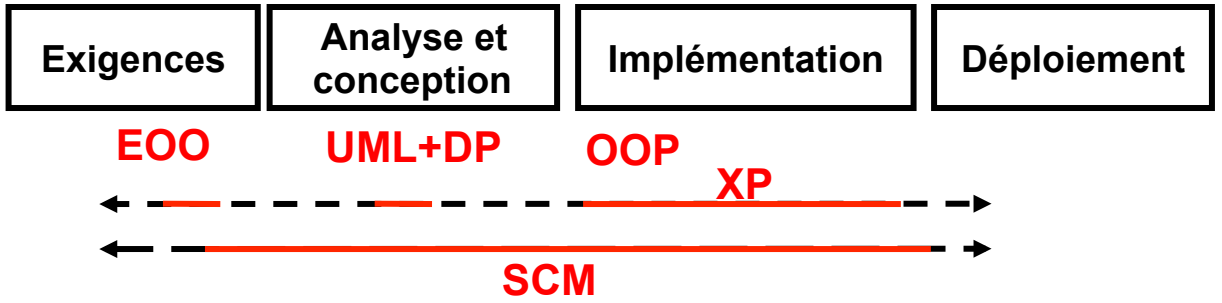
MaIL

6 – Itinéraire d'un objet gâté

Argument

- Constat
  - Les méthodologies agiles semblent répondre à la problématique du changement, mais …
- Questions
  - Quelle courbe de l'apprentissage ?
  - Ne sont-elles pas trop nombreuses ?
  - Comment les articuler efficacement ?
  - Retour sur investissement dès le premier usage (projet, exemple, cours) ?
  - ROI étudiant ?
  - Sont-elles **vraiment** agiles ?
  - Ce cours n'est-il pas dogmatique à son tour ?
- Objectifs
  - Synthétiser, articuler (colle et compléments), simplifier, démystifier & démasquer les rigidités restantes, illustrer des perspectives (technologies adaptées) et vous libérer du poids dogmatique de nos enseignements éphémères

Plan



6. Itinéraire d'un objet gâté[1]

6.1 Modélisation agile
- UML : exemple, méta, plan de route minimaliste, couleurs Coad / archétypes, stéréotypes / navigation IHM
- Metapatterns [Pree]. Exemple évolutif : startup.

6.2 Implémentation agile
- Correspondance UML-Java
- Métriques, exemple de découplage : Loi Demeter
- Modéliser une BD relationnelle : placement / Newton ;-)
- Technologies agiles : AOP, MDA, XUML, DCC

6.3 Conclusion : bilan et perspectives
- Bilan et perspectives de l'agilité
- Nos attentes & devoir
- [Biblio : le kit de survie]

(1) toute ressemblance avec des personnes physiques suivant ce cours est évidement fortuite ☺

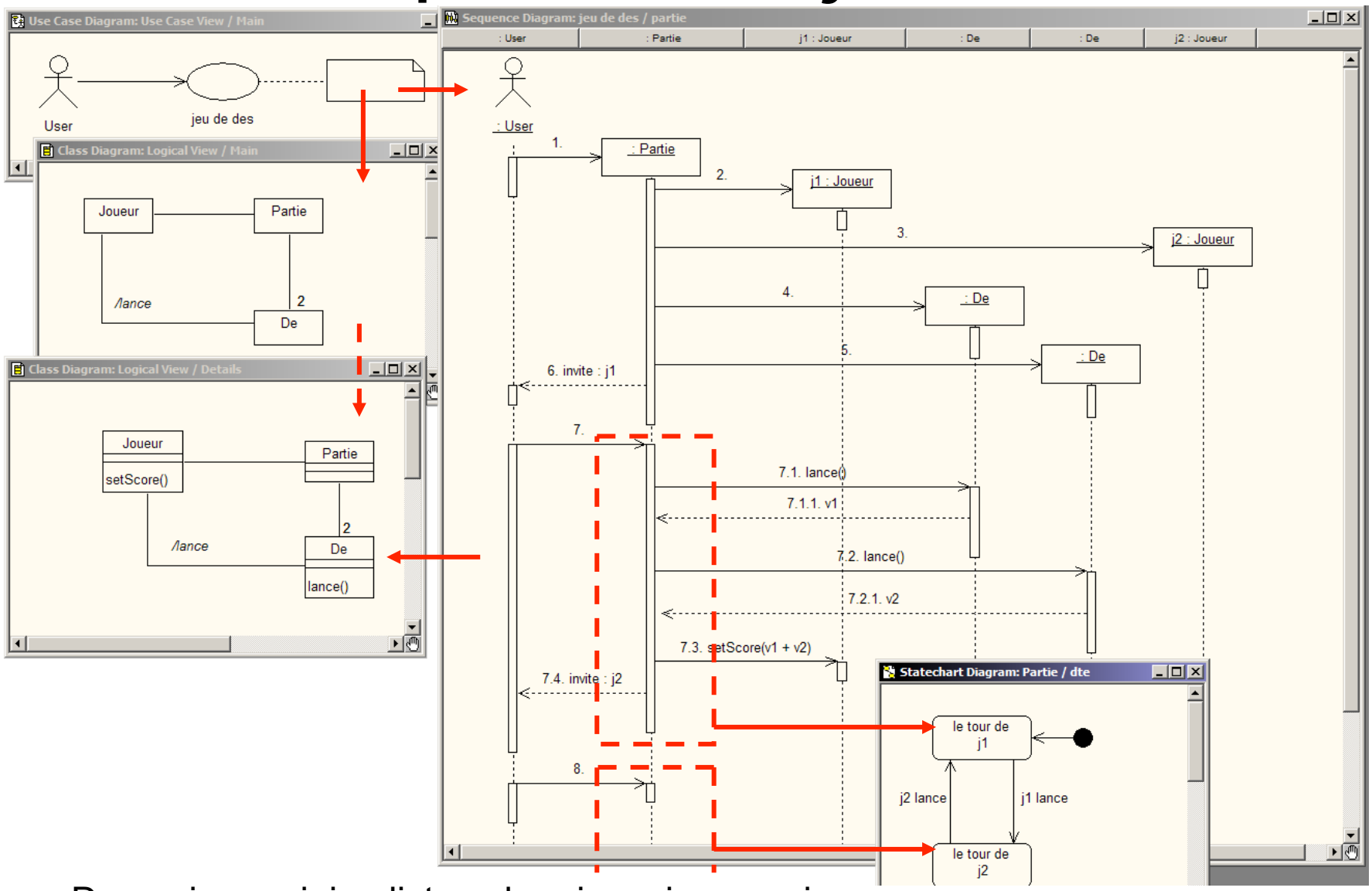
6.1 Modélisation agile

UML & DP

UML

- Complexité ?
  - Dans UML 1.1 => 123 concepts (classe de MM)
  - Comment apprendre et retenir rapidement et efficacement les symboles et les concepts UML ?
- Mes astuces
  - Représenter visuellement (dessins) : parce que je le vaux bien.
  - Effectuer / tourner des exemples
  - Métamodéliser : modéliser les concepts et leurs connexions (UML), même si c'est incomplet
  - Placement de symboles unifiés et adaptés (loi de Newton)
  - Couleurs de Peter Coad. IHM : diag. robustesse
  - Articuler avec les éléments connus : java => 6.2 (implem agile)

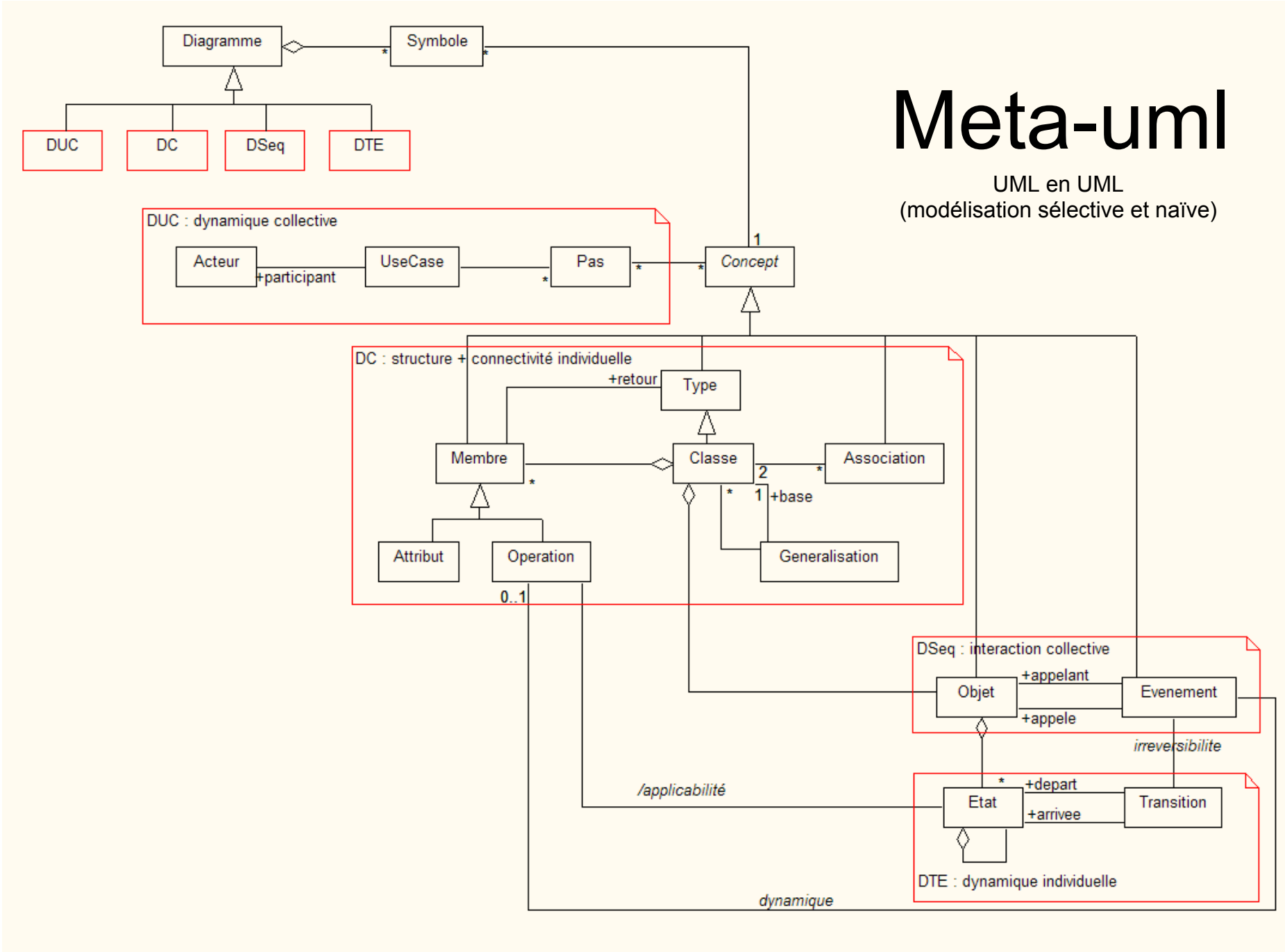
Exemple UML : jeu de dés



- Dynamique minimaliste : chemins microscopiques

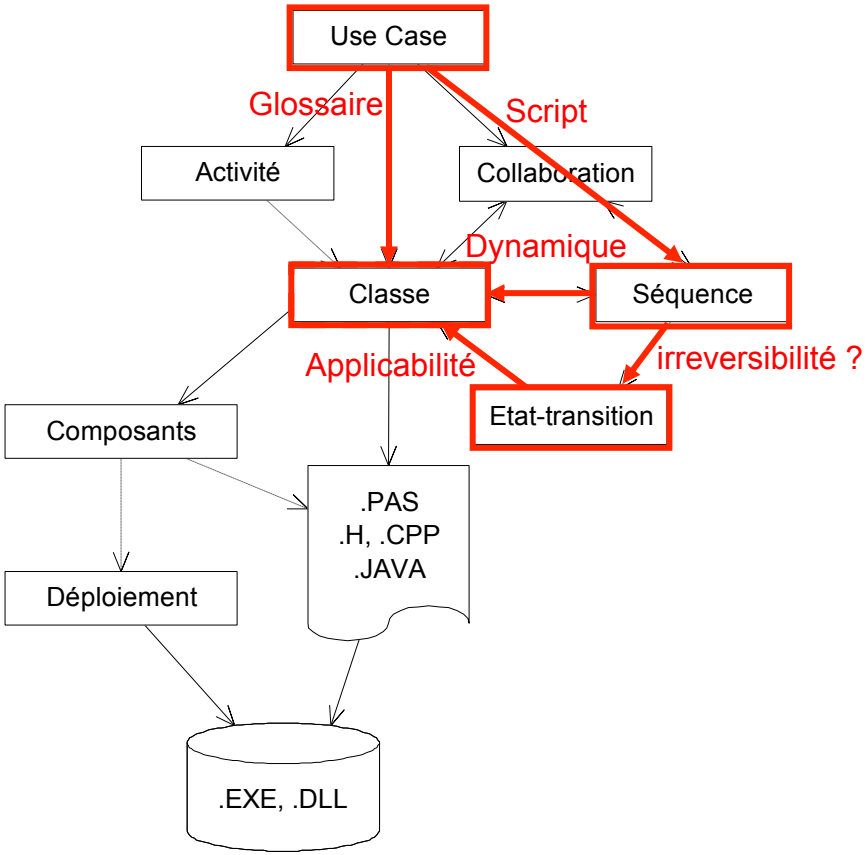
Meta-uml

UML en UML
(modélisation sélective et naïve)

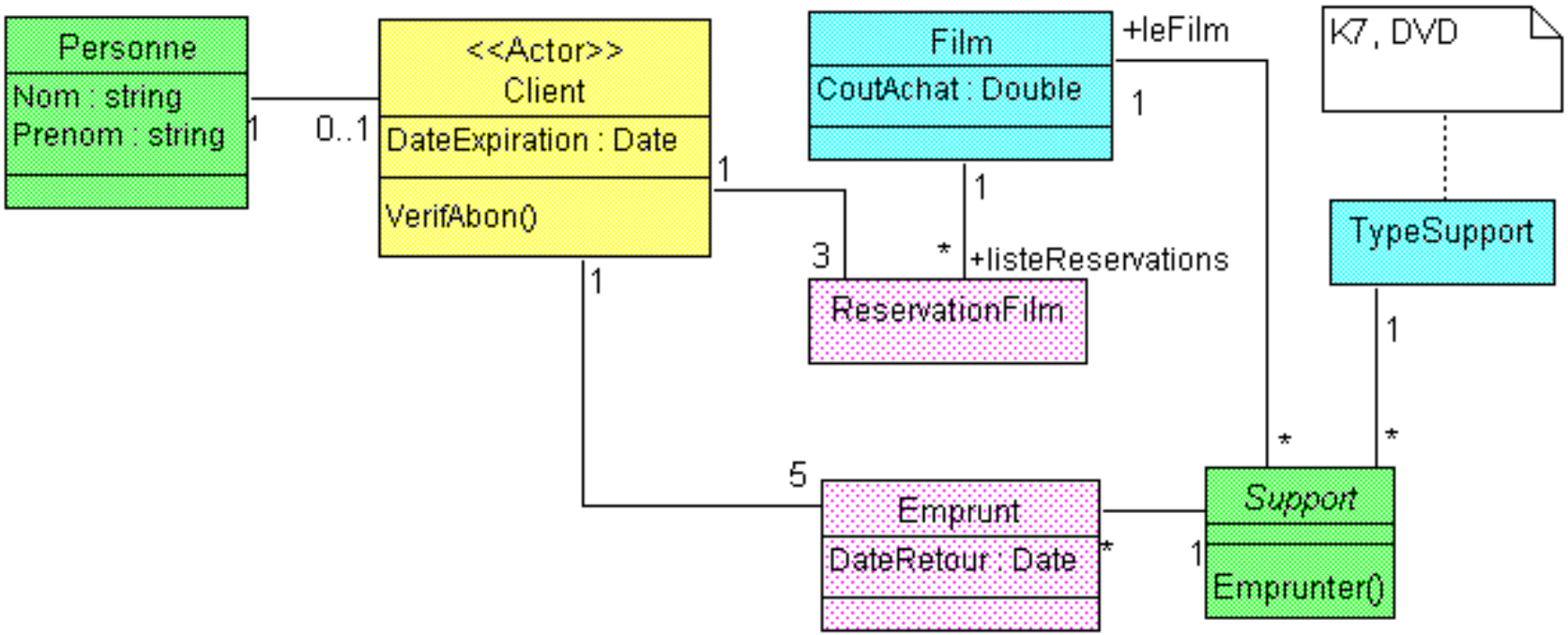


Un plan de route minimaliste

Répertorier les interactions typiques (acteurs & use cases / scripts)



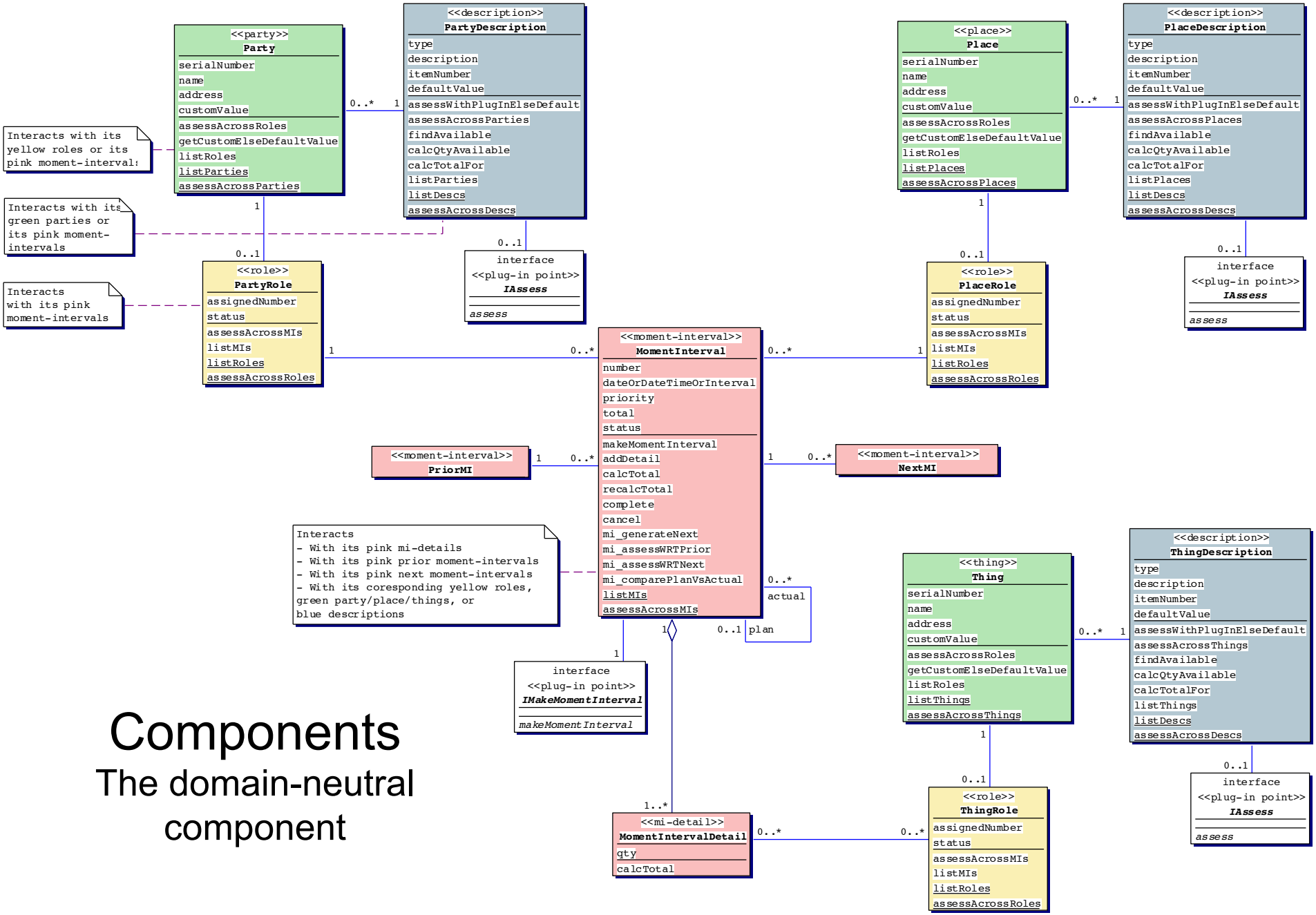
Couleurs de Peter Coad



A green party-place-thing
A yellow role
A pink moment-interval
A blue description

Structuration / découpage (Personne vs. Client) par nature : archetype
Méthodes & attributs prédéfinis (date, etc.)
Met de l'ordre dans le diag de classe
Accessible pour le client
Facilite la lecture / vérification
Adapté à la modélisation du métier => contexte
Indépendant du métier / problème => technique réutilisable

http://www.togethersoft.com/services/tutorials/jmcu/

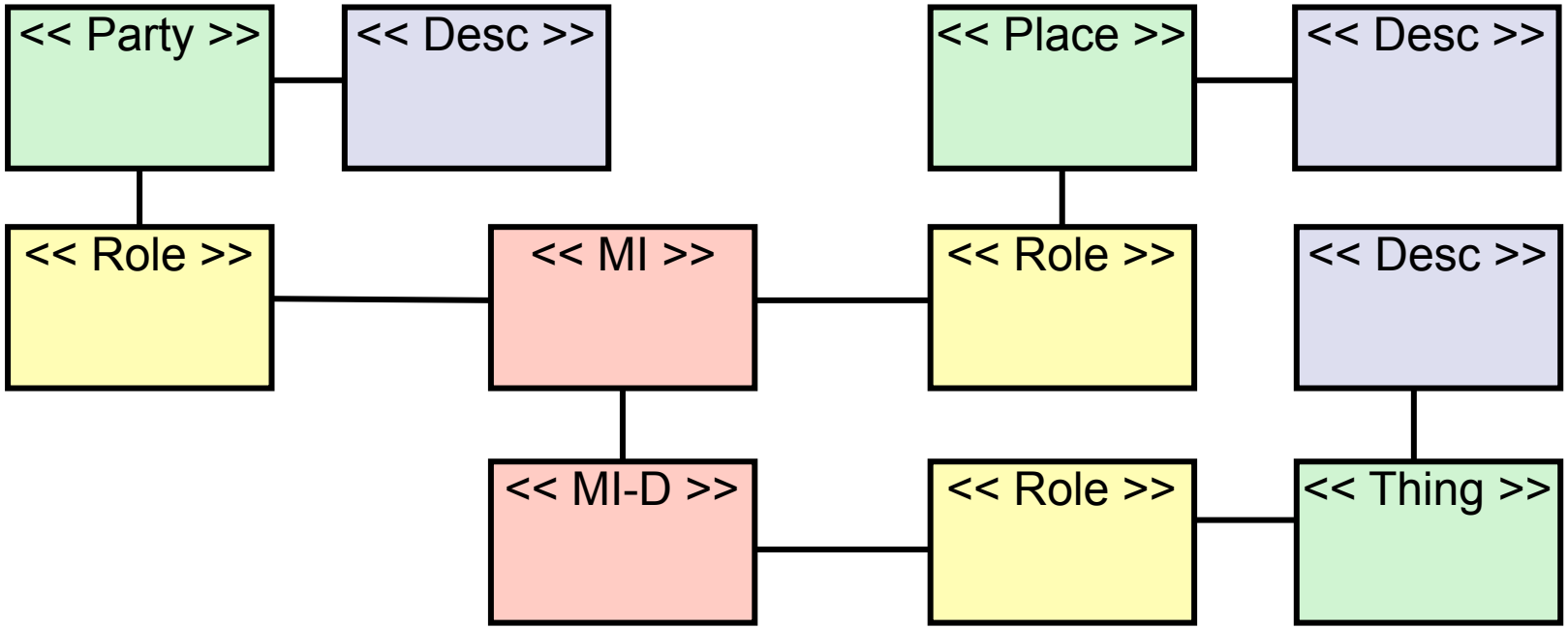


Components

The domain-neutral component

Exercice

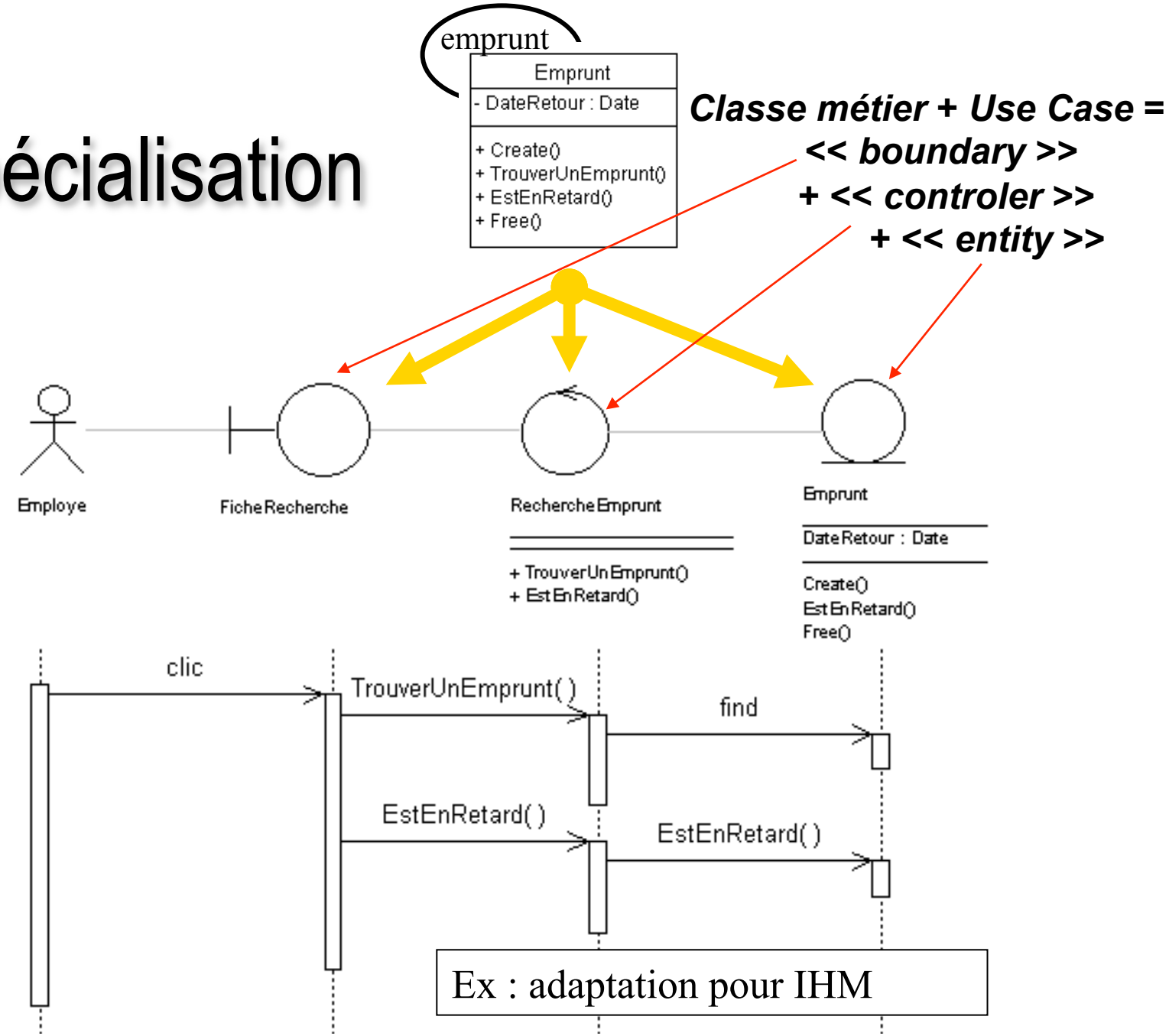
- Afin de tester la pertinence des archétypes, proposer des noms de classes pour le diagramme anonyme suivant :



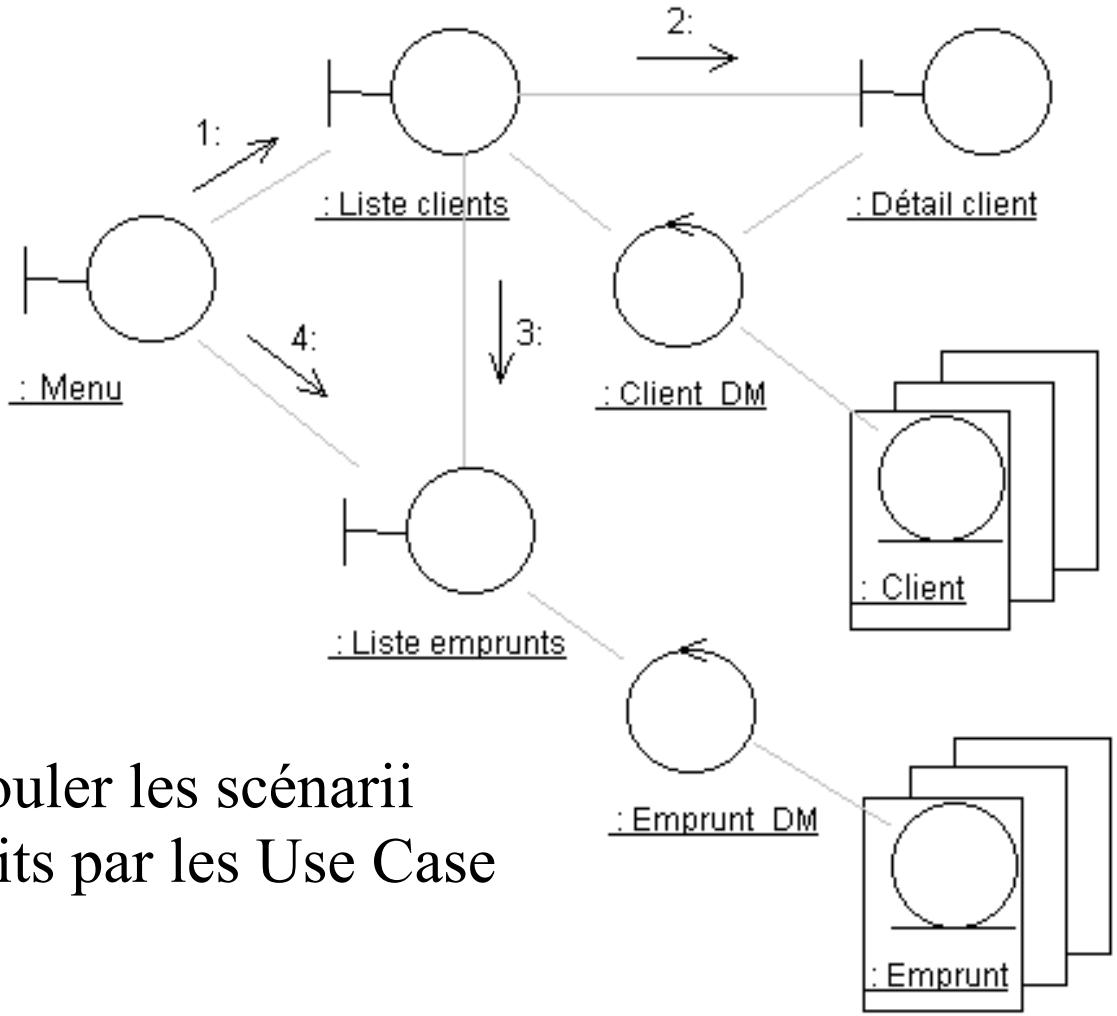
Les 10 meilleures façons de rater un projet objet [Rosenberg]

10. Commencez par un projet critique.
9. Assurez-vous qu'il n'y a pas de compétences OO dans votre équipe
8. Ne testez surtout pas les classes unitairement, ne testez que l'intégration et croisez les doigts.
7. Confiez la réalisation des use cases aux expérimentés et les séquences aux débutants.
6. Ne perdez pas le temps avec la revue des modèles.
5. Séparez soigneusement use case et modèle de classe. Nous savons tous que les UC n'affectent pas le code.
4. Implémenter les parties les plus faciles aux début et laissez les plus difficiles à la fin, juste avant le terme.
3. Prenez une équipe de 20 développeurs VB, confiez-leur un compilateur C++ et un outil visuel de modélisation puis laissez-les se débrouiller.
2. Ignorez le modèle d'analyse et de conception. Ecrivez le code, et générez un modèle global par reverse engineering. Personne ne s'en appercevra.
1. Partez de l'idée que l'outil de modélisation générera un excellent code pour vous et embauchez plein de stagiaires non diplomés pour s'occuper du codage.

Spécialisation



Spécialisation - IHM



Dérouler les scénarii
décrits par les Use Case

Metapatterns

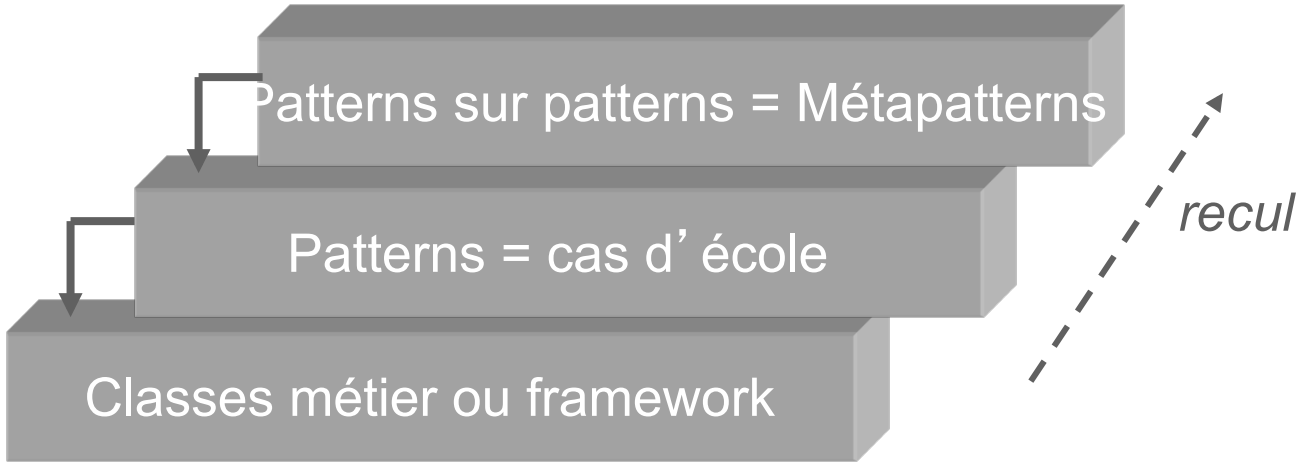
- La généralisation de la modélisation objet
  - Contexte actuel : frameworks et objets métier
  - Qualités visées pour les modèles
    - Généralité, robustesse, flexibilité
    - Réutilisabilité, maintenabilité, évolutivité
  - Résultat : notoriété croissante UML et Design Patterns (DP)
- Malgré tout, des obstacles subsistent
  - Approche souvent dogmatique
  - Solutions « miraculeuses »
- Cette section
  - Démystification : principes fondateurs des DP
  - Montre comment bien modéliser avec encore moins d'effort … à travers les *metapatterns*
    - Prétexte => *patterns*

Classification des patterns

- Analyse
  - Peter Coad, Martin Fowler
- Conception
  - Gamma, Vlissides
- Programmation
  - Coplien, Beck
- Anti-patterns
  - Mauvaises solutions
- Meta-patterns
  - Aspect rédactionnel [Meszaros/Doble98]
  - **Aspect modélisationnel** [Pree95]

Metapatterns [Pree95]

- Méta-X
  - Au dessus, avec recul, plus abstrait, mais reste un X à son tour par rapport au niveau inférieur

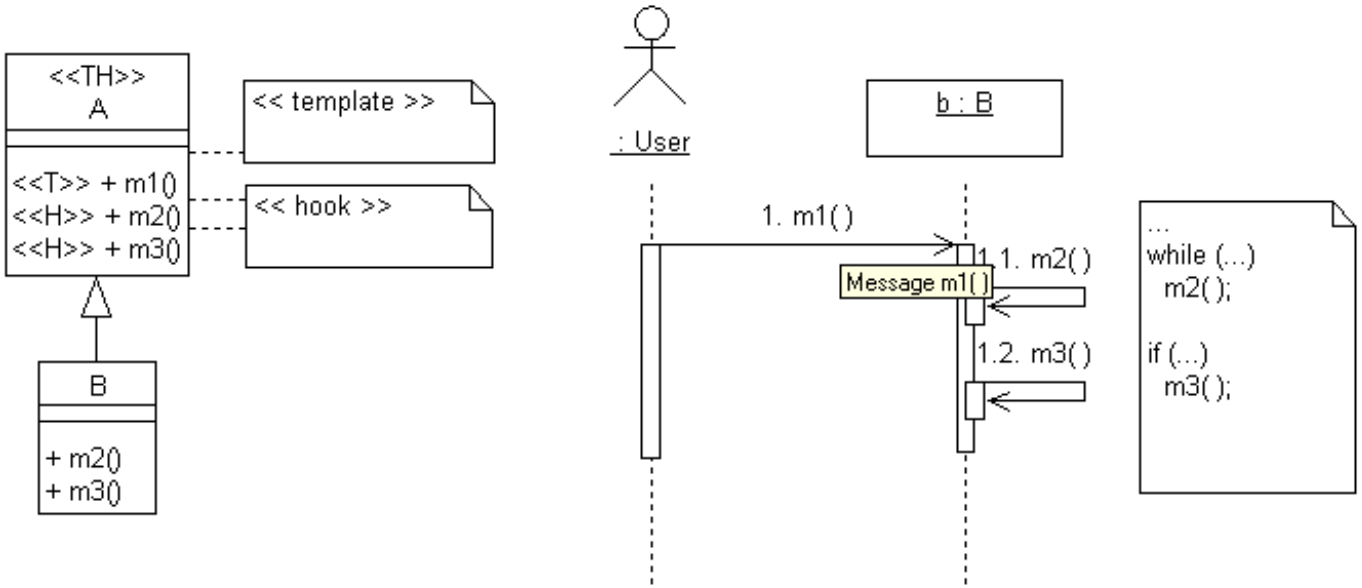


- Matière première des métapatterns
  - Classes, méthodes, associations, appels de méthode
  - Granularité fine : blocs élémentaires de conception
- Avantage visé (rappel)
  - « Sagesse » hyper-concentrée et ré-applicable plus aisément, sans dogme

Métapatterns

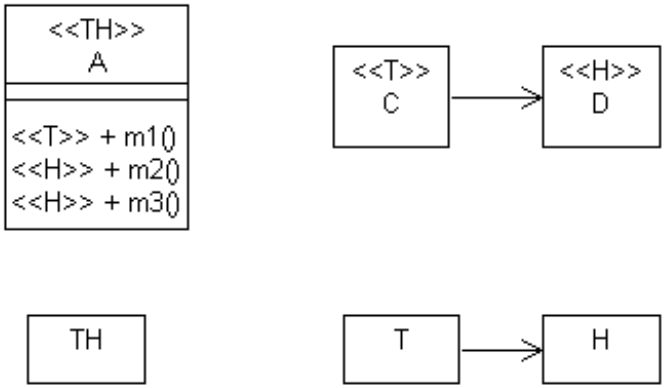
- Repère fondamental modélisation/patterns
  - Distinction entre stabilité et variabilité
- [Pree]
  - Points névralgiques dans un framework
    - Points rigides : méthodes « template »
    - Points flexibles : méthodes « hook »
  - Hooks: points prédéfinis de spécialisation
    - Whitebox : par héritage (et redéfinition de méthode) ou par la réalisation d'une interface (Java) => par le bas
    - Blackbox : par composition (et délégation) => par le côté

Exemple [Pree99]

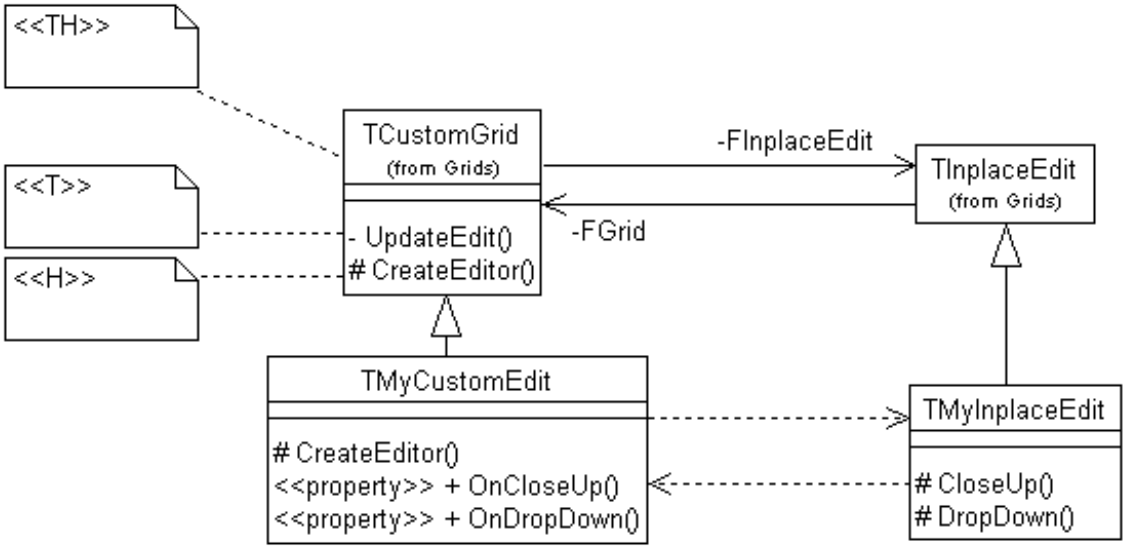


- m1 : template (méthode complexe)
  - La pièce rigide
- m2, m3 : hooks (points d'extensibilité)
  - Delphi : override;

- Conventions
  - « T » et « H »
  - Méthodes
  - Classes
- Variations
  - A. Unification vs. Séparation
  - B. Récursivité (ou pas)
- Métapattern
  - Toute combinaison de T-H selon les variations citées



Unification TH

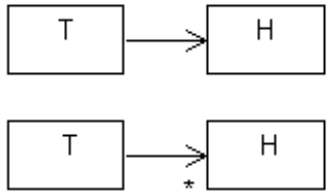


- Factory method (VCL)

  « Definir une interface pour créer un objet, mais laisser les sousclasses décider quelle classe instancier » [GoF]

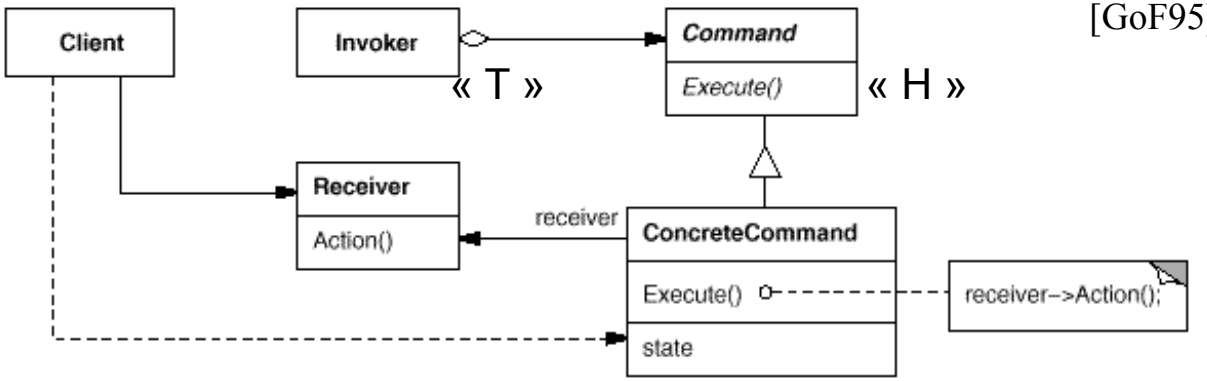
- Autre exemple : TTreeView en VCL

Séparation



- Exemples
  - Abstract factory, Bridge, Builder, Command, Interpreter, Observer, Prototype, State et Strategy
    - 1:1 vs. 1:N
- Commande [GoF95]

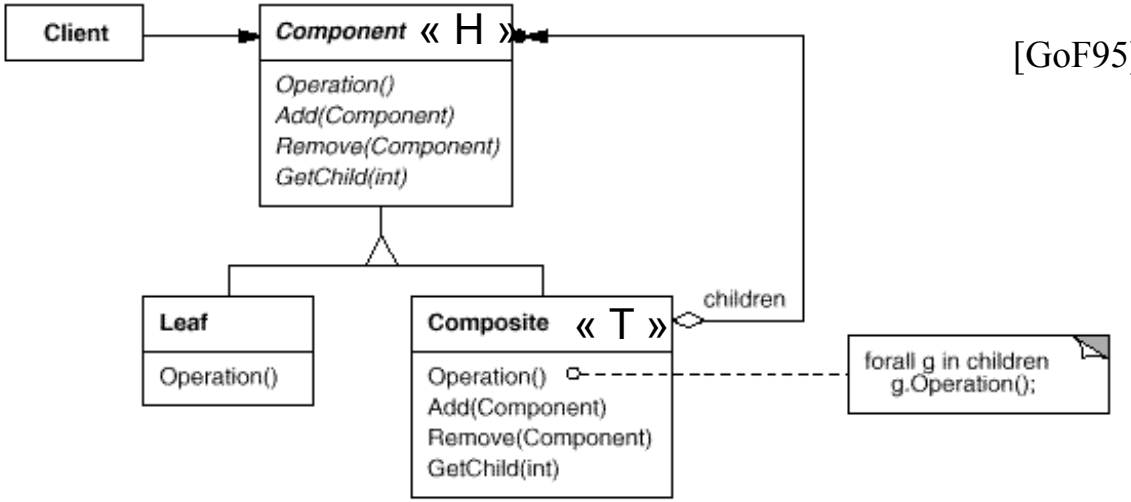
Rappel : les noms des DP G => selon la variabilité (H)



[GoF95]

Récursivité

- Le composite [GoF95]
  - Composer des objets en structures arborescentes pour représenter des hiérarchies « ensemble-partie ».
  - Le composite autorise les clients de traiter des objets individuels et des compositions d'objets de façon uniforme.



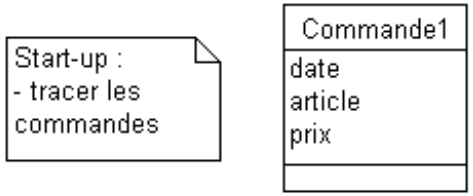
[GoF95]

- Participants
  - composant, composite, feuille et client

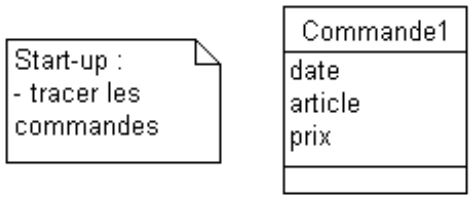
Stratégie d'application des MP

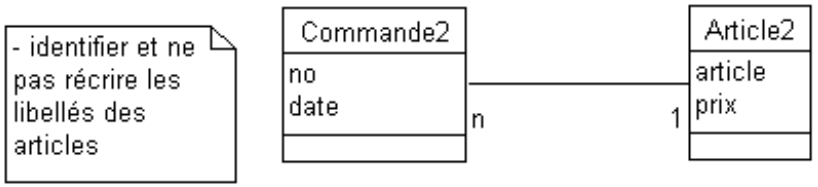
- Bilan métapatterns
  - « Une méthode (T) appelle une autre (H) ». Pffff ☺
- Utilité
  - Application microscopique des principes des DP
  - Mieux découvrir, comprendre, reconnaître et appliquer les DP
  - C'est tout ?
- Le quotidien du développement objet
  - Organisation initiale vs. évolution coûteuse
- Minimiser le poids de l'investissement initial
  - Découpage en classes, accesseurs, interfaces, etc.
- L'opportunisme éclairé appliqué aux métapatterns
  - Evolution incrémentale
  - Refactoring : « Just-in-time modeling »
- Un cas d'école …

Phase 1. Start-up



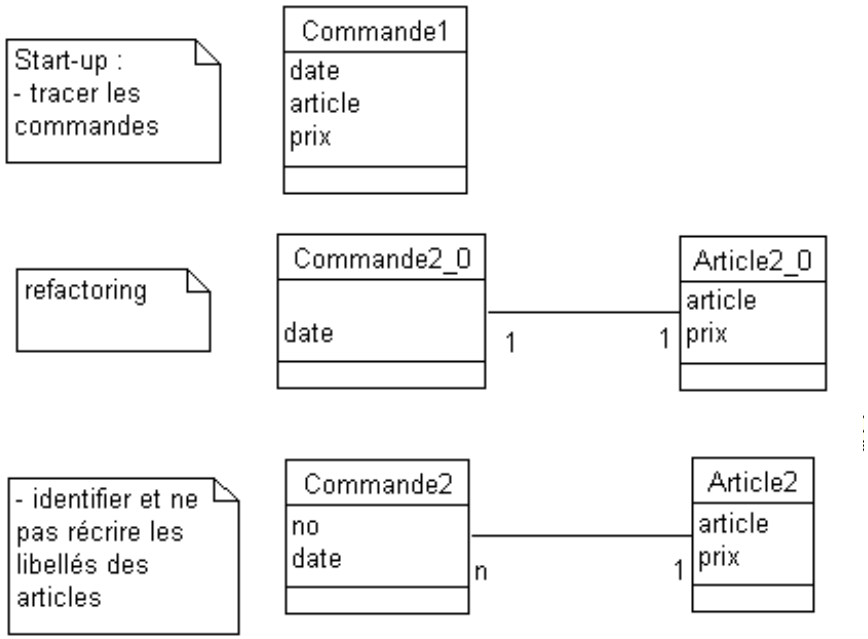
Phase 2. Start-up++





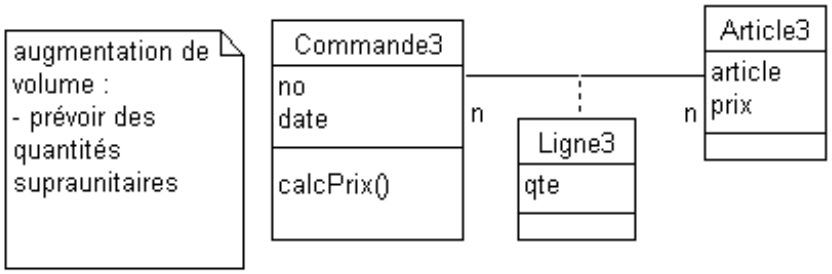
Réproductibilité

Phase 2. Start-up++



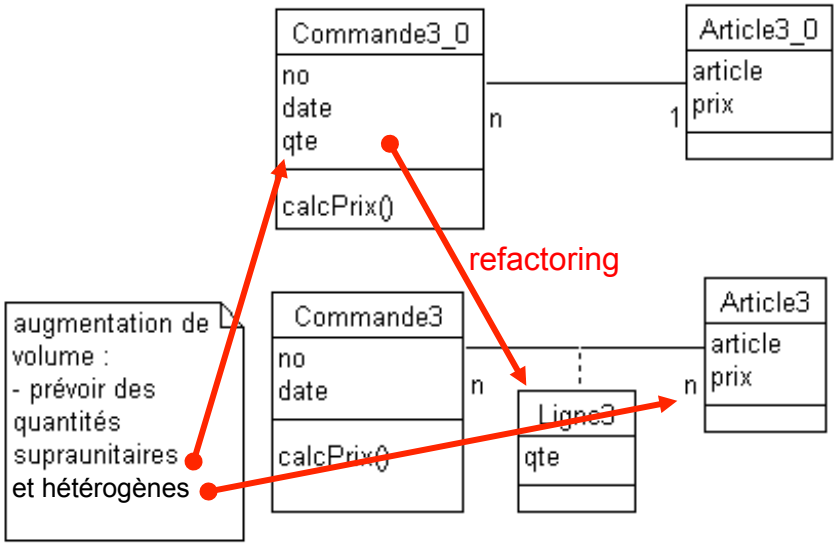
Réproductibilité

Phase 3. Développement



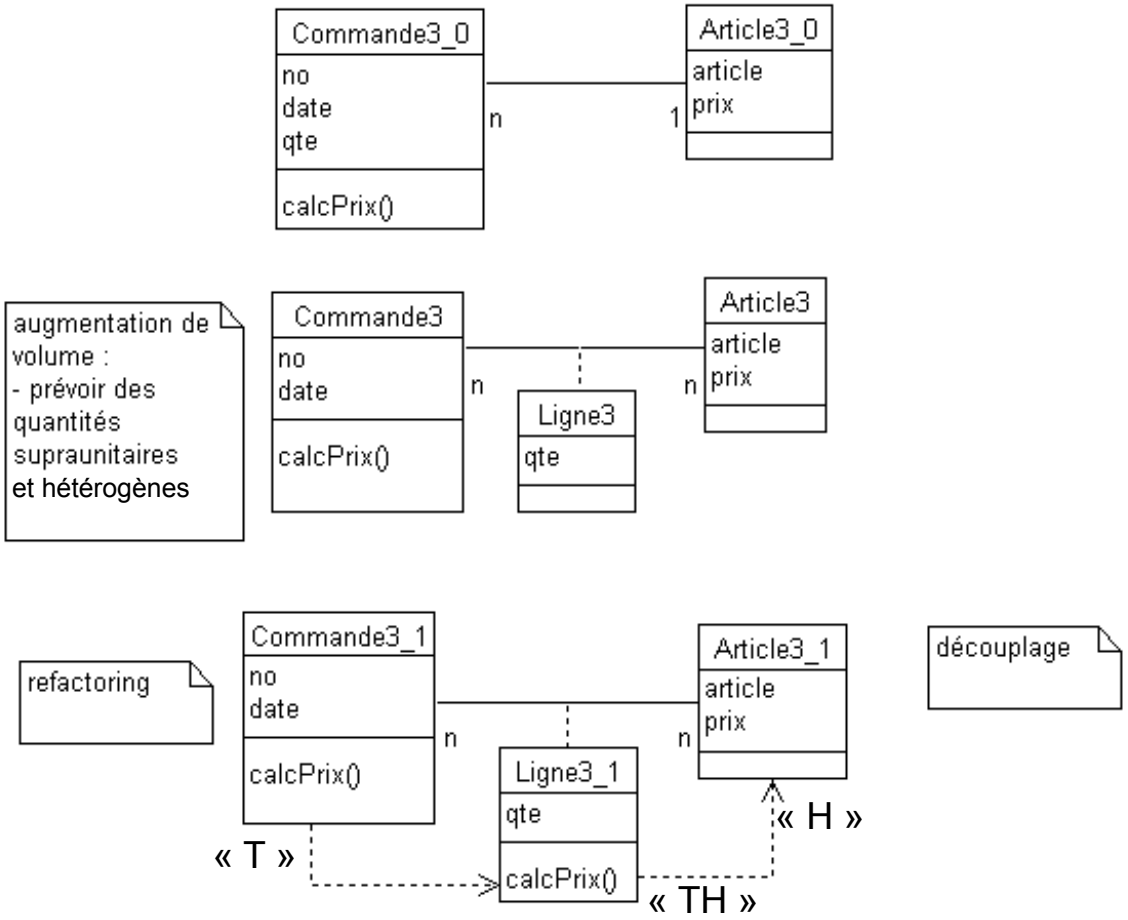
Industrialisation

Phase 3. Développement



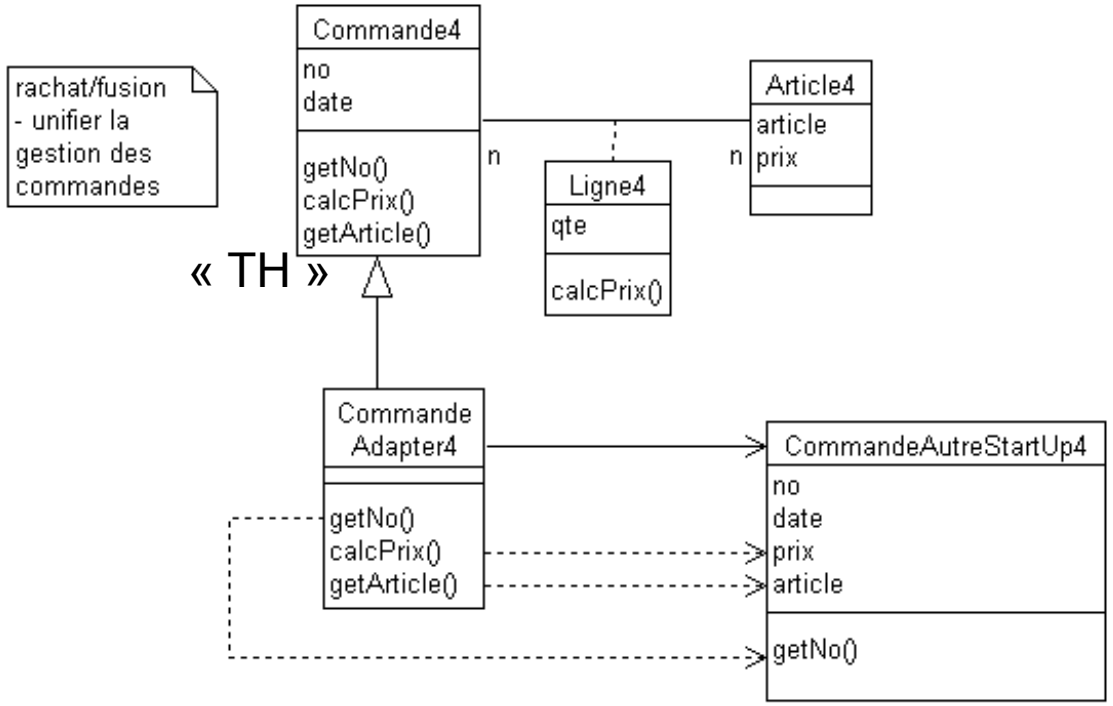
Industrialisation

Phase 3. Développement

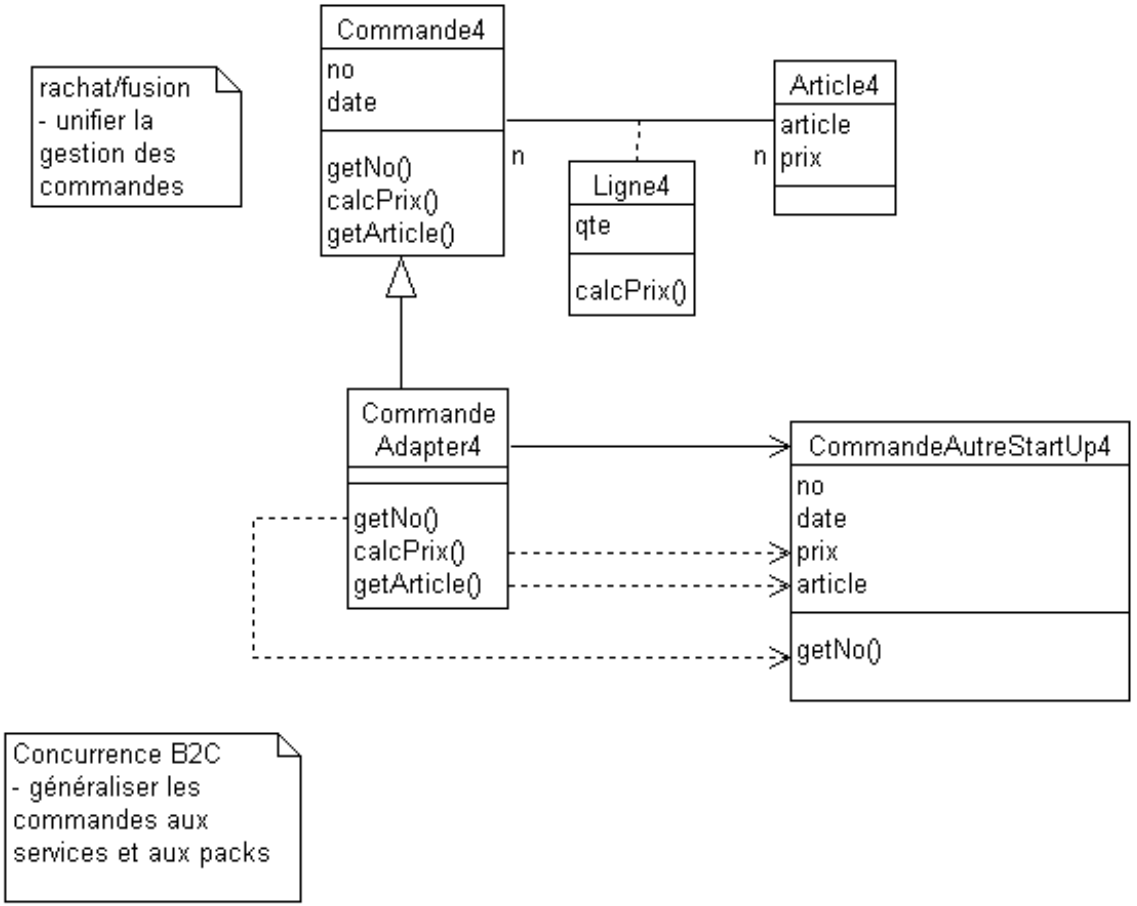


Industrialisation

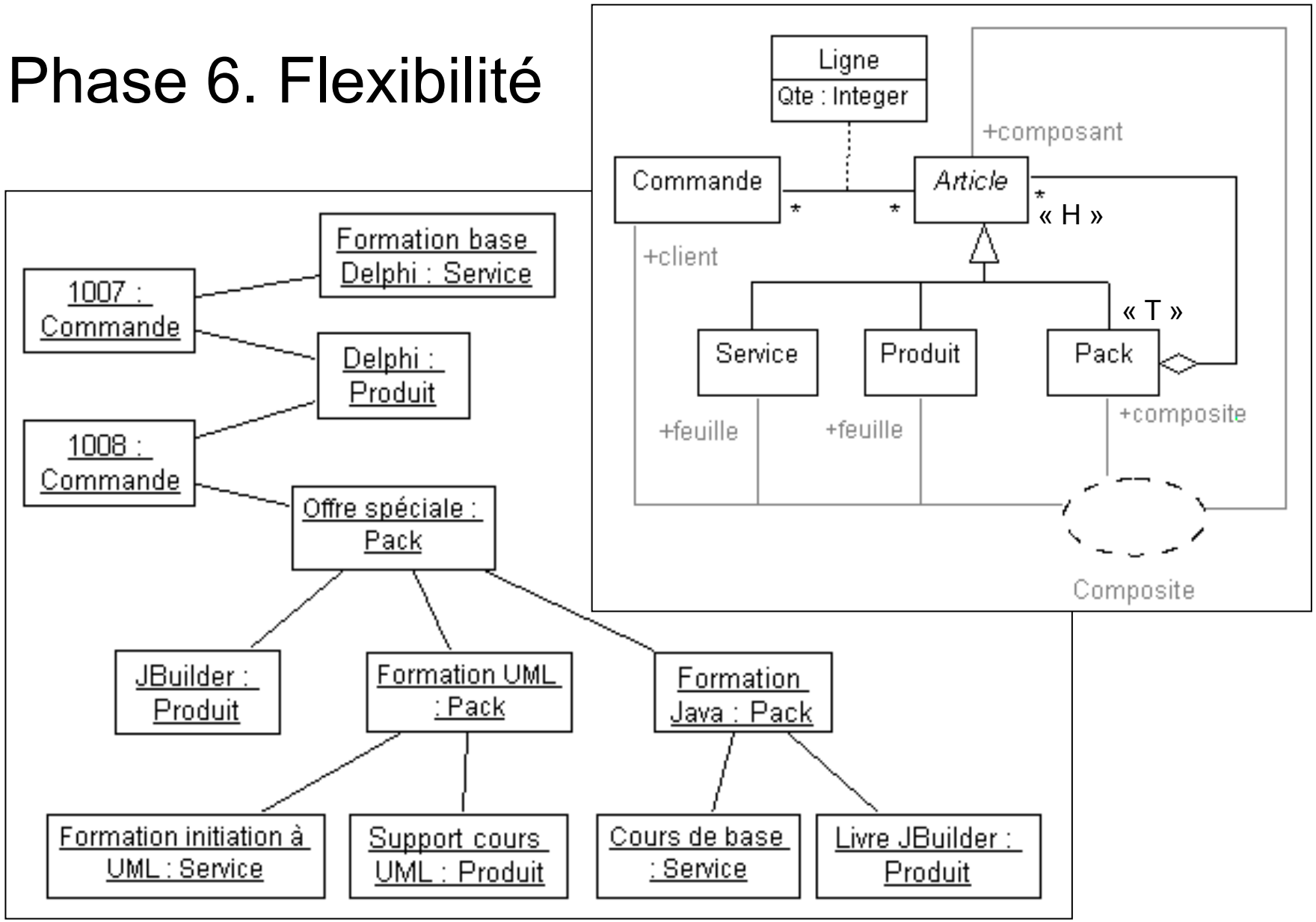
Phase 4. Fusion-rachat



Phase 5. Vers la mondialisation



Phase 6. Flexibilité



Traduction en métapatterns

- Evolution des classes
  - TH => T-H => T-TH-H => T-TH-|>-H
- Transformations opportunistes
  - Refactoring => préservation de sémantique
  - Avant ou après changement de spécification
- Remise en cause des points rigides ?
  - Même technique !

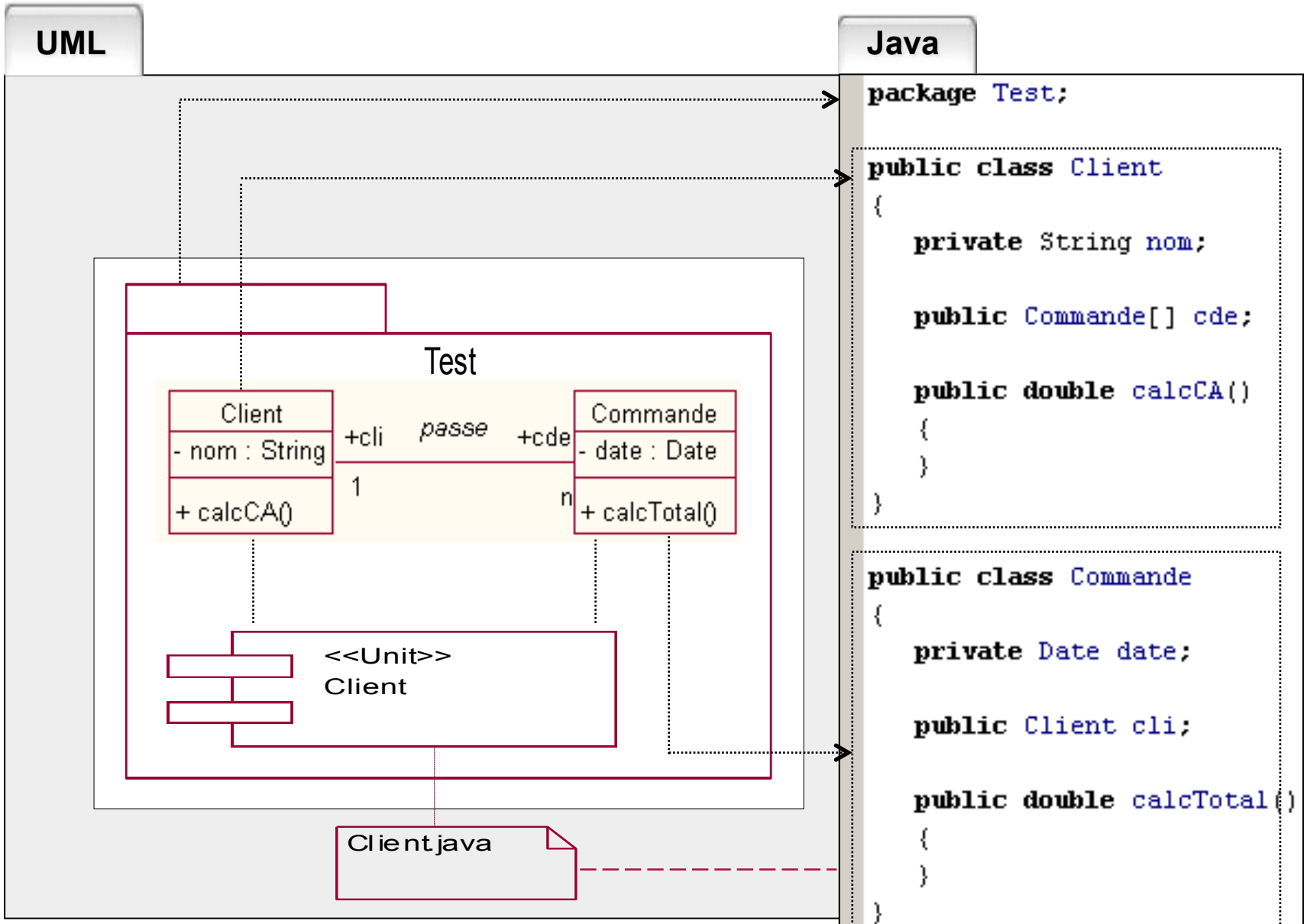
Conclusion MP

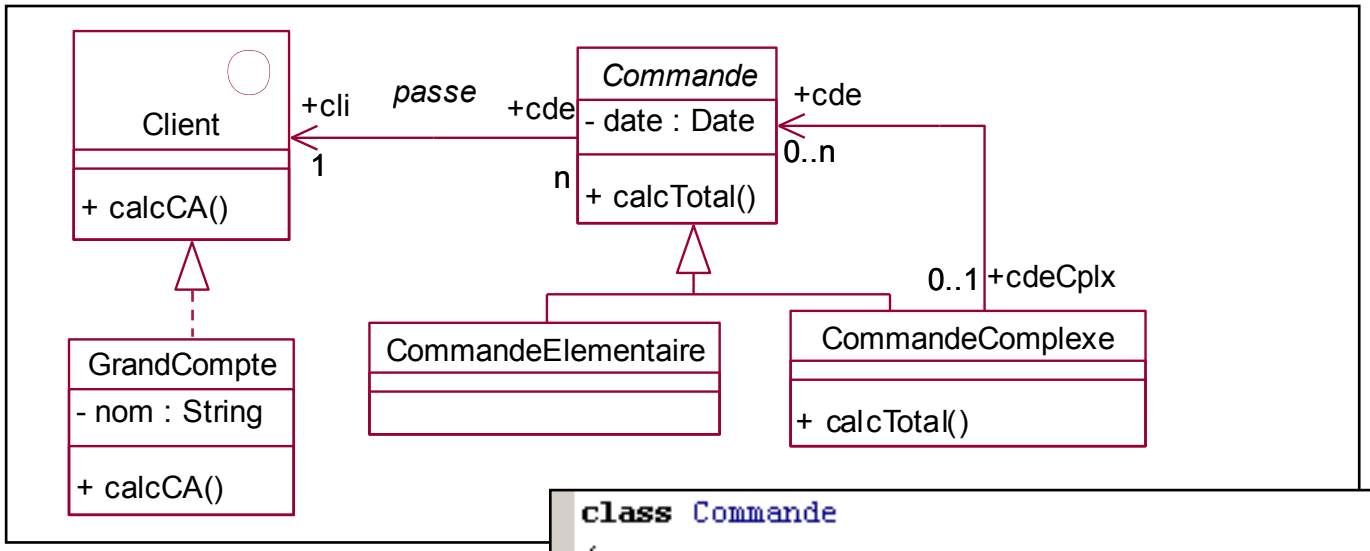
- Modélisation objet des frameworks
  - Pâte à modeler
    - gestes élémentaires avec effet progressif
  - Les éléments (méthodes et classes)
    - N'ont pas de densité uniforme
- Classification par stabilité (T ou H)
  - Assumée a priori, mais …
  - … peut être remise en cause
    - Les « T » peuvent devenir des « H »
    - Mais … au « bon » moment ☺

6.2 Implémentation agile

Du modèle au code : UML – Java
Métriques (Loi Demeter)
BD relationnelle
Technologies agiles : AOP, MDA, XUML

Correspondances UML-Java






```java
package Test;

public interface Client
{
   double calcCA();
}

public class GrandCompte implements Client
{
   private String nom;

   public double calcCA()
   {
      return 0.0;
   }
}
```

```java
class Commande
{
   private Date date;
   public Client cli;
   public double calcTotal() { }
}

public class CommandeComplexe extends Commande
{
   public Commande[] cde;
   public double calcTotal()
   {
      return 0.0;
   }
}

public class CommandeElementaire extends Commande
{
}
```

Correspondances UML-Java

UML

- Classe
- Attribut
  - Tag values
- Opération
- Interface
- Association
  - Nom
  - Rôle
  - Multiplicité (conteneur)

Java

- Classe
- Variable (field)
  - Set/Get
- Méthode
- Interface
- Pas de lien direct
  - Rien !
  - Variable (si navigable)
  - Valeur unique / multiple (tableau, vecteur, …)

Correspondances UML-Java

UML

- Généralisation
- Réalisation
- Classe d'association
  - Transformation préliminaire => deux associations
- Tag-values ⬅======

Java

- Extends
- Implements
- Autres éléments

Métriques

- Découplage : Loi de Demeter
  - **loi de Demeter** - (1987) Ian Holland
  - Le résumé en est *Do not talk to strangers.*

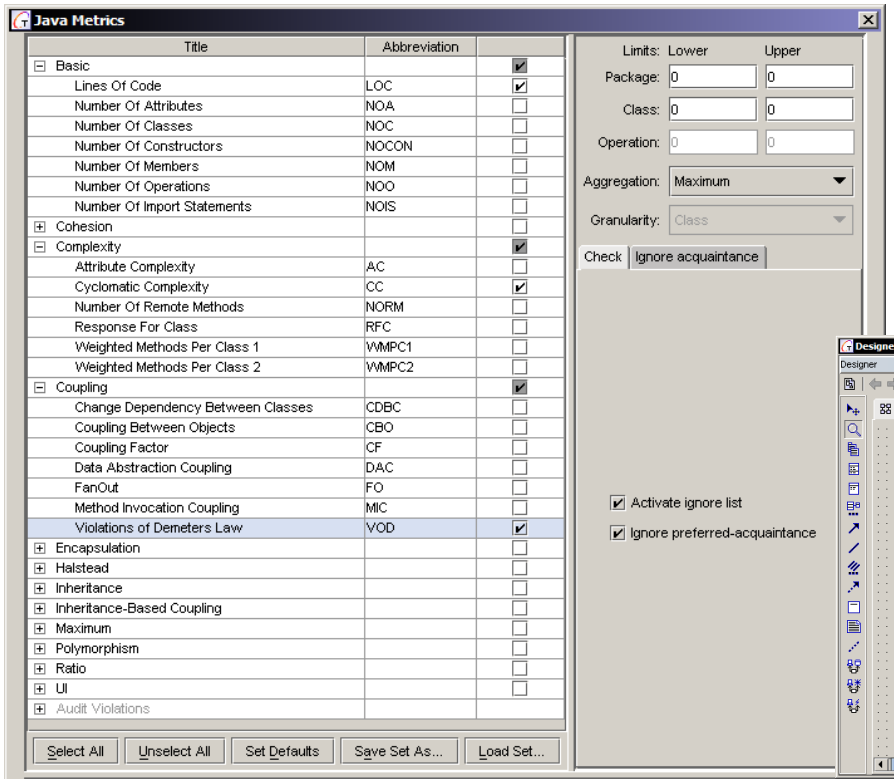
```
class Demeter {
    private :
      A *a;
      int func() { … }
    public
      example(B& b) {
        C c;
        int f = func();
        b.invert();
        a = new A();
        a->setActive();
        c.print();
    }
}
```

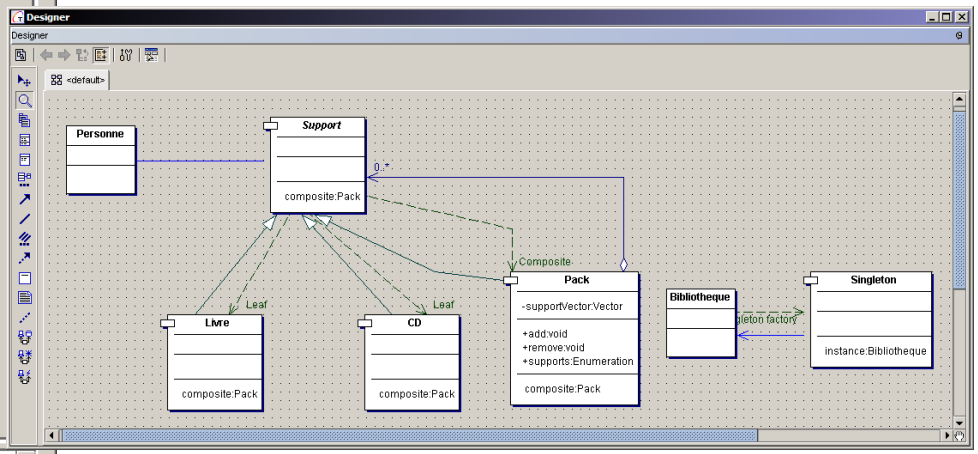
La loi du Demeter pour les fonctions
Toute méthode d'un objet ne doit appeler que des méthodes apparenant :

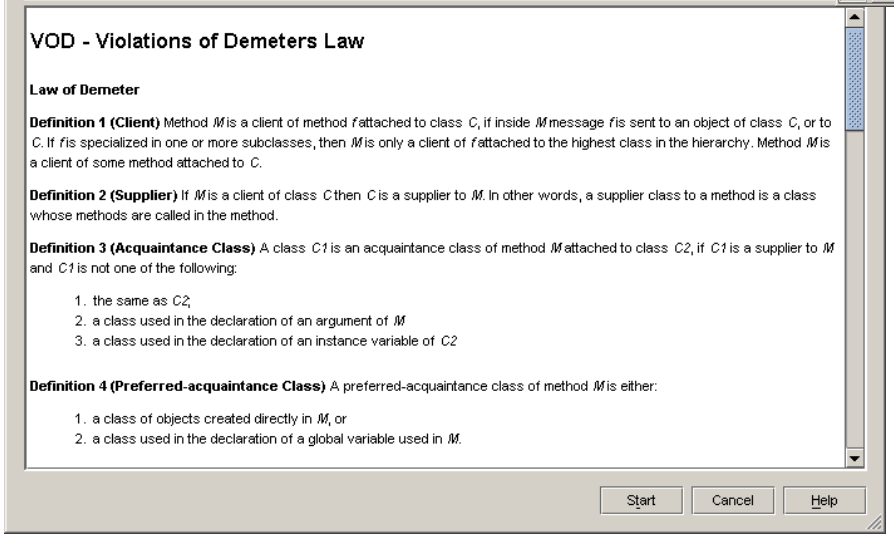
- à soi-même
- aux objets reçus en paramètre
- à tout objet créé par soi-même
- à tout objet possédé directement

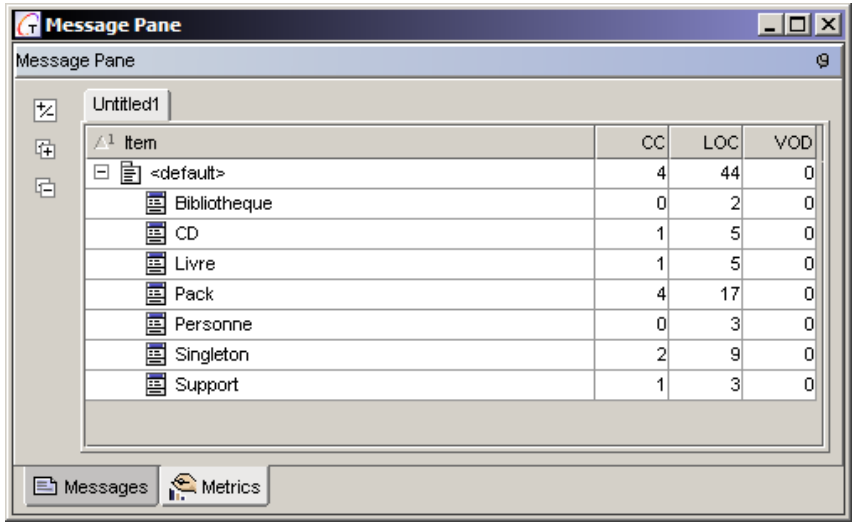
www.pragmaticprogrammer.com

TCC / Métriques

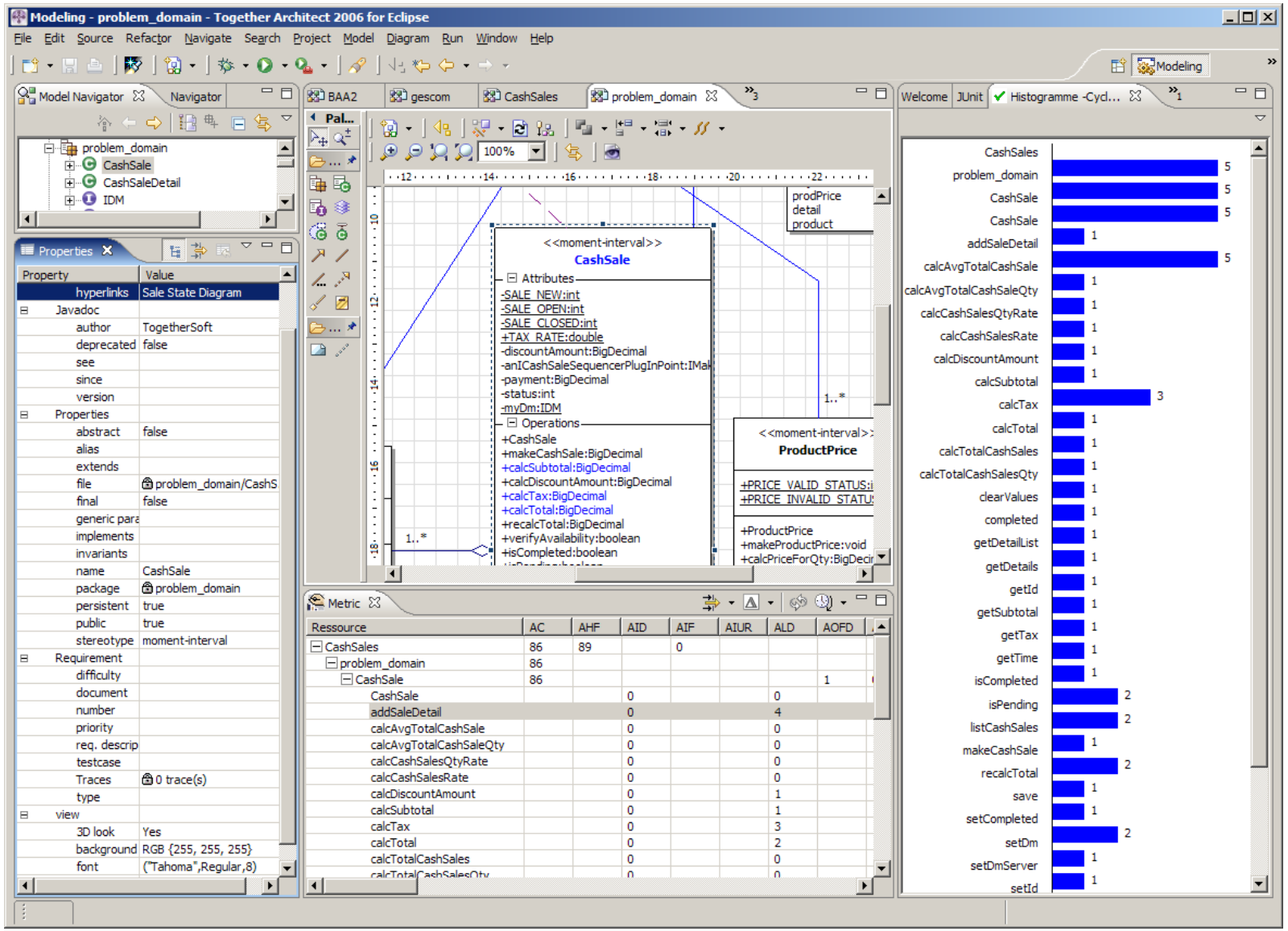






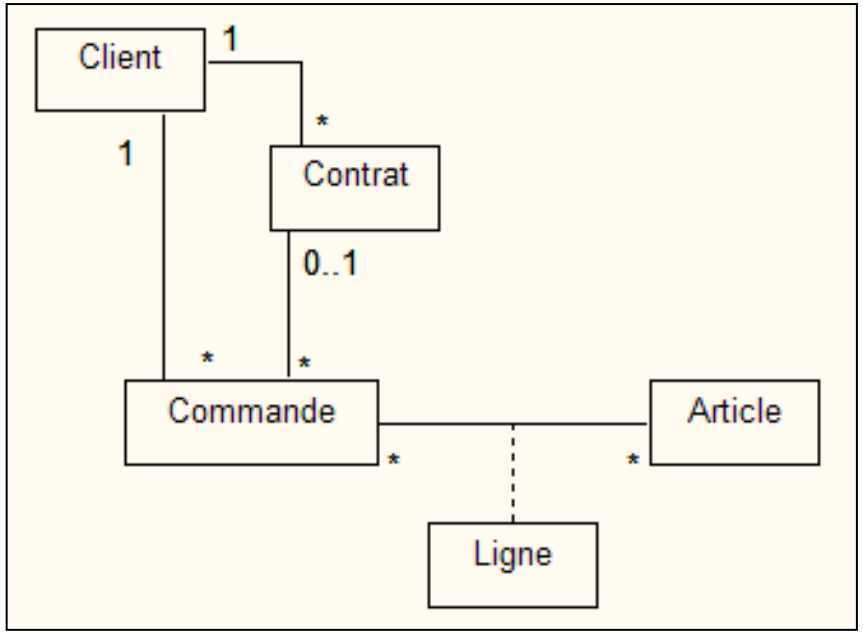


Métriques orientées objet



Modélisation d'une BD relationnelle

- Tables
  - Attributs et clés (pk)
- Références
  - Clés étrangères (fk)
- Placement
  - Loi de Newton [MZ]
- Mapping O/R
  - Associations
    - 1:1 => même table (ou pas)
    - 1:N => tables distinctes
    - N:N => table d'association
  - Ambler, Keller
    - http://www.agiledata.org/essays/mappingObjects.html
  - Outils
    - TopLink, JDO, Hibernate

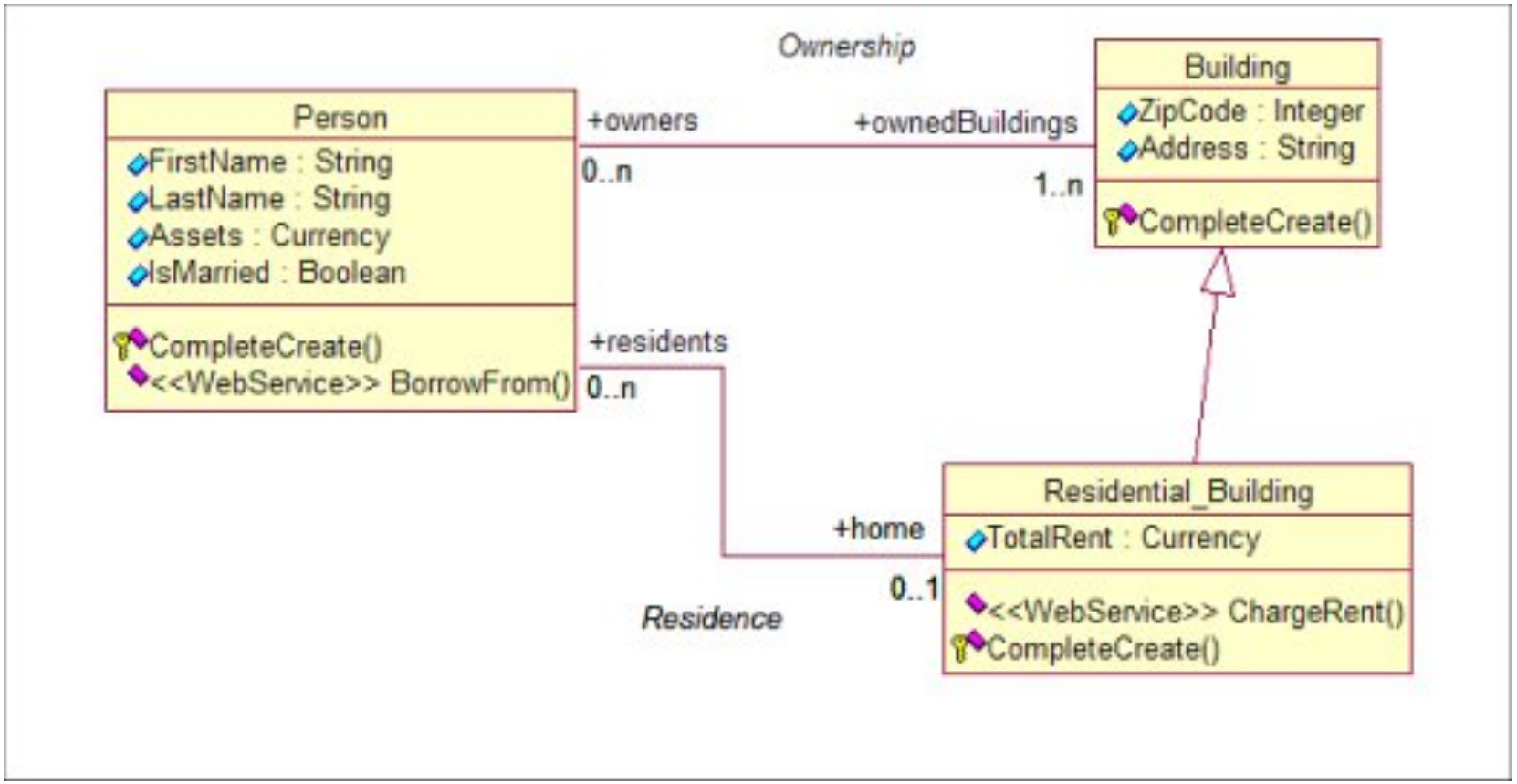


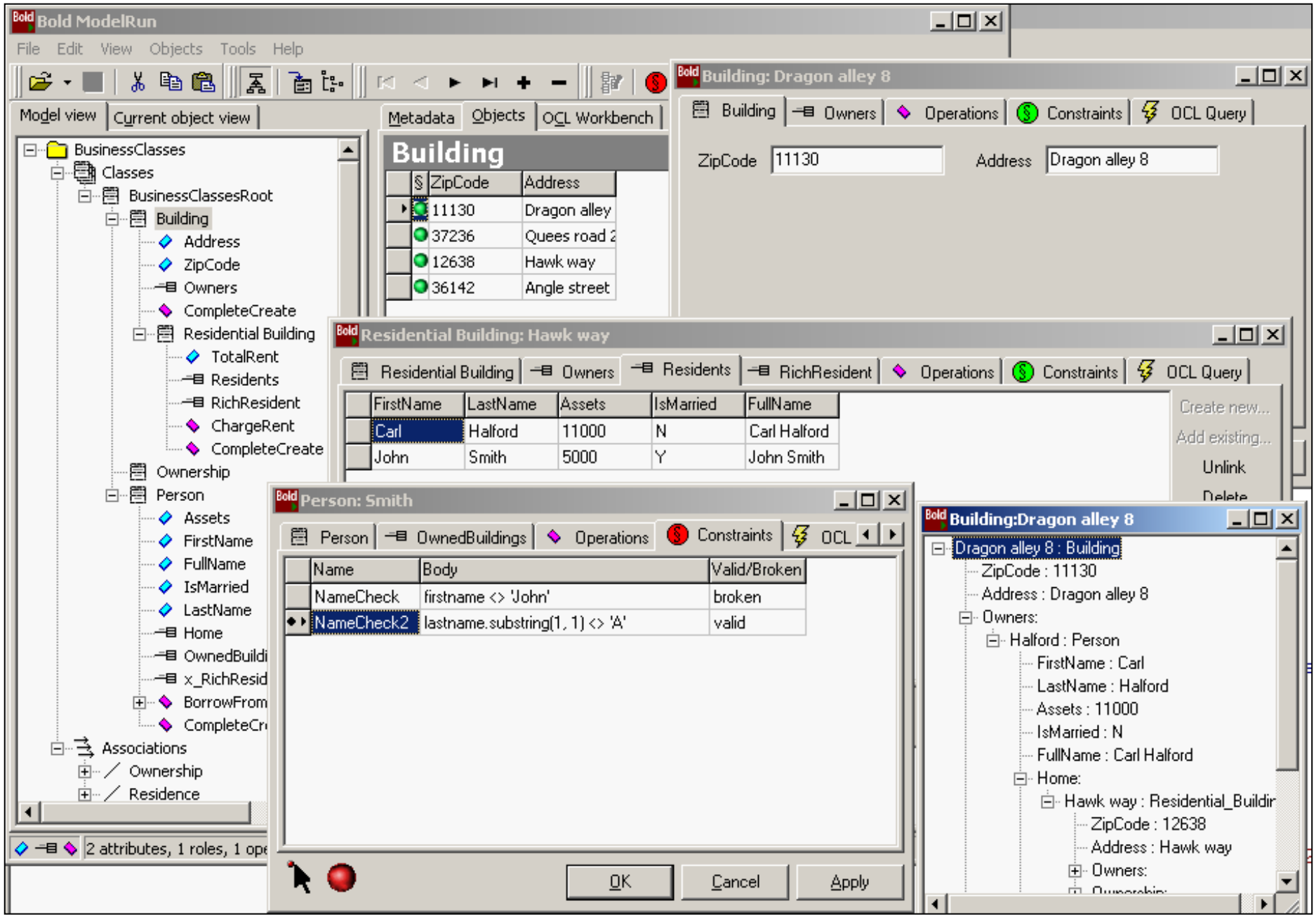
Technologies agiles

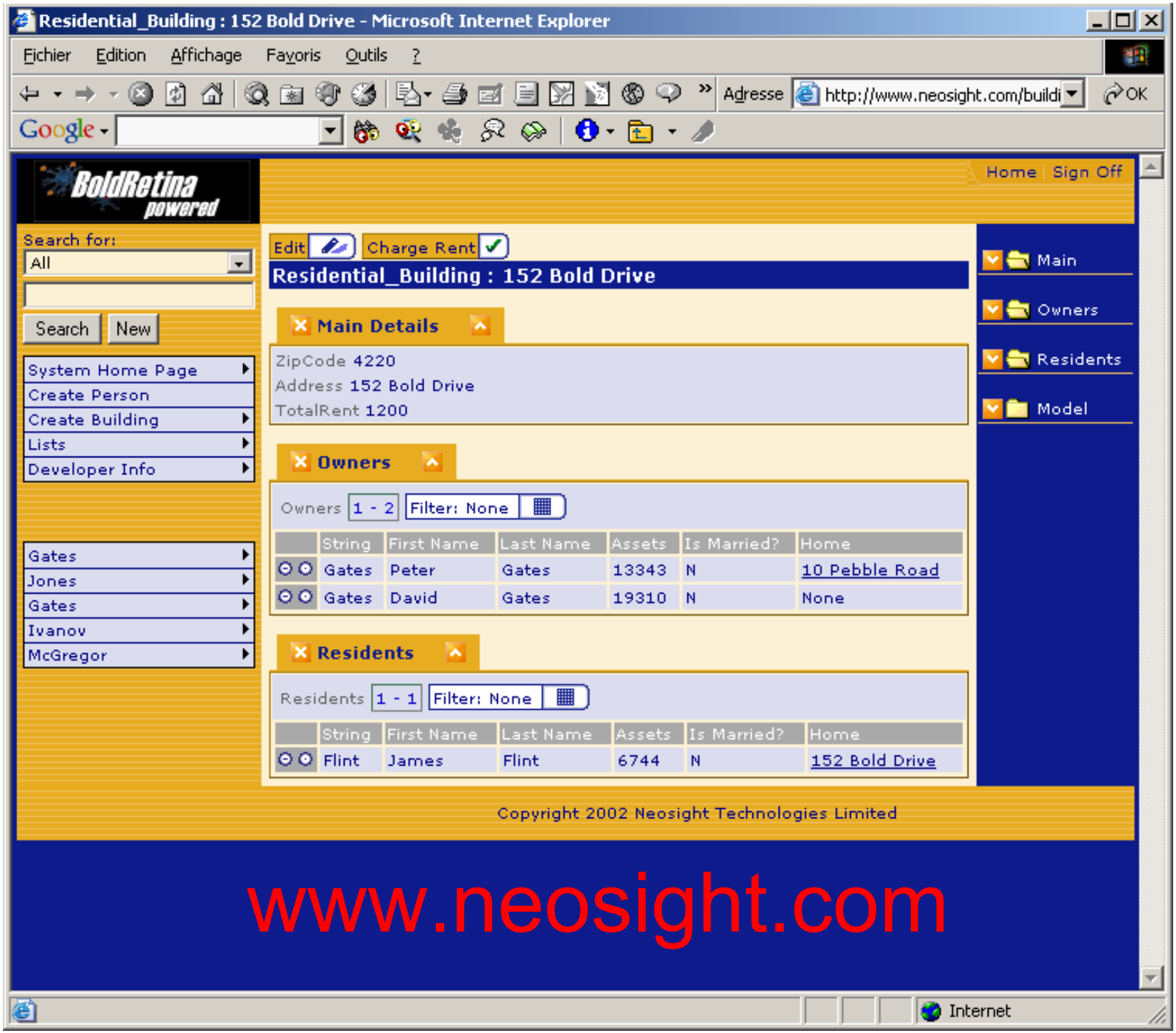
- Contrainte
  - Résultats optimisés à long, moyen ET court terme
- Exemples
  - XUML – exécution du modèle UML (simulateur)
  - AD – assemblage dynamique de composants (DCC)
  - MDA – générer le code
  - AOP – tisser les aspects indépendament

XUML

- ModelRun/BoldSoft
  - Borland Studio / Delphi / Architect




Bold ModelRun
File Edit View Objects Tools Help
Model view
Current object view
BusinessClasses
Classes
BusinessClassesRoot
Building
Address
ZipCode
Owners
CompleteCreate
Residential Building
TotalRent
Residents
RichResident
ChargeRent
CompleteCreate
Ownership
Person
Assets
FirstName
FullName
IsMarried
LastName
Home
OwnedBuildi
x_RichResid
BorrowFrom
CompleteCr
Associations
Ownership
Residence
2 attributes, 1 roles, 1 ope
Metadata
Objects
OCL Workbench
Building
§ ZipCode Address
11130 Dragon alley
37236 Quees road 2
12638 Hawk way
36142 Angle street
Bold Building: Dragon alley 8
Building
Owners
Operations
Constraints
OCL Query
ZipCode 11130
Address Dragon alley 8
Bold Residential Building: Hawk way
Residential Building
Owners
Residents
RichResident
Operations
Constraints
OCL Query
FirstName LastName Assets IsMarried FullName
Carl Halford 11000 N Carl Halford
John Smith 5000 Y John Smith
Create new...
Add existing...
Unlink
Delete
Bold Person: Smith
Person
OwnedBuildings
Operations
Constraints
OCL
Name Body Valid/Broken
NameCheck firstname <> 'John' broken
NameCheck2 lastname.substring(1, 1) <> 'A' valid
OK
Cancel
Apply
Bold Building:Dragon alley 8
Dragon alley 8 : Building
ZipCode : 11130
Address : Dragon alley 8
Owners:
Halford : Person
FirstName : Carl
LastName : Halford
Assets : 11000
IsMarried : N
FullName : Carl Halford
Home:
Hawk way : Residential_Buildir
ZipCode : 12638
Address : Hawk way
Owners:


Residential_Building : 152 Bold Drive - Microsoft Internet Explorer
Fichier Edition Affichage Favoris Outils ?
Adresse http://www.neosight.com/buildi
OK
Google
BoldRetina powered
Home | Sign Off
Search for:
All
Search
New
System Home Page
Create Person
Create Building
Lists
Developer Info
Gates
Jones
Gates
Ivanov
McGregor
Edit
Charge Rent
Residential_Building : 152 Bold Drive
Main Details
ZipCode 4220
Address 152 Bold Drive
TotalRent 1200
Owners
Owners 1 - 2
Filter: None
String First Name Last Name Assets Is Married? Home
Gates Peter Gates 13343 N 10 Pebble Road
Gates David Gates 19310 N None
Residents
Residents 1 - 1
Filter: None
String First Name Last Name Assets Is Married? Home
Flint James Flint 6744 N 152 Bold Drive
Main
Owners
Residents
Model
Copyright 2002 Neosight Technologies Limited
www.neosight.com
Internet

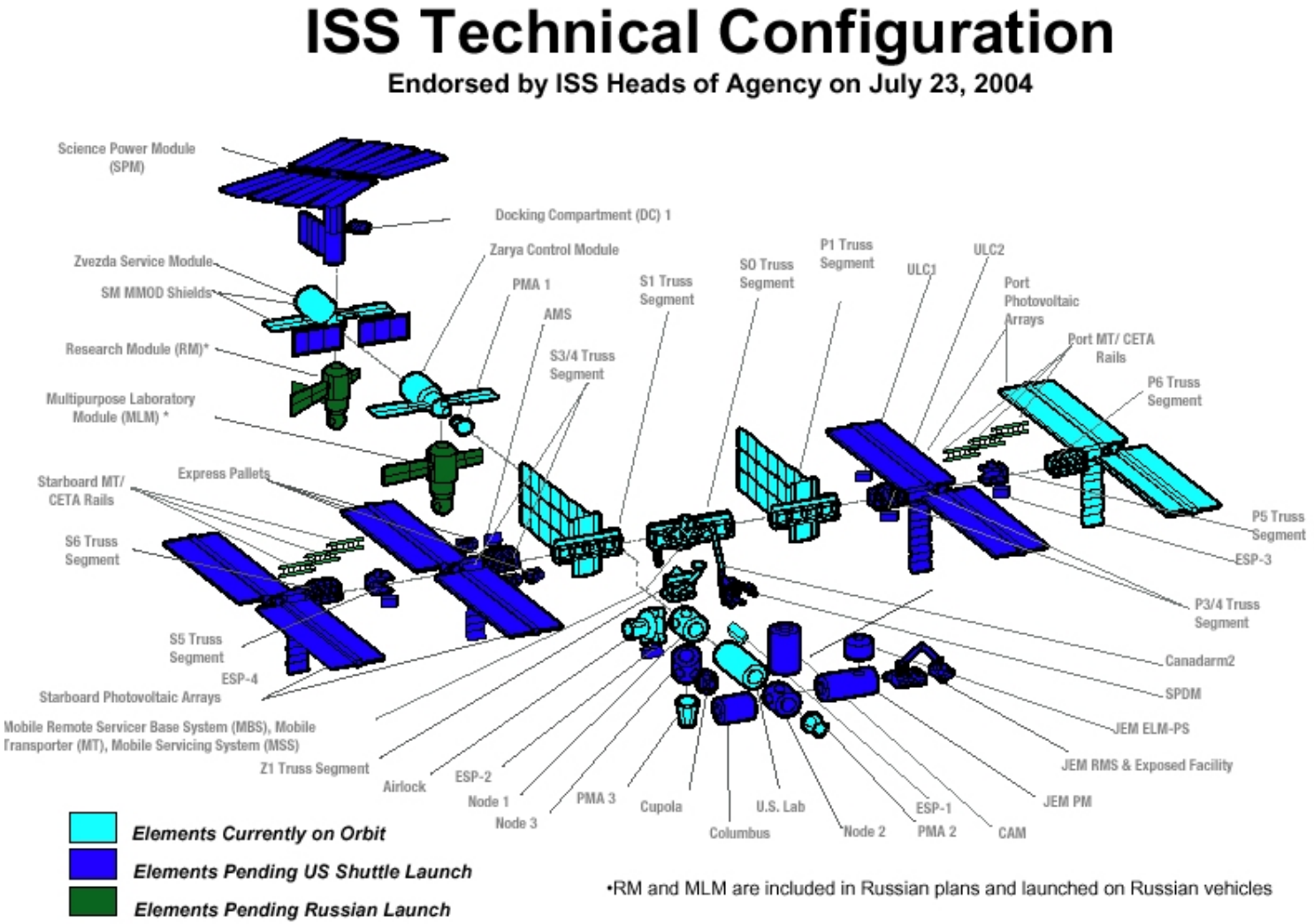
AD - Retour vers le futur : la nouvelle division sociale du travail

- STNG
  - Idéalisme de l'humanité ➔ propre amélioration
  - Épreuves et crises
    - Captain : « J'ai besoin de *distorsion 9* »
    - O'Brien : « Je dois reconfigurer le système, il me faut 4H »
    - Captain : « Tu as 18' »
  - Pensez-vous qu'il code ? Paramétrage ? Meta ?
- Nouvelle division sociale du travail de développement
  - Ancien développeur, reformaté en UML ➔ spécificateur
  - Développeur de « factories » : UML + DP + Refactoring + xUnit
  - Développeur de plugin : règles métier spécifiques : RAD

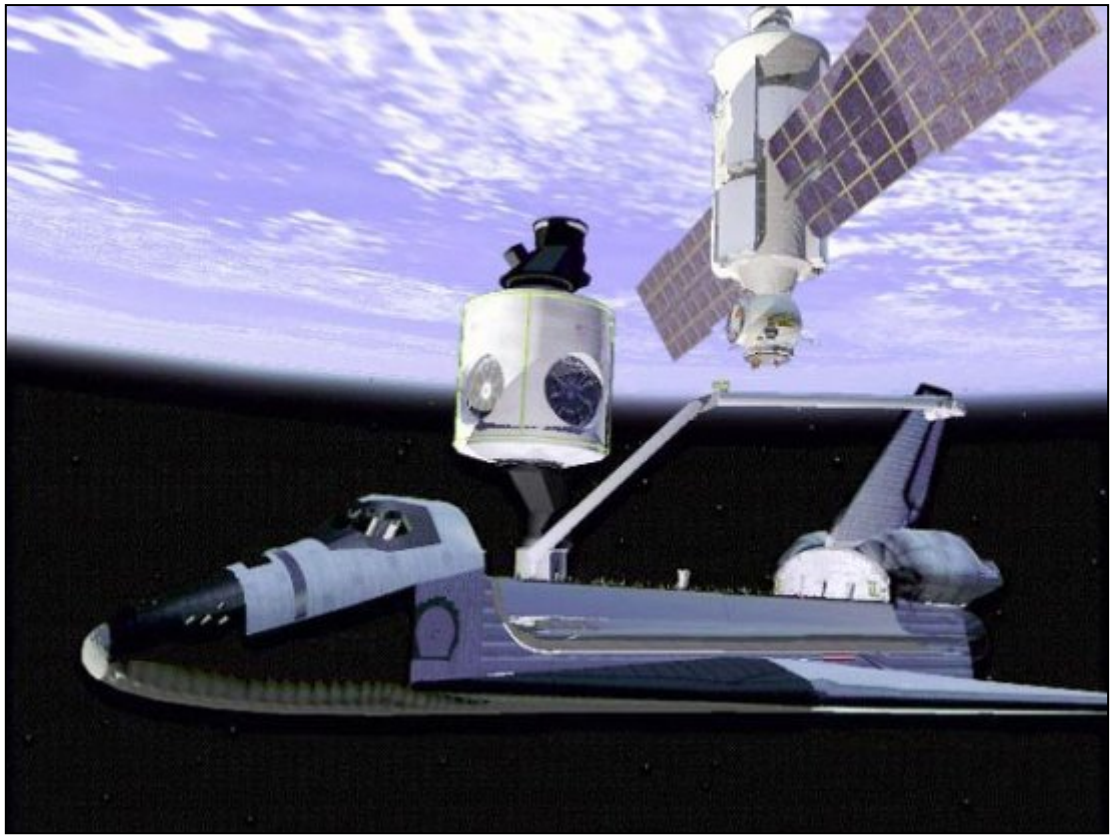
ISS d'assemblage dynamique

- L’aventure spatiale vs l’évolution du logiciel

ISS assemblée dans l'espace

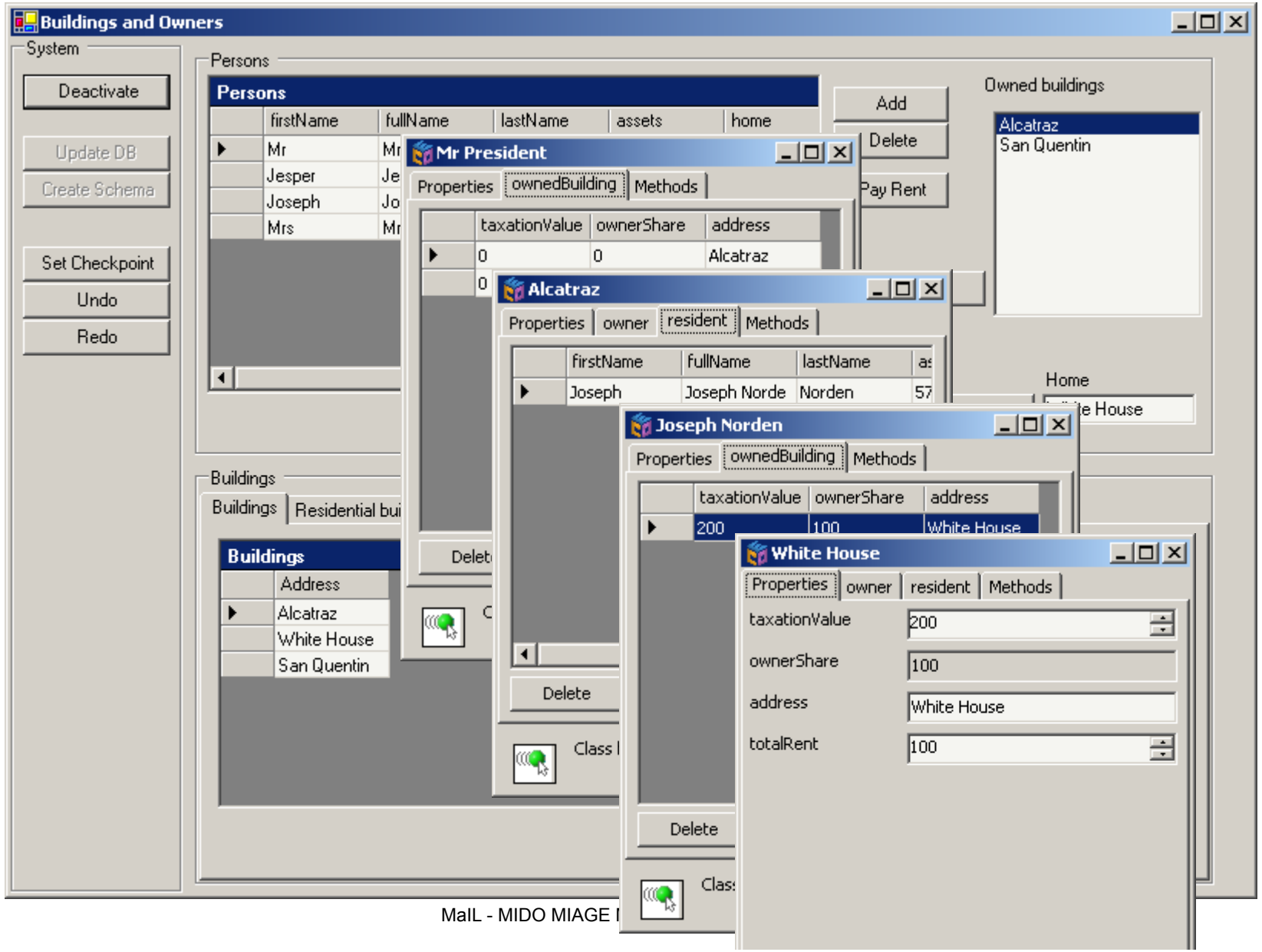


Séparation : Fabrication + Transport + Assemblage



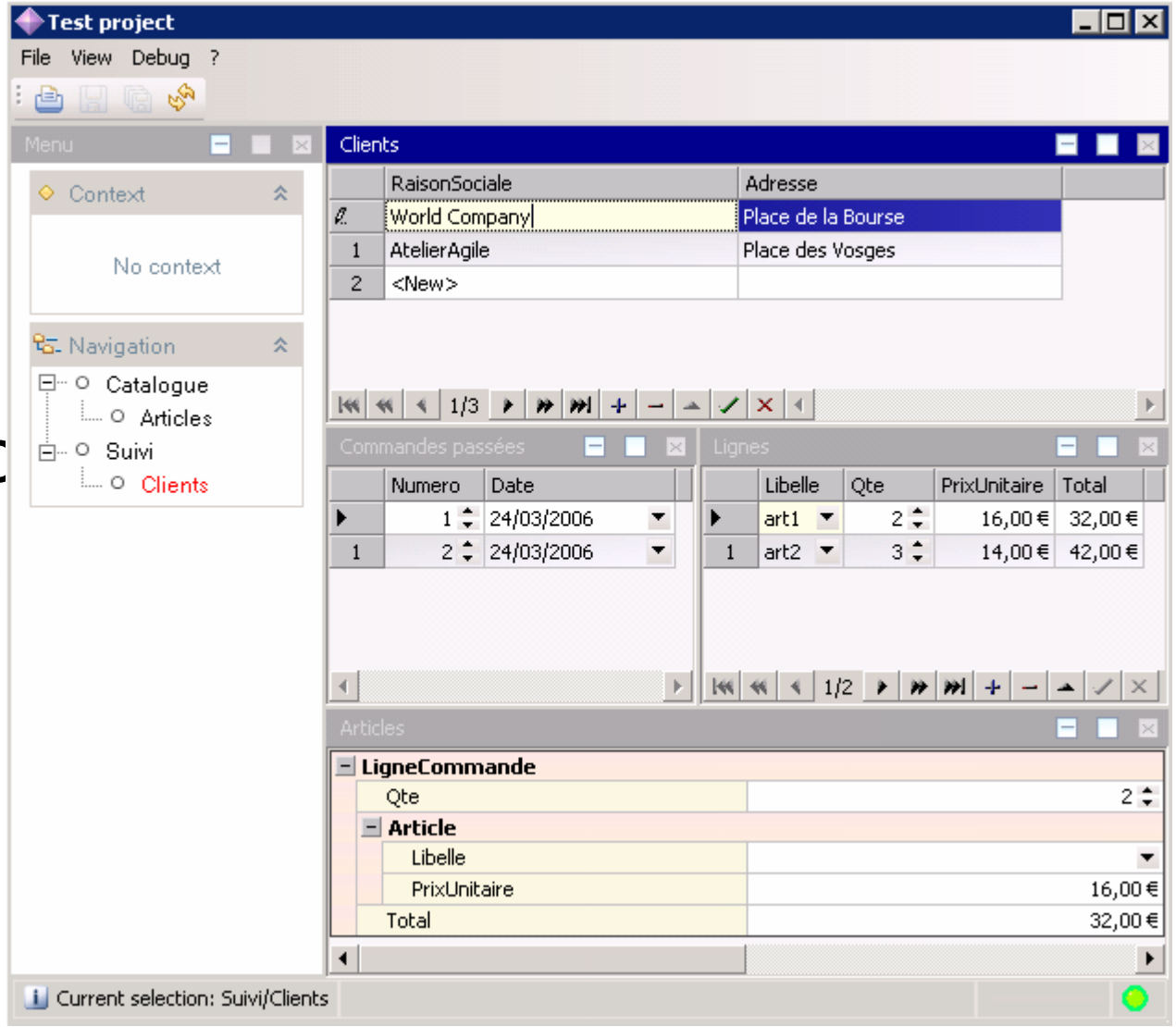
➔ Déjà vu ?! IKEA®

Exemple avec ECO III – IHM

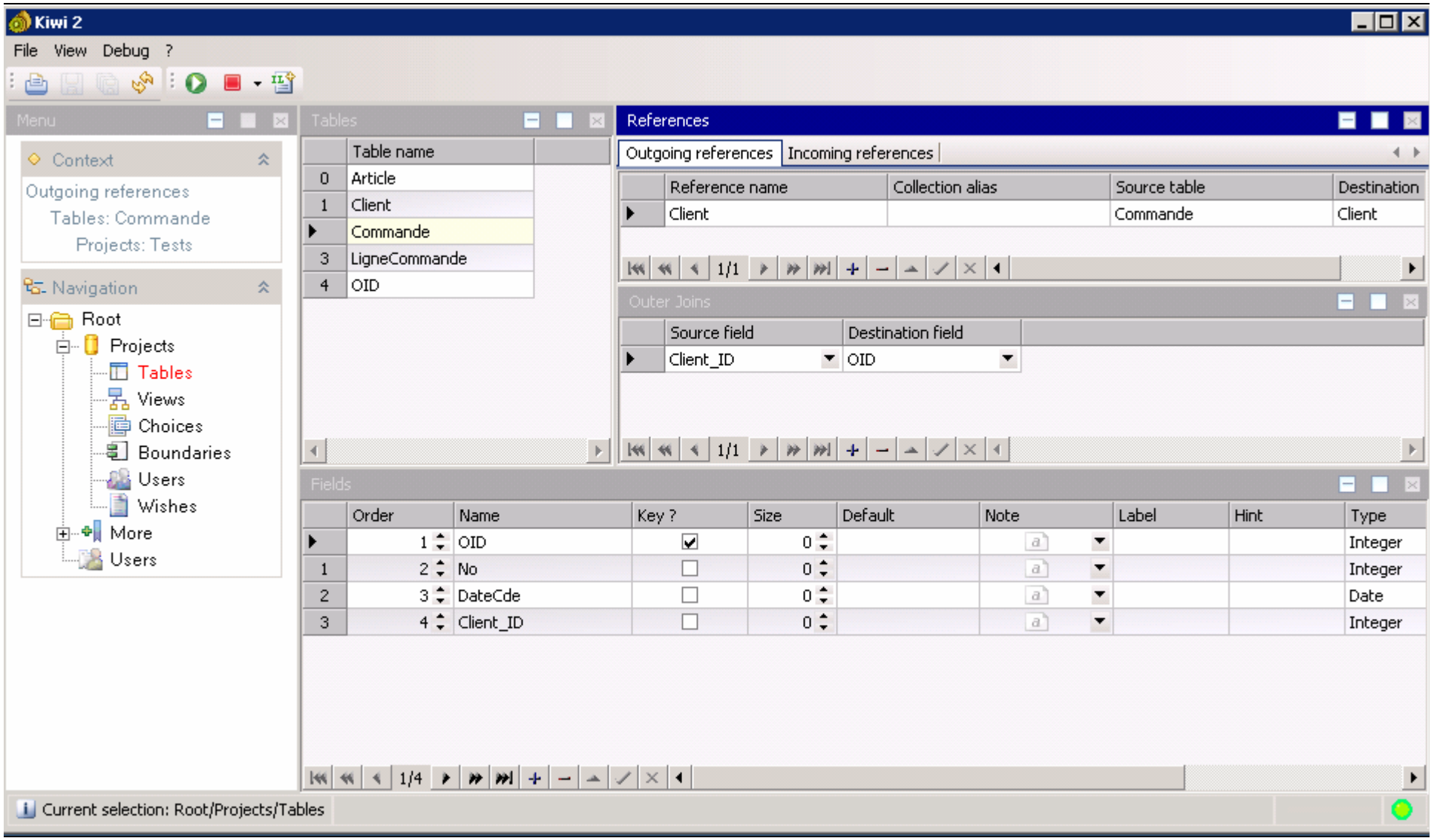


Prototype dynamique

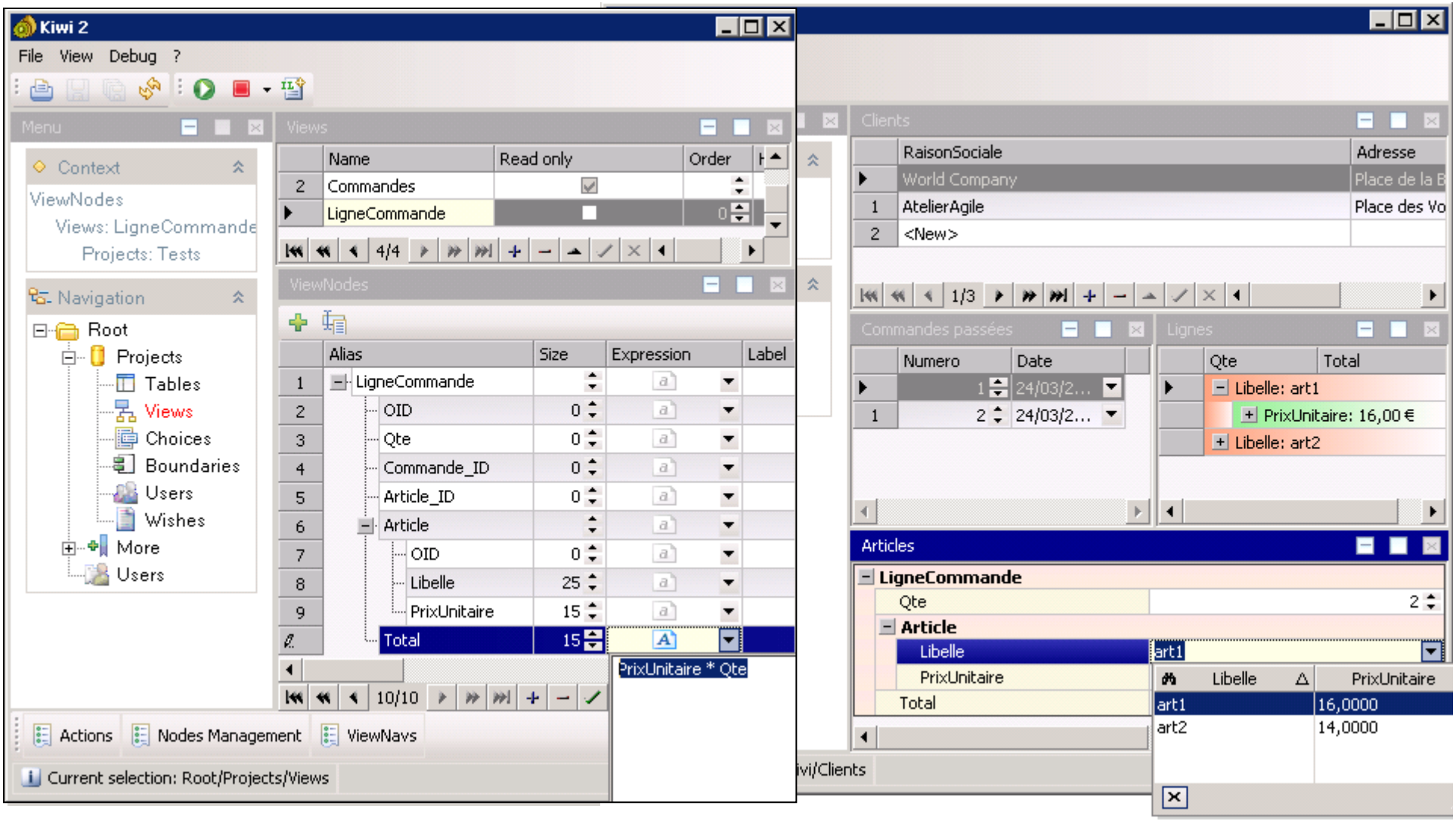
- Co
ça



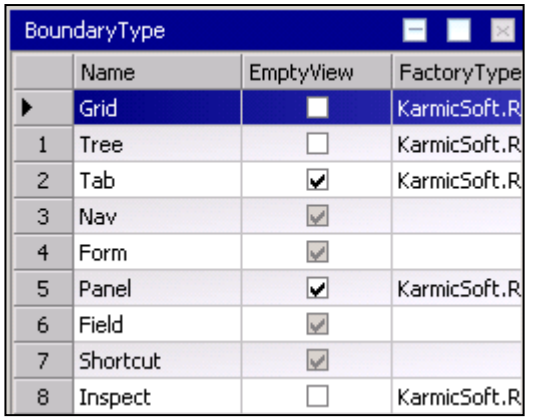
Référentiel métier



Spécification déclarative ➔ Assemblage dynamique

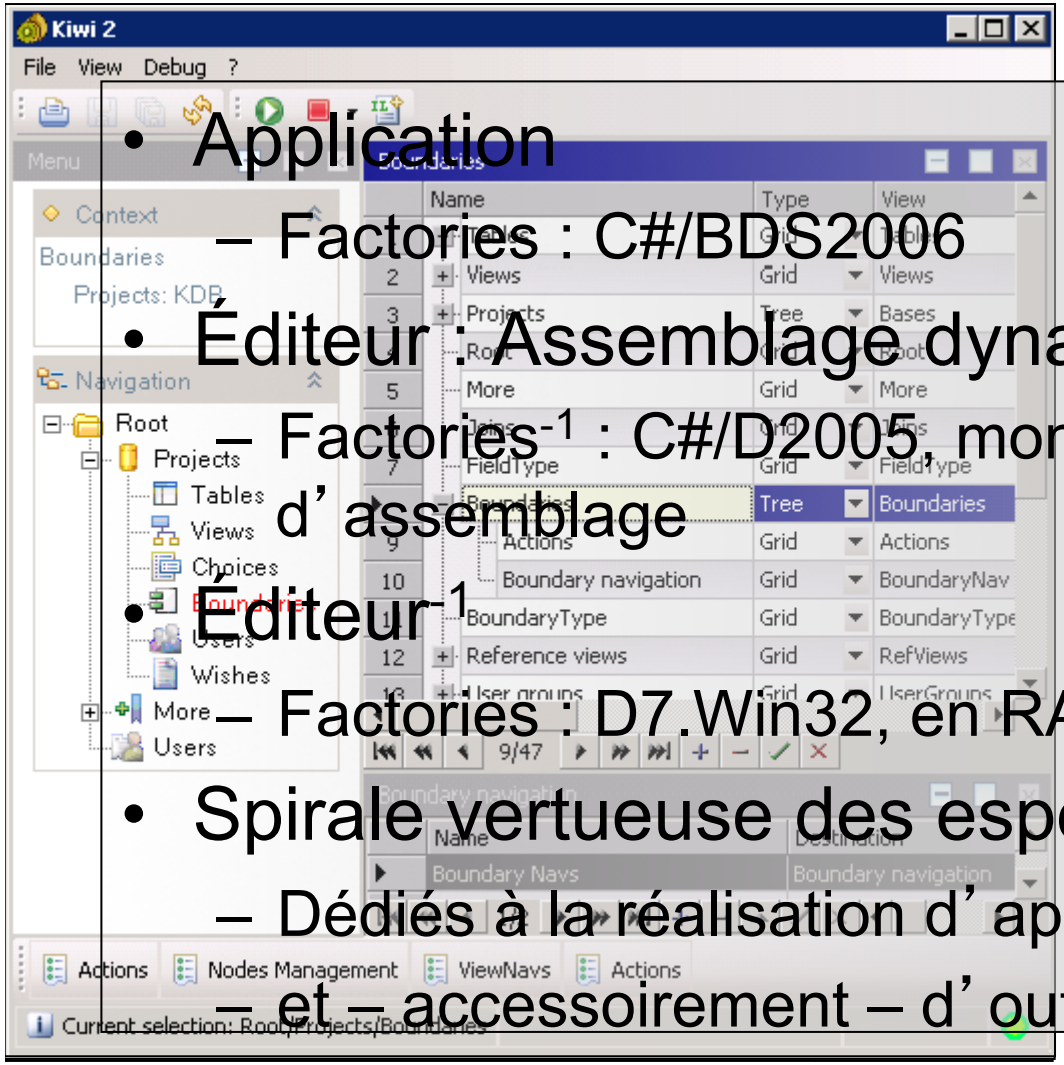


Stratégie de développement par A.D.



- Application
  - Objets génériques et faiblement typés :
    - Instanciés dynamiquement
    - À partir du Référentiel métier
      - contrôles + chemins + structures
    - Validation modèle métier (UML)
      - par maquettage ➔ exécution UC
  - Plugins spécifiques associés dynamiquement / proxies
    - Pour couvrir les besoins jusqu'à 100%
- Éditeur
  - Permet la spécification de l'application
    - En termes macroscopiques
  - Est une application à son tour …

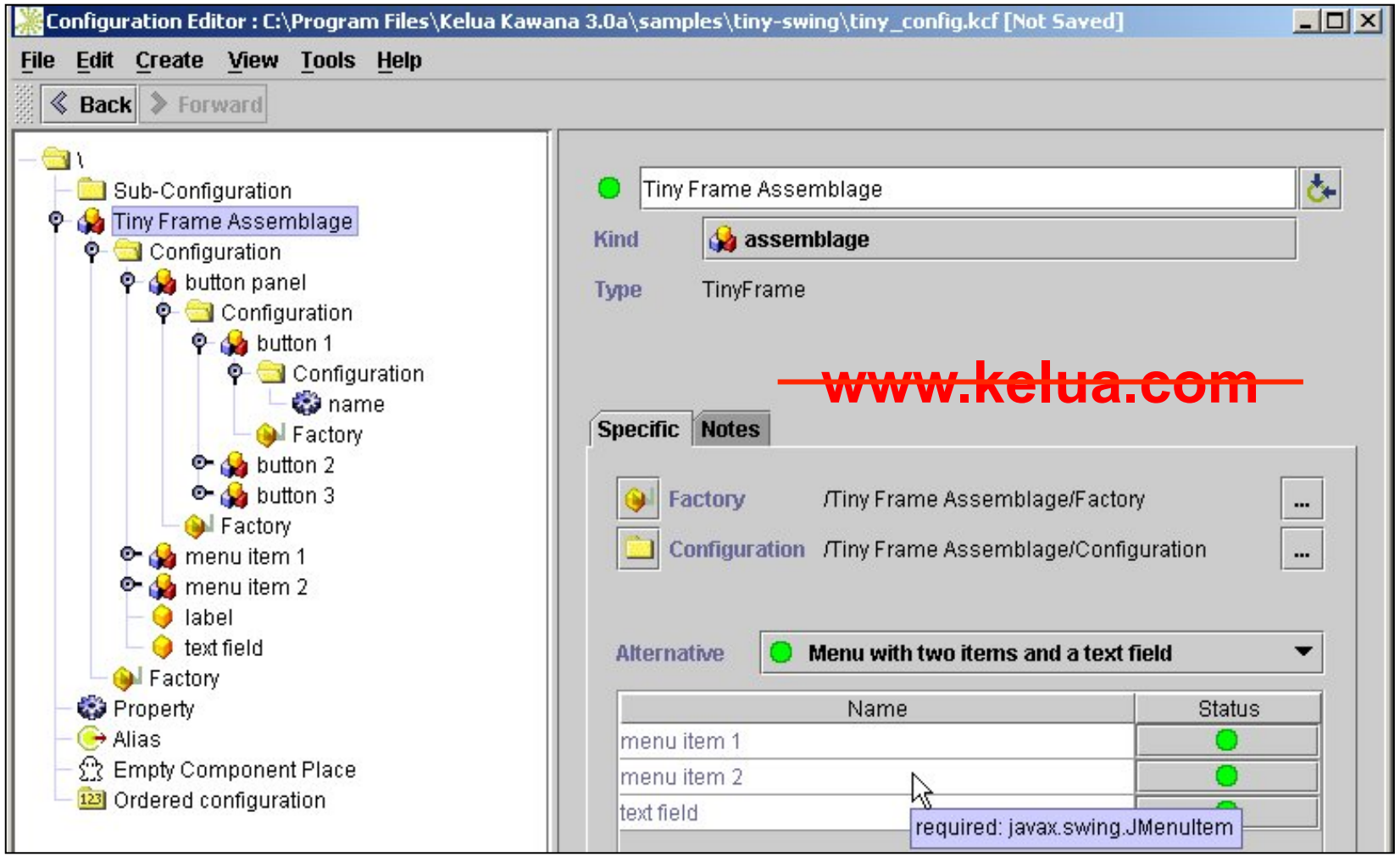
Meta-spécification



- Application
  - Factories : C#/BDS2006
- Éditeur : Assemblage dynamique (AD)
  - Factories^{-1} : C#/D2005, mono-stratégie d'assemblage
- Éditeur^{-1}
  - Factories : D7.Win32, en RAD ➔ RIP
- Spirale vertueuse des espèces d'outils
  - Dédiés à la réalisation d'applications évolutives
  - et – accessoirement – d'outils du même type ☺

Assemblage dynamique (DCC[1])

- Exemple Kilim/ObjectWeb + Kelua Kawana



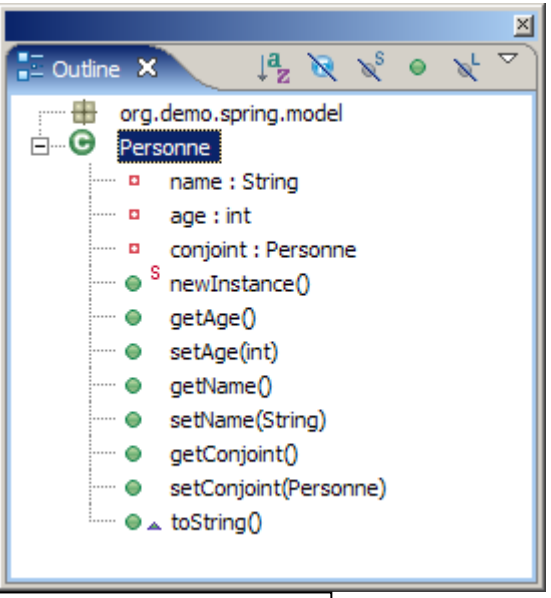
1: DCC : Dynamic Component Composition

Spring

Personne.java

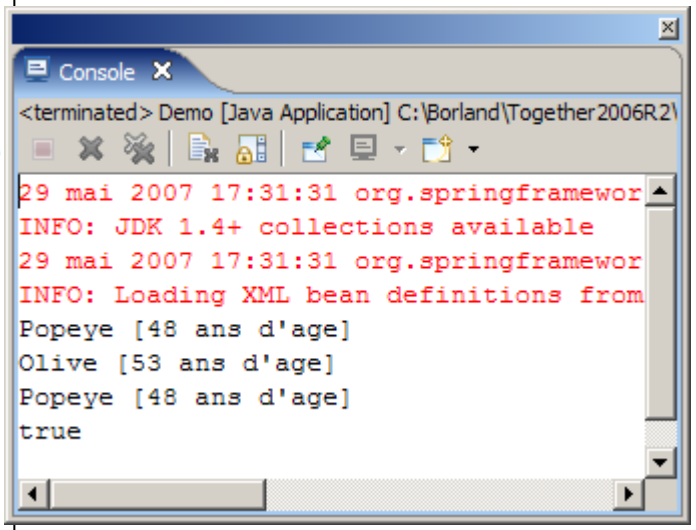
```
package org.demo.spring.model;

public class Personne
{
    private String name;

    private int age;

    private Personne conjoint;

    public static Personne newInstance()
    {
        return new Personne();
    }
```

Outline




```
package org.demo.spring.main;

import org.demo.spring.model.Personne;

public class Demo
{
    public static void main(String[] args)
    {
        Resource resource = new ClassPathResource("spring.xml");
        BeanFactory factory = new XmlBeanFactory(resource);

        Personne p = (Personne) factory.getBean("Popeye");

        System.out.println(p);
        System.out.println(p.getConjoint());
        System.out.println(p.getConjoint().getConjoint());
        Personne p2 = (Personne) factory.getBean("Popeye");

        System.out.println(p2 == p);

    }
}
```

Console



Le fichier « spring.xml »

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
    <bean id="Popeye"
          class="org.demo.spring.model.Personne">
        <property name="name" value="Popeye"/>
        <property name="age" value="48"/>
        <property name="conjoint" ref="Olive"/>
    </bean>
    <bean id="Olive"
          class="org.demo.spring.model.Personne"
          factory-method="newInstance">
        <property name="name" value="Olive"/>
        <property name="age" value="53"/>
        <property name="conjoint" ref="Popeye"/>
    </bean>
    <bean id="Brutus"
          class="Personne">
        <property name="name" value="Brutus"/>
        <property name="age" value="48"/>
    </bean>
</beans>
```

MDA : Architecture pilotée par le modèle

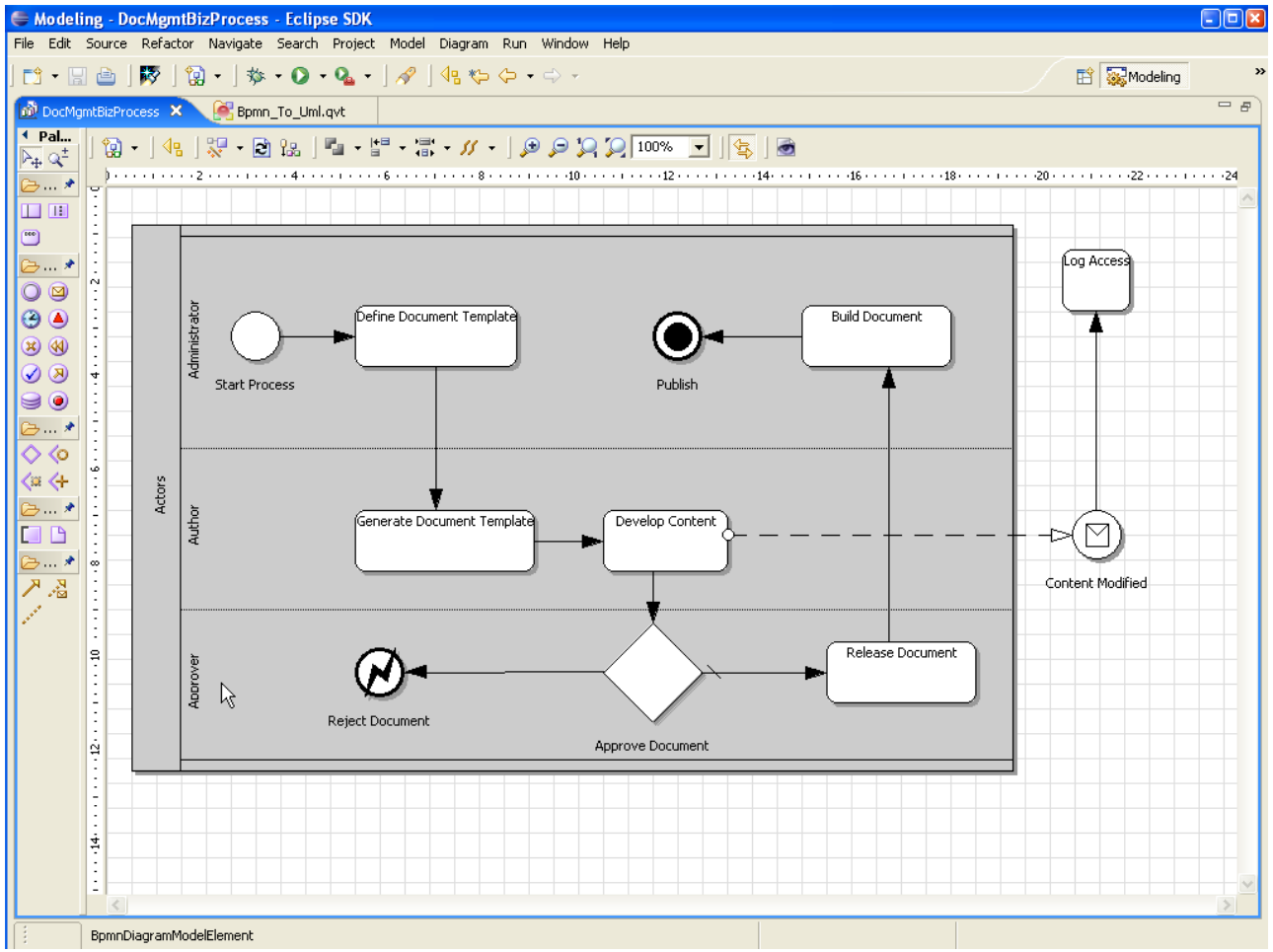
- Le MDA est une nouvelle **façon** d'écrire des spécifications pour développer des applications,
- basée sur un **modèle indépendant de plateforme** (PIM) et
- accompagné d'un ou plusieurs **modèles spécifiques** à des plateformes (PSM) et d'ensembles de définition d'interfaces, chacune décrivant comment le modèle de base est implémenté sur chaque plateforme différente.
- Cibles : CORBA, EJB, MTS, etc. …
- Exemples :
  - IO, Kennedy Carter, Kabira, Secant, Metys
  - Borland Together Architect 2006
    - Support MDA : UML 2.0, OCL 2.0, QVT, XMI 2.0

Transformer un modèle Business Process en un modèle UML

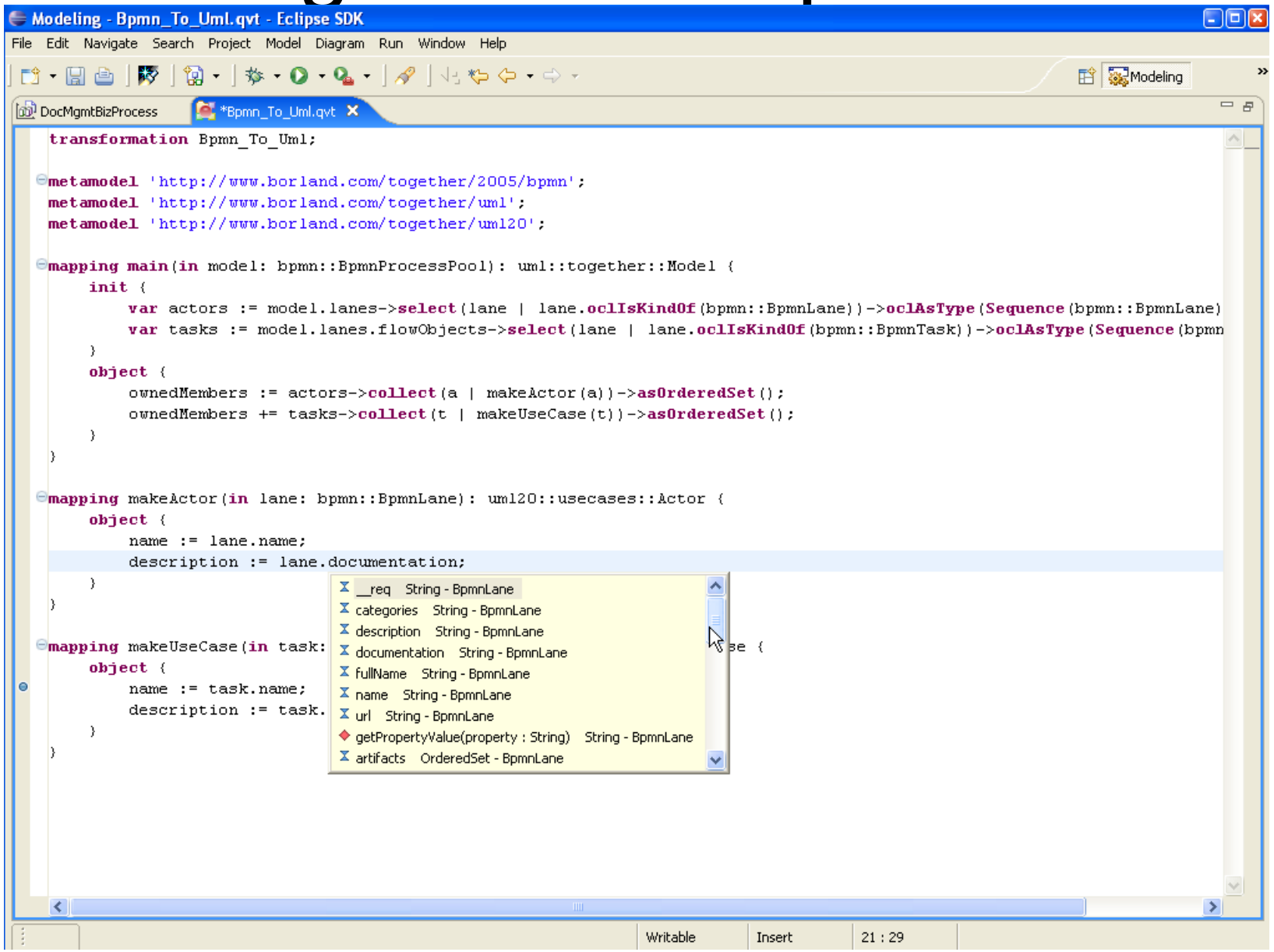
- Exemple simple

- Couloirs (BPD) => Acteurs UML

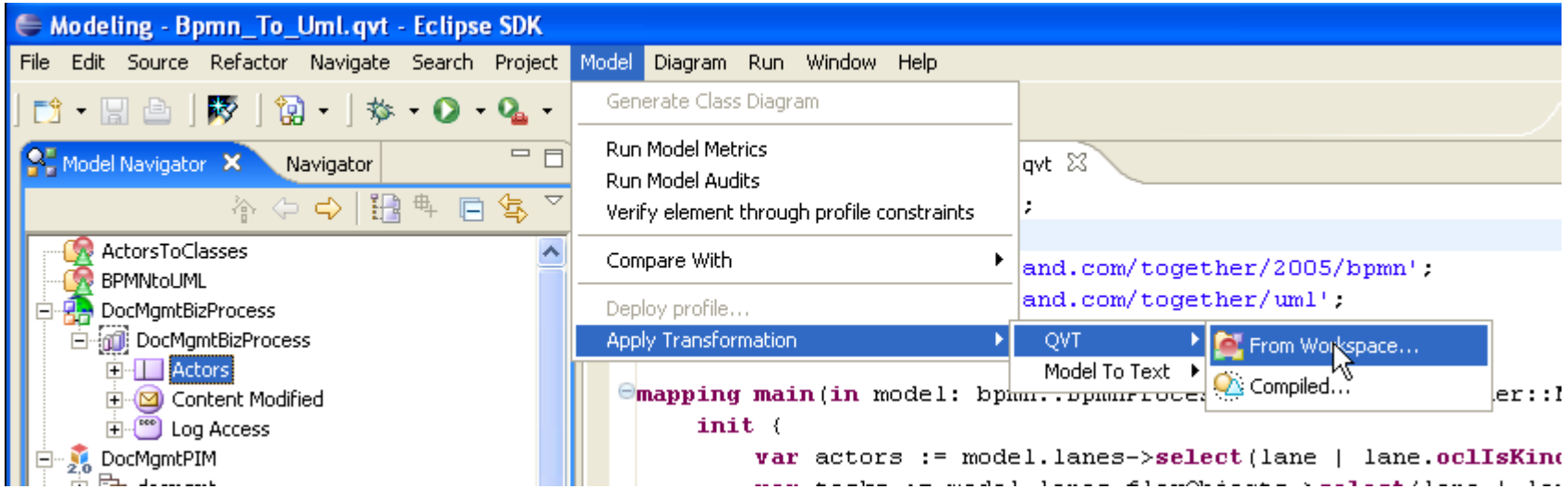
- Tâches => Use Cases



Together - Script QVT

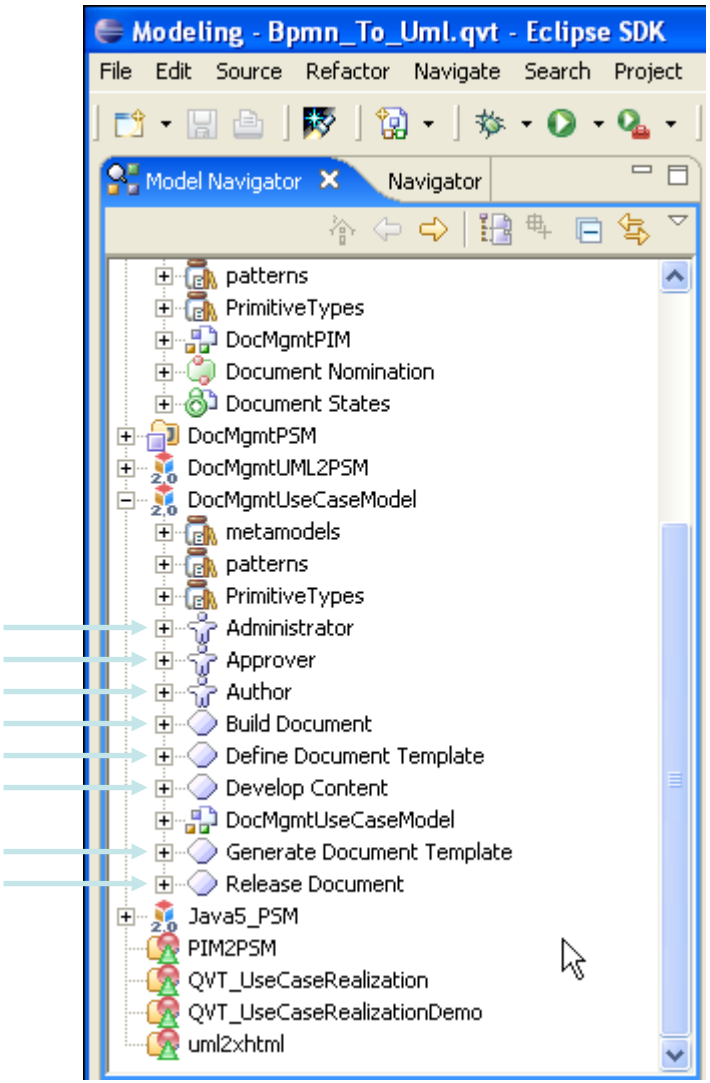


Exécuter une transformation Model-to-Model



Transformation effectuée

- Acteurs & Use Cases
- Dans le modèle cible



Quel est l'intérêt de QVT?

- QVT
  - QVT fournit le socle des transformation model-to-model pour MDA
  - Un langage standardisé pour exprimer les transformations intégrables dans les outils et manipulables par les [sur]humains
- Together
  - La toute première implémentation QVT commerciale disponible sur le marché et bientôt au CRIO UNIX
  - Fournit un environnement de développement robuste pour les architectes
    - Editeur QVT syntaxe en couleurs, code completion, vérification d'erreurs
    - Debuggage interactif
    - Usage facile de QVT dans le workspace
    - Déploiement simplifié
    - Génération du fichier de trace automatisé
      - Navigation dans le modèle depuis le viwer de trace
- Facilement partageable avec le rôle Designer

AOP : Aspect Oriented Programming

Avant

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;
    Display.update(this);
  }
  void setP2(Point p2) {
    this.p2 = p2;
    Display.update(this);
  }
}

class Point {
  private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;
    Display.update(this);
  }
  void setY(int y) {
    this.y = y;
    Display.update(this);
  }
}
```

```
aspect DisplayUpdating {

  pointcut move(FigureElement figElt):
    target(figElt) && (call(void
   FigureElement.moveBy(int, int) ||
     call(void Line.setP1(Point)) ||
     call(void Line.setP2(Point)) ||
     call(void Point.setX(int))   ||
     call(void Point.setY(int)));

  after(FigureElement fe) returning: move(fe) {
    Display.update(fe);
  }
}
```

- Réduction des interférences
- Augmentation significative de la lisibilité

Après

```
class Line {
  private Point p1, p2;

  Point getP1() { return p1; }
  Point getP2() { return p2; }

  void setP1(Point p1) {
    this.p1 = p1;
  }
  void setP2(Point p2) {
    this.p2 = p2;
  }
}

class Point {
  private int x = 0, y = 0;

  int getX() { return x; }
  int getY() { return y; }

  void setX(int x) {
    this.x = x;
  }
  void setY(int y) {
    this.y = y;
  }
}
```

6.3 Conclusion

Bilan de l’agilité. Perspectives.

Devoirs. Bonus

Bibliographie (Kit de survie)

L’agilité

- Définitions. Dictionnaire de L'Académie française, Première édition, 1694
  - Agile (AGIR) Agile. adj. de tout genre, Leger & dispos, qui a une grande facilité à se mouvoir à agir, il ne se dit guère que du corps. *Merveilleusement agile. le tigre, le singe sont des animaux fort agiles.*
  - Agilité (AGIR). subst. fem. Légèreté, disposition du corps à se mouvoir avec facilité.
    Il saute avec une grande agilité.
  - Rigidité (RIGUEUR)
    Rigidité. substant. fem. Grande séverité, inflexibilité. *Les Magistrats font observer la loy contre les duels avec une extrême rigidité. la rigidité de ses moeurs.*
- Repères
  - Stimuli ou besoin
  - Vitesse de réaction
  - Facilité => coût minimal / économique.

« Besoin-vitesse-coût »

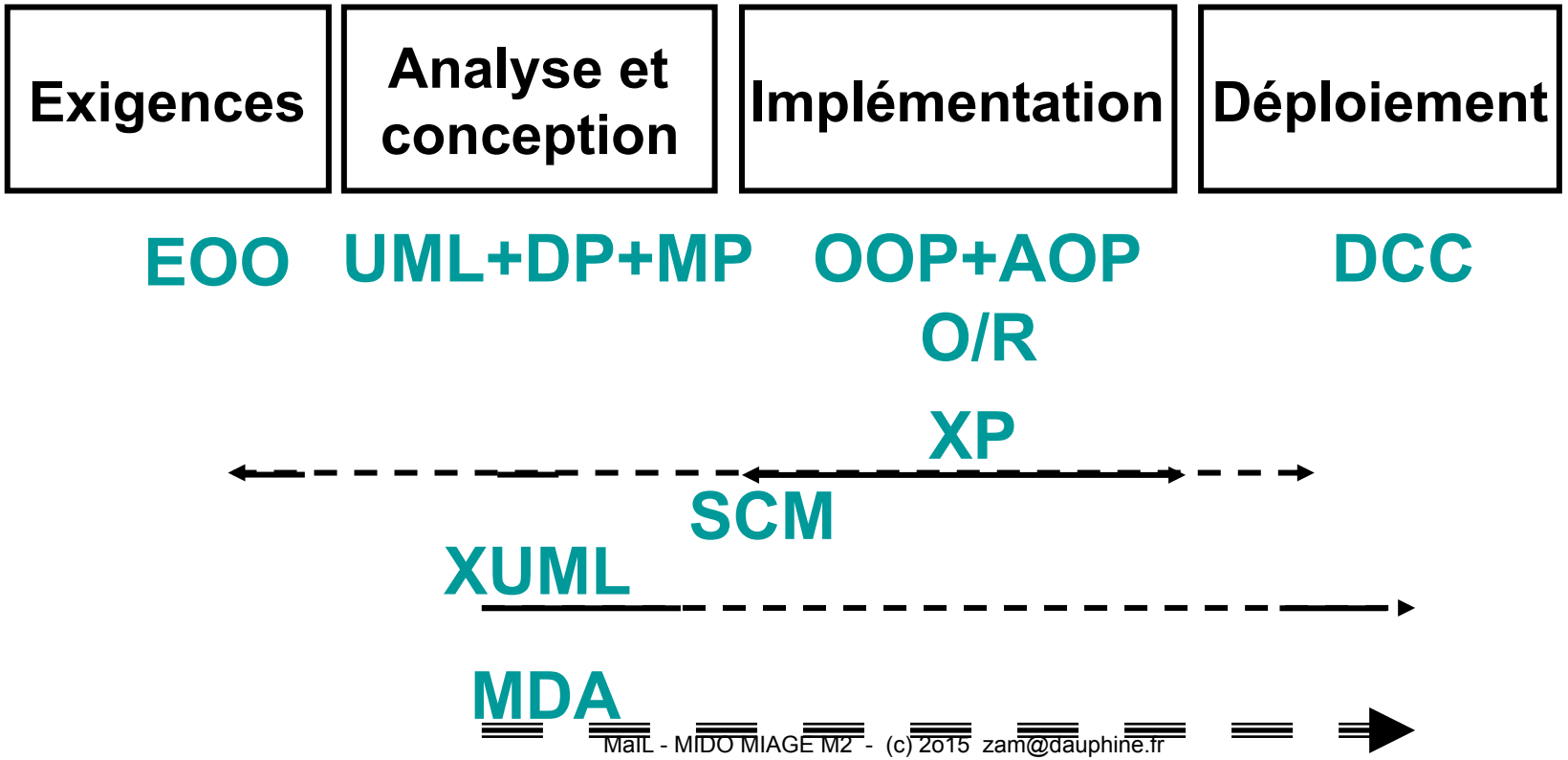
- Application de la grille dans les SI
  - Performances. « 1'' du clic au résultat »
    - Performances individuelles des applications
      - » Mesurables avec des outils de profiling, montée en charge
    - Échanges efficaces entre applications (de générations différentes).
      - » Problématique : partage, répartition, notification
    - Concerne : applis, réseau, matériel, OS, EAI, BD
  - Couverture des besoins
    - Exprimé : analysé, conçu, implémenté, déployé, documenté
    - Non exprimé, mais proche : couvert si flexibilité, généricité, voire générosité de la part d'informaticiens visionnaires
      - » Disponibilité directe (documenté ou pas), passage FF->€
      - » Réclame une configuration accessible / run-time
      - » Configuration prévue à la conception
    - Non prévu : capacité à l'intégrer à faible coût
      - » Maîtrise de l'impact, modularisation, productivité
      - » Vision claire, actions planifiées, non régression

Solutions flexibles

- Tableur ?
  - Typage absent, sans contraintes, ouvert, parfois utile
- Allier rigueur et souplesse
  - Table de référence, fichier/table de configuration
  - Orientation objet, modélisation visuelle, UML
  - Patterns (composite)
  - Stratégies évolutives
    - Refactoring, metapatterns & xUnit
    - Méthodologie souple
- Vigilance permanente / performances & robustesse
  - Optimize-it, jProbe, gpProfiler, xUnit
  - Tests quotidiens et automatiques

Bilan

- Problématique de l'agilité : changement fréquents
  - Grille besoin-vitesse-coûts
  - Solutions techniques et méthodologiques
- L'agilité au long du cycle de vie



Perspectives

- Spéculation « crédible » (à quel terme ?, crise => techno++)
  - Fonctionnel orienté composants
    - IDE / RAD : Drag&Drop de classes métier
  - Pilotage par le modèle : MDA
    - Cibles diverses, coexistantes : Windows, Linux, J2EE, DOTNET
  - Technologie de programmation : souterraine
    - SQL, XML, EJBQL, JDOQL, OCL, UML, L3G
  - Ponts
    - Génération de code, tissage ou assemblage dynamique
- Nouvelles problématiques
  - Traçabilité au long du cycle de vie : E-A&C-I-D, mais aussi entre itérations : versions et configurations cohérentes
  - Cohabitation entre les applications du SI : niveau d'agilité et d'évolution non uniforme : versions de schéma et données
  - Historisation, variantes de Base de Données …
  - … mais ça, c'est une autre « histoire » ☺

Bibliographie

- **Fowler & Scott** : UML Distilled, Second ed., Addison Wesley (AW), 2000
- **Rosenberg & Scott** : Use Case Driven Object Modeling using UML. AW/99
- **Gamma & al**. : Design Patterns / CD. 95
- **Blaha & Rumbaugh** : Object-Oriented Modeling and Design with UML, P/PH, 2005
- Coad, Lefebre & DeLuca : Java modeling in Colors, PH99, www.coad.com/peter
- Pree : Design patterns for Object-Oriented Software Development, AW, 1995
- Pree W., *Hot-Spot-Driven Development,* pp.379-394, in *Building Application Frameworks*, Ed Fayad, M, Wiley, 1998
- Meszaros & Doble, *A Patterns Language for Pattern Writting*, pp.530-574, in *Patterns Languages of Program Design* Vol 3, Ed. Martin R., Riehle D., Buschmann F., Addison-Wesley Longman, 1998
- Tokuda, PhD disertation http://citeseer.nj.nec.com/tokuda99evolving.html
- www.extremeprogramming.org
- http://c2.com/cgi/wiki?ExtremeProgrammingSummary
- www.xprogramming.com
- www.xp123.com
- Google : AOP, AspectJ, MDA/OMG, Ambler, Keller, JDO

Le mot de la fin

Give a man a fish; you have fed him for today.

Teach a man to fish; and you have fed him for a lifetime.

Give him a religion, and he'll starve to death while praying for a fish.

Merci