstraits (graphiquement par exemple),
 - limiter les ambiguïtés (parler un langage commun, au vocabulaire précis, indépendant des langages orientés objet),
 - faciliter l'analyse (simplifier la comparaison et l'évaluation de solutions).
- 2 une démarche d'analyse et de conception objet, pour
 - ne pas effectuer une analyse fonctionnelle et se contenter d'une implémentation objet, mais penser objet dès le départ,
 - définir les vues qui permettent de décrire tous les aspects d'un système avec des concepts objets.

En d'autres termes : **il faut disposer d'un outil qui donne une dimension méthodologique à l'approche objet et qui permette de mieux maîtriser sa richesse.**

La prise de conscience de l'importance d'une méthode spécifiquement objet ("comment structurer un système sans centrer l'analyse uniquement sur les données ou uniquement sur les traitements, mais sur les deux"), ne date pas d'hier. Plus de 50 méthodes objet sont apparues durant le milieu des années 90 (Booch, Classe-Relation, Fusion, HOOD, OMT, OOA, OOD, OOM, OOSE...). Aucune ne s'est réellement imposée.

L'absence de consensus sur une méthode d'analyse objet a longtemps freiné l'essor des technologies objet. Ce n'est que récemment que les grands acteurs du monde informatique ont pris conscience de ce problème. L'unification et la normalisation des méthodes objet dominantes (OMT, Booch et OOSE) ne datent que de 1995. UML est le fruit de cette fusion.

UML, ainsi que les méthodes dont il est issu, s'accordent sur un point : une analyse objet passe par une modélisation objet.

I-A - Modélisation, modèle ?

Un modèle est une abstraction de la réalité. L'abstraction est un des piliers de l'approche objet. Il s'agit d'un processus qui consiste à identifier les caractéristiques intéressantes d'une entité en vue d'une utilisation précise. L'abstraction désigne aussi le résultat de ce processus, c'est-à-dire l'ensemble des caractéristiques essentielles d'une entité, retenues par un observateur.

Un modèle est une vue subjective, mais pertinente de la réalité. Un modèle définit une frontière entre la réalité et la perspective de l'observateur. Ce n'est pas "la réalité", mais une vue très subjective de la réalité. Bien qu'un modèle ne représente pas une réalité absolue, un modèle reflète des aspects importants de la réalité, il en donne donc une vue juste et pertinente.

Le caractère abstrait d'un modèle doit notamment permettre de faciliter la compréhension du système étudié. Il réduit la complexité du système étudié, permet de simuler le système, le représente et reproduit ses comportements. Concrètement, un modèle réduit (décompose) la réalité, dans le but de disposer d'éléments de travail exploitables par des moyens mathématiques ou informatiques.

UML permet donc de modéliser une application selon une vision objet. Mais plus concrètement, qu'est UML ?

UML possède plusieurs facettes. C'est une norme, un langage de modélisation objet, un support de communication, un cadre méthodologique. UML est tout cela à la fois, ce qui semble d'ailleurs engendrer quelques confusions...

Je vous invite maintenant à découvrir ces différents visages d'UML et par là même, à vous forger votre propre opinion...

II - UML, OMG, OMA et c ie.

Fin 1997, UML est devenu une norme **OMG** (Object Management Group). Cela vous semble certainement sans importance, mais pourtant...

L'OMG est un organisme à but non lucratif, créé en 1989 à l'initiative de grandes sociétés (HP, Sun, Unisys, American Airlines, Philips...). Aujourd'hui, l'OMG fédère plus de 850 acteurs du monde informatique. Son rôle est de promouvoir des standards qui garantissent l'interopérabilité entre applications orientées objet, 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). Pour rester simple, on peut considérer CORBA comme une généralisation de l'architecture clients/serveurs aux objets.

Au centre de l'architecture CORBA, un routeur de messages (ORB : Object Request Broker) permet à des objets clients d'envoyer des requêtes et de recevoir des réponses, sans avoir à se préoccuper des détails techniques propres à l'infrastructure du réseau. L'ORB se charge de transmettre les requêtes aux objets concernés, où qu'ils se trouvent (dans le même processus, dans un autre, sur un autre noeud du réseau...). Le bus CORBA (dont l'ORB est le composant central) permet d'assurer la transparence des invocations de méthodes ; les requêtes aux objets semblent toujours être locales.

De plus, l'ORB assure une coopération entre objets qui est indépendante des langages de programmation. Le modèle CORBA permet en effet de se focaliser sur les interfaces des objets, qu'on exprime en IDL (Interface Definition Language). L'IDL décrit de manière standard l'interface d'un objet, en faisant abstraction des détails qui relèvent de son d'implémentation. Avec CORBA, il n'est donc pas nécessaire de savoir comment les objets sont codés pour utiliser leurs services. L'utilisation d'IDL comme langage pivot dans la construction d'une architecture, permet de gérer l'hétérogénéité.

Cette approche, qui consiste à masquer les couches techniques du réseau (système d'exploitation, processeur, langage de programmation...), permet d'assurer l'interopérabilité de toute application à objets distribués, conforme au modèle CORBA.

CORBA fait partie d'une vision globale de la construction d'applications réparties, appelée OMA (Object Management Architecture) et définie par l'OMG. Sans rentrer dans les détails, on peut résumer cette vision par la volonté de **favoriser l'essor industriel des technologies objet**, en offrant un ensemble de **solutions technologiques non propriétaires**, qui suppriment les clivages techniques.

UML a été adopté (normalisé) par l'OMG et intégré à l'OMA, car il participe à cette vision et parce qu'il répond à la "philosophie" OMG.

III - Penser objet avec UML, pour concevoir objet.

Pour penser et concevoir objet, il faut savoir "prendre de la hauteur", jongler avec des concepts abstraits, indépendants des langages d'implémentation et des contraintes purement techniques. Les langages de programmation ne sont pas un support d'analyse adéquat pour "concevoir objet". Ils ne permettent pas de décrire des solutions en terme de concepts abstraits et constituent un cadre trop rigide pour mener une analyse itérative.

Pour conduire une analyse objet cohérente, il ne faut pas directement penser en terme de pointeurs, d'attributs et de tableaux, mais en terme d'association, de propriétés et de cardinalités... Utiliser le langage de programmation comme support de conception ne revient bien souvent qu'à juxtaposer de manière fonctionnelle un ensemble de mécanismes d'implémentation, pour résoudre un problème qui nécessite en réalité une modélisation objet.

Au risque d'en décourager certains et d'en décevoir d'autres, l'approche objet nécessite une analyse réfléchie, qui passe par différentes phases exploratoires ! Bien que raisonner en terme d'objets semble naturel, l'approche fonctionnelle reste la plus intuitive pour nos esprits cartésiens... Voilà pourquoi il ne faut pas se contenter d'une implémentation objet, mais se discipliner à "penser objet" au cours d'une phase d'analyse préalable.

Toutes les dérives fonctionnelles de code objet ont pour origine le non respect des concepts de base de l'approche objet (encapsulation...) ou une utilisation détournée de ces concepts (héritage sans classification...). Ces dérives ne sont pas dues à de mauvaises techniques de programmation ; la racine du mal est bien plus profonde ! Bref, programmer en C++ ou en Java n'implique pas forcément concevoir objet...

Les difficultés de mise en oeuvre d'une approche "réellement objet" ont engendré bien souvent des déceptions, ce qui a longtemps constitué un obstacle important à l'essor des technologies objet. Beaucoup ont cédé au leurre des langages de programmation orientés objet et oublié que le code n'est qu'un "moyen". Le respect des concepts fondamentaux de l'approche objet prime sur la manière dont on les implémente. Ne penser qu'à travers un langage de programmation objet est un mirage qui vous détourne de l'essentiel.

Pour sortir les technologies objet de cette impasse fatale, l'OMG propose UML.

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

Mais UML offre bien plus encore ! C'est un **langage formel**, défini par un **métamodèle**.

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) et 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). Cela vous semble peut-être anecdotique, mais prouve si nécessaire le soin apporté par l'OMG pour définir UML.

Véritable clé de voûte de l'OMA, UML est donc un outil indispensable pour tous ceux qui ont compris que programmer objet, c'est d'abord concevoir objet. UML n'est pas à l'origine des concepts objets, mais il en constitue une **étape majeure**, car il unifie les différentes approches et en donne une définition plus formelle.

IV - Un langage universel et visuel.

UML est avant tout un support de communication performant, 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 font un langage universel.

Petit aparté :

La notation graphique d'UML n'est que le support du langage. La véritable force d'UML, c'est qu'il repose sur un métamodèle.

En d'autres termes : **la puissance et l'intérêt d'UML, c'est qu'il normalise la sémantique des concepts qu'il véhicule !**

Qu'une association d'héritage entre deux classes soit représentée par une flèche terminée par un triangle ou un cercle, n'a que peu d'importance par rapport au sens que cela donne à votre modèle. La notation graphique est essentiellement guidée par des considérations esthétiques, même si elle a été pensée dans ses moindres détails.

Par contre, utiliser une relation d'héritage, reflète l'intention de donner à votre modèle un sens particulier. Un "bon" langage de modélisation doit permettre à n'importe qui de déchiffrer cette intention de manière non équivoque ! Il est donc primordial de s'accorder sur la sémantique des éléments de modélisation, bien avant de s'intéresser à la manière de les représenter.

Le métamodèle UML apporte une solution à ce problème fondamental.

UML est donc bien plus qu'un simple outil qui permet de "dessiner" des représentations mentales... **Il permet de parler un langage commun**, normalisé mais accessible, car visuel. Il représente un juste milieu entre langage mathématique et naturel, pas trop complexe mais suffisamment rigoureux, car basé sur un métamodèle.

V - UML comme cadre d'une analyse objet.

Une autre caractéristique importante d'UML, est qu'il cadre l'analyse. UML permet de représenter un système selon différentes vues complémentaires : les diagrammes. Un diagramme UML est une représentation graphique, qui s'intéresse à un aspect précis du modèle ; c'est une perspective du modèle.

Chaque type de diagramme UML possède une structure (les types des éléments de modélisation qui le composent sont prédéfinis) et véhicule une sémantique précise (il 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. Les diagrammes permettent donc d'inspecter un modèle selon différentes perspectives et guident l'utilisation des éléments de modélisation (les concepts objet), car ils possèdent une structure.

Une caractéristique importante des diagrammes UML, est qu'ils supportent l'abstraction. Cela permet de mieux contrôler la complexité dans l'expression et l'élaboration des solutions objet.

UML opte en effet pour **l'élaboration des modèles**, plutôt que pour une approche qui impose une barrière stricte entre analyse et conception. Les modèles d'analyse et de conception ne diffèrent que par leur niveau de détail, il n'y a pas de différence dans les concepts utilisés. UML n'introduit pas d'éléments de modélisation propres à une activité (analyse, conception...) ; le langage reste le même à tous les niveaux d'abstraction.

Cette approche simplificatrice facilite le passage entre les niveaux d'abstraction. L'élaboration encourage une approche non linéaire, les "retours en arrière" entre niveaux d'abstraction différents sont facilités et la traçabilité entre modèles de niveaux différents est assurée par l'unicité du langage.

UML favorise donc le prototypage, et c'est là une de ses forces. En effet, modéliser une application n'est pas une activité linéaire. Il s'agit d'une tâche très complexe, qui nécessite une approche itérative, car il est plus efficace de construire et valider par étapes, ce qui est difficile à cerner et maîtriser.

UML permet donc non seulement de représenter et de manipuler les concepts objet, il sous-entend une démarche d'analyse qui permet de concevoir une solution objet de manière itérative, grâce aux diagrammes, qui supportent l'abstraction.

VI - UML : le chemin vers l'unification des processus.

Grâce au principe d'élaboration des modèles, UML permet de mieux maîtriser la part d'inconnu et d'incertitudes qui caractérisent les systèmes complexes. Mais cet aspect méthodologique d'UML ne doit pas vous induire en erreur. UML est un langage qui permet de représenter des modèles, mais il ne définit pas le processus d'élaboration des modèles ! Qualifier UML de "méthode objet" n'est donc pas tout à fait approprié.

Une méthode propose aussi un processus, qui régit notamment l'enchaînement des activités de production d'une entreprise. Or UML n'a pas été pensé pour régir les activités de l'entreprise ; ce n'est pas DOD-STD-2167A ou CMM.

Les auteurs d'UML sont tout à fait conscients de l'importance du processus, mais ce sujet a été intentionnellement exclu des travaux de l'OMG. Comment prendre en compte toutes les organisations et cultures d'entreprises ? Un processus est adapté (donc très lié) au domaine d'activité de l'entreprise ; même s'il constitue un cadre général, il faut l'adapter au contexte de l'entreprise. Bref, améliorer un processus est une discipline à part entière, c'est un objectif qui dépasse très largement le cadre de l'OMA.

Cependant, même si pour l'OMG, l'acceptabilité industrielle de la modélisation objet passe d'abord par la disponibilité d'un langage d'analyse objet performant et standard, les auteurs d'UML préconisent d'utiliser une démarche :

- guidée par les besoins des utilisateurs du système,
- centrée sur l'architecture logicielle,
- itérative et incrémentale.

D'après les auteurs d'UML, un processus de développement qui possède ces qualités fondamentales "devrait" favoriser la réussite d'un projet.

Une source fréquente de malentendus sur UML a pour origine la faculté d'UML de modéliser un processus, pour le documenter et l'optimiser par exemple. En fin de compte, qu'est-ce qu'un processus ? Un ensemble d'activités coordonnées et régulées, en partie ordonnées, dont le but est de créer un produit (matériel ou intellectuel). UML permet tout à fait de modéliser les activités (c'est-à-dire la dynamique) d'un processus, de décrire le rôle des acteurs du processus, la structure des éléments manipulés et produits, etc...

Une extension d'UML ("UML extension for business modeling") propose d'ailleurs un certain nombre de stéréotypes standards (extensions du métamodèle) pour mieux décrire les processus.

Le RUP ("Rational Unified Process"), processus de développement "clé en main", proposé par Rational Software, est lui aussi modélisé (documenté) avec UML. Il offre un cadre méthodologique générique qui repose sur UML et la suite d'outils Rational.

Plus récemment, VALtech propose le 2TUP ("**2 Tracks Unified Process**", cf. "**UML en action**", ed. Eyrolles), un processus unifié (c'est-à-dire construit sur UML, itératif, centré sur l'architecture et conduit par les cas d'utilisation), qui apporte une réponse aux contraintes de changement continu imposées aux systèmes d'informations des entreprises.

L'axiome fondateur du 2TUP a été le constat que toute évolution imposée au système d'information peut se décomposer et se traiter parallèlement, suivant un axe fonctionnel et un axe technique. A l'issue des évolutions du modèle fonctionnel et de l'architecture technique, la réalisation du système consiste à fusionner les résultats de ces deux branches du processus.

Bien qu'un processus universel soit une utopie, la capitalisation des meilleures pratiques, à travers une famille de processus unifiés (tels que le RUP et le 2TUP), devient donc une réalité.

VII - Conclusion.

Comme UML n'impose pas de méthode de travail particulière, il peut être intégré à n'importe quel processus de développement logiciel de manière transparente. UML est une sorte de boîte à outils, qui permet d'améliorer progressivement vos méthodes de travail, tout en préservant vos modes de fonctionnement.

Intégrer UML par étapes dans un processus, de manière pragmatique, est tout à fait possible. La faculté d'UML de se fondre dans le processus courant, tout en véhiculant une démarche méthodologique, facilite son intégration et limite de nombreux risques (rejet des utilisateurs, coûts...).

Intégrer UML dans un processus ne signifie donc pas révolutionner ses méthodes de travail, mais cela devrait être l'occasion de se remettre en question, en s'inspirant des meilleures pratiques, capitalisées à travers les processus unifiés (RUP et 2TUP).

