
# Un panorama des techniques de transformation de modèles

*Groupe de travail WP5 TOPCASED*

# Participants

| | |
|---|---|
| Pierre Bazex | FéRIA / IRIT |
| Laurent Blondon | Sodifrance |
| Benoît Combemale | FéRIA / IRIT |
| Xavier Crégut | FéRIA / IRIT |
| Patrick Farail | Airbus |
| Louis Féraud | FéRIA / IRIT |
| Gilles Hannoyer | Siemens VDO |
| Adel Ouardani | FéRIA / LAAS |
| Mario Paludetto | FéRIA / LAAS |
| Marc Pantel | FéRIA / IRIT |
| Jean-Claude Pascal | FéRIA / LAAS |
| Christian Percebois | FéRIA / IRIT |

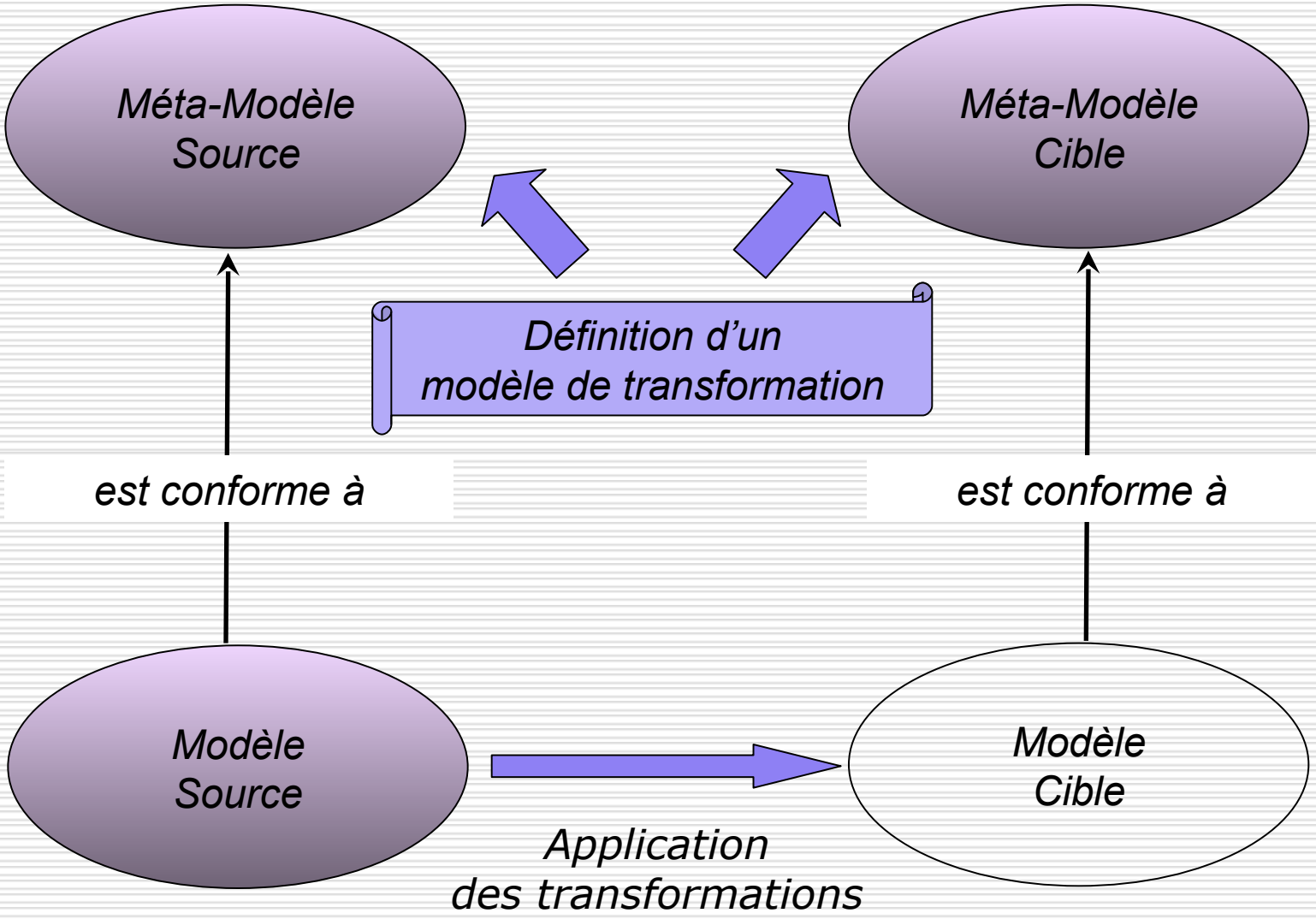
# Sommaire

- Taxonomie des transformations
- Modèles de transformation
- Techniques de transformation
- Langages et outils

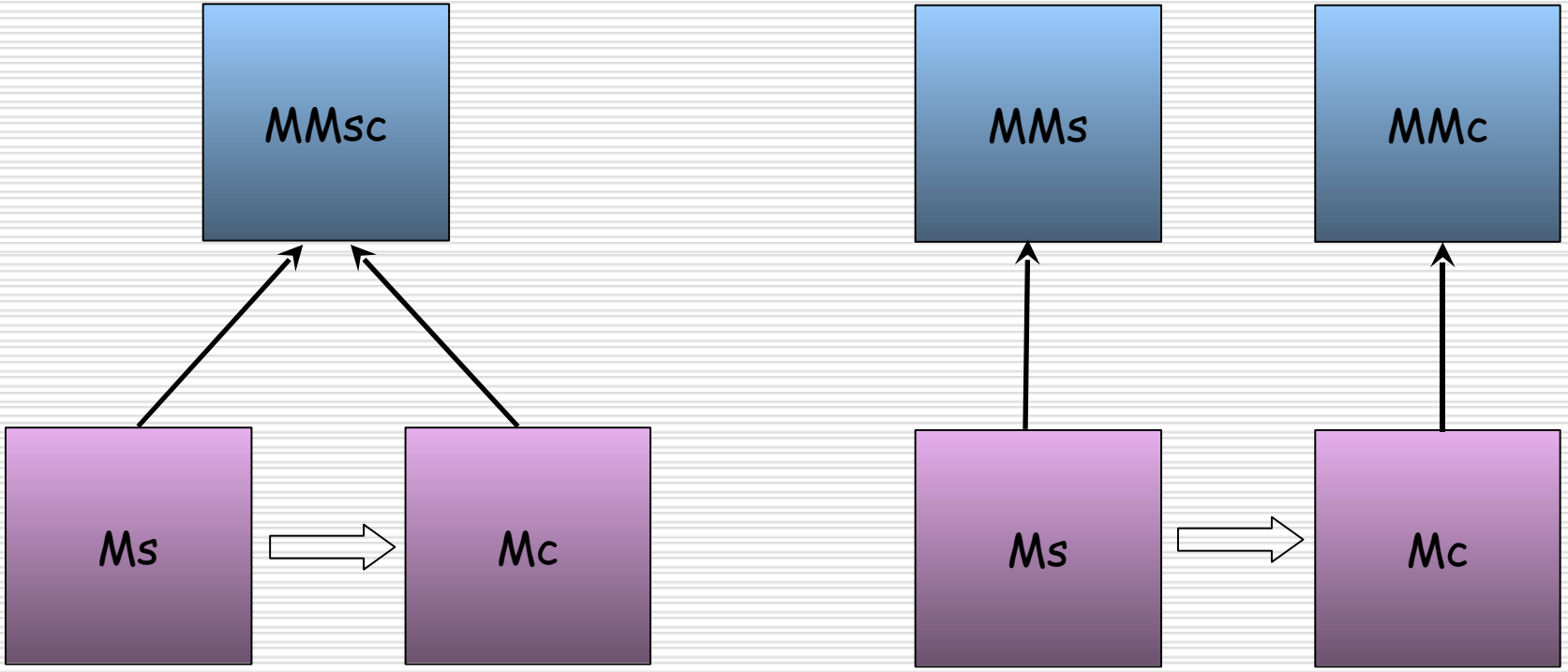
# Taxonomie des transformations

# Définitions

- Processus de transformation
  - Génération de modèles cibles à partir de modèles sources
- Modèle de transformation
  - Correspondance entre un modèle source et un modèle cible
- Règle de transformation
  - Description de la correspondance entre une (ou plusieurs) construction(s) du modèle source et une (ou plusieurs) construction(s) du modèle cible


Méta-Modèle
Source
Méta-Modèle
Cible
Définition d'un
modèle de transformation
est conforme à
est conforme à
Modèle
Source
Modèle
Cible
Application
des transformations

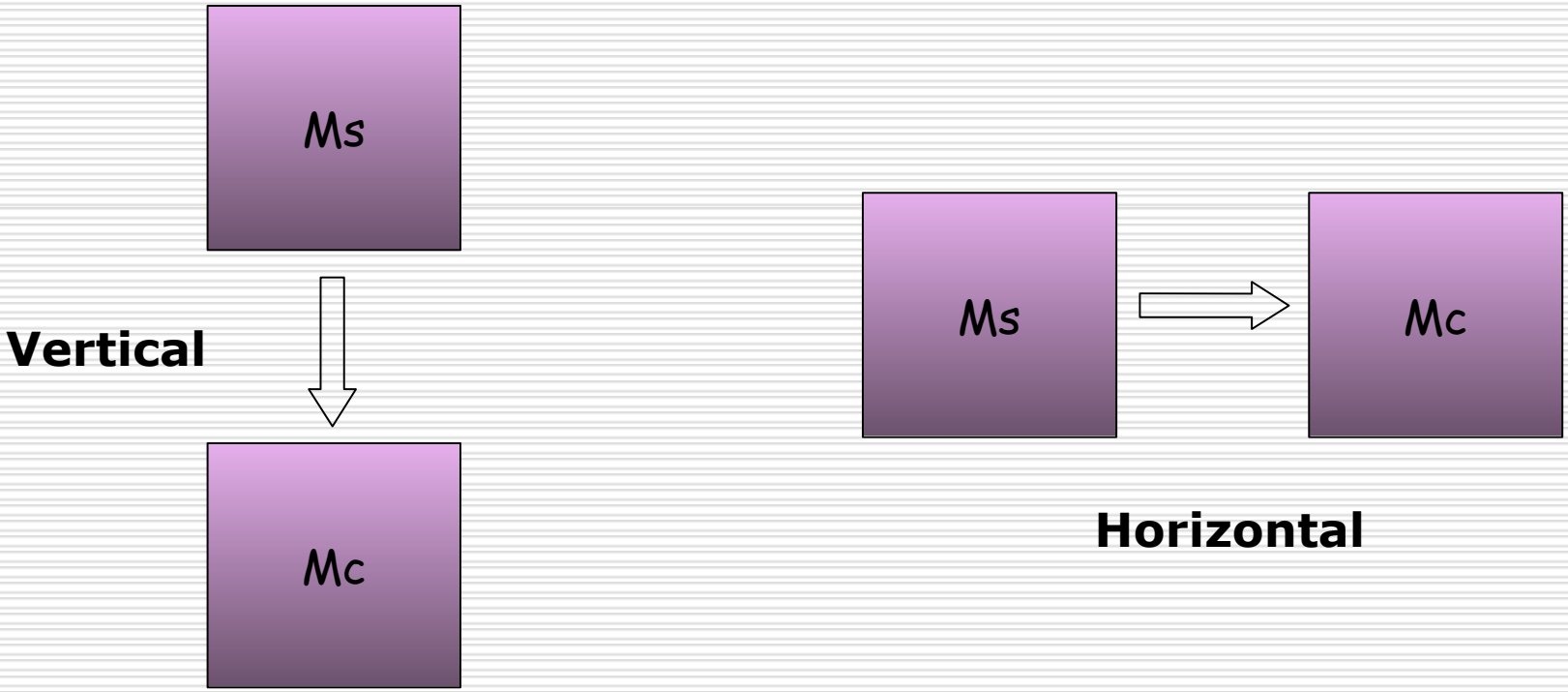
# Transformations endogènes vs exogènes




**Endogène**

**Exogène**

# Transformations verticales vs horizontales



# Classes de transformation typiques

| ***Transformation*** | **Horizontale** | **Verticale** |
|---|---|---|
| **Endogène** | Restructuration<br>Normalisation<br>Intégration de patrons | Raffinement |
| **Exogène** | Migration de logiciels<br>Fusion de modèles | PIM vers PSM<br>Rétro-conception<br>Génération de code |

# Propriétés et caractéristiques

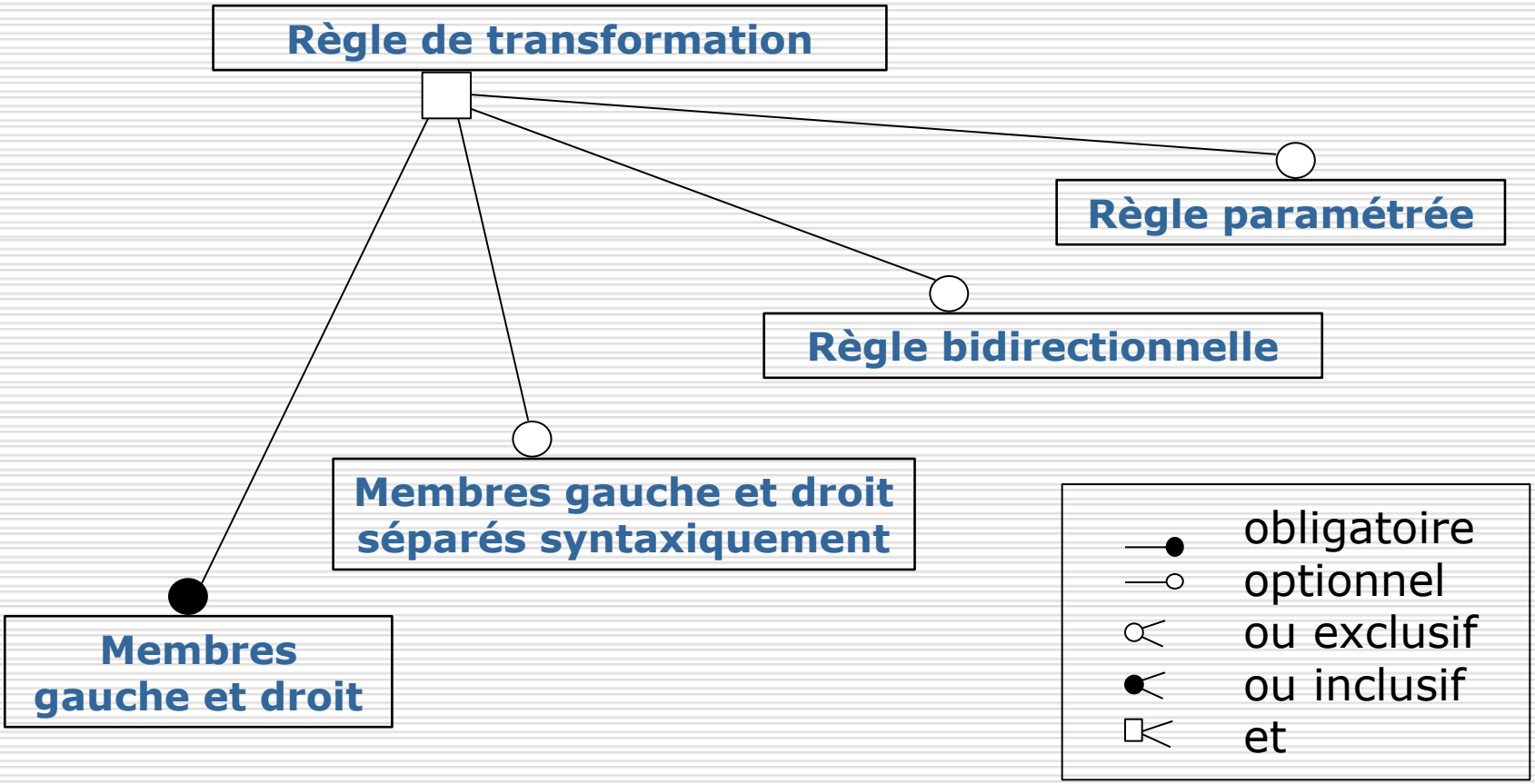
- Propriétés
  - Réversibilité
  - Traçabilité
  - Incrémentabilité
  - Réutilisabilité
  - …
- Caractéristiques
  - Automatique / manuel / assisté
  - Préservation des propriétés prouvées
  - Complexité des algorithmes
  - …

# Modèles de transformation

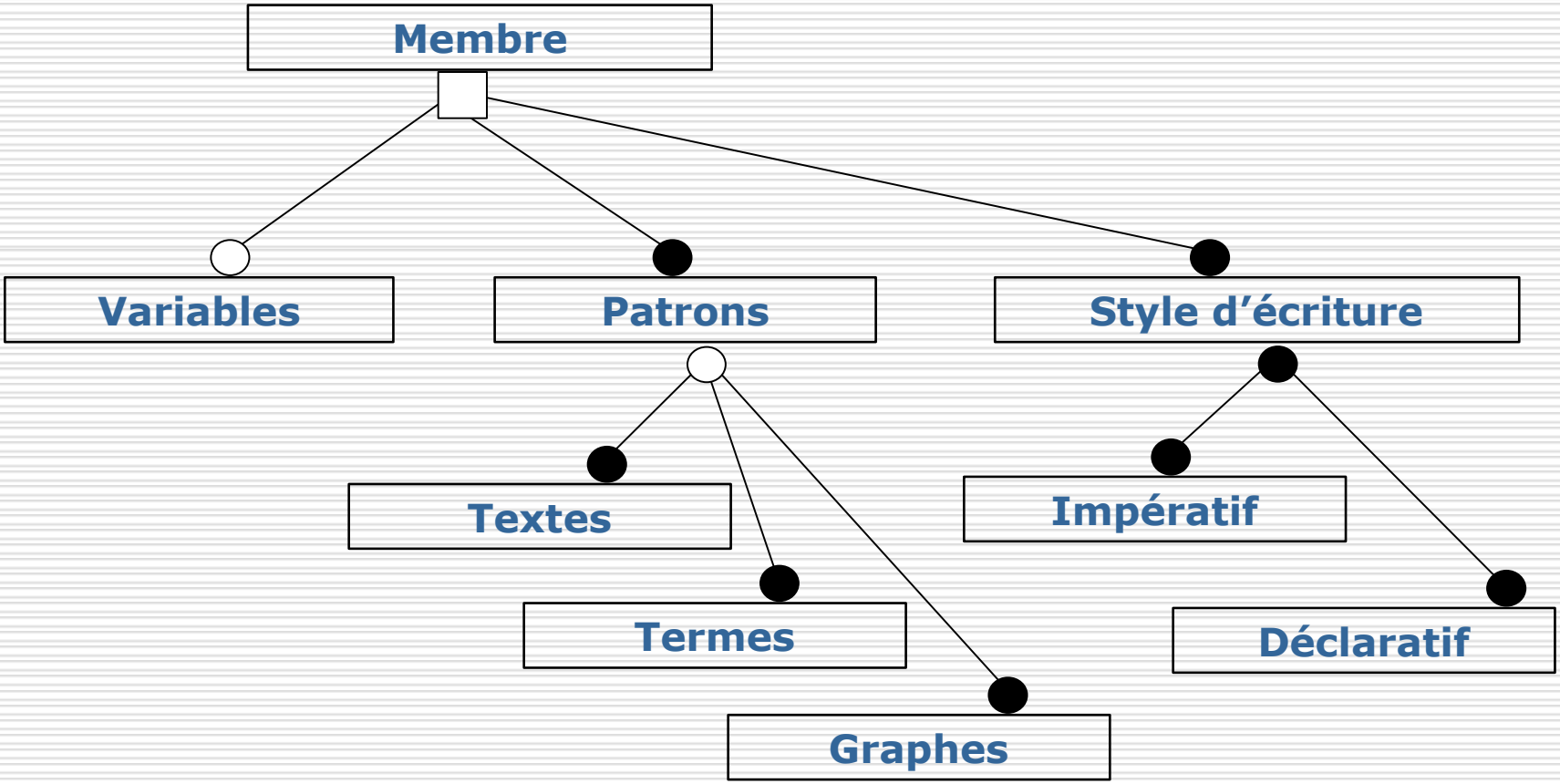
# Modèles de transformation

- Expression des règles de transformation
- Relation entre les modèles source et cible
- Stratégie d'application des règles
- Ordonnancement des règles
- Modularité et réutilisabilité des règles
- Liens de traçabilité
- Réversibilité des règles

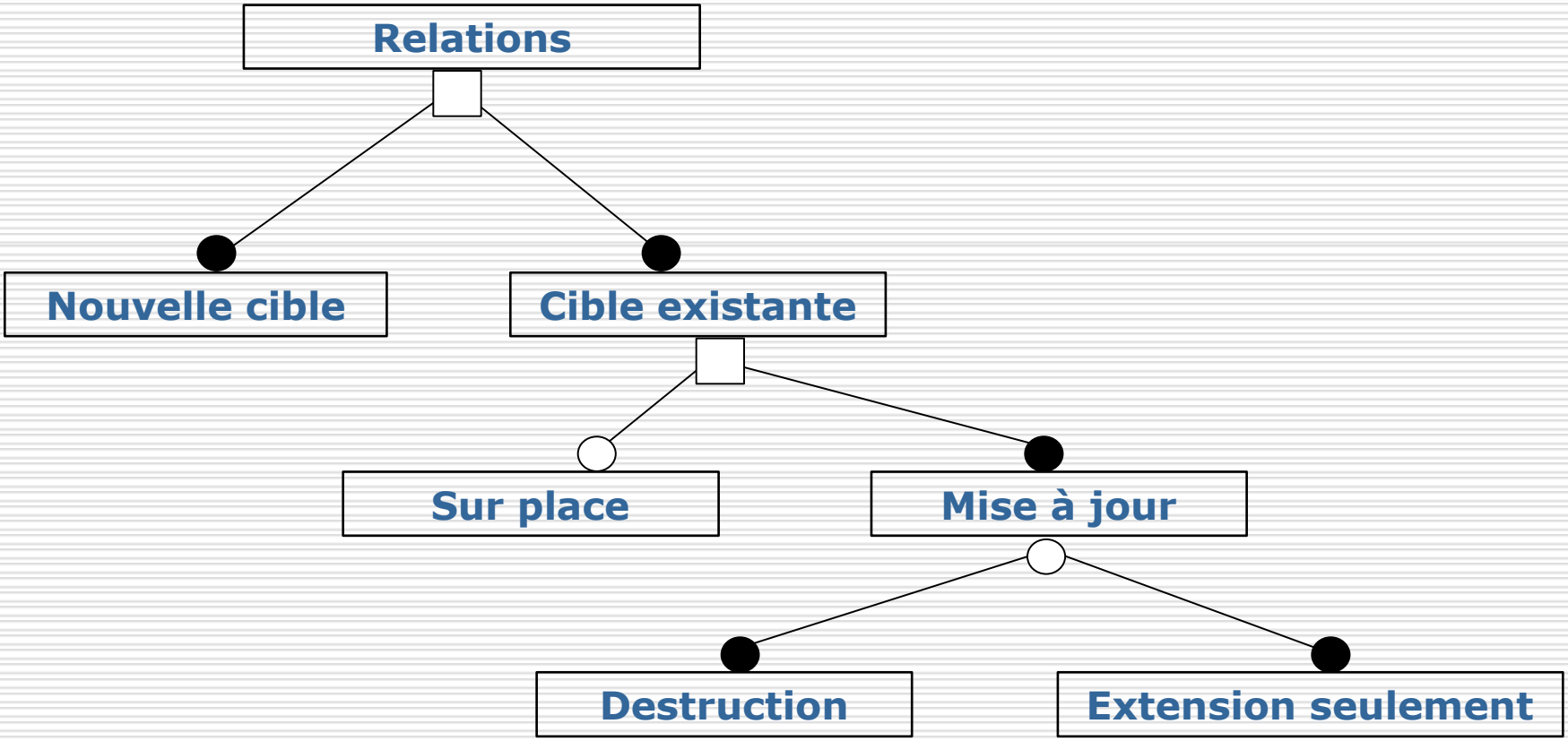
# Règles de transformation (1)



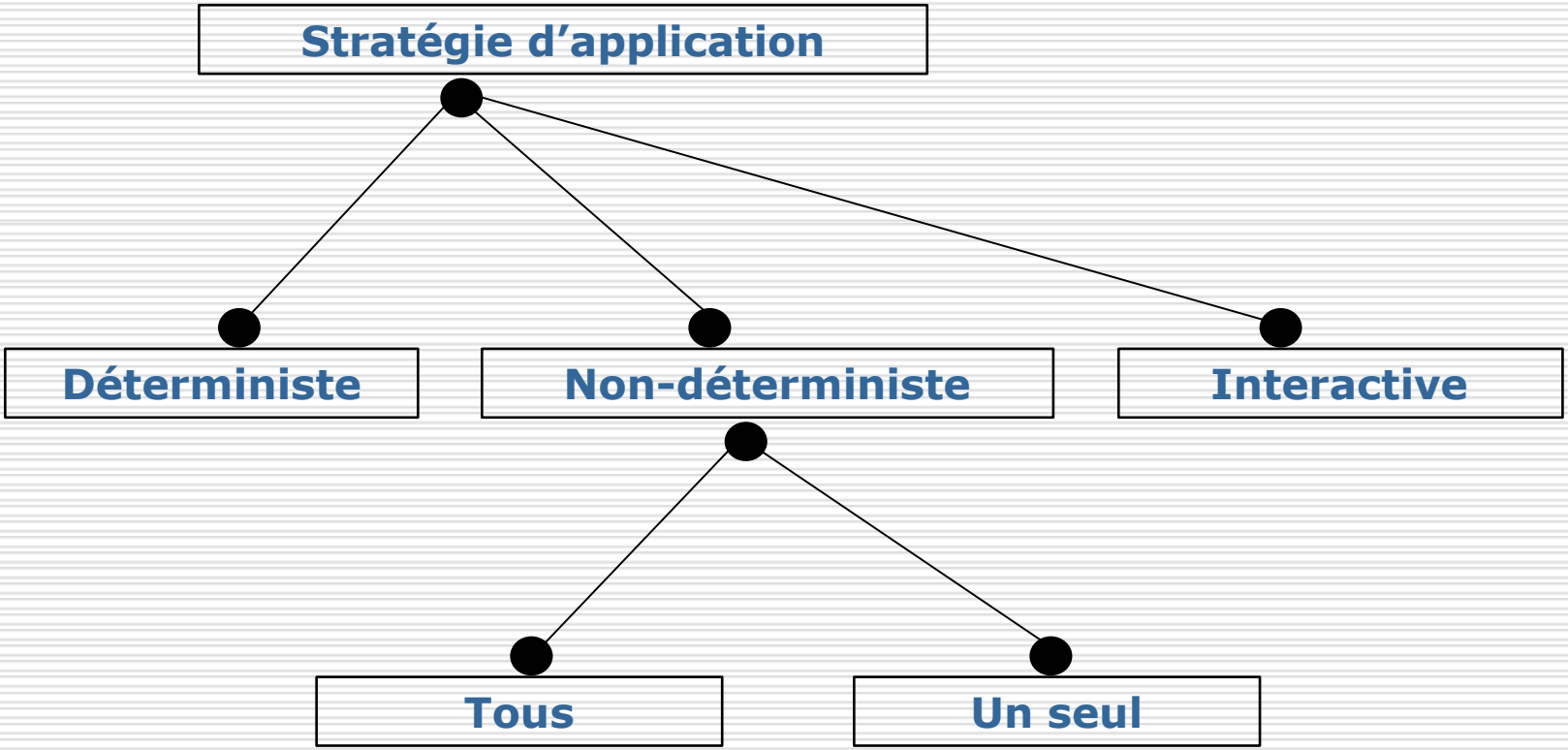
# Règles de transformation (2)



# Relations entre modèles source et cible



# Stratégie d'application des règles



# Techniques de transformation

# Techniques de transformation

- Manipulations directes (API)
- Approches relationnelles
- Approches guidées par la structure
- Transformation de graphes
- Approches hybrides
- …

# Manipulations directes

- Gestion d'un dépôt de données et de métadonnées compatible MOF
- API (Application Program Interface)
- Programmation explicite du processus de transformation
- Exemple : JMI (Java Metadata Interface)

# Exemple JMI

- La classe *Element*

```
public interface Element extends javax.jmi.reflect.RefObject {
        public String getName () ;
        public void setName (String s) ;
        public xmlmodel.Node getContainer () ;
        public void setContainer (xmlmodel.Node n) ; }
```

- L'association *Contains*

```
public interface Contains extends javax.jmi.reflect.RefAssociation {
    public boolean exits (xmlmodel.Node n) ;
    public List getElements (xmlmodel.Node n) ;
    public xmlmodel.Node getContainer (xmlmodel.Element e) ;
    public boolean add (xmlmodel.Element e, xmlmodel.Node n) ;
    public boolean remove (xmlmodel.Element e, xmlmodel.Node n) ; }
```

# Approches relationnelles

- Relations entre entités du modèle source et entités du modèle cible
- Implémentation en logique
- Règles de réécriture (unification et résolution)
- Exemples : Mercury et F-Logic

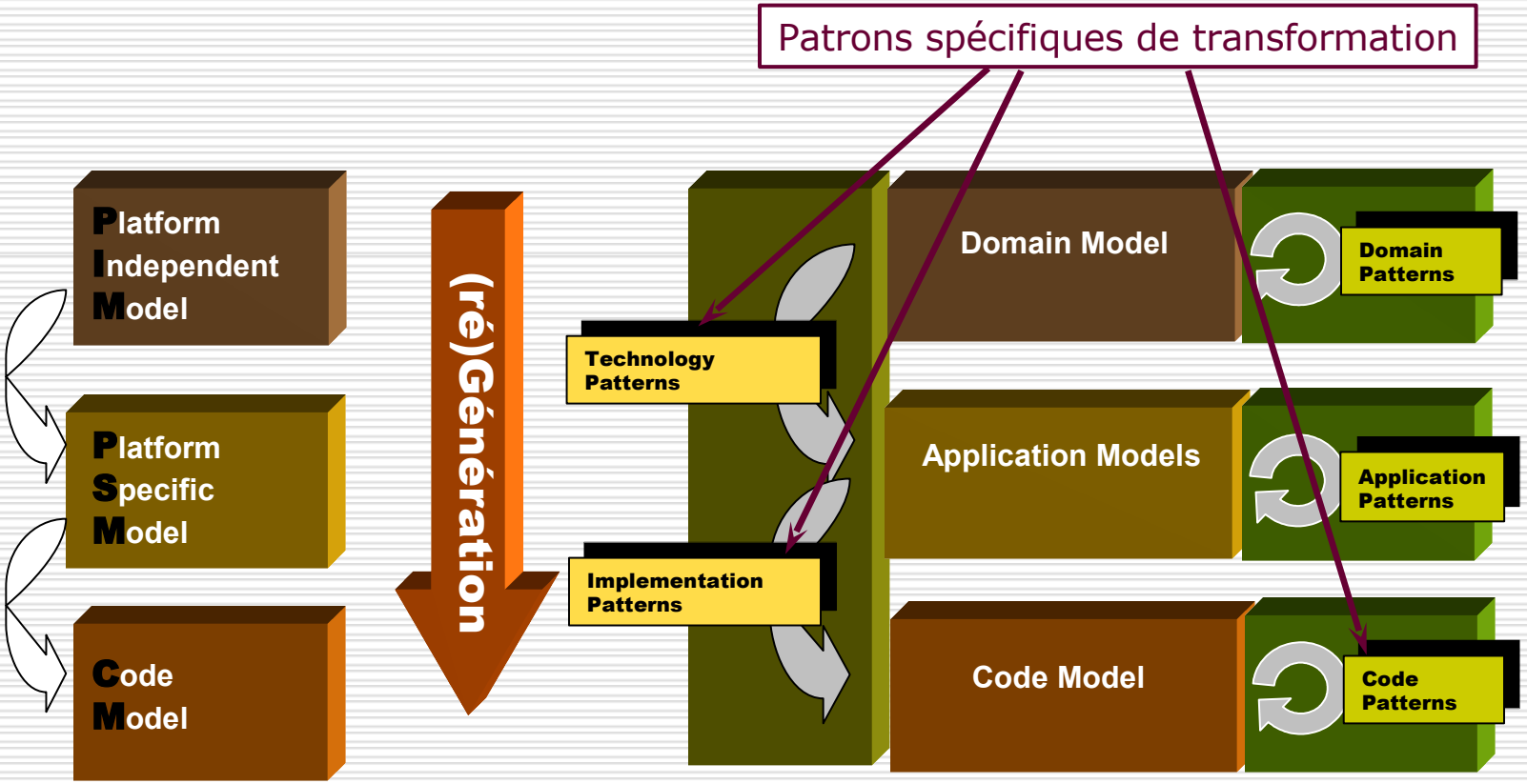
# Exemple Mercury

```
conditionaltask(Id) :-
        conditionaltask_for_outputgroup_of_activity(Id, _OutputGroup).

conditionaltask_for_outputgroup_of_activity(Id, OG) :-
        outputgroup_of_activity(OG, _Activity),
        mapId(OG^og_id, conditionaltask_for_outputgroup, Id).

outputgroup_of_activity(OutputGroup, Activity) :-
        outputgroup(OutputGroup),
        contains(Activity^a_id, OutputGroup^og_id),
        activity(Activity).
```

# Approches guidées par la structure

- Deux étapes :
  - Création d'une structure cible (PSM)
  - Mise en place des références de la cible
- Ajout de patrons de transformation au processus métier (PIM)
- Enchaînement des règles défini par l'environnement
- Exemple : OptimalJ

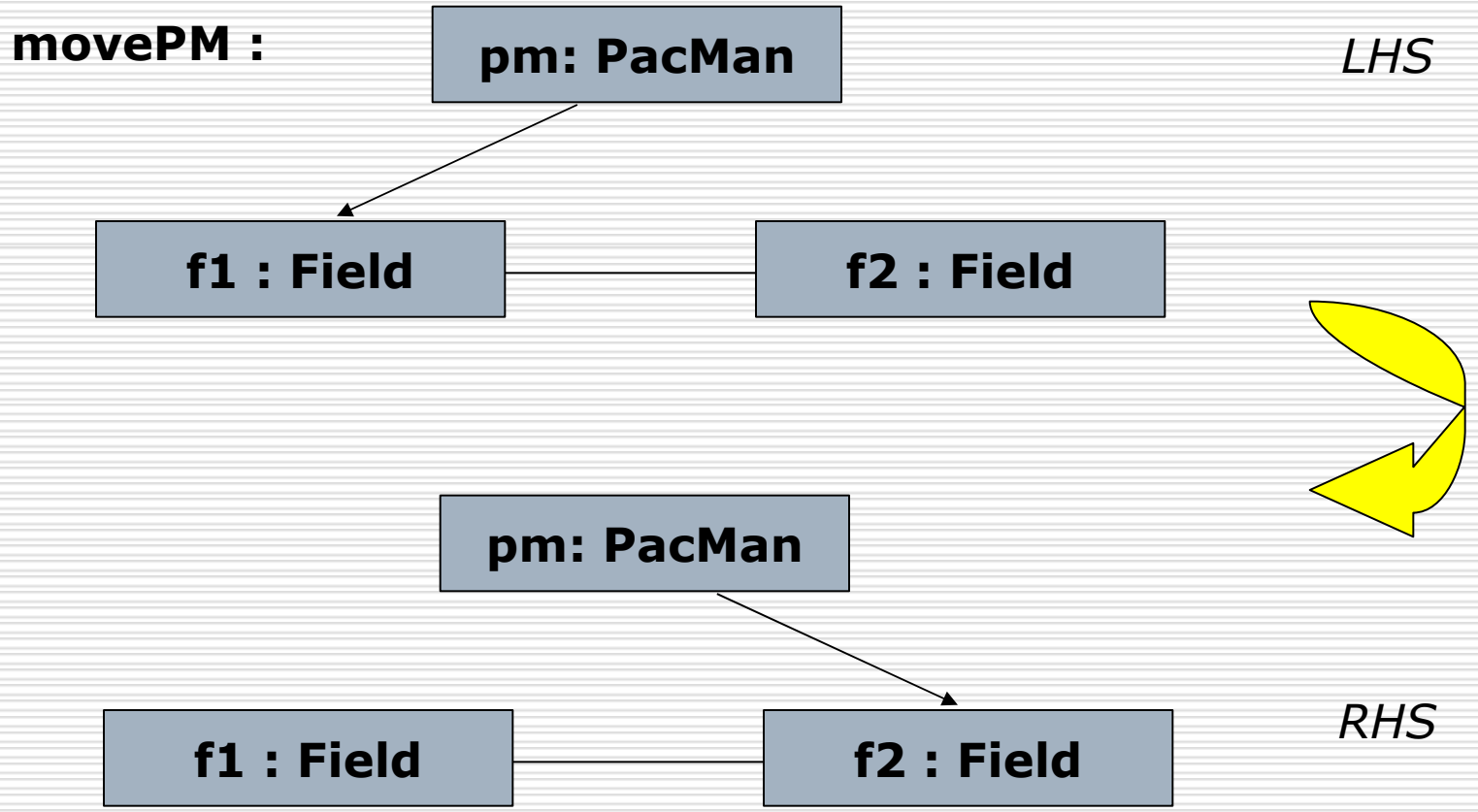
# OptimalJ


Patrons spécifiques de transformation
Platform Independent Model
Platform Specific Model
Code Model
(ré)Génération
Technology Patterns
Implementation Patterns
Domain Model
Application Models
Code Model
Domain Patterns
Application Patterns
Code Patterns

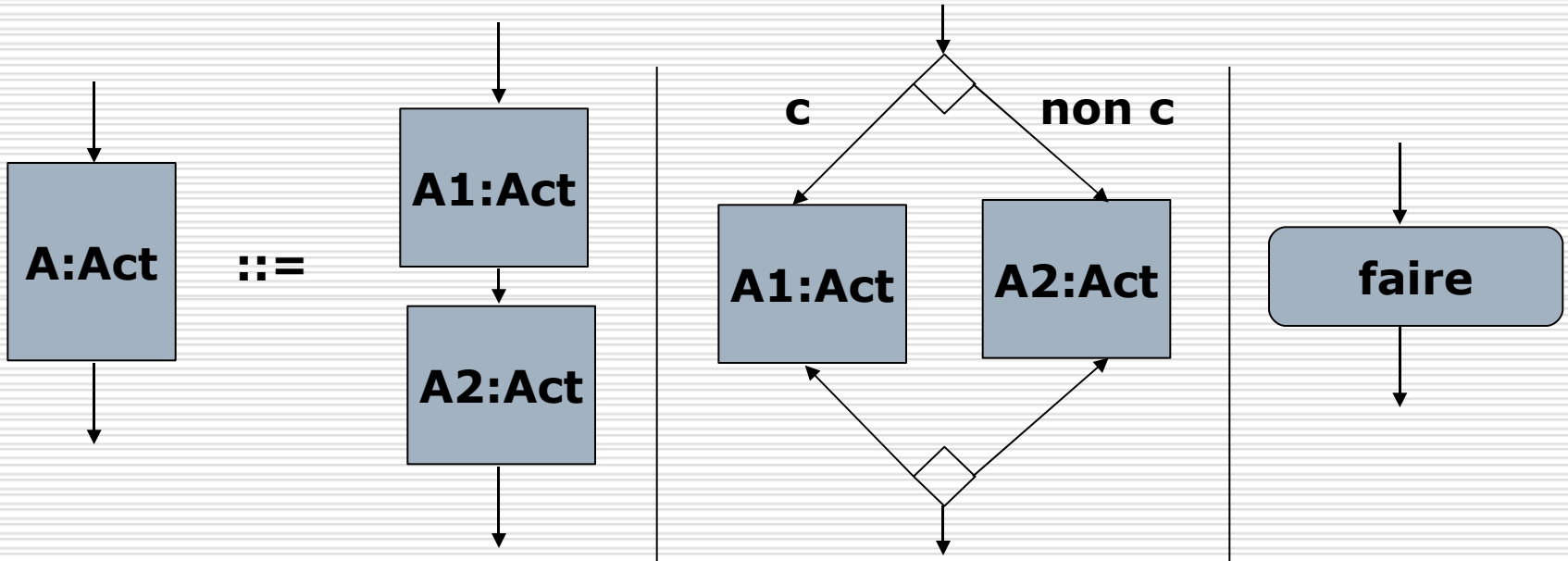
# Transformation de graphes

- *Patterns* de graphe (LHS et RHS)
- Non-déterminisme de l'application et de l'ordonnancement des règles
- Transformation *sur place*
  - Conservation des entités filtrées (LHS)
  - Ajout des nouvelles entités conformément à la règle de réécriture (RHS)
- Exemples : AGG, BOTL, MTrans et UMLX

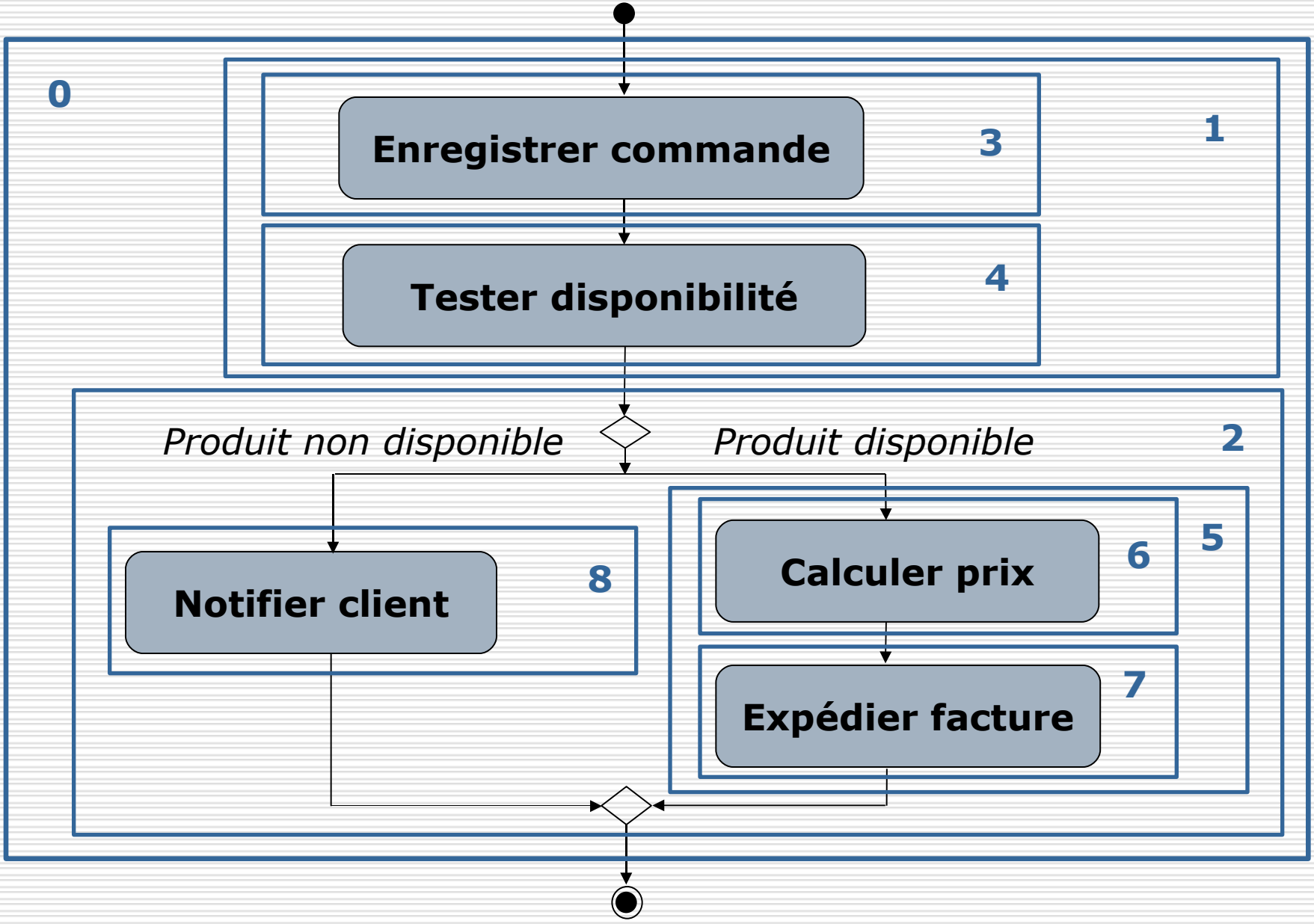
# Exemple de règle



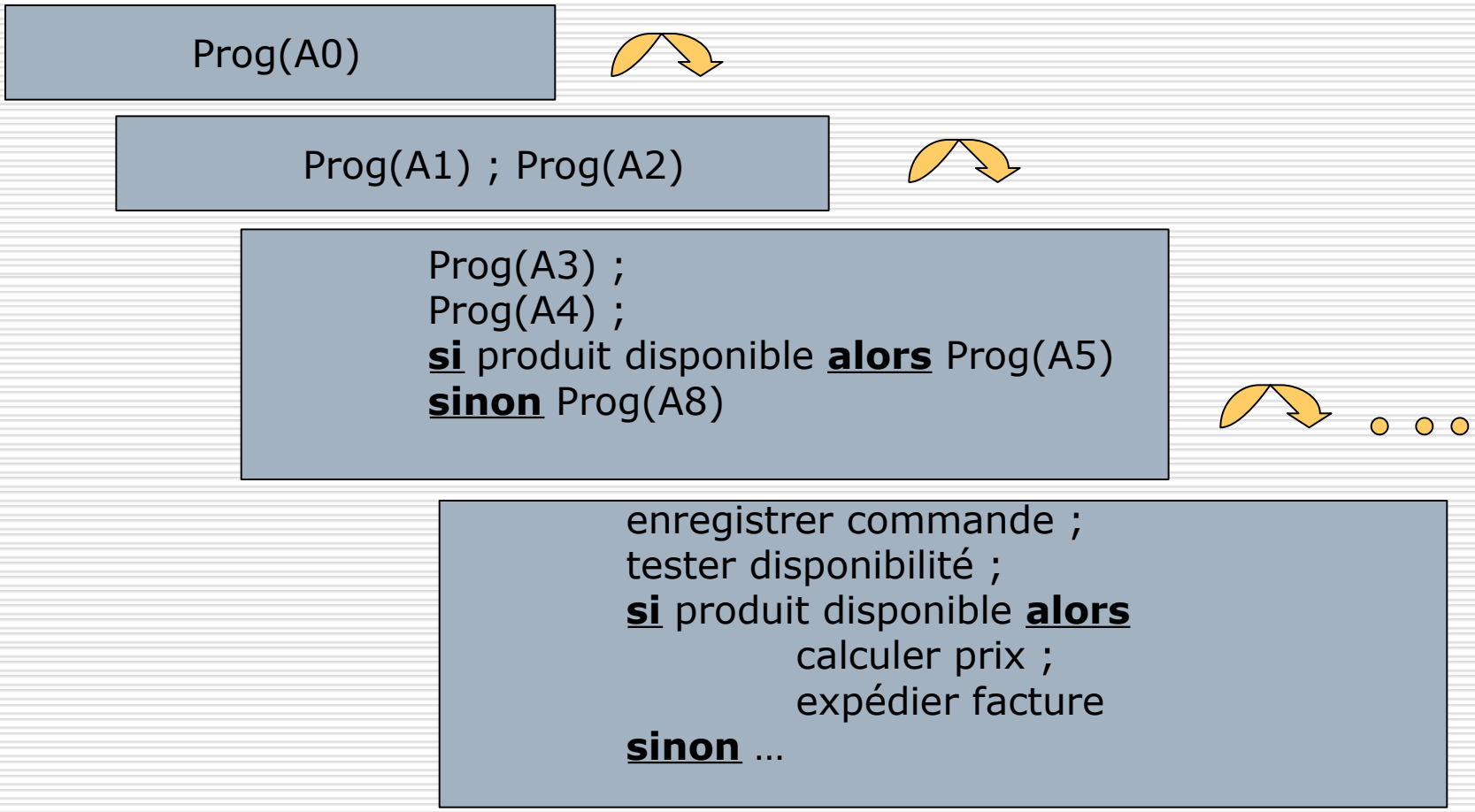
# Exemple : diagrammes d'activités vers code




| | | | |
|---|---|---|---|
| **Prog(A) ::=** | **Prog(A1) ;**<br>**Prog(A2)** | **si c alors Prog(A1)**<br>**sinon Prog(A2)** | **faire** |


0
1
Enregistrer commande
3
Tester disponibilité
4
2
Produit non disponible
Produit disponible
Notifier client
8
5
Calculer prix
6
Expédier facture
7

# Code généré


Prog(A0)
Prog(A1) ; Prog(A2)
Prog(A3) ;
Prog(A4) ;
si produit disponible alors Prog(A5)
sinon Prog(A8)
enregistrer commande ;
tester disponibilité ;
si produit disponible alors
calculer prix ;
expédier facture
sinon …

# Approches hybides

- Style d'écriture des règles
  - Déclaratif
  - Impératif
- Stratégie d'application des règles
  - Implicite
  - Explicite
- …
- Exemple : ATL

# Exemple ATL

- *Classifier* UML vers *classe* Java

```
rule UMLClassifierToJavaClass {
        from e : UML!Classifier
        to out : JAVA!JavaClass
                (name <- e.name)
}
```

- *Attribut* UML vers *attribut* Java

```
rule UMLAttributeToJavaAttribute {
        from e : UML!Attribute
        to out : JAVA!Attribute
                (name <- e.name, owner <- e.owner)
}
```

# Langages et outils

# Langages et outils

- Langages
  - XSLT, ATL, AGG, GreAT, MTL, Mercury, TGG, F-Logic, ATOM, IMPROVE, Fujaba, PROGRES, …
- Environnements de transformation
  - OptimalJ, MIA, BOTL, XDE, JMI, VIATRA, UMLX, …
- Environnements de méta-modélisation
  - Kermeta, K3M, …

# Comparatif XSLT, ATL, GA, …

| | **Réversibilité** | **Réutilisabilité** | **Traçabilité** | **Incrémentabilité** |
|---|---|---|---|---|
| **XSLT** | Unidirectionnel | Appel de motifs<br>Paramètres | Attributs supplémentaires | - |
| **MTrans** | Unidirectionnel | Héritage | Implicite et automatique | - |
| **ATL** | Unidirectionnel | Héritage | Implicite et automatique | - |
| **GA** | Unidirectionnel | Fonctions de transformation | Attributs supplémentaires | Evaluateurs incrémentaux |
| **BOTL** | Bidirectionnel | Héritage d'attributs | - | - |

# Comparatif AGG, Fujaba, VIATRA, … (1)

| | Number of source and target models | Kind of transformation | Technical space | Level of automation |
|---|---|---|---|---|
| **AGG** | One-to-one | Endogenous | XML, GXL, GTXL | Graph grammars |
| **Fujaba** | Many-to-many | Endogenous | MDA, UML, Java | Story-driven modelling |
| **VIATRA** | Many-to-many | Endogenous / exogenous | MDA, XSD, business process models | Transformations driven by abstract state machines |
| **GReAT** | Many-to-many | Endogenous / exogenous | MDA, UML | Explicit sequenced transformation steps with context parameters |

# Comparatif AGG, Fujaba, VIATRA, … (2)

| | Complexity | Customisability and reusability | Verification and validation | Composition |
|---|---|---|---|---|
| **AGG** | Layered grammars | Parameterised transformations | Termination checking, consistency checking, critical pair analysis | Layered graph grammars |
| **Fujaba** | Controlled graph transformations | Parameterised transformations / inheritance | Graph transformation based on JUnit tests | Method calls in story-driven modelling |
| **VIATRA** | Higher-order and meta-transformations | Reuse of predefined patterns | Full-fledged verification / validation by Check-VML | Non-recursive composition of patterns into rules |
| **GReAT** | Controlled graph transformations | Reuse of transformation rules and blocks | Well-formedness constraints on transformation results | Hierachical blocks of sequences, recursion |

# Comparatif AGG, Fujaba, VIATRA, … (3)

| | **Usability** | **Extensibilty** | **Acceptability** | **Standardisation** |
|---|---|---|---|---|
| **AGG** | Low (not very performant or scalable) | By extending AGG'internal engine | For research purposes | GXL, GTXL |
| **Fujaba** | High | Plug-in mechanism | For software development | UML, Java, XMI, MOF |
| **VIATRA** | High | Powerful plug-in mechanism for writing importers and exporters | For model transformation | UML, XMI, MOF |
| **GReAT** | High | Procedural code | For model transformation | UML, XMI |

# Anatomie d'ATL (1)

| ***Caractéristique*** | ***Spécificité ATL*** |
|---|---|
| **Règles de transformation** | Mélange impératif (*called rules*) et déclaratif (*matched rules*) |
| | Membres gauche et droit séparés syntaxiquement |
| | Variables et patrons |
| | Paramètres additionnels |
| | De la source vers la cible uniquement |
| **Relation entre modèles source et cible** | Modèles source et cible séparés |
| | Pas de mise à jour *sur place* |

# Anatomie d'ATL (2)

| ***Caractéristique*** | ***Spécificité ATL*** |
|---|---|
| **Stratégie d'application des règles** | Déterminisme des règles impératives |
| | Non-déterminisme du filtrage d'un patron du modèle source |
| | Pas d'interactivité pour la sélection des règles |
| **Liens de traçabilité** | Support automatique |
| | Mémorisation par le moteur |

# Anatomie d'ATL (3)

| ***Caractéristique*** | ***Spécificité ATL*** |
|---|---|
| **Ordonnancement des règles** | Mélange implicite et explicite |
| | Implicite vis-à-vis du filtrage d'un patron |
| | Explicite vis-à-vis des règles invoquées |
| | Gardes pour la sélection des règles (pas de conflit autorisé) |
| | Récursivité |
| | Séquence de transformations |

# Anatomie d'ATL (4)

| ***Caractéristique*** | ***Spécificité ATL*** |
|---|---|
| **Organisation des règles** | Règles indépendantes des modèles source et cible |
| | Règles dirigées par le modèle source |
| | Structures intermédiaires (*helpers*) |
| | Modules de transformation |
| | Inclusion de modules |
| | Héritage de règles |

# Bibliographie

- J. Bézivin, O. Gerbé - Towards a precise definition of the OMG/MDA framework, ASE'01, Automated Software Engineering, San Diego, USA, November 26-29, 2001.
- E. Biermann, K. Ehrig, C. Köhler, G. Kuhns, G. Taentzer, E. Weiss - Graphical Definition of In-Place Transformations in the Eclipse Modeling Framework. In Proc. 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS'06), Genova, Italy, October 2006.
- K. Czarnecki, S. Helsen - Classification of Model Transformation Approaches, OOPSLA'2003, Workshop on Generative Techniques in the Context of Model Driven Architecture.
- K. Ehrig, E. Guerra, J. Lara (de), L. Lengyel, T. Levendovszky, U. Prange, G. Taentzer, D. Varró, S. Varró-Gyapay - Model Transformation by Graph Transformation: A Comparative Study, MTiP 2005, International Workshop on Model Transformations in Practice (Satellite Event of MoDELS 2005).
- J.-M. Favre, T. Nguyen - Towards a a Megamodel to Model Software Evolution Through Transformations, Workshop on Software Evolution through Transformation, Rome, Italy, October 2, 2004.
- A. Gerber, M. Lawley, K. Raymond, J. Steel, A. Wood - Transformation: The Missing Link of MDA, LNCS vol. 2505, Springer-Verlag, 2002, pp. 90 – 105.
- T. Mens, P. Van Gorp - A Taxonomy of Model Transformation, GraMoT'2005.
- OMG (Object Management Group) http://omg.org

Questions

# A Taxonomy of Model Transformations
# T. Mens, K. Czarnecki et P. Van Gorp
# Dagsthul Seminar Proceedings 04101

- **What needs to be transformed into what?**
  - Program and model transformation
  - Endogenous versus exogenous transformations
  - Horizontal versus vertical transformations
  - Technological space

- **What are the important characteristics of a model transformation?**
  - Level of automation
  - Complexity of the transformation
  - Preservation

## What are the success criteria for a transformation language or tool? (1)

- Ability to create/read/update/delete transformations
- Ability to suggest when to apply transformations
- Ability to customise or reuse transformations
- Ability to guarantee correctness of the transformations
- Ability to deal with incomplete or inconsistent models

## What are the success criteria for a transformation language or tool? (2)

- Ability to group, compose and decompose transformations
- Ability to test, validate and verify transformations
- Ability to specify generic ans higher-order transformations
- Ability to specify bidirectional transformations
- Support for traceability and change propagation

- **What are the quality requirements for a transformation language or tool?**
  - Usability and usefulness
  - Verbosity versus conciseness
  - Scalability
  - Mathematical properties
  - Acceptability by user community
  - Standardization

- **Which mechanisms can be used for model transformation?**
  - Functional programming
  - Logic programming
  - Graph transformation