



# **Model-Based Engineering and Code Generation**

## Application to BPMN

**F. Mallet**

Frederic.Mallet@unice.fr

Université Nice Sophia Antipolis

# Introduction

## □ Outline

- Meta-Modeling & EMF

## □ Application

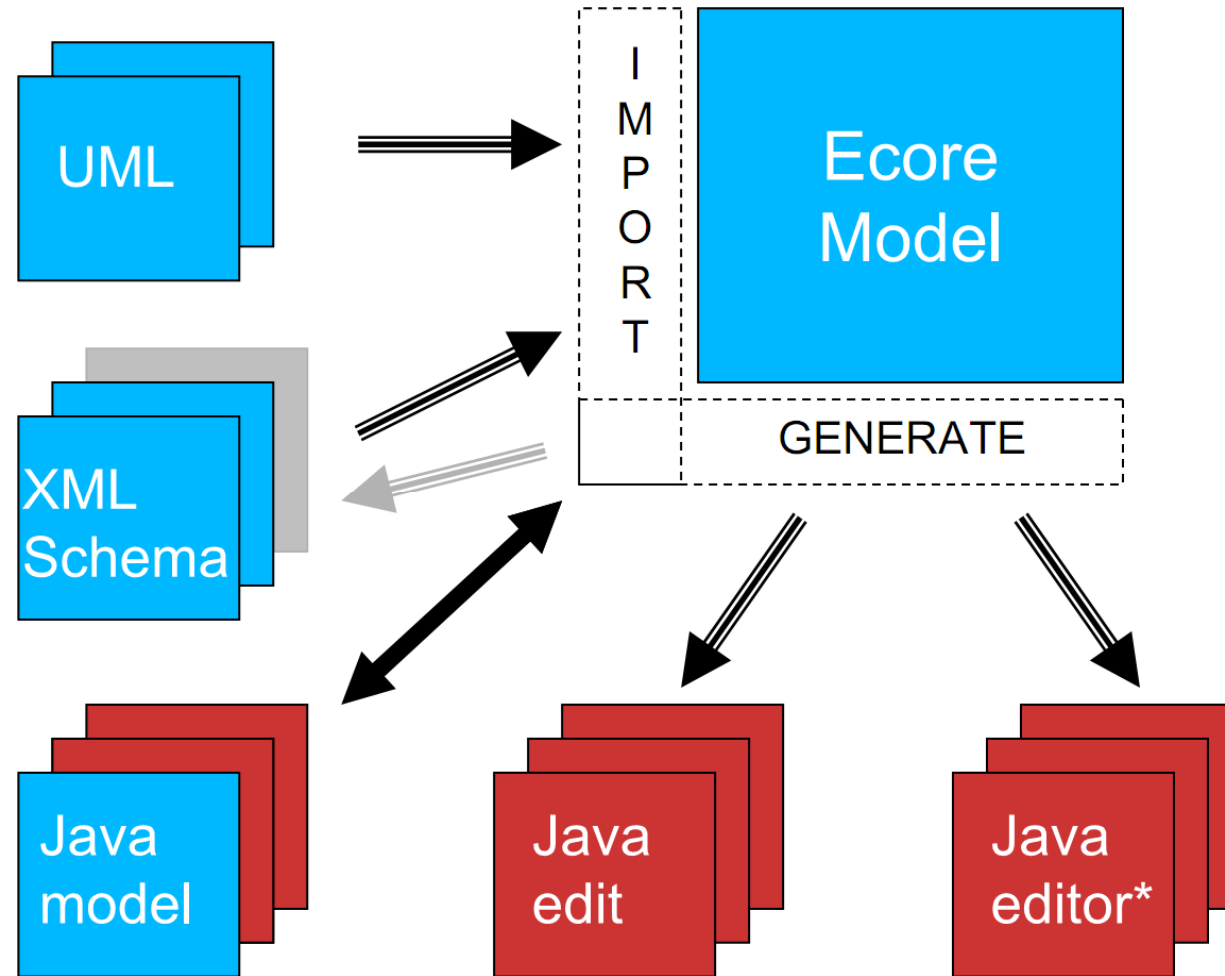
- BPEL & BPMN
  - Domain Model
  - UML Profile
  - EMF Metamodel

# **ECLIPSE META-MODELING FRAMEWORK**

# Eclipse Modeling Framework

- ❑ Modeling Framework with code generation
  - To build tools based on data models
  - The model is captured as a **XMI** (**X**ML **M**etadata **I**nterchange) file
  
- ❑ **Import existing code** to build the model
  - Java code with annotations
  - XML documents (**XSD** – **X**ML **S**chema **D**efinition)
  - UML tools (e.g., Rational Rose)
  
- ❑ **Code generation** from the model
  - Set of Java classes and interfaces
  - An Edit/Editor environment (editing tree)
  
- ❑ Many (and many more coming) extensions
  - Generate a graphical editor (**G**raphical **M**odeling **F**ramework, Sirius)
  - Generate a parser, syntax highlighting (XText, TCS, Sintaks)

# EMF import/export



IBM, Ed. Merks & D. Steinberg, EclipseCon 2005

## EMF and UML ?

### □ **MOF**: Meta-metamodel (M3) by OMG

- **M**eta-**O**bject **F**acilities
- EMOF (Essential MOF) is a subset

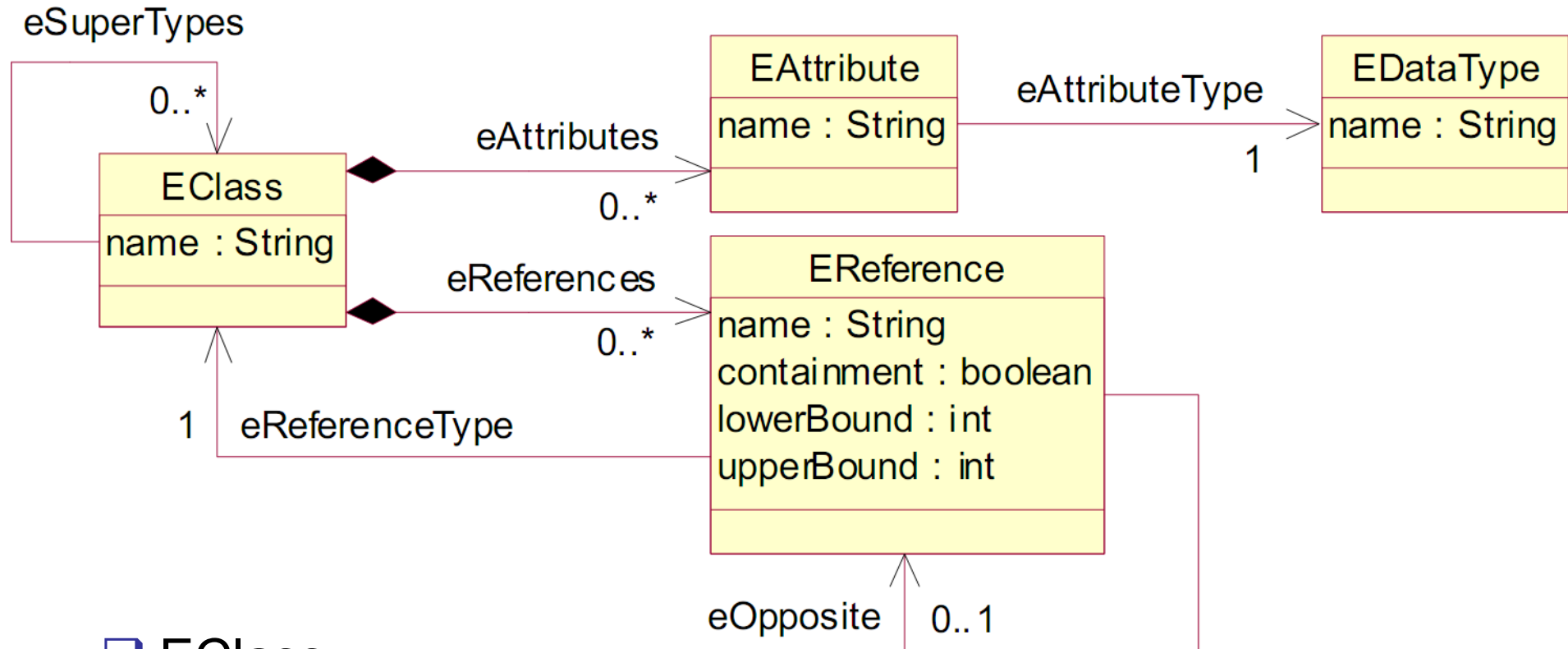
### □ **EMF**

- Was initially just an implementation of the MOF
- Has evolved to become **ECORE**

### □ Two very different business models

- OMG # Eclipse Foundation

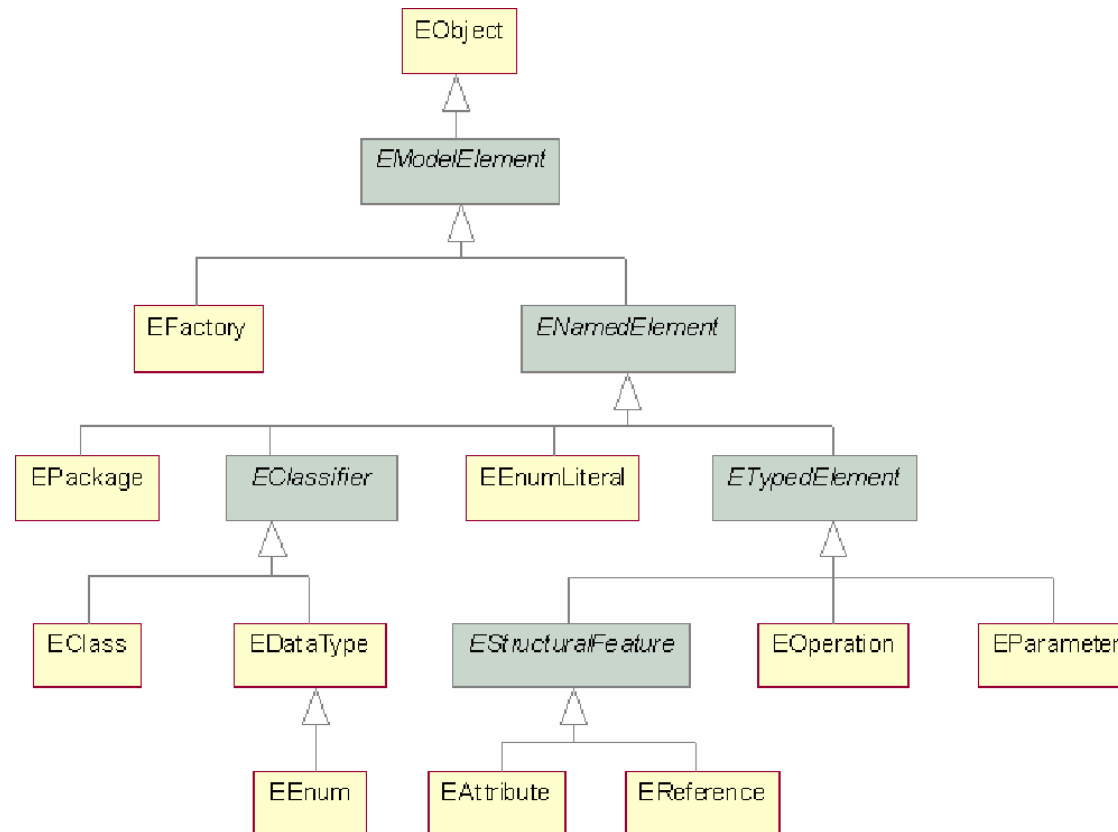
# Ecore: EMF meta-metamodel (M3)



## □ EClass

- Own attributes (data types) and typed references (classes)
- Can have a super type (autres EClass) **[specialization]**

# Ecore: EMF meta-metamodel (M3)

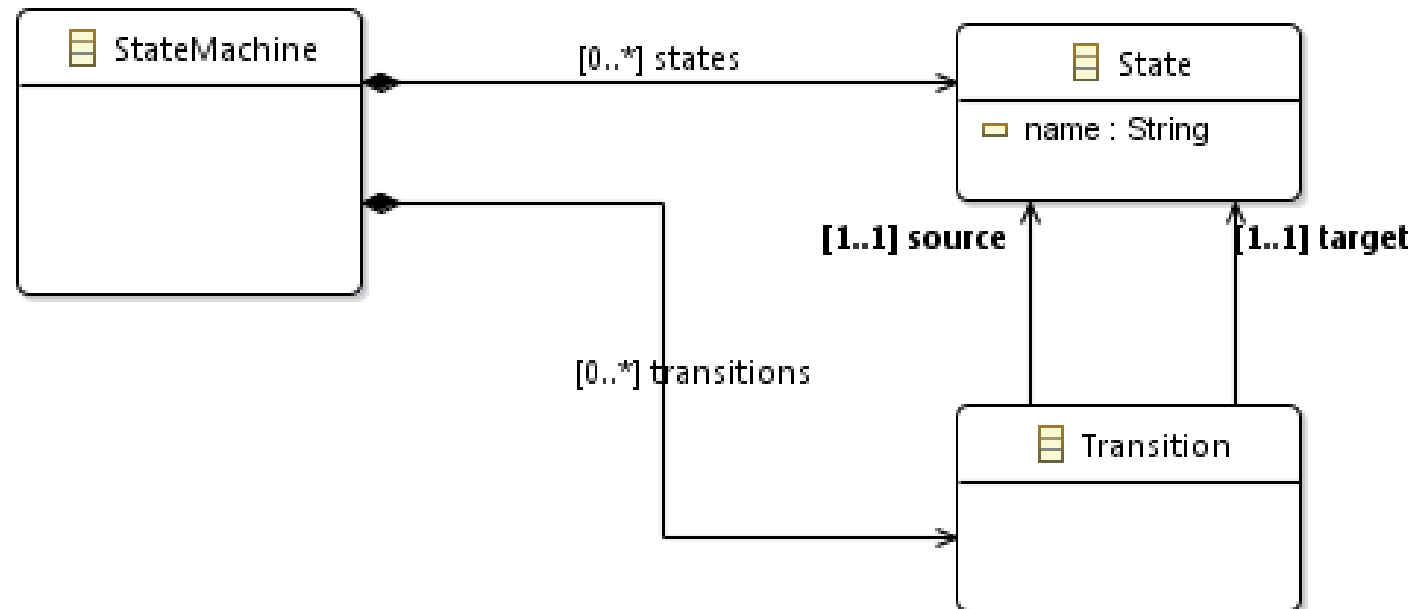


## □ Small meta-metamodel

- Enhance the native introspection of Java
- Only structural information (no access to the code)



## Example – State Machines



### □ Ecore Modeling Project (Eclipse Modeling Mars)

- StateMachines.aird
- StateMachines.ecore

# Generation model: .genmodel

The screenshot shows a project named 'stateMachines.genmodel' with a tree view containing:

- StateMachines
  - StateMachines
    - StateMachine
      - states : State
      - transitions : Transition
    - State
      - name : EString
    - Transition
      - source : State
      - target : State

Below the tree view is a 'Properties' panel with the following table:

Property	Value
▶ All	
▶ Edit	
▶ Editor	
▲ Model	
Array Accessors	<input checked="" type="checkbox"/> false
Binary Compatible Reflective Methods	<input checked="" type="checkbox"/> false
Class Name Pattern	<input type="text"/>
Containment Proxies	<input checked="" type="checkbox"/> false
Feature Delegation	<input type="text"/> None
Generate Schema	<input checked="" type="checkbox"/> false
Interface Name Pattern	<input type="text"/>
Minimal Reflective Methods	<input checked="" type="checkbox"/> true
Model Directory	<input type="text"/> /StateMachines/src-gen
Model Plug-in Class	<input type="text"/>
Model Plug-in ID	<input type="text"/> StateMachines

- Import Ecore model
- Create a .genmodel
  - What to generate
  - Where to generate
  - How to generate

Generate Model Code  
 Generate Edit Code  
 Generate Editor Code  
 Generate Test Code  
 Generate All

# Generate code/Edit/Editor

## □ Abstract model

- Java interfaces

```
public interface State extends EObject {  
    /** @generated */ String getName();  
    /** @generated */ void setName(String value);  
}
```

## □ Implementation

- With listeners

```
public abstract class StateImpl extends EObjectImpl implements State {  
    protected static final String NAME_EDEFAULT = null;  
    protected String name = NAME_EDEFAULT;  
    public void setName(String newName) {  
        String oldName = name;  
        name = newName;  
        if (eNotificationRequired())  
            eNotify(new ENotificationImpl(this, Notification.SET, StsPackage.STATE__NAME, oldName, name));  
    }  
}
```

# Automatic code generation

## □ Automatic generation

- Tree editor
- XML Marshalling/Unmarshalling
- XML Validator
- Wizard for creating new models

The screenshot displays a tree view of a state machine model. The tree structure is as follows:

- My.statemachines
  - Resource Set
    - platform:/resource/Test%20StateMachines/My.statemachines
      - State Machine
        - Region
          - State S0
          - State S1
          - Transition

Below the tree view, there are tabs for Selection, Parent, List, Tree, Table, and Tree with Columns. The Properties tab is active, showing the following table:

Property	Value
Source	State S0
Target	State S1

# CODE GENERATION

# Kinds of model transformations

## □ Model to text

- Generate text (or code) from a model
- Dedicated languages: XSLT
- Manual: in Java through the Ecore API

## □ Model to model

- Transform a model into another model
  - Ex: UML State Machines into NuSMV files
- Dedicated transformation languages
  - ATL, Kermeta, QVTo

# Accessing the model

## □ Standalone applications

- EMF generates a set of helpers to access/parse/generate models

## □ Through an eclipse plugin

- Small Java program that augments Eclipse
  - Add menu, button, editors, ...
- Better/easier integration with other tools
- Needs *Eclipse Modeling (Kepler)*
- File/New/Plug-in project...

# An example of plug-in

## □ Add a menu to Eclipse (3 extensions needed)

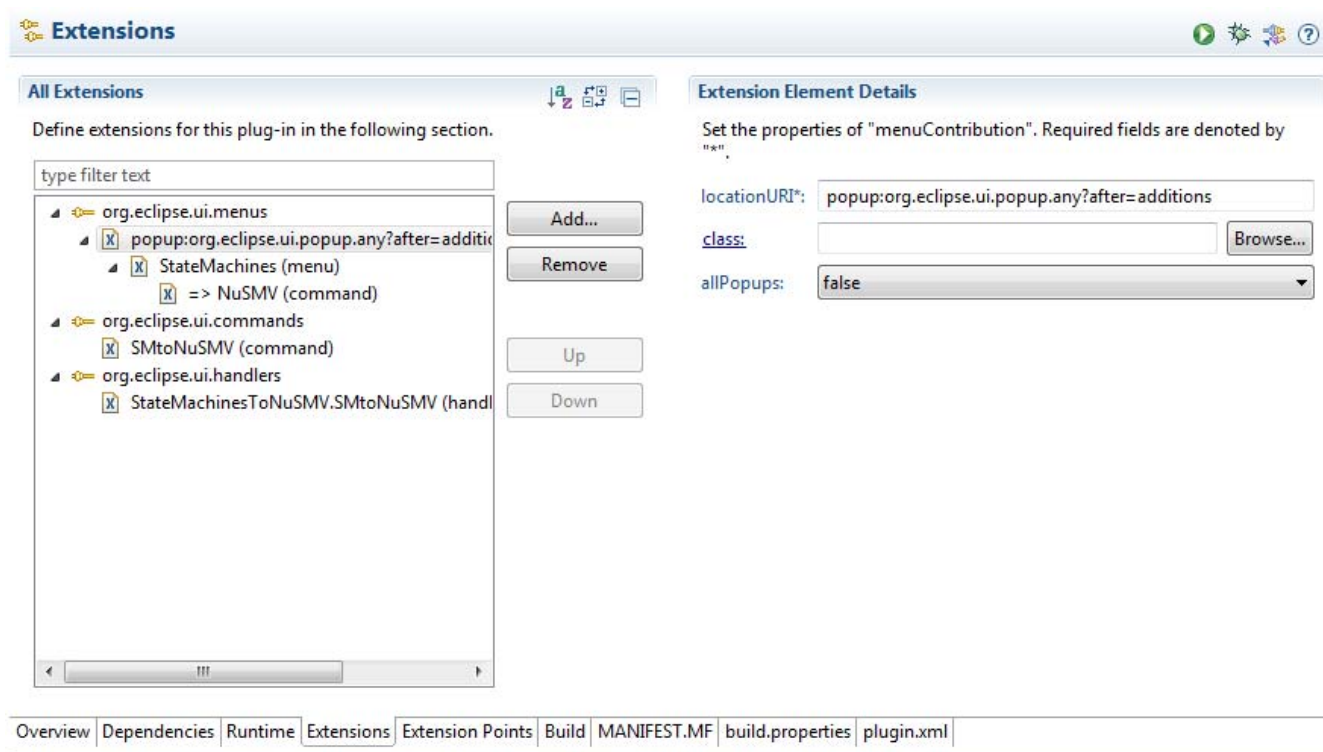
- [org.eclipse.ui.menus](#)
  - Add a menu and menu item into Eclipse
- [org.eclipse.ui.commands](#)
  - Add a command: can be (un)done through menus or toolbars
- [org.eclipse.ui.handlers](#)
  - Attach a handler to a command (code to be executed)



# org.eclipse.ui.menu

## □ 3 stages

- **menuContribution**: popup:org.eclipse.ui.popup.any?after=additions
- **menu**: with a label
- **command**: MenuItem that references a command



# org.eclipse.ui.menu

## □ Select when the menu is visible

- Ex1: only when a statemachines.StateMachine is selected
  - Requires a **dependency** to the code generated by EMF
- Ex2: org.eclipse.uml2.uml.StateMachine
  - Requires a **dependency** to org.eclipse.uml2.uml

The screenshot shows the Eclipse IDE's 'Extensions' dialog box. The 'All Extensions' tab is active, displaying a tree view of extensions for the 'org.eclipse.ui.menu' category. The tree includes 'org.eclipse.ui.menu', 'org.eclipse.ui.commands', and 'org.eclipse.ui.handlers'. Under 'org.eclipse.ui.menu', there are sub-extensions like 'popup:org.eclipse.ui.popup.any?after=additional', 'StateMachines (menu)', '=> NuSMV (command)', 'false (visibleWhen)', 'activeMenuSelection (with)', and '(iterate)'. The 'stateMachines.StateMachine' extension is selected. To the right, the 'Extension Element Details' tab is active, showing the 'type\*' property set to 'statemachines.StateMachine'. Below the tree are buttons for 'Add...', 'Remove', 'Up', and 'Down'.

# Visibility of the menu

❑ Either programmatically in the code

❑ Through the extension interface

- menuContribution locationURI: popup:org.eclipse.ui.popup.any?after=additions
- menu label: « Name of the menu »
- Command
  - Label: « Name of the command »
  - Id: same as the id of the command
- isVisibleWhen: checkIfEnabled=true
- With variable: activeMenuSelection
- Iterate operator: and isEmpty:false
- Adapter (depends on where to integrate)
  - Type: org.eclipse.core.resources.IFile (File in the Navigator)
  - Type: org.eclipse.jdt.core.ICompilationUnit (Java file in Package explorer)

# org.eclipse.ui.commands

- Allows for the creation of commands
  - A command can be done/undone by clicking a menu or a toolbar icon or by code
  - Give a unique id
    - Ex: fr.unice.m1.SMtoNuSMV.command
  - Must be referenced by menus, toolbars, handlers

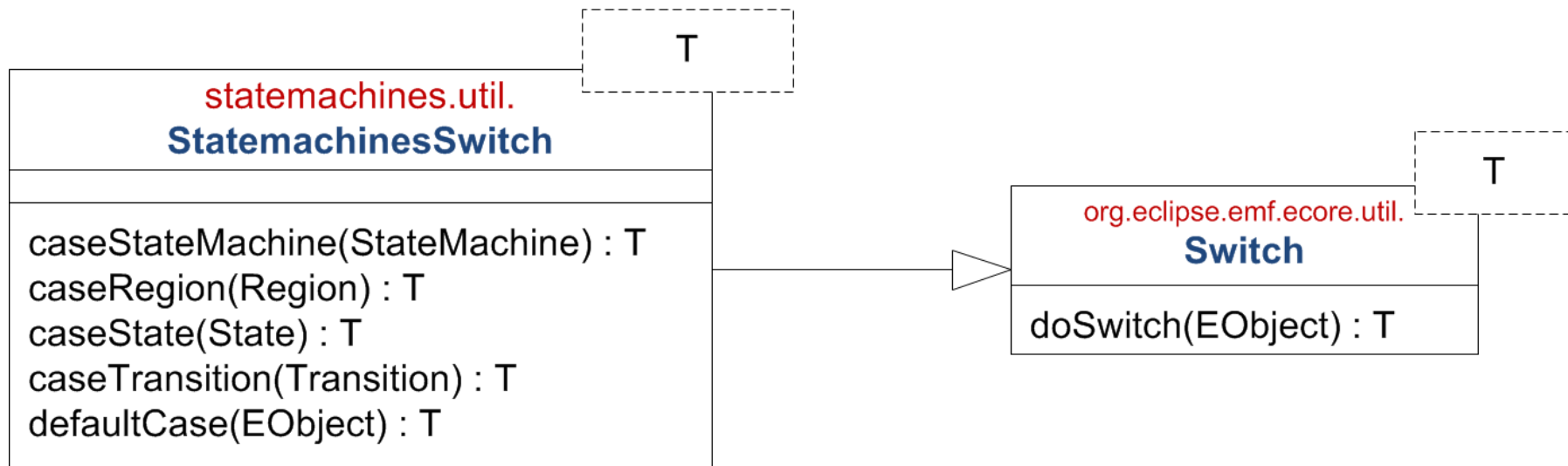
# org.eclipse.ui.handlers

- ❑ Specify what code should be executed/attached to a command
  - Reference a command through its id
  - Define a class that must implement **org.eclipse.core.commands.IHandler**

```
public class SMTToNuSMV implements IHandler {
    public void addHandlerListener(IHandlerListener handlerListener) {}
    public void dispose() {}
    public Object execute(ExecutionEvent event) throws ExecutionException {
        // TODO Auto-generated method stub
        return null;
    }
    public boolean isEnabled() { return true; }
    public boolean isHandled() { return true; }
    public void removeHandlerListener(IHandlerListener handlerListener) {}
}
```

# EMF Switches

- ❑ Realize the **visitor** design patterns
  - Automatically generated by EMF
  - Allows for *visiting* a complex hierarchical structure



# Switch: do it yourself

## □ Example that counts the number of elements

```
public class SMCountElements extends StateMachinesSwitch<Boolean> {
    private int nbStateMachines = 0;
    private int nbStates = 0;
    private int nbTransitions = 0;

    public Boolean caseStateMachine(StateMachine object) {
        nbStateMachines++;
        for(Region region : sm.getRegions()) doSwitch(region);
        return true;
    }
    public Boolean caseRegion(Region region) {
        for(State state : region.getStates()) doSwitch(state);
        for(Transition transition : region.getTransitions()) doSwitch(transition);
        return true;
    }
    public Boolean caseState(State object) {
        nbStates++;
        return true;
    }
    public Boolean caseTransition(Transition object) {
        nbTransitions++;
        return true;
    }
}
```

# Use the Switch

## □ Define the right handler

```
public class SMTToNuSMVHandler extends AbstractHandler {
    @Override
    public Object execute(ExecutionEvent event) throws ExecutionException {
        ISelection selection = PlatformUI.getWorkbench().getActiveWorkbenchWindow()
            .getActivePage().getSelection();
        if (!(selection instanceof StructuredSelection)) return null;
        Object selected = ((StructuredSelection)selection).getFirstElement();

        // The type should be guaranteed by the "isVisibleWhen"
        assert(selected instanceof StateMachine);
        SMCountElements counter = new SMCountElements();
        counter.doSwitch((StateMachine)selected);
        JOptionPane.showMessageDialog(null, counter.getNbStatemachines()+" state machines\n"+
            counter.getNbStates()+" states\n"+
            counter.getNbTransitions()+" transitions",
            "State Machines", JOptionPane.INFORMATION_MESSAGE);

        return null;
    }
}
```