The fun and easy way® to build buzz for your company or product

Model Driven Engineering

FOR ~~DUMMIES~~® **EES***

Sinon, si vous vouliez faire une sieste, c'est le moment !

Comme les spécialistes du génie logiciel, apprenez à placer le mot méta dans toutes vos phrases !

Steven Derrien
IRISA/CAIRN

* **Electrical Engineers**

Le génie Logiciel vu par les électroniciens

- L’histoire du roi et du grille pain …

Génie Logiciel et communauté CAO/HLS

- Positionnement de la communauté HLS
  - « Arrêtez de concevoir vos circuits en VHDL, et élevez le niveau d'abstraction de vos méthodologies de conception ».
  - « Vous allez devoir remettre en cause votre façon de travailler, mais à moyen terme vous serez gagnants ».
  - « Les pertes en qualité (surface, performance) seront largement compensées par vos gains en productivité ».

- Une communauté schizophrène ?
  - Faites ce que je dis, pas ce que je fais …

Pourquoi ne commencerait-on pas par appliquer ce principe à nous même et à nos outils ?

Objectif de la présentation

- Expliquer les principes et concepts (mais pas trop) sous-jacents à l'ingénierie des modèles.
- Présenter un panel des technologies qui existent aujourd'hui autour *d'Eclipse Modeling Framework*
- Expliquer pourquoi ces technos sont potentiellement très intéressantes pour une équipe comme CAIRN.
- Essayer de vous convaincre que l'investissement dans ces technologies en vaut la peine

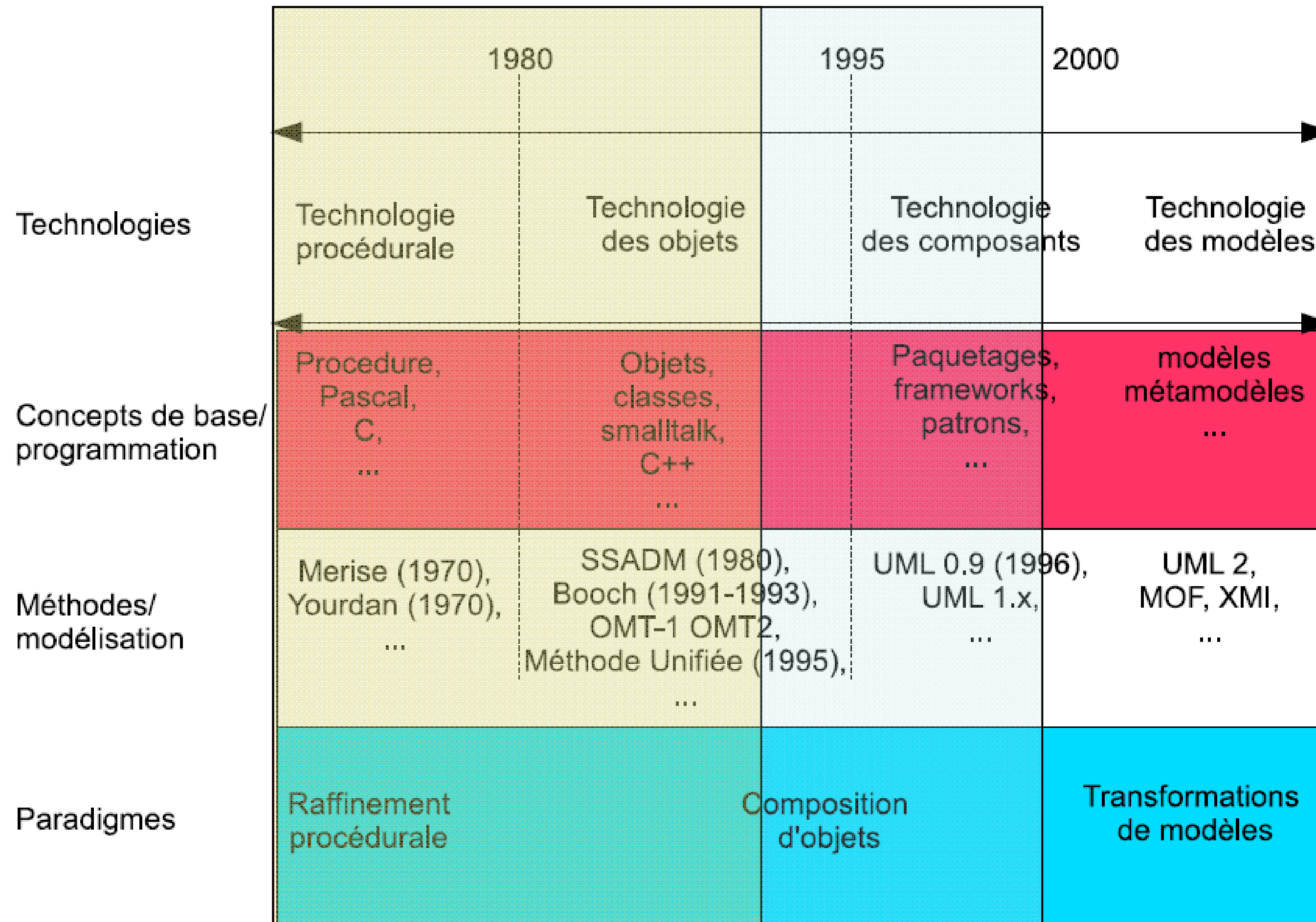
- Si j'ai du temps, montrer ce qu'on a pu faire avec …

Plan de la présentation

- Contexte/historique
  - Evolution des approches de génie logiciel
  - Notion de modèle, méta-modèle, etc.
- Le framework EMF d'Eclipse
  - Un exemple fil rouge : graphes flot de signal
  - Manipulation de modèles EMF en Java
- Les outils autour d'EMF
  - Génération de code (JET)
  - Conception d'éditeurs graphiques (GMF)
  - Conception de *Domain Specific Languages* (XTEXT)
- Quelques exemples

Evolution des techniques de programmation

Spectre de compétences de l'informaticien « moyen »



Spectre de compétences de l'électronicien « moyen »

[KABORE08]

Modèles contemplatifs / productifs

- Modèles contemplatifs
  - Utilisés pour communiquer, voire pour spécifier (UML 1).
  - Ces modèles sont interprétés (traduits) à la main par les développeurs en implémentations logicielles.
- Modèles productifs
  - La machine (et ses outils) exploite la spécification du modèle, afin d'être capable de le transformer, de l'interpréter, etc.
  - Le modèle devient alors l'épine dorsale du logiciel, tous les développements découlent/se basent sur lui.
  - On passe ainsi du « tout est objet » au « tout est modèle »

Pour pouvoir faire cela, il faut évidement des modèles auxquels on peut donner du *sens*, et qui respectent une certaine trame (i.e. un modèle de modèle)

Définitions

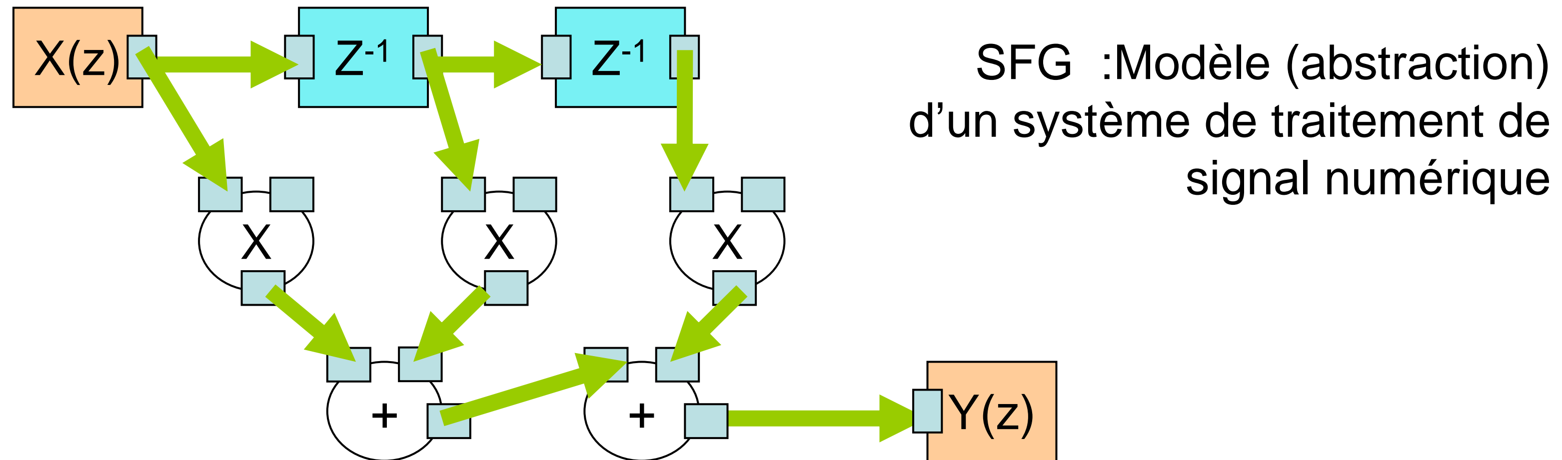
- Modèle (niveau M1) :
  - Un modèle est une abstraction d'un système étudié, construite dans une intention particulière [Bézivin01].
    - Exemple : une représentation intermédiaire utilisé dans un outil de CAO/compilation (ex : graphe flot de signal, archi RTL)
- Métamodèle* (niveau M2)
  - Un métamodèle est un langage qui permet d'exprimer des modèles. Il définit les concepts ainsi que les relations entre concepts nécessaires à l'expression de modèles [Bézivin01]
- Métamétamodèle* (niveau M3)
  - Un métamétamodèle est un langage qui permet d'exprimer des métamodèles. C'est un langage unificateur pour les outils qui manipulent les métamodèles.

* « Got It ? If not, don't worry about it as it is just an academic issue anyway »

Ed Mercks, EMF book

Exemple fil rouge

- Graphe flot de signal (Signal flow Graph)



- Vers un méta modèle pour SFG …
  - Un SFG est formé de nœuds et d'arcs
    - Différents types de nœuds (opérations, délais (Z^{-1}), E/S)
    - Un nœud contient des ports d'entrées et de sorties
  - Un arc relie un port de sortie à un port d'entrée.
    - Il ne peut y avoir plusieurs arcs sur un port d'entrée
    - La fonction de transfert doit être causale

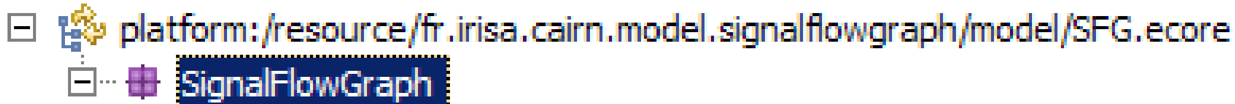
L'infrastructure EMF/Eclipse

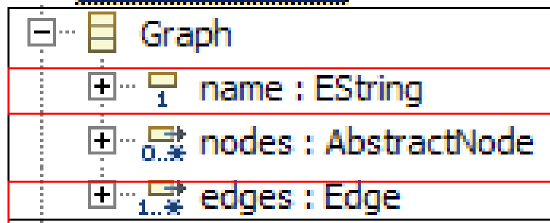
Le framework Eclipse/EMF

- Infrastructure Eclipse dédiée à l'Ingénierie des Modèles
  - Infrastructure stable et relativement mature (V2.4)
  - C'est le standard utilisé par la communauté académique et industrielle (IBM) pour l'Ingénierie dirigée par les Modèles.
  - Basée sur Java+Eclipse (forget C++, no Netbeans …)
- Basée sur le langage de méta-modélisation **`ecore`**
  - Ecore peut-être vu comme un méta-méta-modèle, c.a.d un langage qui permet d'exprimer des méta modèles (arghh …)
  - On peut exprimer un méta modèle de plusieurs façons
    - Textuelle à l'aide d'un langage dédié (EMFatic)
    - Avec un éditeur graphique à la UML
    - ***À l'aide d'un éditeur arborescent simple***

Exemple de métamodèle `ecore`

- Méta-modèle de Graphe Flot de Signal (SFG)

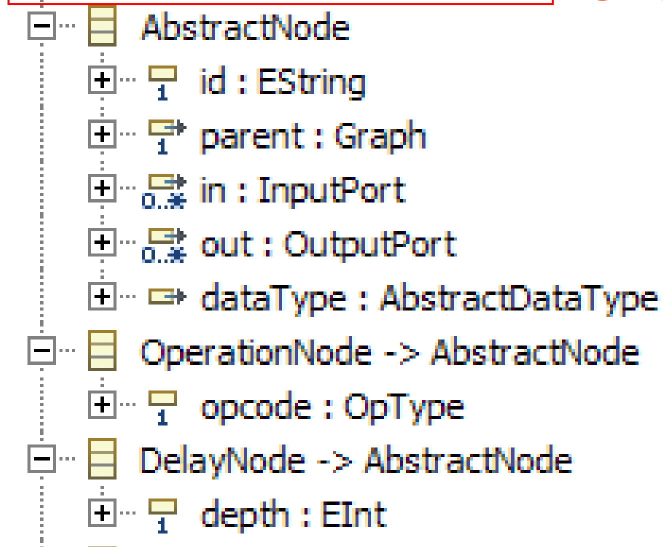




Une classe de mon méta-modèle

Un champ de type attribut

Un champ de type référence

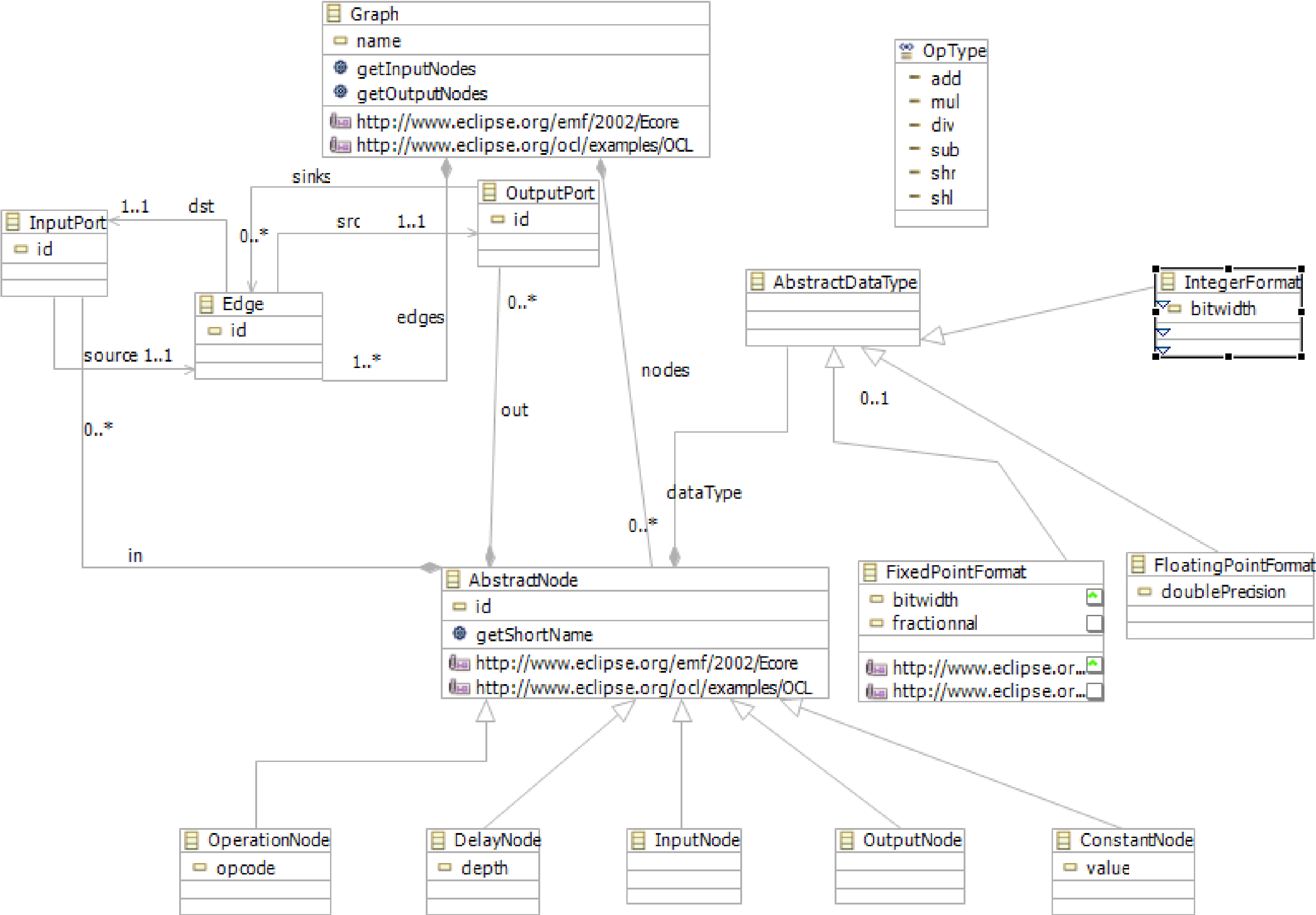


Property	Va
Changeable	
Container	
Containment	
Default Value Literal	
Derived	
EKeys	
EOpposite	parent : Graph
EType	Edge
Lower Bound	1
Name	edges
Ordered	true
Resolve Proxies	true

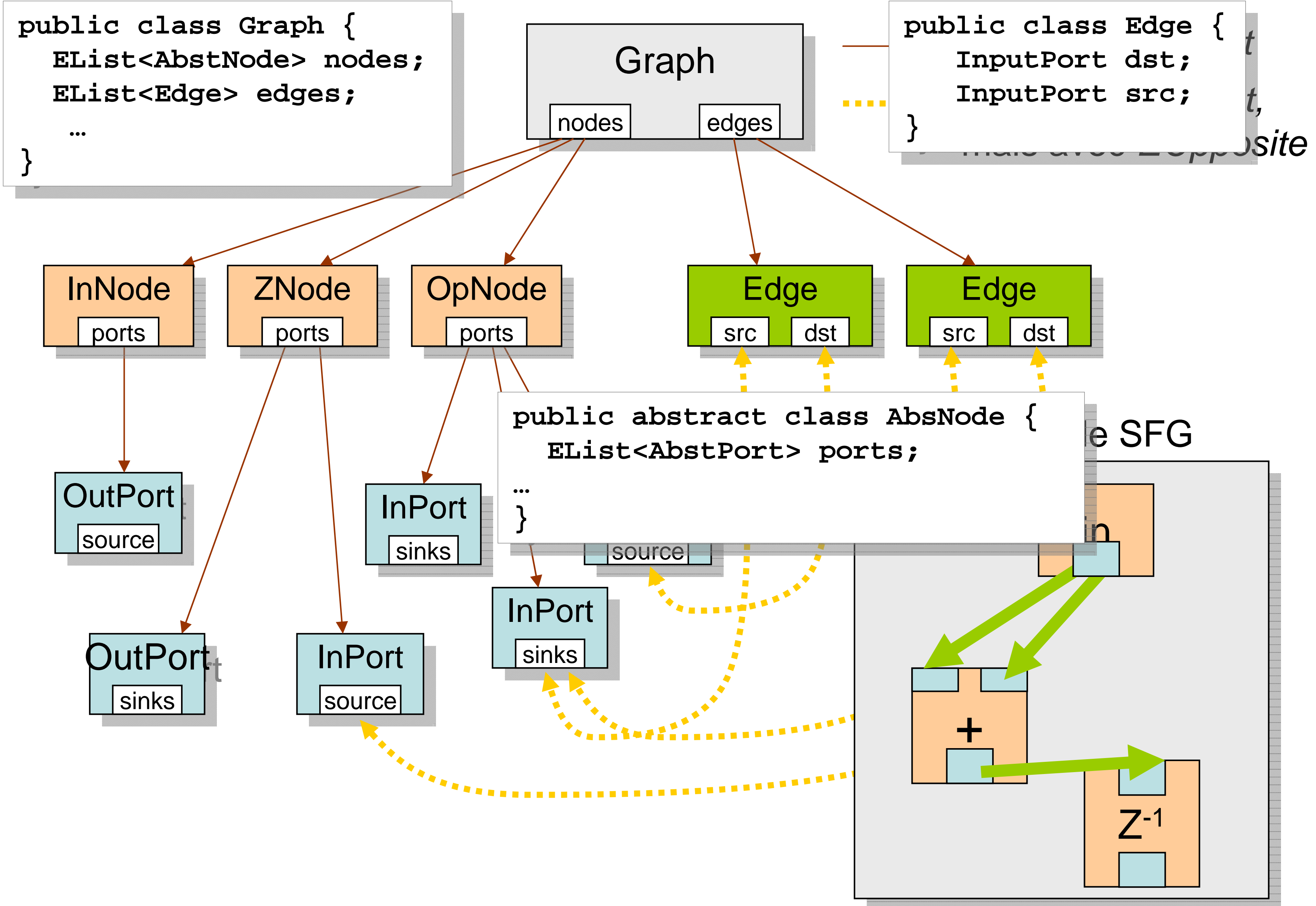
Property	Value
Changeable	true
Default Value Literal	
Derived	false
EAttribute Type	EString [java.lang.String]
EType	EString [java.lang.String]
ID	true
Lower Bound	1
Name	name
Ordered	true
Transient	false
Unique	true
Unsettable	false

Exemple de métamodèle `ecore`

- On peut aussi avoir une vision « à la UML »

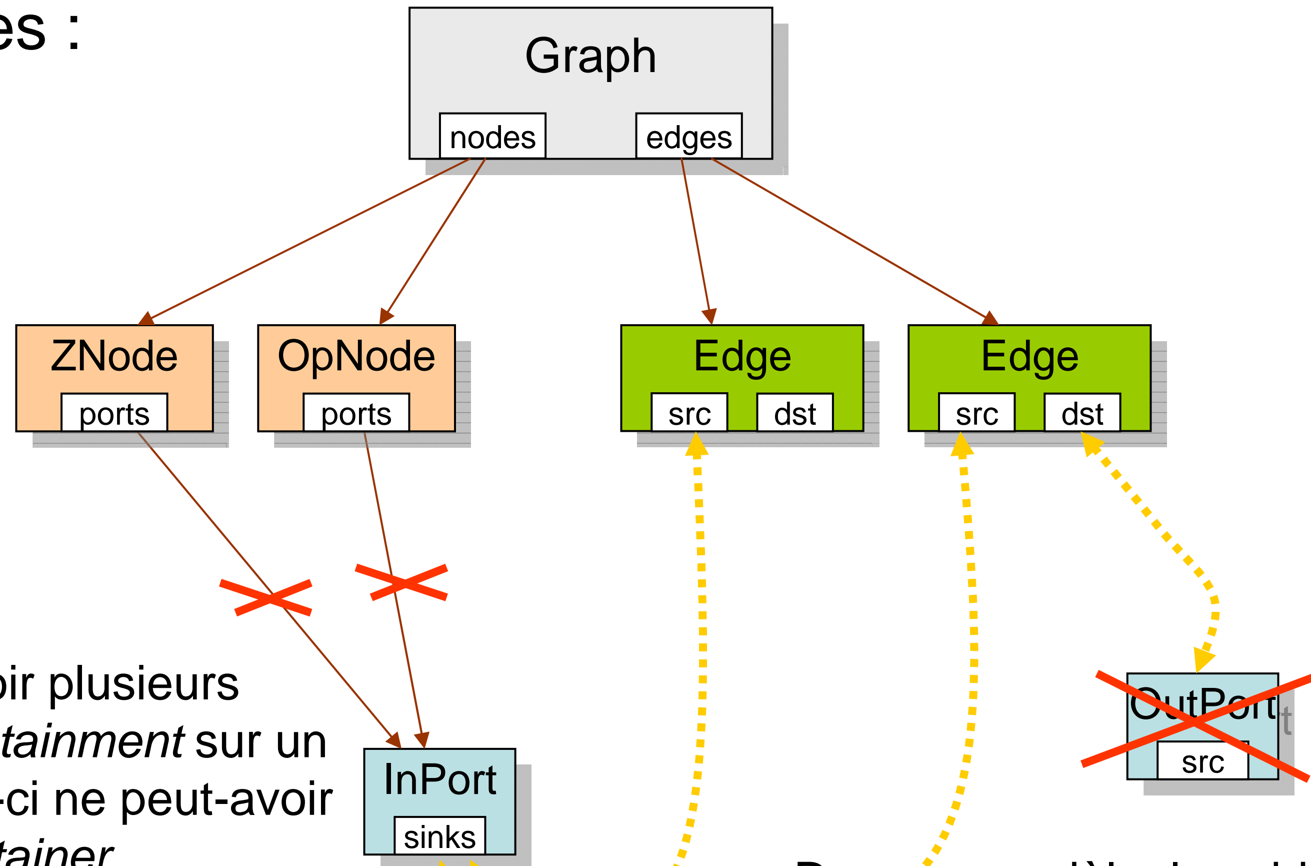


Notion de « containment »



Notion de « containment »

- Contraintes :



On ne peut avoir plusieurs références *containment* sur un objet, car celui-ci ne peut-avoir qu'un seul *container*.

Dans un modèle, les objets (sauf l'objet *toplevel*) doivent tous être contenus par un autre objet du modèle.

Que faire avec un méta-modèle sous EMF ?

- On peut en générer une représentation Java
  - Permet de construire/analyser/transformer des modèles en Java
  - Fournit des outils de navigation/recherche/construction
- On peut importer/exporter le modèle vers/de XML
- On peut définir des « validateurs » sur ce modèle
  - Pour garantir qu'une instance de modèle vérifie bien certaines propriétés (ex: pas de cycles sans délais Z^{-1} dans mon SFG)
- On peut générer un éditeur arborescent pour ce modèle
  - Pour construire « à la main » des instances du modèle
  - Pour visualiser des instances produites par ailleurs

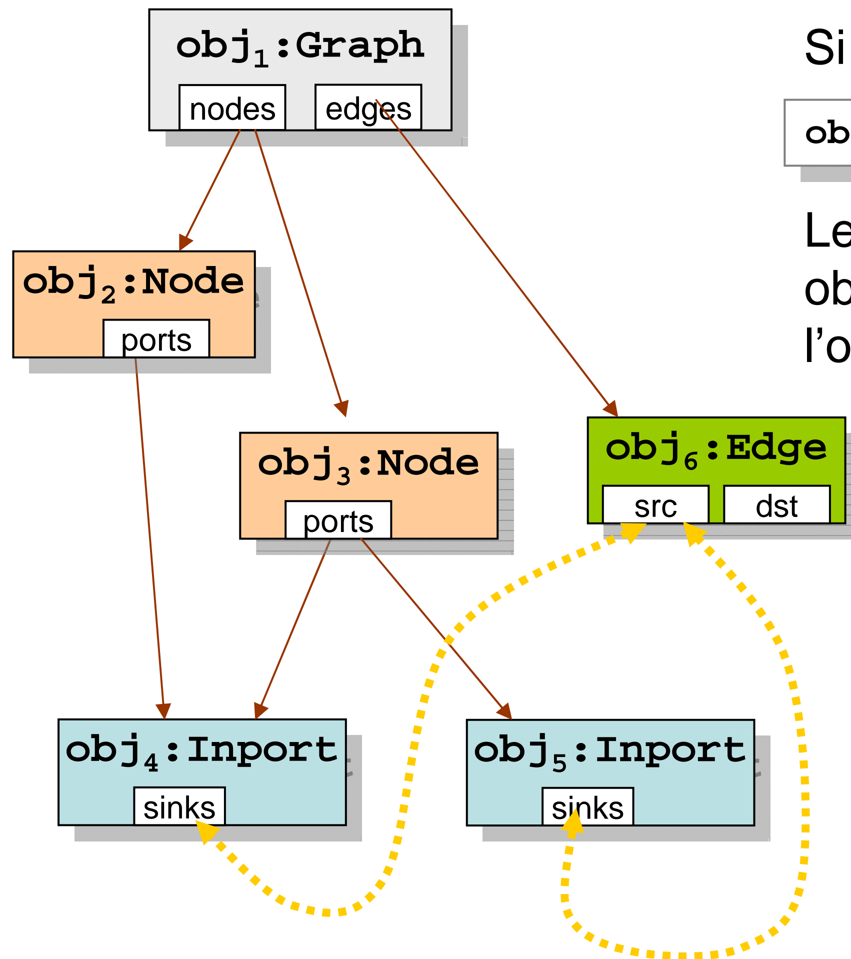
Génération de code Java

- Eclipse/EMF génère le code Java des classes/interfaces permettant de manipuler le modèle.
- Ex : pour la classe Edge, EMF/Eclipse génère :
  - une interface Java **`Edge`**
  - une classe d'implémentation **`EdgeImpl`**
- Eclipse/EMF génère également des classes « outils »
  - une *Factory**, un *Visiteur**, et un *AdapterFactory**
  - On peut « surveiller » le modèle à l'aide de *Notifiers* *
- Toutes les classes héritent/implémentent EObject
  - EObject est la classe de base de EMF

* Ces nom barbares correspondent à des *Design Patterns* (patrons de conception)

Maintient de cohérence

- Ex : le modèle Java généré par EMF garantit le respect des relations de *containment* et des liens bidirectionnels.



Si on modifie le modèle comme suit :

```
obj3.getPorts().add(obj4);
```

Le code Java généré s'assure que l'objet obj2 ne « contienne » plus obj3 après l'opération.

Si on modifie le modèle comme suit :

```
obj6.setSrc(obj5);
```

Le code Java généré s'assure que l'objet obj4 n'ai plus de référence sur obj6 après l'opération.

Création d'objets : les classes **`Factory`**

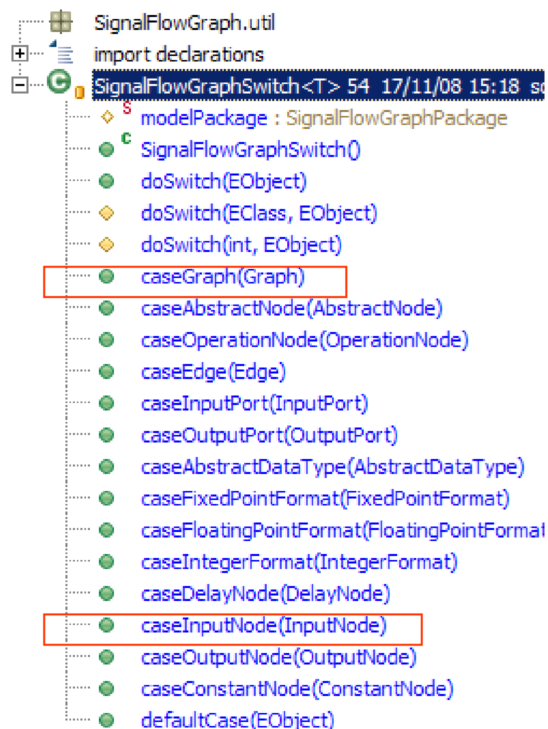
- *Design Pattern* Factory = objet qui « regroupe » les fonctionnalités de création d'objets du modèle.
  - Utilisé par les autres Classes/outils généré(e)s par EMF
- Si il y a besoin de «spécialiser » la construction d'objet, on modifie cette classe, pas les constructeurs.

```
 * <copyright>
package SignalFlowGraph.impl;

import SignalFlowGraph.*;

/**
 * <!-- begin-user-doc -->
 * An implementation of the model <b>Factory</b>.
 * <!-- end-user-doc -->
 * @generated
 */
public class SignalFlowGraphFactoryImpl extends EFactoryImpl implements
        SignalFlowGraphFactory {

    /**
     * <!-- begin-user-doc -->
     * <!-- end-user-doc -->
     * @generated NOT
     */
    public OutputNode createOutputNode() {
        OutputNodeImpl node  = new OutputNodeImpl();
        node.setId("O" + (out_node_id++));
        node.getIn().add(createInputPort("I"));
        return node;
    }
```

Le marqueur **`@generated not`** empêche l'écrasement du code lors d'une régénération du code Java

Parcours : les classes **`SwitchVisitor`**

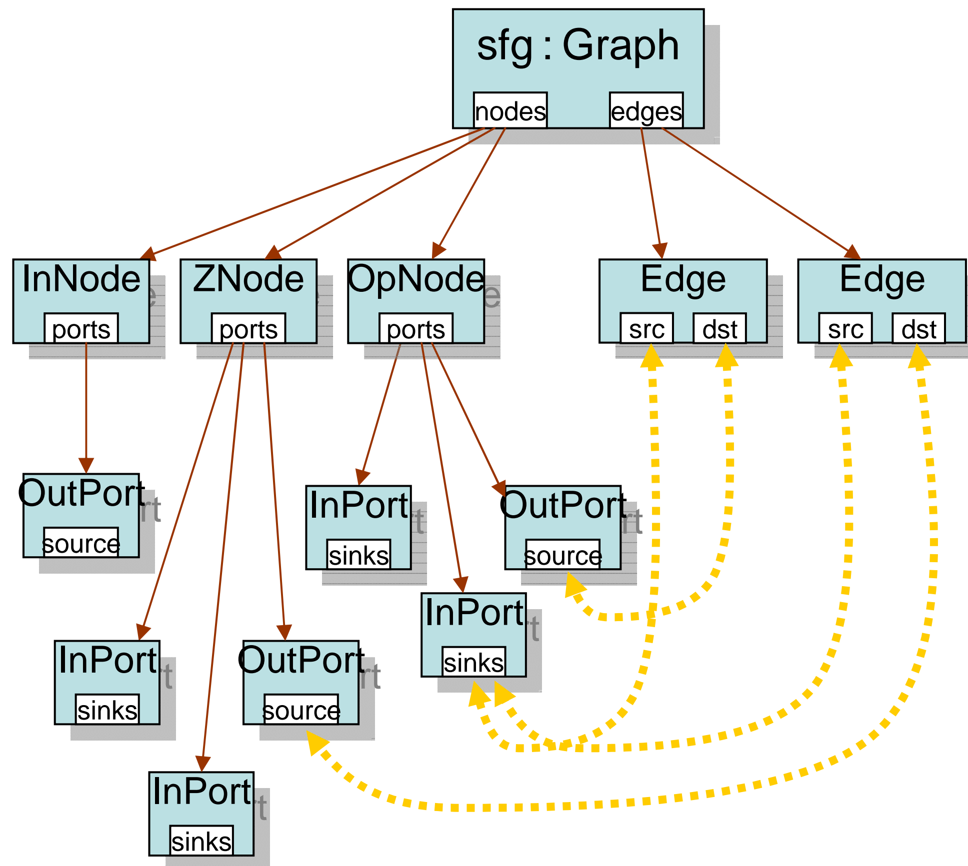
- Pour parcourir le modèle et appliquer des traitements.
  - Traitement en fonction de la Classe concrète de l'objet visité.
  - Le programmeur contrôle le parcours de la hiérarchie d'objets.
    - Il crée une classe qui hérite du **`SwitchVisitor`** généré par EMF
    - Il surcharger les méthodes **`case`*****`MyClass`*****`()`** selon ses besoins




```java
import org.eclipse.emf.common.util.BasicEList;

public class TopologicalSort extends SignalFlowGraphSwitch {

    @Override
    public Object caseGraph(Graph object) {
        for (AbstractNode node  : object.getNodes()) {
            doSwitch(node);
        }
        return null;
    }

    @Override
    public Object caseInputNode(InputNode object) {
        debug("Visiting InputNode "+object.getId());
        sortedNodes.add(object);
        for (OutputPort oport : object.getOut()) {
            for (Edge sink : oport.getSinks()) {
                doSwitch(sink);
            }
        }
        return null;
    }
```

Import/Export de XML (sérialisation)

- La notion de *containment* permet de « voir » le modèle comme une arborescence d'objets.
  - Simplifie grandement la génération de XML



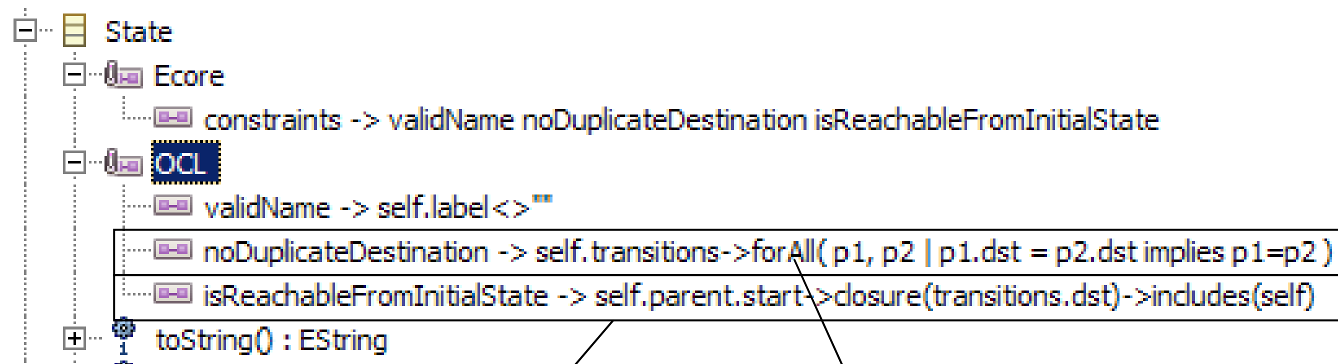

```
<graph>
   <InNode>
      <InPort src=… />
   </InNode>
   <ZNode>
      <InPort src=… />
      <OutPort sinks=… />
   </ZNode>
   <Edge src= … dst />
   <Edge src= …. />
</graph>
```

Validation des modèles

- Lorsque l'on crée/transforme des modèles, il est fréquent d'obtenir des représentations incohérentes.
  - Ex : un graphe dont un arc (Edge) n'est connecté à aucun nœud.
- EMF génère un validateur qui vérifie que les contraintes exprimées par le modèle sont bien respectées
  - Ex : si la cardinalité des champs src et dst de la classe Edge vaut 1, alors un arc non connecté rendra le modèle « invalide ».
- Evidement, cela n'est pas suffisant …
  - En pratique, un modèle valide doit satisfaire de nombreuses contraintes qui ne peuvent être rendues explicites ds le modèle.
  - Exemple : tous les cycles du graphe doivent contenir au moins un nœud de type « retard (Z)»

Validation des modèles

- Pour exprimer des contraintes plus complexes, on peut utiliser OCL (*Object Constraint Language*)
  - Langage très expressif et sans effets de bord
  - On ajoute les contraintes OCL dans le méta-modèle
- Exemple : machine à état



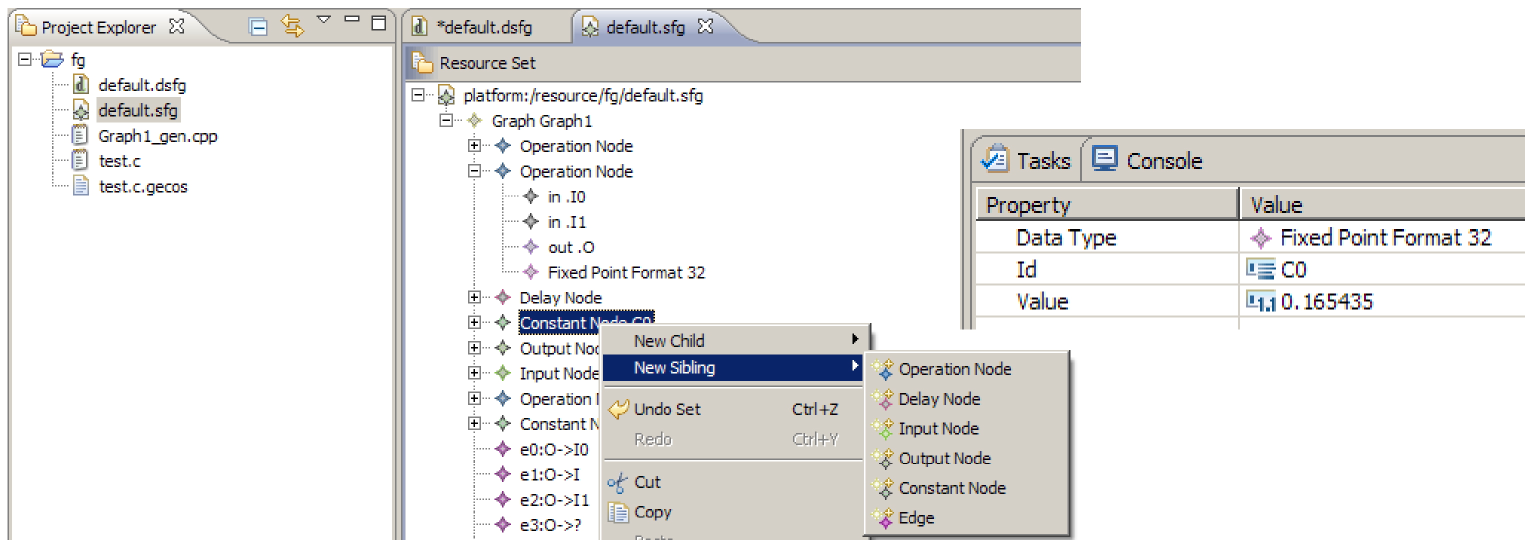
Tout état est atteignable à partir de l'état initial (graphe convexe).

On utilise l'opérateur `closure()` qui réalise la fermeture transitive.

Il ne peut pas y avoir deux transitions sortantes ayant la même destination.

Génération d'éditeurs (simples)

- EMF génère directement à partir du `.ecore` un éditeur arborescent permettant d'éditer des modèles.

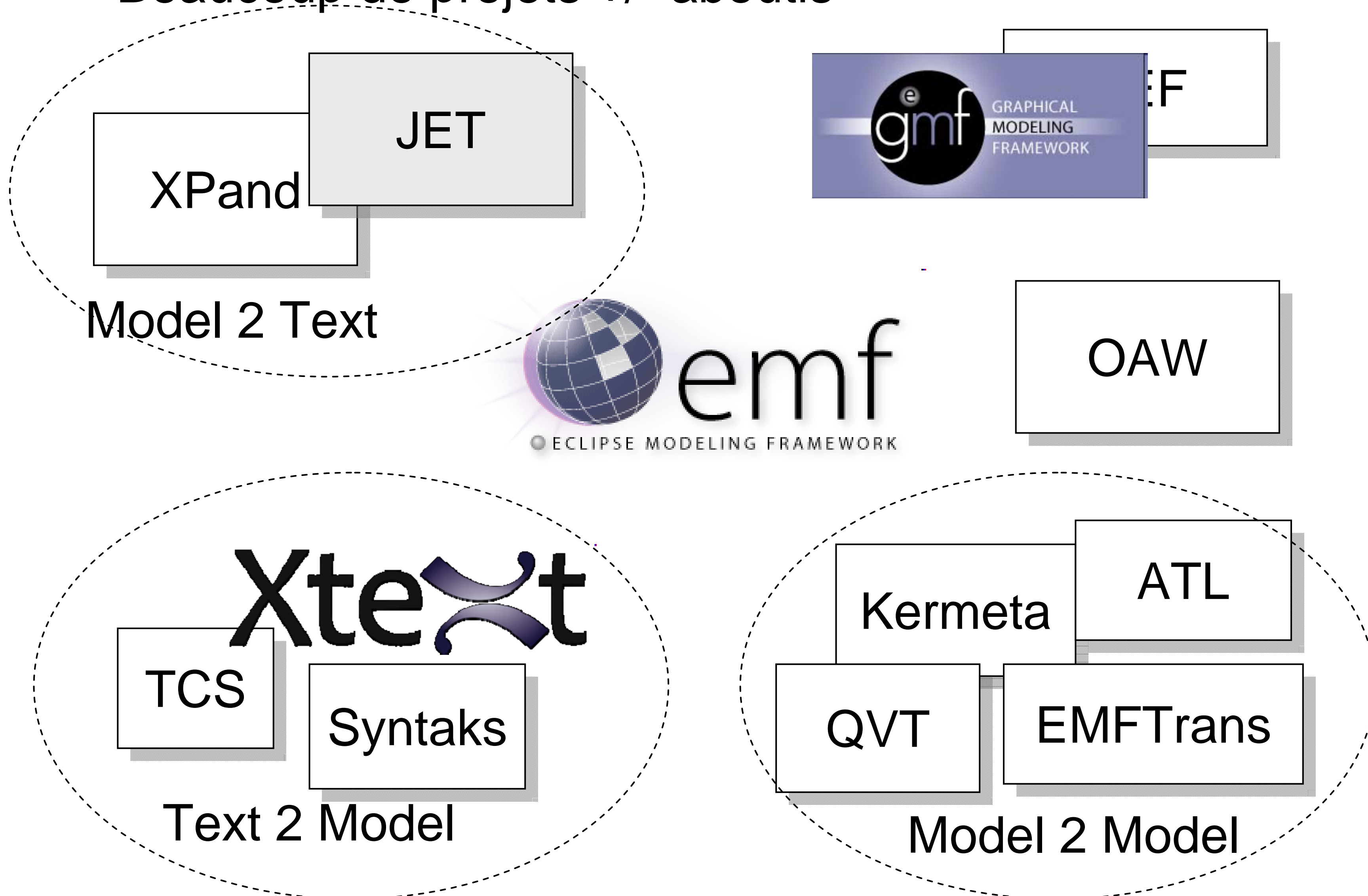


- Il s'avère **très** pratique pour la visualisation de modèles
- Cet éditeur est très facilement « customisable »
  - Labels, icônes, ajout de commandes, etc …

Autres outils

La jungle EMF

- Beaucoup de projets +/- aboutis



Lexique

- M2M = *Model to Model*
  - Transformation d'une famille de modèles vers une autre
    - Exemple : transformation d'une machine à état (modèle N°1) vers une *netlist* à base de portes logiques (Modèle N°2)
  - Transformation *in situ* d'un modèle
    - Exemple : simplification d'un modèle de graphe flot de signal après une propagation de constante.

- M2T = *Model to Text*
  - Transformations de génération de code/texte à partir d'une instance de modèle,
    - Exemple : génération de VHDL synthétisable pour un modèle de SFG « à plat »
  - Approches basées sur la notion de Template (squelettes)

Lexique

- T2M = *Text to Model*
  - Outils permettant de définir des *Domain Specific Languages* associés à des modèles particuliers.
  - Permet de créer des langages dédiés pour spécifier textuellement un type de modèle.
    - Exemple : un langage pour décrire des SFG qui seront ensuite transformés en un modèle
- Remarques :
  - Souvent ces outils permettent de faire la transformation dans le sens inverse (régénérer le code à partir d'un modèle)
  - Outils extrêmement puissants mais qui restent pourtant assez simples à utiliser

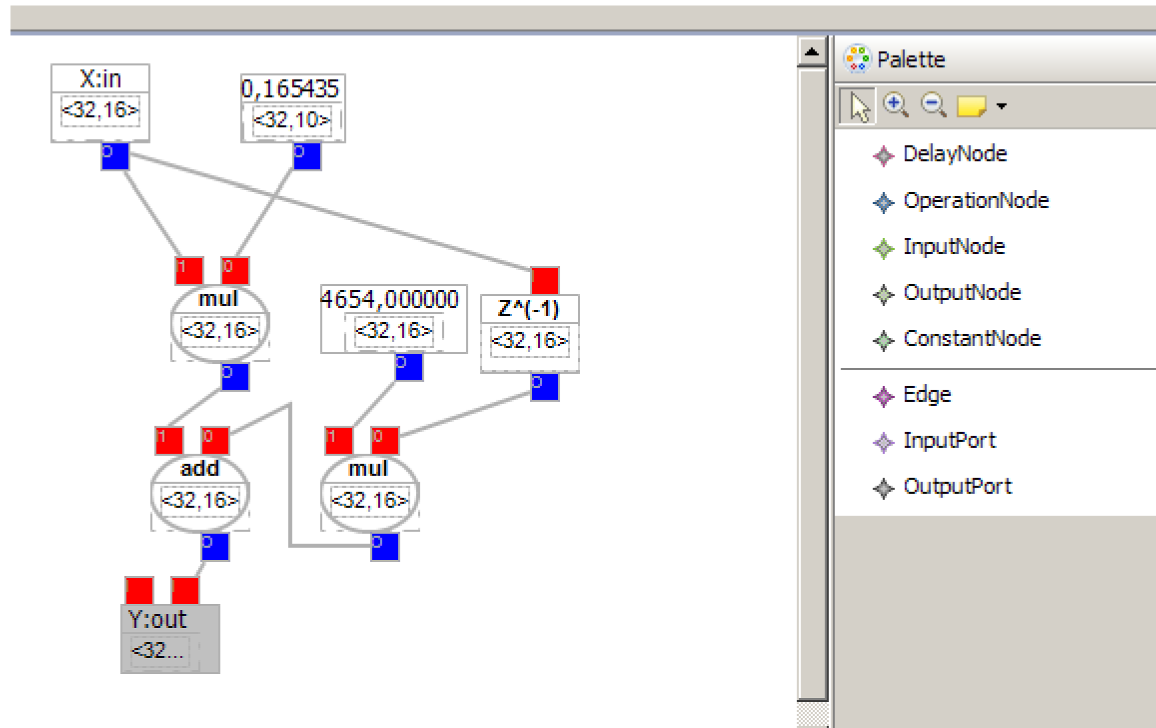
Génération d'éditeurs avec GMF

- Permet de définir des éditeurs « à la Ptolemy »
  - GMF définit un méta-modèle d'éditeur visuel (notion de boites, de connexions, de commandes, etc.)
  - La conception de l'éditeur revient à lier les objets de son méta-modèle aux objets du méta-modèle GMF
    - On spécifie quel objet du modèle est un arc, un nœud, etc.
- Outils assez complexes à utiliser
  - Beaucoup de « magie noire », peu de documentation
  - Permet qd même à un utilisateur expérimenté d'obtenir un éditeur complexe en qq heures.

- *À surveiller* : Epsilon Eugenia qui simplifie cette étape

Génération d'éditeurs avec GMF

- Exemple : éditeur pour SFG
  - Généré sans avoir à saisir une seule ligne de code Java



M2T : Génération de code avec JET

- JET = Java Emitter Template

```
<%@ jet
    package="fr.irisa.cairn.model.signalflowgraph.codegen.mentorc.jetgen"
    imports="java.util.* org.eclipse.emf.common.util.EList
    fr.irisa.cairn.model.signalflowgraph.codegen.mentorc.TopologicalSort
    SignalFlowGraph.*"
  class="SignalFlowGraphMentorCGenerator"
%>

<% Graph sfg = (Graph) argument; %>

#include<stdio.h>
#include<mentor.h>

  <%
    for (Iterator i = sfg.getNodes().iterator(); i.hasNext(); ) {
        AbstractNode an = (AbstractNode) i.next();
        if (an instanceof ConstantNode) {
            ConstantNode c = (ConstantNode) an;
            %>#define  <%=c.getId()%> <%=c.getValue()%><%
        }
    }
    %>
```

JET spécifier un squelette de texte à générer (*template*)

Les morceaux manquants du squelette sont ajoutés à l’aide de code Java.

L’outil JET génère ensuite le code d’une classe Java qui produit le texte/code à partir d’un objet argument

T2M : Domain Specific Languages : XText

- Ou comment créer un langage en 30 minutes chrono …
  1. Définition de la grammaire du langage
  2. Inférence d'un méta modèle à partir de la grammaire

```
grammar org.xtext.example.MyDsl with org.eclipse.xtext.common.Terminals

generate myDsl "http://www.xtext.org/example/MyDsl"

Seminaire : 'seminaire' name=ID '{'
    membres+=Membre (',' membres+=Membre)* ';'
    groupes+=Groupe ( ',' groupes+=Groupe)* ';'
    '}';

Membre : 'participant' name=ID '{'
        'site' '=' site=("CAIRN-Rennes"|"CAIRN-Lannion") ';'
        'statut' '=' statut=("Doctorant" | "Permanent" | "Ingenieur" ) ';'
'}';

Groupe: 'chambre' numero=INT '{'
     personnes+=[Membre] ( ',' personnes+=[Membre])
```

(1)

On fait l'Edition de lien directement à partir de la grammaire !

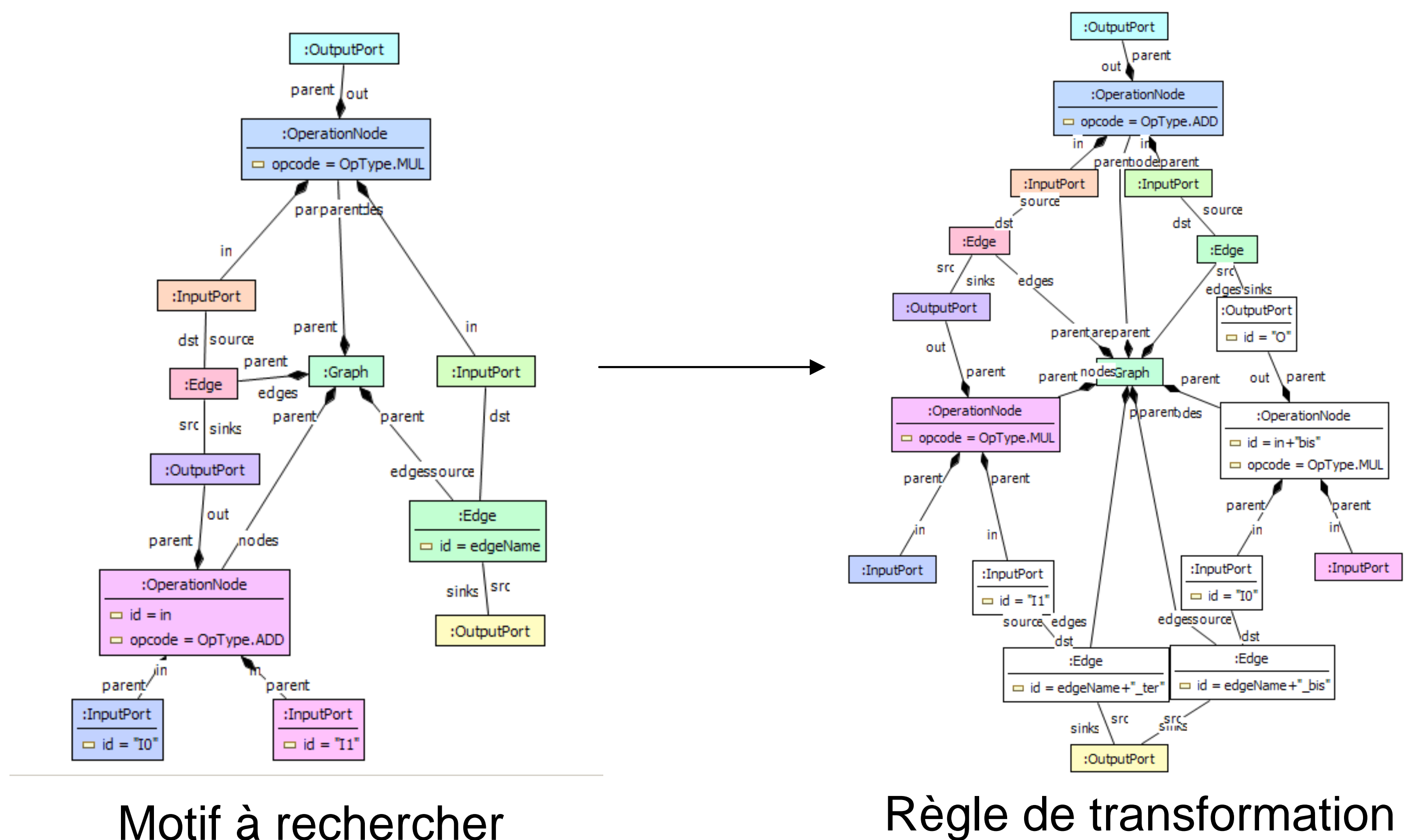
```
myDsl
  Seminaire
    name : EString
    membres : Membre
    groupes : Groupe
  Membre
    name : EString
    site : EString
    statut : EString
  Groupe
    numero : EInt
    personnes : Membre
```

(2)

  3. Génération *automatique* du modèle à partir de sa repr. textuelle
  4. Génération *automatique* de l'éditeur avec coloration+complétion

M2M : Transformations de modèles avec EMFTrans

- Transformations basées sur des grammaires de graphes
- Revient à chercher des motifs/patterns dans un graphe
  - On peut définir de anti-patterns (patterns *bloquants*)
- Exemple : distributivité dans un SFG (a (b+c)=ab+ac)



Motif à rechercher

Règle de transformation

Conclusion

Pourquoi utiliser cette techno ?

- Normalisation/standardisation du code
  - Le programmeur est pris dans ‘un carcan’, il ne fait plus ce qu’il veut, le code est du coup un peu plus « standardisé ».
- L’essentiel de l’information est dans le modèle
  - Le modèle peut (doit) servir de spécification logicielle.
  - Lors de la reprise de code écrit par qqu’un d’autre, le modèle permet d’appréhender + rapidement le travail.
- Il existe de nombreux outils à forte valeur ajoutée
  - XText : génération d’éditeurs de texte (complétion +coloration)
  - GMF : génération d’éditeur graphiques (schéma-blocs)

Pourquoi utiliser cette techno ***chez CAIRN*** ?

- Parce que nous sommes les MM Jourdain de l'IdM
  - On passe notre temps à manipuler/transformer des modèles
    - Compilation HLS (AST, CDFG, SFG, HCDG, RTL , …)
    - IdM faite pour la conception de compilateurs (au sens large)
  - On se place de + en plus en fournisseur d'outils, donc il faut essayer d'améliorer notre façon de produire du logiciel.
- Les autres communautés de l'embarqué y sont déjà
  - Communauté temps-réel/logiciel embarqué (projet *OpenEmbed*)

Oui mais c’est trop compliqué …

- Faux : expérience du projet M1 C2Silicium
  - Un groupe d’étudiant de M1Info a réalisé un prototype de compilateur C->VHDL en utilisant un méta-modèle de circuit
  - Ils ne connaissaient rien à l’IdM avant de commencer, et n’avaient que peu de pratique du dev. orienté objet.
  - Vous jugerez du résultat à la séance démo …

- Mais :
  - Il faut quand même des connaissances de base en P.O.O
  - Il faut avoir un expert à portée de main/téléphone au début.
    - Même à Rennes, on s’est débrouillé sans …

Des méta-modèles chez CAIRN ?

- Méta modèle du CDFG de Gecos
  - Visualisation du CDFG par un éditeur très pratique
  - Passes de Gecos « relativement » faciles à porter
- Méta modèle d’architecture FSMD
  - Objectif disposer d’un modèle pour un back-end matériel
  - FSMD = FSM + Datapath (circuit synchrone)
  - Analyses statiques (BDD sur les prédicats de la FSM)
  - Génération de code VHDL à partir du modèle
  - À venir : transformations de retiming, extraction de primitives, datapath merging, …

Questions

Un méta-modèle ‘RTL’ synchrone

- Une approche basée sur le modèle FSMD
  - FSMD = FSM+Datapath
  - Typage des connections (flot de contrôle/donnée)
- Analyses statiques
  - Chemin critique
  - Atteignabilité d’état (parcours + BDD)
  - Transformations (retiming, c-slow, pipeline, etc …)
- Génération de code
  - Sortie dotty (FSM)
  - Back-end VHDL
  - Back-end Java (Ptomemy II Discrete-event)

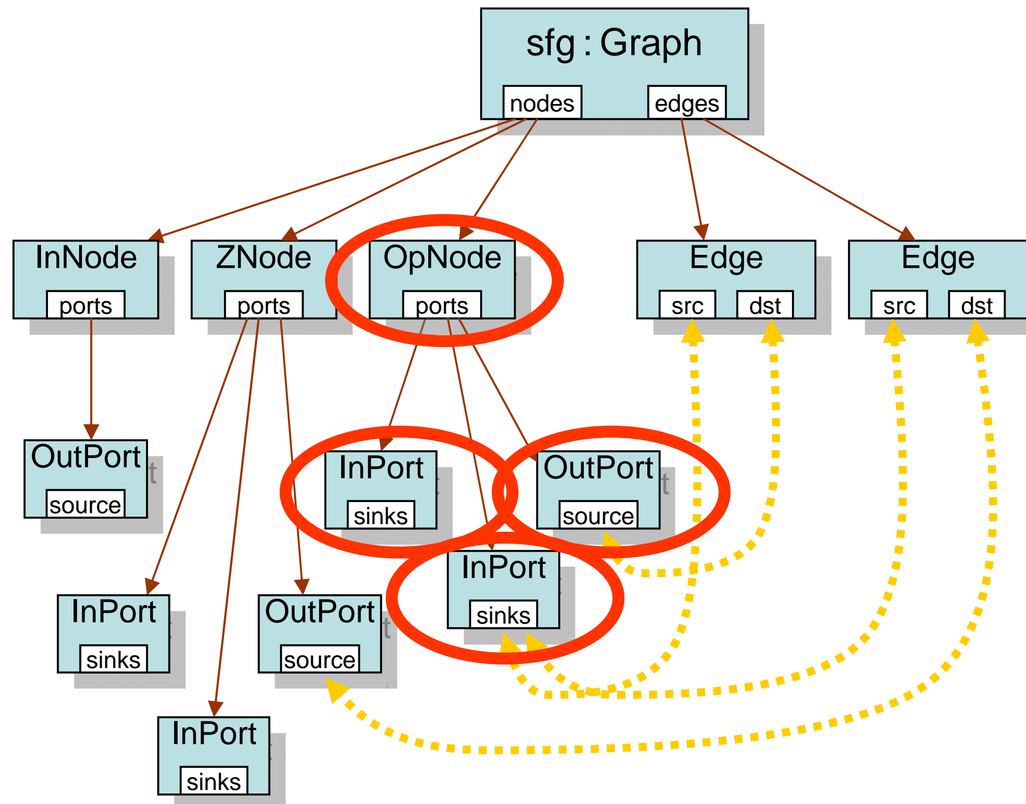
Un méta-modèle FSMD

- Permet de construire et manipuler programmatiquement des représentations intermédiaire de circuits.
  - Principe général très proche de JHDL, PAMDC, sauf qu'ici on manipule explicitement un modèle de circuit synchrone.
  - Permet de « raisonner » sur ce modèle car les éléments qui le composent ont une sémantique claire.
- Offre une représentation intermédiaire intéressante pour un outil de HLS ou de spécification « système ».
  - La génération de modèle exécutable (ex :VHDL) se fait directement à partir de cette représentation intermédiaire.

Poubelle

Copie

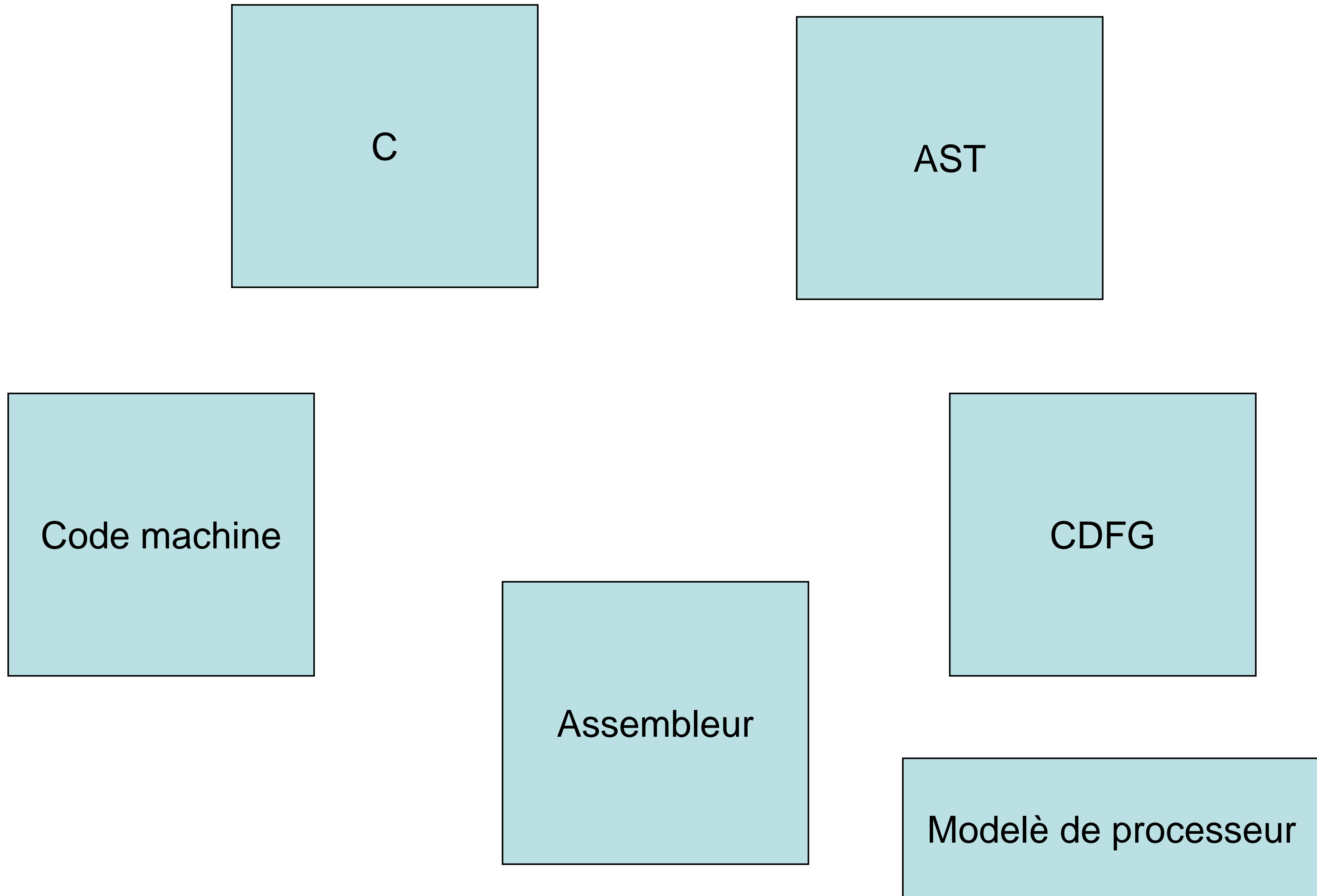
- EMF fournit une classe **`Copier`** qui permet de réaliser une copie « intelligente » des objets.
  - On réalise un copie de l'objet et des tous ses descendants liés par une relation de *containment*.



Kermeta

- M2M basé sur un langage impératif, dont les concepts sont ~~faciles~~ moins difficiles à appréhender que les autres outils

Une vue MDE d'un compilateur reciblable



Bibliographie