Transformations de Modèles
Génération de Code
Editeurs Textuels et Graphiques

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.

M2M: Model To Model

Transformations de Modèles

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

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

- Transformation exogène
  - Les modèles source et cible sont conformes à des méta-modèles différents
  - Exemples : Transformation d'un modèle UML en schéma de BDD

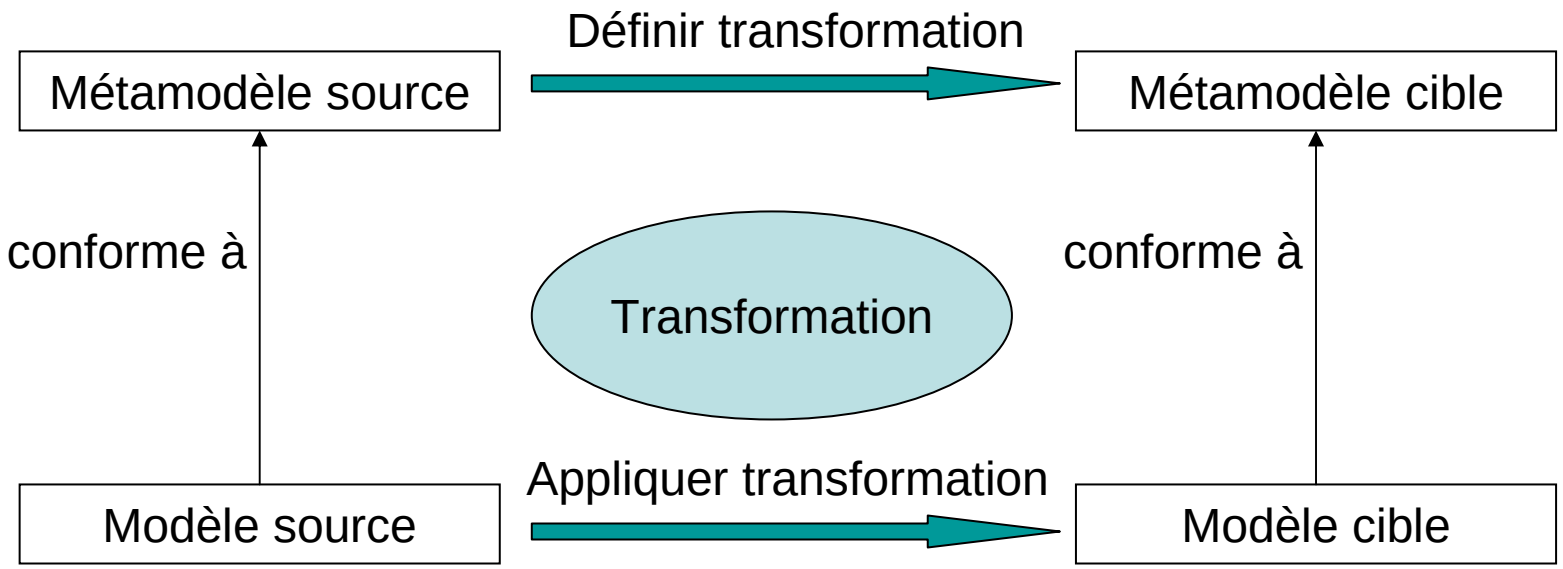
Principe de base

- La transformation de modèle consiste à parcourir un modèle pour le modifier ou pour générer un autre modèle
  - D'un graphe d'objets vers un autre graphe d'objets

- Un transformateur est un programme objet qui visite les objets représentant le modèle et construit de nouveaux objets représentant un autre modèles

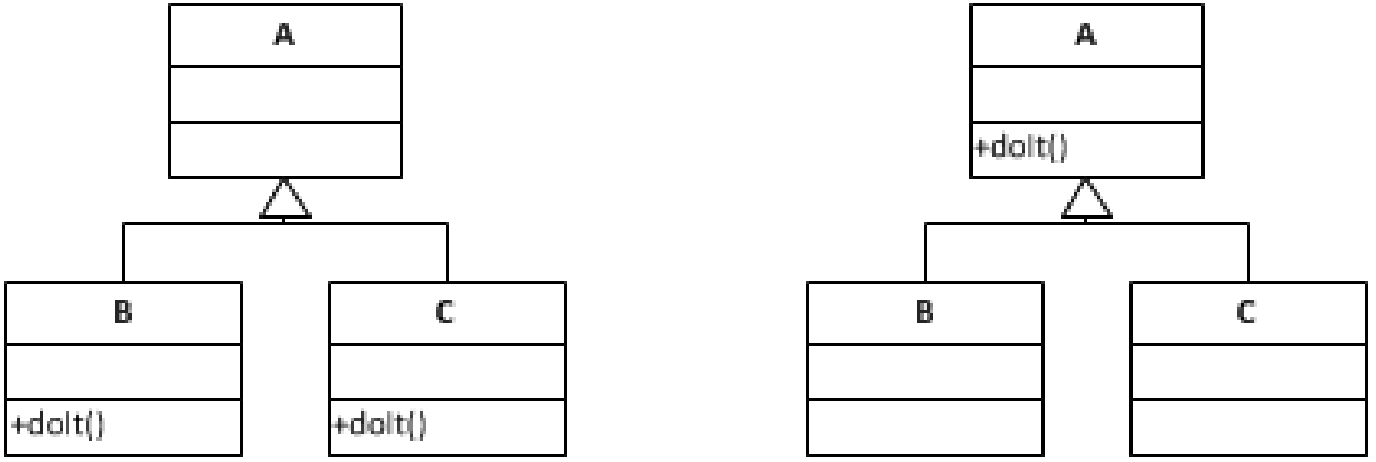
- Les modèles source et cible doivent correspondre à leur Méta-modèles respectifs (peut être le même)

Architecture des transformations



Transfo. Endogène: Exemple

- Si deux classes héritent d'une même classe et qu'elles ont une opération qui a le même nom, il faut remonter l'opération dans la classe mère.



Transfo. Endogène: Exemple

- Transformation d'un diagramme de classes UML vers un schéma de base de données relationnelle

UML

RDBMS

Table PERSON	
PERSONNE _ID(pk)	NAME
1	Dupond
2	Tintin

Table Car	
NUMBER (pk)	PERSONNE_ID (fk)
234 GH 62	2
1526 ADF 77	1

Transformations : types de modèles

- 3 grandes familles de modèles et outils associés
  - Données sous forme de séquence
    - Ex : fichiers textes (AWK)
  - Données sous forme d'arbre
    - Ex : XML (XSLT)
  - Données sous forme de graphe
    - Ex : diagrammes UML
    - Outils
      - Transformateurs de graphes déjà existants ex. AGG
      - Nouveaux outils du MDE et des AGL (QVT, J, ATL, Kermeta ...)

Techniques de transformation

- Principe : découpage des transformations en règles
  - Une règle définit la manière dont un ensemble de concepts du méta-modèle source est transformé (mappé) en un ensemble de concepts du méta-modèle destination.

- Approche déclarative : focalisation sur ce *qui* est créé
  - Fonctionnement :
    - Recherche de certains patrons (d'éléments et de leurs relations) dans le modèle source
    - Chaque patron trouvé est remplacé dans le modèle cible par une nouvelle structure d'élément
    - Le moteur de règle choisi la façon dont sont exécutées les règles (ordre)
  - Propriété :
    + Ecriture de la transformation « assez » simple
    - mais ne permet pas toujours d'exprimer toutes les transformations facilement
    - Pas de maîtrise sur la navigation (le moteur)

Techniques de transformation

- Approche impérative : focalisation sur *comment* la règle est exécutée

  – Proche des langages de programmation

  – On parcourt le modèle source dans un certain ordre (explicite) et on génère le modèle cible lors de ce parcours

  \+ Ecriture transformation plus complexe mais permet de toutes les définir

  \- Savoir naviguer dans le modèle!

Techniques de transformation

- Approche hybride : à la fois déclarative et impérative
  - Celle qui est utilisée en pratique dans la plupart des outils
  - Approche principalement à base de règles
  - Des « helper » sont programmés afin de faciliter la navigation ou la construction de certaines parties du modèle
  - Ces « helper » peuvent être appelés à partir des règles (en navigation ou en construction)

Exemple en impératif (1/2)

Naviguer dans le modèle (exemple transfo Endogène)

1.Héritage puis même opération

1. Parcourir toutes les classes du package
2. Isoler celles qui ont deux classes filles
3. Vérifier que les deux classes filles ont une opération de même nom

•Même opération puis héritage

1. Parcourir 2 à 2 toutes les classes du package
2. Vérifier qu'elles ont une opération de même nom
3. Vérifier qu'elles ont une même classe mère

Même complexité ?

Exemple en impératif (2/2)

Construire le modèle

1.Déplacer une opération

1. Choisir une des deux opération
2. La déplacer dans la classe mère
3. Supprimer l'autre opération

•Construire une nouvelle opération

1. Supprimer les deux opérations
2. En construire une nouvelle de même nom
3. L'ajouter dans la classe mère

Quid des liens existants (les références vers l'opération) ?

Modèle en Java d'un diag. d'états : Pattern Visiteur

```
public class StateMachine {
        protected Vector states = new Vector();
        protected Vector transitions = new Vector();
        public void addState(State s) {
                states.add(s); }
        public void addTransition(Transition t) {
                transitions.add(t); }
... }


public class Transition {
        protected State from, to;
        protected String event;
        public Transition(State from, State to, String evt) {
                this.from = from;
                this.to = to;
                event = evt; }
}
public class State {
        protected String name;
        public State(String name) {
                this.name = name; }
}
```

Modèle en Java d'un diag. d'états

Définition d'un modèle : instances et liaisons de ces 3 classes

```
...
StateMachine sm;
State s1,s2;
...
sm = new StateMachine();
s1 = new State("Ouvert");
s2 = new State("Ferme");
sm.addState(s1);
sm.addState(s2);
sm.addTransition(new Transition(State.Initial, s1, ""));
sm.addTransition(new Transition(s1, s2, "fermer"));
sm.addTransition(new Transition(s2, s1, "ouvrir"));
```

Modèle en Java d'un diag. d'états : Pattern Visiteur

- Ajout de méthodes pour lire les éléments du modèle
  - Ex. pour **StateMachine : getStates(), getTransitions()**
- Parcours du modèle via ces méthodes
  - Approche impérative
- Utilisation possible du patron Visiteur
- Génération d'un nouveau modèle à partir de ce parcours
  - En utilisant éventuellement des méthodes des différentes classes pour gérer la transformation des éléments un par un

```
public class State {
...
public String toXML() {
        return new String("<state>\n\t<name>"+name
                +"</name>\n</state>\n"); } ...
```

Modèle en Java d'un diag. d'états : Pattern Visiteur

- Parcours de type « impératif » du graphe d'objet pour sérialisation en XML

```
String xml = "<statemachine>";
Iterator it = sm.getStates().iterator();
while(it.hasNext())
    xml += ((State)it.next()).toXML();
it = sm.getTransitions().iterator();
while(it.hasNext())
    xml += ((Transition)it.next()).toXML();
xml += "</statemachine>";
System.out.println(xml);
```

Modèle en Java d'un diag. d'états : Pattern Visiteur

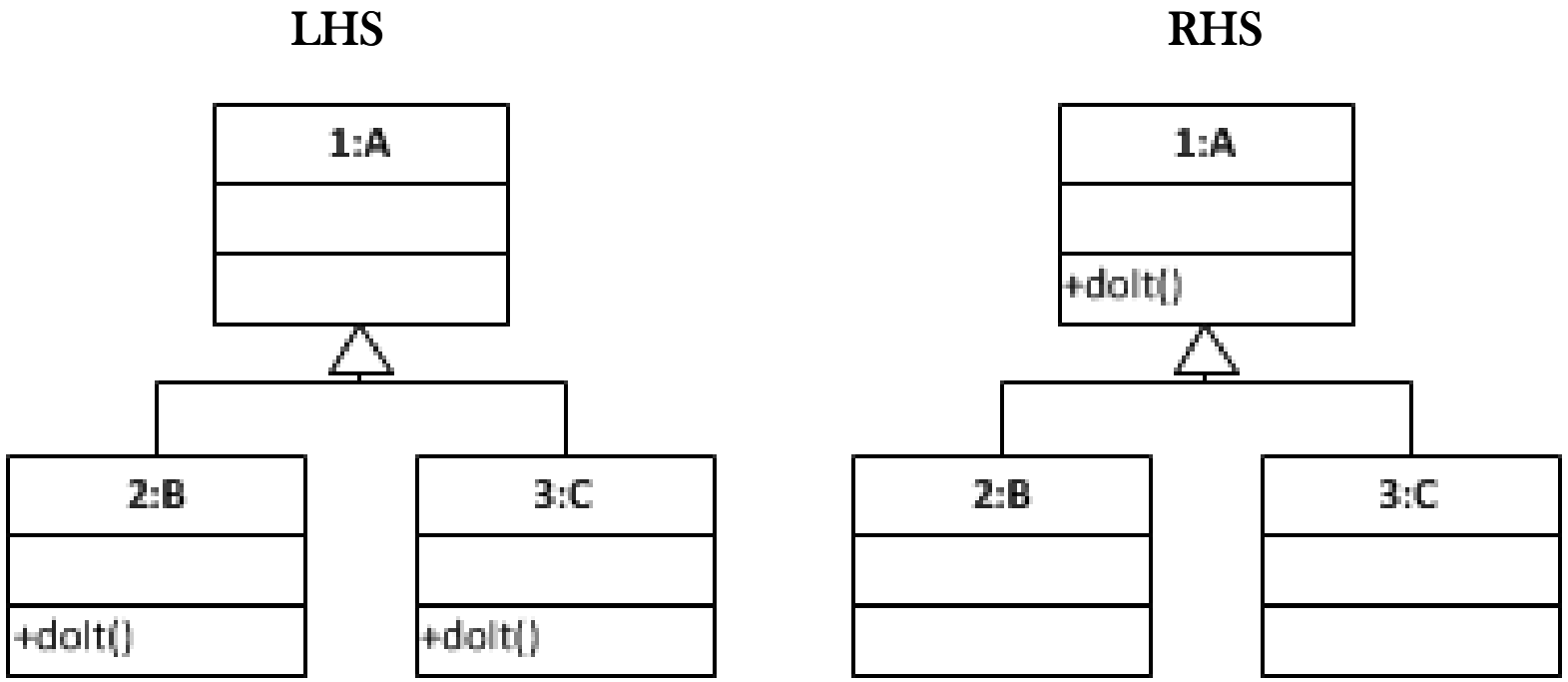
- Résultat sérialisation XML

```
<statemachine>
   <state>
        <name>Ouvert</name>
   </state>
   <state>
        <name>Ferme</name>
   </state>
    <transition>
      <from>initial</from>
      <to>Ouvert</to>
      <event></event>
    </transition>
    <transition>
        <from>Ouvert</from>
        <to>Ferme</to>
        <event>fermer</event>
    </transition>
    <transition>
        <from>Ferme</from>
        <to>Ouvert</to>
        <event>ouvrir</event>
    </transition>
</statemachine>
```

Approche Déclarative: L'approche AGG

- Règles de correspondance avec un membre gauche et un membre droit
- Le membre gauche d'une règle précise les éléments devant être absolument présent pour déclencher la règle
- Le membre droit d'une règle précise les éléments devant être absolument présent après le déclenchement de la règle
- Si un élément appartient :
  - aux deux membres (spécifié à l'aide d'un id), il est préservé
  - qu'au membre droit, il est construit
  - qu'au membre gauche, il est supprimé

Approche Déclarative: Exemple avec AGG

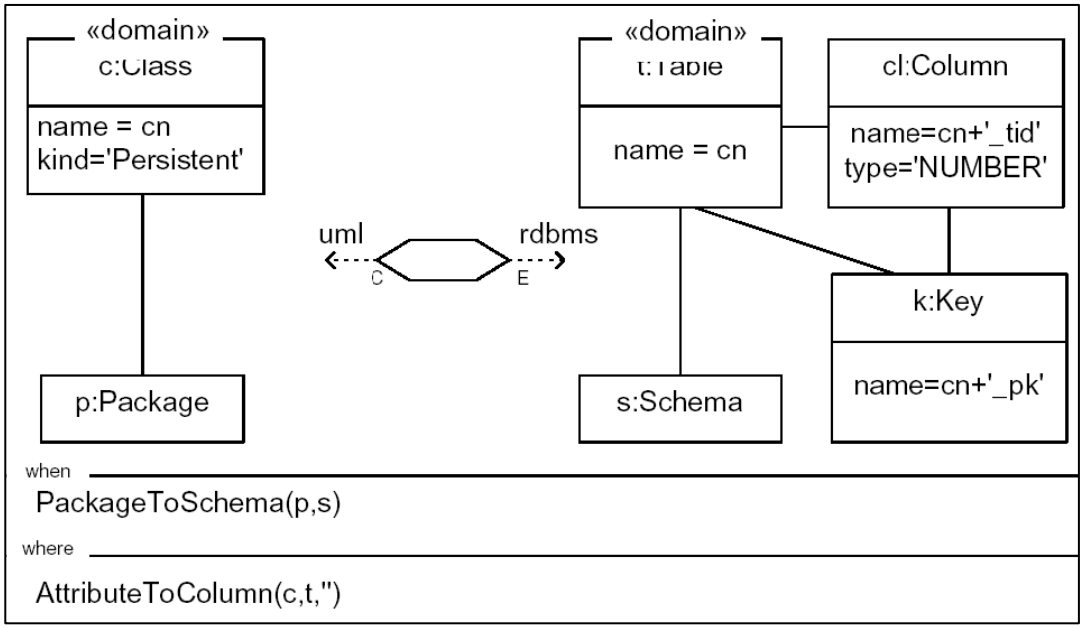


Comment se font la navigation et la construction ?

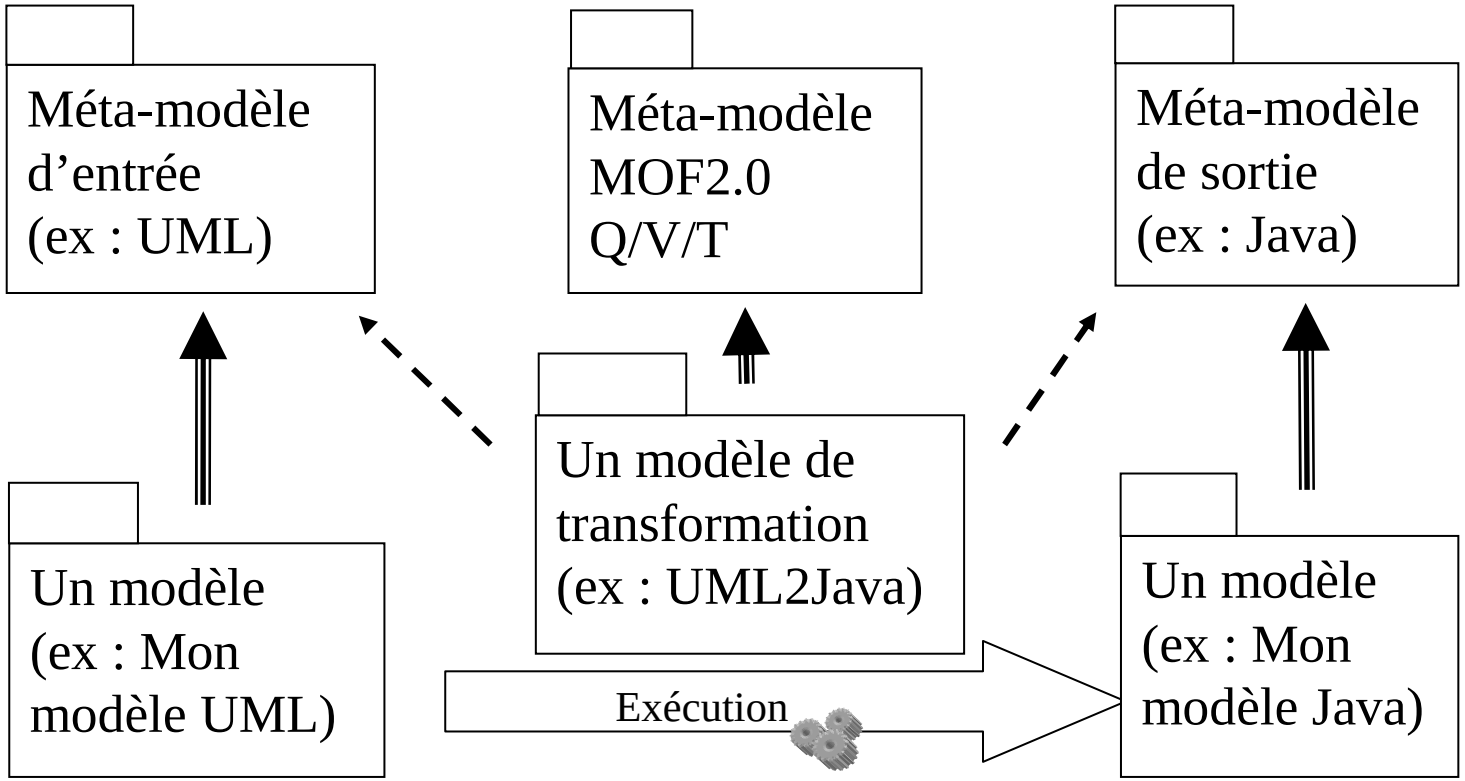
Modélisation des transformations

- Une transformation est une sorte de programme qui manipule des modèles
- Il est donc possible de modéliser ce programme
- Il est donc possible de définir le méta-modèle correspondant

⇒ La transformation devient un modèle



Méta-modèle de transformation



Maintenance du code

- L'approche déclarative semble beaucoup plus intéressante pour la maintenance du code
- Pour autant, les langages à base de règles restent propriétaires
  - Pour 1 développeur AGG combien de développeurs Java ?
- Plus une génération est complexe, plus il faut gérer un ensemble de règles ou un ensemble de classes
- Les tests et le debugage sont plus outillés dans les langage de programmation classiques (Java)

Performance des transformateurs

- La performance est un critère lorsque les transformateurs doivent parcourir de grands modèles

- La gestion de la performance en Java souffre de la montée en mémoire du modèle
  - Utilisation de cache

- Les langages déclaratifs ne précisent pas souvent la façon dont ils s'exécutent. De plus, leurs algorithmes de résolution de règles sont parfois très consommateurs.

Adaptabilité

- Pour couvrir tous les besoins des utilisateurs, une transformation doit pouvoir être adaptable
  - Paramètres d'exécution, ouverture

- Le langages par règles offrent bien souvent des mécanismes dédiés à cette adaptabilité
  - Variable d'entrée, appel de règles, héritage de règles

- Les générateurs Java peuvent être adaptables si leur architecture le prévoit

Support de l'évolution du texte

- Un modèle transformé tout comme le modèle source de la transformation peuvent subir des évolutions
- Comment assurer une synchronisation des modèles ?
- Quelles propriétés attend-t-on de la synchronisation ?

Transformation bi-directionnelle

- L'objectif est de définir deux transformations (une par sens)
- Et surtout de faire en sorte que ces deux transformations convergent
- A chaque modification d'un des deux modèles, il faut alors appliquer la transformation correspondante

Des Langages de Transformation

- QVT (Query/View/Transformation)
  - Standard OMG
  - Plusieurs implementations (exp. Borland QVTO, SmartQVT)
  - Hybride
- ATL
  - Initiative académique (INRIA, Nantes)
  - Projet Eclipse
- Kermeta
  - INRIA Rennes (Projet Triskell)

M2T: Model To Text (Code)

Principe de base

- La génération de code consiste à parcourir un modèle afin de générer du texte
  - D'un graphe d'objets vers une séquence de caractères
- Un générateur est un programme objet qui visite les objets représentant le modèle et écrit du texte
- Un standard OMG: MOF to Text

Exemple UseCase vers Texte

```
Système s = loadXMI(« monModel.xmi »);
print(« le système a pour nom »+s.name);
for (int i=0 ; i < s.cas.size() ; i++) {
   print(« il possède un cas intitulé »+cas[i].intitule);
}
```

Le système a pour nom PetStore
Il possède un cas intitulé Commander Panier
Il possède un cas intitulé Valider Article

L'approche Template

- Les programmes « générateur de code » ont pour inconvénient de rendre peu lisible le texte généré
- Cet argument est important pour les phases de développement et de maintenance
- L'approche par **Template** permet au développeur de préciser le texte généré en définissant des **Variables**
- Ces variables seront affectées par le modèle

Exemple avec JET

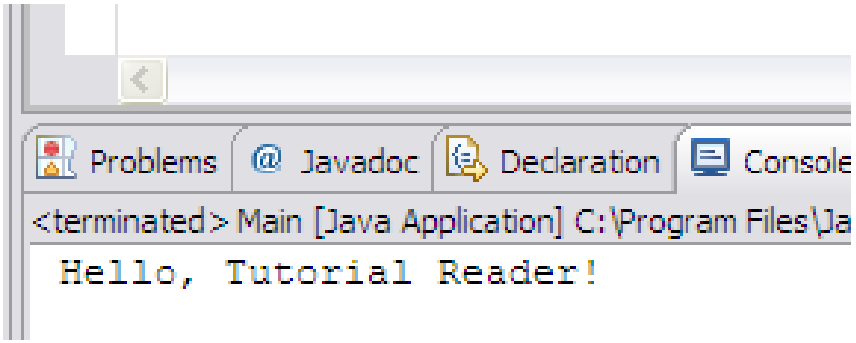
Fichier contenant le template (greetings.txtjet)

```
<%@ jet package="hello" class="GreetingTemplate" %>
 Hello, <%=argument%>!
```

Client Java (main):

```
    GreetingTemplate sayHello = new GreetingTemplate();
     String result2 = sayHello.generate("Tutorial Reader");
     System.out.println(result2);
```

Résulat sur la console:



Maintenance du code

- L'approche par template semble beaucoup plus intéressante pour la maintenance du code

- Pour autant, les langages template restent propriétaires
  - Pour 1 développeur template combien de développeurs Java ?

- Plus une génération est complexe, plus il faut gérer un ensemble de templates ou un ensemble de classes

- Les tests et le debugage sont plus outillés dans les langage de programmation classiques (Java)
  - Dans les langages à Templates, on retrouve l'erreur en regardant le code généré et non pas celui du template

Performance des générateurs

- La performance est un critère lorsque les générateurs doivent parcourir de grands modèles

- La gestion de la performance en Java souffre de la montée en mémoire du modèle
  - Utilisation de cache

- Les langages template ne précisent pas souvent la façon dont ils s'exécutent

Support de l'évolution du texte

- Un texte généré tout comme le modèle source de la génération peuvent subir des évolutions

- Comment assurer une synchronisation du texte et du modèle ?

L'approche par marqueur

- Le générateur de code écrit des marqueurs dans le texte
- Ces marqueurs délimitent des zones dans lesquels le texte peut être changé
- Ce texte ne sera pas modifié lors de la re-génération
- Cette approche assure une synchronisation du modèle vers le texte
  - generated dans EMF
  - Identifiants dans Objecteering (projet round-trip)

Les générateurs du moment

- JET (Java Emitter Templates)
  - Pionnier dans les projets Eclipse
  - Utilisé pour la génération de code dans EMF
  - Facile à utiliser, JSP-like
  - JET 2 en cours de développement

- Xpand/Xtend/Workflow
  - Nouveau projet Eclipse
  - Très prometteur
  - Un peu plus complexe mais permet d'exprimer des générations très compliquées

- MTL
  - Implémentation du standard OMG: MOF to Text
  - Outilleur: Acceleo

Graphical Modeling Framework

Framework d'éditeur de diagrammes

- GMF est un Framework permettant la génération d'éditeur de diagrammes

- Il est basé sur EMF
  - le framework Eclipse de base pour la modélisation

- Il est basé sur GEF
  - Un framework Eclipse de composants graphiques

- A été utilisé pour générer un éditeur de modèle UML2
  - Les modèles UML2 sont très complexe en terme d'IHM

3 concepts de base

- Le domaine
  - Le domaine est spécifié par un méta-modèle qui précise tous les concepts du langage de modélisation
- La définition des entités graphiques
  - Les entités graphiques sont des entités GEF
- La définition de l'outil d'édition
  - L'outil est spécifié afin de préciser les services qu'il offre

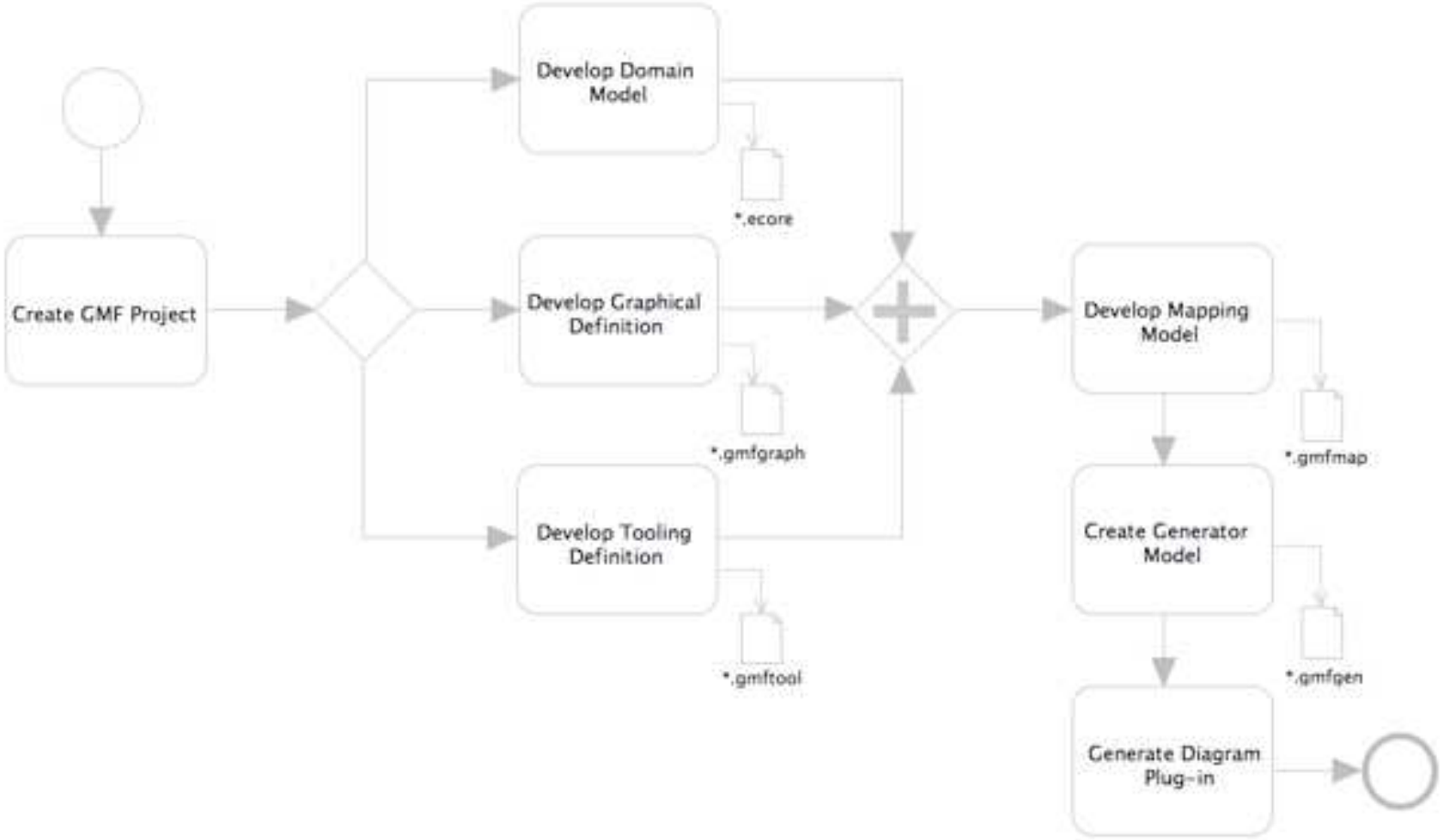
Génération d'un plugin

- Les 3 concepts de base (domaine, entité et éditeur) sont liées afin de préciser leurs relations

- Le modèle de liens permet la génération d'un plugin Eclipse constituant l'éditeur graphique

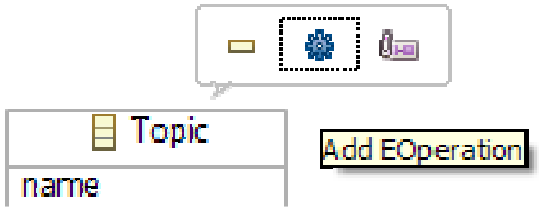
- L'éditeur généré peut être adapté afin d'inclure des fonctionnalités particulières

Le Workflow de GMF

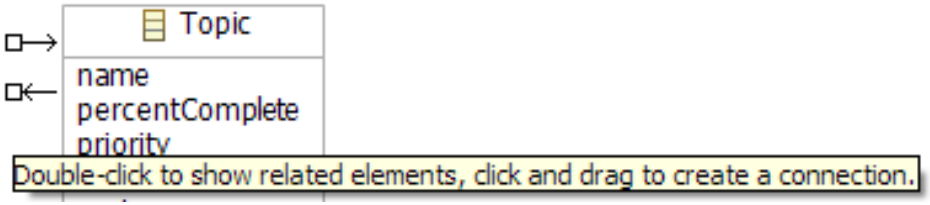


Composants de l'éditeur (1/3)

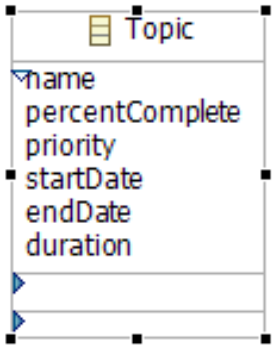
Action Bars:



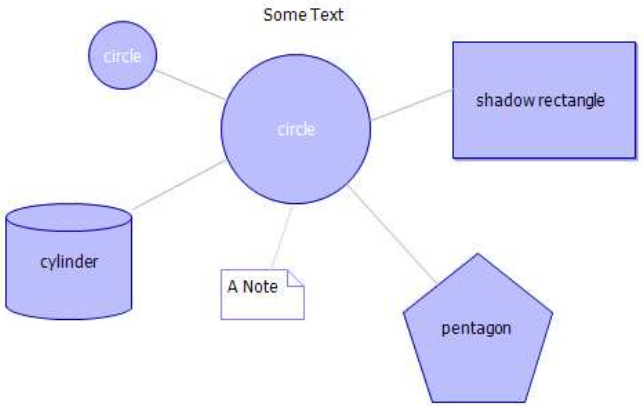
Connection Handles:



Collapsed Compartments:



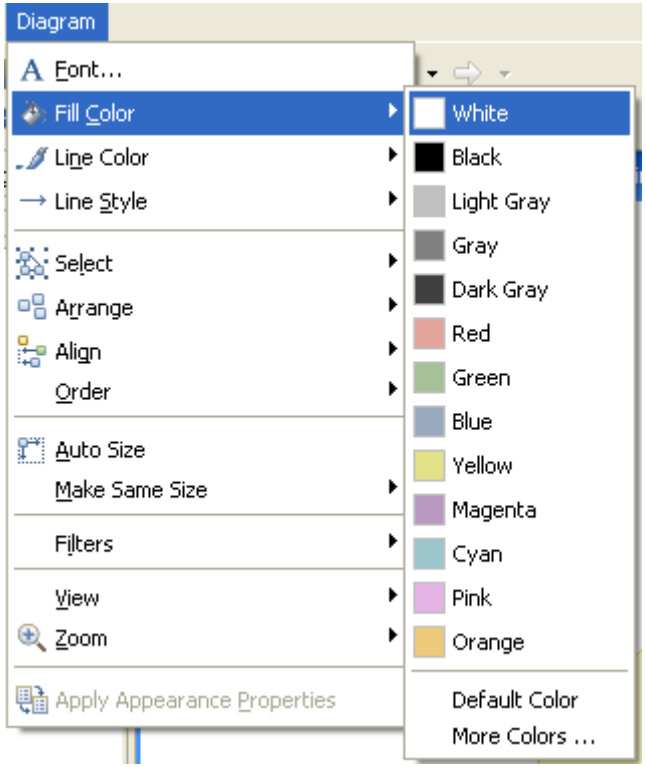
Geometrical Shapes:



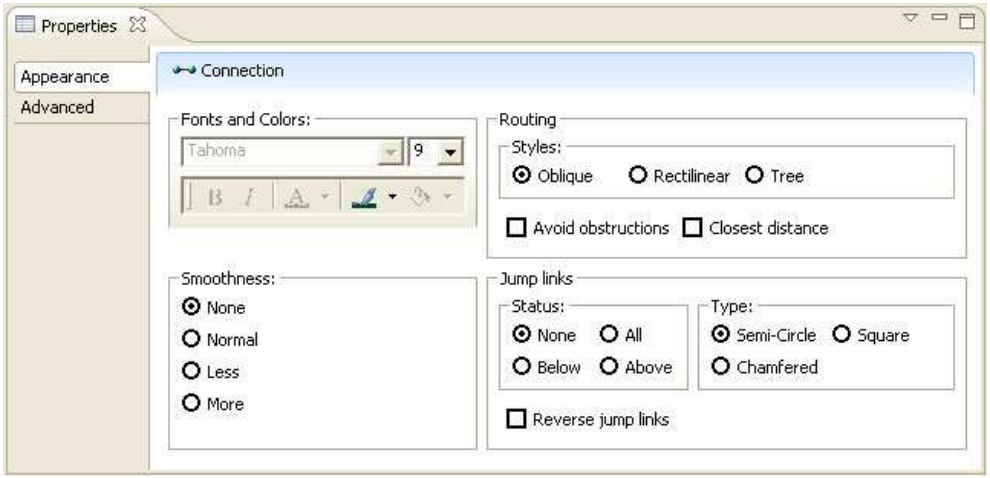
Composants de l'éditeur (2/3)

Toolbar:

Actions:

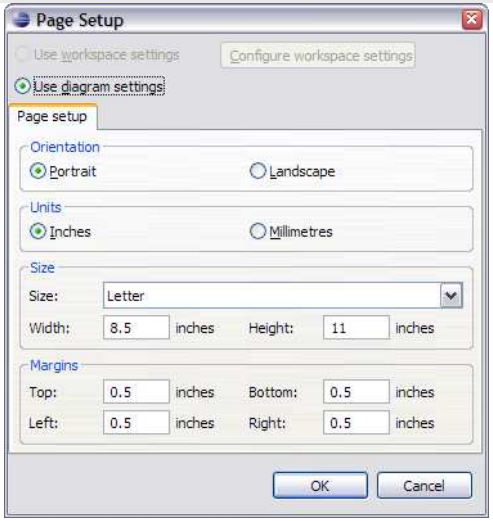


Properties View:

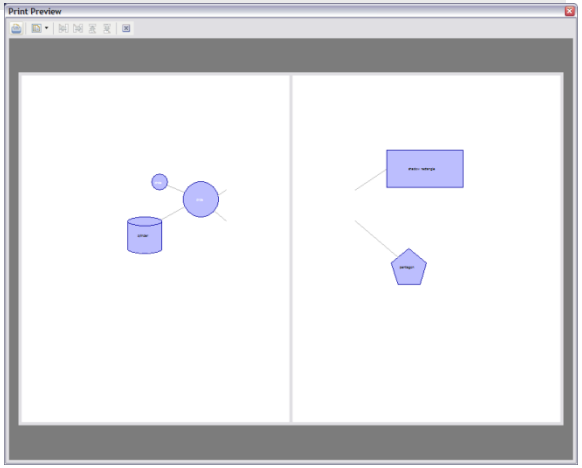


Composants de l'éditeur (3/3)

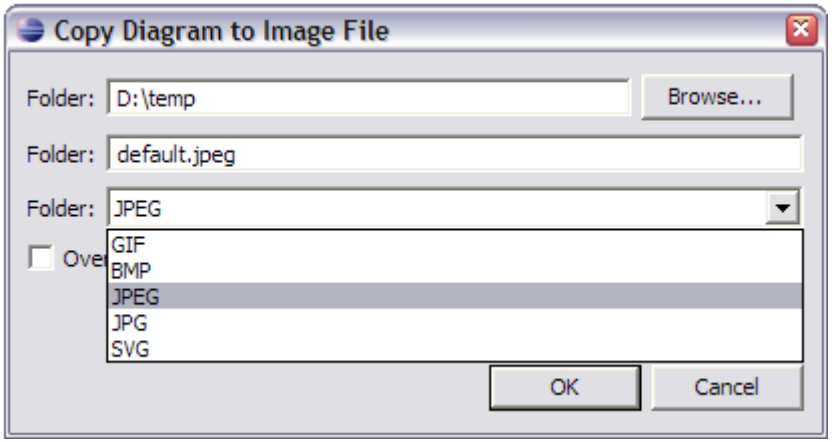
Page Setup:



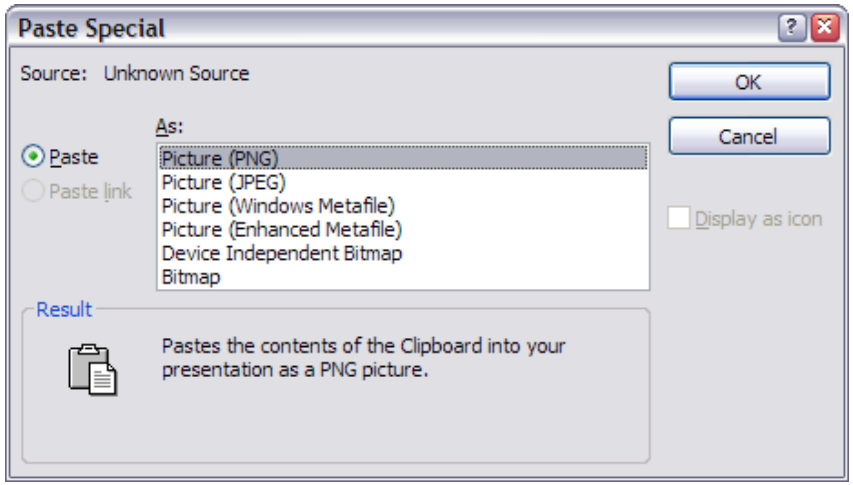
Print Preview:



"Copy Diagram to Image File" formats:



System clipboard formats:



Synthèse

- Un langage de modélisation sans éditeur n'est pas réellement exploitable pour des utilisateurs
- Grace à GMF, il est possible de prototyper rapidement des éditeurs de diagrammes
- La définition d'un éditeur industriel nécessite une forte adaptation du code généré par GMF

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