“Model-driven engineering and model-based approaches have permeated all branches of software engineering to the point that it seems that we are using models, as Molière’s Monsieur Jourdain was using prose, without knowing it”.

Source « Modeling Modeling Modeling » Pierre Alain Muller et al. Sosym 2010, Springer

Model-Driven Engineering: Des Principes aux Concepts Avancés

Reda Bendraou
reda.bendraou{{@}}Lip6.fr
http://pagesperso-systeme.lip6.fr/Reda.Bendraou/

Le contenu de ce support de cours a été influencé par les lectures citées à la fin de ce support.

L'importance du Génie Logiciel

Le logiciel contrôle le monde!

- Processus métiers (administrative etc.)
- Gouvernement
- L'industrie (Usines, chaines de fabrication)
- Transports
- Défense, finance, santé...
- Edition, médias...
- Les nouvelles technologies du web, commerce éléctronique...
- Et bien plus!!

Des enjeux aussi bien économiques que politiques

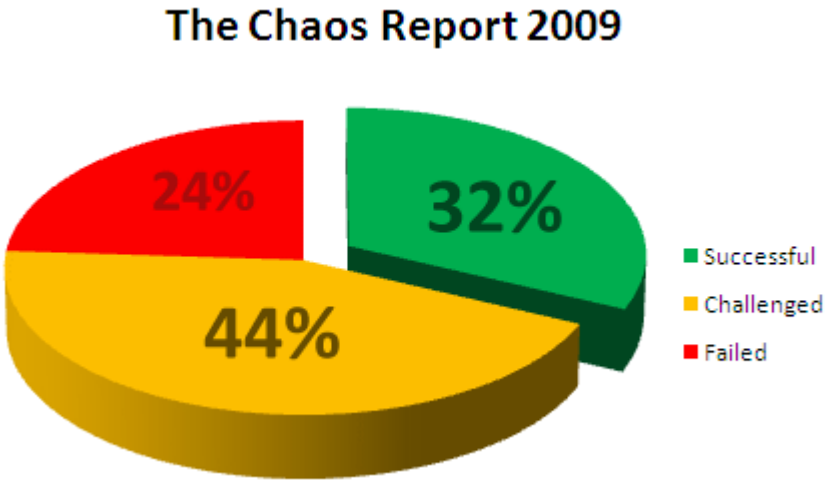
Pourquoi c'est si important?

Les conséquences en cas de problèmes peuvent être très lourdes!

- Therac 25, '85‘ 87 : 6 patients irradies, 2 morts
- Syst. Bagages Aeroport Denver, '95 : 16mois, 3.2 Mds$
- Ariane 5 vol 88/501, '96: 40s de vol, destr., 850 M$
- Mars Climate Orbiter & Mars Polar Lander, '99 : destr.
- Bourse de Londres (projet d'informatisation) 4 ans, 100 M.L, abandonné

Constat des Projets GL en quelques chiffres!

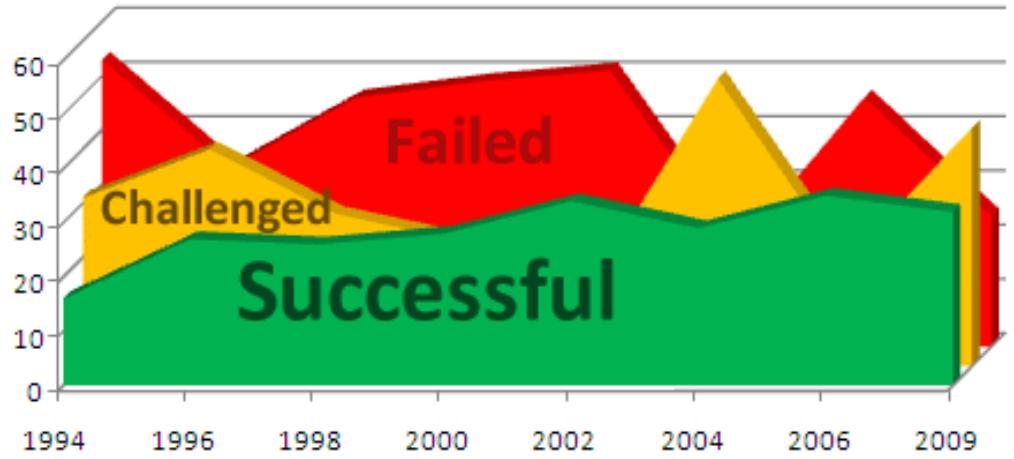
Source: The Standish Group



- Successful means on-time, on-budget, and with all features and functions as defined in the initial scope;
- challenged means late, over budget, and/or with less features and functions than defined in the initial scope;
- failed means cancelled prior to completion, or delivered but never used.

Quelques chiffres!

- Le taux de succès en constante hausse, mais le taux d'échec reste aléatoire



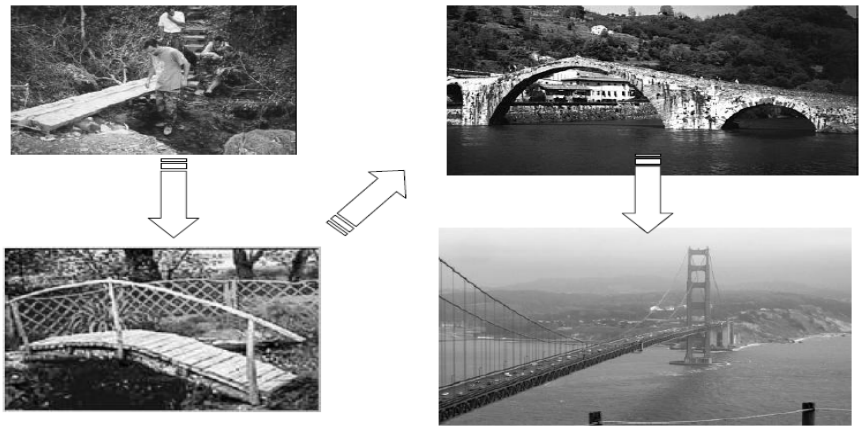
Le talon d'Achille du GL

- Le coût
- Le temps
- La qualité

Face à cela, de nombreux défis auxquels devra faire face le GL d'aujourd'hui (diapos suivantes!)

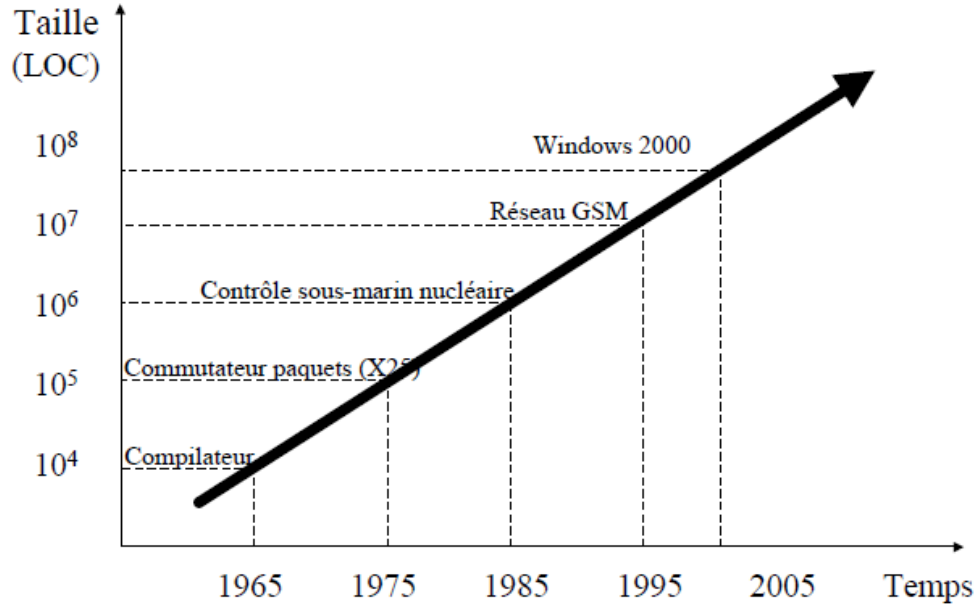
Défis et obstacles du logiciel d'aujourd'hui

- Composants de plus en plus complexes/distribués
  - Plusieurs préoccupations/points de vues/aspects
    - Métier, Sécurité, Communication, Passage à l'échelle
  - Ces points de vues sont noyés dans le code
    - Impossibilité de raisonner dessus, de les isoler!
  - Problème de la validation intra et inter-composants
    - Pas ou peu de preuves sur la fiabilité du code des composants



Défis et obstacles du logiciel d'aujourd'hui

- Applications de plus en plus larges
  - Le nombre de lignes de code ne cesse d'augmenter
  - Le Code pour communiquer? Pas Facile!!!!
    - Windows XP: 40 Millions de LoC, Mac OS 10.4: 86Millions LoC, Linux Kernel 2.6.32: 12 Millions LoC (source: Wiki, Source Line of Code)



Défis et obstacles du logiciel d'aujourd'hui

- Des besoins qui évoluent en cours de route
  - Problème de maintenance évolutive et corrective
  - Nokia rapporte à propos de son infrastructure GSM
    - 50% des exigences (besoins numérotés) ont changé *après* le gel du cahier des charges
    - 60% de ceux-ci ont changé au moins 2 fois!
    - C'est le cas général plutôt que l'exception
    - Cahier des charges figé = rêve des années 70

      ⇒Besoin de Réactivité!

  Des besoins de variabilité

Défis et obstacles du logiciel d'aujourd'hui

- La mentalité des informaticiens ne change pas
  - Code is King
  - Ingénierie dirigée par le code
  - Pas de documentation
  - Peu de modélisation

Défis et obstacles du logiciel d'aujourd'hui

Techno, Techno, Techno

- beaucoup de méthodes, de langages
- évolution des outils de développement,...
- effets de mode, nouveaux paradigmes, visions : objets, composants (middlewares), modèles (MDA), services (SOA), Cloud Computing, etc.

Que faire pour contrer l 'évolution permanente des technologies?

- Standardisation?
  - Cycles ISO trop long! Cycles OMG trop rapides
  - Les meilleures solutions sont bien protégées/cachées
- Migration, Formation, investissements?
  - Trop coûteux!
- Concevoir d'une manière plus anticipative?
  - Manque de savoir faire

Quelles solutions* aujourd'hui pour ces défis?

- Ingénierie Dirigée par les Modèles (IDM ou MDE)
  - (4 cours)
- Aspect Oriented Programming/Modeling AO(P/M)
  - (1 cours)
- Software Product Lines (Lignes de Produit)
  - (2 cours)
- Design Patterns
- Autres??

* Solutions au niveau applicatif (soft, méthode, visions) et non pas infrastructure (hard)

Modalités du cours

- 7 séances de cours (2h00, en théorie ☺), présence, à votre avis?
- 7 séances de TP (2h), présence obligatoire!
- 1 Projet comme fil conducteur pour appliquer ce que vous aurez appris en Cours/TPs
- 1 Examen
- Note Finale= Note Examen *60% + Note CC*40%
  - » CC à définir après une semaine ou deux d'observation

IDM: Ingénierie Dirigée par les Modèles
ou
MDE: Model-Driven Engineering

-Principes et Définitions
-Promesses & Enjeux
- MDA et ses standards
- Notion de DSMLs
- Outils: EMF

MDE: Définition

- **Model-Driven Engineering (MDE)** is a software development methodology/vision which focuses on creating models, or abstractions, more close to some particular domain concepts rather than computing (or algorithmic) concepts.

- A modeling paradigm for MDE is considered effective if its models make sense from the point of view of the user and can serve as a basis for implementing systems.

Les Principes de l'IDM

- P1: "Models as first class entities"
- P2: Montée en abstraction
- P3: Séparation des préoccupations (separation of concerns)
- P4: Modèle productifs (Vs. Contemplatifs)

P1: “Models as first class entities”

- Passage du « tout objet » au tout « modèle »
  - Modéliser au lieu de coder
  - De ~~Write Once, Run Anywhere~~ à Model Once, Generate Anywhere
- Le modèle objet a atteint ses limites
  - Les concepts objets sont plus proches de l’implémentation/technologie que du domaine métier
  - « Objects have failed » by Richard. P. Gabriel, Openning OOPSLA’02
    - Notes: http://www.dreamsongs.com/ObjectsHaveFailedNarrative.html
  - Difficile de communiquer, raisonner avec du code.
    - aspects fonctionnels/non-fonctionnels noyés dans les lignes de code
- Défis: expertise métier, un bon langage de modélisation!!

Modèle: Définition

A Model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality

Modeling, in the broadest sense, is the cost-effective use of something in place of something else for some cognitive purpose. It allows us to use something that is simpler, safer or cheaper than reality instead of reality for some purpose

Par Jeff Rothenberg, « The nature of modeling »

- Attention au débat: abstraction = simplification?
  - La modélisation simplifie la compréhension et la communication autour du problème, elle ne simplifie pas le problème lui même!
- Attention, un modèle n'est pas forcément graphique

Modèle: Définition (pour les matheux!)

Un modèle est un graphe

- **Definition 1**. *A directed multigraph* $G = (N_G, E_G, \Gamma_G)$ consists of a set of distinct nodes N_G, a set of edges E_G and a mapping function $\Gamma_G : E_G \rightarrow N_G \times N_G$

- **Definition 2**. *A model* $M = (G, \omega, \mu)$ is a triple where:
  - ✓ $G = (N_G, E_G, \Gamma_G)$ is a directed multigraph
  - ✓ ω is itself a model, called the reference model of M, associated to a graph $G_\omega = (N_\omega, E_\omega, \Gamma_\omega)$
  - ✓ $\mu: N_G \cup E_G \rightarrow N_\omega$ is a function associating elements (nodes and edges) of G to nodes of G_ω (metaElements)

Modèle: Exemple

- La pipe selon Magritte

Modèle: Exemple

représente

Système **Modèle**

Méta-Modèle: Définition

- Pour être productifs, les modèles doivent être utilisés par des machines

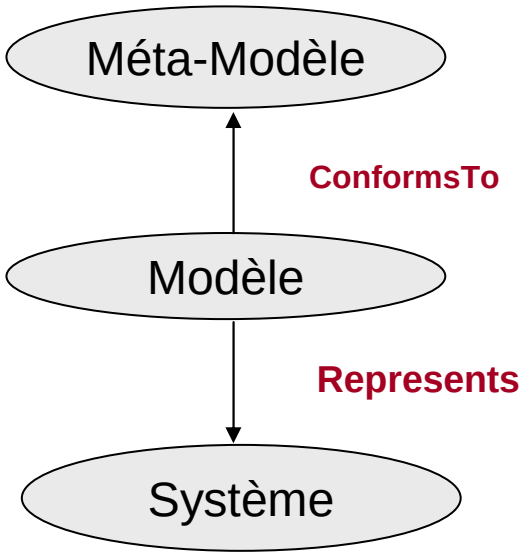
  => Besoin d'un langage précis pour les définir = **Méta-Modèle**

- A "*metamodel is a model that defines the language for expressing a model*" [OMG].

- "metamodel makes statements about what can be expressed in the valid models of a certain modeling language" [Seidewitz].

Méta-Modèle: Définition

- The relation between a model and its metamodel is a "*Conforms To*" relation and a model conforms to its metamodel if the elements and relationships between these elements are defined in the metamodel.



Méta-Modèle: Exemple

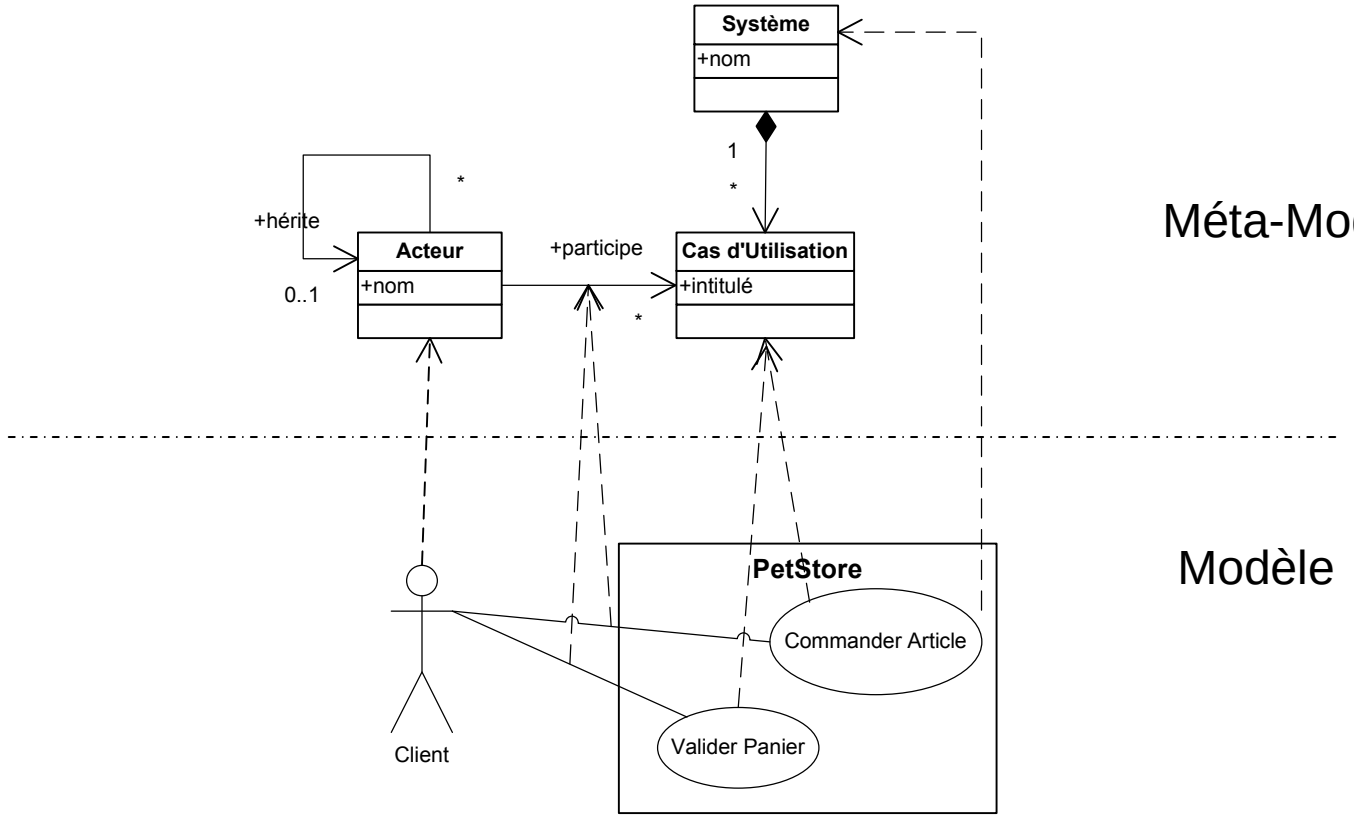


Fig. from X.Blanc

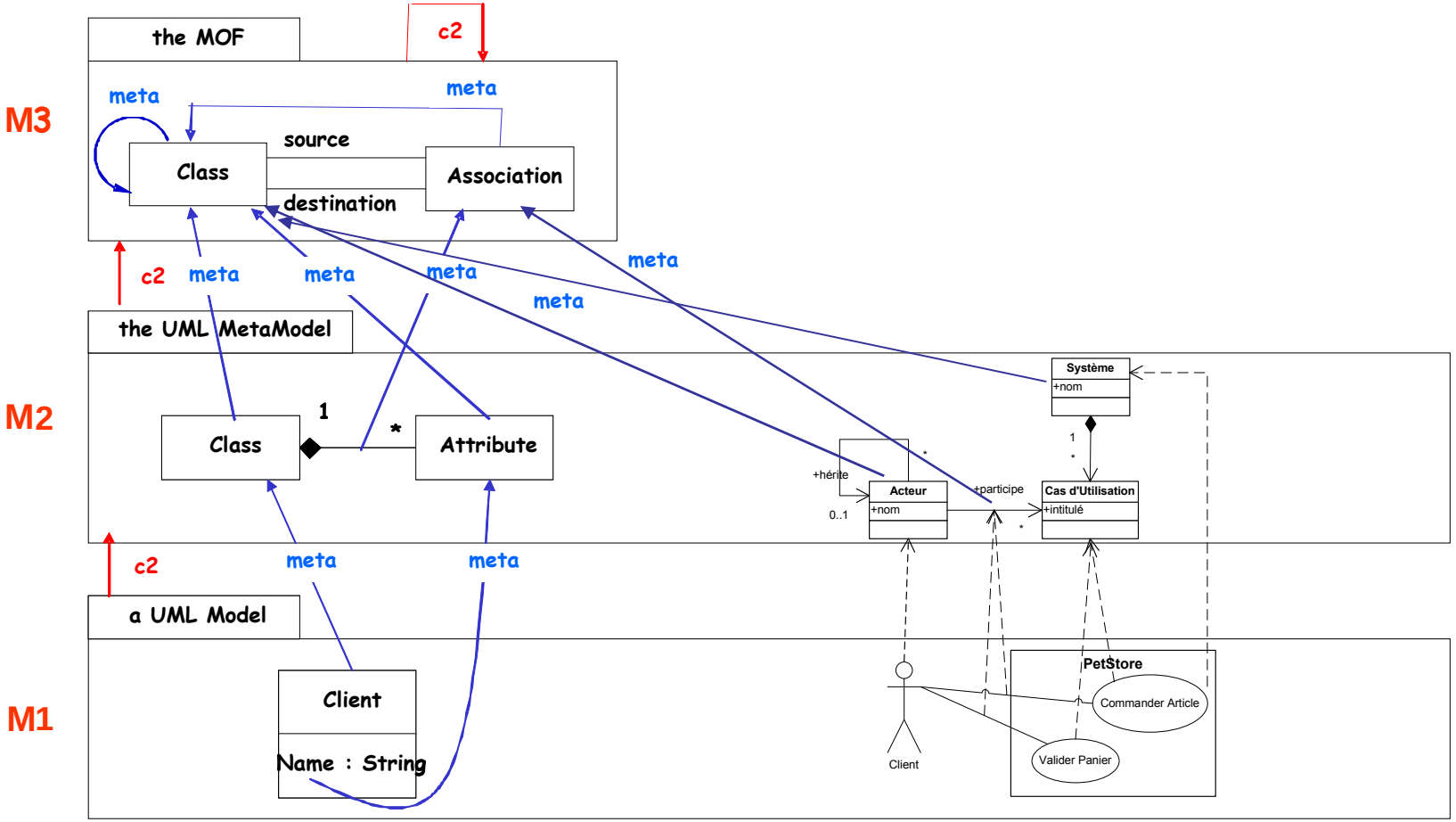
Méta Méta-Modèle: Définition

- A "*metametamodel is a model that defines the language for building metamodels*" [OMG].

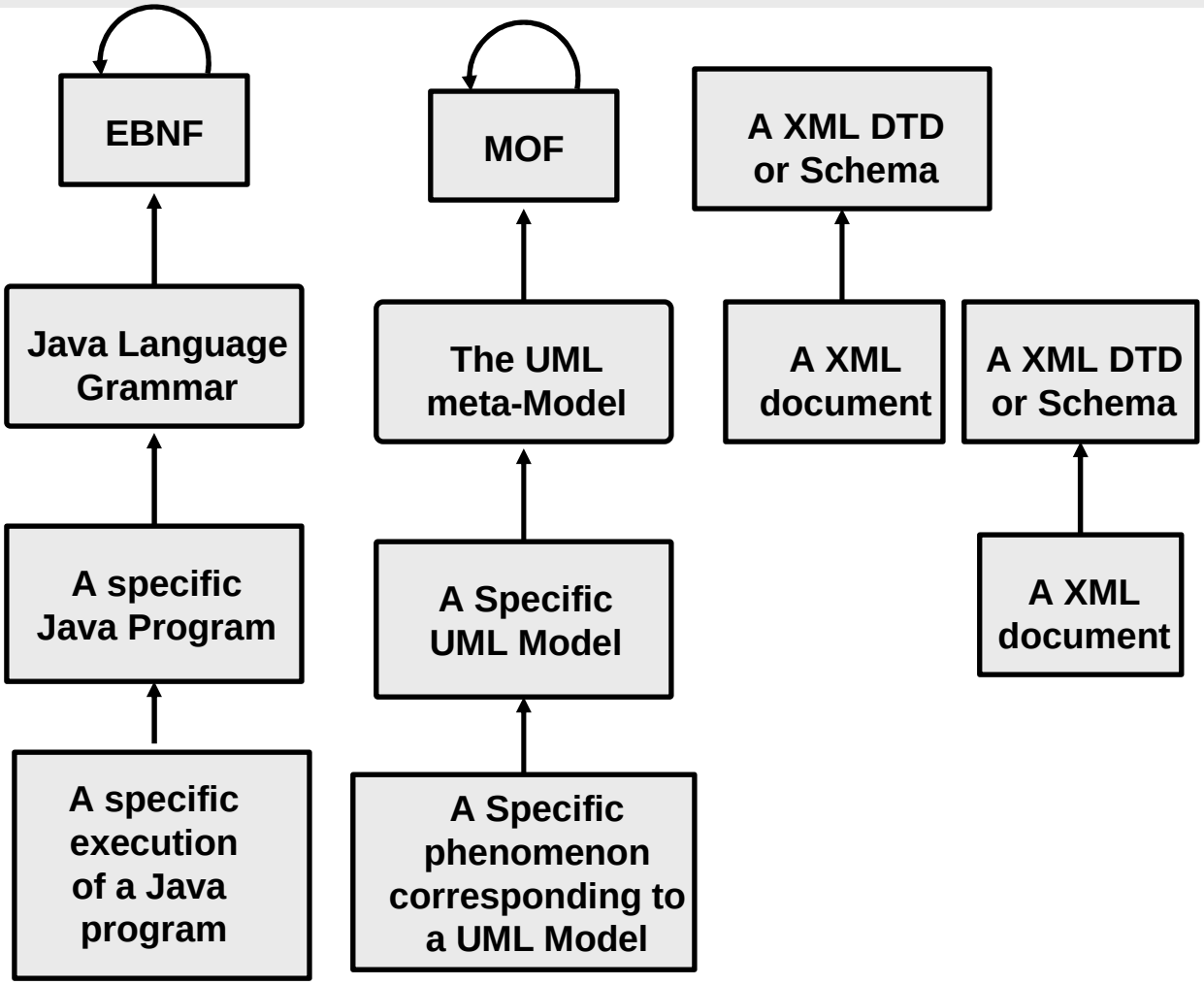
- À l'OMG, MOF représente le langage pour la spécification de nouveaux méta-modèles (langages). Exp. UML, SPEM, OCL...

- Le MOF est conforme à lui-même, auto descriptif.

Méta Méta-Modèle: Exemple



Méta Méta-Modèle: autres systèmes



P2: Abstraction

Abstraction =

- Ignorer les détails insignifiants
- Ressortir des détails les plus importants
  - Important= décider de ce qui est signifiant et de ce qui ne l'est pas, dépend directement de l'utilisation visée du modèle

Abstraction: « Opération intellectuelle qui consiste à isoler par la pensée l'un des caractères de quelque chose et à le considérer indépendamment des autres caractères de l'objet. »

Source Larousse en ligne.

Les modèles permettent de s'abstraire de certains détails qui ne sont pas indispensable pour comprendre le système selon un point de vue donné

Abstraction

- Histoire du GL: l'amélioration de la productivité s'est toujours faite en augmentant le niveau d'abstraction

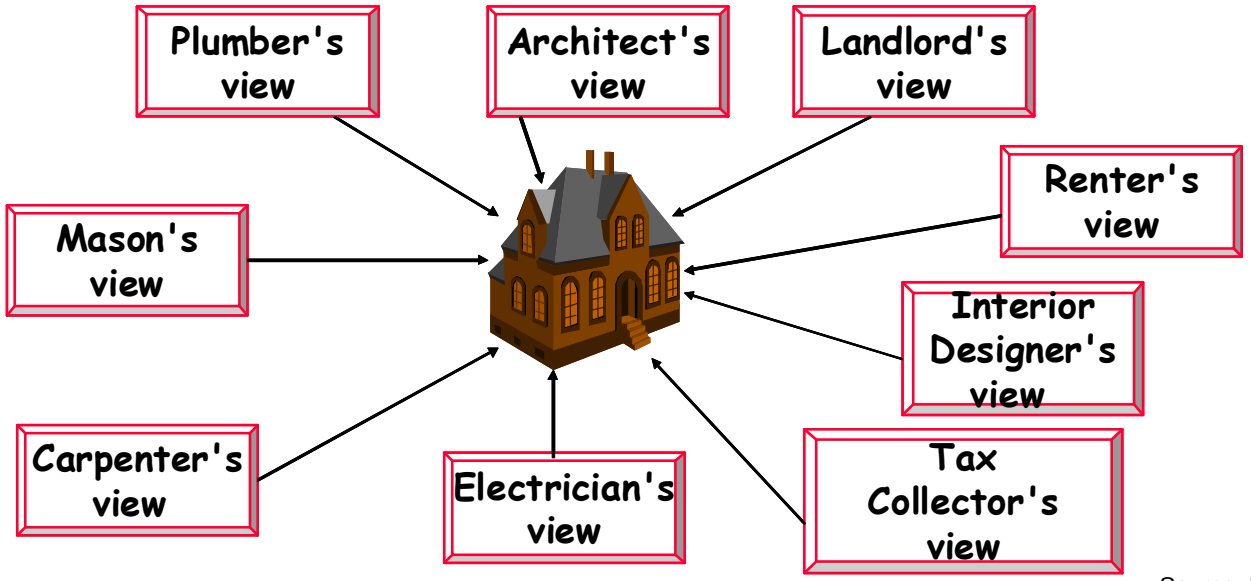
- Selon Caper's Jones Software Productivity Research (SPR 2006), les langages de 3ème G ont amélioré la productivité du développeur par 450% (C, Fortran)
  - L'introduction des langages OO n'ont pas fait autant!
    - Java 20% plus productif que le Basic

- Les langages de prog. aujourd'hui arrivent à leur limite en terme d'abstraction

- Défis:
  - Préoccupation Majeure: Assurer la transformation/transition vers le niveau le plus bas (c'est votre mission!!!)
  - Choisir le bon niveau d'abstraction

P3: Séparation des préoccupations

- Indispensable dans le cas d'applications complexes!
  - Plusieurs aspects/points de vues, plusieurs métiers!
    - Exp. Sécurité, Communication, GUI, QoS
  - Un point de vue=> un modèle



Source: Bézivin

Séparation des préoccupations: Une Vue, un Langage

- Chaque vue peut être exprimée en utilisant un langage de modélisation différent

- Plusieurs intervenants dans la boucle

- Défis:
  - Intégration des vues
  - Assurer la cohérence entre les vues

Abstraction, séparation des préoccupations
Exemple: Google Maps

- Différents niveaux d'abstractions
- Différents points de vues

P4: Modèles Productifs

- Modéliser reste un investissent qu'il faudra rentabiliser

- Vers de plus en plus de code généré automatiquement à partir des modèles
  - 100% dans certains domaines: web, scripts de config., réseaux
  - Plus difficile dans le cas d'applications complexes (nous verrons pourquoi plus loin)

- Beaucoup d'expertise dans les générateurs de code
  - Moins d'erreurs dans le code généré
  - gains en temps et en qualité

Modèles Productifs

- Défis:
  - Des langages de modélisation expressifs (proche du métier) et précis
  - Des générateurs de code fiables
  - Intégration du code généré à partir de plusieurs vues vers un seul système!
  - Génération du comportement?
  - Définition de la chaine de transformation (raffinement) modèles vers code
    - Compliqué dans le cas où plusieurs langages de modélisation sont utilisés en même temps pour modélisé le même système

MDE: Promesses

Gérer la complexité

- Des applications de plus en plus énormes
  - Windows 2000: ~40 millions de lignes de code
  - Impossible à gérer par un seul programmeur

- Exemple: Google
  - 400 000 Serveurs, une capacité de calcul gigantesque
  - 1 milliard de requêtes par jour
    - Chacune interrogeant 8 milliards de pages web en moins d'1/5 de S.

MDE: Promesses

Meilleure communication

- Le code n'est pas toujours compréhensible par les développeurs qui ne l'ont pas écrit
  - On peut difficilement communiquer avec du code, pas assez abstrait!

- Dans de gros projets souvent :
  - Des centaines de personnes
  - un peu partout dans le monde

- L'apparition de nouvelles façons de travailler
  - L'outsourcing, sous-traitance, etc.

MDE: Promesses

Pérenniser un savoir-faire

- Certains projets peuvent durer des années
  - Pas toujours les mêmes personnes qui travaillent sur le projet
  - Besoin de capitaliser un savoir faire indépendant du code et des technos
  - Capturer le métier sans se soucier des détails techniques

- Exemples de projets:
  - Contrôleur aérien (Thalès): Projet ~8 ans, durée de vie 40 ans
  - La construction d'un avion (Airbus): Projet ~10 ans, durée de vie 50 ans

MDE: Promesses

Augmenter la productivité

- Génération de code à partir des modèles

- Maîtrise de la variabilité
  - La vision ligne de produit
    - Un modèle générique pour un produit, plusieurs variantes

- Exemple: Nokia
  - 1,1 milliard de téléphones portables (100 millions de plus par an)
  - Des milliers de versions de logiciels
  - Time-to-market ~3 mois

Mais quel langage utiliser pour modéliser, méta-modéliser (créer de nouveaux langages)?

Plusieurs approches existent

- L'approche MDA: utilisation d'un ensemble de standards fournis par l'OMG i.e. MOF, UML
  - UML pour la modélisation d'applications OO
  - Définir un nouveau langage si UML ne suffit pas
    - Soit from scratch en utilisant MOF
    - Soit par extension d'un langage existant exp. UML, SPEM
    - Soit à l'aide du mécanisme de profile
- L'approche DSML (Domain-Specific Modeling Languages):
  - Des solutions propriétaires, exp. EMF, KM3, Kermeta, MetaEdit+
  - Propres à des domaines, au métier de l'entreprise
  - Objectif principal: se rapprocher du domaine métier, générer 100% du code

L'approche MDA

-Vision

- Architecture

- Standards

MDA: Model-Driven Architecture

- Une vision promue par l'OMG depuis 2001
  - (à lire, MDA Guide by Richard Soley sur www.omg.org)

- Une architecture

- Une application du MDE qui s'articule autour des standards OMG

MDA: Vision

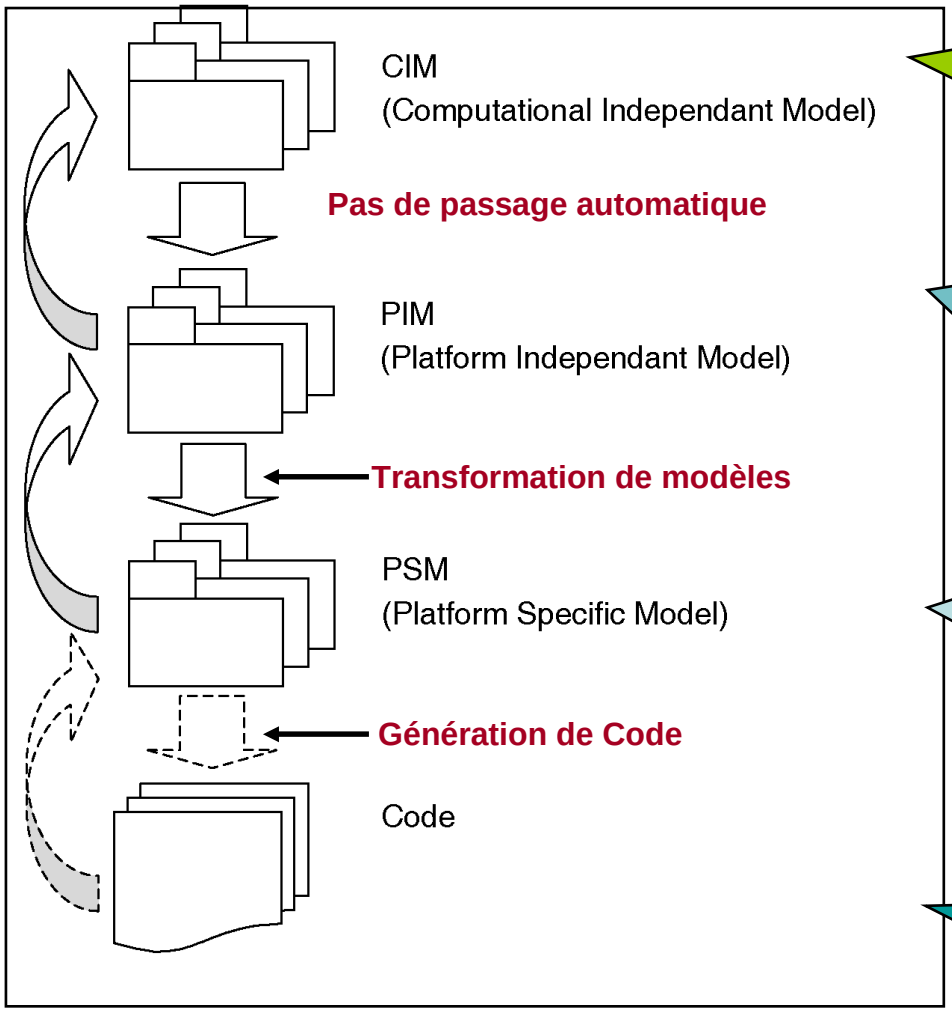
Un nouveau problème

Obsolescence et rotation rapide des plates-formes technologies
et besoin de pérennité des applications métier.

Nous ne voulons plus payer le prix fort uniquement pour porter notre système informatique vers une nouvelle plate-forme de middleware (COM, CORBA, Java, HTML, XML, DotNet, Cluster, Grid, etc.) alors que notre modèle métier reste stable. D'autant plus que nous avons déjà donné pour ce type de migration sans aucun retour sur investissement ($$$!!!).
Tout ce que nous pouvons accepter c'est de payer une dernière fois pour la construction de modèles abstraits de notre métier et des services associés, modèles qui nous garantiront contre l'obsolescence technologique des plate-formes.
À partir de ce moment, tout nouveau fournisseur de plate-forme, s'il désire nous voir acheter sa solution, sera prié de nous livrer en même temps que sa plate-forme les outils de transformation permettant de générer vers cette plate-forme à partir des modèles neutres de métier et de service.

From MDDays 09, J. Bézivin

MDA: Vision



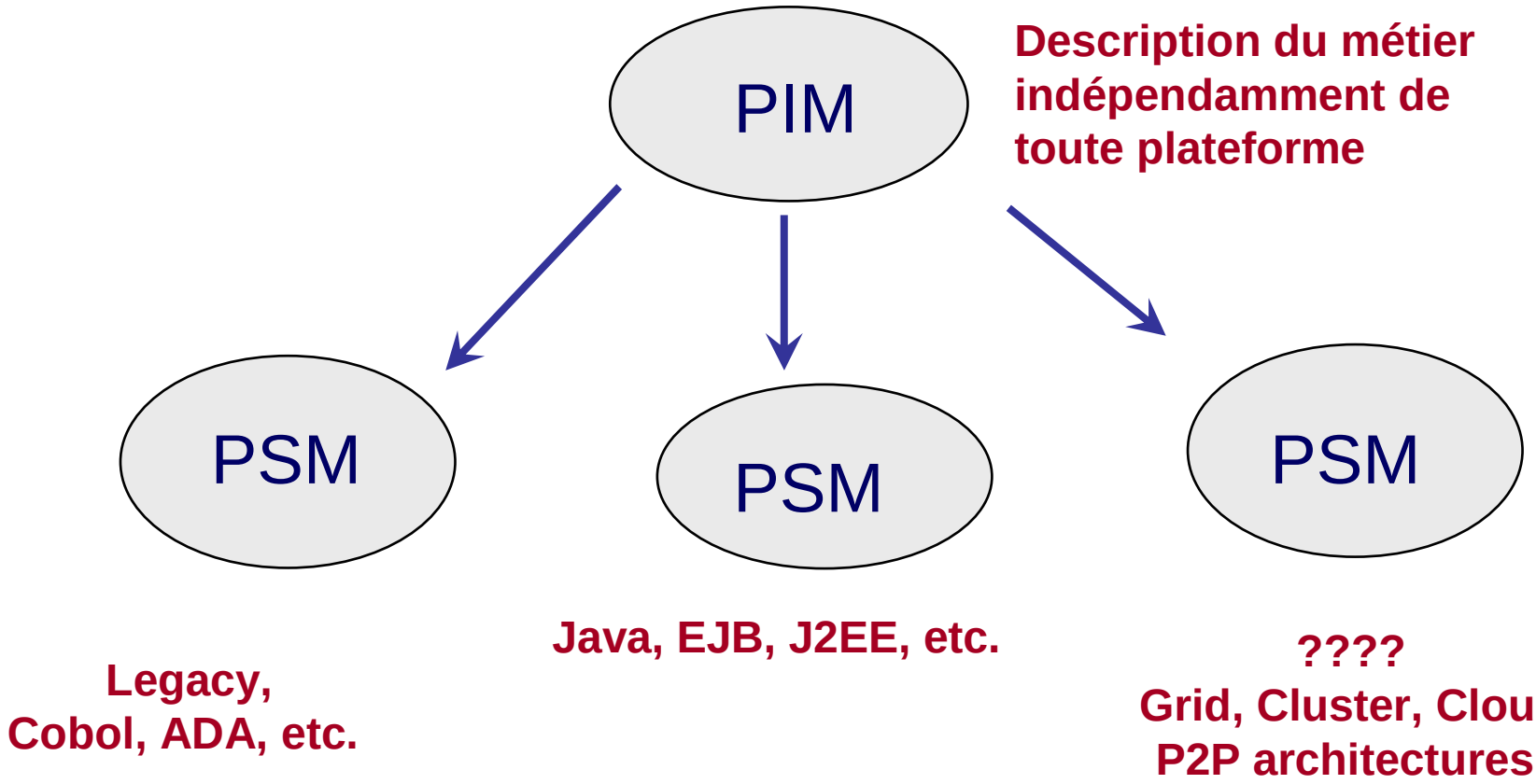
Focuses on the on the environment of the system, and the requirements for the system; the details of the structure and processing of the system are hidden

Focuses on the operation of a system while hiding the details necessary for a particular platform. A PIM shows that part of the complete specification that does not change from one platform to another

Combines the PIM with an additional focus on the detail of the use of a specific platform by a system

Code

MDA: Vision


PIM
Description du métier indépendamment de toute plateforme
PSM
PSM
PSM
Legacy,
Cobol, ADA, etc.
Java, EJB, J2EE, etc.
????
Grid, Cluster, Cloud
P2P architectures

MDA: Vision

- Les modèles CIM décrivent les exigences sur le logiciel
  - Services rendus, Environnement d'exécution, procédés métiers exploitant

- Les modèles PIM décrivent la conception abstraite du logiciel
  - Abstraction par la définition d'une machine conceptuelle

- Les modèles PSM décrivent l'exploitation d'une plate-forme
  - Choix d'une machine d'exécution

Abstraction dans le MDA

- Le CIM est-il une abstraction du PIM ?
  - Est-ce que les exigences sont une abstraction de la conception abstraite ?
    - Oui si l'on suit le principe du raffinement
    - Non dans les autres cas

- Le PIM est-il une abstraction du PSM ?
  - Est-ce que la conception abstraite est une abstraction de la conception concrète ?
    - Oui

Recouvrement dans le MDA

- Le CIM et le PIM se recouvrent-ils ?
  - Les exigences du CIM doivent être supportées par la conception

- Le PIM et le PSM se recouvrent-ils ?
  - La conception concrète doit être conforme à la conception abstraite

Transformation de modèles: Brique de base de l'MDA

- Permet le passage PIM => PSM, PIMa=>PIMb, etc.

- Permet le raffinement PIMa=>PIMà, PSM_1=>PSM_2=>PSM_n, etc.
  - Intégration d'un aspect métier, d'un design pattern

- Différents types de transformations, différentes techniques

Transformation de modèles: Brique de base du MDA

- Une transformation est une opération qui
  - Prend un (ou plusieurs) modèle source en entrée typé par son Méta-Modèle
  - Fournit un (ou plusieurs) modèle cible en sortie typé par son Méta-Modèle

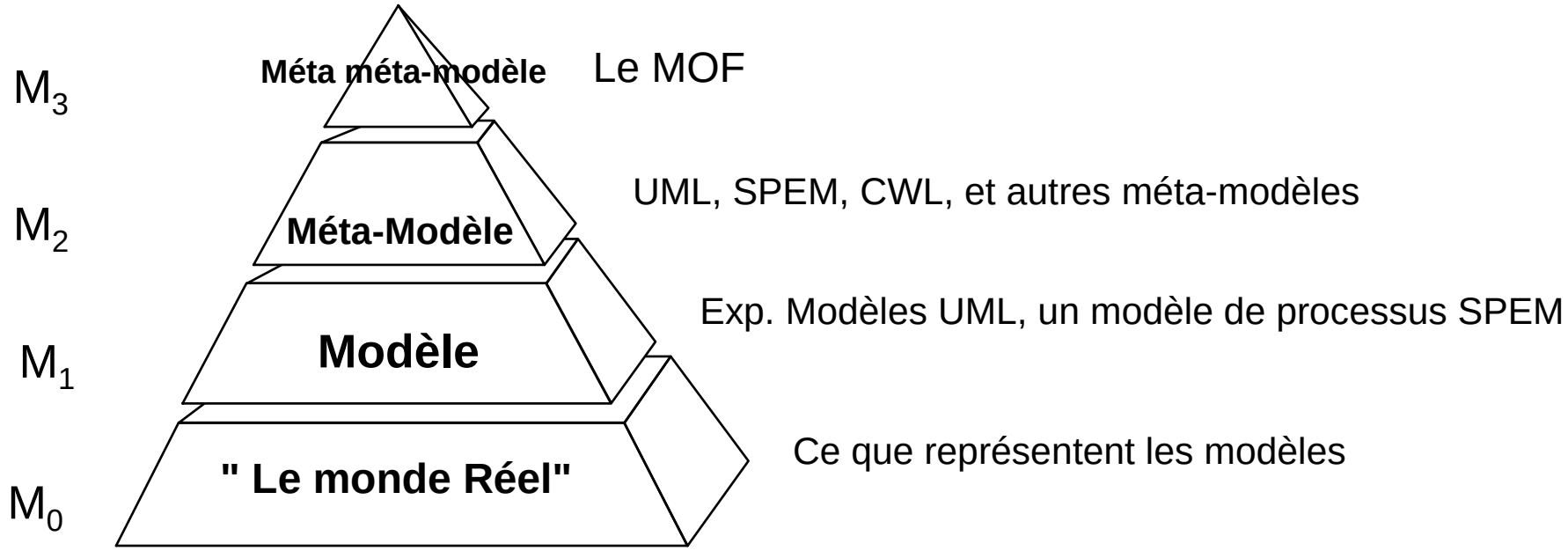
- Transformation endogène
  - Les modèles source et cible sont conformes au même métamodèle
  - Transformation d'un modèle UML en un autre modèle UML

- Transformation exogène
  - Les modèles source et cible sont conformes à des méta-modèles différents
  - Transformation d'un modèle UML en programme Java
  - Transformation d'un fichier XML en schéma de BDD

- Nous verrons ça plus en détails (cours)

MDA: Une Architecture

- Une architecture à 4 niveaux, un vocabulaire i.e. M0, M1, etc.

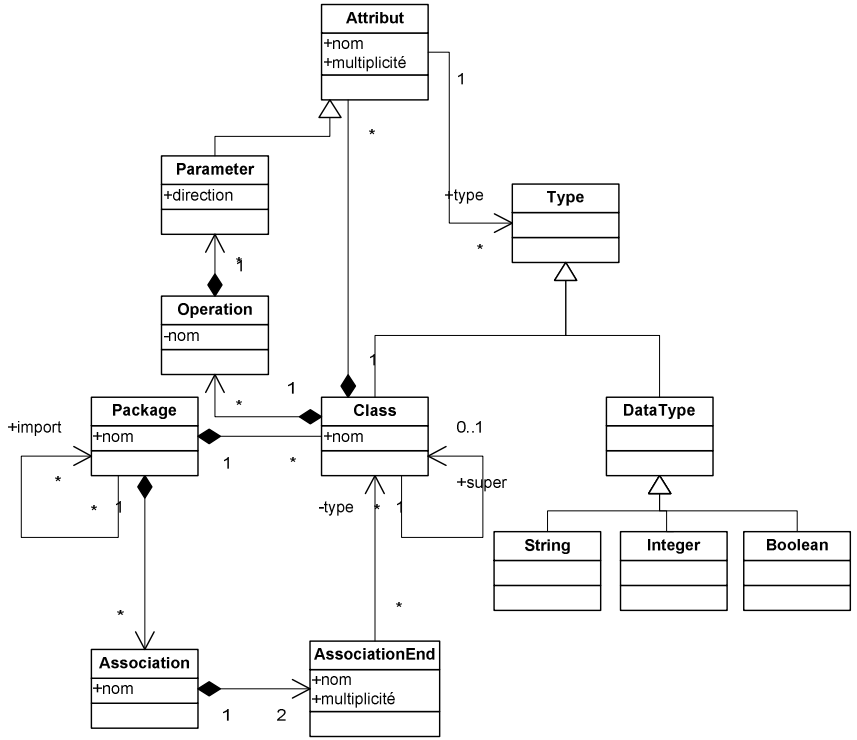


MDA: Architecture

- M0 (aucun 'M'): le monde réel peuplé d'individus
- M1 (1 'M' pour Modèle): l'univers du discours peuplé d'éléments de modélisation
- M2 (2 'M' pour Méta-Modèle): l'univers de description des structures des modèles peuplé de méta-éléments
- M3 (3 'M' pour Méta-Méta-Modèle): l'univers de description des description des structures des modèles peuplé d'un ensemble fini de méta-méta-éléments

MDA: Des Standards

Le standard MOF



- Le MOF définit le langage permettant de définir des méta-modèles
- MOF peut être défini à l'aide d'un diagramme de classe. Ce diagramme est le méta-méta-modèle
- Le méta-méta-modèle s'auto-définit.

Le standard MOF: Concepts

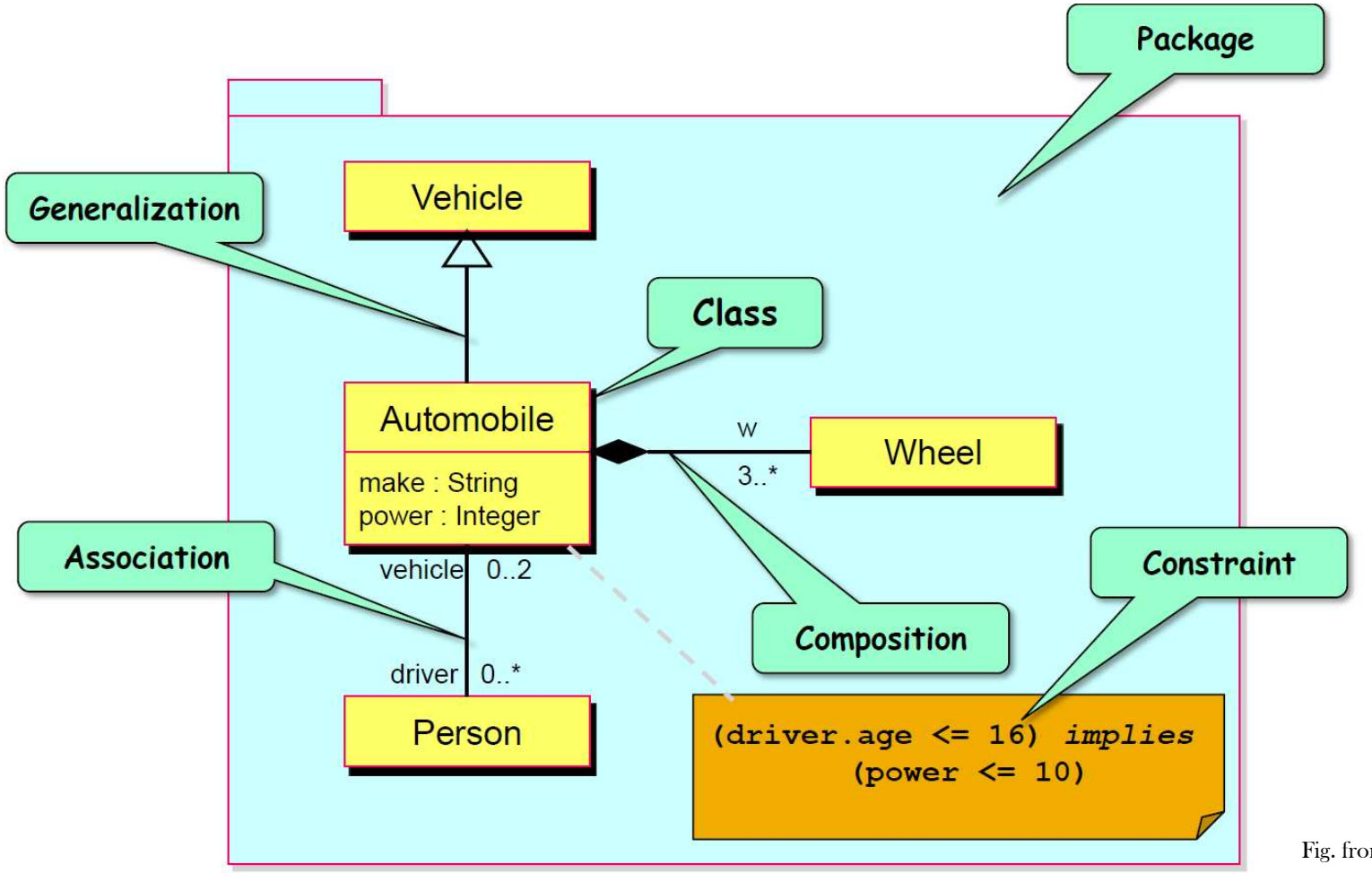


Fig. from B.Selic

UML2.0

- UML est actuellement le méta modèle le plus important (visible) de l'approche MDA.

- C'est le méta-modèle qui définit la structuration des modèles des applications OO
  - UML2.0 est un méta-modèle instance de MOF2.0.
  - La sémantique de UML2.0 est définie à l'aide de langage naturel
  - Les diagrammes UML2.0 sont définis à partir du méta-modèle

- UML2.0 supporte différents niveaux d'abstractions et différents points de vue
  - Cas d'utilisation, Séquences, Structuration Interne, Etats, Déploiement, etc.

UML dans MDA

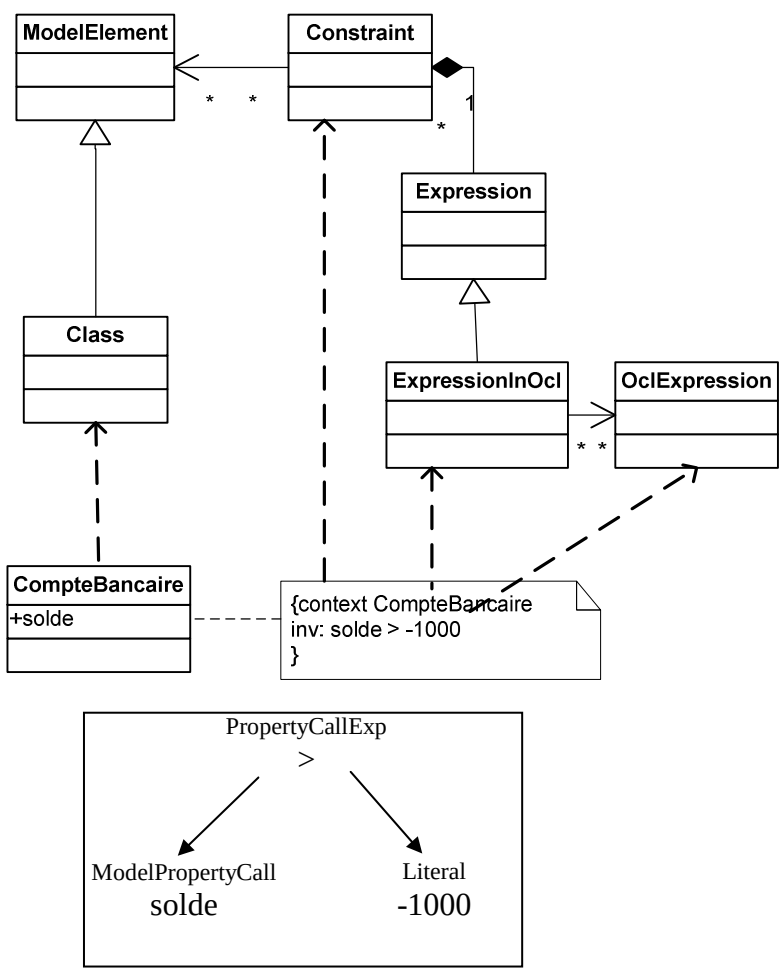
- UML2.0 est le méta-modèle naturel pour les PIM (Platform Independant Model)

- UML2.0 permet aussi de représenter une application dans son environnement afin d'en faire ressortir les exigences (cas d'utilisation)
  - UML2.0 peut être utilisé pour les CIM (Computational Independant Model)

- UML2.0 peut être profilé afin de cibler des plates-formes d'exécution (ex: profil pour CORBA, profil pour EJB)
  - UML2.0 peut être utilisé pour les PSM (Platform Specific Model)

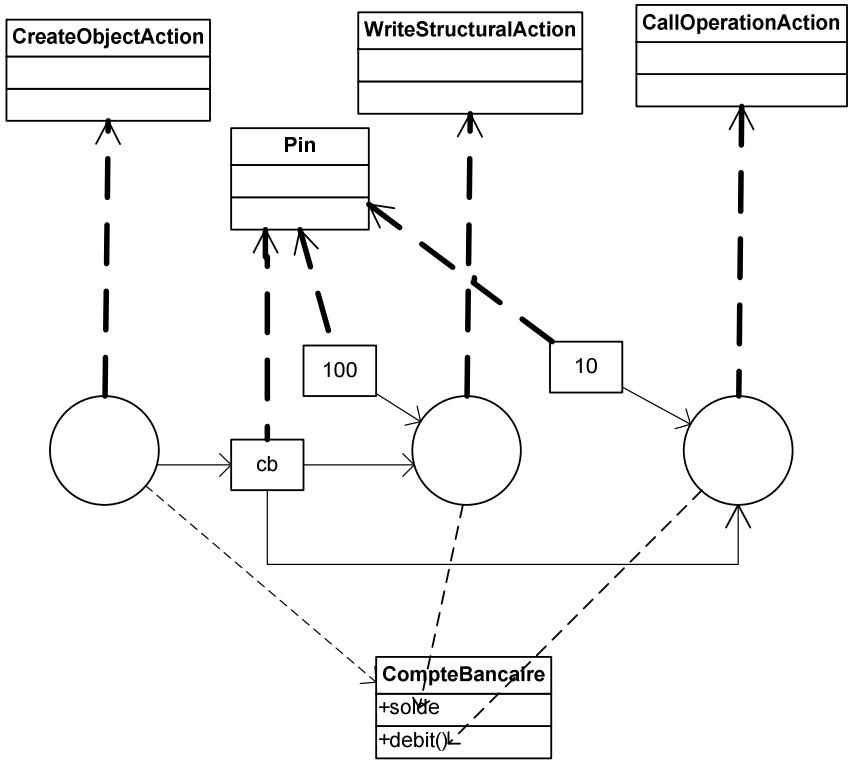
- Il serait donc possible d'appliquer MDA en utilisant uniquement UML
  - Nous verrons quelques limites d'UML dans la partie DSML

OCL: Object Constraint Language



- OCL définit la structuration des modèles représentant des contraintes sur les applications informatiques
  - Invariant, Pré-post conditions
- OCL est un méta-modèle instance de MOF
- OCL est fortement couplé au méta-modèle UML
- OCL définit sa sémantique à l'aide d'un méta-modèle (opération sans effet de bord)
- OCL définit une syntaxe concrète permettant de facilement saisir les contraintes

Action Semantics



- AS définit la structuration des modèles représentant des séquences d'instructions
- AS est un méta-modèle qui a été totalement intégré à UML2.0
- AS n'a pas de syntaxe concrète propre (utilisation des diagrammes dynamiques UML)
- Un autre standard en cours de finalisation: fUML Foundation of Executable UML

XMI: XML Meta data Interchange

Fonctionnement

- Le standard XMI (XML Metadata Interchange) permet le passage des modèles aux documents XML
- Il définit des règles permettant de construire des schéma XML à partir de méta-modèle
- Ainsi il est possible d'encoder un modèle dans un document XML

XMI et UML

- XMI se décline en 6 versions et UML se décline en 4 versions
  - Cela explique pourquoi l'échange de modèles UML en XMI pose quelques problèmes

XMI et diagrammes

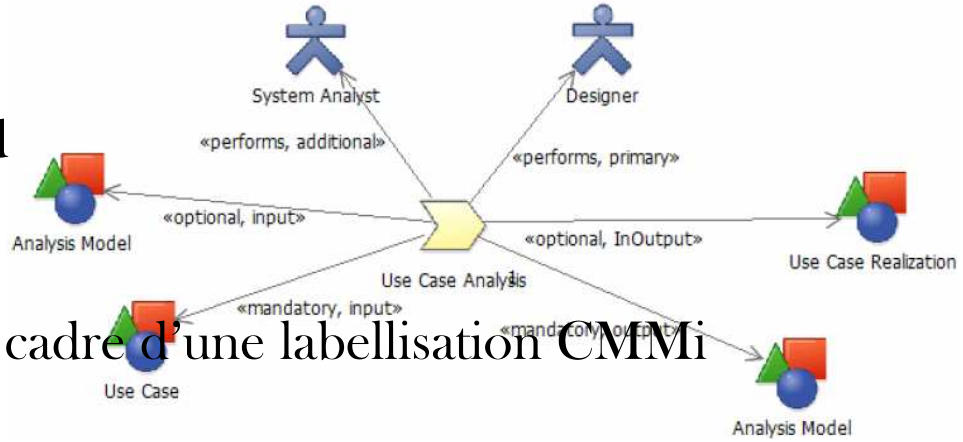
- DI (Diagram Interchange) est une extension à XMI et UML qui permet la représentation des diagrammes UML sous forme de document XML.

QVT (Query View Transformation)

- Le standard QVT permet de modéliser les règles de transformation
- Ce standard comporte 2 façons différentes de modéliser les transformations
- La façon relationnelle permet de définir des relations entre concepts. L'exécution d'une transformation n'est pas déterministe
- La façon opérationnelle se base sur les appels d'opérations de navigation et de modification de modèles
- Le LIP6 a contribué à ce Standard

SPEM

- SPEM (Sofware Process Engineering MetaModel) permet de modéliser les procédés de génie logiciel
- Tous les procédés (RUP, Cycle en V, ...) ont été modélisés en SPEM
- L'objectif est de pouvoir capitaliser les procédés mais aussi à terme de pouvoir contrôler leur exécution
- LIP6 a contribué dans ce standard
- Ce standard est important dans le cadre d'une labellisation CMMi
- Outils Eclipse: EPF



Standards MDA: Synthèse

- Grâce au MOF, tous les méta-modèles sont homogènes (diagramme de classe)
- Grâce à UML, il existe un langage de modélisation très généraliste et ciblant plusieurs niveaux d'abstraction
- Grâce à OCL, il est possible de préciser des contraintes d'exploitation des modèles
- Grâce à AS, il est possible de modéliser finement des comportements
- Grâce à XMI, il existe un format unique de sauvegarde des modèles vers XML
- Grâce à SPEM, les procédés de développement sont modélisables
- Grâce à QVT, les transformations sont modélisables.

General-Purpose Modeling Language
Vs.
Domain-Specific Modeling Languages
L'éternel débat!

DSML: Définition

- Dans le monde du GL, un DSML est un langage de modélisation dédié à un domaine précis à l'inverse d'un general-purpose modeling language (GPML) qui lui pourra être utilisé dans plusieurs domaines.
  - UML Vs. SPEM,
  - dans le domaine des langages de prog. Java, Cobol, SQL, etc.
- MOF DSML pour définir des méta-modèles
- SPEM DSML pour définir des modèles de procédés logiciels
- OCL DSML pour définir des contraintes sur les modèles
- Etc.,

DSML: Motivation

- Complexité: UML (General-Purpose Modeling Language) trop large, trop complexe
  - On ne veut pas tout utiliser!
  - Liens entre les vues pas toujours évident

- Abstraction: Concepts UML plus proches de la technologie d'implémentation (i.e. OO avec Classe, Interface, héritage, etc.) que du domaine métier (i.e. Formulaire, Client, connecteur, communication, message, etc.)
  - Les experts métier sont plus à l'aise avec les concepts qu'ils utilisent tous les jours qu'avec des concepts liés à une plateforme donnée

- Productivité: difficile d'atteindre les 100% de code généré à partir de modèles UML
  - No one-size-fits-all solution!
  - Les meilleures solutions restent propriétaires

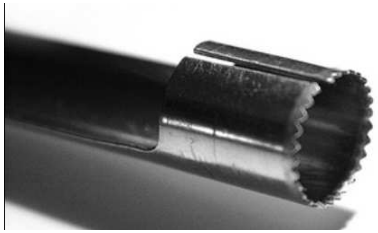
DSML: Motivation

- La métaphore de la Pomme (from P. Friese, Itemis): Quel est le meilleur outil pour peler/vider une pomme?

- Votre couteau suisse qui sert également de tire-bouchon, de ciseaux, décapsuleur, etc.?

DSML: Motivation

- Et si j'avais plusieurs pommes (pour faire une tarte par exp.)?
  - Motiver votre réponse!!



?

?

?

?

DSML: Motivation

- Plus l'outil est spécialisé, plus la productivité augmente
  - Même chose pour les langages

- Autour de nous plusieurs langages cohabitent
  - Java, SQL, ANT, XML, C, etc.
  - Pas de notion de « meilleur langage » mais quel langage pour quel objectif?
- Impossible de réutiliser l'outil (le langage) dans un autre domaine

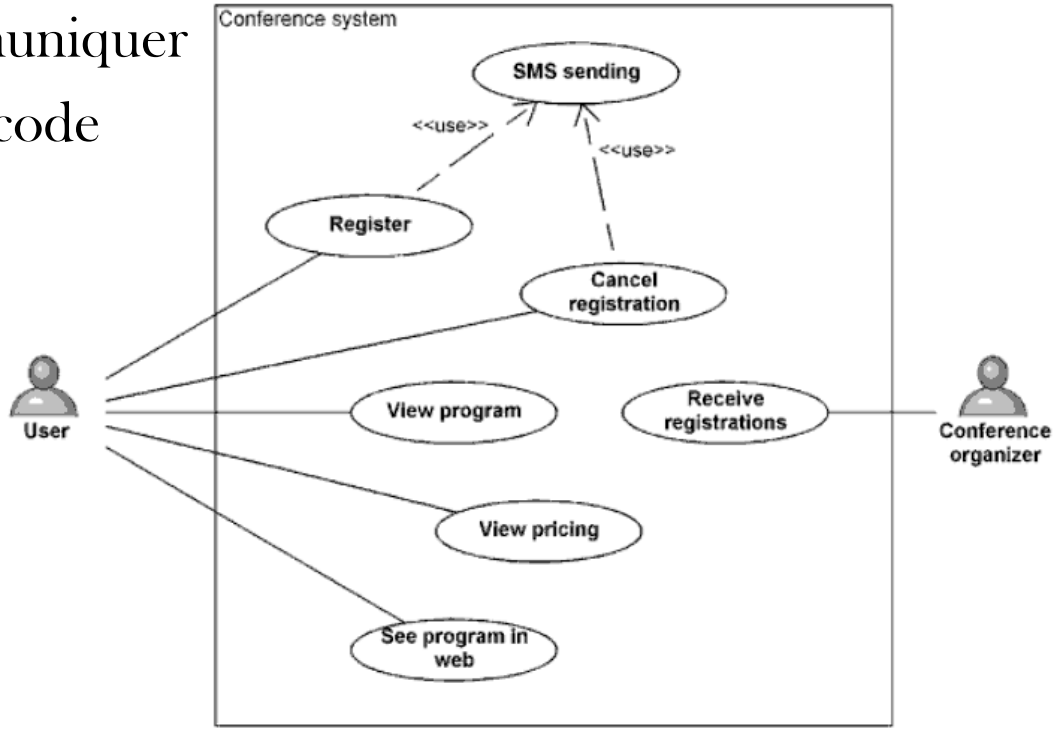
- General-Purpose langage Vs. Domain Specific Languages=> un éternel débat
  - Pas de gagnant! Le choix n'est pas forcément exclusif
  - Une seule règle: savoir choisir le bon langage pour la bonne tâche

Exemple d'un projet avec UML

- Le cas d'une application dans le domaine de la téléphonie mobile (from J-P. Tolvanen)
- Étapes classiques: recueil des besoins, modélisation, code, etc.
- Les modèles utilisés pour générer une partie du code
  - Le reste du code à la main (intégration à un Framework, utilisation des librairies du langage cible, etc.)
- À la fin du projet, défi: garder une synchro. entre code et modèles
  - Souvent les modifications se font au niveau du code
  - Modèles complètement obsolètes au bout de plusieurs modifs.

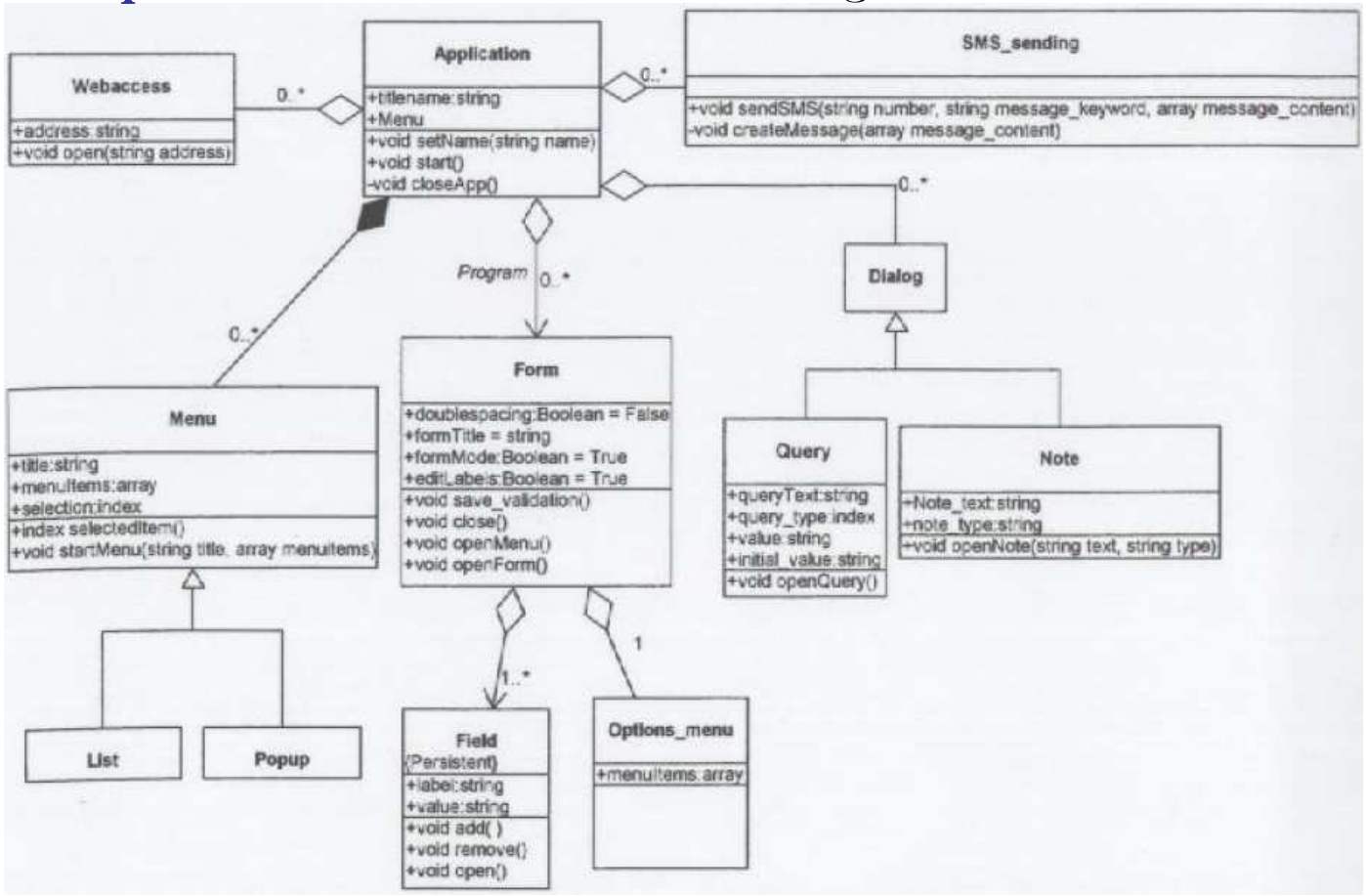
Exemple d'un projet avec UML

- Projet: une application pour s'enregistrer à une conférence depuis son mobile
- Étape 1: point de vue fonctionnel => Use Cases
  - Très bien pour communiquer
  - Pas de génération de code



Exemple d'un projet avec UML

- Étape 2: point de vue structurel=> Diagramme de classes

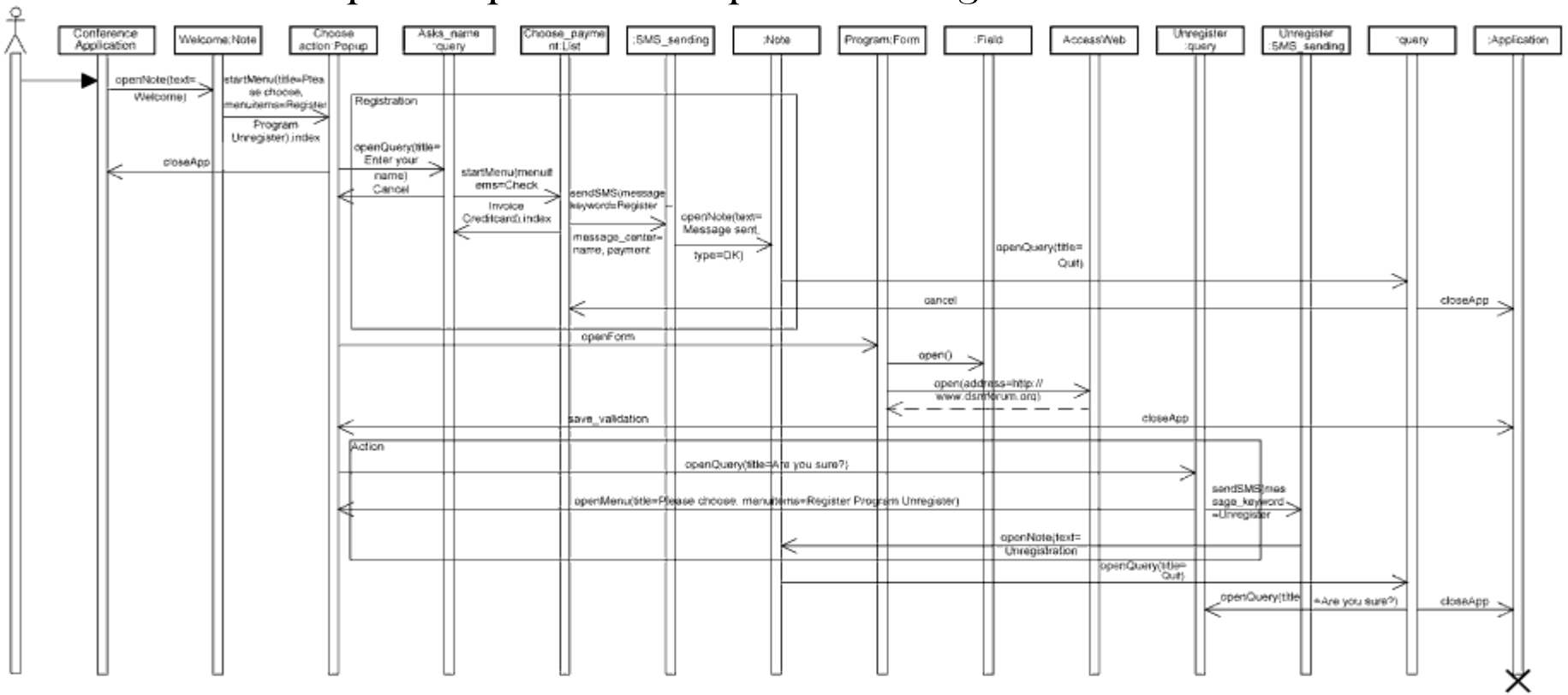


Exemple d'un projet avec UML

- Étape 2: point de vue structurel=> Diagramme de classes
  - "Classes" et "associations" au lieu de "widget", "menu", "champs"
  - Notation graphique sans aucun lien avec le domaine
  - Aucune vision de comment cela va s'intégrer avec le Framework de l'application mobile
  - Le code des opérations?
  - Code généré en 100%?

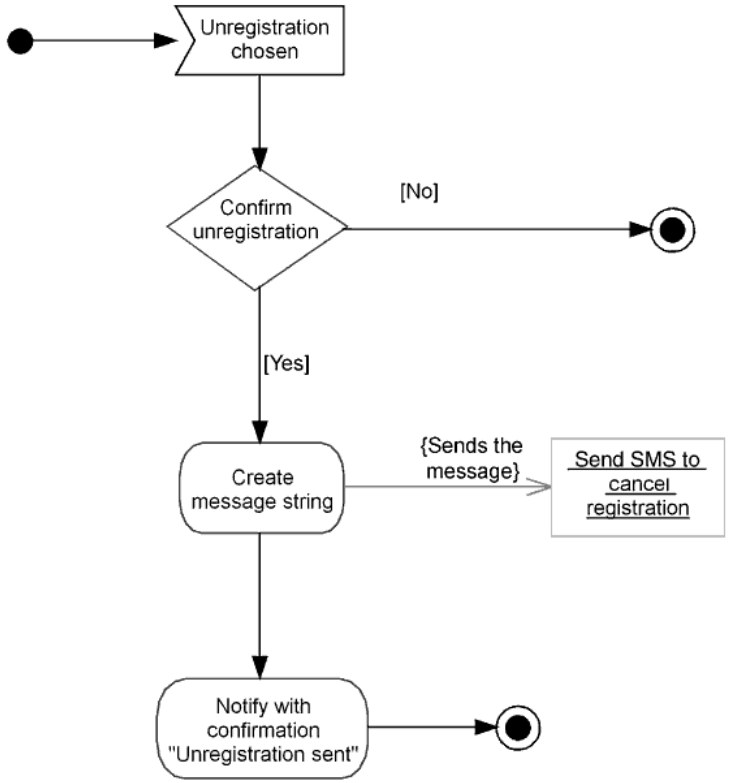
Exemple d'un projet avec UML

- Étape 3: point de vue comportemental=> Diag. de Séquences
  - Le lien avec les classes/librairies de la plateforme d'exec?
  - Assez expressif pour tout capturer? Diag. d'activités?



Exemple d'un projet avec UML

- Étape 3: point de vue comportemental=> Diag. d'activités
  - Pour les traitements complexes, est-ce plus lisible que du code?
  - Combien de personnes maîtrisent les diag. d'activités?
  - « It's worth using UML as a programming language only if it results in something that's significantly more productive than using another programming language » Martin Fowler

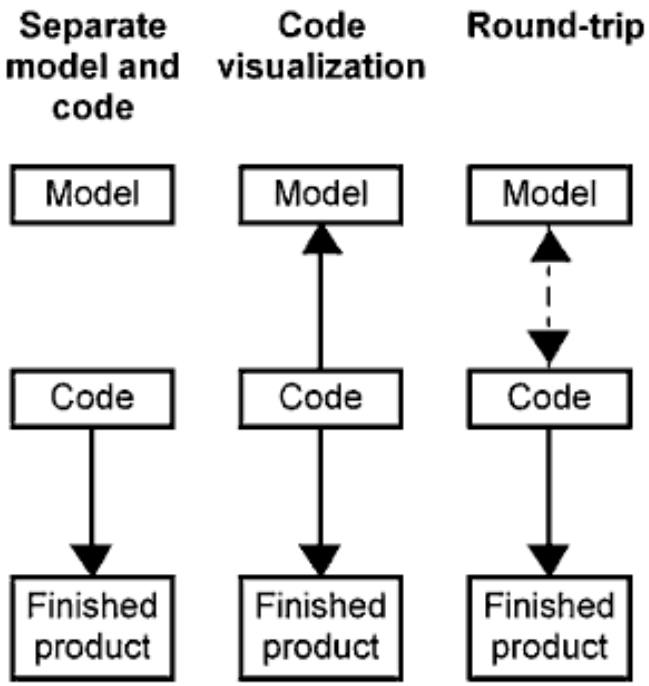


Exemple d'un projet avec UML

- Doit-on utiliser les 9 diagrammes UML pour tout spécifier?
  - The UML fever (Bell, 2004)
- Si je modifie une classe, l'impact sur les autres diagrammes?
- Est-ce que le code généré à partir de ces vues est opérationnel?
  - Problème d'intégration des vues?
  - Que le squelette de l'application
- Est-ce que « remonter tout au niveau du modèle » est la solution?

Exemple d'un projet avec UML

- Que dois-je faire ensuite pour la maintenance? Quelle approche?



General-Purpose Modeling Language: Constat

- Si on veut aller vers plus de productivité, il faut des langages de modélisation qui se rapprochent le plus possible de notre domaine métier => **DSML**

- Plus le domaine est délimité plus la productivité sera grande

- Des générateurs de code très spécifiques, dédiés au domaine
  - Ça ne marchera pas ailleurs (un autre domaine, une autre boite)!
  - Fini le one-size-fits-all

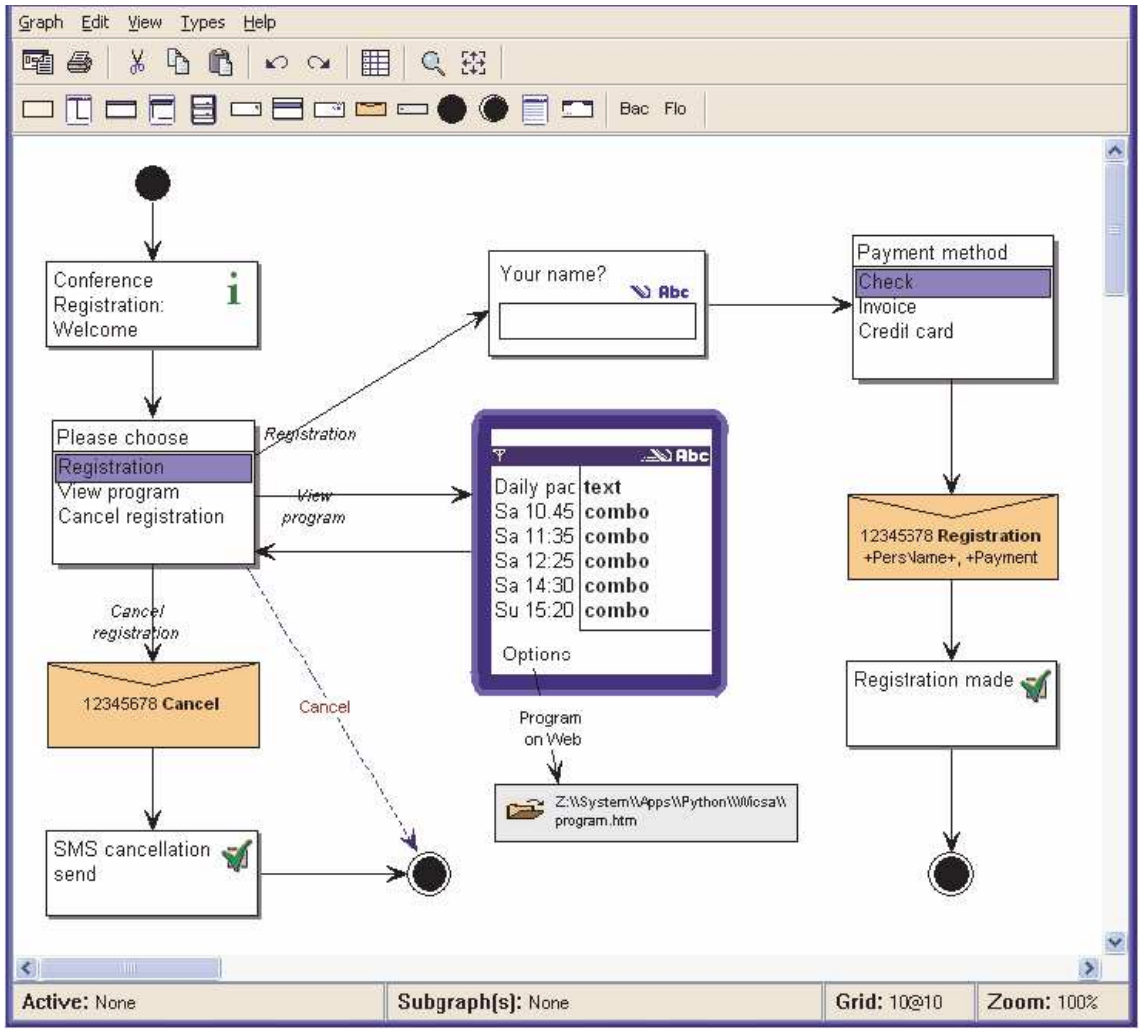
Exemple d'un projet avec un DSML

- Un langage complètement dédié à un domaine i.e. téléphonie mobile et à une compagnie précise i.e. Nokia

- Utilisation directe des concepts du domaine Pop-up, SMS, Formulaire, Service, etc.

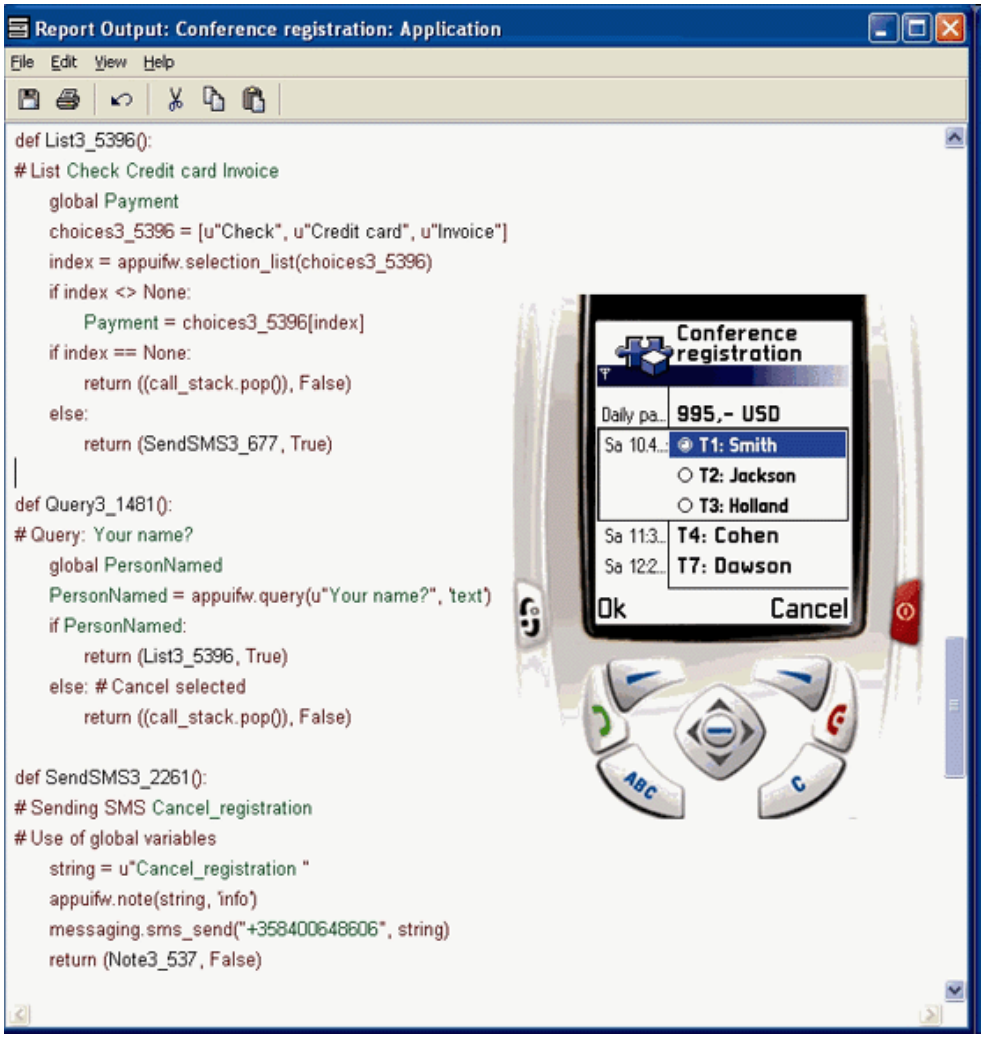
- L'enchaînement est décrit à travers des flux de navigation

Exemple d'un projet avec un DSML



Exemple d'un projet avec un DSML

- Le code!



Exemple d'un projet avec un DSML: Résultats

- Complexité: les modèles sont accessibles et sont faits par des experts métier et non pas forcément des développeurs/analystes
  - Langage avec peu de concepts!
  - Le « tout graphique »

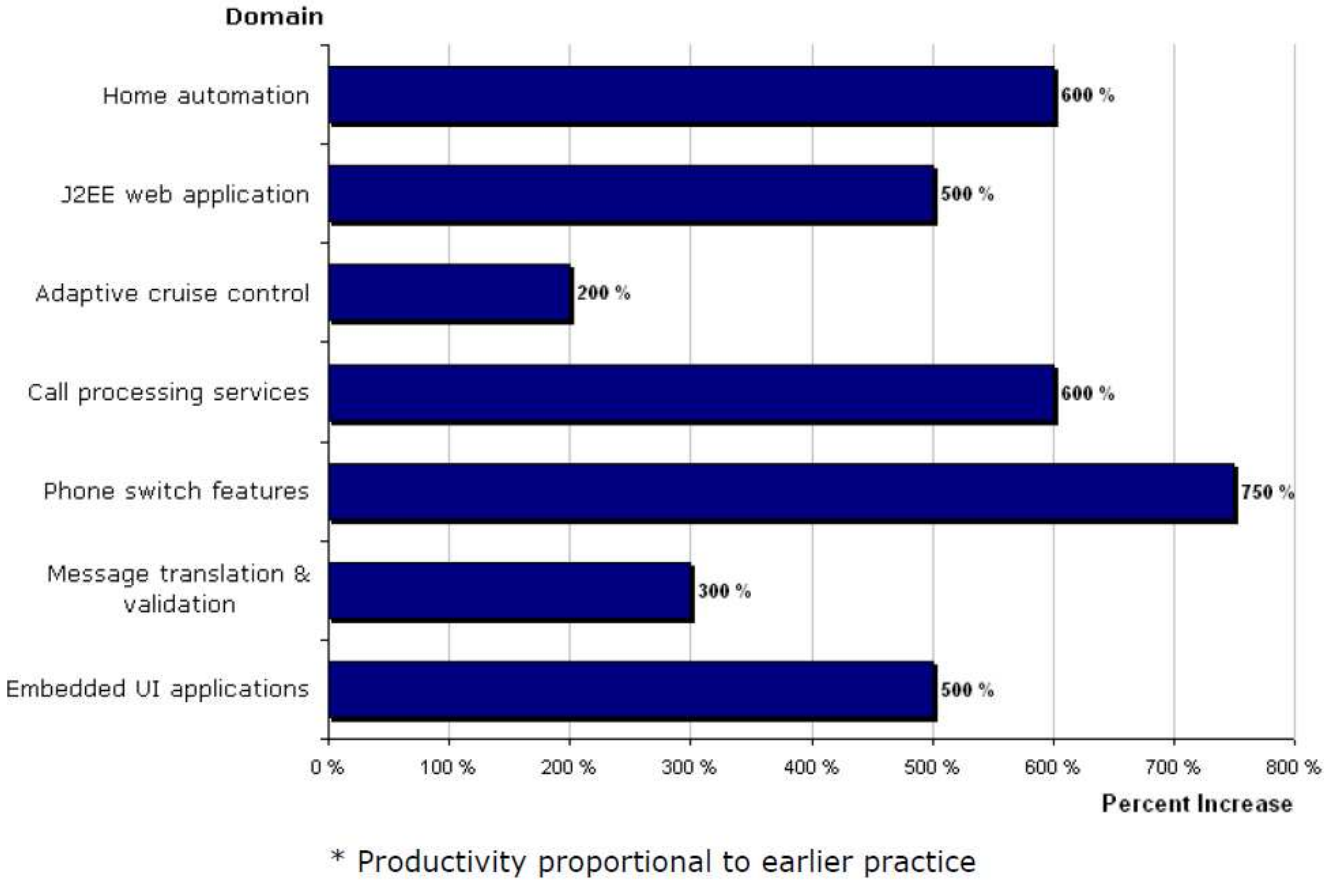
- Un meilleur niveau d'abstraction=>utilisation de concepts propres au domaine de modélisation

- Productivité: les générateurs de code intègrent directement les librairies du Framework/plateforme d'exécution
  - 100% du code généré
  - Peu de maintenance (tout le savoir faire est dans les générateurs)

DSMLs: Success Stories

- Nokia jusqu'à 1000%
  - « UML and other methods and technologies say nothing about mobile phones. We are looking for more » Nokia Project Manager
- US Air Force 300%
- Lucent plus 300%
  - Développement de plusieurs DSMLs pour leurs clients

DSML: Retombées Economiques -Productivité en chiffres-



Source MetaCase 2009

DSML: Retombées Economiques

- Time-to-market plus court
- Feedback plus rapide avec les clients
- Réduction des coûts (si répétition + plusieurs variantes)
- S'intègre bien avec la vision Lignes de Produits
- Le nombre de participants au projet devient potentiellement plus grand
- Réduction des coûts de formation
  - « Nokia case showed that the domain knowledge contained in the modeling language produced faster learning curve for new employees or transferred personnel. Before, a new developer becomes productive after 6 monts. 2 weeks with the DSML » (source Narraway)

GPML Vs. DSML: Productivité

- En moyenne entre 300 et 1000% plus productif que l'utilisation de GPML ou du code directement
  - Source (Kieburtz et al., 1996, Ripper 1999, Weiss and Lai 1999, Kelly and Tolvanen 2000, Cutter Consortium report (Rosen 2006) and the Burton Group (Haddad 2004)

- Comparaison avec UML
  - Meilleure productivité recensée =35% (Compuware 2003). Etude faite par l'outilleur UML lui-même
  - Certaines études montrent que les gains sont minimes (L. Briand TSE)
    - Peu de gains en terme de productivité (très minimes)
    - Qualité quasi similaire
  - Etude sur 5 compagnies : aucune différence entre l'utilisation d'UML et les techniques traditionnelles de dev. (Dubinsky et al., 2006)

Quand investir dans un DSML?

- Quand les langages généralistes n'ont pas le bon niveau d'abstraction ou sont loin du domaine métier
- Généralement le DSML est bien pour un DSM particulier et rien d'autre!
- Il faut bien connaître son domaine
- Construire un DSML suppose que le DSM va durer un certain temps afin de rentabiliser les coûts de construction du DSML
  - N'est pas rentable pour les projets à court terme, sans variantes
- Si ça marche déjà très bien, ne rien toucher!
- Quand les experts du domaine ne sont pas des informaticiens
- Toujours vérifier qu'il n'existe pas déjà un DSML pour votre domaine

DSML: à prévoir!

- Apprendre à modéliser
- Apprendre le domaine et le DSML
  - Selon une étude (Ruuska 2001) : 2.5 jours pour apprendre une solution DSML. 11 sur 17 dev l'ont fait en un jour
- Difficile de réutiliser des solutions existantes car la plupart du temps les boites les gardent secrètement. C'est un critère de compétitivité
  - Parfois obligé par l'outsourcing
  - Ou par soucis de standardisation, exp. Autosar, Health Level Seven
- Le coût de l'outillage à ne pas négliger
  - **Exp UML-eclipse editeur, + de 20 H/An (source Ströbele 2005)**
  - Souvent peu de développeurs mais plus d'experts du domaine
  - **Coût qui a tendance à baisser avec les nouveaux outils/formalismes**
    - **C'est votre mission!!!**

DSML ou comment apprendre à méta-modéliser

Note: dans ce qui suit, nous utilisons le terme DSML au lieu langage de modélisation

DSML: Choix de conception

- Objectif
  - Documentation, exécution, les deux?
- Portée
  - Spécifique, général, extensible?
- Formel, informel?
  - Sémantique?

DSML: c'est quoi?

- Une syntaxe abstraite (le méta-modèle)
  - Un ensemble de concepts et de relations entre ces concepts
    - Exp. Class, Association, Package, Device, Flow, Process, Action, etc.
  - Des règles/contraintes sur les concepts (well-formedness rules)
    - Exp. Un rôle peut être assuré par un acteur de type agent ou équipe mais pas de type outil
- Une syntaxe concrète
  - Représentation graphique et/ou textuelle
- Une sémantique
  - la signification de chaque concept du langage? Que doit-il représenter/capturer?

=>Illustrer avec UML?

DSML: c'est quoi?

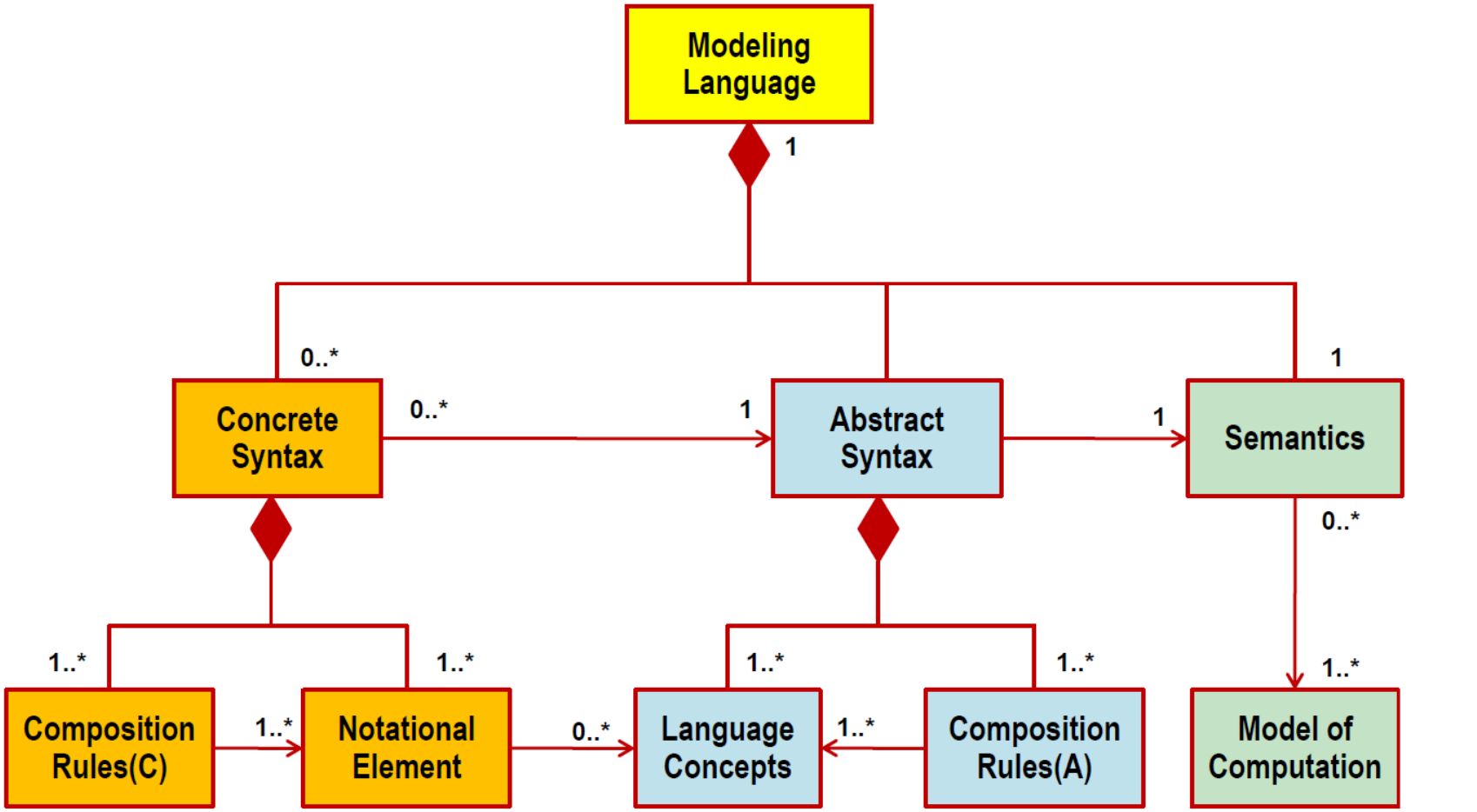


Fig. from B. Selic

DSML: Syntaxe Abstraite

- Syntaxe Abstraite: Concepts + règles

- Définition par méta-modélisation
  - Le choix du langage: MOF, EMF, GME, Kermeta, etc?
  - L'approche de méta-modélisation (diapos suiv.)?
  - Règles /Contraintes: choix du formalisme? Math, OCL (cours 3)

- Autres approches (EBNF, XSD, Propriétaires, etc.)
  - GOPRR (Graph, Object, Property, Role, Relationship) de MetaEdit (S. Kelly 97)
  - Software Factories de Microsoft

DSML: Syntaxe Concrète

- Graphique ou textuelle, ou les deux?
- Plusieurs représentations (textuelles/graphiques)?
- Plusieurs vues/diagrammes

DSML: Sémantique

La sémantique derrière vos modèles!

- Structurelle?
  - Décrit des éléments du domaine, les relations qui les lient et les données [et de comportement] qu'ils portent (état)
- Comportementale?
  - Décrit la réaction/comportement de vos éléments du langage face à des stimuli
- Les deux?
  - Exp. Un diagramme d'activités, d'états UML2, etc.
- Notion de points de variations dans la sémantique
  - Le cas UML

DSML: Sémantique

Approches pour la définition de la sémantique

- Opérationnelle
  - un programme vu/interprété comme une suite d'états successifs
- Dénotationnelle
  - formalisation par les maths (Christopher Strachey et Dana Scott.)*
- Axiomatique
  - preuve sur le programe
- Informelle
  - langage naturel

*Joseph E. Stoy, *Denotational Semantics: The Scott-Strachey Approach to Programming Language Semantics*, MIT Press, Cambridge, Massachusetts, 1977

* Glynn Winskel, The Formal Semantics of Programming Language, MIT Press, Cambridge, Massachusetts, 1993

DSML: étapes de réalisation

- S'assurer qu'il n'existe pas déjà un DSML qui répond à vos besoins
- Spécifier la syntaxe abstraite
  - Méta-modèle (la suite du cours)
  - Règles/Contraintes (Cours 3)
- Spécifier la syntaxe concrète (Cours 4)
- Spécifier la sémantique d'exécution, les générateurs de code (Cours 5)
- Tester, itérer, raffiner, etc.

Création d'un DSML: Syntaxe abstraite

- Commencer par identifier les abstractions du domaine
- Choisir une des approches pour la définition du méta-modèle
  - Définir un méta-modèle à partir de rien (définir le MMM)
  - Utiliser un méta méta-modèle existant pour définir votre méta-modèle exp. MOF, Ecore, GME, Kermeta
  - Étendre un méta-modèle existant (exp. UML, SPEM) pour lui rajouter des concepts propres à votre domaine
  - Restreindre, raffiner un méta-modèle existant pour le spécialiser/l'adapter à votre domaine (exp. Principe des profils UML)
    - Profils Corba, SysML, Marte, SoaML, etc.

Création d'un DSML: Syntaxe abstraite

Comment choisir l'approche de définition du méta-modèle?

- Y'a-t-il une grande différence dans la sémantique entre UML et votre DSML?
  - ⇒ Si oui, définir un nouveau méta-modèle (avec MOF ou Ecore par exp.)
  - ⇒ Si non, voir point suivant

- Est-ce que certains concepts de votre DSML sont dans le méta-modèle UML (ou autre DSML) et vous voulez en rajouter d'autres ?
  - ⇒ Si oui, étendre le méta-modèle UML (ou autre DSML)
  - ⇒ Si non, voir point suivant

- Est-ce que les concepts de votre DSML sont dans le méta-modèle UML (ou autre DSML) et vous voulez les raffiner (ne pas tout utiliser, spécialiser) ?
  - ⇒ Si oui, mécanisme de restriction tel que les profils UML (ou autre DSML) ou le modèle pruning (Triskell, 2009 MoDELS)
  - ⇒ Si, non, ...?

À votre avis, les avantages et limites de chaque approche?

Syntaxe Abstraite avec le Framework EMF

- Présentation
- Le méta-modèle
- Principes et architecture
- Génération de code et manipulation de modèles

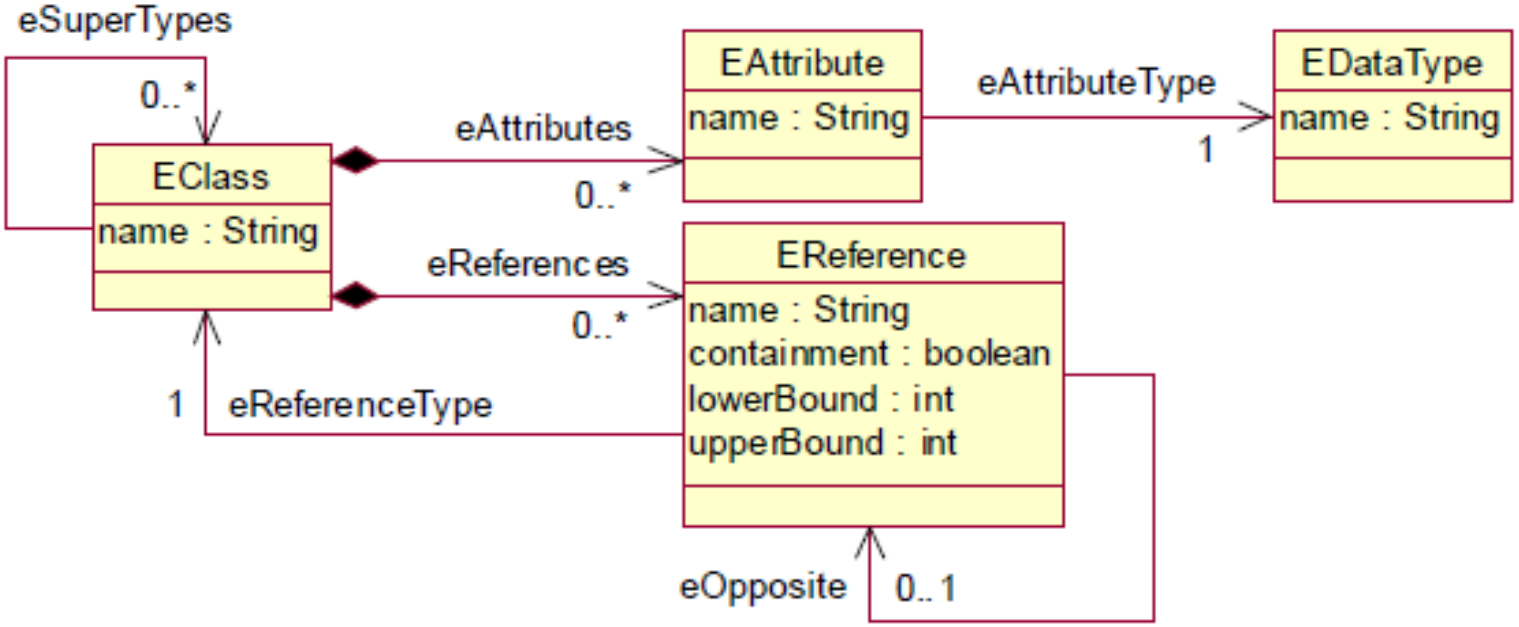
EMF: Présentation

- Eclipse Modeling Framework permet la création d'un langage et de sa suite outillée dans l'environnement Eclipse (depuis 2002)
  - Un Framework riche
  - Un éditeur pour créer des méta-modèles
  - Une génération de bibliothèque Java support à la manipulation de modèles
  - Un éditeur pour construire des modèles
  - Des facilités dédiées (transformation, éditeur graphique, vérificateur de contraintes, ...)

EMF: Le méta méta-modèle

- Utilise/implémente l'essentiel du MOF (i.e. eMOF)
- Le méta méta-modèle s'appelle Ecore (décrit avec lui-même Ecore.ecore)
- Notation: le diagramme de classes UML

Un bout du méta méta-modèle



EMF: Les Types

Ecore Data Type	Java Primitive Type or Class
EBoolean	boolean
EChar	char
EFloat	float
EString	java.lang.String
EByteArray	byte[ ]
EBooleanObject	java.lang.Boolean
EFloatObject	java.lang.Float
EJavaObject	java.lang.Object

EMF: Modèle? Persistance?

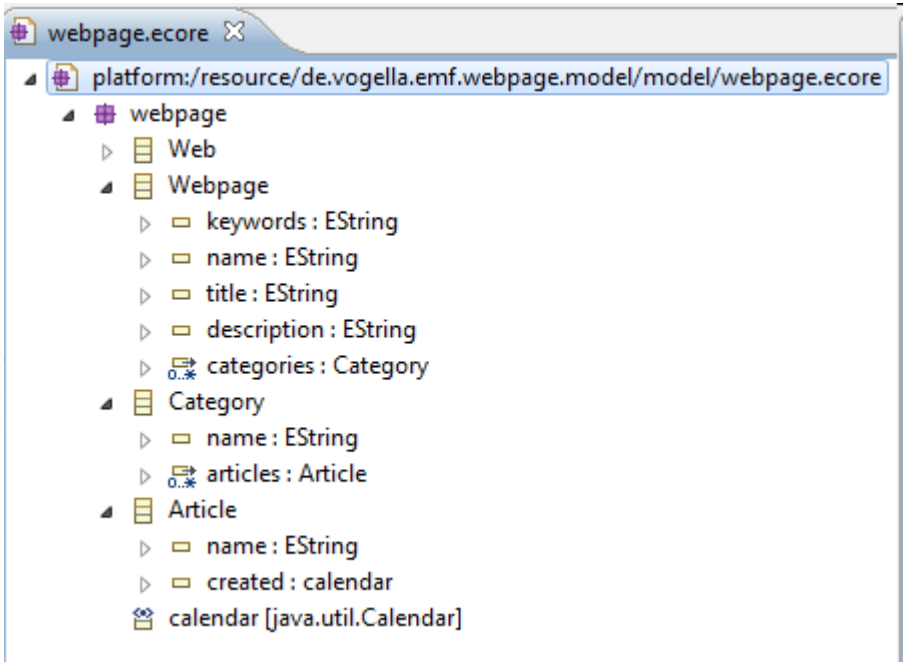
- Un modèle EMF (i.e. Ecore) représente la spécification des données d'une application
  - Attributs
  - Opérations
  - Associations
  - Contraintes, etc.

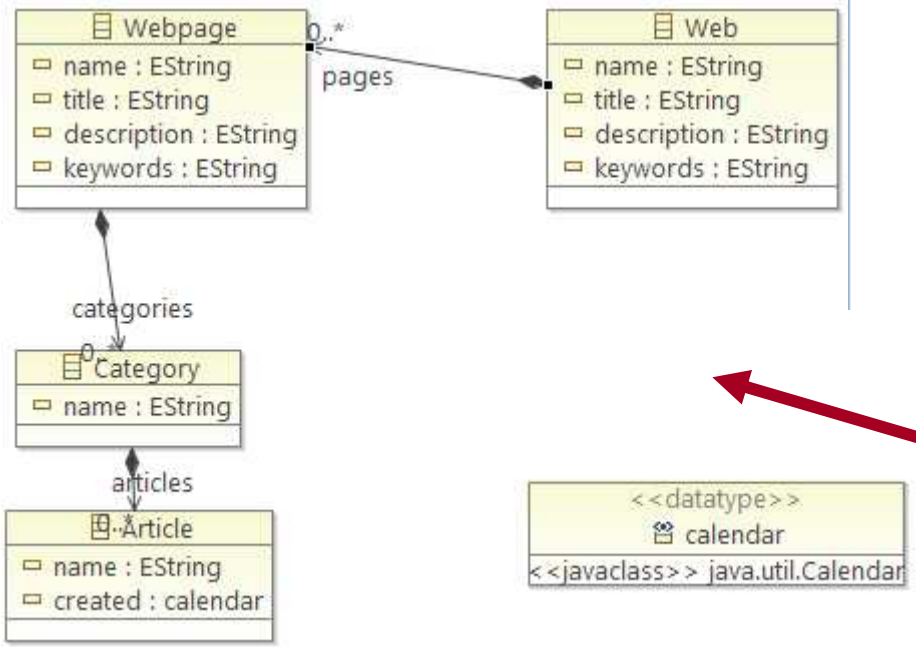
- Dans le cas où votre application c'est la définition d'un DSML=> le modèle Ecore est le méta-modèle de votre DSML

- La persistance des modèles Ecore se fait en XMI
  - Possibilité de les sérialiser en un format compatible avec eMOF
  - De créer son propre format XMI
  - D'autres mécanismes de persistance sont également possible

EMF: Comment créer des méta-modèles?

- En utilisant l'éditeur arborescent d'EMF

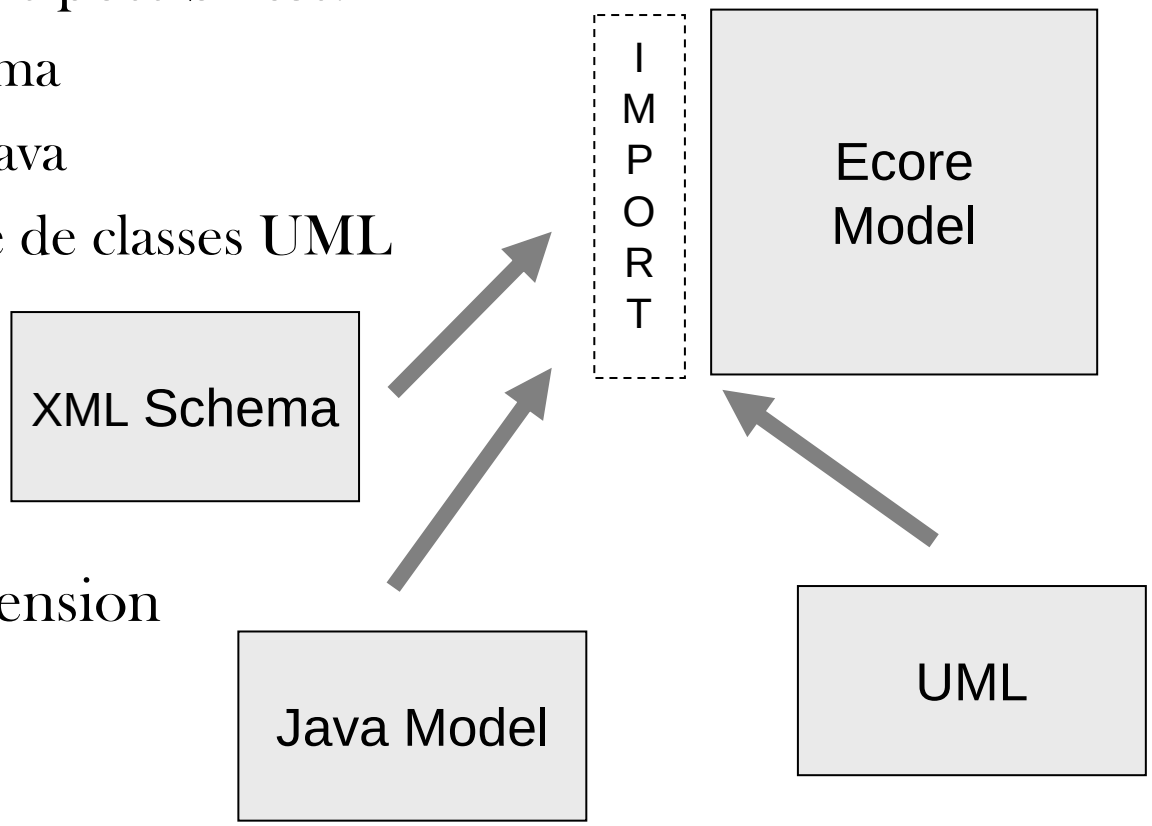




- En utilisant l'éditeur Graphique d'EMF

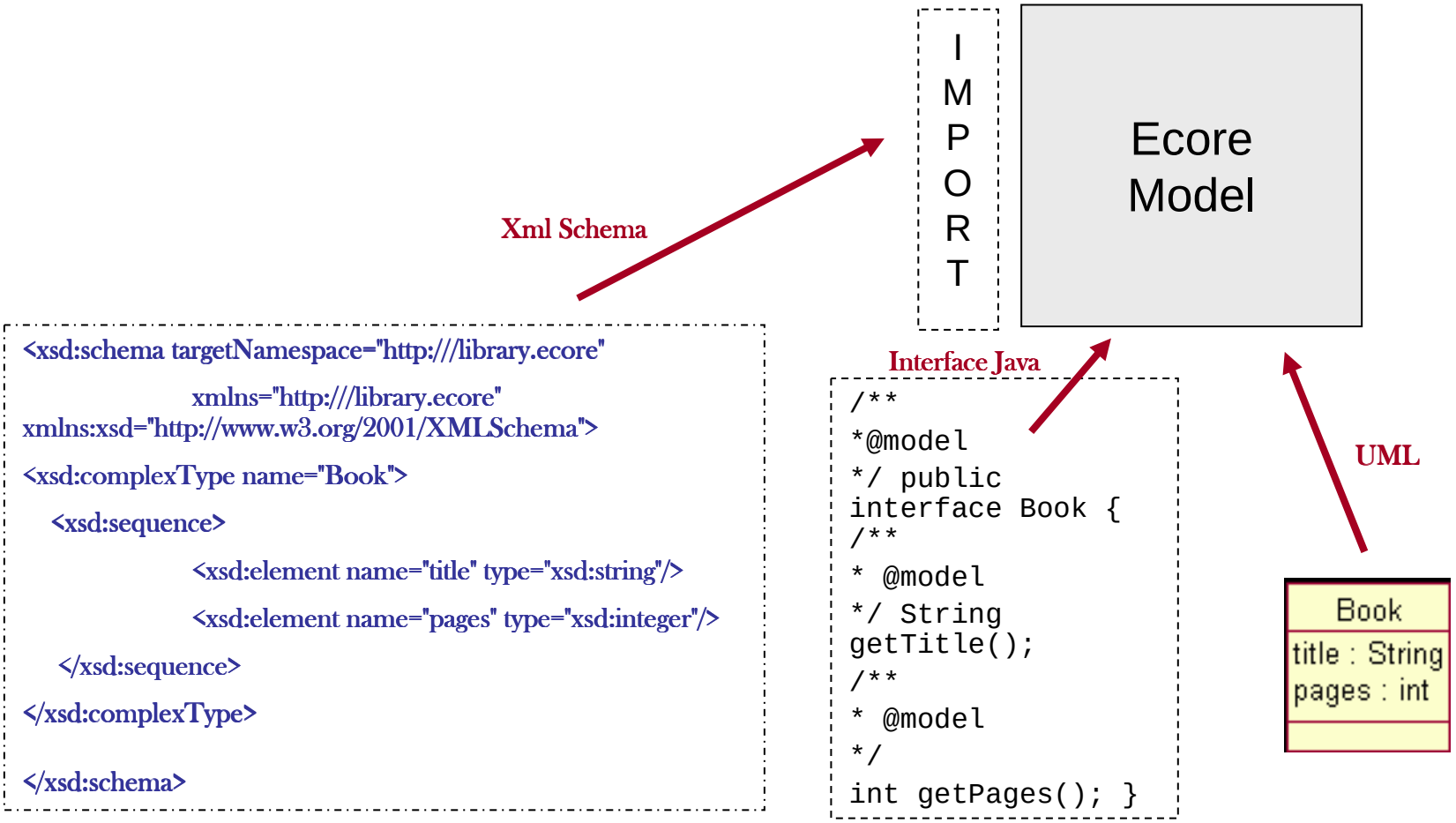
EMF: Comment créer des méta-modèles?

- Un autre moyen est de les créer en dehors d'EMF et de les importer. Trois possibilités:
  - XML Schema
  - Interfaces Java
  - Diagramme de classes UML



- Aussi: Par extension d'un autre Ecore

EMF: Comment créer des méta-modèles?



EMF: Comment spécifier les compositions/associations bidirectionnelles

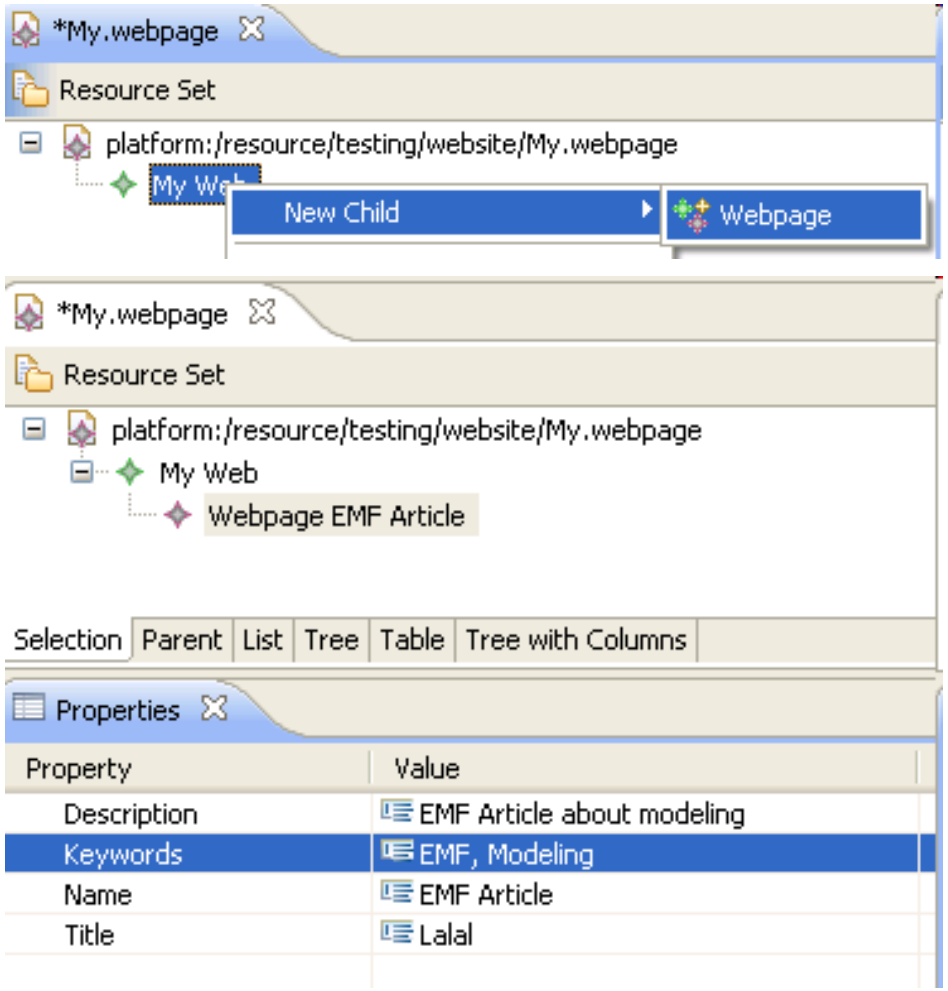
Composition

Assoc. Bidirectionnelle

Multiplicité * =-1 en EMF

EMF: Créer des modèles instances

- Le Framework offre la possibilité de créer des modèles instances de votre DSML à partir de l'Ecore
  - Éditeur arborescent
  - Notion de *dynamic instance*



EMF: Génération du code

- EMF offre la génération de code à partir de modèles

- Code: Interfaces + Classes Java
  - Pour manipuler les modèles/les créer/Les sauvegarder, etc.
    - Éléments du modèles, Factories, Adaptateurs, etc.
    - Héritent de la classe EMF EObject
  - Pour assurer la consistance par rapport au méta-modèle
  - Génération de l'éditeur de modèles + les cas de tests

- Les classes Java sont générées à partir d'un modèle *.genmodel*
  - Lui-même obtenu automatiquement à partir de l'ecore
  - Contient les informations pour la génération vers Java

EMF: Génération du code

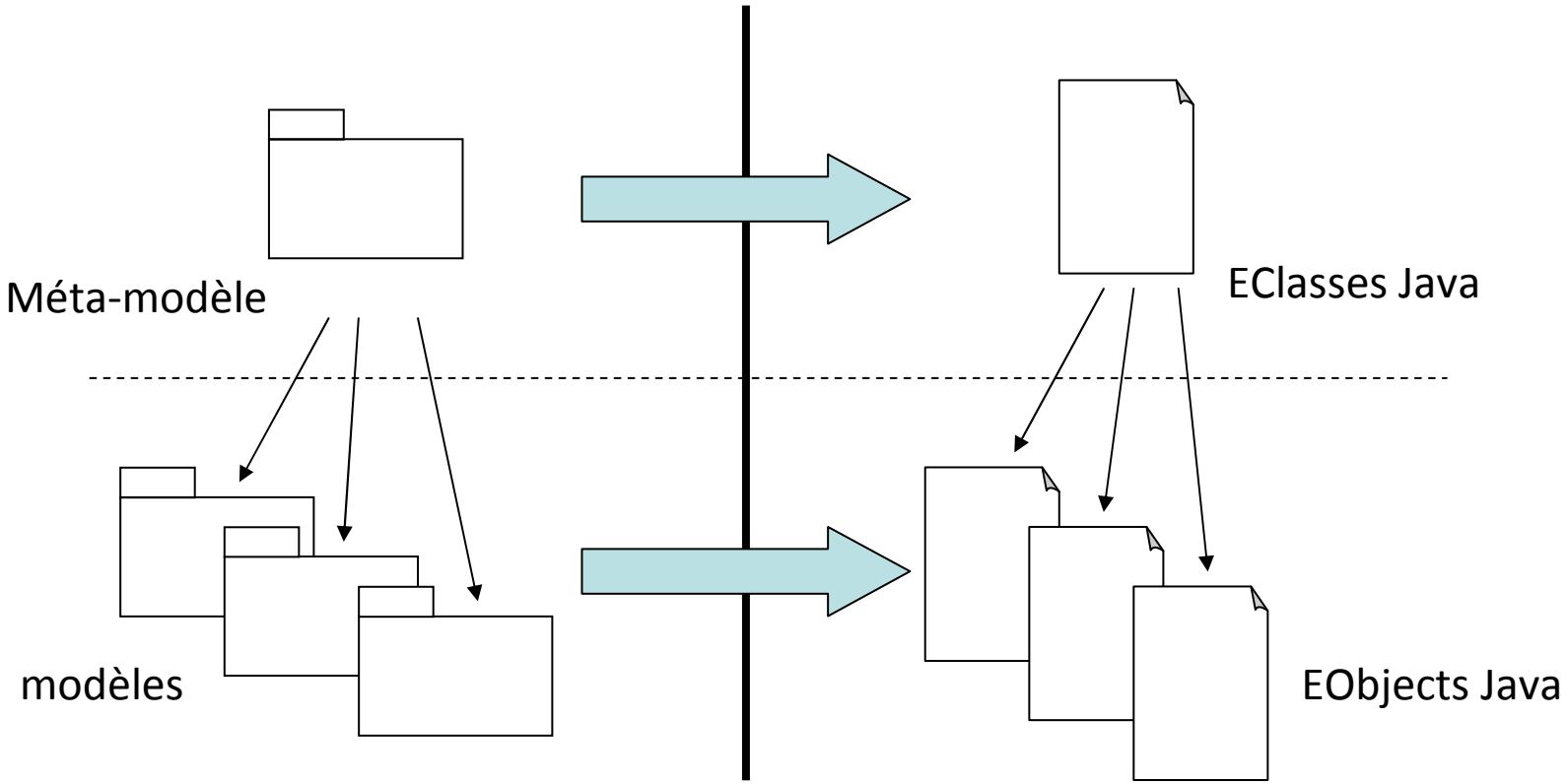


Fig. X. Blanc

EMF: Génération du code

- En cas de re-génération de code, tous les éléments annotés par @generated seront remplacés/effacés

- Afin de préserver votre code, vos ajouts, annoter avec @generated NOT

```
/**
 * <!-- begin-user-doc -->
 * <!-- end-user-doc -->
 * @generated
 */
public String getName()
{
  return name;
}
```

EMF: Manipulation des modèles

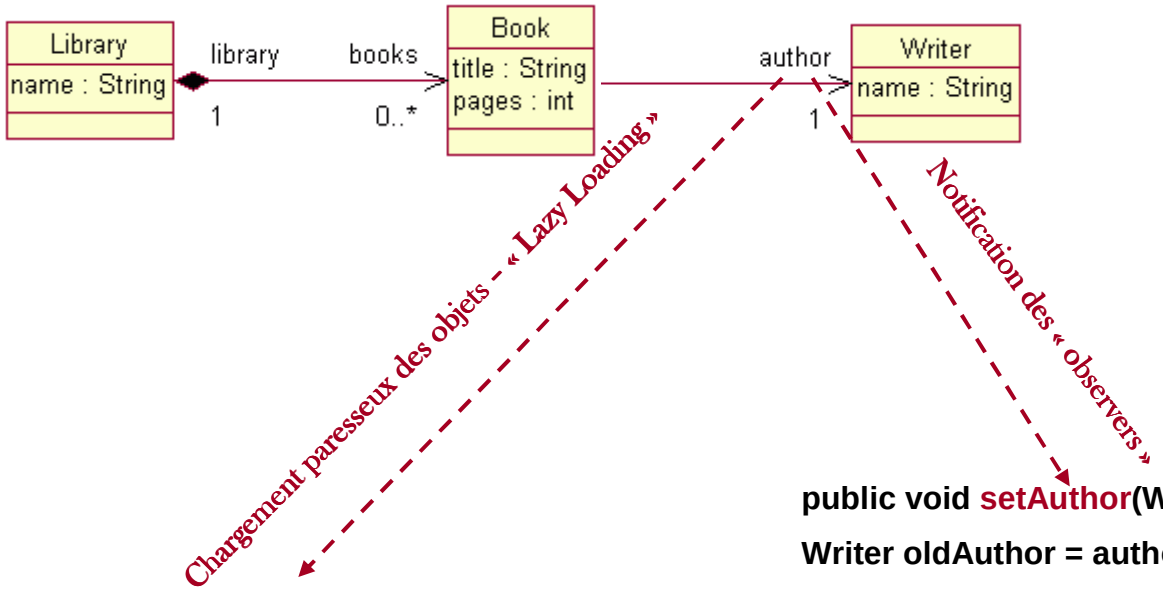
- Sur l'exemple de la « Library »
- Un bout de code pour créer des instances de Book et Writer

```
LibraryFactory factory = LibraryFactory.eINSTANCE;
Book book = factory.createBook();
Writer writer = factory.createWriter();
writer.setName("William Shakespeare");
book.setTitle("King Lear");

book.setAuthor(writer);
```

Si l'association était bidirectionnelle entre Book et Writer, le lien se fait automatiquement grâce aux adaptateurs générés automatiquement par EMF

EMF: Génération du code




```
public void setAuthor(Writer newAuthor) {
Writer oldAuthor = author;
author = newAuthor; if(eNotificationRequired())
eNotify(new ENotificationImpl(this, ...));
}
```

```
public Writer getAuthor() {
if (author != null && author.eIsProxy()) {
Writer oldAuthor = author;
author = (Writer)eResolveProxy((InternalEObject)author);
if (author != oldAuthor) {
if (eNotificationRequired())
eNotify(new ENotificationImpl(this, Notification.RESOLVE, ...)); } }
return author; }
```

EMF: Manipulation des modèles

Pour sauvegarder votre modèle dans un fichier xmi (si le fichier n'existe pas encore)

```
// Create a resource set.
        ResourceSet resourceSet = new ResourceSetImpl();

// Register the default resource factory
        resourceSet.getResourceFactoryRegistry().getExtensionToFactoryMap().put(
        Resource.Factory.Registry.DEFAULT_EXTENSION, new XMIResourceFactoryImpl());

// Get the URI of the model file.
        URI fileURI = URI.createFileURI(new File("mylibrary.xmi").getAbsolutePath());

// Create a resource for this file.
        Resource resource = resourceSet.createResource(fileURI);

// Add the book and writer objects to the contents.
        resource.getContents().add(book);
        resource.getContents().add(writer);

// Save the contents of the resource to the file system.
        try { resource.save(Collections.EMPTY_MAP); }

        catch (IOException e) {}
```

EMF: Manipulation des modèles

Pour charger votre modèle à partir d'un fichier xmi

```
// Create a resource set.
    ResourceSet resourceSet = new ResourceSetImpl();
// Register the default resource factory
    resourceSet.getResourceFactoryRegistry().getExtensionToFactoryMap().put(
    Resource.Factory.Registry.DEFAULT_EXTENSION, new XMIResourceFactoryImpl());
// Register the package
    LibraryPackage libraryPackage = LibraryPackage.eINSTANCE;
// Get the URI of the model file.
    URI fileURI = URI.createFileURI(new File("mylibrary.xmi").getAbsolutePath());
// Demand load the resource for this file.
    Resource resource = resourceSet.getResource(fileURI, true);
// Print the contents of the resource to System.out.
    try { resource.save(System.out, Collections.EMPTY_MAP); } catch (IOException e) {}
```

EMF: Conclusion

- Framework très puissant
- Commence à être mature, mais tous les projets!
- Une communauté qui ne cesse de croître!
- Bien documenté, pas mal de forums
- Dès qu'on sort des exemples simples, ça commence.....

Les Profils ou l'autre manière de créer vos DSML

-Présentation

-Concepts

-Exemple

Profils UML

Un profil permet d'adapter UML pour :

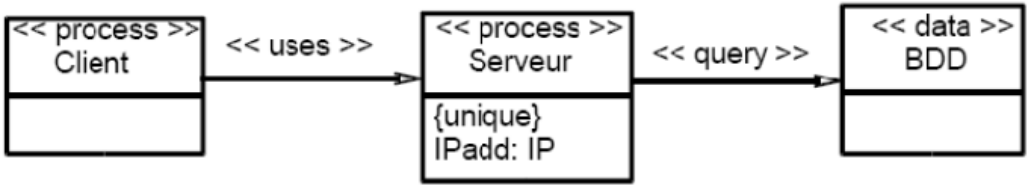
- Préciser une terminologie propre à un domaine ou à une plateforme
  - Exp. Terminologie EJB : Stateless, Statefull, Entity
- Donner une syntaxe à des concepts qui n'en ont pas
  - Les actions, des noeuds de contrôle dans le domaine du temps réel
- Donner une notation différente à des symboles existants
  - Une image d'ordinateur au lieu d'un nœud ordinaire pour représenter un ordinateur dans un réseau
- Ajouter de la sémantique
  - Gestion des priorités à la réception d'un signal, des timers

Profils UML: Concepts

- Un profil UML est composé de 3 types d'éléments
  - Des stéréotypes
  - Des tagged value
  - Des contraintes (exprimables en OCL)
    - Sur ces stéréotypes, tagged values
    - Sur des éléments du méta-modèle existant
    - Sur les relations entre les éléments
- Un profil UML est défini sous la forme d'un package stéréotypé << profile >>
- Exemple de profil : architecture logicielle
  - Des composants clients et serveur
  - Un client est associé à un serveur via une interface de service par l'intermédiaire d'un proxy

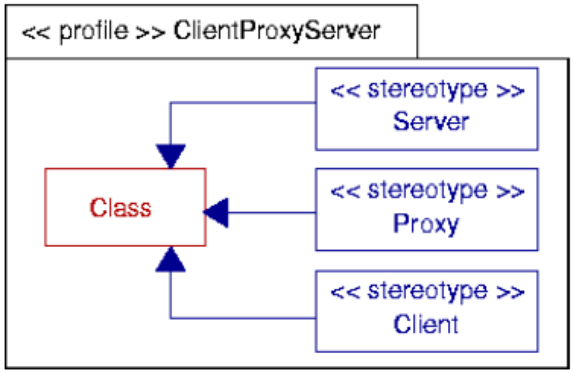
Profils UML: Concepts

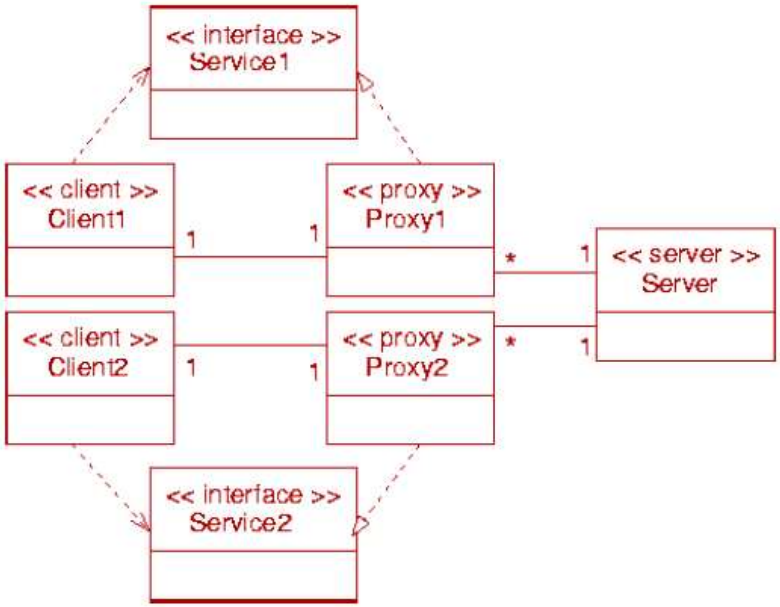
- Stéréotype
  - Extension, spécialisation d'un élément du métamodèle
  - Classe, association, attribut, opération ...
  - Le nom d'un stéréotype est indiqué entre << ... >>
  - Il existe déjà des stéréotypes définis dans UML
    - << interface >> : une interface est un classifier particulier (sans attribut)
- Tagged value (valeur marquée)
  - Pour marquer des attributs d'une classe pour préciser une contrainte ou un rôle particulier
    - Exemple : {unique} id: int



Profils UML: Exemple

- Profil défini
  - nommé ClientProxyServer
  - Définit trois stéréotypes
    - Trois classes jouant un rôle particulier : extensions de la métaclasse Class du méta-modèle UML
      - Server, Proxy, Client





Profils UML: Exemple

- Pour compléter le profil, ajout de contraintes OCL
  - Navigation sur le métamodèle UML (simplifié) en considérant que la méta-classe Class a trois spécialisations (Server, Client, Proxy)
  - Un proxy associé à un client doit implémenter une des interfaces dont dépend le client et un proxy implémentant une interface d'un client doit avoir une association avec ce client
- **context** Client **inv:**

  let proxies = self.associationEnd.association.associationEnd.class -> select (c | c.oclIsTypeOf(Proxy)) **in**

  let interfaces = self.dependsOn **in**

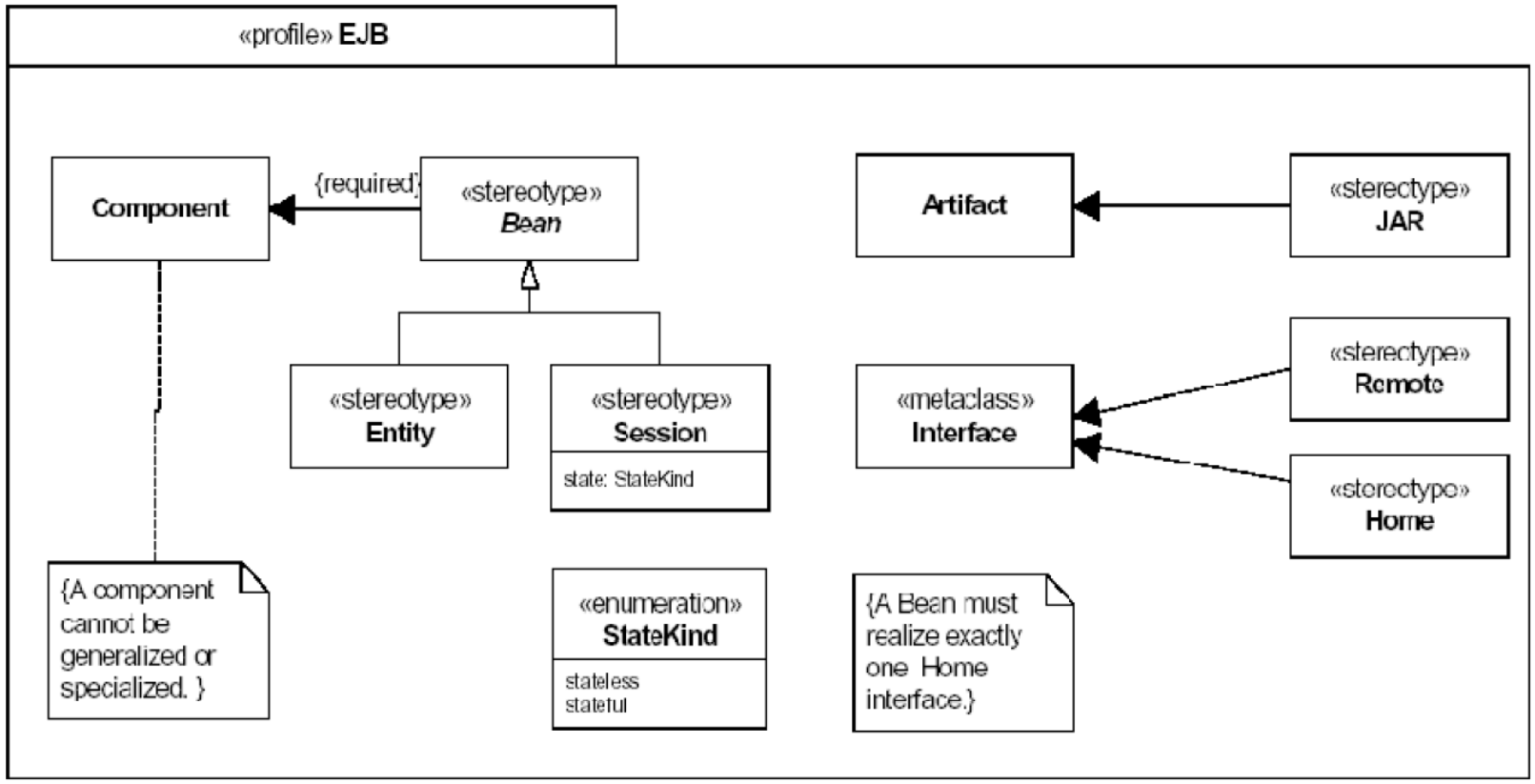
  interfaces -> forAll (i | proxies.implements -> includes (i) **and**

  proxies -> forAll (p | p.implements -> includes (i)

  implies p.hasClassRefWith(self)))
- **context** Class **def:** hasClassRefWith(cl : Class) :

  Boolean = self.associationEnd.association.associationEnd.class-> exists (c| c = cl)

Profils UML: Exemple d'un profil EJB



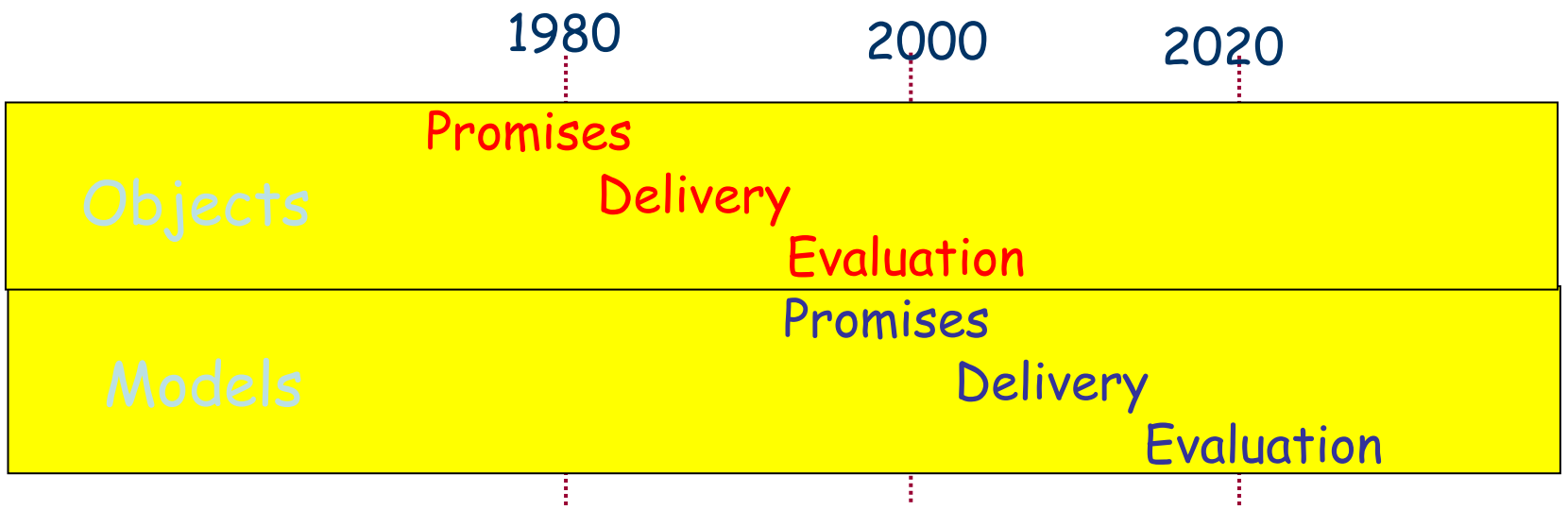
Profils UML: Limites

- On ne pas prendre une partie d'UML, c'est le tout ou rien ou alors écrire 1000 règles OCL
- On ne peut pas rajouter de nouveaux types
- Il faut être pro OCL
- L'outillage!

Conclusion

MDE: Recul?

- Object technology realized some promises but failed to achieve others
  - Stopping the search for generality by unification may be one of the causes for this
- Model engineering is making many promises today
  - Will it be able to deliver correspondingly?
  - Sticking with the principle that "everything is a model" seems a good way to make progresses



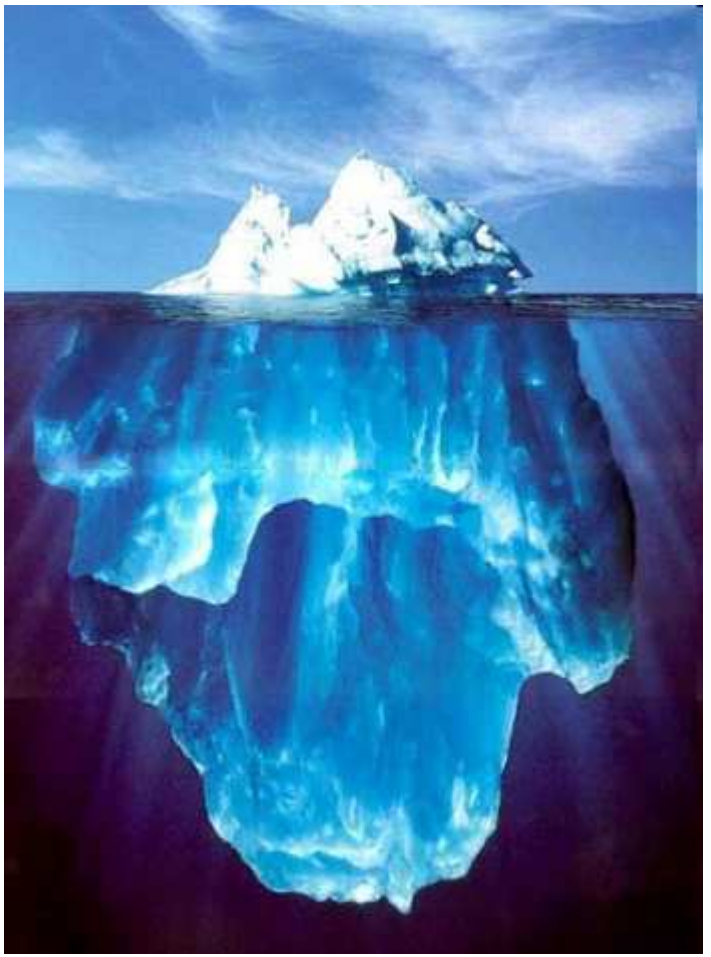
MDE: Recul?

- La métaphore de l'iceberg

- À ne pas confondre avec la brève de comptoir

« *C'est un iceberg, celui-là, sept fois plus c.. que ce qu'on voit.* »

Source: Jean-Marie Gourio - *Extrait de Brèves de comptoir* »



Lectures

- Software Engineering,
  - Ian Sommerville, Addison Wesley; 8 edition (15 Jun 2006), ISBN-10: 0321313798
- The Mythical Man-Month
  - Frederick P. Brooks JR., Addison-Wesley, 1995
- UML Distilled 3rd édition, a brief guide to the standard object modeling language
  - Martin Fowler, Addison-Wesley Object Technology Series, 2003, ISBN-10: 0321193687
- MDA en Action, de Xavier Blanc, 2005, chez Eyrolles,
- Domain-Specific Modeling: Enabling full code generation, par S. Kelly et J-P, Tolvanen, Wiley Interscience 2008
- Cours de Software Engineering du Prof. Bertrand Meyer à cette @:
  - http://se.ethz.ch/teaching/ss2007/252-0204-00/lecture.html
- Cours d'Antoine Beugnard à cette @:
  - http://public.enst-bretagne.fr/~beugnard/
- Cours très intéressants du Prof. Jean-Marc Jézéquel:
  - http://www.irisa.fr/prive/jezequel/
- Cours de Jean Bézivin, Benoit Combemale, Sébastien Mosser, Mireille –blay fornarino, Anne Etien (Google is your friend: nom + mde ou page perso)
- Cours Xavier Blanc pour l'école d'été MDE 2009 (supports non disponibles en ligne)
- La page de l'OMG dédiée à UML: http://www.omg.org/mda/ + Le guide MDA de Richard Soley sur omg.org
- Design patterns. Catalogue des modèles de conception réutilisables
  - Richard Helm (Auteur), Ralph Johnson (Auteur), John Vlissides (Auteur), Eric Gamma (Auteur), Vuibert informatique (5 juillet 1999), ISBN-10: 2711786447