

# Que nous apporte Réellement UML2?

## 2<sup>ème</sup> partie : UML2 pour chaque phase du cycle de vie

© copyright SOFTEAM 2005

Philippe Desfray (voir à [propos de l'auteur](#))  
jeudi 24 novembre 2005  
Version 0.1

## I. Présentation

Décembre 2006. La commission "UML RTF" de l'OMG<sup>1</sup> va terminer ses travaux sur UML2, consacrant ainsi l'aboutissement de ce standard. Adopté en Juin 2004, UML2.0 qui est une révision majeure de UML1.4, est déjà supporté par un nombre important d'ateliers UML, d'autres devant annoncer prochainement leur support. Il reste que UML1.4 demeure très majoritairement utilisé, les nouveaux concepts UML2 n'étant pas encore bien compris, mis en œuvre et méthodologiquement supportés.

Cependant que la poussière des volutes marketing retombe, il est temps de faire un point sur les vrais apports du standard UML2.0. Ce "white paper" rédigé par un contributeur actif<sup>2</sup> à l'élaboration de ce standard décrit ses vrais apports et les fausses nouveautés, en fournissant un éclairage pratique sur ce standard.

Ce document est la deuxième partie d'un White Paper décomposé en trois articles publiés séparément:

- 1 *Cible et portée de UML2: Nous décrivons à qui UML2 s'adresse, et son degré d'universalité,*
- 2 *UML2 pour chaque phase du cycle de vie: L'intérêt et la pratique de UML pour chaque étape du cycle de vie sont discutés, avec des exemples pratiques,*
- 3 *Choisir UML2 et MDA: La mesure de l'ampleur du changement de UML1.4 vers UML2 est établie, et l'intérêt de l'adoption conjointe d'une démarche MDA est évalué.*

---

<sup>1</sup> Revision Task Force de l'organisme de standardization Object Management Group, chargé de consolider le standard UML2.0, en prenant en compte les remarques émises sur le standard de part le monde

<sup>2</sup> Les soumissionnaires initiaux (core team) ayant construit le standard sont: Computer Associates, Ericsson, I-Logix, IBM, IONA, Kabira Technologies, Motorola, Oracle, Rational Software, SOFTEAM, Telelogic, Unisys, et WebGain

## II. UML2 pour les phases amont

UML dispose depuis ses débuts des diagrammes de Use Case qui adressent les phases amont, en représentant les modes d'utilisation d'un système sans s'intéresser à son fonctionnement et aux choix d'implémentation. UML2 apporte quelques outils complémentaires pour les phases amont, notamment les flux d'information, utiles pour donner une représentation globale d'un système dès les premières réflexions. La Figure 1 donne un premier aperçu d'une entreprise, de ses différents services (sous systèmes), des flux échangés entre ces services ou avec des entités externes comme typiquement le client (au centre de nos préoccupations, faut il le rappeler). L'avantage de ce type de diagramme est qu'il permet de formaliser un problème dès les phases initiales, sous une forme intelligible par tous les intervenants. Même un décideur (en général retors à toute forme de modélisation) comprendra ce type de diagramme, pourra le commenter et se l'approprier. Une fois cette étape franchie, des questions du type:

- Comment sera représenté un "bon de production"?
- Qui depuis le département "commercial" prend l'initiative d'émettre un "bon de production"?
- Qui depuis le département "production" est responsable de traiter un "bon de production"?
- Quand et comment est émis un "bon de production"?

... vont permettre de construire un modèle détaillé, répondant à ces questions. Ce modèle sera guidé par ces premiers éléments de modélisation, validés par le plus grand nombre.

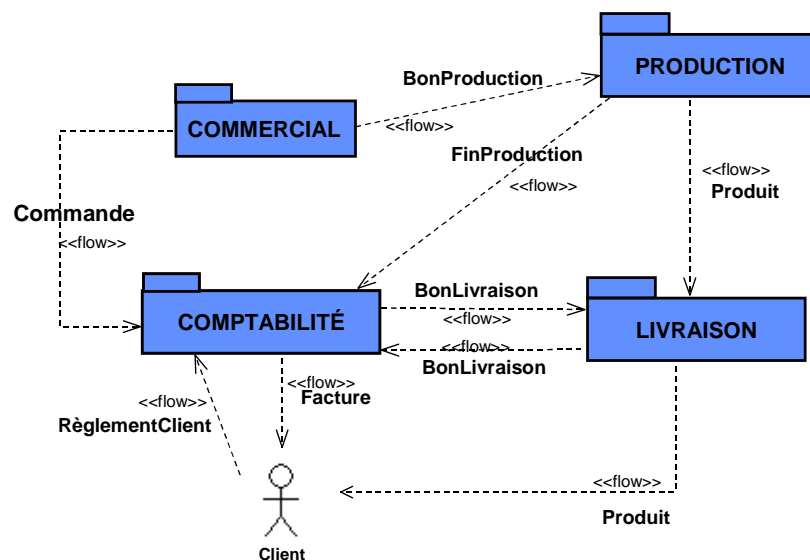


Figure 1 – Représentation générale d'un "système" entreprise

Avec les cas d'utilisation (Use Case), les flux d'information sont des éléments informels, permettant de faire de premiers modèles qui n'induisent pas de décisions sur la manière détaillée de représenter le problème. La notion de modèles informels est très importante pour

adresser les besoins des phases amont, comme par exemple les activités de la maîtrise d'ouvrage ou des activités d'analyse préliminaire<sup>3</sup>.

Par ailleurs, les diagrammes de classe, bien connus sous UML1.4 restent utiles pour réaliser des modèles conceptuels, et les diagrammes d'activité offrent tous les moyens de modéliser les processus métier.

### III. UML2 pour l'architecture

UML2 apporte de nouveaux outils adaptés à la modélisation d'architecture. Les notions de "structure interne" de classe, de port et de part sont des notions nouvellement introduites offrant des mécanismes utiles pour les architectures. A travers ces mécanismes, UML met en avant la notion de "composant", et permet de représenter leur assemblage.

Ces mécanismes d'assemblage permettent de réaliser le vieux rêve du "lego logiciel", c'est à dire d'assembler des composants qui ne se connaissent pas obligatoirement, pour former un système englobant. Pour ceux qui ont connu l'enfer de l'intégration logicielle, ou pire encore l'intégration logiciel/matériel, ces mécanismes sont très prometteurs.

UML2 permet de formaliser les contrats des différents composants, leurs points d'interconnexion, et ensuite d'assembler ceux-ci.

Prenons un exemple couramment pratiqué: *une session vidéo est réalisée en prenant un portable PC, un vidéo projecteur, et en connectant les deux par un câble*. La connexion s'effectuera en branchant le câble sur les prises VGA de ces deux composants. Le miracle de la vie courante fait qu'un non initié à l'électronique et l'informatique peut simplement assembler ces deux composants complexes pour obtenir un système plus sophistiqué encore. La Figure 2 montre comment UML2 permet de représenter ce scénario: ceci n'était pas possible avec UML1.4. Dans cette figure, on identifie les "part" qui sont "mon pc" et "mon vidéo", on visualise les prises de sortie (les "port") ainsi que le câble connectant les prises (le "connecteur").

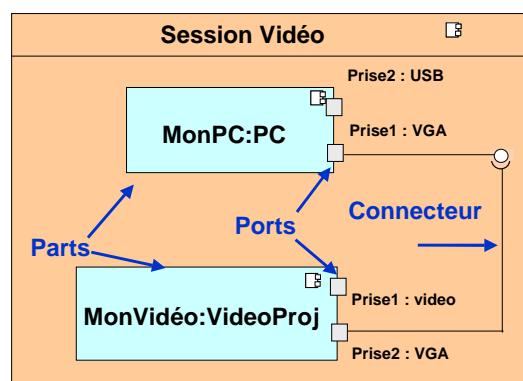


Figure 2 – Assemblage d'un PC et d'un vidéo projecteur

Lors de la définition des composants "PC" et "VideoProj", les ports sont définis, ceux-ci déclarant qu'ils requièrent ou fournissent une interface "VGA". Il sera très important de spécifier l'interface VGA, avec des pré et post conditions déterminant précisément les conditions d'emploi.

<sup>3</sup> Voir du même auteur le white paper: Réussir la modélisation UML des phases amont ([http://www.objecteering.com/pdf/whitepapers/fr/modelisation\\_des\\_phases\\_amont.pdf](http://www.objecteering.com/pdf/whitepapers/fr/modelisation_des_phases_amont.pdf))

Cet exemple simple illustre ce que l'on veut faire pour les architectures: l'architecture se représente sous la forme d'assemblage de composants, via des modes de connexion devant être détaillés. On imagine aisément les types de composants souhaités (infrastructures, bibliothèques, modules fonctionnels, etc.), et on connaît les types de connexion que l'on veut utiliser (réseaux, services web, accès directs, etc.).

UML fournit ainsi un outillage complet pour effectuer la cartographie applicative d'un système d'information, avec par ailleurs la capacité de présenter le déploiement des applications sur le matériel. Il permet de modéliser des architectures, comme typiquement les architectures "SOA" où les notions de port et de connecteur trouvent une correspondance naturelle.

Dans le monde technique, un profil UML a été standardisé pour modéliser l'ingénierie des systèmes. Le domaine de l'ingénierie des systèmes, qui adresse notamment les grands ensembles des domaines avioniques, militaire et technique, va donc bénéficier d'un standard exploitant certains aspects de UML. Les mécanismes d'assemblage de composants ici présentés et les flux d'information ont été deux apports très importants de UML2 pour ce domaine.

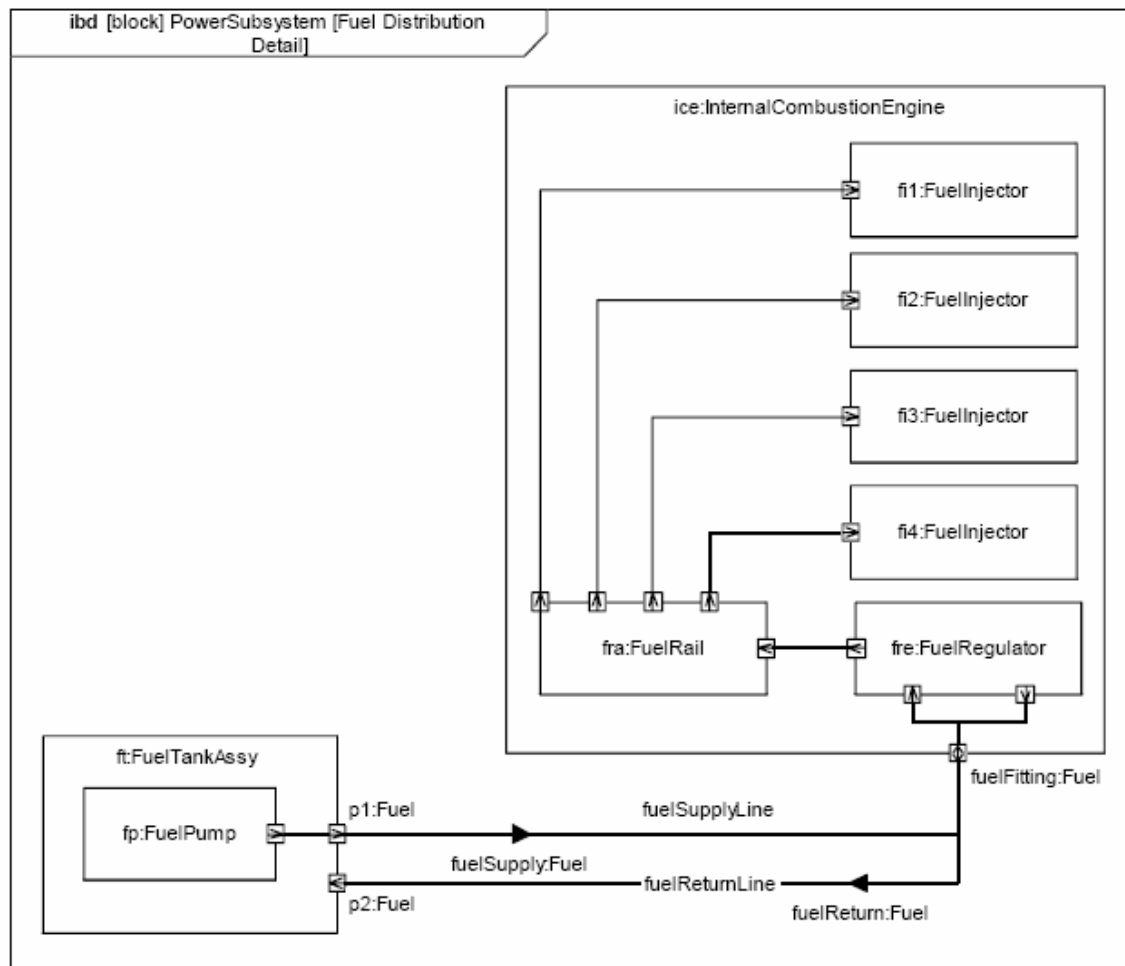


Figure 3 – Système de combustion de carburant – diagramme de structure UML étendu SysML

Dans le domaine des logiciels embarqués, la difficulté majeure est de faire coopérer la définition des architectures "hardware", avec celle des architectures "software". UML2 permet de représenter ces deux architectures, et leurs projections, notamment grâce à ces notions de ports et de parts. Des outils simulent les propriétés techniques de la projection, et permettent de détecter si une architecture globale est viable avant des essais réels coûteux.

Dans le domaine des systèmes d'information, l'approche SOA prend une ampleur croissante. L'approche SOA nécessite plus encore qu'auparavant de modéliser les architectures, qui se traduisent en composants logiciels, connectés à des bus d'entreprise, via des points de connexion définis. Là encore, les modèles de déploiement UML2, les notions de connecteurs, de ports et de composants sont primordiales. Par ailleurs, au-delà de la simple connexion technique des éléments (présences de connecteurs, et ports identifiés, signatures et types compatibles), la modélisation d'une architecture SOA nécessite une formalisation sémantique des services interconnectés. Le support des pré et post conditions de UML devient un moyen incontournable d'assurer la viabilité d'un modèle d'architecture SOA.

## IV. UML2 pour la programmation

Le succès de UML s'est effectué pour une bonne part à travers les capacités de génération de code d'après le modèle. Certains utilisent UML comme un langage de programmation visuelle, qui augmente directement la productivité des développeurs. S'il y a peu de plus value à représenter une classe comme un rectangle au lieu de déclarer "class MaClasse {...}", certaines constructions abstraites n'ayant pas de correspondance directe en code, comme par exemple les associations, apportent effectivement un facteur de génération<sup>4</sup> supérieur à 10. De nombreux ateliers UML permettent ainsi de cibler les langages de programmation d'après un modèle. Ainsi, l'atelier Objecteering produit ou a produit du code pour les langages Java, C++, C#, C, Fortran, Visual Basic, SQL ou CORBA/idl.

UML2 a renforcé son support pour les langages de programmation, d'une part en adressant mieux certaines capacités de représentation de construction de langage, et d'autre part en apportant des constructions nouvelles de haut niveau (comparables aux associations), capables de produire un code opérationnel avec un fort facteur de génération.

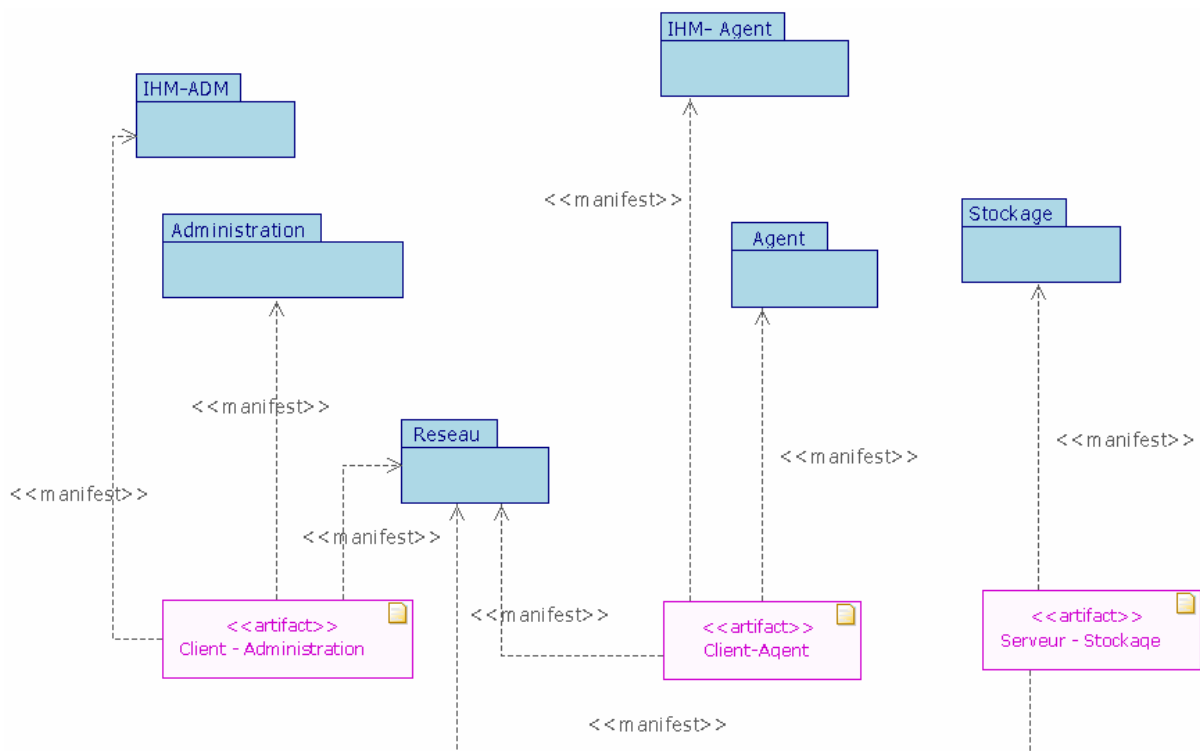
- UML2 a refondu la modélisation des "template", mal supporté sous UML1.4. Inspirée de la notion de généricité initialement développée avec les langages Ada puis Eiffel, la notion de "template" a été supportée par C++, et apparaît dans les dernières versions de Java et C#. UML2 permet de modéliser les "template", de placer des contraintes sur les paramètres de classes "template", de référer des paramètres "template" comme des types pouvant être exploités au sein d'une classe template, et de modéliser l'instanciation de classes templates en classes concrètes (binding). Cette capacité permet de supporter pleinement ce mécanisme de programmation depuis un modèle.
- Les *classes structurées* et les notions de port et part permettent de modéliser des composants logiciels, qui se traduisent au niveau du code par la production de composants indépendants et connectables, et par la génération automatique de l'assemblage de composants codé. Paradoxalement, la plus value maximale n'est pas spécifiquement apparente avec les infrastructures de composants de type EJB ou ".Net", mais avec la capacité de structurer en composants du code réalisé avec des langages tels que C++ ou Java: une application dans ces langages peut être structurée en termes de composants indépendants, avec des points d'interaction prédéfinis, et avec des assemblages mettant en œuvre ces composants dans des contextes différents. Le code généré procure une isolation et une autonomie de chaque partie, très difficile à obtenir par programmation manuelle avec des équipes de développement. Cette capacité est précieuse, car elle permet de réutiliser des composants, de formaliser les parties à développer et leurs modalités d'assemblage, c'est à dire qu'elle permet une sous-traitance

---

<sup>4</sup> nombre de lignes de code produites/nombre de constructions de modèle effectuées

renforcée du développement logiciel, avec une capacité de maîtrise de l'intégration garantie.

- Il est une partie de travail de codage dont la complexité est élevée et la maîtrise problématique: la chaîne de production de code. Des outils spécifiques comme par exemple "makefile", "Ant" ou "Maven" permettent de définir la construction d'un programme et de reproduire la construction d'une application d'après le code source. Difficiles à maintenir, jamais documentées, ces chaînes de production sont de véritables talons d'Achille pour beaucoup d'applications. UML2 a développé la notion d'"Artifact", qui représente les éléments produits à partir d'un modèle: Un "artifact" peut correspondre à un source code, une librairie binaire, une application, une librairie externe utilisée, etc. UML2 permet de modéliser ce que l'on veut produire à partir d'un modèle (les "artifacts"), de manière indépendante des modèles. Ainsi, un même modèle peut produire certains "artifacts" pour les tests locaux, d'autres pour l'intégration, et d'autres pour un portage sur d'autres environnements. Ces modèles peuvent être alors exploités par les ateliers UML<sup>5</sup> pour obtenir la ou les chaînes de production du code à partir d'un modèle: par cette action, la chaîne de production est formalisée, documentée et automatisée. La Figure 4 présente ainsi le modèle d'un système, décomposé en packages, et leur répartition sur 3 "artifacts" pour le client "administration du système", le client "agent dans le système", et le serveur de stockage. En annotant la nature des "artifacts" voulus, comme par exemple "librairie" ou "application", l'atelier UML peut produire les déclarations nécessaires pour une chaîne de production. Ces mêmes "artifacts" sont exploités pour modéliser le déploiement de ces éléments produits sur la plateforme d'exécution.



**Figure 4** – 3 artifacts déterminent les code à produire pour le serveur et deux clients

<sup>5</sup> L'atelier Objecteering, dans sa nouvelle version 6 produira des "makefiles" pour C++, puis des déclarations "Ant ou Maven" pour Java, à partir d'un modèle abstrait des artifacts.

Le potentiel majeur de UML2 sera néanmoins celui fournit par l'extension des capacités de *modélisation de la dynamique*: les capacités étendues de représentation des diagrammes d'activité et des diagrammes de séquence UML permettent en effet de modéliser les constructions dynamiques de code, tels que les envois de message, les boucles, mais aussi les manipulations d'ensemble comme par exemple celle des associations. Les travaux sur UML exécutable sont une piste sur laquelle travaillent de nombreux éditeurs d'ateliers et groupes de standardisation. Déjà existent des prototypes pour simuler UML à partir de modèles dynamiques, et des prototypes de génération de code.

Il reste cependant à trouver le bon compromis, qui apporte une forte plus value en terme de productivité, sans masquer les capacités et la puissance des langages modernes et de leurs environnements.

## V. A Propos de l'auteur

**Philippe DESFRAY**, co-fondateur et directeur technique de SOFTEAM, est un expert internationalement reconnu des modèles et méthodes, en particulier centrés sur UML. Créateur de la méthode objet “Classe Relation” dans les années 1990, il a publié trois livres, en particulier “Object Engineering - The fourth dimension - ADDISON WESLEY - 1994.”, et a piloté le développement de l’atelier UML « Objecteering ». Précurseur des technologies « MDA<sup>6</sup> », il a introduit dès 1994 des outils supports de cette approche. Depuis 1994, Philippe Desfray représente de SOFTEAM au sein de l’OMG où il participe à l’élaboration de plusieurs standards, dont notamment UML. Ses travaux continus sur le développement guidé par le modèle l’ont conduit à participer, dès l’origine, à la définition du standard UML, en y dirigeant la définition de nouvelles notions comme par exemple les « profil UML », et à faire développer son support outillé au sein de l’atelier Objecteering. Dirigeant une forte activité R&D au sein de SOFTEAM et en coopération avec de grands organismes européens, Philippe Desfray œuvre pour une application directe des résultats au sein des diverses activités de SOFTEAM : conseil, formation et support outillé par l’atelier Objecteering.

---

<sup>6</sup> « Model Driven Architecture » : démarche et technologie sous tendant les nouveaux standards de l’OMG