Comment former les ingénieurs système à la modélisation objet ?

4ème Conférence Annuelle d'Ingénierie Système
« Efficacité des entreprises et satisfaction des clients »
Centre de Congrès Pierre Baudis, TOULOUSE, 2-4 mai 2006

Pascal Roques
Valtech Training
Immeuble Tersud - Bâtiment B
5, avenue Marcel Dassault
31500 Toulouse (France)
pascal.roques@valtech-training.fr

Résumé.
Depuis quelques années, la modélisation objet avec le langage UML est devenue incontournable sur la plupart des projets informatiques. Du fait de la part croissante du logiciel dans les systèmes modernes, le langage UML (et son dérivé SysML) est de plus en plus utilisé pour la modélisation système en remplacement d'approches plus anciennes (SADT, etc.).
Comment former les ingénieurs système à ces nouvelles techniques de modélisation, en gommant le plus possible l'orientation informatique, telle est la question que nous allons traiter dans cette présentation. Nous nous appuierons pour cela sur une expérience de plus de 12 ans en formation UML dans des domaines très variés incluant en particulier le secteur aérospace.

UML pour la modélisation système

On trouve sur le site de l'AFIS une excellente explication à la montée en puissance de l'approche objet en modélisation système.

« La complexification des problèmes et des produits a conduit à des démarches de modélisation systémique. Dans une première approche, dite réductionniste, la représentation du système a d'abord été essentiellement fonctionnelle : on représentait le système sous forme d'une architecture de fonctions s'échangeant des flux qui se concrétisait par une architecture d'organes réalisant les fonctions. Confrontés aux problèmes de régulation et de pilotage, les automaticiens ont introduit des modèles de comportement intégrant les aspects continus et événementiels (et représentant l'enchaînement des fonctions). Confrontés à l'abstraction du logiciel, les informaticiens ont ajouté les modèles sémantiques (modèles de données) structurant le monde du problème à informatiser.
Du fait du poids croissant de la part informationnelle des systèmes et de la complexité inhérente à l'aspect purement abstrait des logiciels qui « l'implémentent », ce type de modélisation systémique (fonctions, comportement, sémantique) s'est plus particulièrement développé sous l'influence des informaticiens. Leur évolution vers les approches dites « orientées objet » les a conduit à passer, pour les systèmes à prépondérance informatique, d'une vision fonctionnelle où le système transforme des flux et se décompose en éléments fonctionnels (missions, fonctions de service, sous-fonctions) également transformateurs de flux, à une vision de type « client-service » où le système répond aux sollicitations des acteurs de l'environnement et se décompose ainsi en cas d'utilisation, réalisés par des collaborations d'objets (au sens du logiciel) se rendant mutuellement des services.
Ainsi a émergé un langage unifié **UML** (Unified Modelling Language) de représentation des systèmes d'information dans un contexte de préparation à la programmation par objets. »

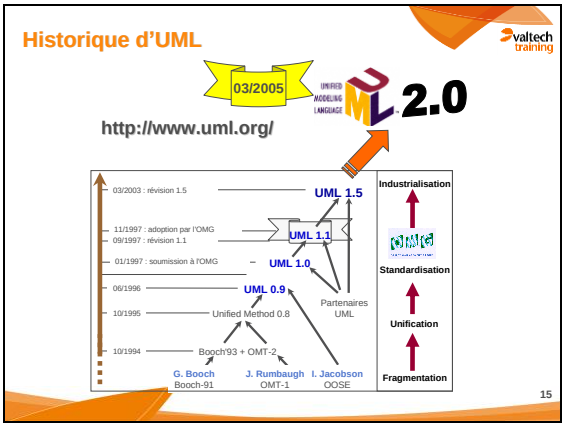

Figure 1. Historique d'UML

Il est intéressant de noter que dès sa création, le langage UML a été pensé pour une utilisation plus large que le seul logiciel. Citons J. Rumbaugh (Rumbaugh 1999), le père de la méthode OMT, et l'un des trois premiers experts à l'origine d'UML : « *UML is intended to be a universal general-purpose modeling*

language for discrete systems such as those made of software, firmware, or digital logic. ».

D'autres spécialistes des systèmes temps-réels ont également prôné l'utilisation des concepts objet pour modéliser des systèmes embarqués ou temps-réel, comme Bran Selic (Selic 1994) et Bruce Douglass (Douglass 1999, Douglass 2004).

Pourquoi cet engouement pour les concepts objets ? C'est que cette approche permet une intégration des différents points de vue de modélisation nécessaires pour les systèmes complexes :

- Le point de vue fonctionnel, représenté par les cas d'utilisation du futur système, complétés par des scénarios opérationnels (diagrammes de séquence ou d'activité) ;
- Le point de vue structurel, représenté sous forme de classes d'objets, encapsulant des attributs et des opérations, et reliées par des associations ou des relations de généralisation / spécialisation ;
- Le point de vue dynamique, représenté à la fois par les diagrammes d'états associés aux classes le nécessitant, et aux diagrammes d'interactions mettant en jeu des objets de classes différentes dans des scénarios particuliers.

L'ensemble des 13 diagrammes d'UML 2.0 permet également d'utiliser les notions de composant et de package pour représenter des éléments d'architecture à gros grain, des « sous-systèmes », tout en gardant la philosophie de modélisation objet.

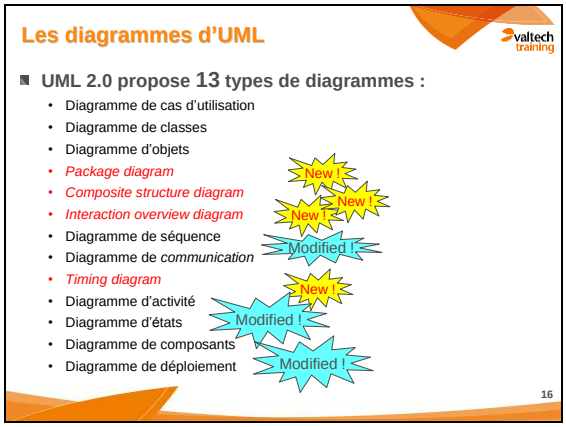

Figure 2. Les 13 diagrammes d'UML 2.0

De la modélisation fonctionnelle à la modélisation objet

Le principal problème à la formation continue des ingénieurs système est d'arriver à les faire passer en douceur d'une approche à dominante fonctionnelle à cette nouvelle approche objet.

Classiquement, les ingénieurs système avaient l'habitude de décrire une vision fonctionnelle du futur système sous forme de diagrammes de flux de données (DFD = Data Flow Diagram). Ces DFD représentaient ce que devait faire le système sous forme d'un réseau de fonctions transformant des flux (matière, énergie, information). Dans cette approche, la structuration du problème et la décomposition en sous-systèmes résultaient d'une démarche de décomposition structurée de la finalité en missions, des missions en fonction de services, de fonctions de service en sous-fonctions.

Par principe même de la représentation par objets (où les objets répondent à des sollicitations), la représentation fonctionnelle sous forme structurelle (où les fonctions transforment des flux) disparaît. L'objet est cependant trop élémentaire pour permettre une décomposition directe du problème en objets. La vision opérationnelle de départ consiste à explorer les cas d'utilisation du système sollicités par les acteurs (systèmes et humains) de l'environnement, et, pour chacun, à définir les scénarios opérationnels attendus. La vision architecturale de haut niveau est obtenue par regroupement et projection des cas d'utilisation (représentant l'aspect comportemental du système) sur les éléments de décomposition du système en sous-systèmes.

Au-delà de ce changement de paradigme, la principale difficulté consiste en fait à introduire les bases de la modélisation objet sans faire référence, comme c'est la pratique habituelle, à la programmation objet. Bref, faire de l'objet sans faire de l'informatique !

Comment présenter les concepts objets aux ingénieurs système ?

La plupart des présentations des concepts objets sont orientées système d'information avec des exemples souvent tirés du domaine bancaire : compte, client, etc.

Les ingénieurs système sont en général réfractaires à ces exemples pour plusieurs raisons :

- Ils ne sont pas représentatifs de leur métier ;
- Ils leur paraissent souvent trop simplistes (taille) par rapport à la complexité de leurs systèmes décomposés en sous-systèmes ;

- Ils passent souvent sous silence la complexité comportementale : les contraintes temps-réel, les états concurrents, etc.

Si l'on ne veut pas risquer un rejet d'emblée, il faut donc adapter le discours à la problématique de l'ingénierie système, et ne pas recourir à des analogies purement informatiques, orientées génération de code Java ou C++ !

Nous allons donner dans la suite de ce paragraphe un exemple de présentation des concepts objet dans le domaine spatial, en utilisant exclusivement des concepts qu'un ingénieur système de ce domaine manipule tous les jours.

Les planches qui suivent ont été simplifiées par rapport à la réalité et ne sont qu'un extrait d'une formation existante, mais donnent une bonne idée de la philosophie de la présentation des concepts.

Commençons par expliquer les concepts d'objet et de classe.

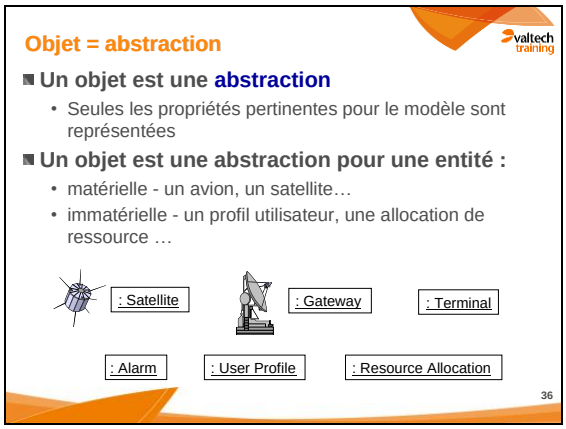

Figure 3. Exemples d'objets non informatiques

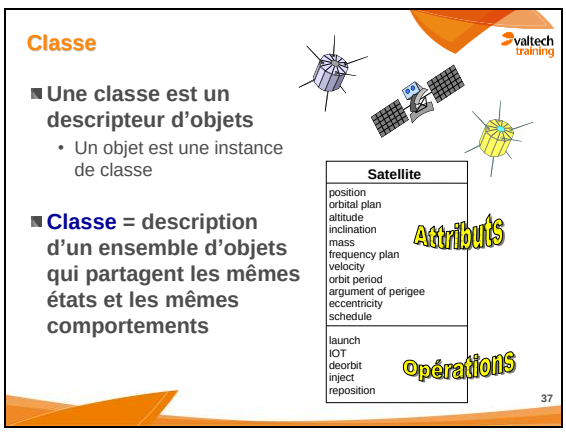

Figure 4. Exemple de classe : Satellite

Introduisons ensuite la notion de collaboration entre objets, en montrant que des messages peuvent circuler sur les liens entre objets. Le diagramme de communication (anciennement collaboration) est tout à fait adapté. Il s'agit d'un premier point de vue dynamique.

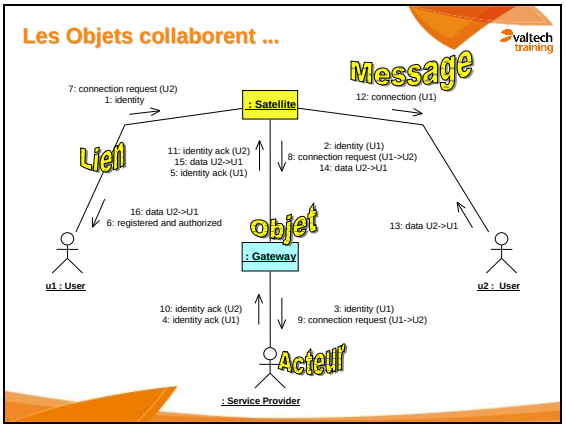

Figure 5. Exemple de collaboration entre objets non informatiques

Une autre forme de représentation graphique est donnée par le diagramme de séquence. Celui-ci est en général très bien accepté par les ingénieurs système.

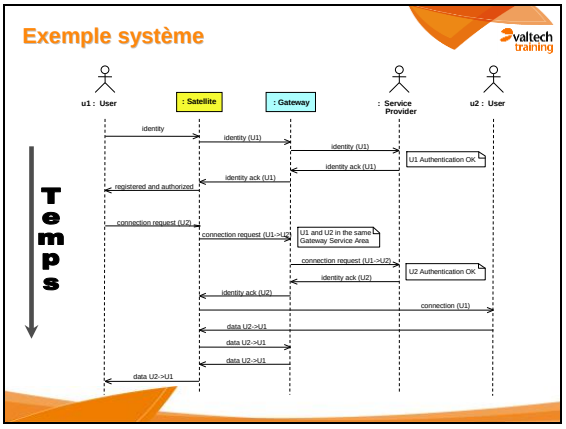

Figure 6. Exemple de diagramme de séquence

Plus délicat : comment expliquer l'héritage sans donner d'exemple de code Java ?
En fait, il faut simplement utiliser le concept bien connu de généralisation / spécialisation. Le satellite de notre projet n'est qu'un cas particulier de satellite à orbite basse (LEO = Low Orbit Satellite), qui est lui-même un cas particulier de satellite. On peut ainsi définir des super-classes par généralisation.

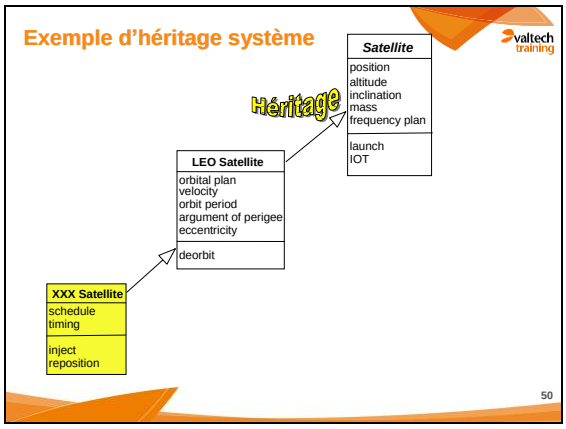

Figure 7. Exemple d'héritage : Satellite

On pourra ensuite introduire la classe satellite géostationnaire (GSO Satellite) comme alternative à LEO Satellite et ainsi capitaliser le savoir-faire des experts métier en distinguant les propriétés communes à tous les satellites de celles qui sont spécifiques aux sous-classes.

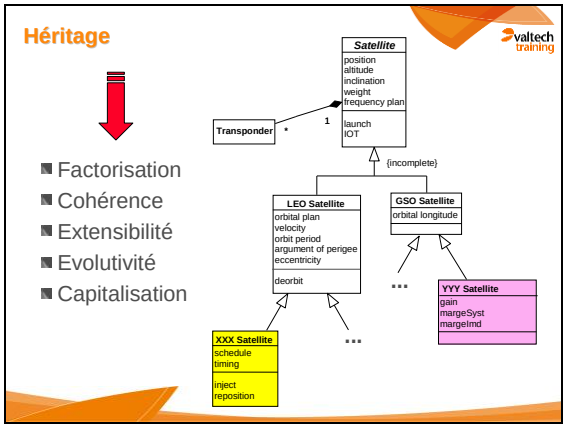

Figure 8. Exemple d'héritage plus complet

D'autres types de relations structurelles entre classes, telles que l'association et la composition, peuvent être expliquées de la même manière.

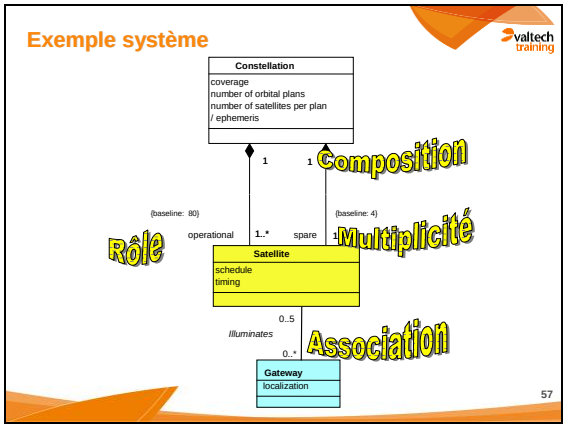

Figure 9. Exemples de relations structurelles entre classes

Les notions d'état, d'événement, de transition, bref de cycle de vie, sont en général familières aux ingénieurs système. Ils sont donc ravis de voir qu'UML propose un diagramme d'états particulièrement puissant dans sa panoplie de diagrammes.

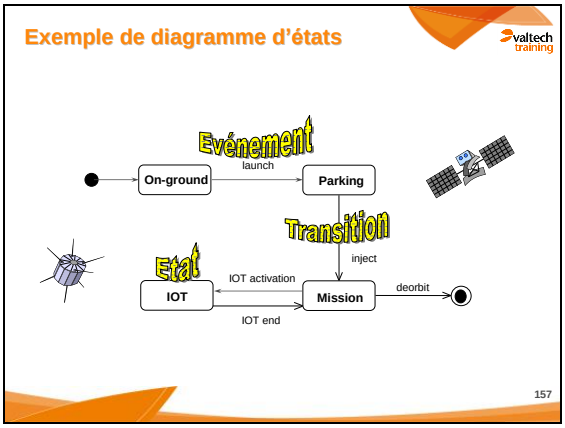

Figure 10. Exemple de diagramme d'états : Satellite

Pour terminer, il est bon de rappeler la complémentarité de tous ces diagrammes UML en insistant sur la notion de point de vue.

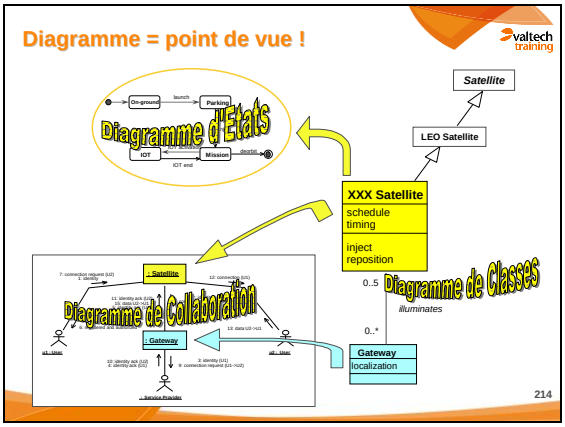

Figure 11. Chaque diagramme propose un point de vue complémentaire

Voilà un aperçu de la façon dont on peut présenter la modélisation objet sans faire aucune référence informatique, et encore moins à la programmation orientée objet !

SysML : un langage de modélisation pour l'ingénierie système

Comme nous venons de le voir, l'utilisation du langage de modélisation UML pour les besoins de l'ingénierie système est tout à fait envisageable. Encore faut-il faire l'effort de ne présenter que ce qui est pertinent pour la problématique système et avec le bon point de vue.

Nous ne pouvons pas nier que les 13 types de diagrammes proposés par UML 2.0 sont un peu intimidants pour les débutants en modélisation. D'où l'effort de normalisation en cours d'un langage de modélisation de système, SysML (www.sysml.org), qui, tenant compte des nouveautés apportées par la dernière version d'UML (2.0), prend en compte les besoins et approches de l'ingénierie système générale tout en en tentant de combiner au mieux la vision fonctionnelle et les apports de l'approche objet.

SysML se présente comme un profil UML 2, c'est-à-dire une extension d'UML (pour certains aspects non couverts en standard) mais également un sous-ensemble (en laissant de côté tout ce qui est trop orienté logiciel et génération de code).

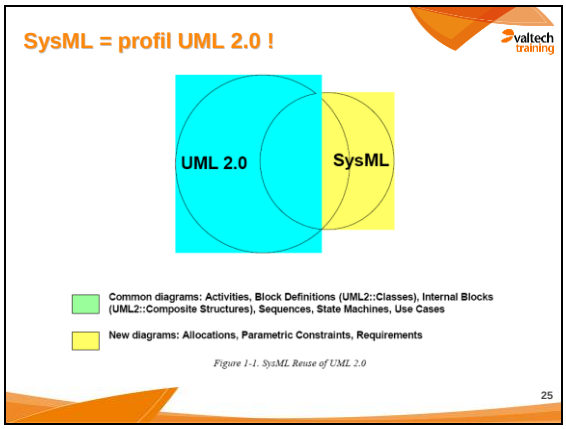

Figure 12. SysML en tant que profil UML

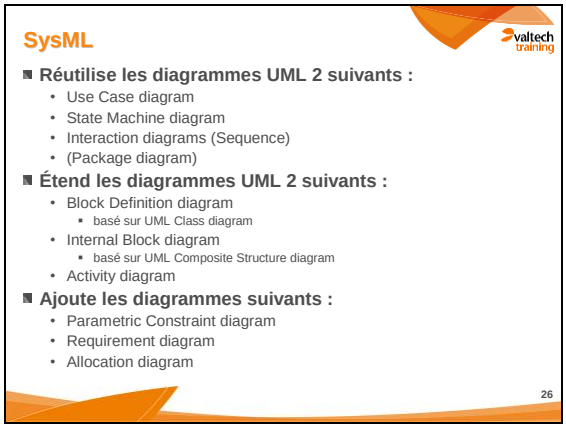

Figure 13. Détail des similarités et différences entre SysML et UML

Actuellement, deux spécifications concurrentes sont en compétition pour une adoption par l'OMG. La convergence rapide vers une spécification satisfaisante pour tous est cruciale pour que les éditeurs d'outils puissent s'engager plus avant dans des fonctionnalités plus exigeantes de simulation et de vérification formelle, condition indispensable pour une généralisation de ces techniques.

Figure 14. Principaux outils actuels

Conclusion

Pour conclure, nous pouvons maintenant affirmer qu'il est tout à fait possible d'enseigner la modélisation objet aux ingénieurs système habitués à la modélisation fonctionnelle.

Cependant plusieurs conditions sont requises :

- Une présentation adéquate des concepts objet et des diagrammes UML dans le contexte du domaine spécifique des ingénieurs système ;
- Aucune référence aux particularités du logiciel et encore moins de la programmation objet ;
- Une démonstration concrète que ça s'applique bien à des systèmes complexes (et pas seulement au distributeur de billets ou à l'éternel ascenseur …) ;
- Une coopération des apprenants : on ne peut rien faire si les ingénieurs système refusent a priori tout changement dans leurs habitudes !

REFERENCES

(Douglass 1999) : *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns*, B. Douglass, Addison-Wesley, 1999.

(Douglass 2004) : *Real Time UML : Advances in the UML for Real-Time Systems (3rd Edition)*, B. Douglass, Addison-Wesley, 2004.

(Roques 2005) : *UML 2 par la pratique*, Pascal Roques, Eyrolles, 2005.

(Rumbaugh 1999) : *The Unified Modeling Language Reference Manual,* J. Rumbaugh et al., Addison-Wesley, 1999.

(Selic 1994) : *Real-Time Object-Oriented Modeling,* B. Selic, Wiley, 1994.

BIOGRAPHIE

Pascal Roques a plus de dix-huit ans d'expérience dans la modélisation de systèmes complexes, d'abord avec les techniques d'analyse structurée (SADT), puis avec les approches objet (OMT, UP, UML).
Consultant senior et formateur dans le groupe Valtech depuis 1995, il a travaillé à promouvoir l'utilisation d'UML dans des domaines variés (aérospatiale, banques, etc.). Il est actuellement responsable de l'ensemble des formations catalogue Valtech Training sur le thème « Modélisation avec UML ». Pascal Roques est également l'auteur de plusieurs livres sur UML parus récemment chez Eyrolles.