

# Développement Orienté Objet

## Ingénierie des modèles

Christine Solnon

INSA de Lyon - 4IF

2011

# Plan du cours

## Introduction

### Model-Driven Architecture (MDA)

Présentation générale de MDA

Méta-modèles et Meta Object Facility (MOF)

Object Constraint Language (OCL)

Profils UML

XMI/JMI et la sérialisation/manipulation de modèles

Synthèse

### Modèles pour l'optimisation et la satisfaction de contraintes

Problèmes d'optimisation sous contraintes

Résolution de problèmes d'optimisation sous contraintes

Programmation par contraintes

# Automatiser la production de logiciels ?

## Saint Graal du génie logiciel :

↪ Générer automatiquement le code à partir d'une spécification

## Point clé : Comment spécifier un système ?

- ▶ Description informelle en langue naturelle ?
  - ▶ Facile à comprendre par l'utilisateur
  - ▶ Mais ambiguë et impossible à exploiter automatiquement
- ▶ Description à l'aide d'un langage formel ?
  - ▶ Difficile à comprendre par un utilisateur non initié
  - ▶ Mais non ambiguë et plus facile à exploiter automatiquement

En pratique : Mélange d'informel et de formel

# Petite parenthèse sur les langages formels

- ▶ Langage = ensemble (potentiellement infini) de “mots”  
↪ Langage Java = ensemble des programmes Java
- ▶ Langage formel = langage décrit par une grammaire formelle  
↪ Ens. de règles définissant les mots syntaxiquement corrects
- ▶ Automate = programme qui décide si un mot appartient au langage décrit par une grammaire  
↪ Construction d'un arbre de syntaxe abstraite

## Exemple de grammaire : langage des expressions arithmétiques

$$\begin{aligned}\langle \text{Expr} \rangle &::= \langle \text{Expr} \rangle ' + ' \langle \text{Terme} \rangle \mid \langle \text{Expr} \rangle ' - ' \langle \text{Terme} \rangle \mid \langle \text{Terme} \rangle \\ \langle \text{Terme} \rangle &::= \langle \text{Terme} \rangle ' * ' \langle \text{Fact} \rangle \mid \langle \text{Terme} \rangle ' / ' \langle \text{Fact} \rangle \mid \langle \text{Fact} \rangle \\ \langle \text{Fact} \rangle &::= \langle \text{Const} \rangle \mid \langle \text{Var} \rangle \mid '(' \langle \text{Expr} \rangle ') \\ \langle \text{Const} \rangle &::= \dots \\ \langle \text{Var} \rangle &::= \dots\end{aligned}$$

**Question : peut-on décrire tous les langages comme ça ?**

↪ **Plus de détails dans le cours “Grammaires et langages”**

# Qu'est-ce que la transformation de programmes ?

## Procédure qui transforme un mot d'un langage vers un autre

- ▶ Mot = élément d'un langage décrit par une grammaire formelle  
    ↪ Ex. : Prog. Java, Diagramme de classe, Spécification algébrique, ...
- ▶ Langage source = langage du mot en entrée
- ▶ Langage cible = langage du mot en sortie

## Reformulation : Langage cible = Langage source

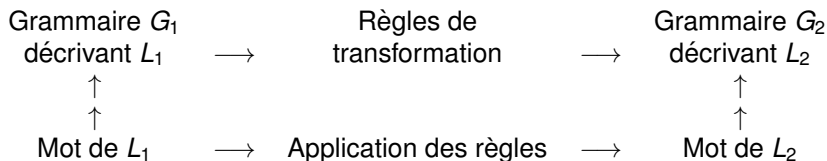
- ▶ Normalisation ↪ Réduire la complexité syntaxique
- ▶ Optimisation ↪ Améliorer les performances en temps ou mémoire
- ▶ Refactoring ↪ Améliorer le code sans changer le comportement
- ▶ Re-engineering ↪ Restructuration forte

## Traduction : Langage cible $\neq$ Langage source

- ▶ Migration ↪ cible et source sont des versions  $\neq$  d'un même langage
- ▶ Synthèse de prog. ↪ source = spécification / cible = lang. exécutable
- ▶ Compilation ↪ source haut niveau / cible de plus bas niveau
- ▶ Reverse engineering ↪ source bas niveau / cible de plus haut niveau

# Comment transformer automatiquement des programmes ?

- ▶ Une grammaire décrit la syntaxe des mots “bien formés” :  
↪ Décomposition d'un mot en catégories syntaxiques
- ▶ Une grammaire ne permet pas de “comprendre” un mot :  
↪ Associer une sémantique aux catégories syntaxiques  
↪ Interprétation dans le monde “réel”
- ▶ Pour pouvoir transformer automatiquement un mot, il faut qu'il existe une correspondance sémantique des catégories syntaxiques du langage source vers le langage cible  
↪ Règles de transformation



# Ingénierie des modèles ?

- ▶ Utilisation “classique” des modèles :
  - ▶ Elaboration d'un modèle (d'une suite de modèles)
  - ▶ Génération d'un squelette de code
  - ▶ Ecriture et maintenance du code
    - ↪ Les modèles deviennent obsolètes
- ▶ L'ingénierie des modèles centre le processus sur les modèles
  - ▶ Ecriture et maintenance de modèles
  - ▶ Génération du code à partir des modèles
  - ↪ Objectif ambitieux... utopiste ou réaliste ?
- ▶ Standard de l'OMG en 2001: Model Driven Architecture (MDA)
  - ▶ Modèles à différents niveaux d'abstraction :
    - ↪ PIM : indépendant des plateformes d'exécution
    - ↪ PSM : dépendant des plateformes (J2EE, .NET, PHP, ...)
  - ▶ Transformation (automatique ou non) de modèles :
    - ↪ PIM ⇒ PSM ⇒ Code exécutable
- ▶ Avantages : Portabilité, interopérabilité, réutilisabilité, amélioration de la qualité, diminution des coûts

## Pour en savoir plus

- ▶ MDA en action : Ingénierie logicielle guidée par les modèles  
Xavier Blanc
- ▶ Model-Driven Software Development  
Thomas Stahl, Markus Völter
- ▶ MDA Guide Version 1.0.1, 2003  
Site de l'OMG  
<http://www.omg.org/cgi-bin/doc?omg/03-06-01>

# Plan du cours

## Introduction

### **Model-Driven Architecture (MDA)**

- Présentation générale de MDA

- Méta-modèles et Meta Object Facility (MOF)

- Object Constraint Language (OCL)

- Profils UML

- XMI/JMI et la sérialisation/manipulation de modèles

- Synthèse

### **Modèles pour l'optimisation et la satisfaction de contraintes**

- Problèmes d'optimisation sous contraintes

- Résolution de problèmes d'optimisation sous contraintes

- Programmation par contraintes

# Plan du cours

## Introduction

### **Model-Driven Architecture (MDA)**

Présentation générale de MDA

Méta-modèles et Meta Object Facility (MOF)

Object Constraint Language (OCL)

Profils UML

XMI/JMI et la sérialisation/manipulation de modèles

Synthèse

### **Modèles pour l'optimisation et la satisfaction de contraintes**

Problèmes d'optimisation sous contraintes

Résolution de problèmes d'optimisation sous contraintes

Programmation par contraintes

# Model Driven Architecture (MDA)

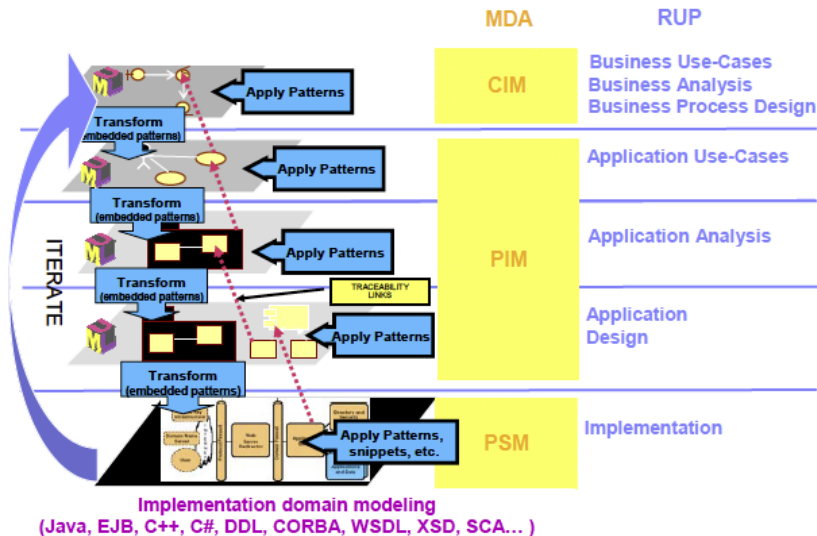
## Processus centré sur les modèles

- ▶ CIM (Computation Independent Model) :  
↪ Modèle pérenne du domaine
- ▶ PIM (Platform Independent Model) :  
↪ Modèle pérenne du système indépdt de plateformes tech.  
↪ Généré à partir du CIM
- ▶ PSM (Platform Specific Model) :  
↪ Dépendant d'une plateforme (ex. : EJB, PHP, .NET)  
↪ Généré à partir du PIM
- ▶ Code exécutable généré à partir du PSM

## Transformation de modèles / Génération de code

- ▶ Idéalement : devrait être automatique et réversible
- ▶ En général : semi-automatique

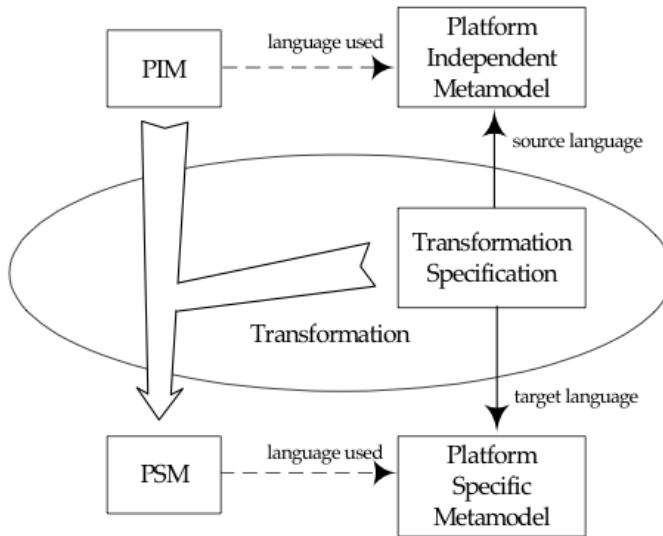
# Automated Model Driven Development selon Rational Rose d'IBM



[Image extraite de "The new IBM Rational design and construction products for Rose and XDE users"]

# Transformation de modèles selon l'OMG

↪ Point déterminant et critique de l'approche MDA !



↪ Méta-modèle = grammaire qui décrit le langage d'un ens. de modèles

# Plan du cours

## Introduction

### **Model-Driven Architecture (MDA)**

Présentation générale de MDA

Méta-modèles et Meta Object Facility (MOF)

Object Constraint Language (OCL)

Profils UML

XMI/JMI et la sérialisation/manipulation de modèles

Synthèse

### **Modèles pour l'optimisation et la satisfaction de contraintes**

Problèmes d'optimisation sous contraintes

Résolution de problèmes d'optimisation sous contraintes

Programmation par contraintes

# Modèles et Méta-modèles

## Qu'est-ce qu'un méta-modèle ?

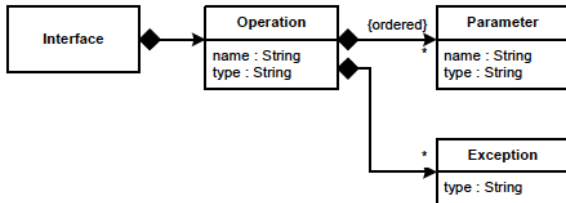
- ▶ Modèle d'un ensemble de modèles  $\rightsquigarrow$  définit la structure
  - ▶ Eléments de base
  - ▶ Relations entre éléments
  - ▶ Contraintes, Règles, ...
- ▶ Méta-modèle  $\simeq$  Grammaire qui définit la syntaxe abstraite

## Pourquoi des méta-modèles ?

- ▶ Pour valider les modèles
- ▶ Pour transformer des modèles / générer du code
  - $\rightsquigarrow$  Règles de transformation entre méta-modèles
- ▶ ...

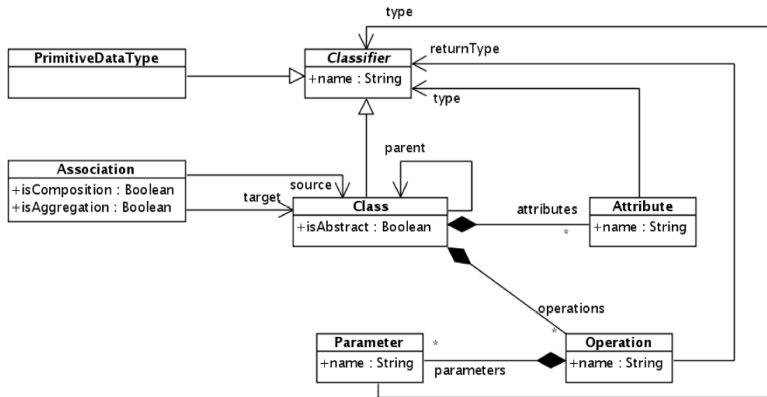
# Exemple: Méta modèle des interfaces

```
interface Sensor {  
    operation start():void;  
    operation stop():void;  
    operation measure():float;  
}  
interface Controller {  
    operation reportProblem(Sensor s,  
                            String errorDesc ):void;  
}
```



[Image empruntée à M. Völter]

# Exemple: Méta modèle des diagrammes de classes (simplifié)



[Image empruntée à [www.eclipse.org](http://www.eclipse.org)]

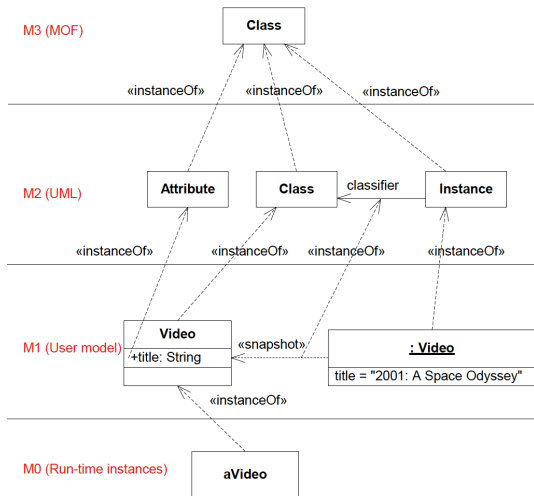
# 4 niveaux de (méta-)modélisation selon l'OMG

- ▶ M3 : Méta-méta-modèle des méta-modèles de M2... et de M3

- ▶ M2 : Méta-modèles des modèles de M1

- ▶ M1 : Modèles (Diagrammes de classes, de séquence, ...)

- ▶ M0 : Instances des modèles à l'exécution

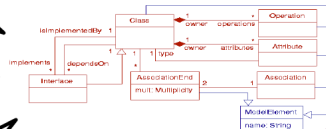


[Image empruntée à [www.omg.org](http://www.omg.org)]

# Illustration des 4 niveaux par Eric Cariou

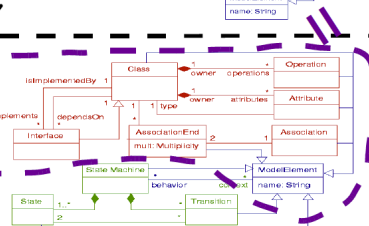
Niveau M3

conforme à



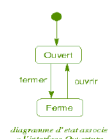
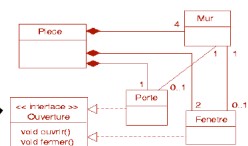
Niveau M2

conforme à



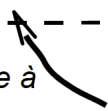
Niveau M1

conforme à



Niveau M0

conforme à



# Description de UML2.0 et MOF2.0 par l'OMG

## Description d'UML2.0

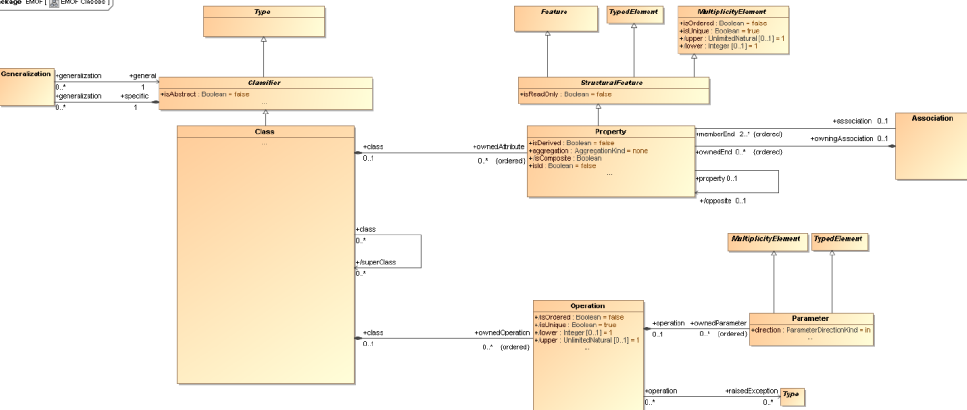
- ▶ UML2.0 Infrastructure
  - ▶ Méta-modèle commun à UML et MOF
    - ↪ Méta-classes partagées : package, class, ...
  - ▶ Constitué d'une trentaine de packages "de base"
    - ↪ *Basic* : décrit diag. de classes sans association
    - ↪ *Constructs* : décrit diag. de classes avec associations
  - ▶ Utilisation de *merge* pour réutiliser les packages
- ▶ UML2.0 Superstructure
  - ▶ Méta modèle d'UML
  - ▶ Intégration de packages de l'infrastructure par *merge*

## Description de MOF2.0

- ▶ Essential MOF (EMOF) : décrit méta-modèles sans association
  - ↪ Intègre le package *Basic* de l'infrastructure
- ▶ Complete MOF (CMOF) : décrit méta-modèles avec associations
  - ↪ Intègre le package *Constructs* de l'infrastructure

# Modèle EMOF de l'OMG

package EMOF [ EMOF Classes ]



[Image empruntée à [www.omg.org](http://www.omg.org)]

# Plan du cours

## Introduction

### **Model-Driven Architecture (MDA)**

Présentation générale de MDA

Méta-modèles et Meta Object Facility (MOF)

**Object Constraint Language (OCL)**

Profils UML

XMI/JMI et la sérialisation/manipulation de modèles

Synthèse

### **Modèles pour l'optimisation et la satisfaction de contraintes**

Problèmes d'optimisation sous contraintes

Résolution de problèmes d'optimisation sous contraintes

Programmation par contraintes

# Object Constraint Language

## Qu'est-ce qu'une contrainte OCL ?

- ▶ Expression booléenne : Relation qui doit être vérifiée
  - ↪ **Description** de règles
  - ↪ Pas d'effet de bord
- ▶ 3 types de contraintes :
  - ▶ Précondition : vérifiée à l'appel d'une méthode
  - ▶ Postcondition : vérifiée au retour d'une méthode
  - ▶ Invariant : vérifié à tout moment
- ▶ Contexte d'une contrainte : Classe ou Opération
- ▶ Syntaxe :  
`<typeContrainte> <nomContrainte> : <exprBool>`

## Programmation par contrat

Spécifie ce que doit faire la classe (ou l'opération)  
... Mais pas comment le faire !

**La vérification de contraintes est un problème difficile...**

## Exemples d'utilisation d'OCL : Modèle d'une pile

```
context Pile::depile(): Object
  pre nonVide: not estVide()
  post sommetRet: result = self@pre.sommet()
  post taille--: taille() = self@pre.taille()-1
```

```
context Pile::sommet(): Object
  pre nonVide: not estVide()
```

```
context Pile::empile(unObjet: Object): void
  post empileAuSommet: sommet() = unObjet
```

```
context Pile
  inv: taille() >= 0
  inv: taille() = 0 implies estVide()
  inv: estVide() implies taille()=0
```

...

# Exemples extraits de UML Superstructure Specification

## Utilisation d'OCL pour spécifier les méta modèles :

- ▶ Contraintes sur *Interface* :

- ▶ Les opérations doivent être publiques

```
inv:self.operation->forall(f|f.visibility=#public)
```

- ▶ Une interface n'a pas d'attribut

```
inv: self.attributes->isEmpty()
```

- ▶ Contraintes sur *Classifier* : les hiérarchies d'héritage doivent être acycliques

```
inv: not self.allParents()->includes(self)
```

- ▶ Contrainte sur *Constraint* : une contrainte ne peut être appliquée à elle-même

```
inv: not self.constrainedElement->includes(self)
```

## Certaines contraintes ne peuvent être exprimées avec OCL :

- ▶ L'évaluation d'une contrainte ne doit pas avoir d'effet de bord
- ▶ L'évaluation d'une contrainte doit rendre une valeur booléenne
- ▶ ...

# Plan du cours

## Introduction

### **Model-Driven Architecture (MDA)**

Présentation générale de MDA

Méta-modèles et Meta Object Facility (MOF)

Object Constraint Language (OCL)

**Profils UML**

XMI/JMI et la sérialisation/manipulation de modèles

Synthèse

### **Modèles pour l'optimisation et la satisfaction de contraintes**

Problèmes d'optimisation sous contraintes

Résolution de problèmes d'optimisation sous contraintes

Programmation par contraintes

# Profils UML

## Objectifs :

Adapter les méta-modèles d'UML à des domaines particuliers

↪ Domain Specific Language (DSL) pour définir des PSM

↪ Permet la transformation de modèles / génération de code

## Moyens :

Un profil est un package composé de

- ▶ Stéréotypes

- ▶ Classe qui étend un élément existant du méta-modèle

- ↪ classe, association, operation, attribut

- ▶ Nom entre << ... >> ou représentation graphique partic.

- ▶ Peut contenir des attributs

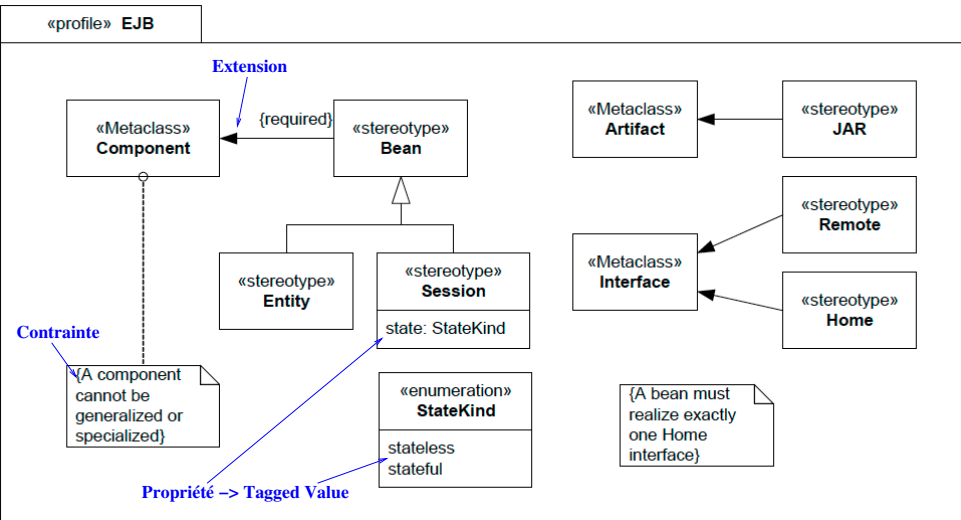
- ↪ Propriétés (tagged values)

- ▶ Contraintes (OCL ou informelles)

- ↪ Conditions d'emploi des stéréotypes et tagged values

Un package définissant un nouveau profil est stéréotypé <<profile>>

# Exemple : Profil EJB (Enterprise JavaBean)



[Image empruntée à [www.omg.org](http://www.omg.org)]

# Plan du cours

## Introduction

### **Model-Driven Architecture (MDA)**

Présentation générale de MDA

Méta-modèles et Meta Object Facility (MOF)

Object Constraint Language (OCL)

Profils UML

XMI/JMI et la sérialisation/manipulation de modèles

Synthèse

### **Modèles pour l'optimisation et la satisfaction de contraintes**

Problèmes d'optimisation sous contraintes

Résolution de problèmes d'optimisation sous contraintes

Programmation par contraintes

# Sérialisation de modèles et XMI

## Objectifs :

Stocker et échanger des modèles

## Moyen : description de modèles avec XML

- ▶ XML (eXtensible Markup Language) du W3C :
  - ↪ Format pour la représentation textuelle de données struct.
  - ↪ Utilisation de balises `< xxx > ... < /xxx >`
- ▶ Grammaire définissant la validité d'un document XML:
  - ▶ DTD (Document Type Definition) :
    - ↪ Relations d'inclusion entre balises
  - ▶ XML Schema :
    - ↪ Doc XML définissant une structuration de doc. XML
- ▶ XMI (XML Metadata Interchange) de l'OMG :
  - ▶ Standard pour représenter des modèles au format XML
    - ↪ Sérialisation de modèles
  - ▶ Utilisation du métamodèle pour définir le DTD / Schema
    - ↪ Génération automatique du DTD à partir du métamodèle

# Manipulation de modèles et JMI

## Objectif :

Manipuler les modèles avec un langage orienté objet

## Moyen : API Java JMI (Java Metadata Interface)

- ▶ Interfaces Java dédiées à un métamodèle (tailored) :  
Opérations pour ajouter/modifier/supprimer/accéder aux éléments du modèle
- ▶ Interfaces Java réflexives : utilisables pour tout modèle

# Plan du cours

## Introduction

### **Model-Driven Architecture (MDA)**

Présentation générale de MDA

Méta-modèles et Meta Object Facility (MOF)

Object Constraint Language (OCL)

Profils UML

XMI/JMI et la sérialisation/manipulation de modèles

Synthèse

### **Modèles pour l'optimisation et la satisfaction de contraintes**

Problèmes d'optimisation sous contraintes

Résolution de problèmes d'optimisation sous contraintes

Programmation par contraintes

# Les standards de l'OMG

## Très nombreux standards

- ▶ UML (Unified Modeling Language) :
  - ↪ Langage pour décrire les modèles
- ▶ MOF (Meta Object Facility) :
  - ↪ Langage pour décrire les méta-modèles
  - ↪ Peuvent être étendus par des profils / plateformes techniques
- ▶ XMI (XML Metadata Interchange) :
  - ↪ Règles pour représenter tout modèle au format XML
- ▶ JMI (Java Metadata Interface) :
  - ↪ Règles pour représenter tout modèle au format Java
- ▶ OCL (Object Constraint Language) :
  - ↪ Expressions pour modéliser des propriétés (pré/post/inv)
- ▶ ...

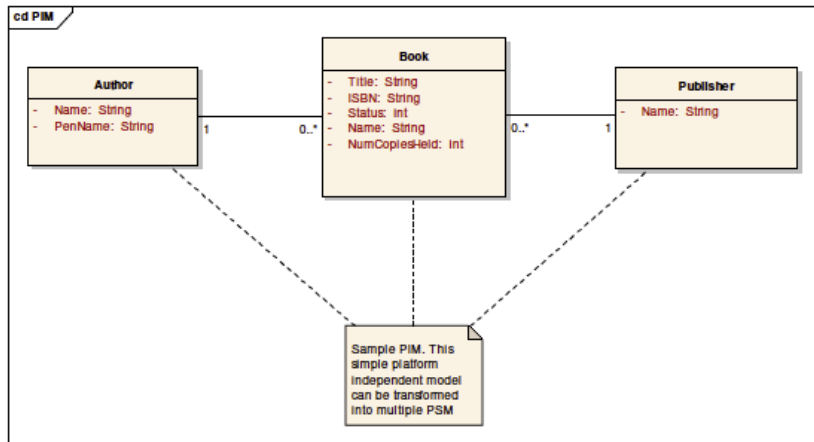
## Intégrés dans de nombreux IDE :

- ↪ Rational Rose, Poseidon, ArgoUML, ...
- ↪ Différents niveaux d'exploitation...

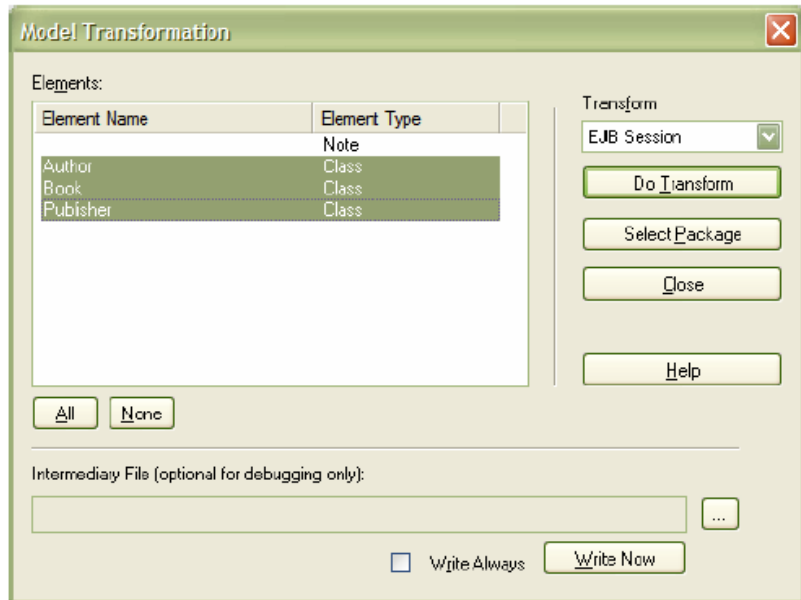
# Exemple d'outil : Enterprise Architect (1/5)

## *The Platform Independent Model*

This diagram shows a simple PIM with three classes, showing the abstract properties and relationships between an Author, their Books and their Publisher. This PIM has no platform specific properties, but is truly an abstract representation.



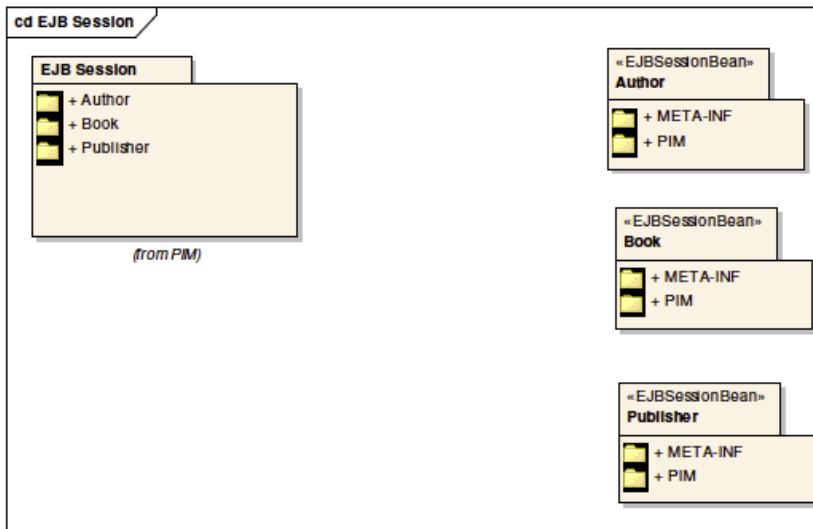
## Exemple d'outil : Enterprise Architect (2/5)



# Exemple d'outil : Enterprise Architect (3/5)

## *Packages Created During Example Transform*

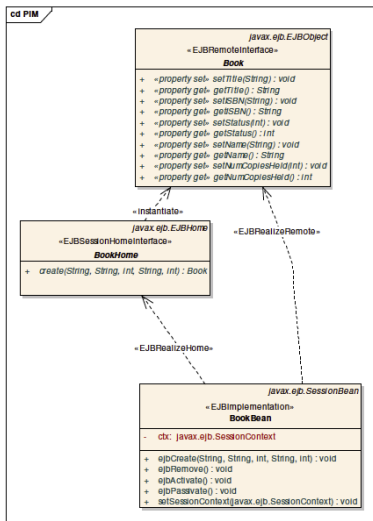
For the EJB Session transform, this diagram shows the PSM packages created – one for each EJB Session bean created from the base PIM classes.



# Exemple d'outil : Enterprise Architect (4/5)

## Example Bean Created from MDA Transformation

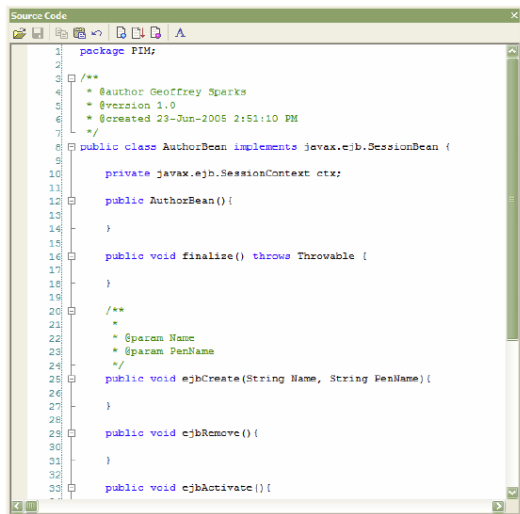
This diagram shows an example of the EJB Session Bean PSM. As a PSM is tightly bound to the target domain, Java and EJB specific properties, code and settings are included in the model. These are logically derived during the transform according to the rules of the transform template.



# Exemple d'outil : Enterprise Architect (5/5)

## Source Code Generated from PSM Using Automatic Code Generation

As the PSM contains specific platform properties, the source code generated can be tailored to the target platform quite tightly. This results in high quality code requiring less manual work and output that is in line with your company's coding standards.

A screenshot of a 'Source Code' editor window. The window title is 'Source Code' and it has a standard toolbar with icons for file operations and editing. The code is displayed on a white background with a vertical scrollbar on the right. The code is as follows:

```
1 package PIM;
2
3 /**
4  * @author Geoffrey Sparks
5  * @version 1.0
6  * @created 23-Jun-2005 2:51:10 PM
7  */
8 public class AuthorBean implements javax.ejb.SessionBean {
9
10     private javax.ejb.SessionContext ctx;
11
12     public AuthorBean() {
13
14     }
15
16     public void finalize() throws Throwable {
17
18     }
19
20     /**
21     *
22     * @param Name
23     * @param PenName
24     */
25     public void ejbCreate(String Name, String PenName){
26
27     }
28
29     public void ejbRemove() {
30
31     }
32
33     public void ejbActivate() {
```

# Plan du cours

## Introduction

### Model-Driven Architecture (MDA)

Présentation générale de MDA

Méta-modèles et Meta Object Facility (MOF)

Object Constraint Language (OCL)

Profils UML

XMI/JMI et la sérialisation/manipulation de modèles

Synthèse

### Modèles pour l'optimisation et la satisfaction de contraintes

Problèmes d'optimisation sous contraintes

Résolution de problèmes d'optimisation sous contraintes

Programmation par contraintes

# Plan du cours

## Introduction

### **Model-Driven Architecture (MDA)**

Présentation générale de MDA

Méta-modèles et Meta Object Facility (MOF)

Object Constraint Language (OCL)

Profils UML

XMI/JMI et la sérialisation/manipulation de modèles

Synthèse

### **Modèles pour l'optimisation et la satisfaction de contraintes**

Problèmes d'optimisation sous contraintes

Résolution de problèmes d'optimisation sous contraintes

Programmation par contraintes

# Définition

## Modèle mathématique d'un pb d'optimisation sous contraintes

↪ Quadruplet  $(X, D, C, F)$  tel que

- ▶  $X$  = ensemble de variables (inconnues du problème)
- ▶  $D$  = fct associant à chaque variable un ensemble de valeurs  
↪  $D(x_i)$  = domaine de  $x_i$  = ens. des valeurs que  $x_i$  peut prendre
- ▶  $C$  = contraintes du problème  
↪ Contrainte = relation entre des variables de  $X$   
↪ Restreint les valeurs pouvant être affectées à ces variables
- ▶  $F$  = fct objectif associant à une valuation de  $X$  une valeur réelle

## Objectif

Affecter une valeur à chaque variable de sorte que

- ▶ Chaque variable est affectée à une valeur de son domaine
- ▶ Toutes les contraintes de  $C$  sont satisfaites
- ▶  $F$  est maximisée (ou minimisée)

## Quelques cas particuliers

- ▶ Pas de contraintes  $C = \emptyset$   
↪ Problème d'optimisation
- ▶ Pas de fonction objectif  $F = 0$   
↪ Problème de satisfaction de contraintes
- ▶ Domaines de  $D$  discrets (énumérables)  
↪ Problème combinatoire
- ▶  $F$  linéaire,  $D = \mathbb{R}$  et  $C =$  ensemble d'inéquations linéaires  
↪ Problème d'optimisation linéaire (programmation linéaire)
- ▶  $F$  linéaire,  $D = \mathbb{Z}$  et  $C =$  ensemble d'inéquations linéaires  
↪ Problème d'optimisation linéaire en nombres entiers
- ▶  $F$  linéaire,  $D = \{0, 1\}$  et  $C =$  ensemble d'inéquations linéaires  
↪ Problème de sac-à-dos multidimensionnel
- ▶  $F$  quadratique,  $D = \mathbb{R}$  et  $C =$  ensemble d'inéquations linéaires  
↪ Problème d'optimisation quadratique (prog. quadratique)
- ▶ ...

# Très très nombreuses applications

- ▶ Conception d'emplois du temps, affectation de ressources
- ▶ Ordonnancement de tâches
- ▶ Tournées de véhicules, voyageur de commerce
- ▶ Découpe de pièces, chargement de véhicules
- ▶ Optimisation du trafic (avions, trains, voitures, fret, ...)
- ▶ ...
- ▶ et développement durable !

# Exemple d'application / Optimisation linéaire

## Description du problème :

Données :

- ▶ Prix de vente du blé ( $p_b$ ) et du maïs ( $p_m$ ) par hectare planté
- ▶ Conso. d'engrais par hectare de blé ( $e_b$ ) et de maïs ( $e_m$ )
- ▶ Conso. d'insecticide par hectare de blé ( $i_b$ ) et de maïs ( $i_m$ )
- ▶ Quantités d'engrais ( $q_e$ ) et d'insecticide ( $q_i$ ) disponibles
- ▶ Taille ( $t$ ) du terrain en hectares

Pb : combien planter de blé et de maïs pour maximiser le profit ?

## Modèle ( $X, D, C, F$ ) :

- ▶ Variables :  $X = \{x_b, x_m\}$
- ▶ Domaines :  $D(x_b) = D(x_m) = \mathbb{R}$
- ▶ Contraintes :  $C =$

$$x_b + x_m \leq t \quad // \text{Borne sur la taille du terrain}$$

$$e_b x_b + e_m x_m \leq q_e \quad // \text{Borne sur la quantité d'engrais}$$

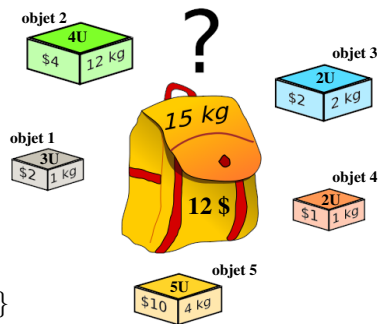
$$i_b x_b + i_m x_m \leq q_i \quad // \text{Borne sur la quantité d'insecticide}$$

$$x_b \geq 0 \quad // \text{Quantité de blé planté non négative}$$

$$x_m \geq 0 \quad // \text{Quantité de maïs planté non négative}$$

- ▶ Fonction objectif :  $F(x_b, x_m) = p_b x_b + p_m x_m$

## Exemple d'application / Sac à dos multidimensionnel



**Modèle  $(X, D, C, F)$  :**

- ▶ Variables :  $X = \{x_1, x_2, x_3, x_4, x_5\}$
- ▶ Domaines :  $\forall x_i \in X, D(x_i) = \{0, 1\}$
- ▶ Contraintes :  $C =$

$$2x_1 + 4x_2 + 2x_3 + x_4 + 10x_5 \leq 12 \quad // \text{Borne sur le prix}$$

$$x_1 + 12x_2 + 2x_3 + x_4 + 4x_5 \leq 15 \quad // \text{Borne sur le poids}$$

- ▶ Fonction objectif à maximiser :

$$F(X) = 3x_1 + 4x_2 + 2x_3 + 2x_4 + 5x_5$$

## Exemple d'application / Problèmes de satisfaction de contraintes

- ▶ Ensemble de voitures à séquencer sur une chaîne de montage



- ▶ Contraintes d'espacement liées aux options à installer

$$\text{blue car} \leq 1/2 ; \text{red car} \leq 2/5 ; \text{green car} \leq 1/5 ; \text{white car} \leq 1/3$$

- ▶ Exemple de solution



### Modèle $(X, D, C)$

- ▶ Variables :  $X = \{C_i \mid 1 \leq i \leq 10\} \cup \{O_i^j \mid 1 \leq i \leq 10, j \in \{b, r, v, c\}\}$
- ▶ Domaines :  $\forall i, D(C_i) = \{b, r, v, bc, rc\}$  et  $\forall i, \forall j, D(O_i^j) = \{0, 1\}$
- ▶ Contraintes :
  - ▶  $(\sum_{i=1}^{10} C_i = b) = 4 \wedge (\sum_{i=1}^{10} C_i = bc) = 1 \wedge \dots$
  - ▶  $\forall i, O_i^b = 1 \Leftrightarrow C_i \in \{b, bc\} \wedge O_i^r = 1 \Leftrightarrow C_i \in \{r, rc\} \wedge \dots$
  - ▶  $\forall i, O_i^b + O_{i+1}^b \leq 1 \wedge O_i^r + O_{i+1}^r + O_{i+2}^r + O_{i+3}^r + O_{i+4}^r \leq 2 \wedge \dots$

↪ Autres modèles possibles (contraintes globales)

# Complexité

## Certains cas particuliers ont des complexités polynomiales :

- ▶ Programmation linéaire (domaines continus)
- ▶ Problème d'affectation linéaire
- ▶ 2-SAT
- ▶ Quelques problèmes dans les graphes :  
ACM, Plus courts chemins, Coupure minimale, ...
- ▶ ...

## Ils sont bien souvent NP-difficiles dans le cas général :

- ▶ SAT, 3-SAT, Planar-3-SAT, ...
- ▶ Programmation linéaire en nombres entiers, Sac-à-dos
- ▶ Nombreux problèmes dans les graphes :  
Coloriage, Voyageur de commerce, Cliques/Stables, ...
- ▶ CSP finis (contraintes quelconques)
- ▶ ...

## Dans certains cas, ils sont indécidables :

- ▶ Equations diophantiennes
- ▶ CSP continus (contraintes quelconques)
- ▶ ...

# Plan du cours

## Introduction

### **Model-Driven Architecture (MDA)**

Présentation générale de MDA

Méta-modèles et Meta Object Facility (MOF)

Object Constraint Language (OCL)

Profils UML

XMI/JMI et la sérialisation/manipulation de modèles

Synthèse

### **Modèles pour l'optimisation et la satisfaction de contraintes**

Problèmes d'optimisation sous contraintes

Résolution de problèmes d'optimisation sous contraintes

Programmation par contraintes

# Difficulté des problèmes NP-complets/difficiles

- ▶ Croissance exponentielle ?!

$n$	$2^n$	Temps (si $10^9$ instr/seconde)
30	$\approx 10^9$	$\approx 1$ seconde
40	$\approx 10^{12}$	$\approx 16$ minutes
50	$\approx 10^{15}$	$\approx 11$ jours
60	$\approx 10^{18}$	$\approx 32$ ans
70	$\approx 10^{21}$	$\approx 317$ siècles

- ▶ En théorie, on ne pourra traiter des données de taille  $> 50$
- ▶ En pratique, on peut souvent faire bien mieux :
  - ▶ Certaines instances de pb NP-complets peuvent être faciles  
↪ Notion de transition de phase
  - ▶ Certains problèmes NP-difficiles admettent des cas particuliers qui ont des complexités polynomiales
  - ▶ Certains problèmes NP-difficiles sont approximables en temps polynomial (avec garantie sur l'erreur)
  - ▶ Exploration "intelligente" de l'espace de recherche :
    - ↪ Contenir l'explosion en structurant et filtrant l'espace
    - ↪ Contourner l'explosion en faisant des impasses

# Résolution de problèmes sur les réels (continus)

## Méthodes dédiées / cas particuliers

- ▶ Equations linéaires  $\rightsquigarrow$  Gauss
- ▶ Inéquations linéaires  $\rightsquigarrow$  Simplex ou Point intérieur
- ▶ Equations polynomiales  $\rightsquigarrow$  Bases de Groëbner
- ▶ ...

## Méthodes génériques

- $\rightsquigarrow$  Calcul d'hyper-boîtes englobant les solutions
- ▶ Arithmétique des intervalles
- ▶ Branch & Propagate & Bound
- ▶ ...

**Attention : calculs sur les flottants et non les réels...**

# Résolution de problèmes combinatoires (discrets)

## Méthodes dédiées / cas particuliers

- ▶ Arbres couvrants minimaux : Kruskal, Prim
- ▶ Plus courts chemins : Dijkstra, Bellman-Ford
- ▶ Coupure minimale (Flot maximal) : Ford-Fulkerson
- ▶ Problèmes d'affectation : Méthode Hongroise
- ▶ ...

## Méthodes génériques

Exploration de l'ensemble des combinaisons candidates

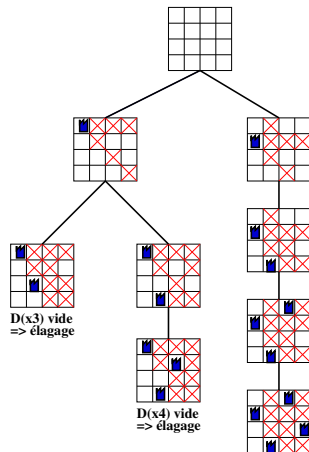
↪ Espace de recherche

- ▶ Approches complètes : Exploration exhaustive
  - + : Preuve d'optimalité
  - : Complexité exponentielle dans le pire des cas
- ▶ Approches incomplètes : Exploration heuristique
  - + : Complexité polynomiale
  - : Pas de garantie d'optimalité

# Méthodes complètes : Contenir l'explosion combinatoire

## Branch & Bound & Propagate

- ▶ Structuration de l'espace de recherche en arbre  
~> Branch
- ▶ Elagage de branches de l'arbre :
  - ▶ Calcul de bornes sur  $F$   
~> Bound
  - ▶ Propagation de  $C$   
~> Propagate
- ▶ Heuristiques pour déterminer dans quel ordre développer l'arbre



Exemple : pb des 4 reines

# Méthodes incomplètes : Contourner l'explosion combinatoire

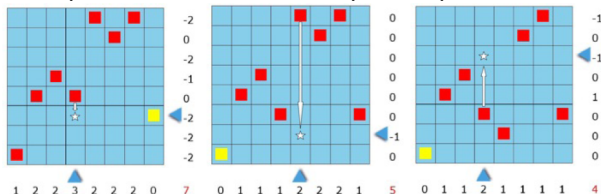
## Exploration guidée par des heuristiques

- ▶ Intensification de la recherche autour des zones "prometteuses"
- ▶ Diversification pour découvrir de nouvelles zones

## Deux familles d'approches incomplètes

- ▶ Perturbatives : modification de combinaisons existantes
  - ▶ Ex: Recherche locale (LS), Algorithmes génétiques (GA), Optimisation par essaims de particules (PSO), ...
- ▶ Constructives : construction de nouvelles combinaisons
  - ▶ Ex: Optimisation par colonies de fourmis (ACO), Algorithmes par estimation de distribution (EDA), ...

Exemple de recherche locale pour le problème des 8 reines :



# Plan du cours

## Introduction

### Model-Driven Architecture (MDA)

Présentation générale de MDA

Méta-modèles et Meta Object Facility (MOF)

Object Constraint Language (OCL)

Profils UML

XMI/JMI et la sérialisation/manipulation de modèles

Synthèse

### Modèles pour l'optimisation et la satisfaction de contraintes

Problèmes d'optimisation sous contraintes

Résolution de problèmes d'optimisation sous contraintes

Programmation par contraintes

# La programmation par Contraintes (CP)

*"Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it."*

Eugene C. Freuder

## En pratique : "CP = Model + Search"

- ▶ Model = Description du problème
  - ▶ Déclaration des variables et de leurs domaines
  - ▶ Déclaration (post) des contraintes
  - ▶ Eventuellement : déclaration d'une fonction objectif
- ▶ Search = Exploration de l'espace de recherche
  - ▶ Utilisation d'algorithmes intégrés
  - ▶ Possibilité de guider la recherche par des heuristiques
  - ▶ Possibilité de définir de nouvelles stratégies

Pas toujours aussi efficace qu'un programme « cousu main »  
... mais tellement plus vite développé !

# Langages et environnements de programmation par Contraintes

- ▶ ALICE [Jean-Louis Laurière, 1976]  
~> Premier système à base de contraintes
- ▶ CHIP, Prolog V, Gnu-Prolog  
~> Extensions de Prolog
- ▶ CHOCO (Java), Gecode (C++)  
~> Bibliothèques open source
- ▶ OPL Development Studio (IBM)  
~> Langage de modélisation + CP + MIP
- ▶ Comet (Dynadec)  
~> Langage de modélisation + CP + CBLS + MIP
- ▶ ...

## Exemple de programme linéaire en OPL

### The production model (gas.mod)

```
{string} Products = ...;
{string} Components = ...;

float Demand[Products][Components] = ...;
float Profit[Products] = ...;
float Stock[Components] = ...;
dvar float+ Production[Products];

maximize
    sum( p in Products )
        Profit[p] * Production[p];
subject to {
    forall( c in Components )
        ct:
            sum( p in Products )
                Demand[p][c] * Production[p] <= Stock[c];
}
```

### Instance data for the production model (gas.dat)

```
Products = { "gas" "chloride" };
Components = { "nitrogen" "hydrogen" "chlorine" };

Demand = [ [1 3 0] [1 4 1] ];
Profit = [30 40];
Stock = [50 180 40];
```

[Programme extrait du manuel utilisateur d'OPL ([www.ibm.com](http://www.ibm.com))]

## Exemple de sac-à-dos en OPL

### A multi-knapsack model (knapsack.mod)

```
int NbItems = ...;
int NbResources = ...;
range Items = 1..NbItems;
range Resources = 1..NbResources;
int Capacity[Resources] = ...;
int Value[Items] = ...;
int Use[Resources][Items] = ...;
int MaxValue = max(r in Resources) Capacity[r];

dvar int Take[Items] in 0..MaxValue;

maximize
    sum(i in Items) Value[i] * Take[i];

subject to {
    forall( r in Resources )
        ct:
            sum( i in Items )
                Use[r][i] * Take[i] <= Capacity[r];
}
```

[Programme extrait du manuel utilisateur d'OPL ([www.ibm.com](http://www.ibm.com))]

## Ex. de programme CP : ordonnancement de voitures en Comet

```
import cotfd;
Solver<CP> cp();

range Cars           = 1..100;
range Configs        = 1..18;
range Options        = 1..5;
int lb[Options]      = [1,2,1,2,1];
int ub[Options]      = [2,3,3,5,5];
int demand[Configs] = [5,3,7,1,10,2,11,5,4,6,12,1,1,5,9,5,12,1];
int requires[Configs,Options] = [[1,1,0,0,1],[1,1,0,1,0],..., [0,0,1,0,0]];
set<int> options[o in Options] = filter (c in Configs)(requires[c,o] == 1);

var<CP>{int} line[Cars](cp,Configs);
solve<cp>
  forall (o in Options)
    cp.post(sequence(line,demand,lb[o],ub[o],options[o]));
using
  labelFF(line);
```

[Programme extrait du tutoriel Comet ([dynadec.com](http://dynadec.com))]

## Exemple de programme CBLIS : les reines en Comet

```
1 import cotls;
2 int    n = 16;
3 range Size = 1..n;
4 UniformDistribution distr(Size);
5
6 Solver<LS> m();
7 var{int} queen[Size](m,Size) := distr.get();
8 ConstraintSystem<LS> S(m);
9
10 S.post(alldifferent(queen));
11 S.post(alldifferent(all(i in Size)(queen[i] + i)));
12 S.post(alldifferent(all(i in Size)(queen[i] - i)));
13 m.close();
14
15 int it = 0;
16 while (S.violations() > 0 && it < 50 * n) {
17     selectMax (q in Size) (S.violations(queen[q]))
18     selectMin (v in Size) (S.getAssignDelta(queen[q],v))
19     queen[q] := v;
20     it++;
21 }
```

[Programme extrait du tutoriel Comet ([dynadec.com](http://dynadec.com))]