Eclipse Modeling Framework

Horváth Ákos

Dániel Varró

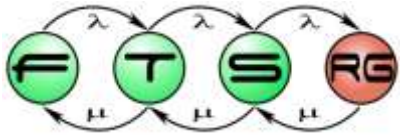


Table of Contents

- Overview of EMF
- EMF by Example
  - Prerequisites: the domain metamodel
  - Overview of autogenerated EMF interfaces
  - XMI Serialization
- Advance EMF

What does EMF provide?

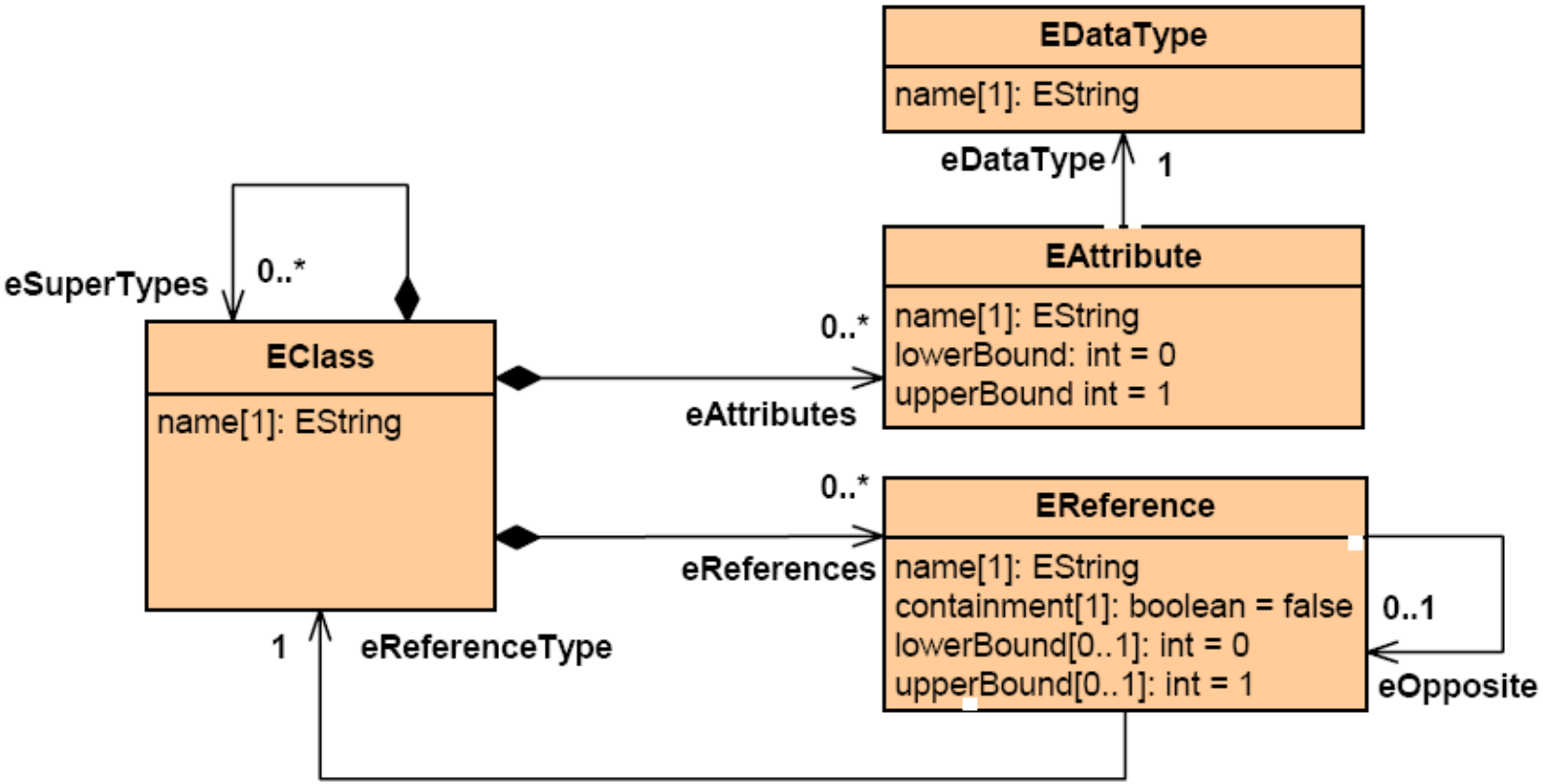
- EMF = Eclipse Modeling Framework
  - Reflective Metamodeling Core
    (Ecore ➔ MOF 2.0)
  - Support for Domain Specific Languages
  - Editing Support
    (Notification, Undo, Commands)
  - Basic Editor Support
  - XMI Serialization
  - Eclipse Integration

Ecore - why?

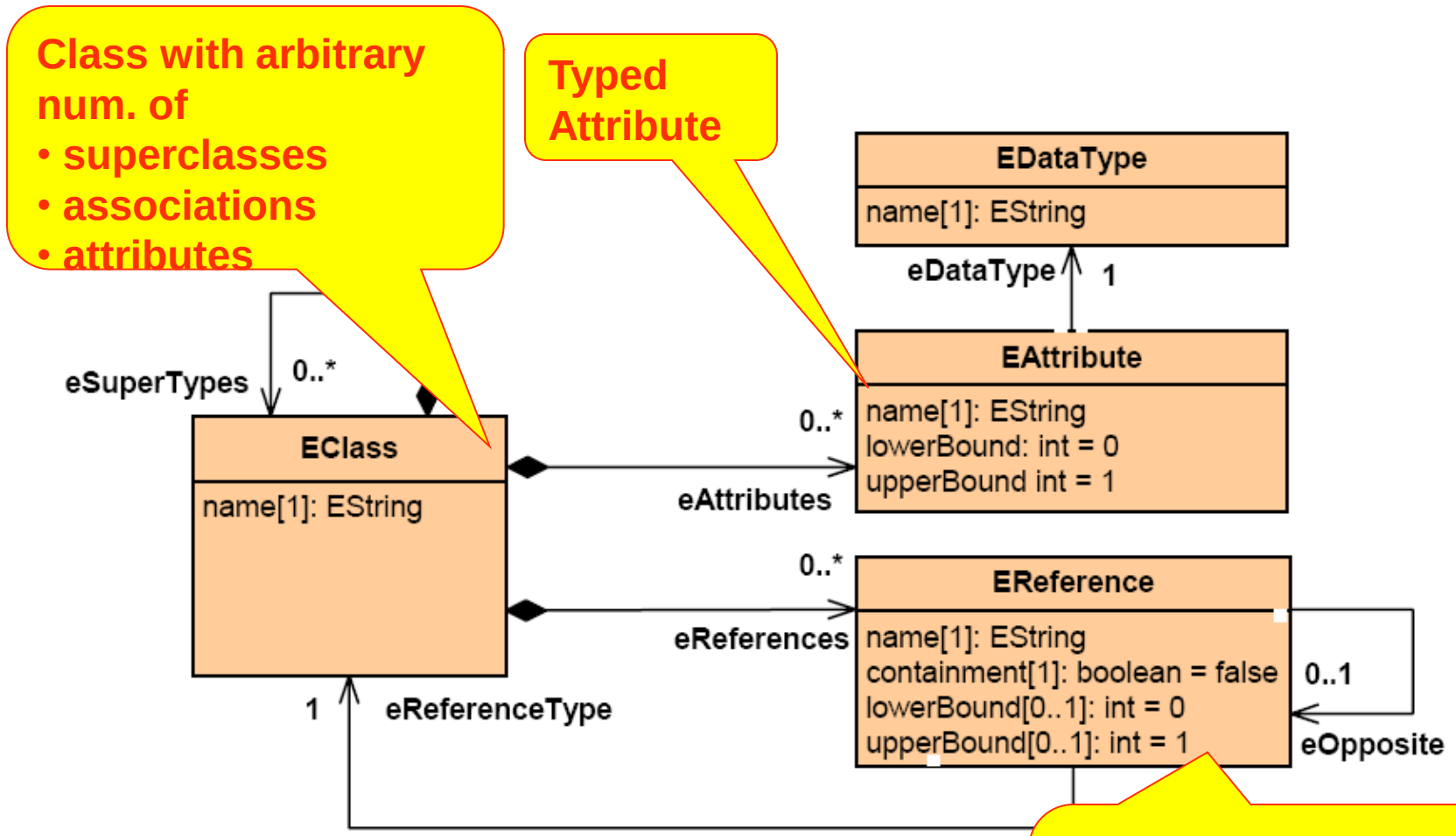
- Metamodeling language of EMF
- Metamodels are platform independent
- Defining structural models
- Eclipse supports Ecore based:
  - Code generation
  - Import/Export (various platforms)
  - Supported by IBM, etc.
- Ecore captures only the core functionalities
- Close to implementation models

Core EMF

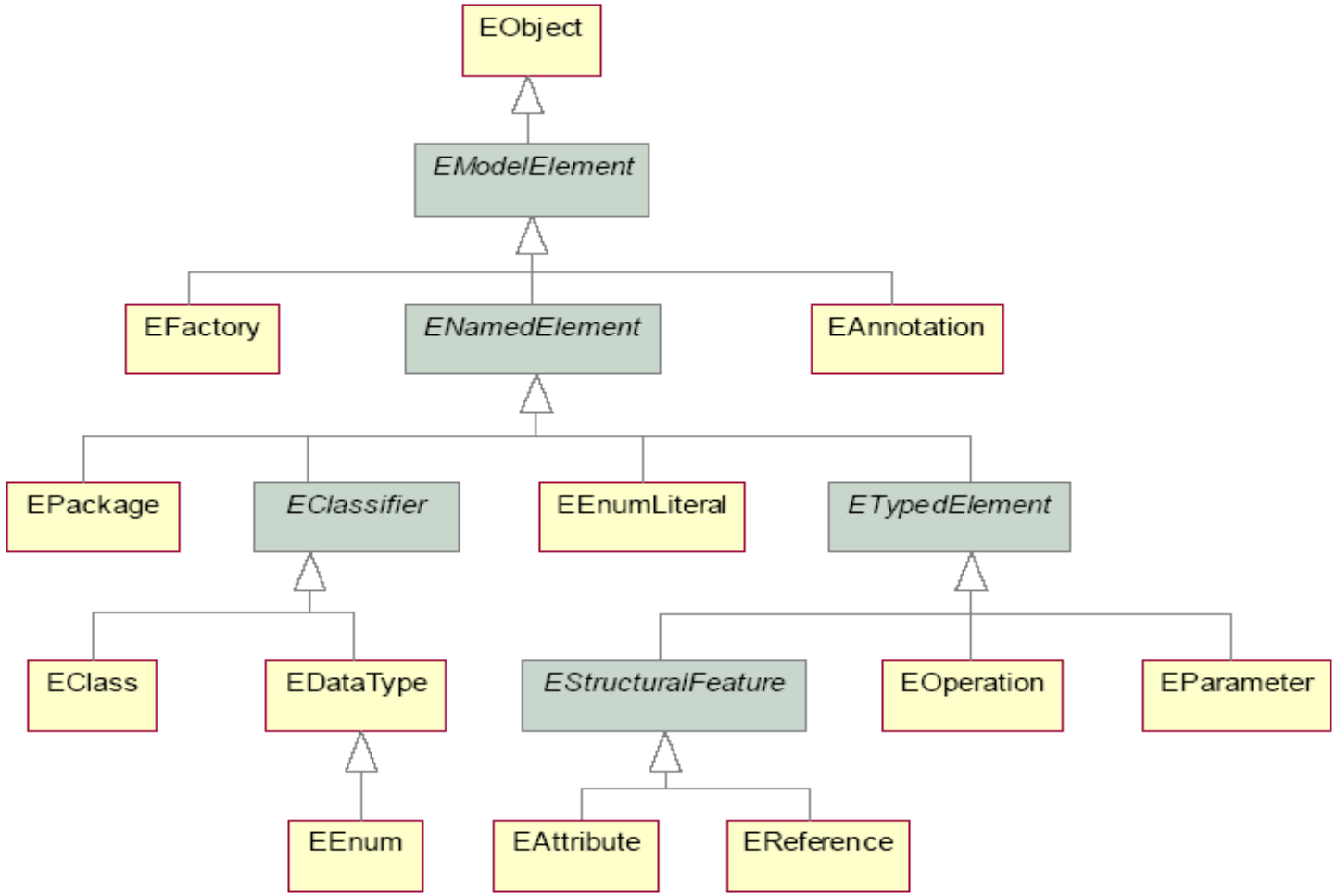
Core ECore constructs



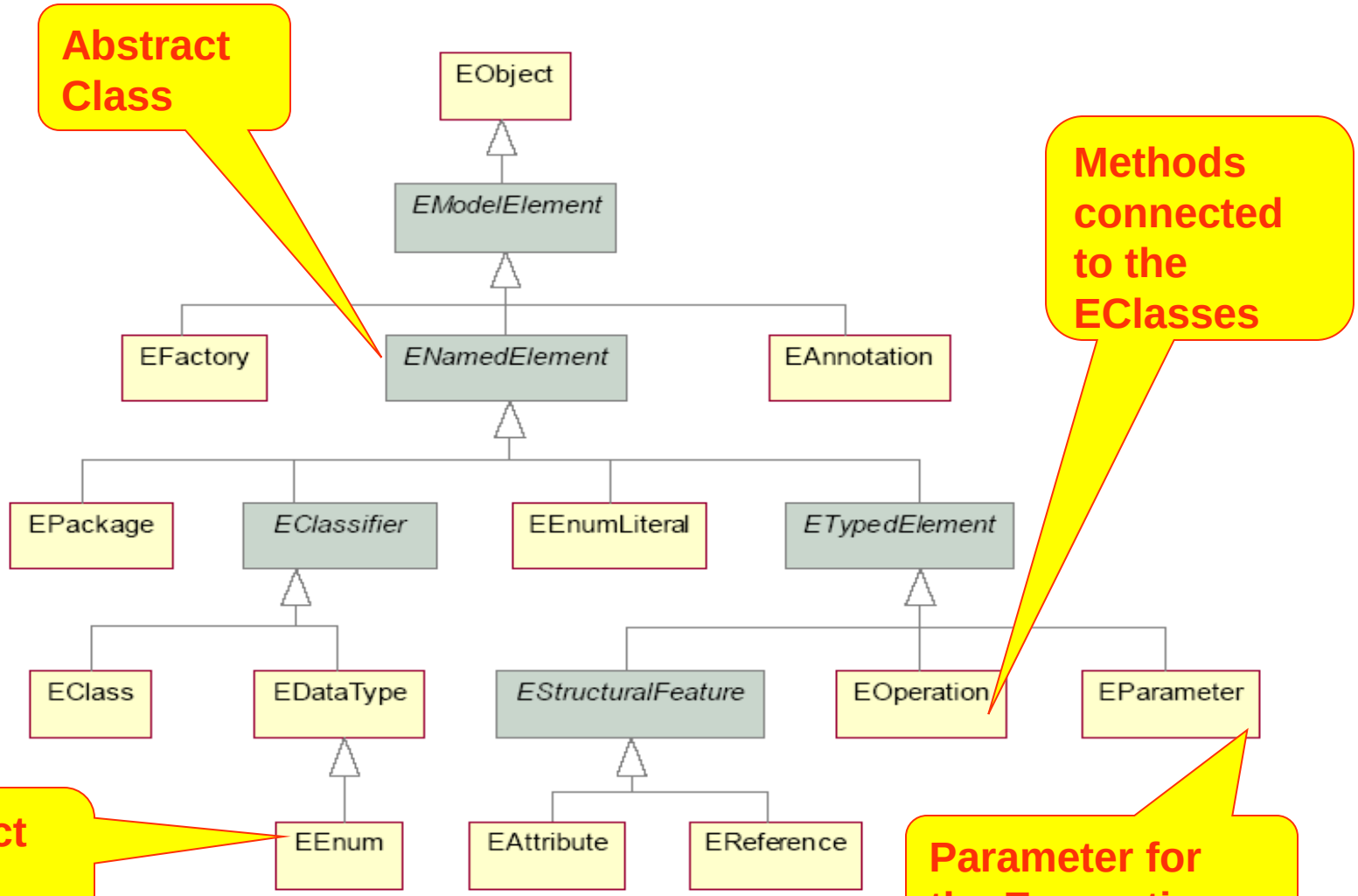
Core ECore constructs



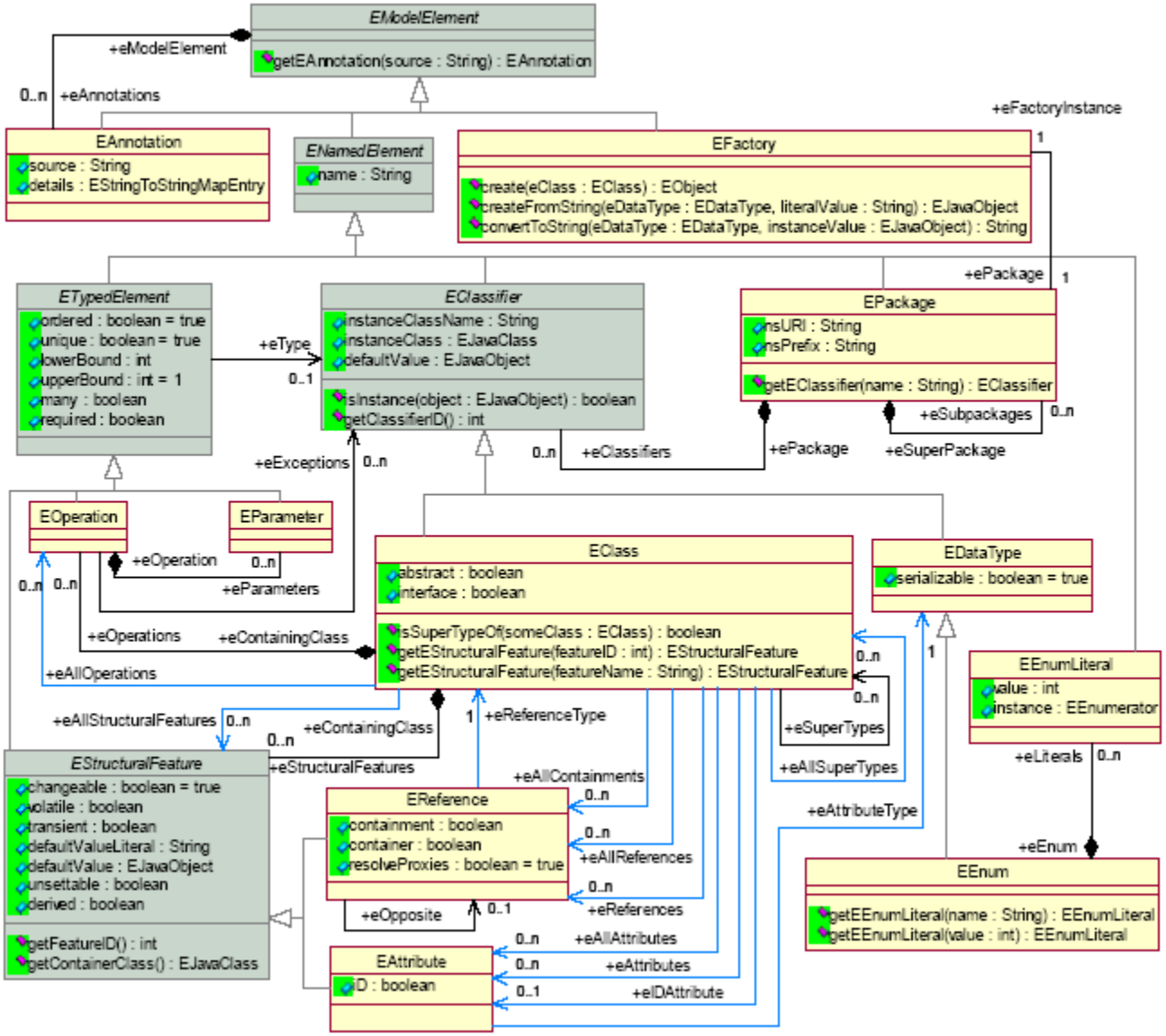
Complete ECore hiearchy



Complete ECore hiearchy

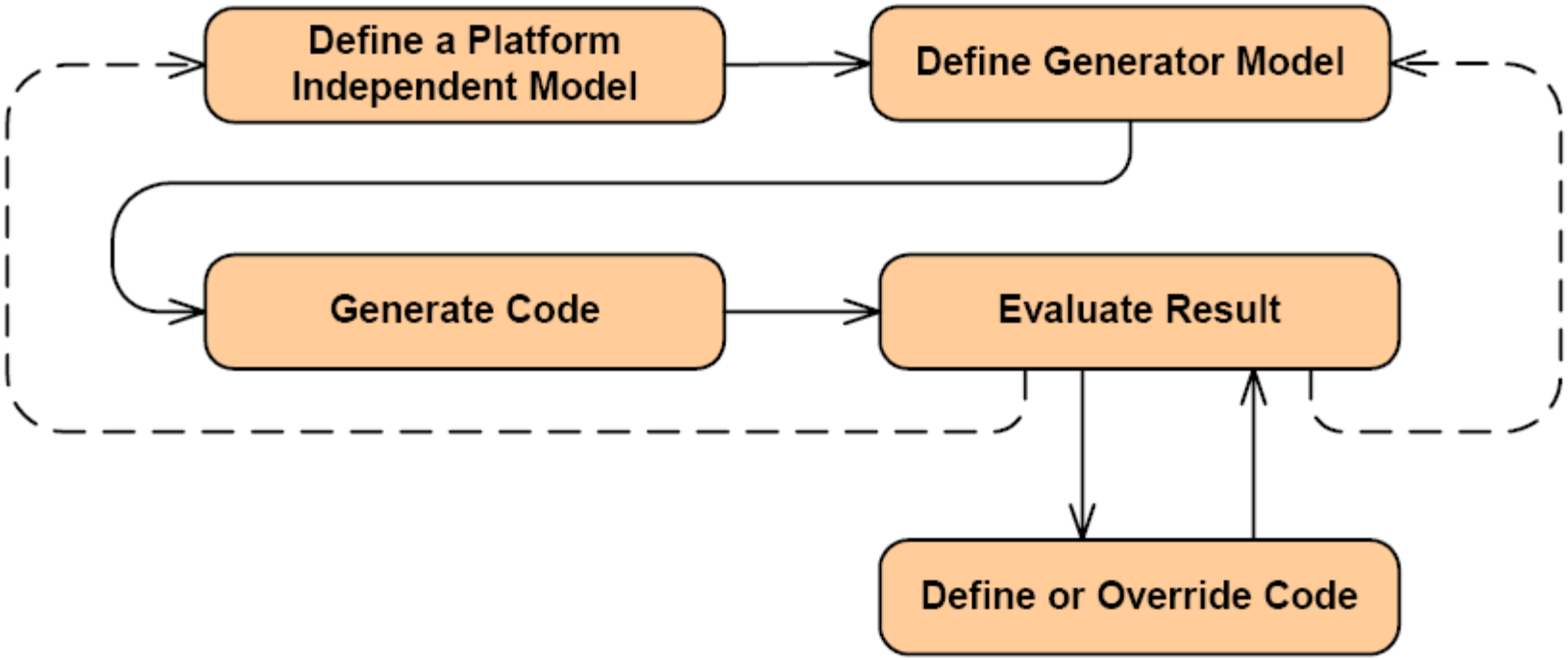


ECore Implementation

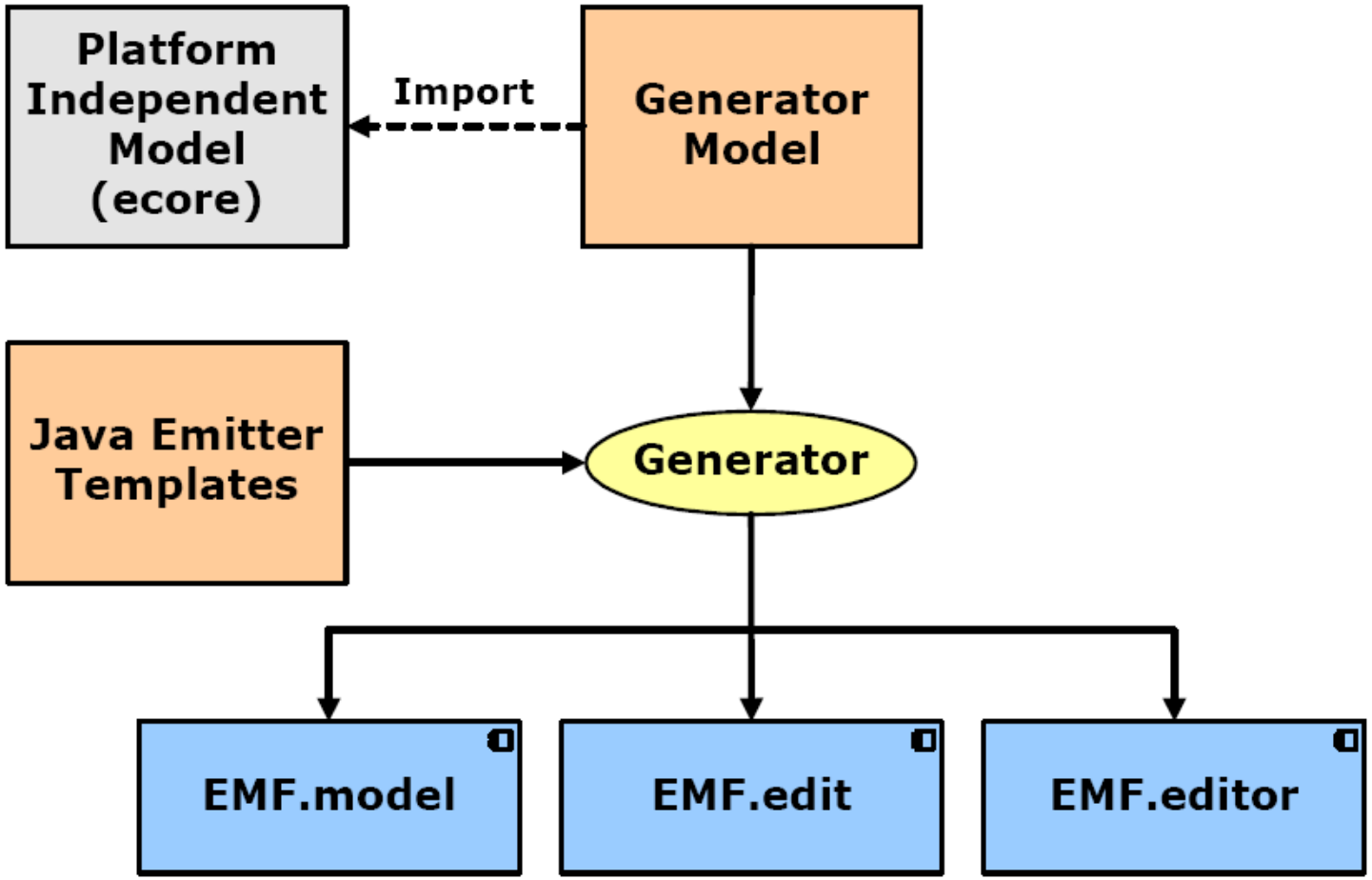


Defining a DSM
The EMF way

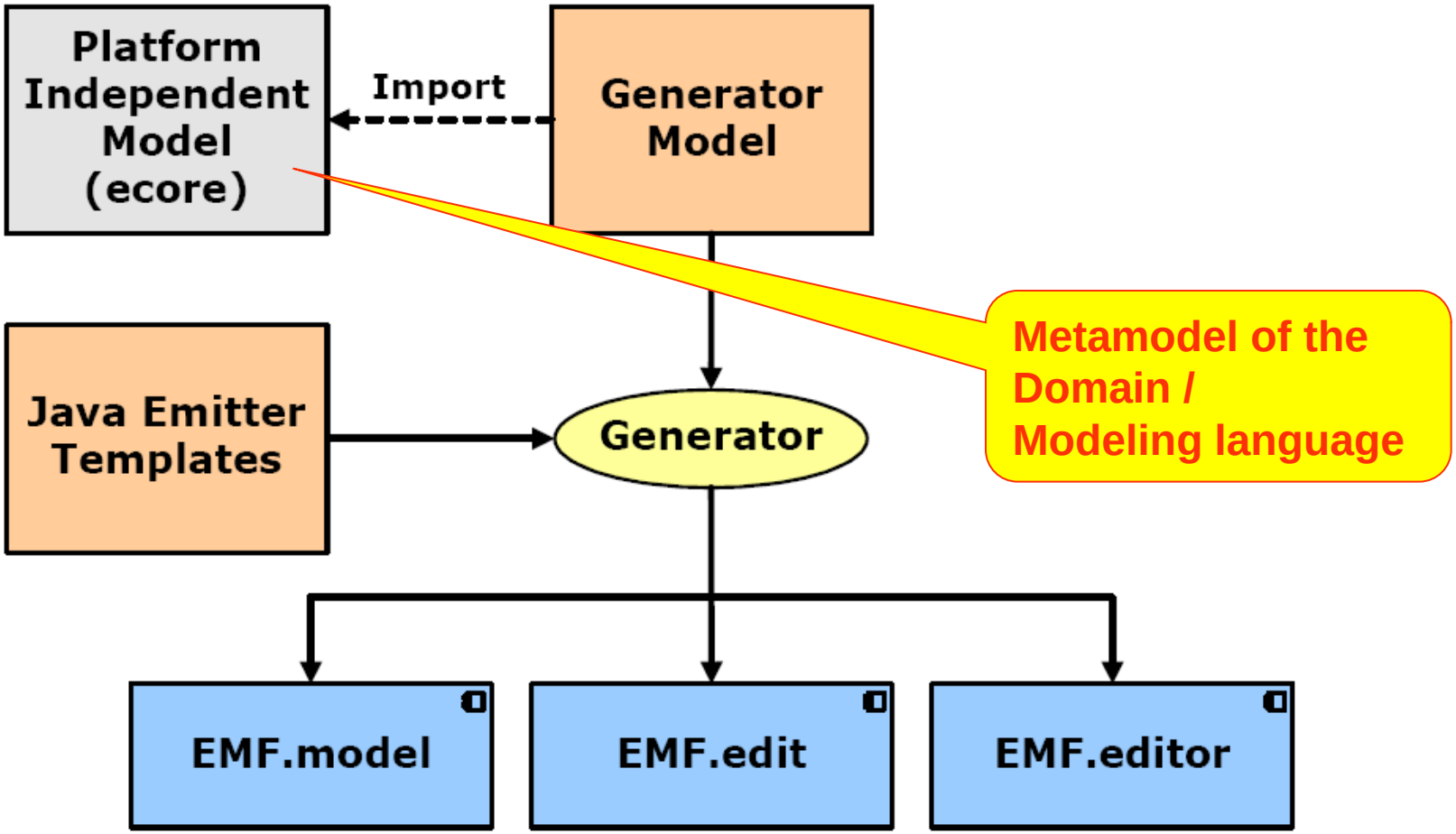
The Simple EMF Workflow



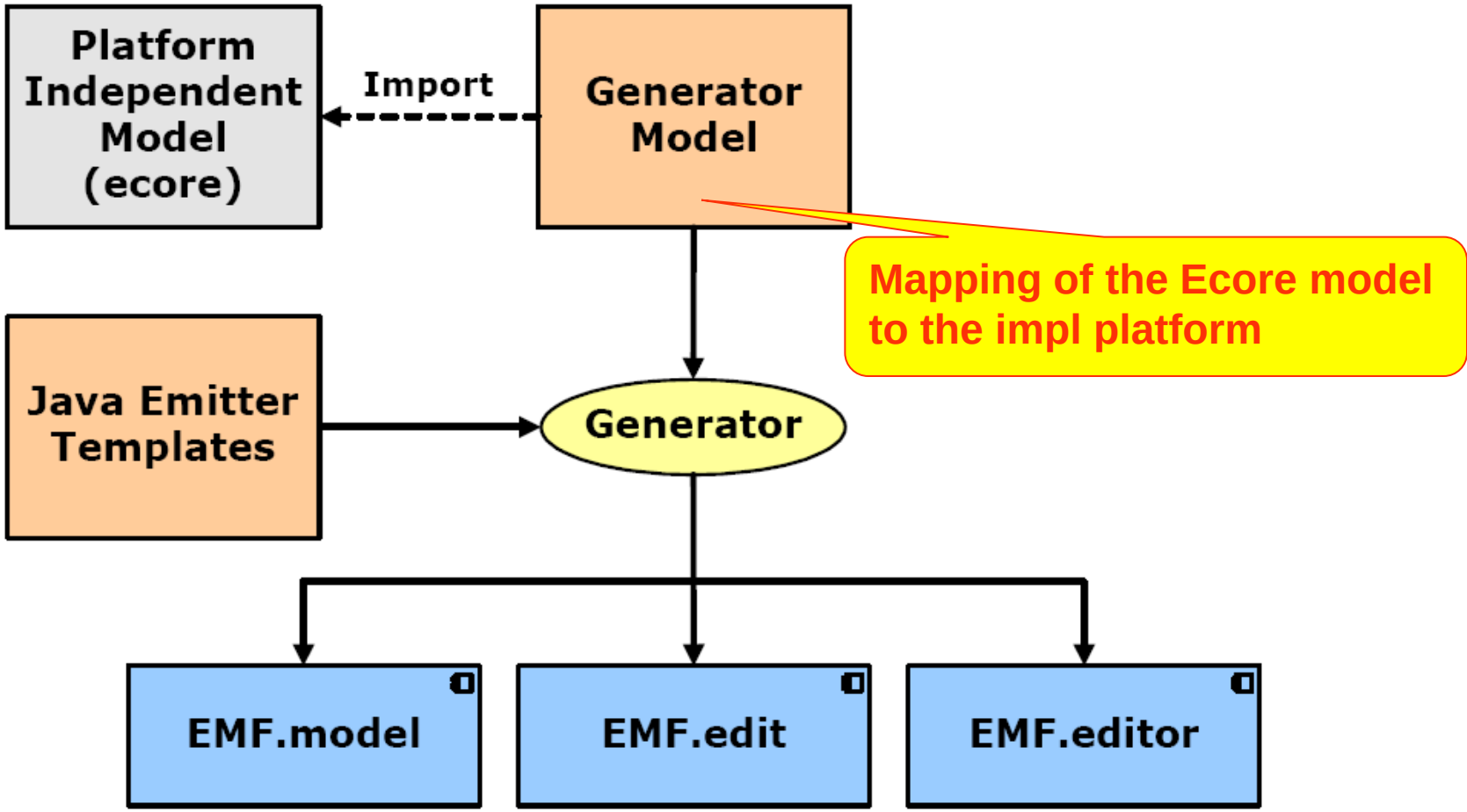
The EMF Toolkit



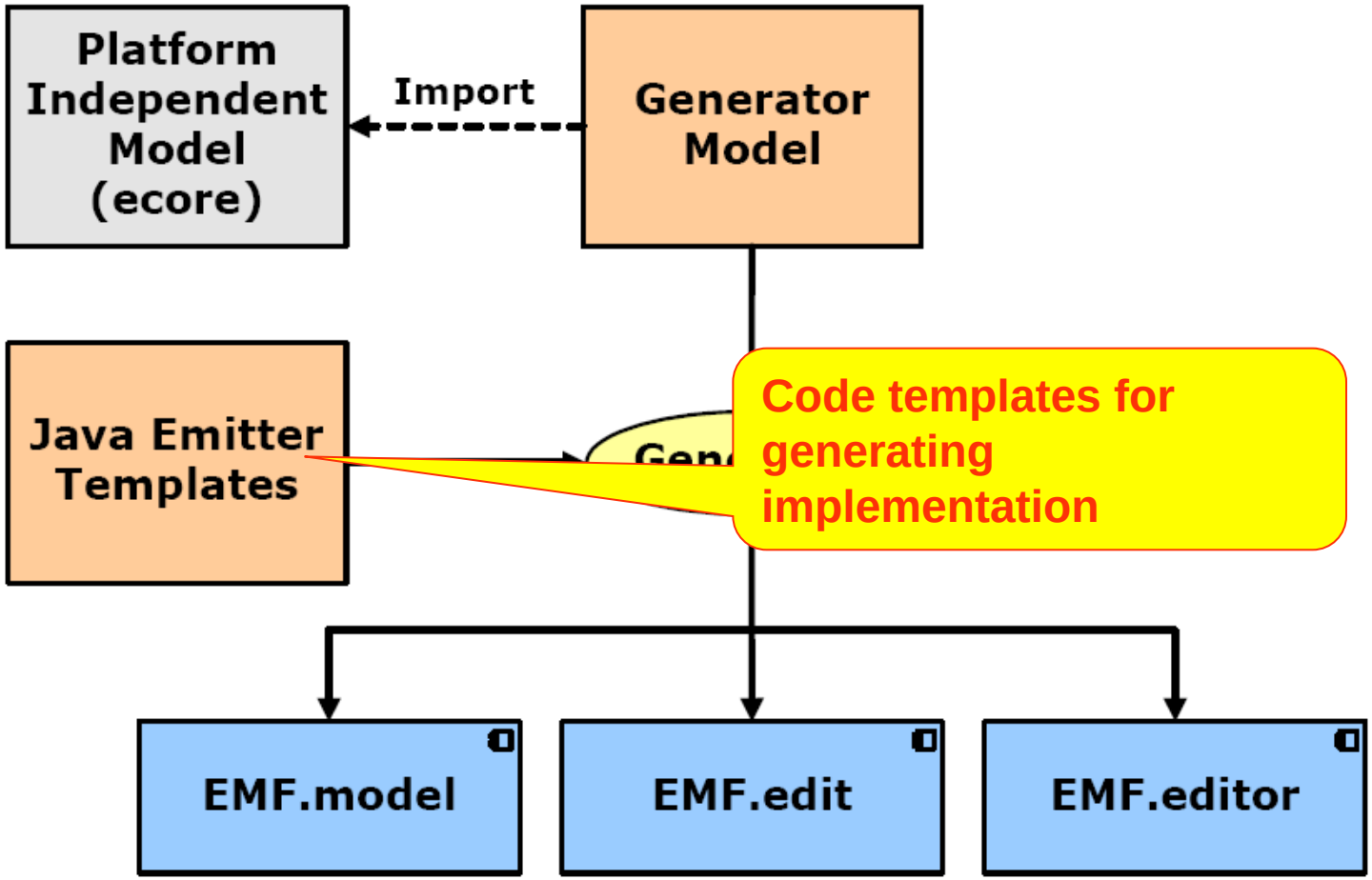
The EMF Toolkit



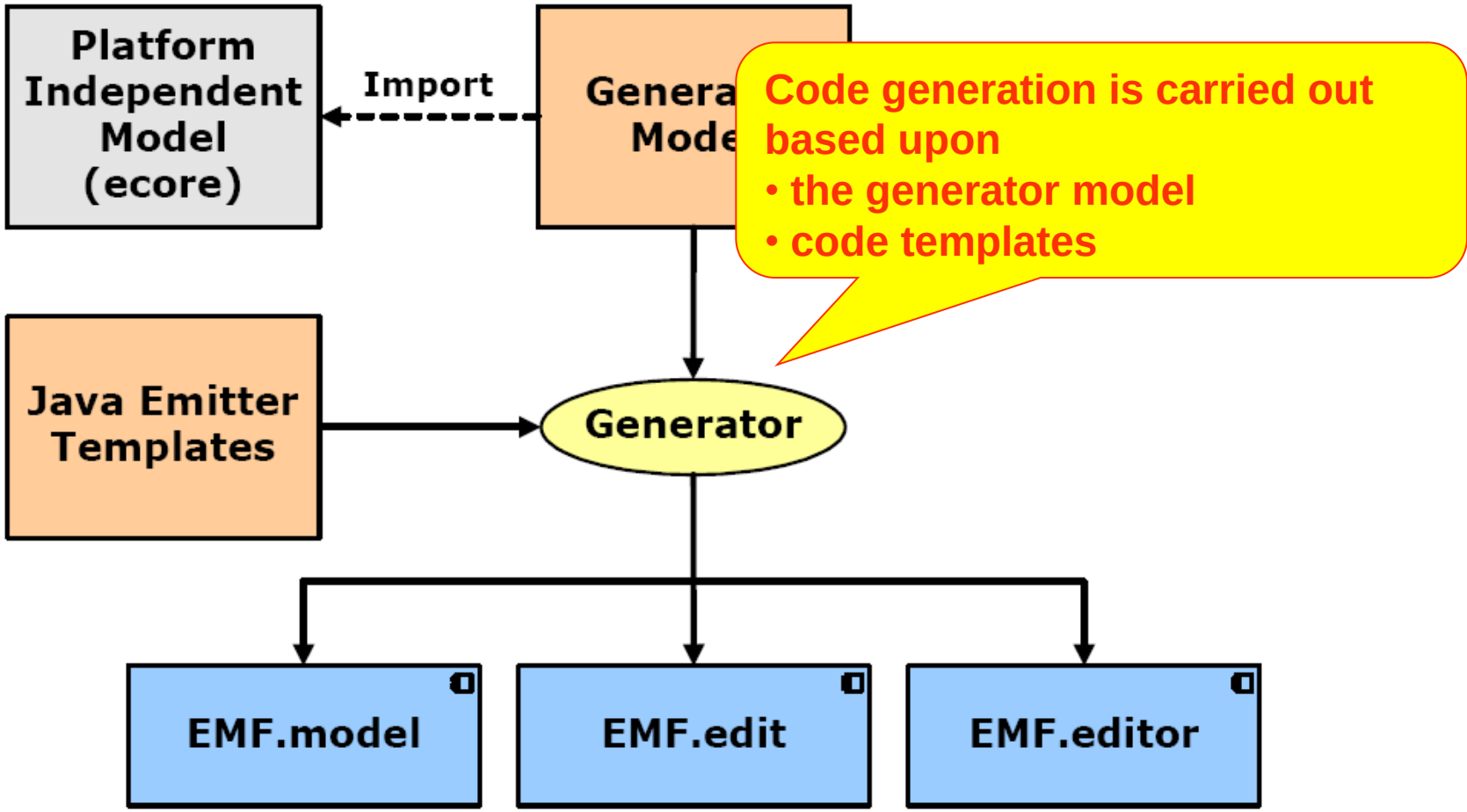
The EMF Toolkit



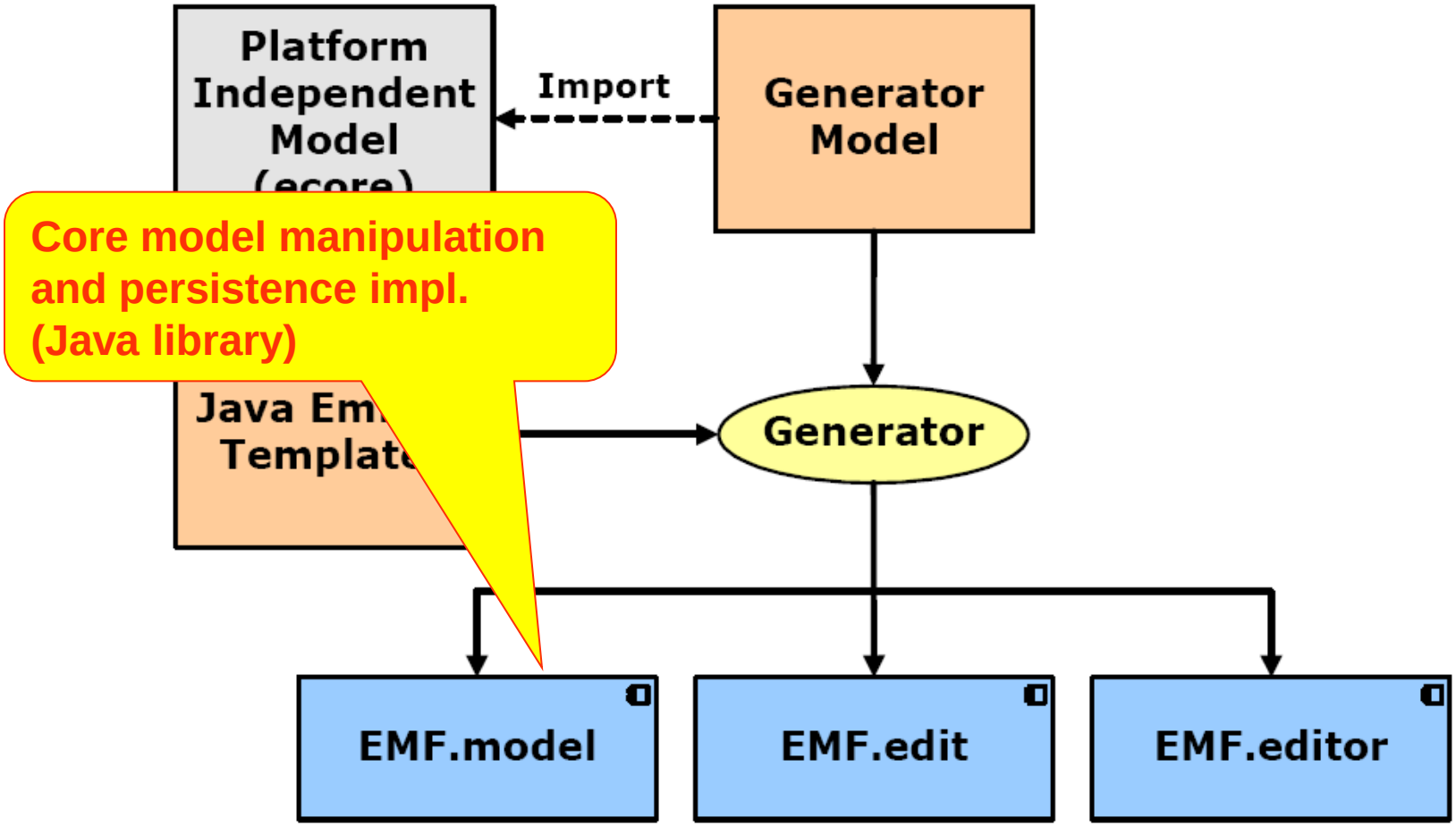
The EMF Toolkit



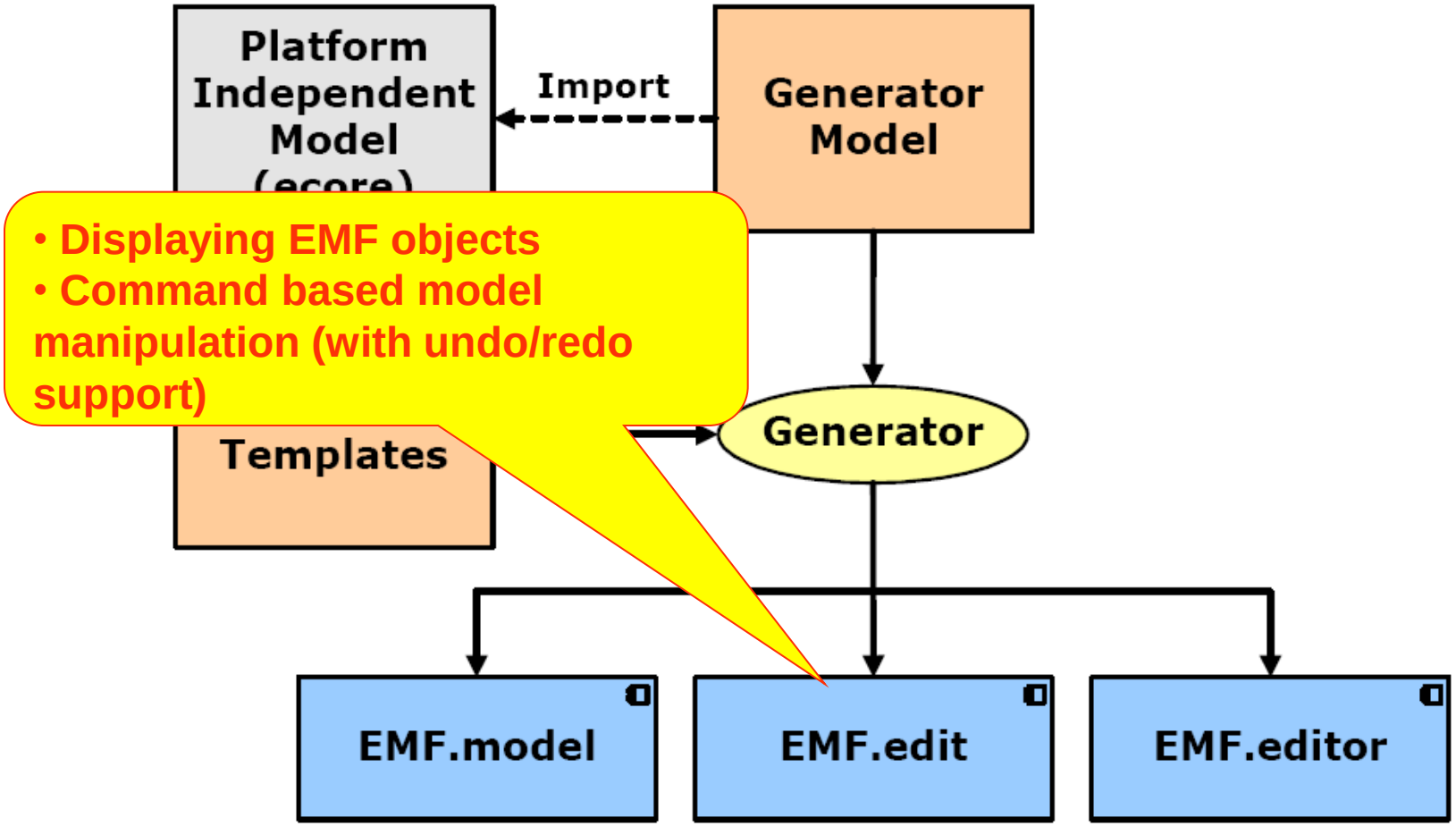
The EMF Toolkit



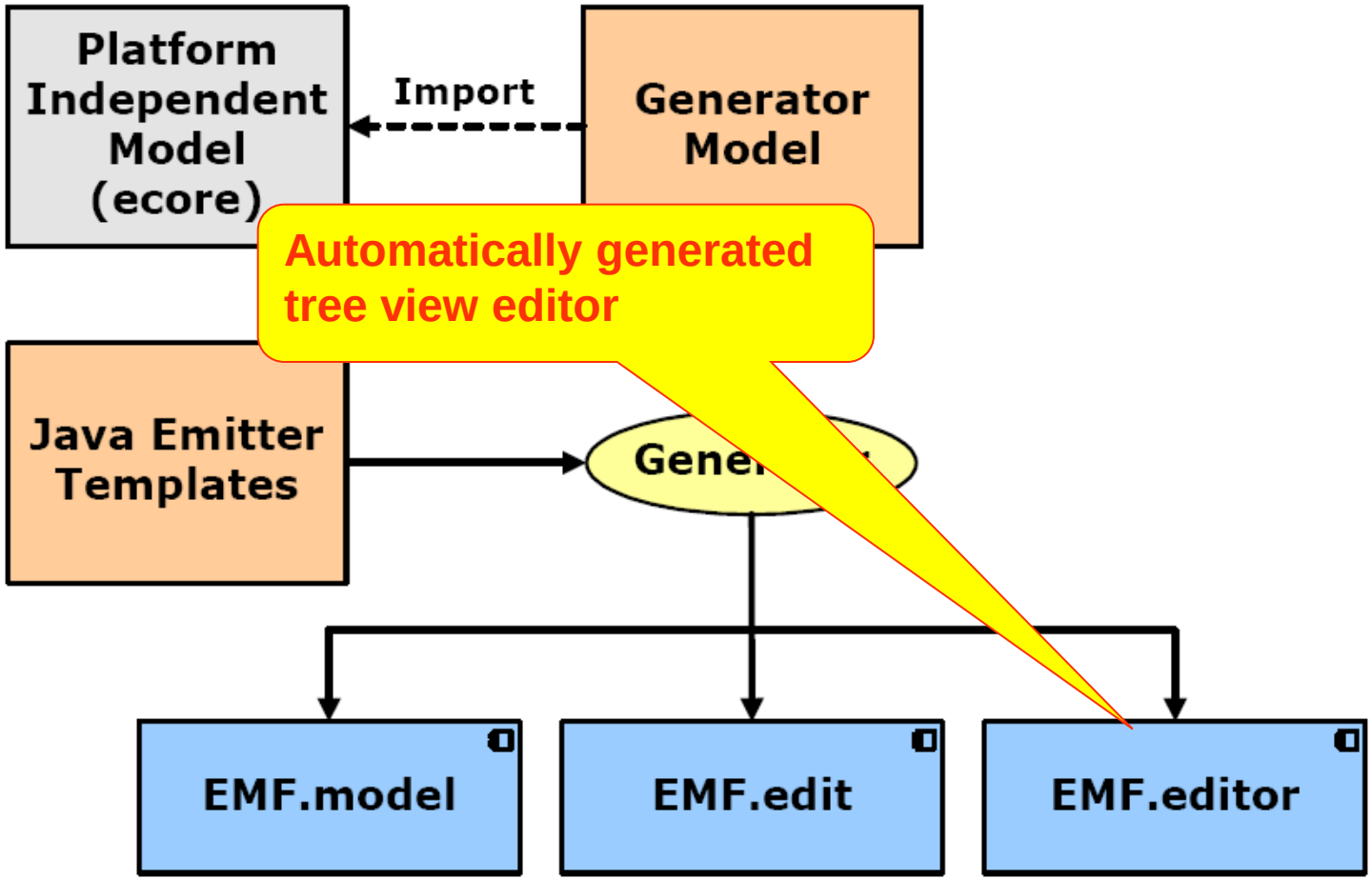
The EMF Toolkit



The EMF Toolkit

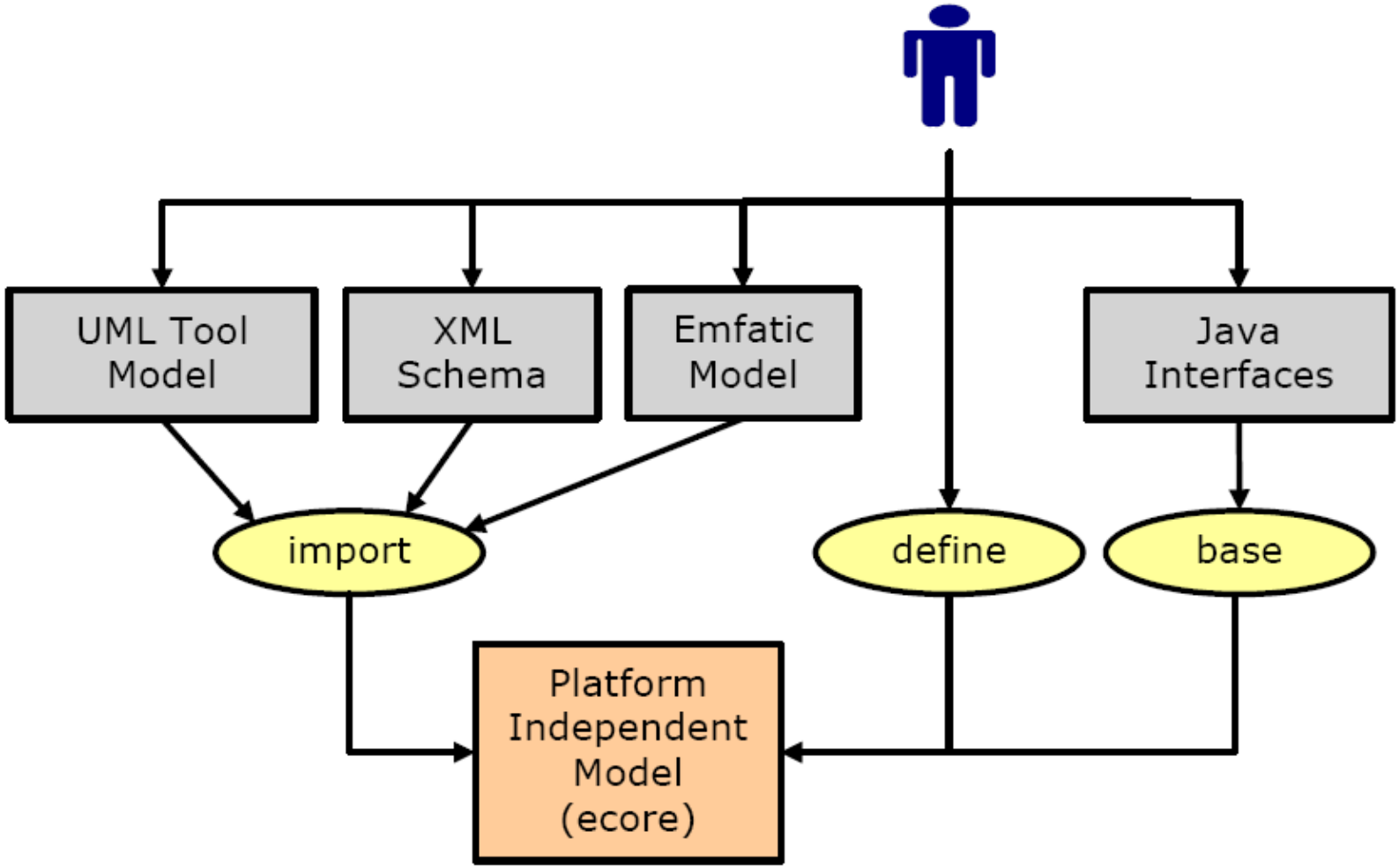


The EMF Toolkit

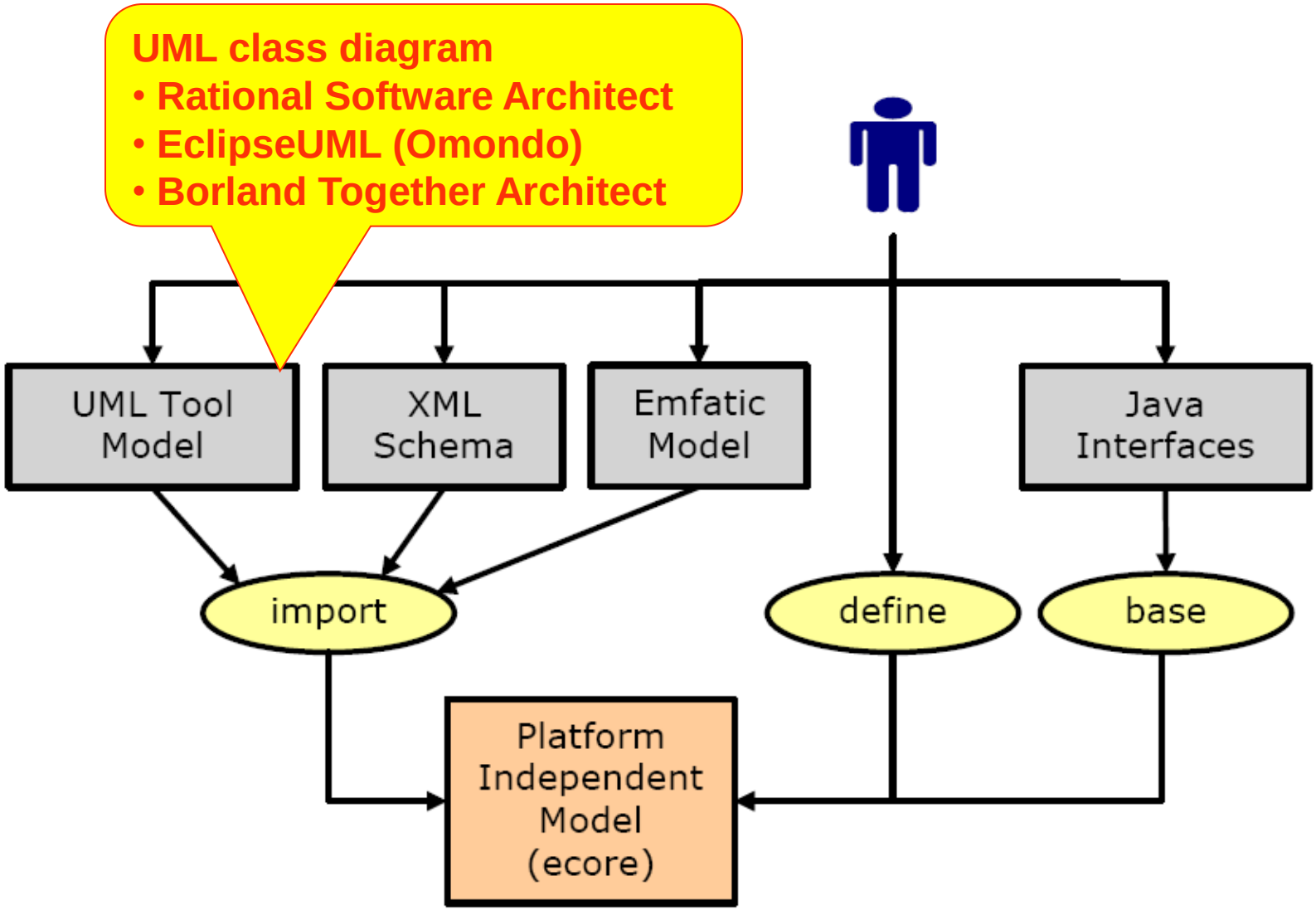


Creating an Ecore model (metamodel)

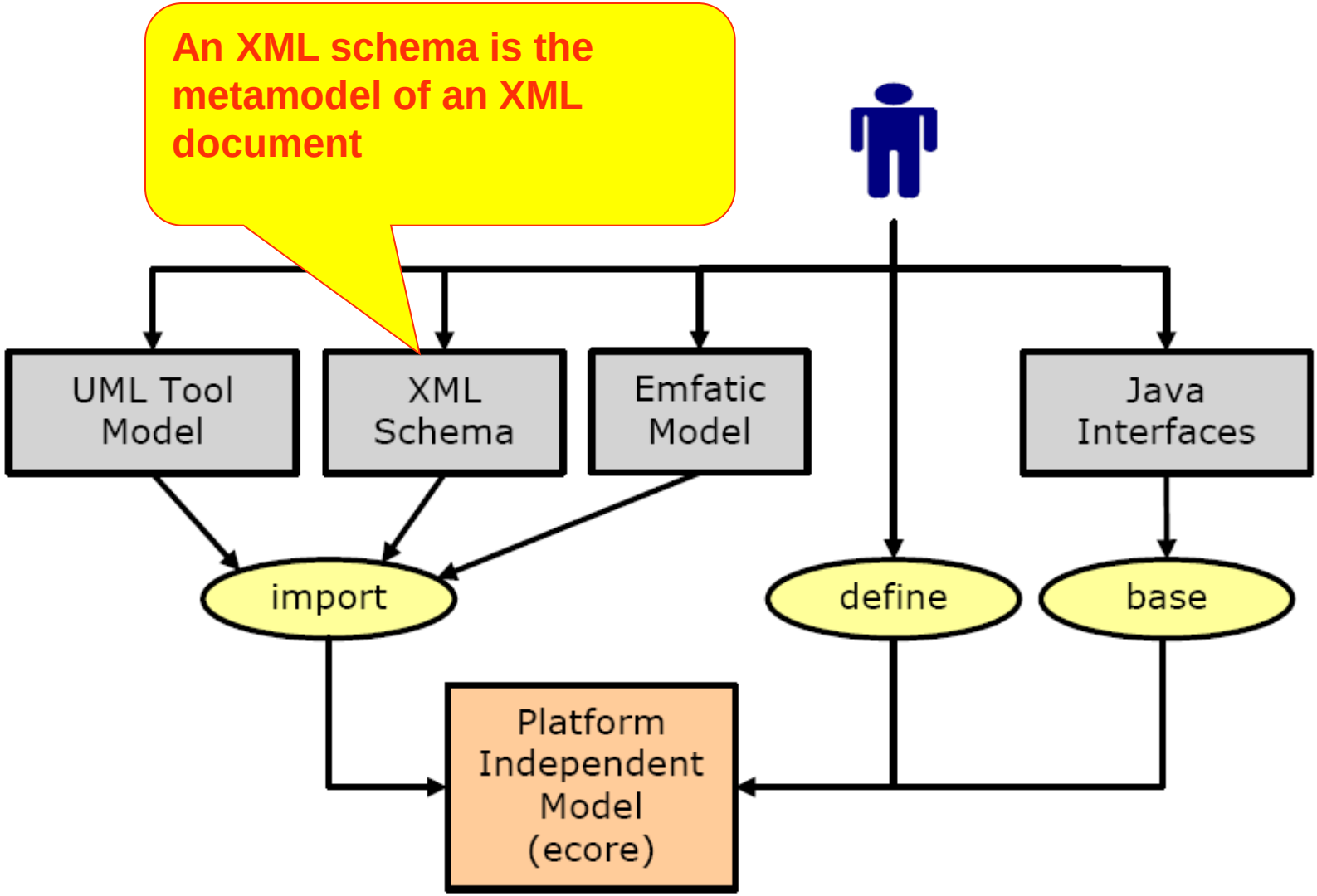
Creation of ECore models



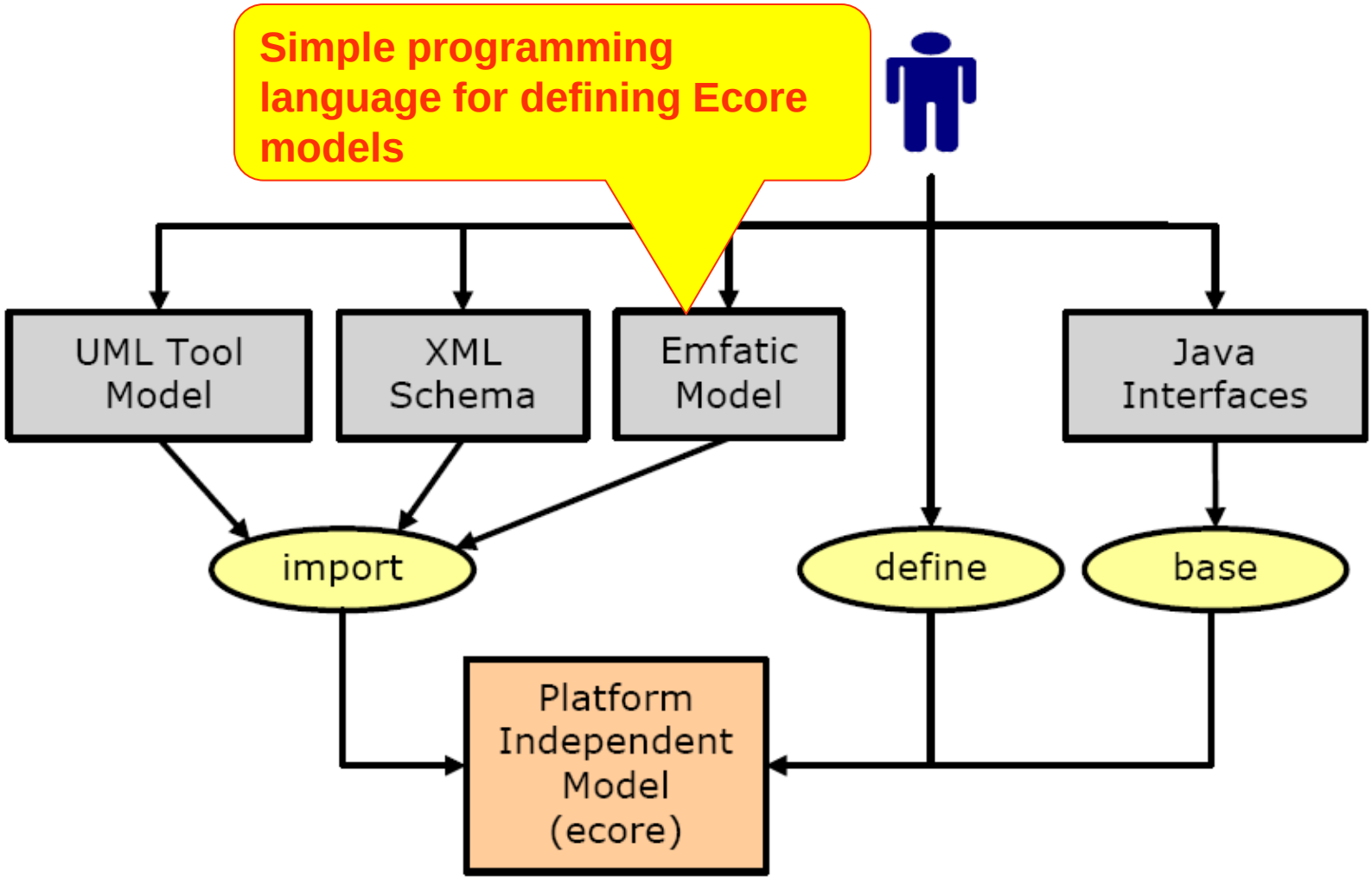
Creation of ECore models



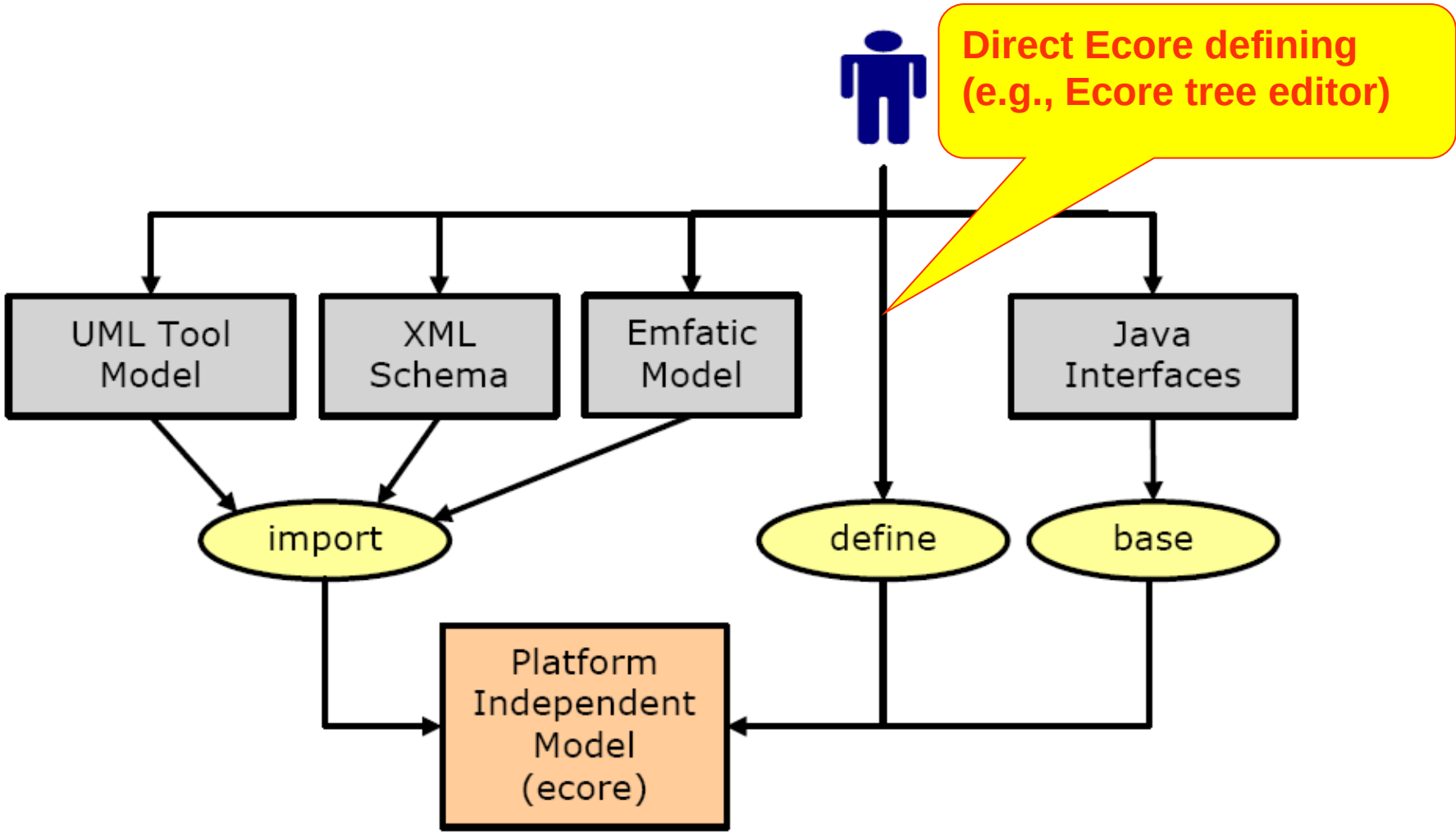
Creation of ECore models



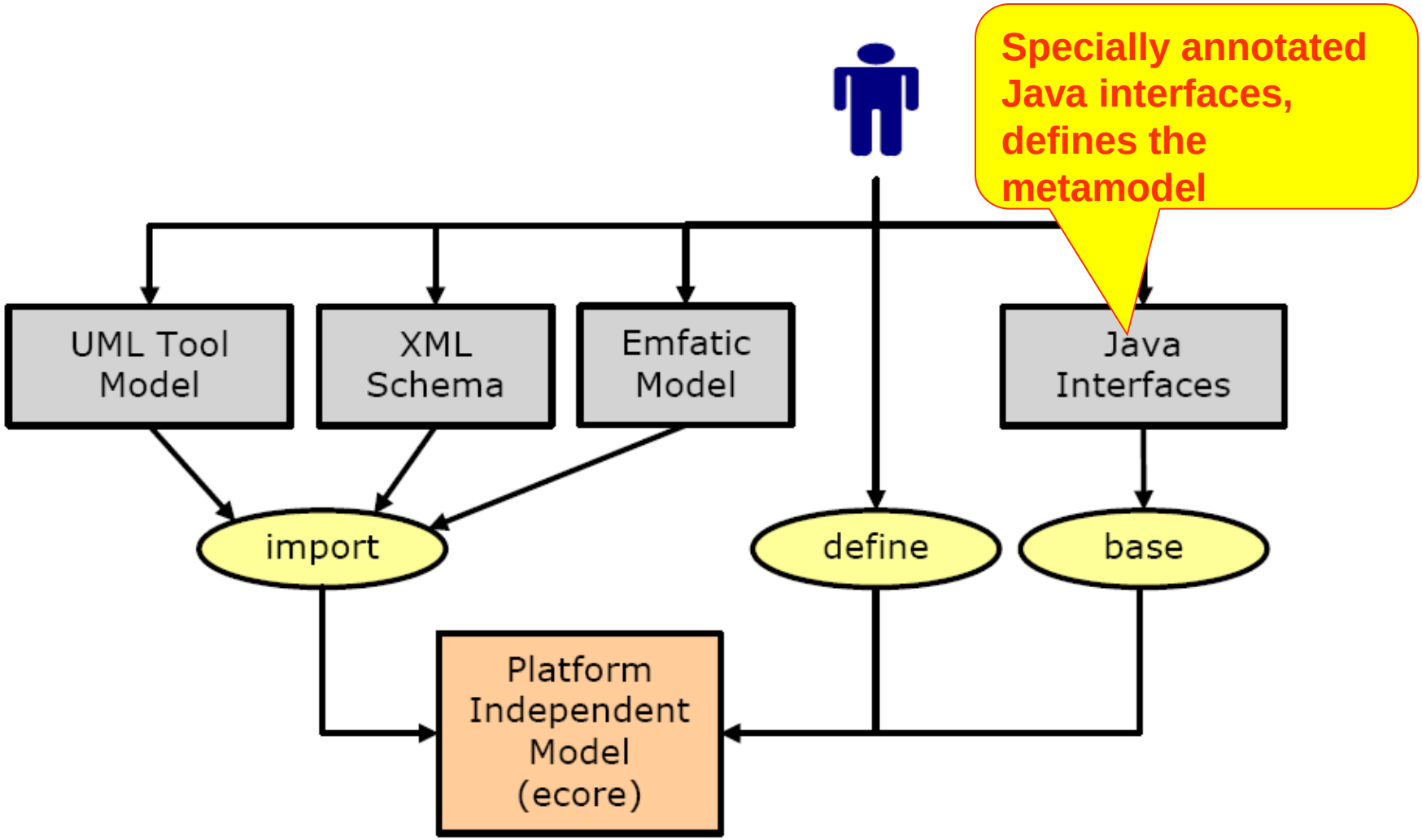
Creation of ECore models



Creation of ECore models

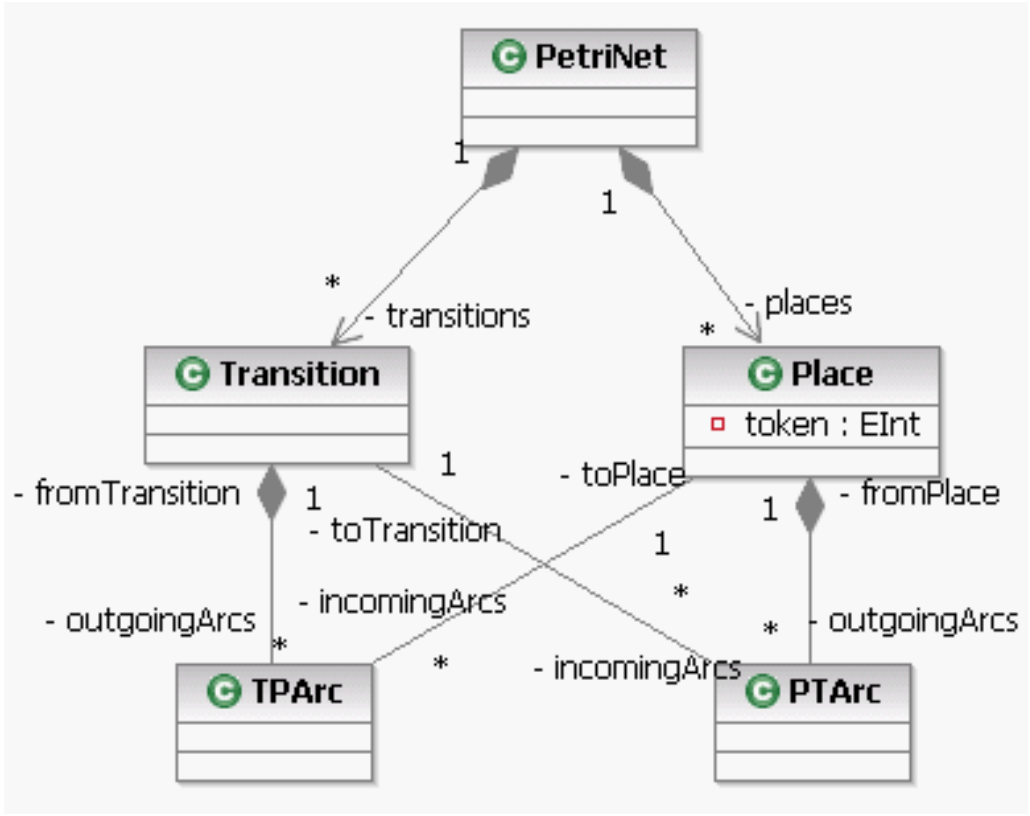


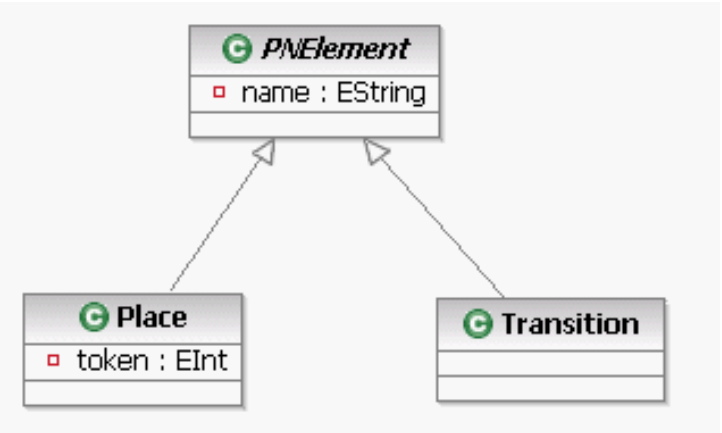
Creation of ECore models

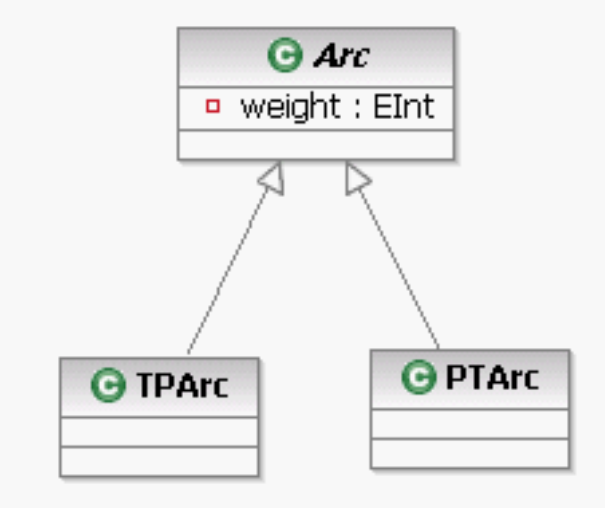


The Petri net Example

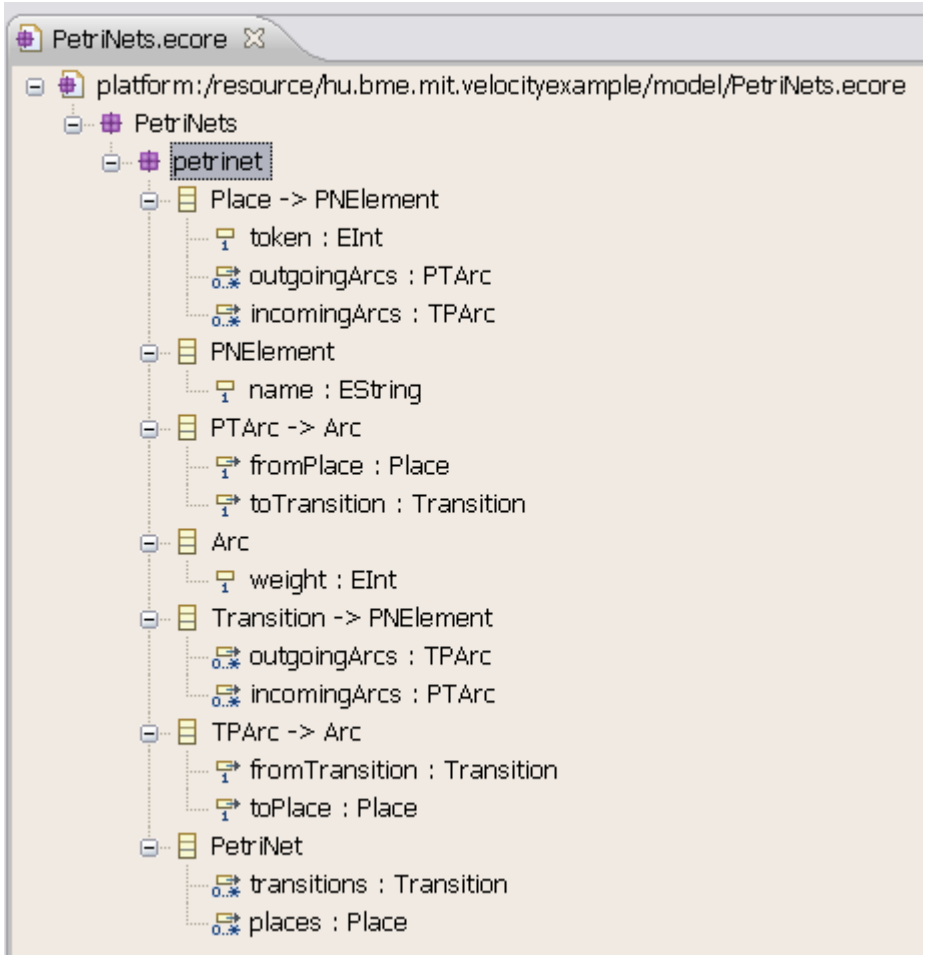
Domain Metamodel: Petri Nets



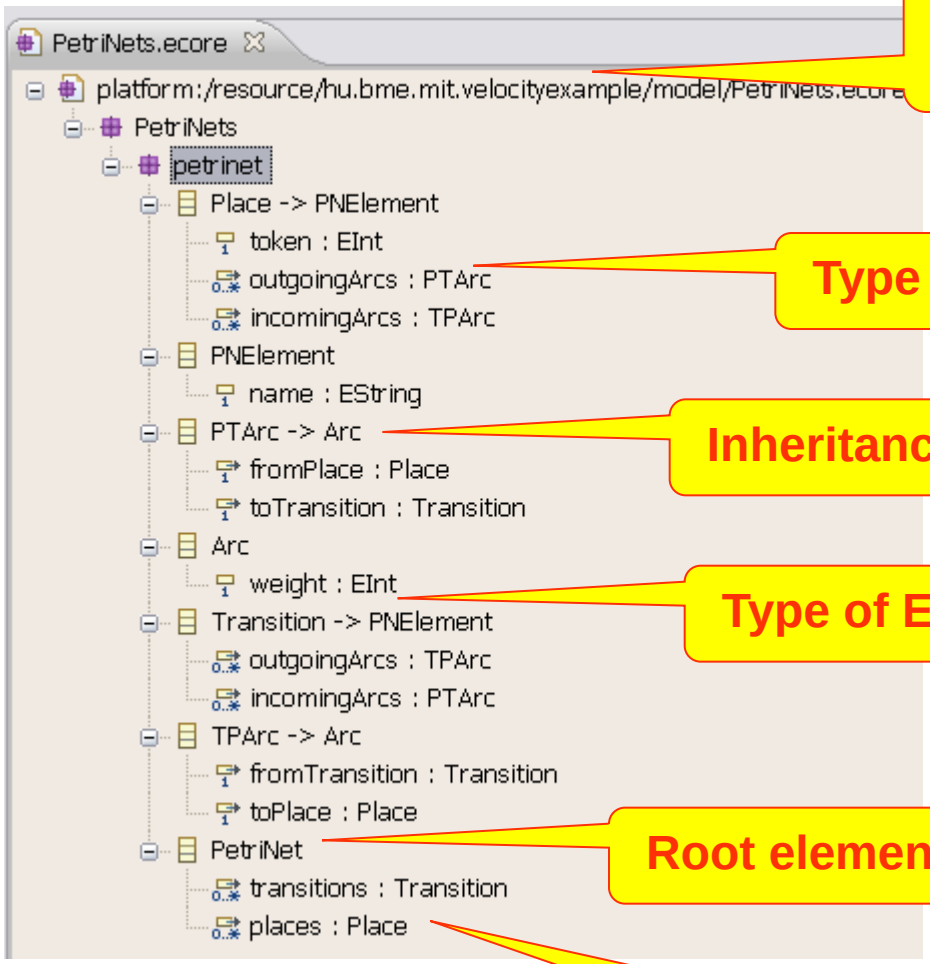




EMF model Ecore representation



EMF model Ecore representation



Path of containing resource

Type of EReference

Inheritance

Type of EAttribute

Root element

Reference to all model elements

Class Definition in PetriNet.ecore

```
<eClassifiers xsi:type="ecore:EClass" name="Place"
    eSuperTypes="#//petrinet/PNElement">

 <eStructuralFeatures xsi:type="ecore:EAttribute" name="token" lowerBound="1"
    eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>

 <eStructuralFeatures xsi:type="ecore:EReference" name="outgoingArcs"
    upperBound="-1"
    eType="#//petrinet/PTArc" containment="true"
    eOpposite="#//petrinet/PTArc/fromPlace"/>

 <eStructuralFeatures xsi:type="ecore:EReference" name="incomingArcs"
    upperBound="-1"
    eType="#//petrinet/TPArc" eOpposite="#//petrinet/TPArc/toPlace"/>
</eClassifiers>
```

Class Definition in PetriNet.ecore

```
<eClassifiers xsi:type="ecore:EClass" name="Place"
    eSuperTypes="#//petrinet/PNElement">

<eStructuralFeatures xsi:type="ecore:EAttribute" name="token" lowerBound="1"
    eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>

<eStructuralFeatures xsi:type="ecore:EReference" name="outgoingArcs"
    upperBound="-1"
    eType="#//petrinet/PTArc" containment="true"
    eOpposite="#//petrinet/PTArc/fromPlace"/>

<eStructuralFeatures xsi:type="ecore:EReference" name="incomingArcs"
    upperBound="-1"
    eType="#//petrinet/TPArc" eOpposite="#//petrinet/TPArc/toPlace"/>
</eClassifiers>
```

Class

Attribute

Reference

Containment

Multiplicity

Type

Opposite End

Code generation from Ecore

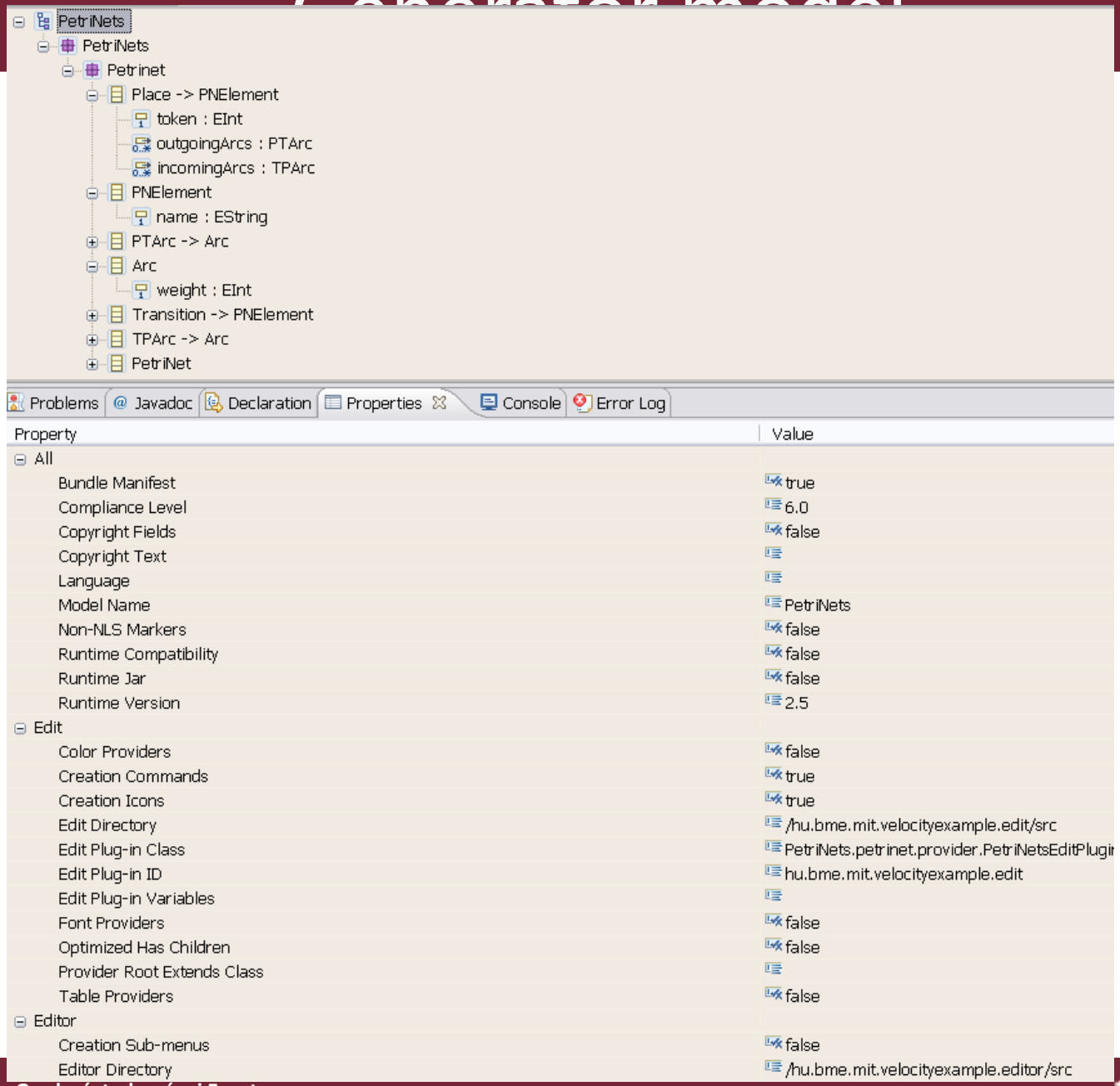
Code Generation from Ecore (.genmodel)

- ECore model remain pure and independent
- Customizable (wrappers, code formatters, etc.)
- Generated parts:
  - Model persistency (EMF.model)
  - Model management (EMF.edit)
  - Model editor (EMF.editor)
- EMF plugins
- Has some limitations
  - Simple tree editor
  - No concrete syntax

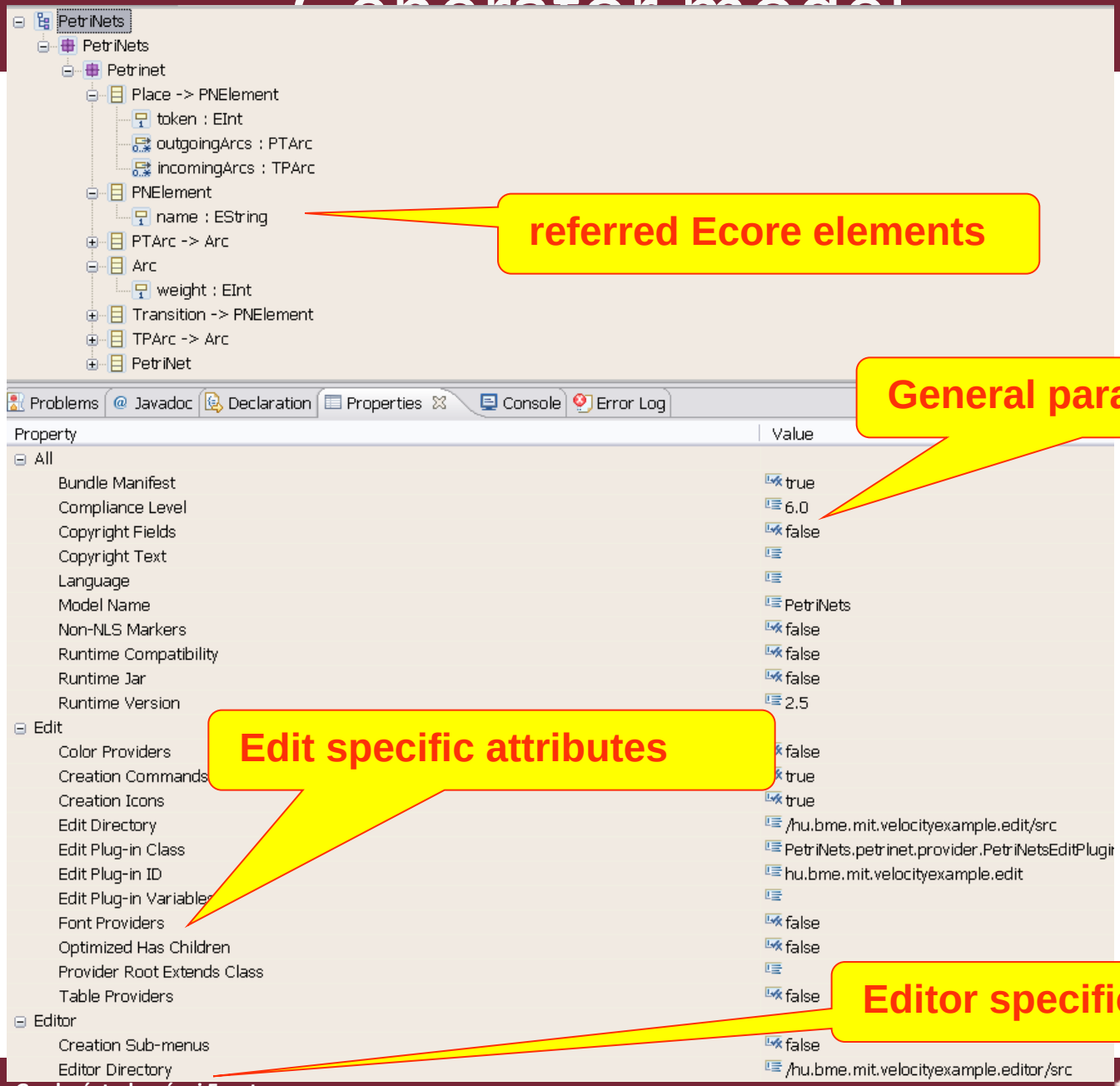
Generator model

- Goal:
  - Specify the attributes of the code generation
- EMF model
  - Tree Editor
  - Refers to the ecore model
- Code generation attributes
  - Java version (e.g., use Enums in case of Java 5 and higher)
  - Package/project names
  - …

Generator model



Generator model

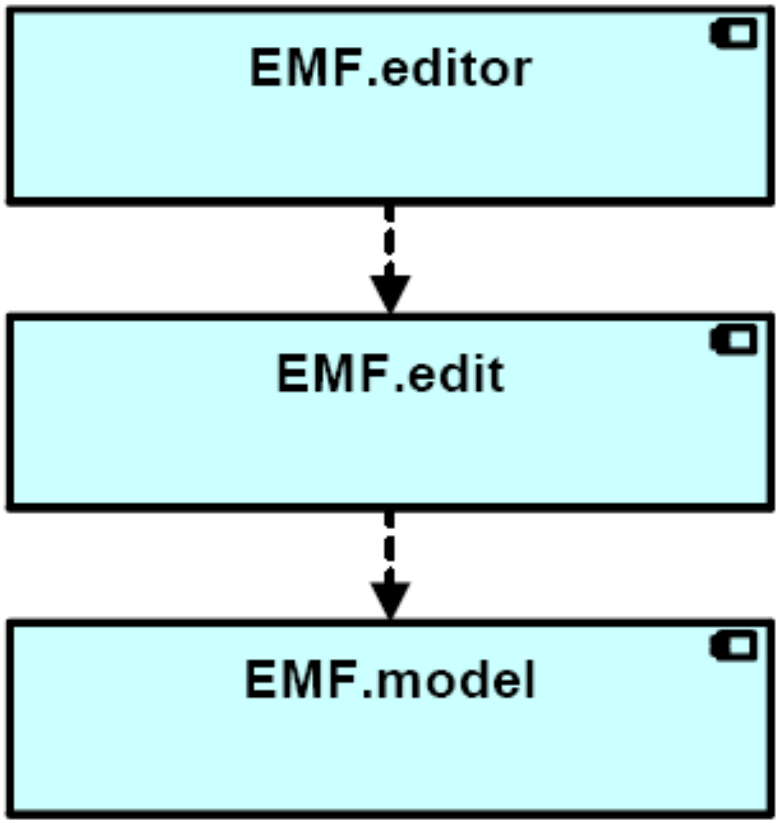


Generated EMF components

Automation in EMF

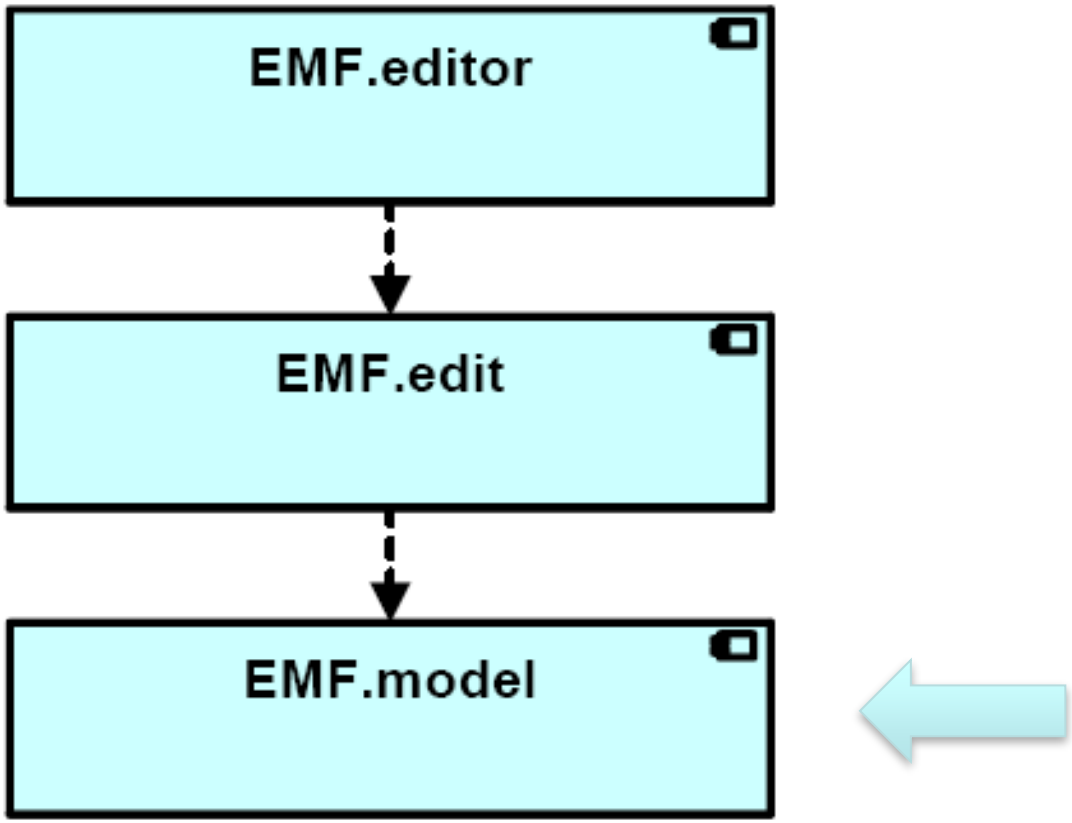
- Model manipulation
  - Bidirectional references (also m..n),
  - Containments
  - Enumerations
  - Factories
  - Multiple inheritance
- Notification, Tree Editors
- Schema Generation
- Core Validation + Unit Tests
- Example Client Usage
- Etc.

Generated EMF components



- 3. Tree Editor
- 2. Model Manipluation
- 1. Model Persistency

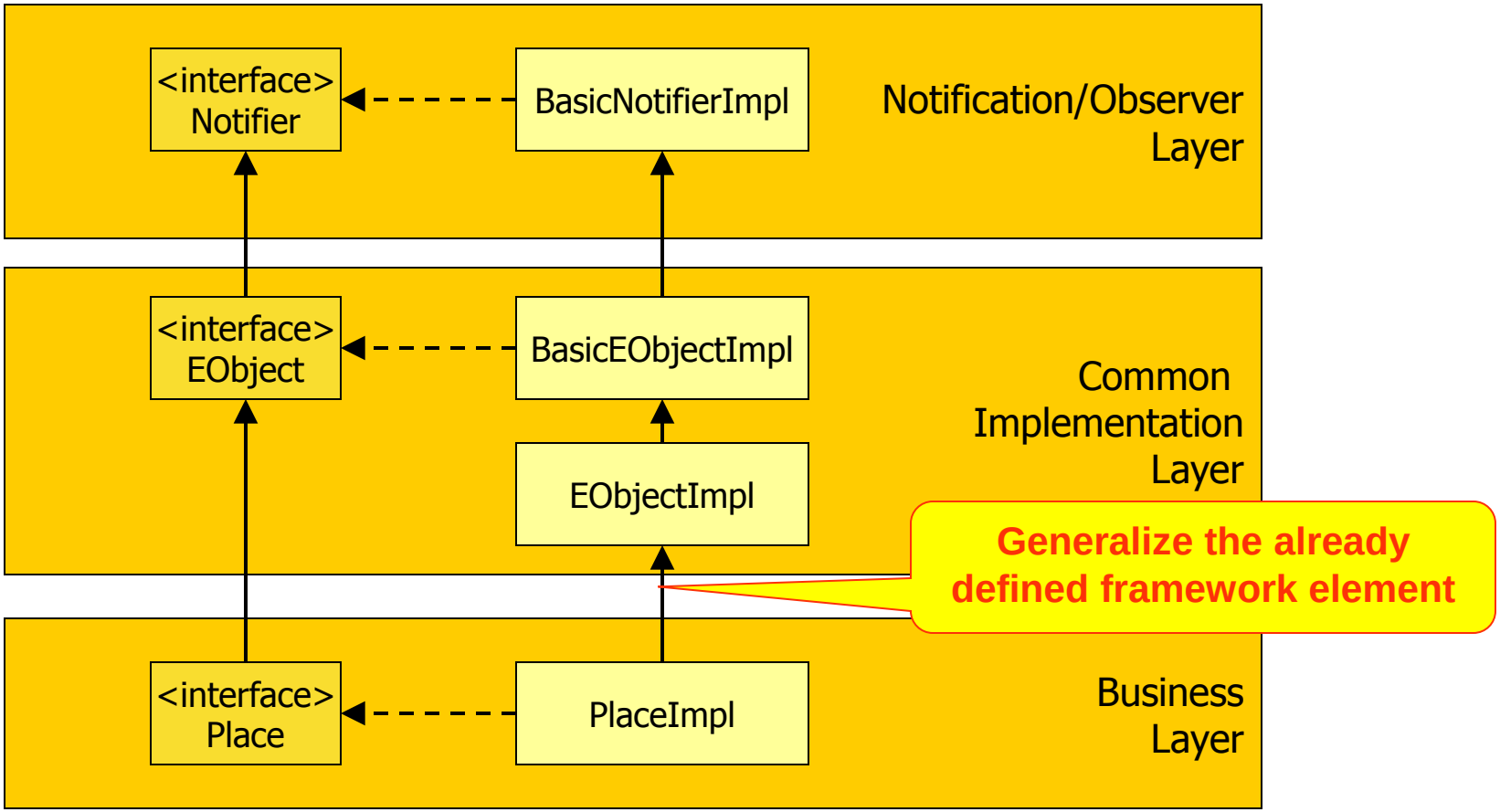
Generated EMF components



EMF.model

- Optimized persistency handling
- Fully featured Java code of the ECore model
- Specific factories for all packages
- Notification mechanism (observer pattern)
- Possible Extension points:
  - Advanced editor
  - Own file format with parser

EClass implementation



Auto-Generated Interface

```
 * @model
 * @generated
 */
public interface Place extends PNElement {
    /**
     * @model required="true"
     * @generated
     */
    int getToken();

    /**
     * @see #getToken()
     * @generated
     */
    void setToken(int value);

    /**
      * @model opposite="fromPlace" containment="true"
     * @generated
     */
    EList<PTArc> getOutgoingArcs();

    /**
     * @model opposite="toPlace"
     * @generated
     */
    EList<TPArc> getIncomingArcs();

} // Place
```

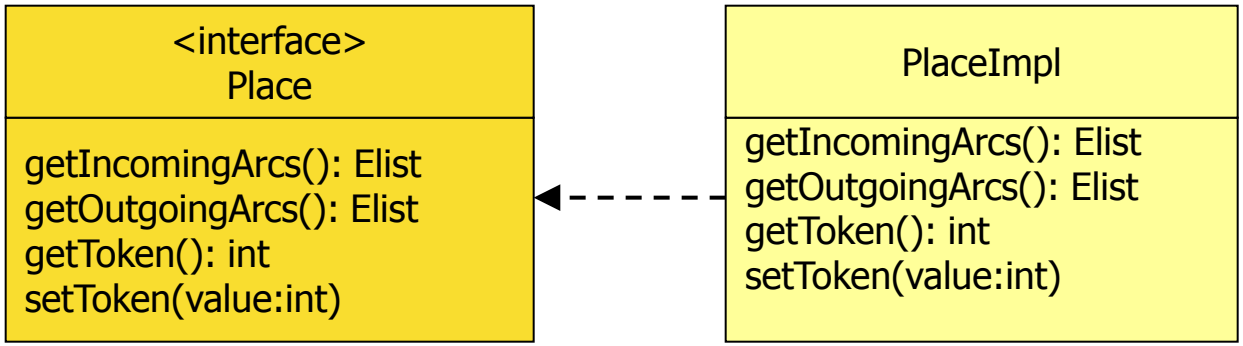
EMF specific „annotations”

Getters/Setters for attributes

Only getter when Multiplicity > 1

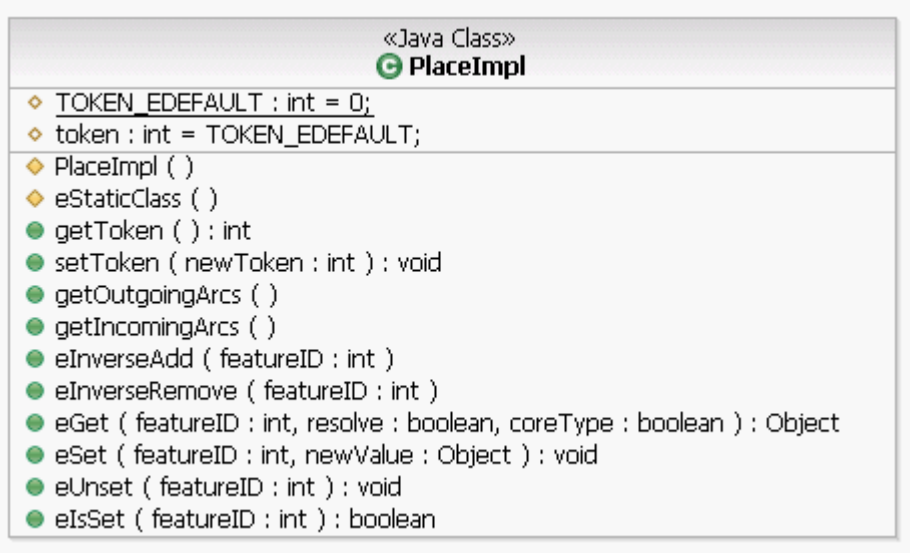
EList, EMF list interface (~10 implementations

EClass implementation



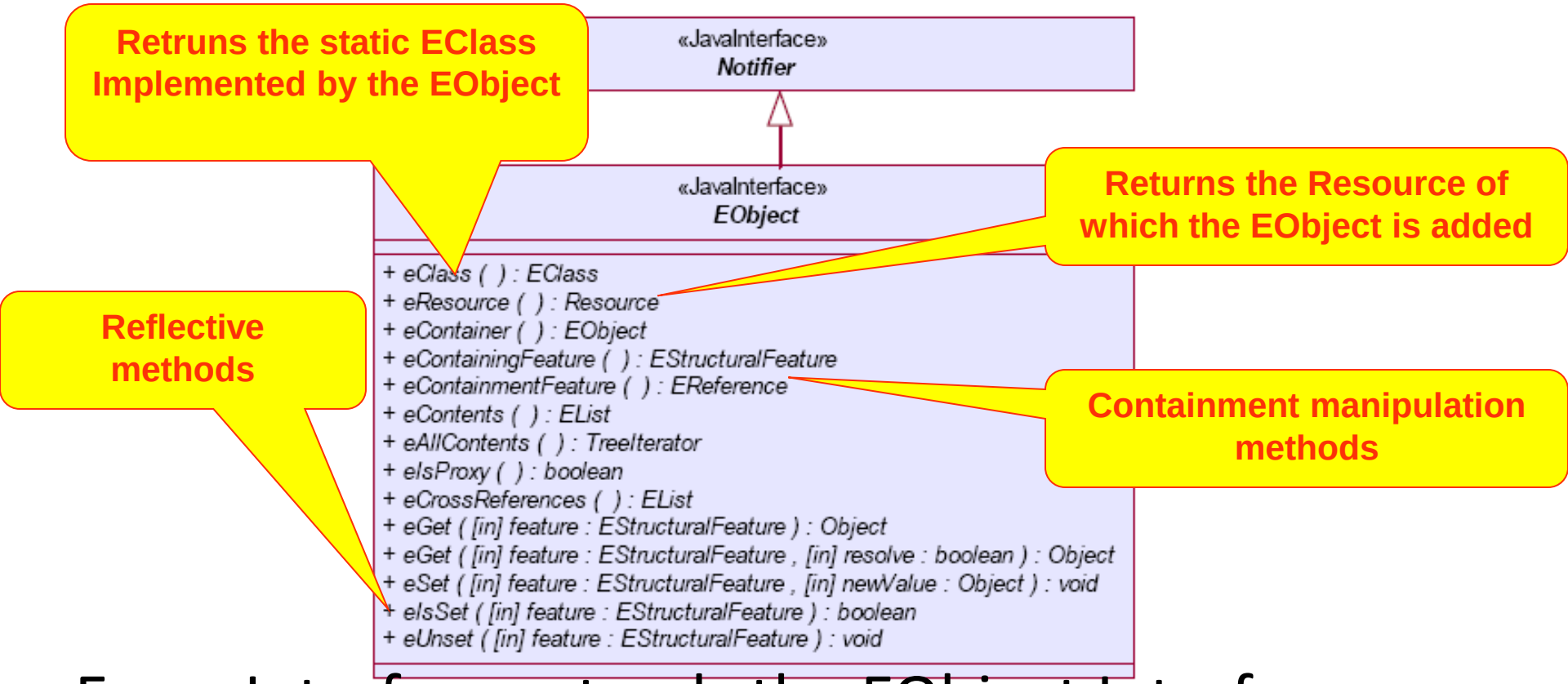
- EAttribute -> get/set methods (multiplicity <= 1)
- EReference:
  - „many” -> get
  - „one” -> get / set

EClass implementation



- Every Class contains Framework specific methods:
  - Reflective get/set (eGet, eSet)
  - Consistency manipulation (eInverseRemove)

EObject



- Every Interface extends the EObject Interface

EAttribute implementation

```
protected static final int TOKEN_EDEFAULT = 0;
/**
 * @generated
 */
public int getToken() {
    return token;
}
/**
 * @generated
 * @ordered
 */
protected int token = TOKEN_EDEFAULT;
/**
 * @generated
 */
public void setToken(int newToken) {
    int oldToken = token;
    token = newToken;
    if (eNotificationRequired())
        eNotify(new ENotificationImpl(this, Notification.SET,
                PetrinetPackage.PLACE__TOKEN, oldToken, token));
}
```

- Attribute is mapped to get and set methods
- Notification
- Type is normally a simple Java type (int, String, etc.)

EReference implementation

- Similar to the EAttribute
- Notification thrown by EList
- Type is normally an other EObject
- Handles containmnet
  - Uses proxy, must be resolved
  - Have to check opposite EReference integrity

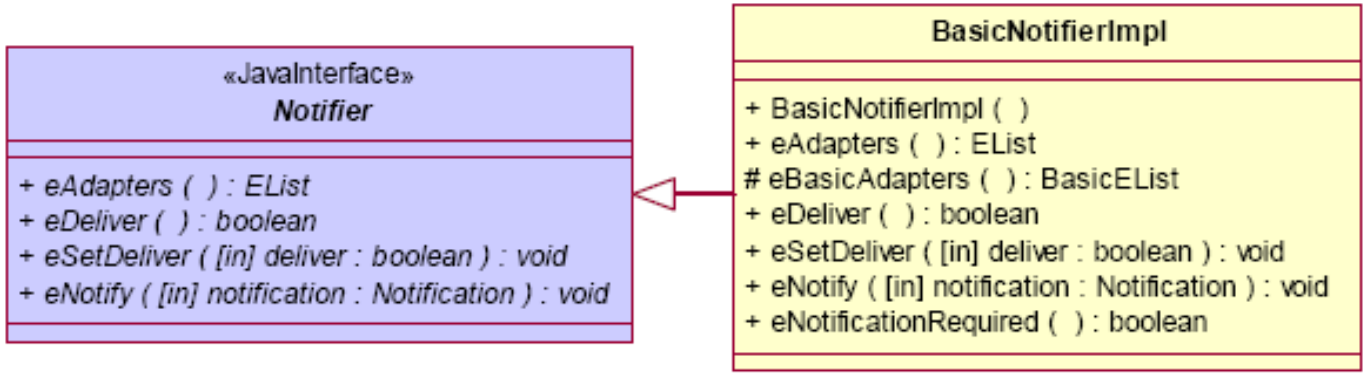
```
/**
 * @see #getIncomingArcs()
 * @generated
 * @ordered
 */
protected EList incomingArcs = null;
/**
 * @generated
 */
public EList getIncomingArcs() {
    if (incomingArcs == null) {
        incomingArcs = new EObjectWithInverseResolvingEList(TPArc.class,
                this,
                PetrinetPackage.PLACE__INCOMING_ARCS,
                PetrinetPackage.TP_ARC__TO_PLACE);
    }
    return incomingArcs;
}
```

EOperation Implementation

```
public class XImpl extends EObjectImpl implements X {

    /**
     * @generated NOT
     */
    void f() {
      // Provide the implementation
    }
}
```

- Represents the frame of a Java method
- Represented both in the interface and implementing class
- Important:
  - Have to change the generated annotation to NOT
  - Have to implement the method manually

Notification implementation



- Observer pattern
- Behavior extension
- Events stored in a Notification class
- Can be parametrized in the genmodel

EFactory implementation

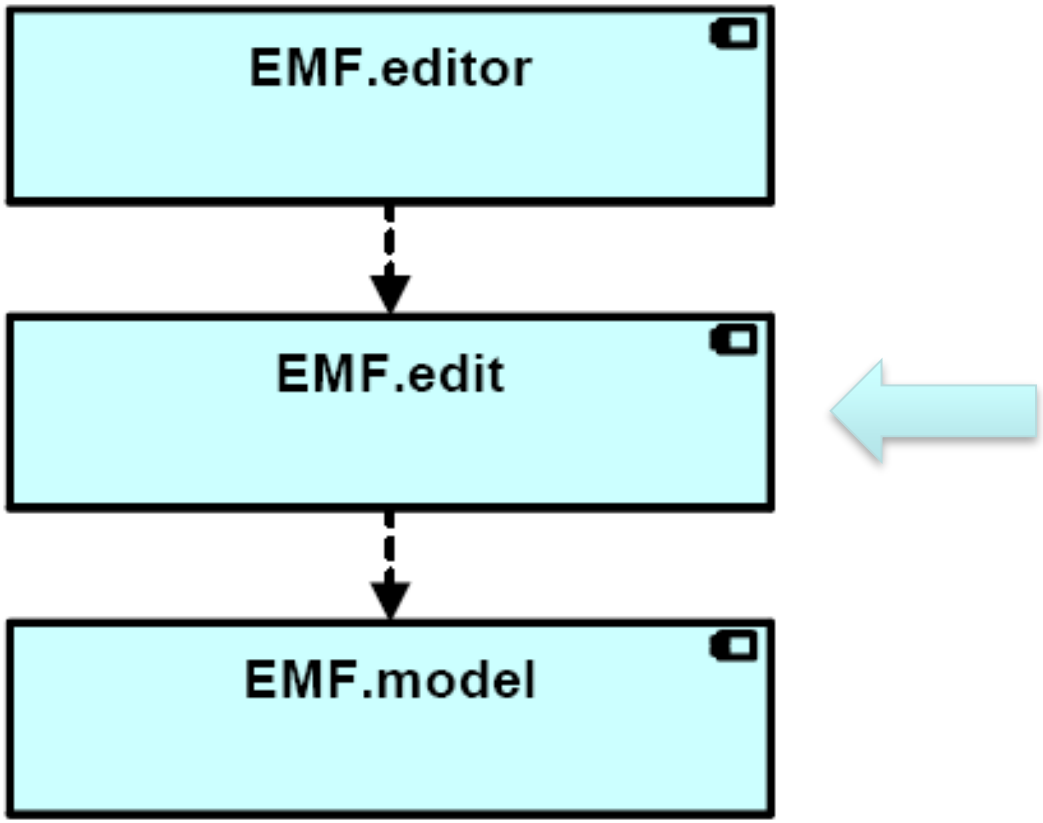
```
public interface PetrinetFactory extends EFactory {
    /**
     * The singleton instance of the factory.
     * @generated
     */
    PetrinetFactory eINSTANCE = PetriNets.petrinet.impl.PetrinetFactoryImpl.init();

    Place createPlace();

    PTArc createPTArc();

    Transition createTransition();

    TPArc createTPArc();


    PetriNet createPetriNet();

    PetrinetPackage getPetrinetPackage();

} //PetrinetFactory
```

Single instance

Create methods by type

One-to-one reference to Petrinet EPackage

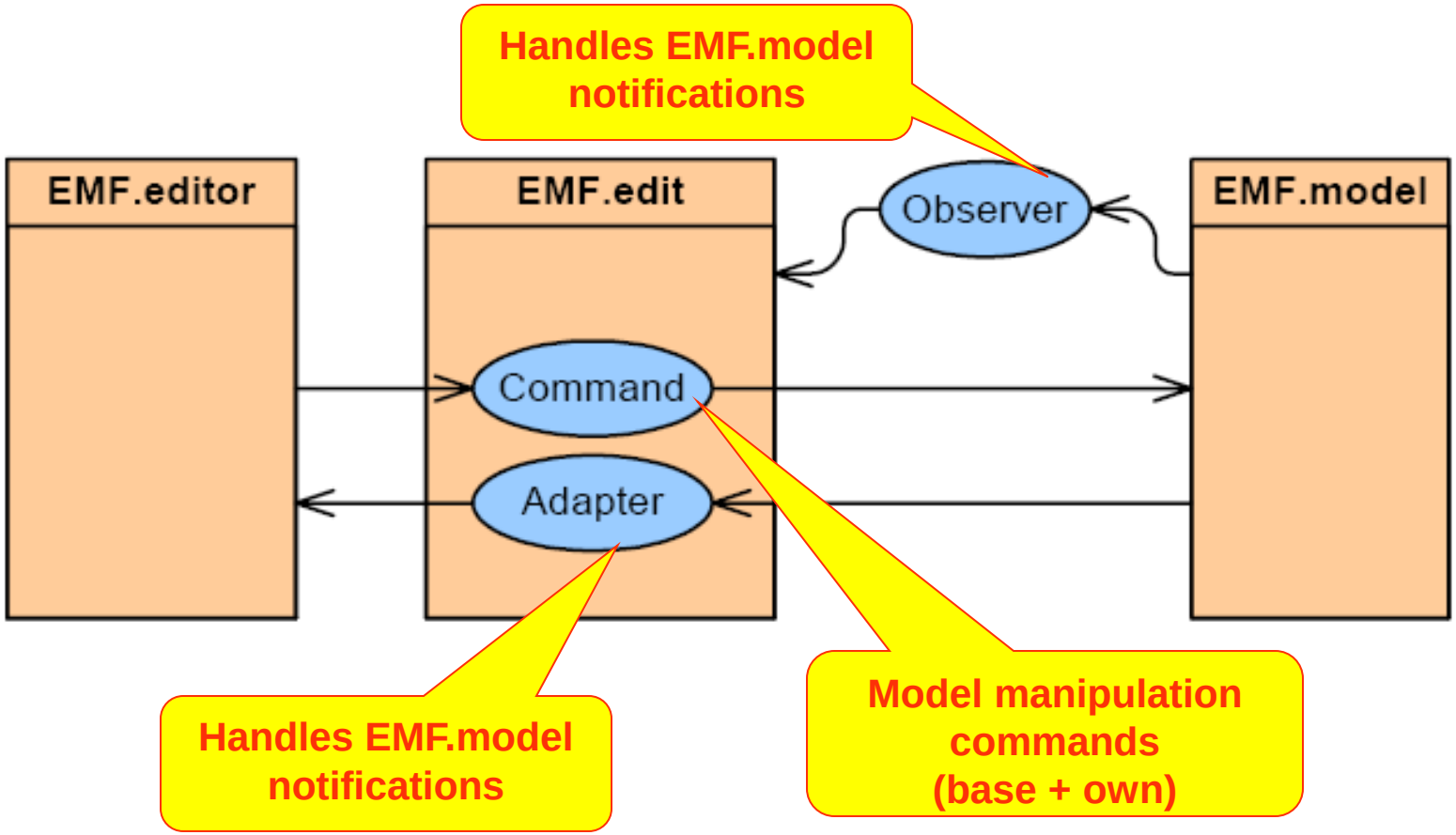
Generated EMF components



EMF.Edit

- Separates the GUI and the model
- GUI independent commands
- Generator pattern:
  - Provider class for each model element
  - Base class: ItemProvider
  - Forward EMF model change notifications to the viewer
- Captures core provide utilities
- Flexible

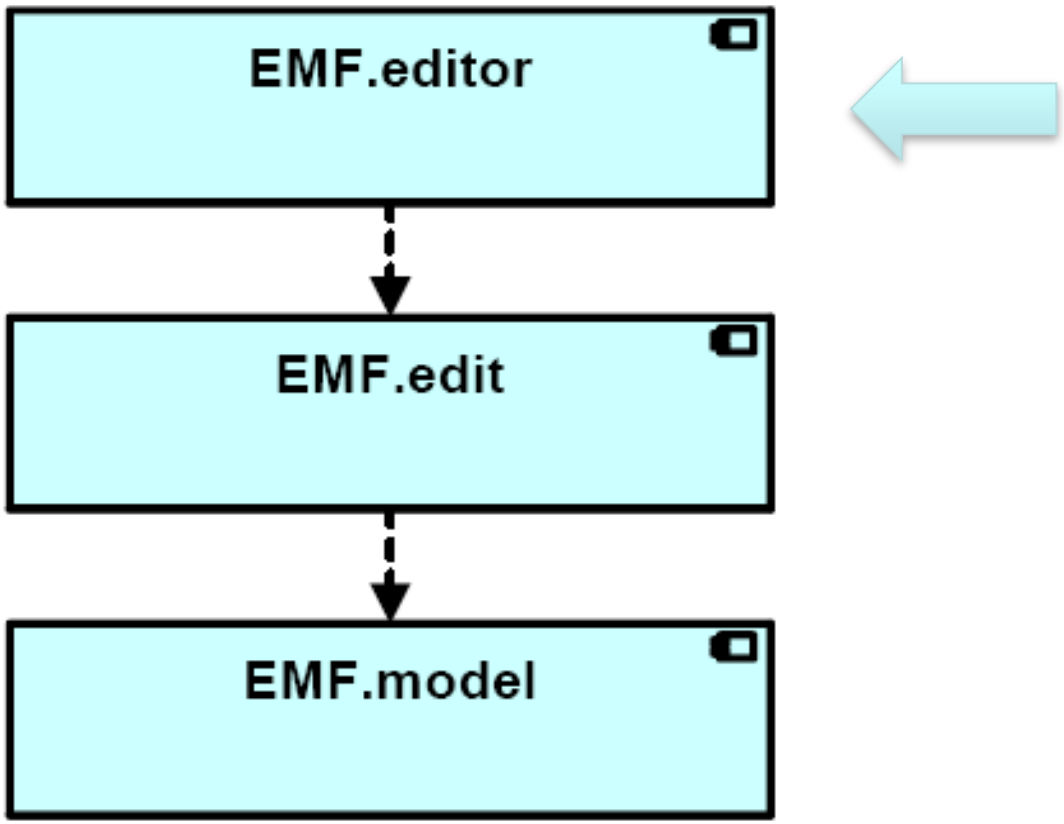
EMF.Edit


Handles EMF.model notifications
EMF.editor
EMF.edit
EMF.model
Observer
Command
Adapter
Handles EMF.model notifications
Model manipulation commands (base + own)

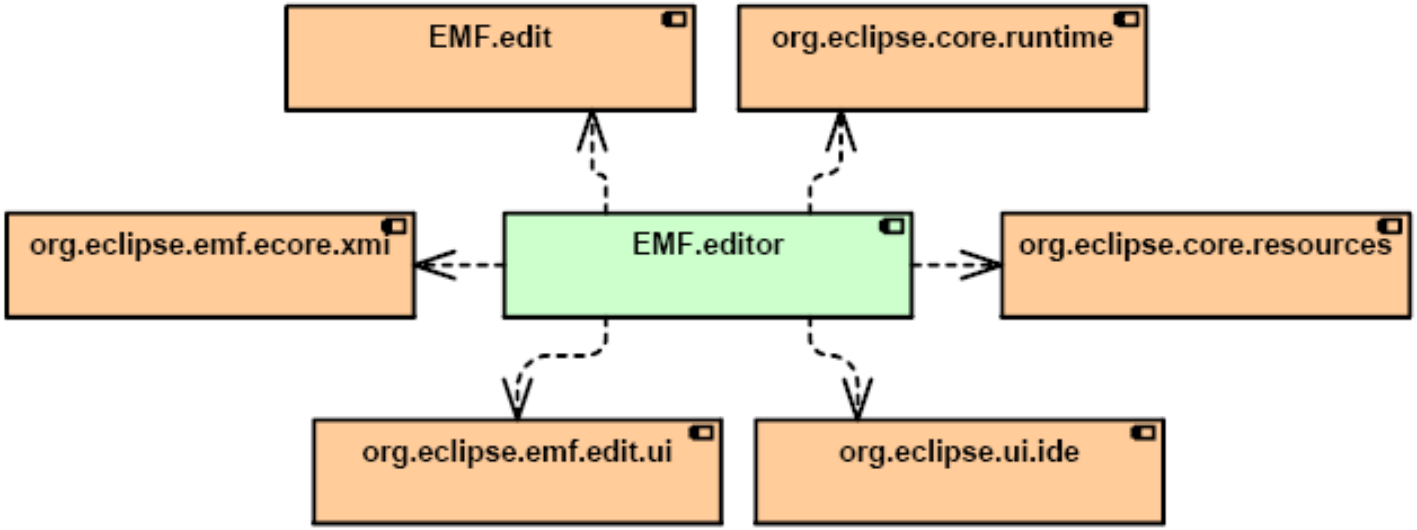
EMF.Edit Command

- All model manipulation through commands
- Based on template method pattern
- The ItemProvider implements the createCommand(…) method, which calls one of its protected command method
- Customizable (usually, modify already implemented „protected” commands)

Generated EMF components

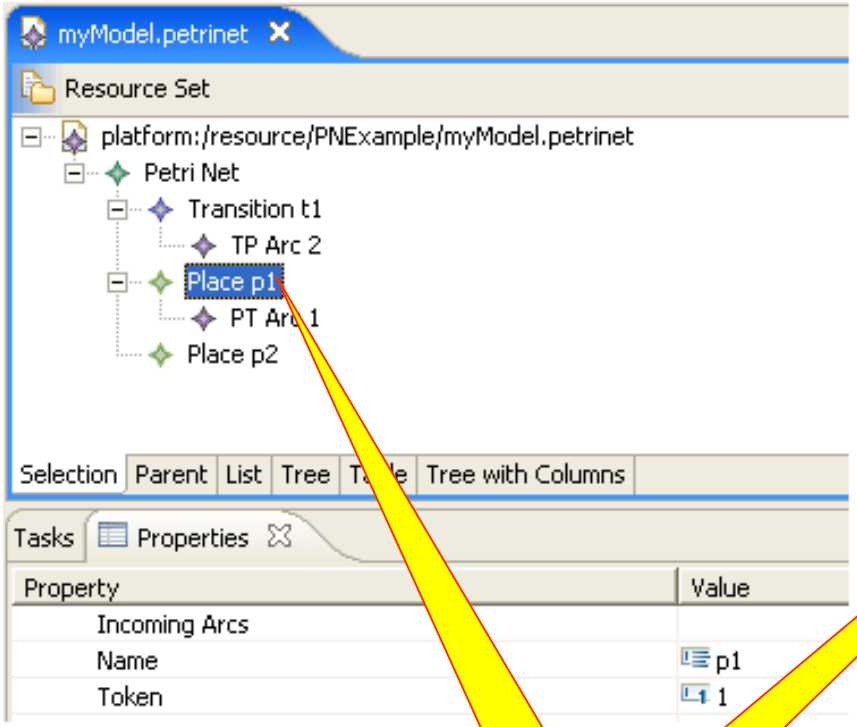


EMF.Editor



- EMF.Editor generates the SWT/JFace for the graphical editor
- Generates:
  - Tree editor
  - Wizards
  - Menus
  - plugins

The editor of Petri Net models




```
<?xml version="1.0" encoding="UTF-8"?>
<PetriNets.petrinet:PetriNet xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:PetriNets.petrinet=
    "http:///PetriNets/petrinet.ecore">
 <transitions name="t1" incomingArcs=
    "//@places.0/@outgoingArcs.0">
  <outgoingArcs weight="2"
    toPlace="//@places.1"/>
 </transitions>
 <places name="p1" token="1">
  <outgoingArcs weight="1"
    toTransition="//@transitions.0"/>
 </places>
 <places name="p2" incomingArcs=
    "//@transitions.0/@outgoingArcs.0"/>
</PetriNets.petrinet:PetriNet>
```

Place p1

Reference: URI (or XMI.id)

Tree View

XMI 2.0 View

Tools, API and Utilities

The org.eclipse.emf.ecore.util Package

- Contains a number of miscellaneous utility classes and interfaces:
  - *ECoreEContentAdapter*: maintains itself as an adapter for all contained objects (add,remove)
  - *UsageCrossReferencer*: finds each ModelElement pointing to the corresponding EObject
  - *ContentTreeIterator*: An iterator over the tree contents of a collection of EObjects
  - *Copier*: deep copy of EObject Elements and EReferences
  - Etc.

Client Programming with EMF

```
Place p1 = PetrinetFactory.eINSTANCE.createPlace();
p1.setName("p1");
Place p2 = PetrinetFactory.eINSTANCE.createPlace();
p2.setName("p2");
Transition t1 = PetrinetFactory.eINSTANCE.createTransition();
t1.setName("t1");

// Inverse direction (p1.outgoingArcs) is set automatically
PTArc a0 = PetrinetFactory.eINSTANCE.createPTArc();
a0.setFromPlace(p1);
a0.setToTransition(t1);
TPArc a1 = PetrinetFactory.eINSTANCE.createTPArc();
a1.setToPlace(p2);
a1.setFromTransition(t1);
```

Create a place

Create a transition

Create a PT arc

Set source of PT arc

Set target of PT arc

Advanced client programming: Reflective Ecore API

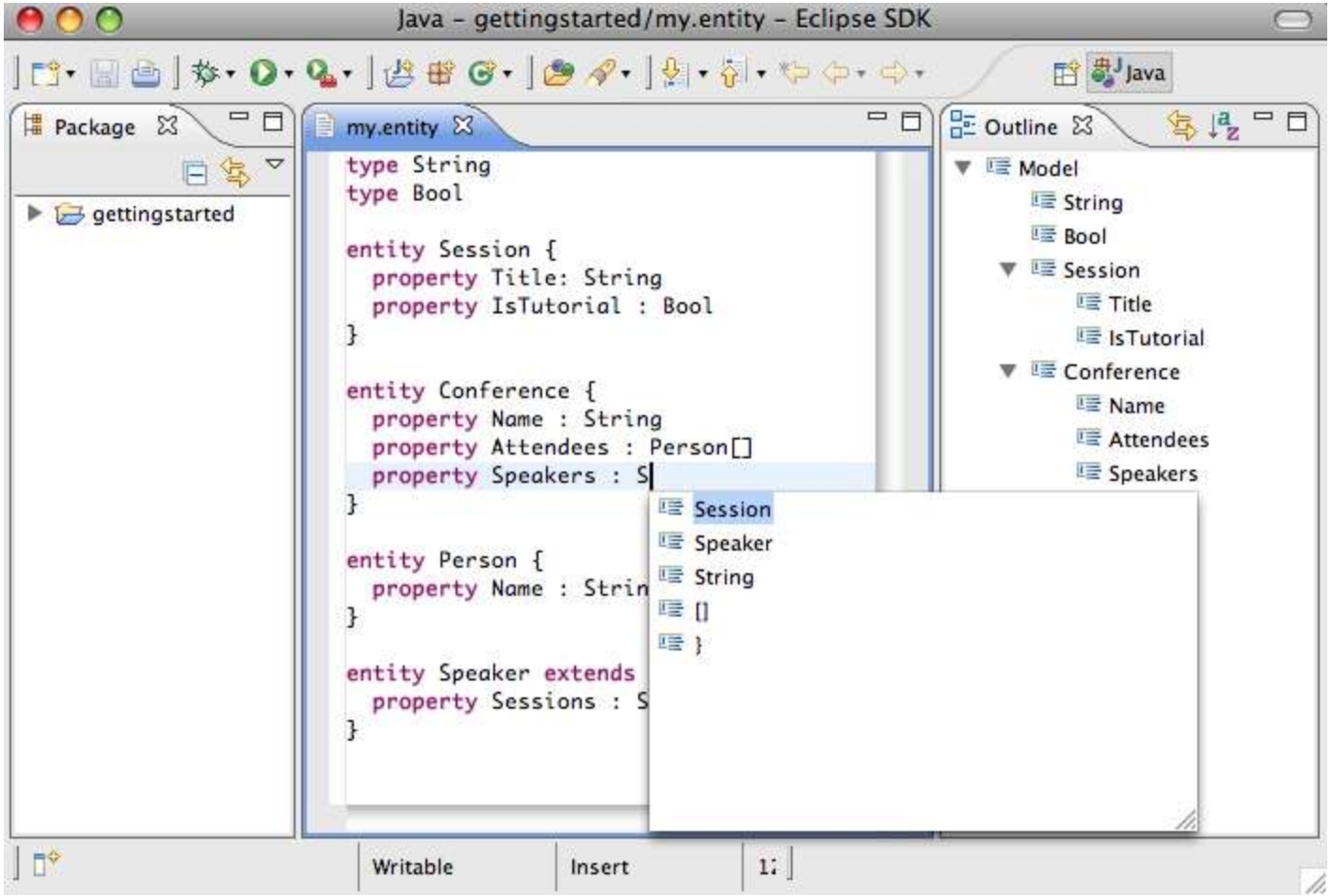
Basic EMF tools

- Validation
  - Validate constraints over EMF models
- Query
  - High-level query language for EMF
- Compare
  - To structurally compare EMF models (e.g., versioning)

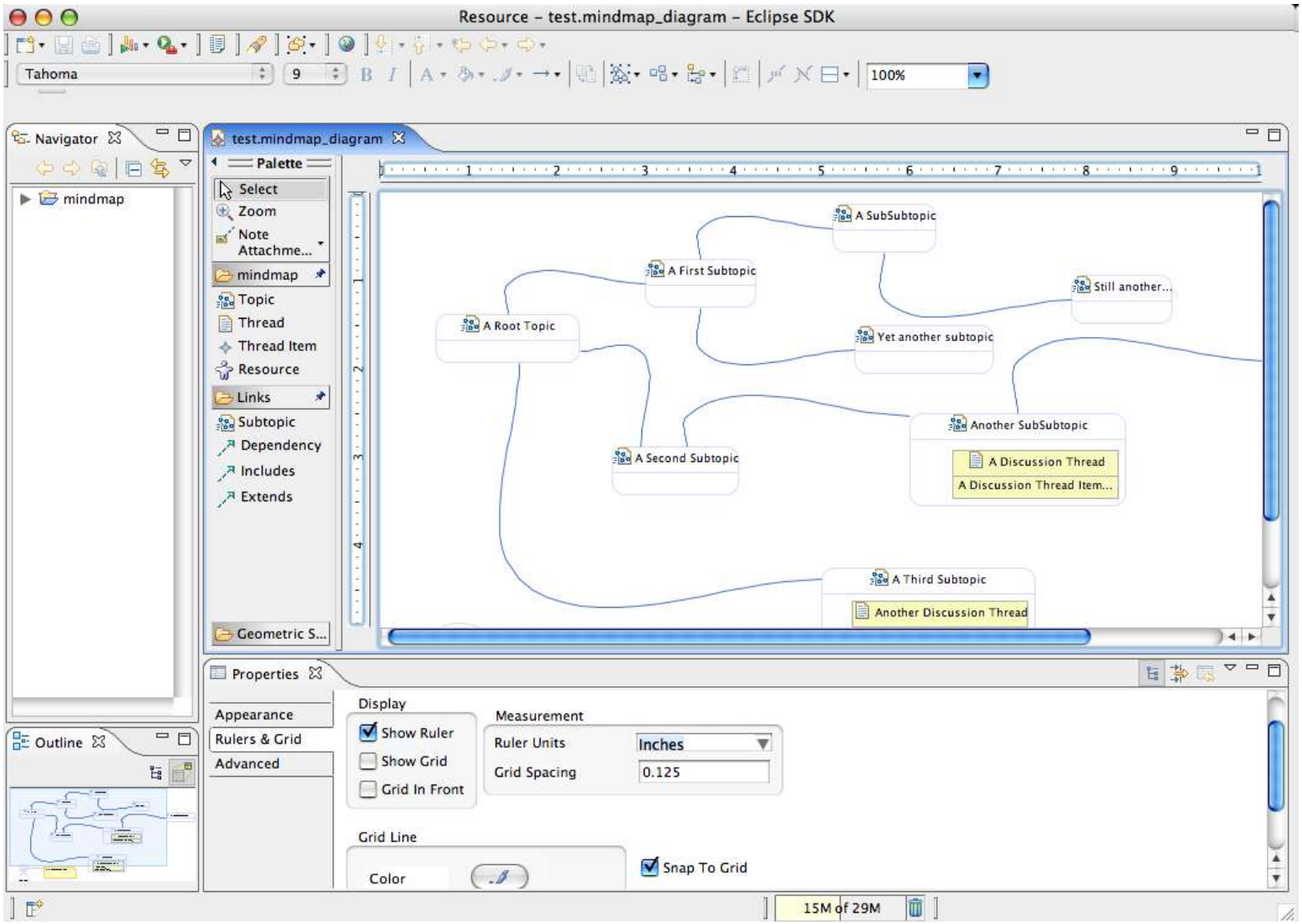
- Teneo
  - Persistency layer over relation databases
- SDO
  - Service Oriented Architecture based on EMF
- CDO
  - distributed, client-server EMF models

XText

- Textual DSL EMF for metamodels



GMF


Resource – test.mindmap_diagram – Eclipse SDK
Tahoma
9
100%
Navigator
mindmap
test.mindmap_diagram
Palette
Select
Zoom
Note Attachme...
mindmap
Topic
Thread
Thread Item
Resource
Links
Subtopic
Dependency
Includes
Extends
Geometric S...
A SubSubtopic
A First Subtopic
Still another...
A Root Topic
Yet another subtopic
Another SubSubtopic
A Discussion Thread
A Discussion Thread Item...
A Second Subtopic
A Third Subtopic
Another Discussion Thread
Outline
Properties
Appearance
Rulers & Grid
Advanced
Display
Show Ruler
Show Grid
Grid In Front
Measurement
Ruler Units
Inches
Grid Spacing
0.125
Grid Line
Color
Snap To Grid
15M of 29M

Ecore Tools: Ecore Diagram Editor (GMF)

- Graphical DSL EMF to define EMF metamodels