USING A QVT LIKE APPROACH TO AUTOMATE UML2 TO JAVA EE TRANSFORMATIONS.

By Alexis HENRY
Engineering Director

BLU AGE™
JAVA EE .NET APPLICATION GENERATOR

Java

Sun microsystems

Agenda

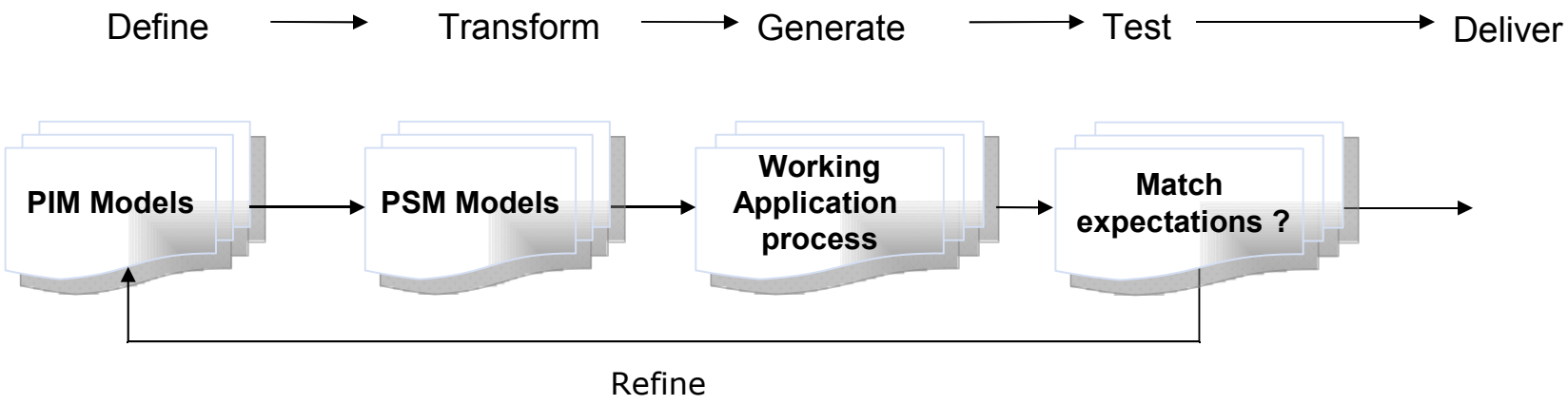
- I Introduction
  - I.I Software delivery issues
  - I.II MDD software generation
  - I.III Example of MDD engine
- II Model transformation basic principles
  - II.I Atlas Transformation Language (ATL)
  - II.II Eclipse as integration platform
- III Generation workflow
  - III.I Generation principles and model executability
  - III.II Model debugging
  - III.III Partial transformation
- IV Transformation Use Case: UML to Java™ 2 platform, Enterprise Edition (J2EE™ platform)
  - IV.I Business needs and modelling
  - IV.II JET in action: generating code
  - IV.III Model debugging

I.I Automatic transformation: what for ?

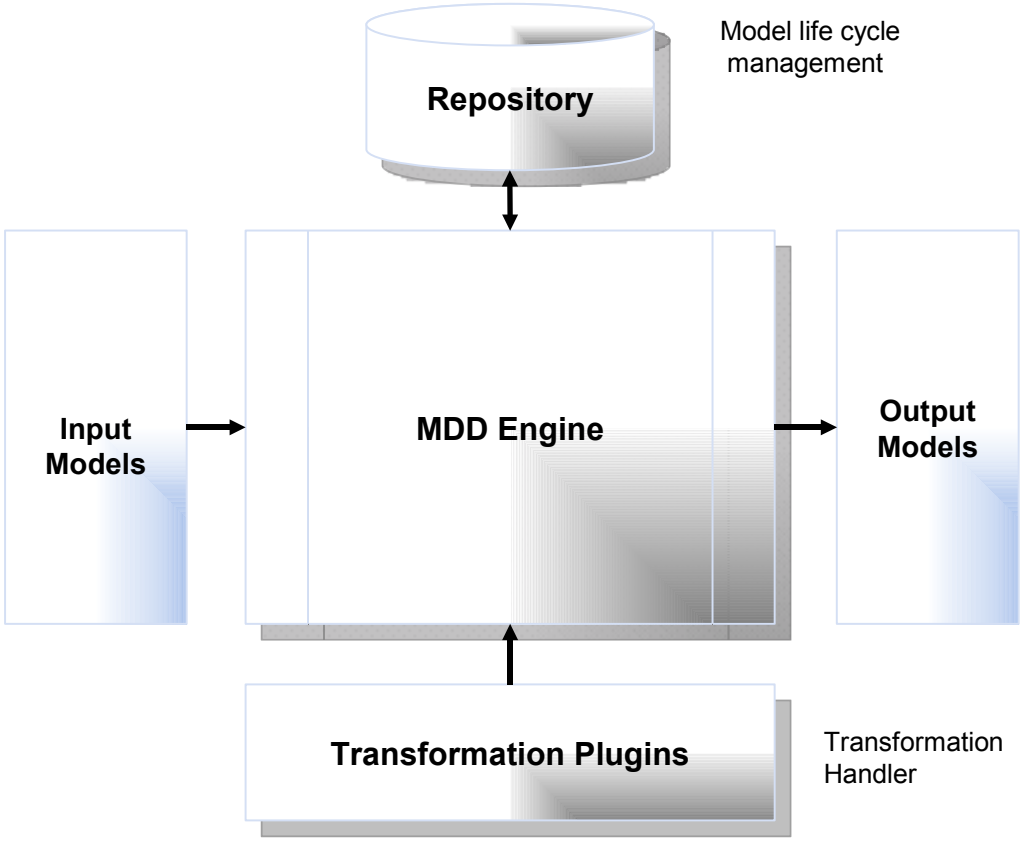
- Major Issues for software delivery
  - Align software developments with changing business issues
  - Ensure comprehensive application specifications and user satisfaction
  - Manage application development cycles and application life cycle
  - Maintain agile applications and take advantage of IT improvements
- Model engineering assets to solve these issues?
  - Focus on business needs and achieve architecture scalability
  - Software engineers and architects are involved in business processes and user requirements
  - Model transformation into application ensure application code conforms to requirements
  - No unnoticed change in code, models makes your software factory agile
  - Application delivery is shorten and can support business changes
  - Reduce maintenance cost, focus on investment and improvement
- Next: AN APPROACH TO AUTOMATE UML2 TO J2EE CLASS TRANSFORMATIONS

I.II MDD Software Generation chain

- Agile delivery: based on expected business process
  - Short itérations based on user requirements (PIM)
  - Choose target technology (PSM)
  - Generate ready to execute application
  - Test and validate
  - New requirements?

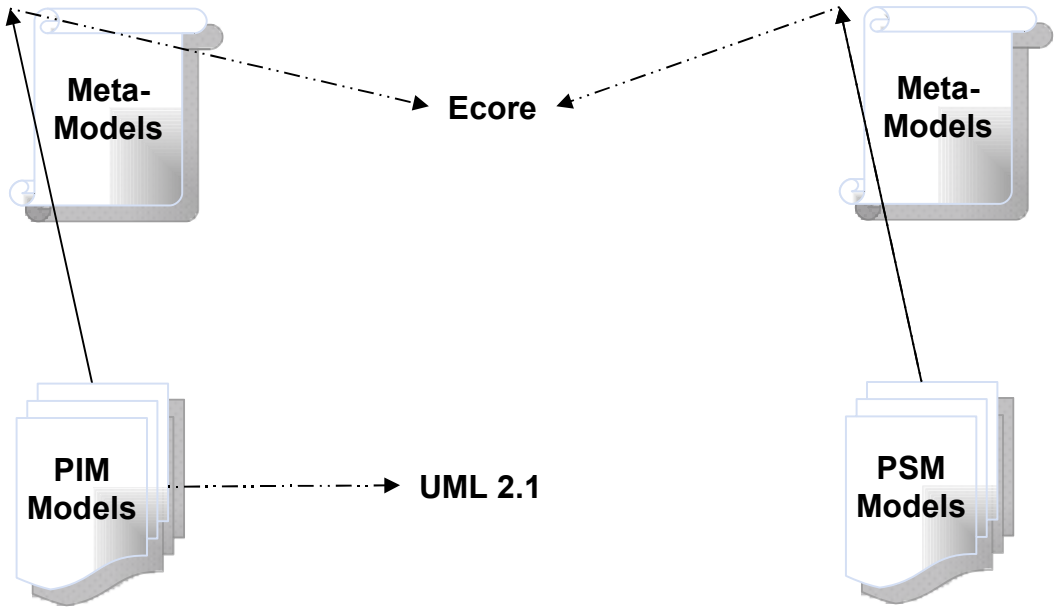


I.II MDD Transformation



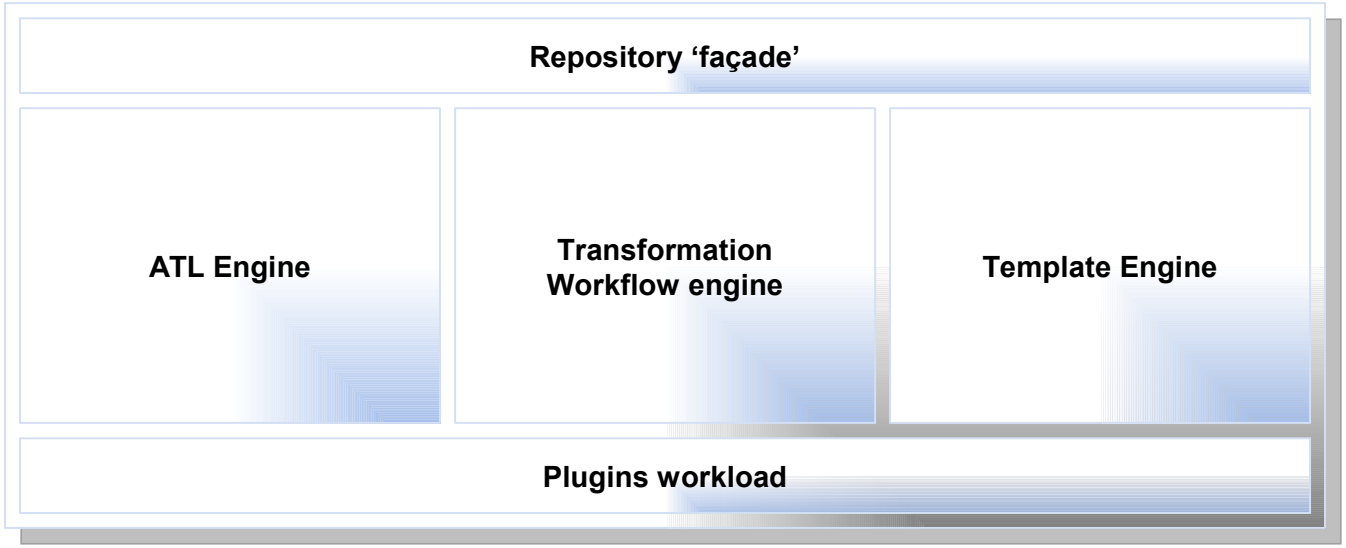
I.II Models

- UML 2.1 models
- Based on Ecore metamodels

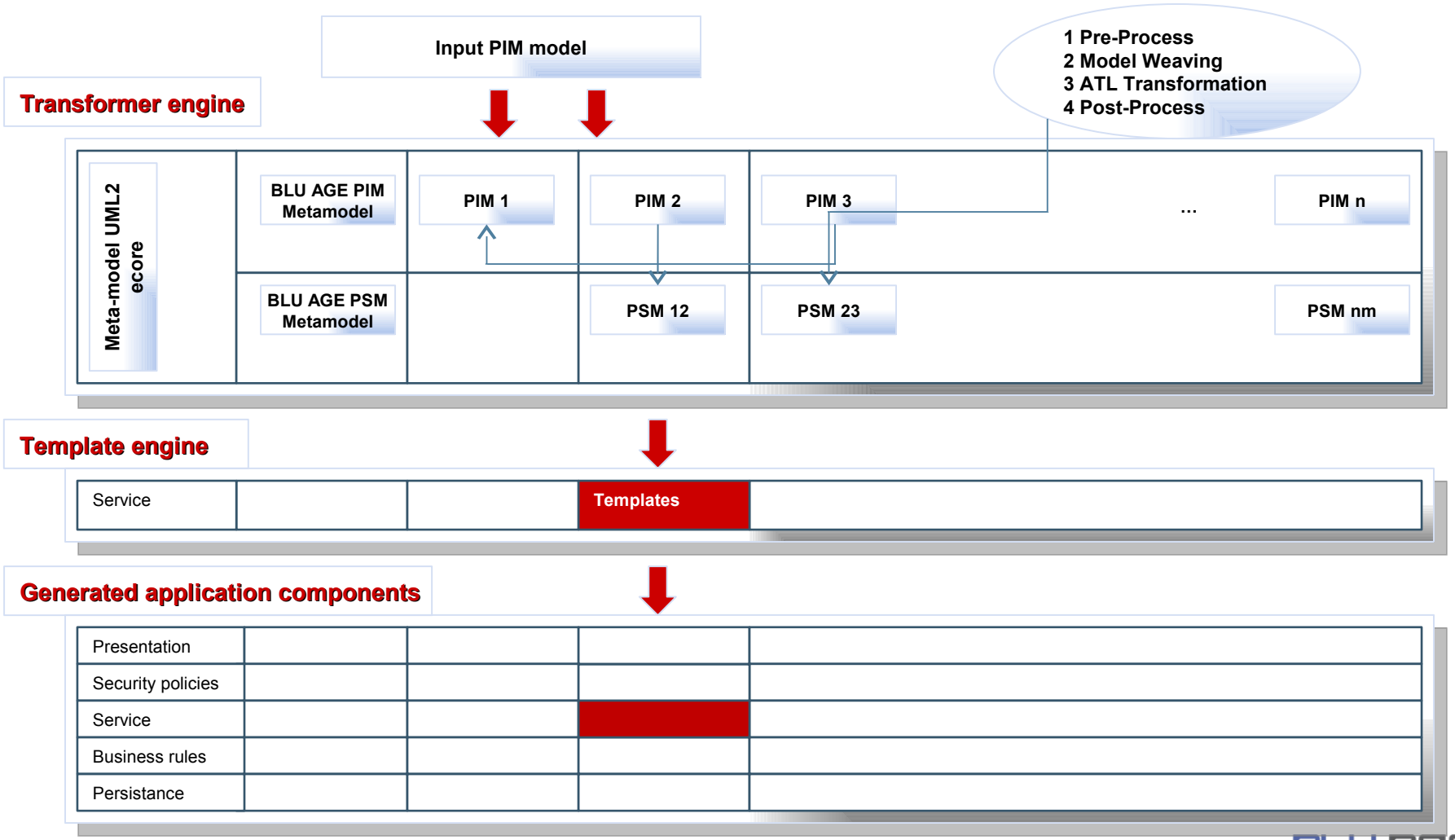


I.III Example of MDD Engine

- Repository ‘Façade’ (model Access)
- ATL Engine (QVT like transformation language)
- Template Engine (Model to text)
- Plugins Workload (Transformation plugins management)
- Transformation Workflow management (configuration)

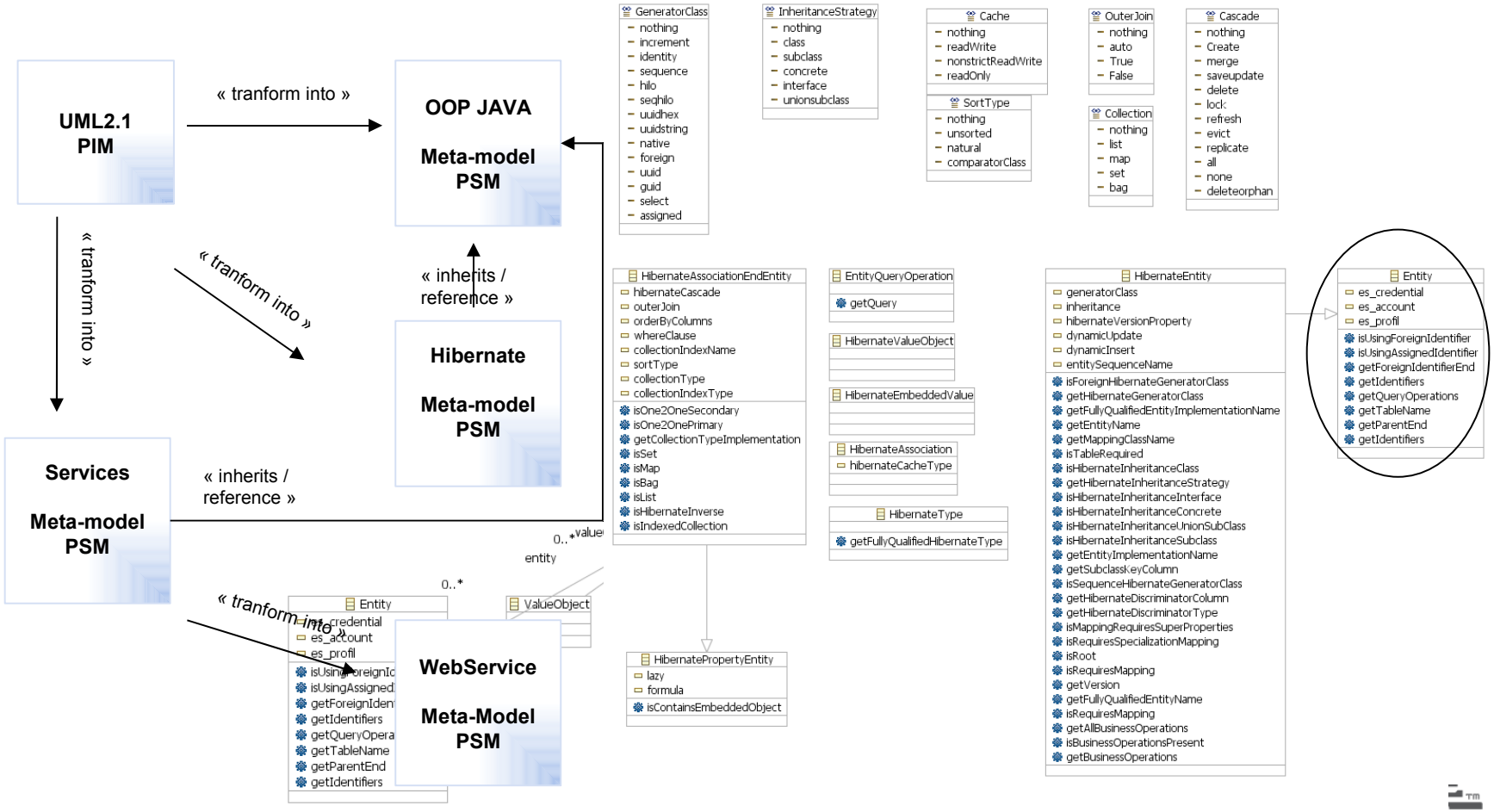


I.III Transformation Workflow Engine



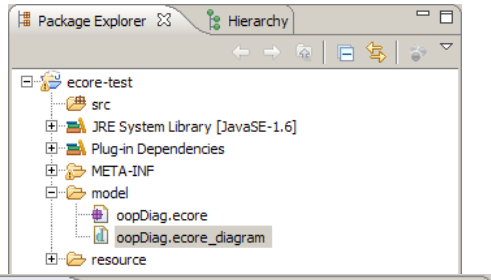
I.III The "Meta-model forest"

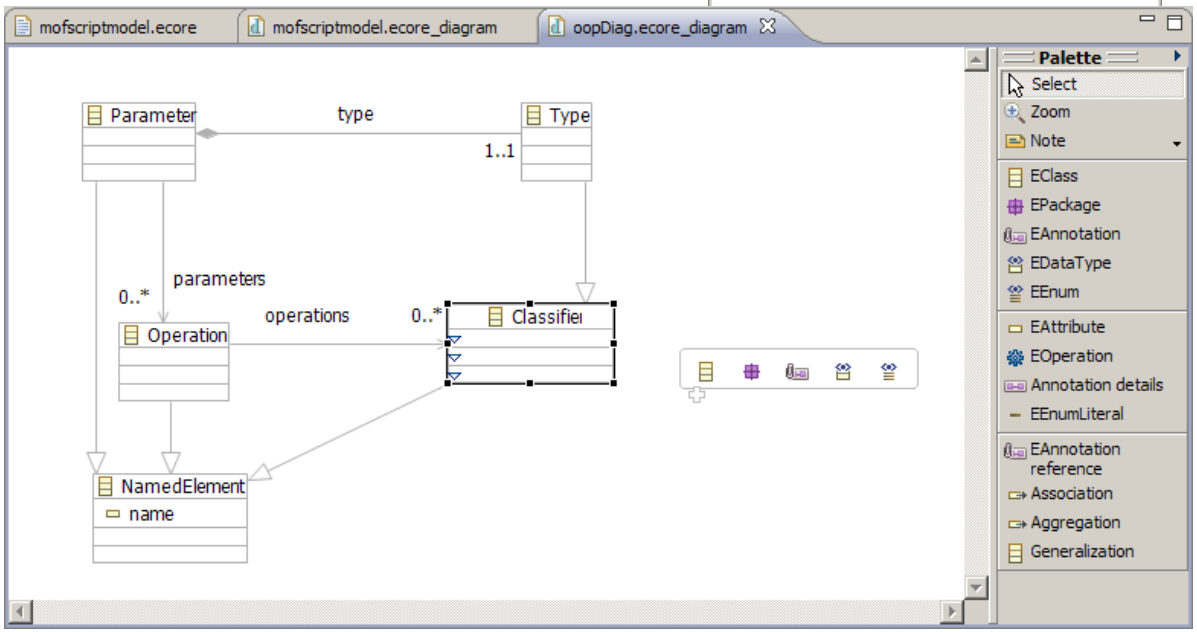
- Example of a "meta-model forest":

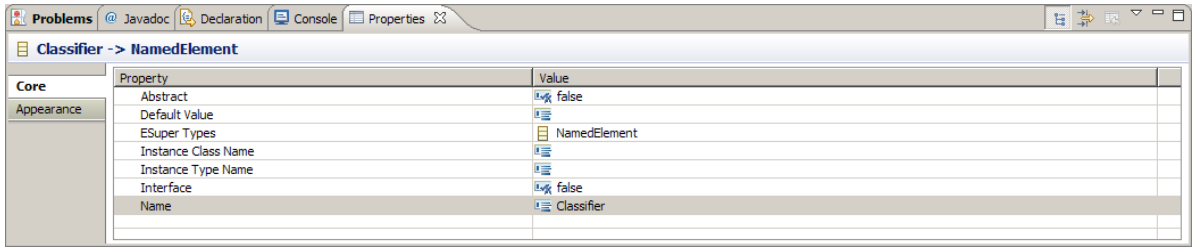


I.III The meta-model specification

- The meta-model is defined using EMF, the Eclipse Modelling Framework
- EMF provides tree-based editors to define the meta-model

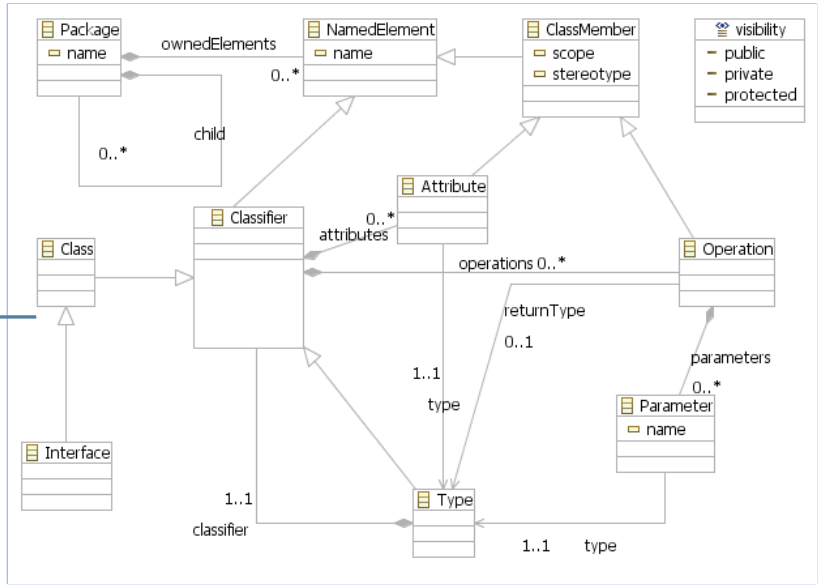




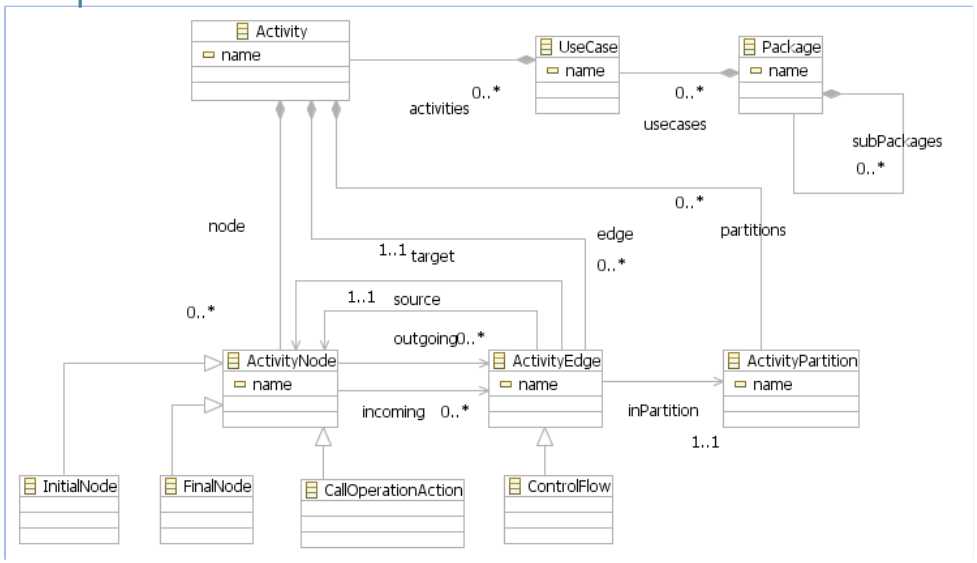


I.III Simplified UML 2.1 meta-model

Static Classes meta-model



Dynamic Behavior meta-model



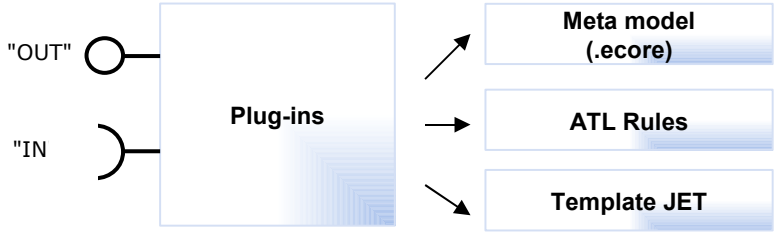
Referenced meta-model specification

- platform:/resource/bluage-zoo/metamodel/bluageDynamicBehavior.ecore
  - bluageDynamicBehavior
    - Activity
    - ActivityEdge
    - ActivityPartition
    - CallOperationAction -> ActivityNode
    - ControlFlow -> ActivityEdge
    - ActivityNode
    - InitialNode -> ActivityNode
    - FinalNode -> ActivityNode
    - UseCase
- platform:/resource/bluage-zoo/metamodel/bluageStaticClass.ecore
  - bluageEcore
    - Package
    - NamedElement
    - Classifier -> NamedElement
    - Type -> Classifier
    - Attribute -> ClassMember
    - visibility
    - Operation -> ClassMember
    - Class -> Classifier
    - Interface -> Class
    - Parameter
    - ClassMember -> NamedElement

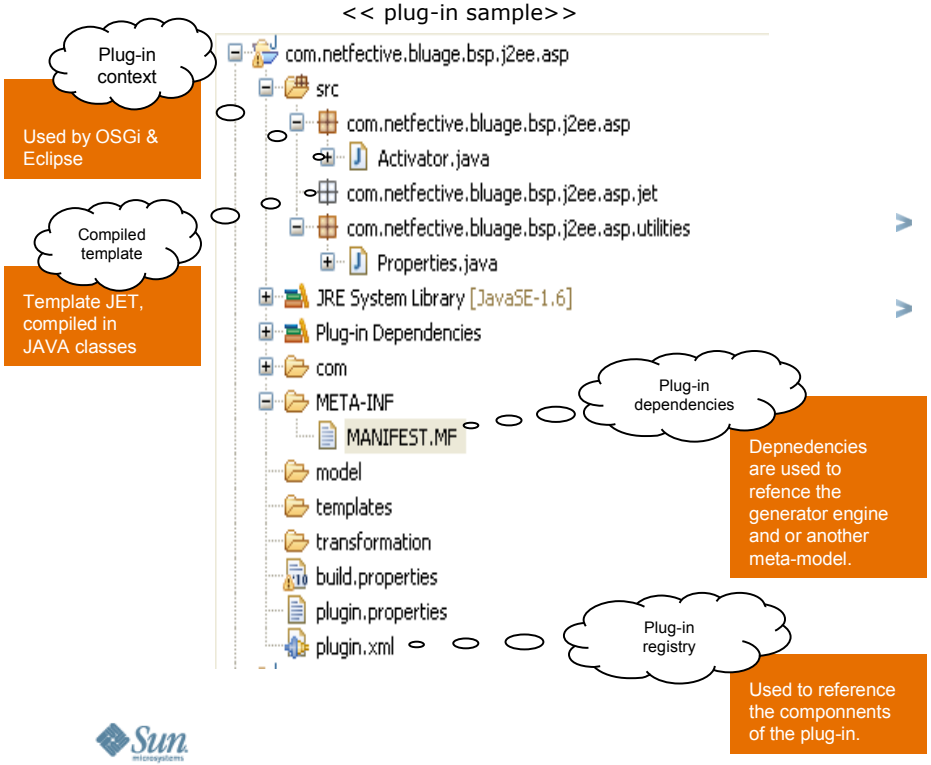
BLU AGE™ JAVA EE .NET APPLICATION GENERATOR

I. III Transformation Plug-ins

- The transformation plug-ins
  - Contains transformation rules M2M
  - Contains transformation templates M2T
- Chaining the transformations
  - Each plug-ins contains a interface specifying
    - an input model
    - an output model



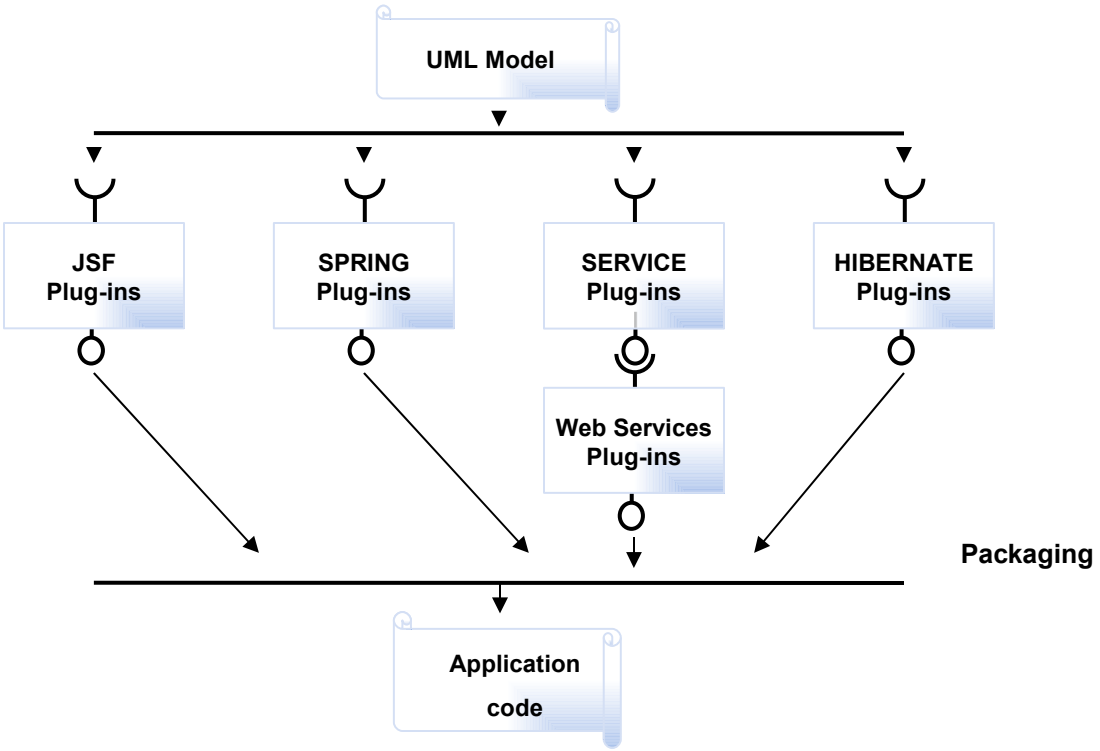
<< plug-in sample>>



- Plug-ins compositions
  - Each elements of the plug-in is referenced inside « plugin.xml »
- Plug-ins technology
  - A plug-in may contain:
    - Template utility classes,
    - The meta model,
    - The java implementation of the meta-model,
    - A model loading facility.

I.III Transformation Plug-ins

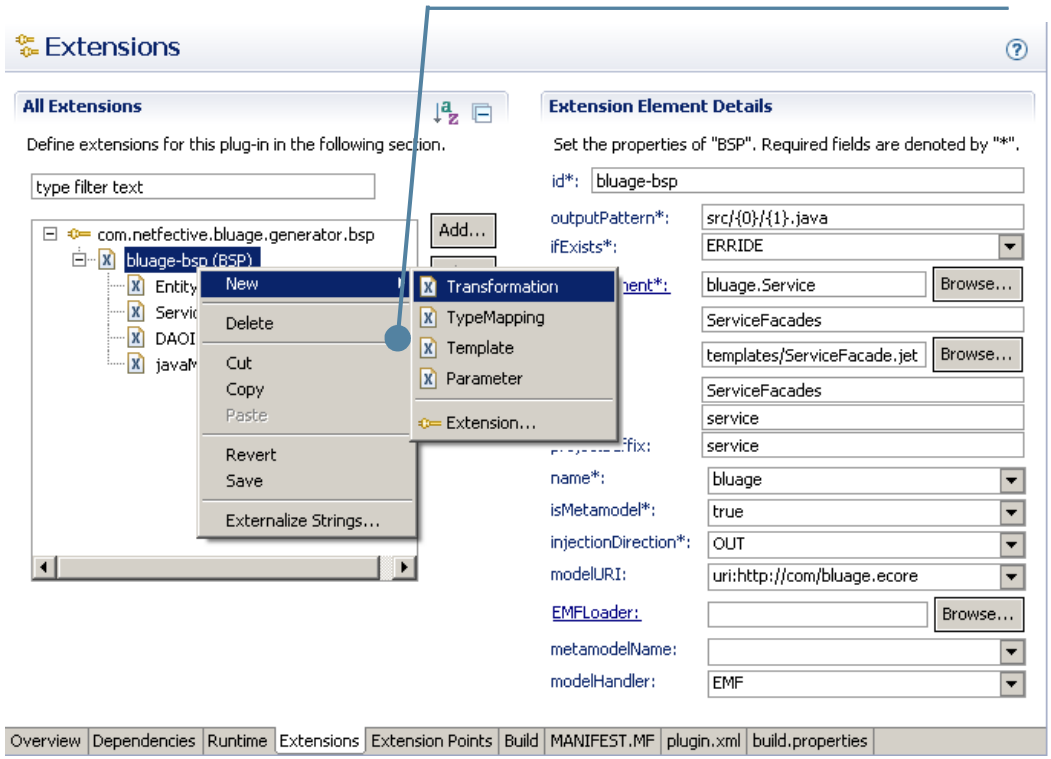
- Chaining the transformations
  - Each plug-ins contains a interface specifying
    - an input model
    - an output model
  - Each plug-in is responsible for a set of technology
  - Eventually all layers of the model are transformed



I.III Integration platform based on Eclipse plug-ins

- Eclipse integration and tooling
  - transformation highlighting
  - Meta-model awareness: Auto completion, type checker
  - Refactoring support
  - Debugger
  - Meta-models evolve: Transformations must be updated

- Transformation configuration
- Mapping settings
- Template parameters

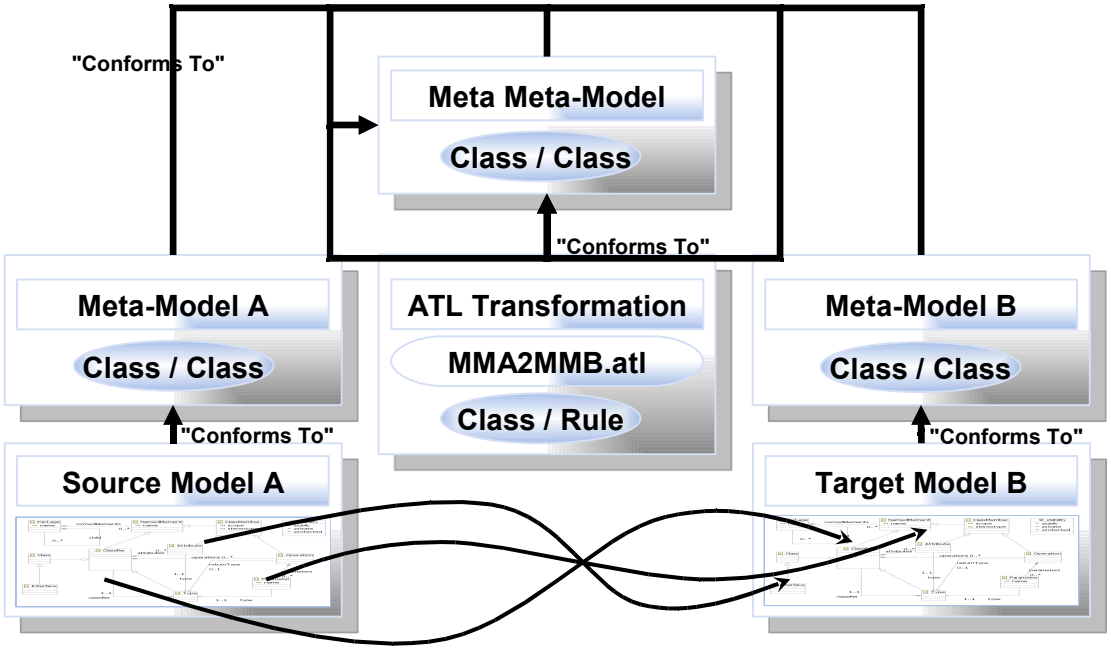


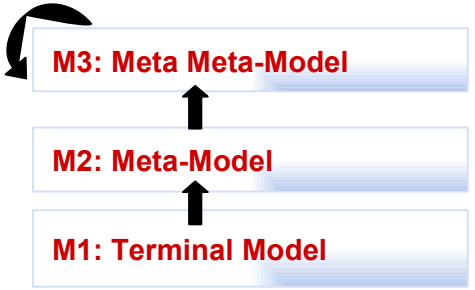
Agenda

- I Introduction
  - I.I Software delivery issues
  - I.II MDD software generation
  - I.III Example of MDD engine
- II Model transformation basic principles
  - II.I Atlas Transformation Language (ATL)
  - II.II Eclipse as integration platform
- III Generation workflow
  - III.I Generation principles and model executability
  - III.II Model debugging
  - III.III Partial transformation
- IV Transformation Use Case: UML to J2EE platform
  - IV.I Business needs and modelling
  - IV.II JET in action: generating code
  - IV.III Model debugging

II.I ATL Transformation principles

- A model transformation is the automatic creation of target models from source models
- ATL language is a declarative-imperative hybrid:
  - There are declarative matched rules
  - There are imperative called rules and action blocks
- All declarative rules that match are applied

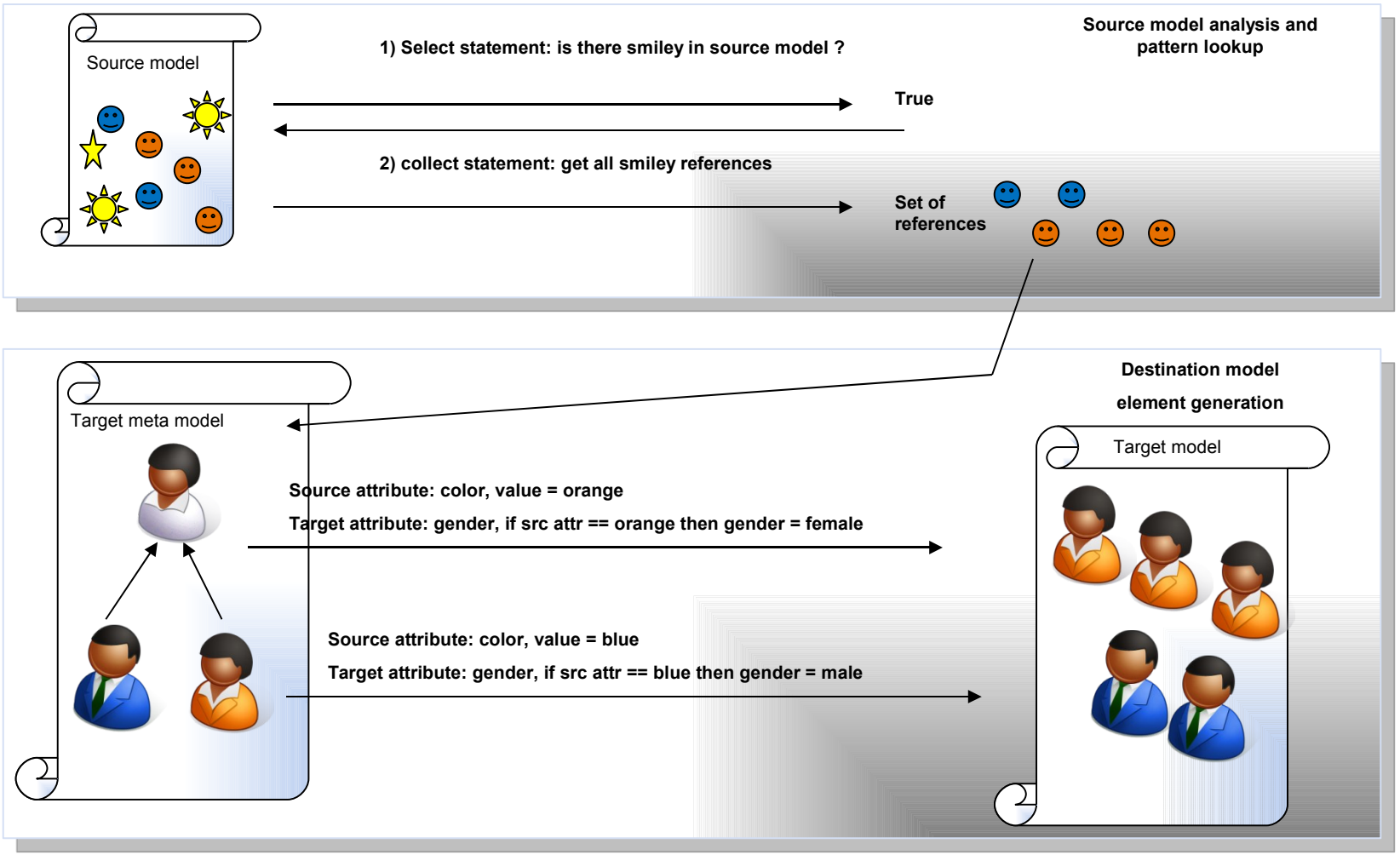




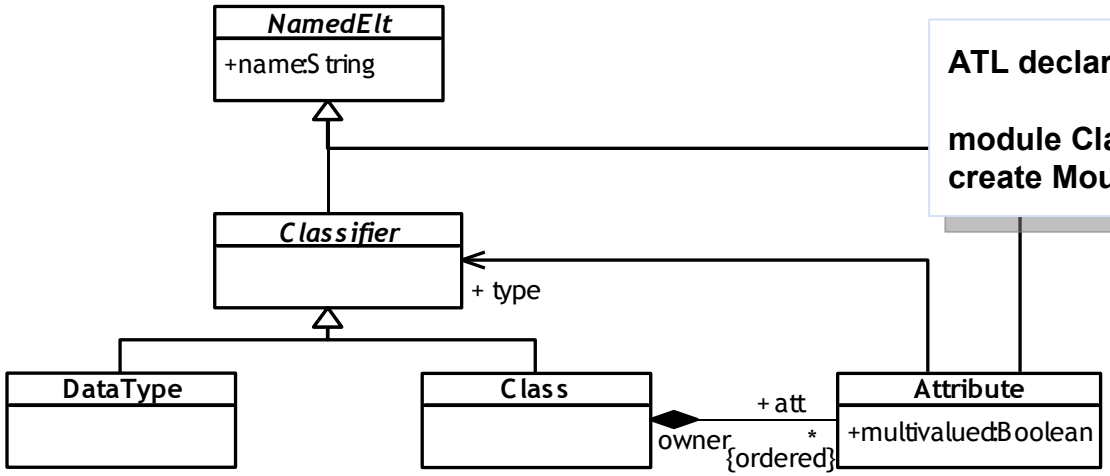
II.I ATL declarative rules

- Source pattern
  - The source pattern is made of:
    - A labeled set of types coming from the source metamodels
    - A guard (Boolean expression) is used to filter matches
  - The source pattern is used to select a set of elements coming from the source model:
    - Which complies to the pattern (guard = true)
    - Which is then transformed using ATL imperative rules
- Target pattern
  - The target pattern is made of:
    - A set of types coming from the target metamodels
    - For each element of this set, a group of bindings
    - A binding specifies the initialization of a target element properties using an expression
  - For each match, the target pattern is applied:
    - Elements are created in the target models (one for each type of the target pattern)
    - Target elements are initialized by executing the bindings:
      - First evaluating their value
      - Then assigning this value to the corresponding property

II.I ATL transformation overview



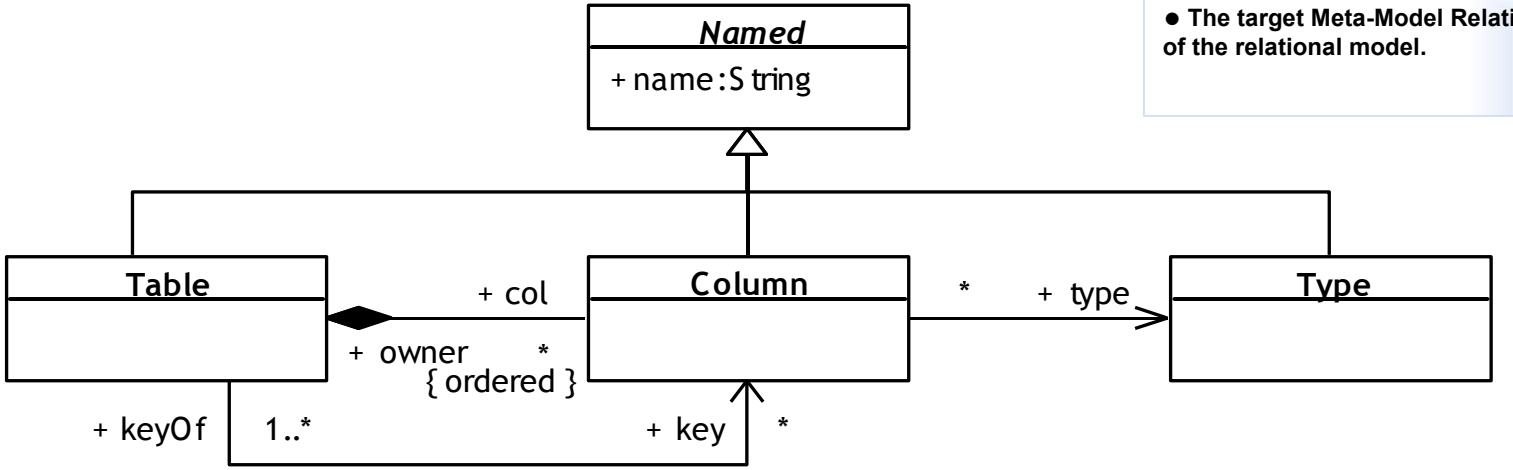
II.I ATL transformation example: Class to Relational



ATL declaration of this transformation:

```
module Class2Relational;
create Mout: Relational from Min: Class;
```

- **The source Meta-Model Class is a simplification of class diagrams.**
- **The target Meta-Model Relational is a simplification of the relational model.**



II.I ATL transformation example: Class to Relational

```
rule Class2Table {
   from
      c: Class!Class
   to
      t: Relational!Table (
         name <- c.name,
         col <- c.attr->select (e | not
         e.multiValued
            )->union(Sequence {key}),
               key <- Set {key}
        ),
        key: Relational!Column (
           name <- 'Id'
        )
}
rule SingleValuedAttribute2Column {
     from
               a: Class!Attribute (not a.multiValued)
     to
               c: Relational!Column ( name <- a.name )
}
```

A Table is created for each Class

The columns of the table correspond to the non-multi-valued attributes of the class.

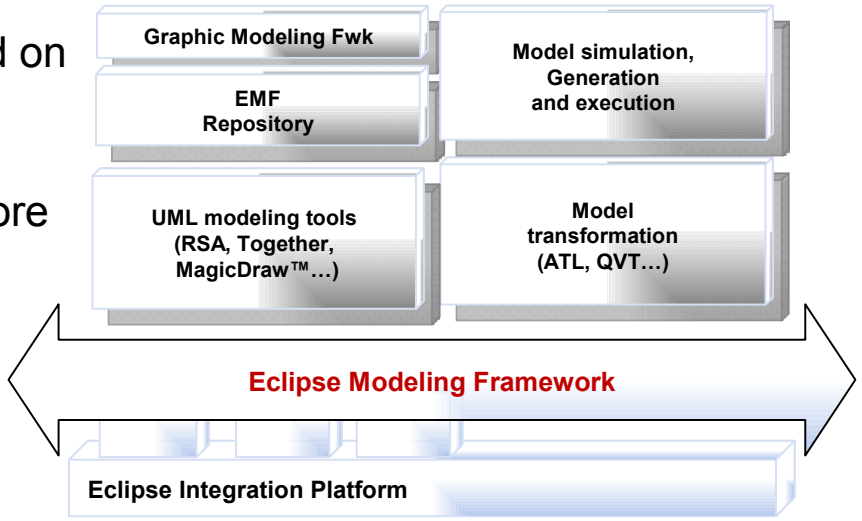
Each Table owns a key containing a unique identifier.

A Column is created for each single-valued Attribute.

II.II Eclipse integration platform

- Goals:
  - End-to-end integration tools that allows to build models, verify them and generate various artifact from them

- Challenges:
  - Editors integration to design models based on meta-models
  - Verifying the models as you build them
  - Transforming/modifying models, one or more model-to-model transformation steps
  - Generating application source code from models using templates, the model is transformed to executable code



Agenda

- I Introduction
  - I.I Software delivery issues
  - I.II MDD software generation
  - I.III Example of MDD engine
- II Model transformation basic principles
  - II.I Atlas Transformation Language (ATL)
  - II.II Eclipse as integration platform
- III Generation workflow
  - III.I Generation principles and model executability
  - III.II Model debugging
  - III.III Partial transformation
- IV Transformation Use Case: UML to J2EE platform
  - IV.I Business needs and modelling
  - IV.II JET in action: generating code
  - IV.III Model debugging

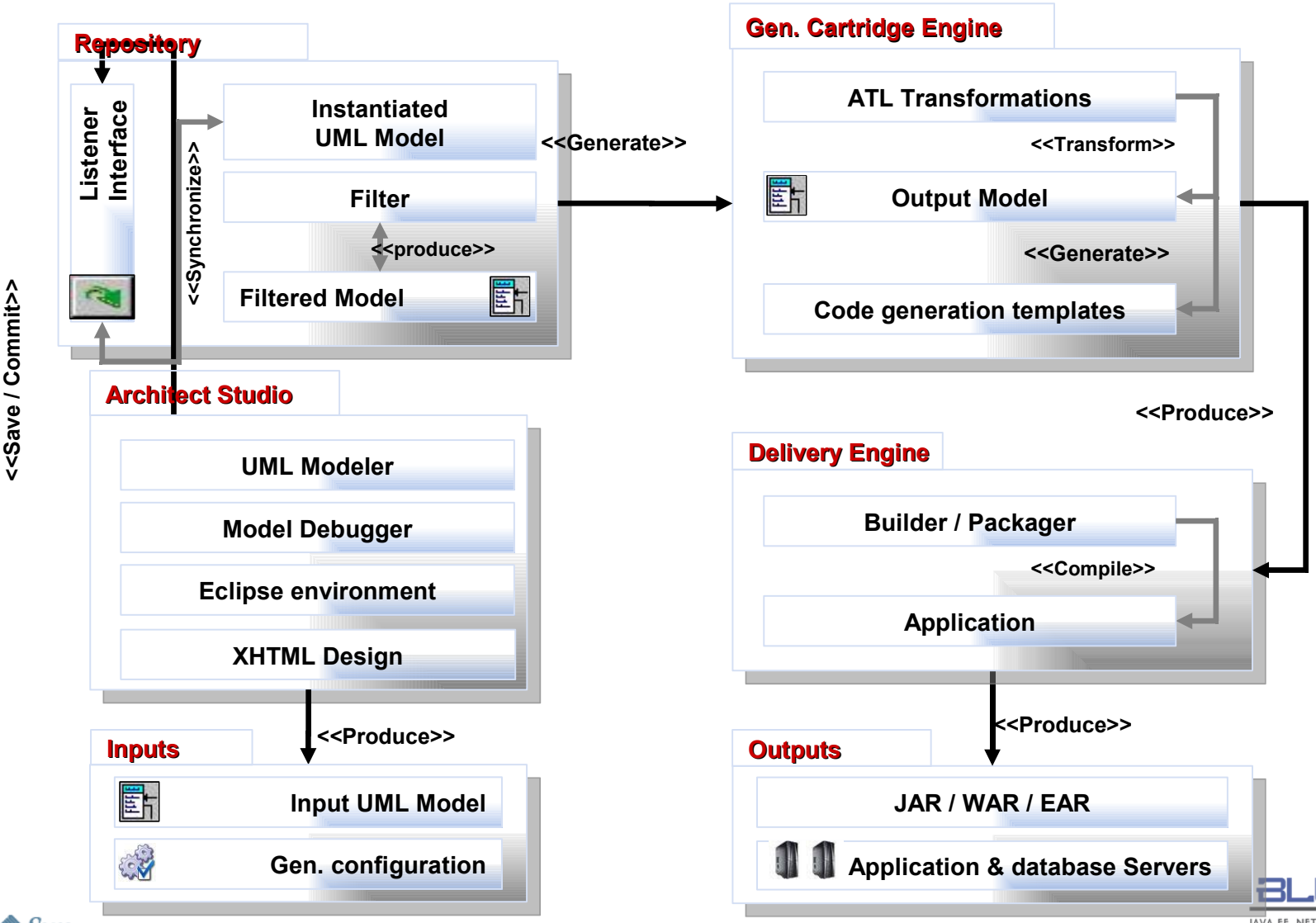
III.I Generation principles (1/2)

- Application generation and deployment from UML 2.1 models
  - Executable UML models
  - Use of referenced meta-model for model transformation
  - OCL to apply specific constraints for validating the transformation
  - Use of referenced technical environment (PDM) for source code generation and deployment
- Model update trigger code generation
  - Advanced on-the-fly model level integration approach
  - Synchronization and notification mechanisms
  - Model modifications are propagated immediately
  - Partial generation, only impacted elements are generated
- UML debugger with stepping and introspection
  - Select to apply transformation rules step-by-step
  - Integration of breakpoint support in the UML models

III.I Generation principles (2/2)

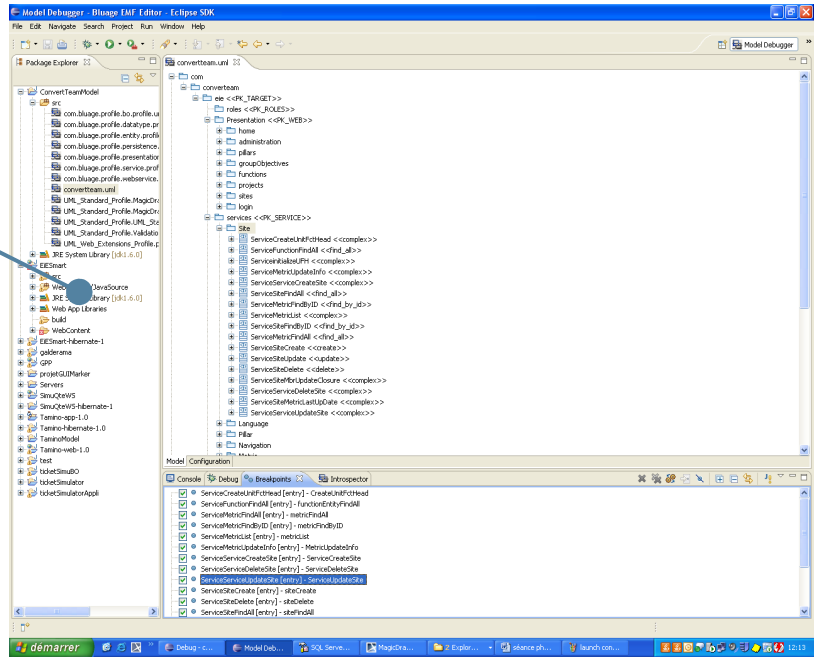
- EMF support integration
  - Models transformation
  - Referenced meta-model for code generation
  - Transformation of different kind of model: BPMN/BPEL, SySML, DSL…
- MDA Query/View/Transformation (QVT) like implementation with ATL (ATLAS Transformation Language)
- Partial model transformation
  - Interactive setting where various models need to be synchronized in real time
  - A transformation automatically produces a trace as a side effect
  - Changes in the source model produce trace information used to determine which (partial) transformation must be reevaluated
- Workbench Handler (projects and workspace views) available as Eclipse plug-ins

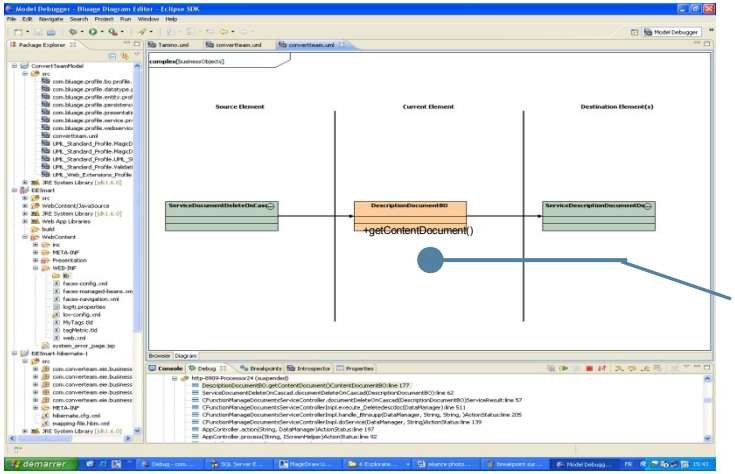
III.I Real Time application generation



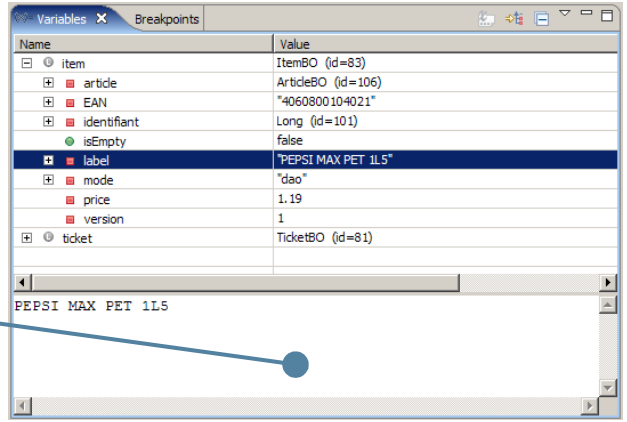
III.II Model debugging

Tree-based editors to define the UML model



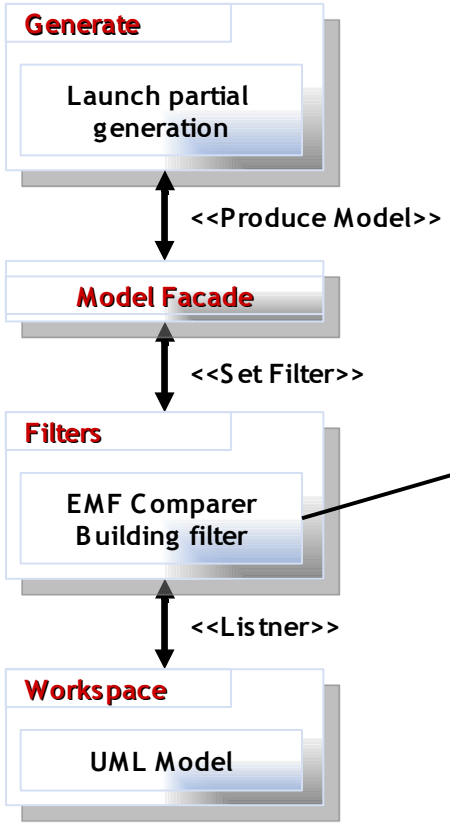


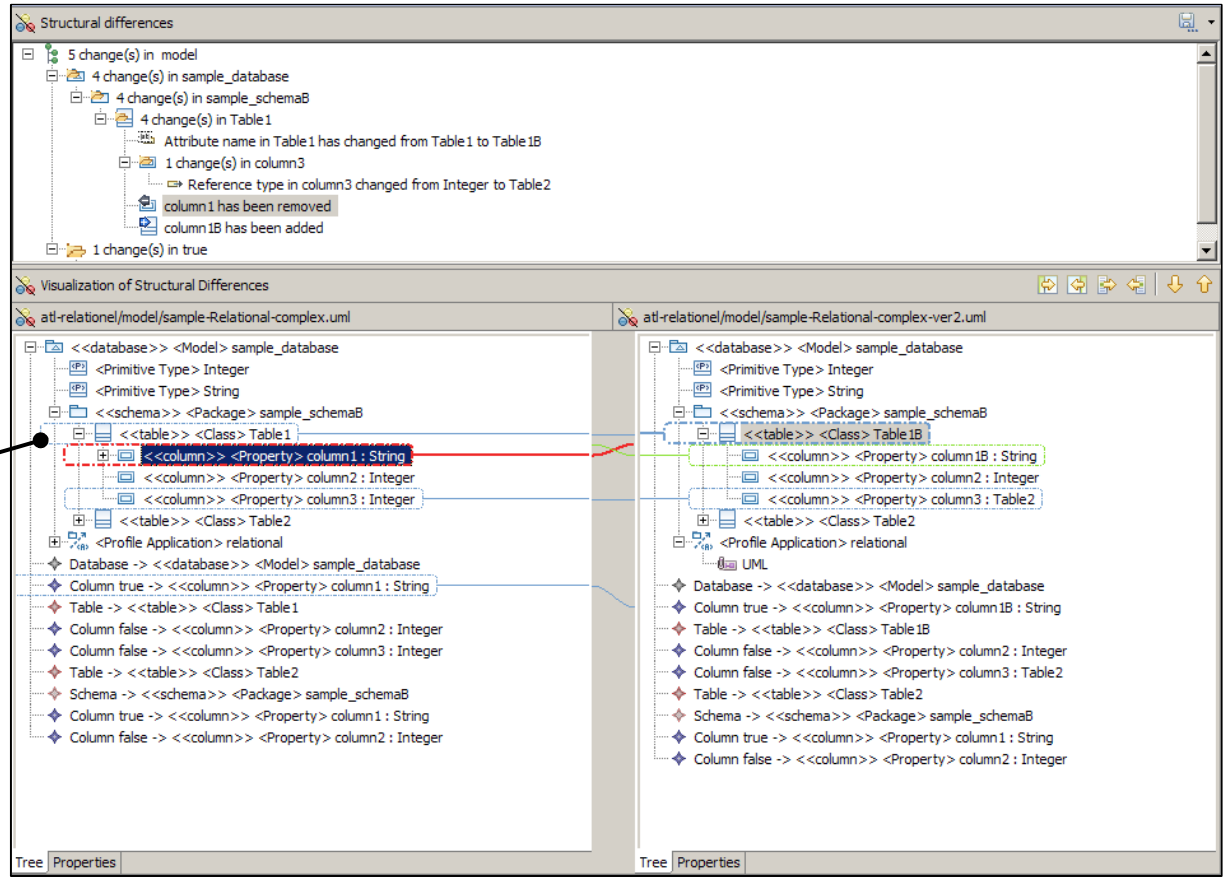
Breakpoint in debug mode with step-by-step execution



Introspection view in debug mode

III.II Partial model transformation

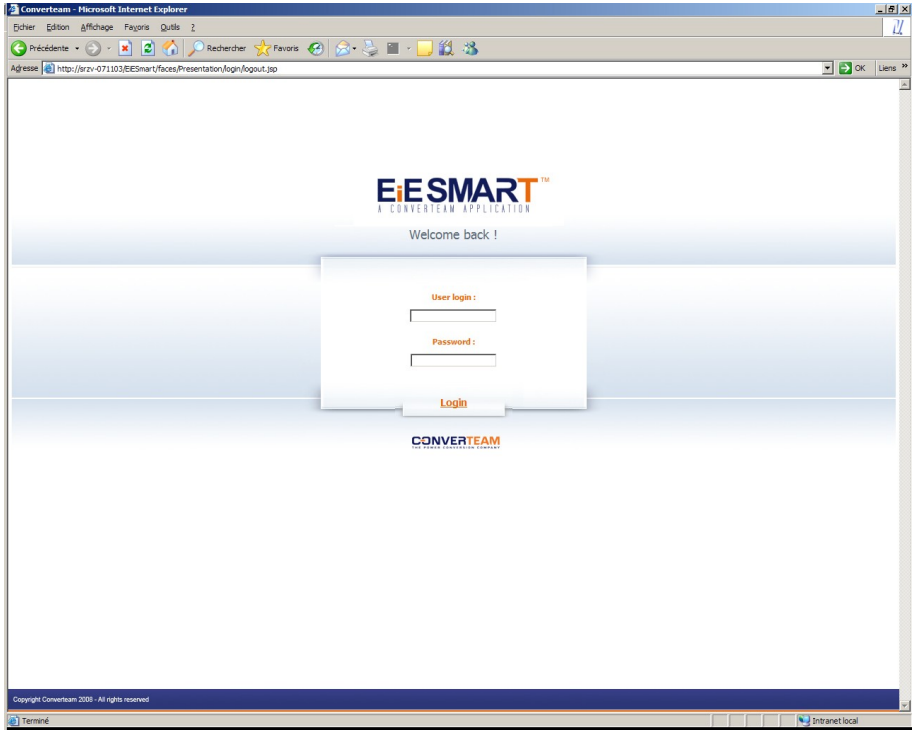




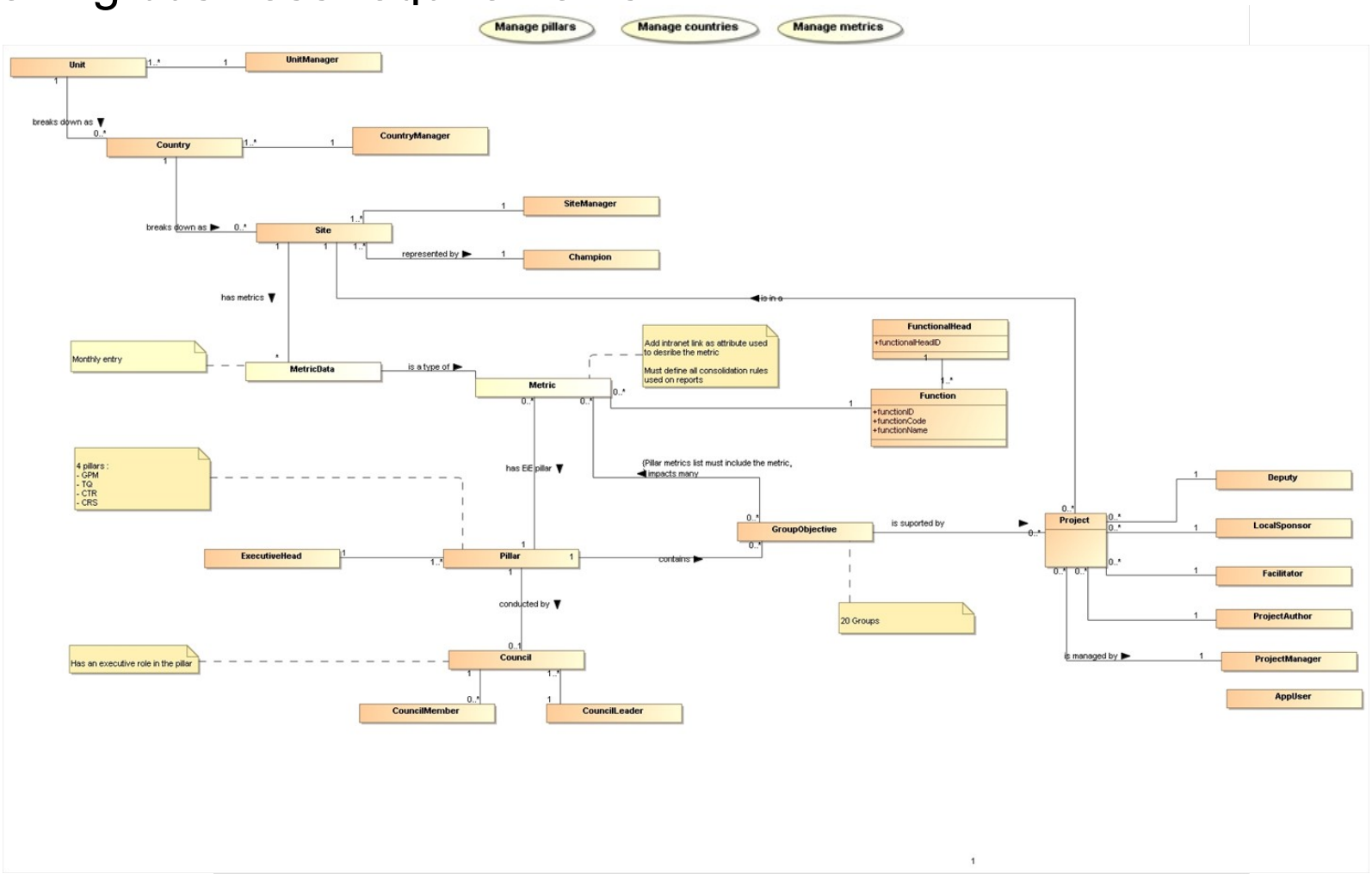
Agenda

- I Introduction
  - I.I Software delivery issues
  - I.II MDD software generation
  - I.III Example of MDD engine
- II Model transformation basic principles
  - II.I Atlas Transformation Language (ATL)
  - II.II Eclipse as integration platform
- III Generation workflow
  - III.I Generation principles and model executability
  - III.II Model debugging
  - III.III Partial transformation
- IV Transformation Use Case: UML to J2EE platform
  - IV.I Business needs and modelling
  - IV.II JET in action: generating code
  - IV.III Model debugging

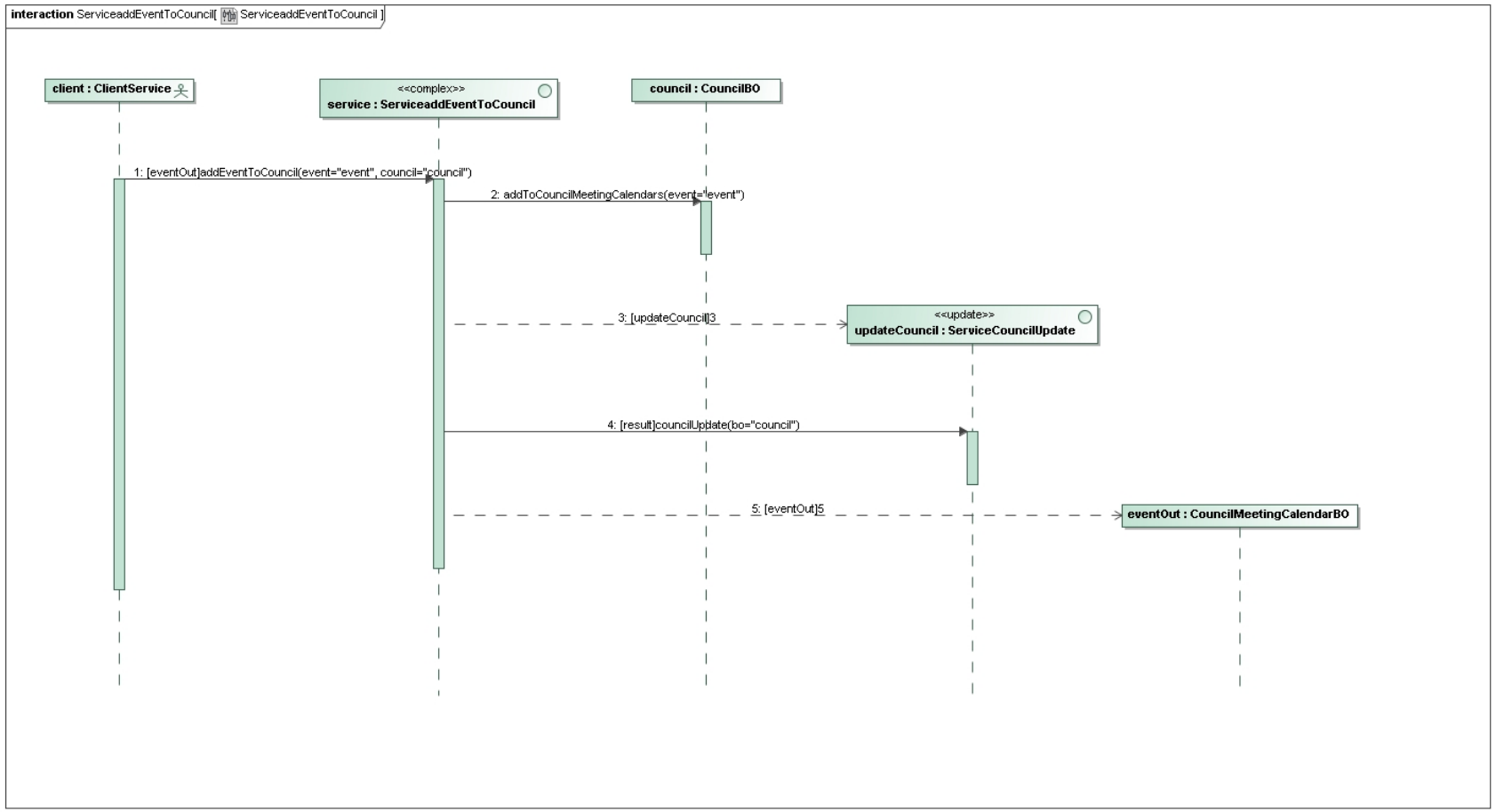
IV.I USE CASE : UML to J2EE Application



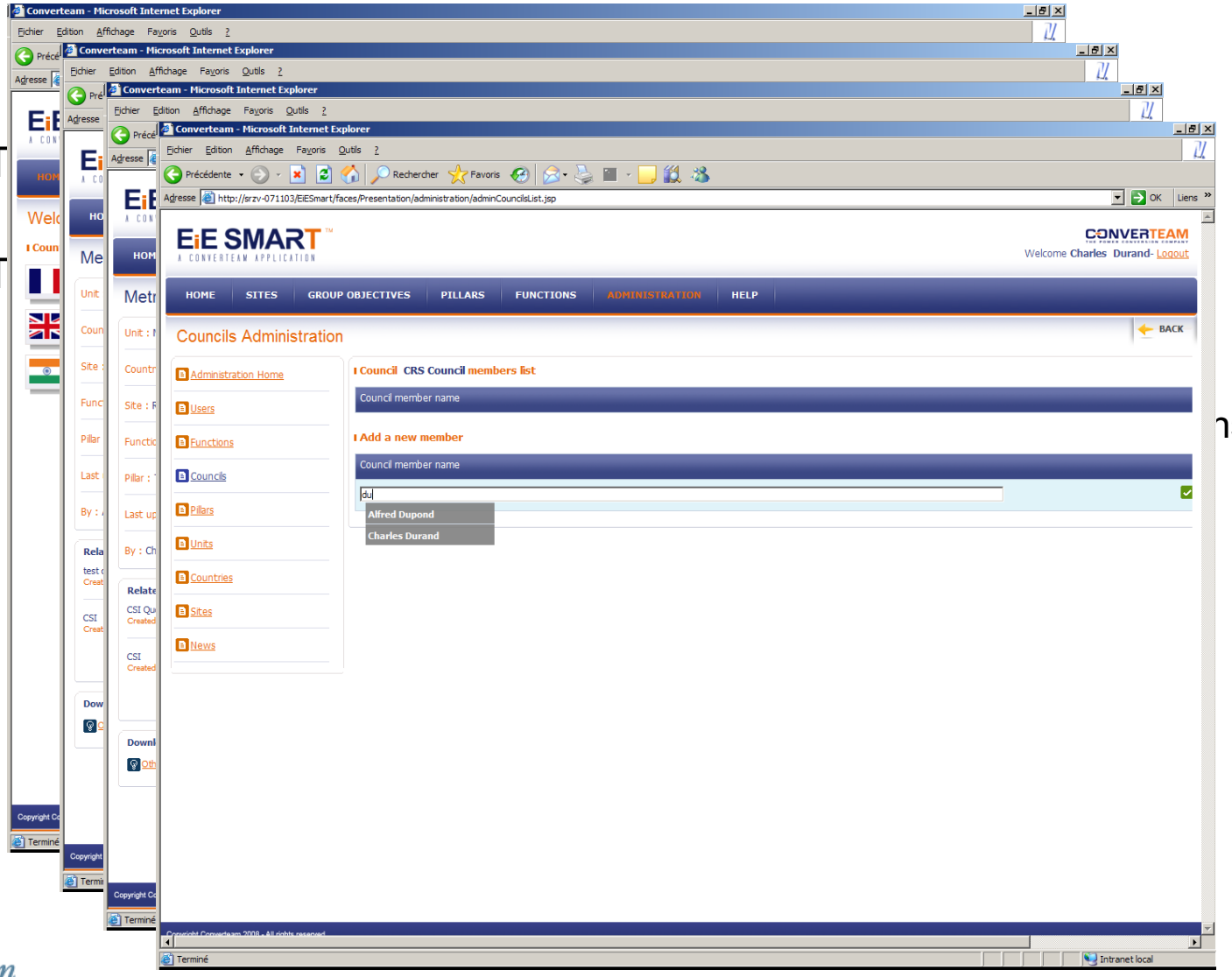
IV.I The CIM (Computed Independent Model): getting business requirements



IV.I The PIM: business process models

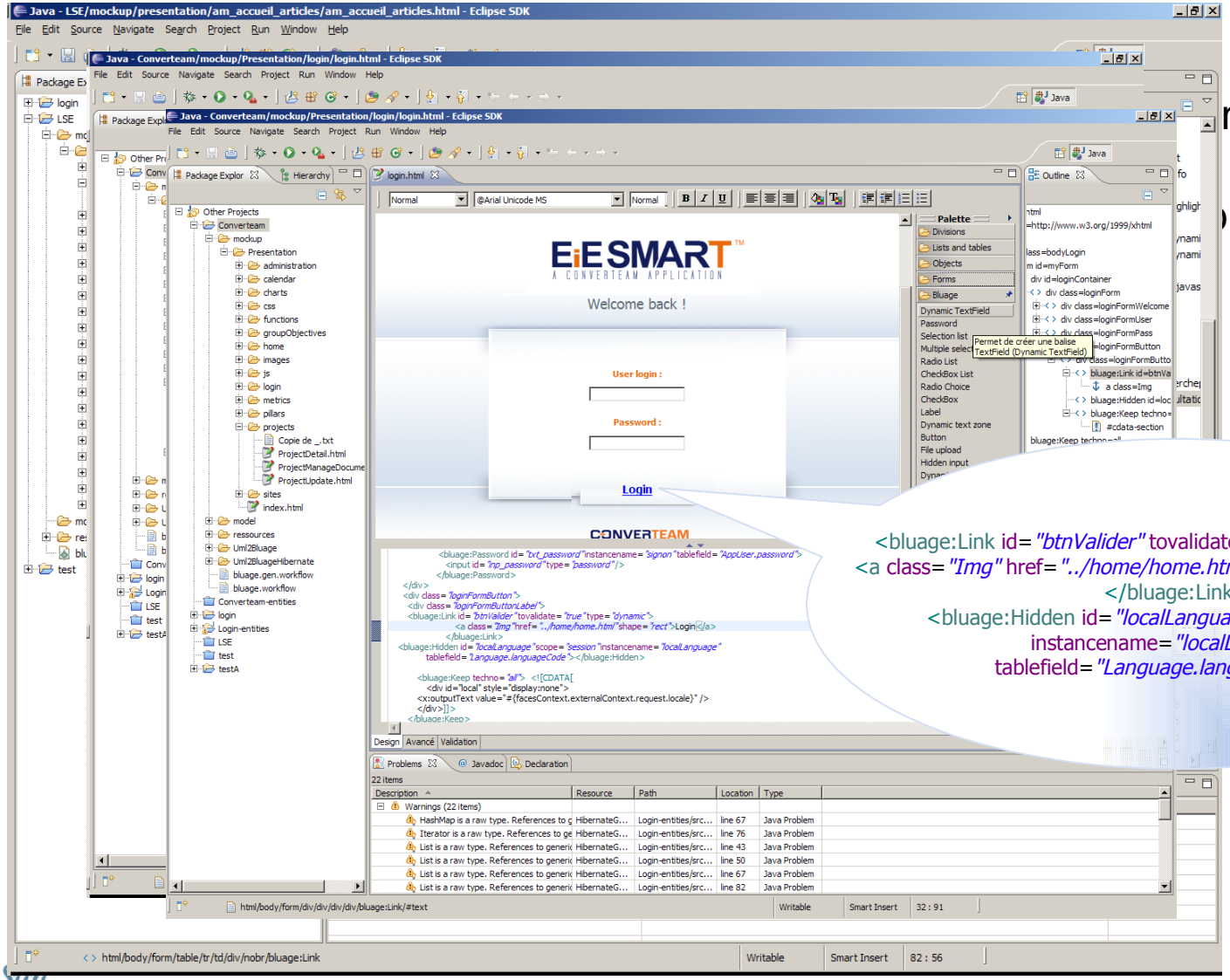


IV.I The Mock-up



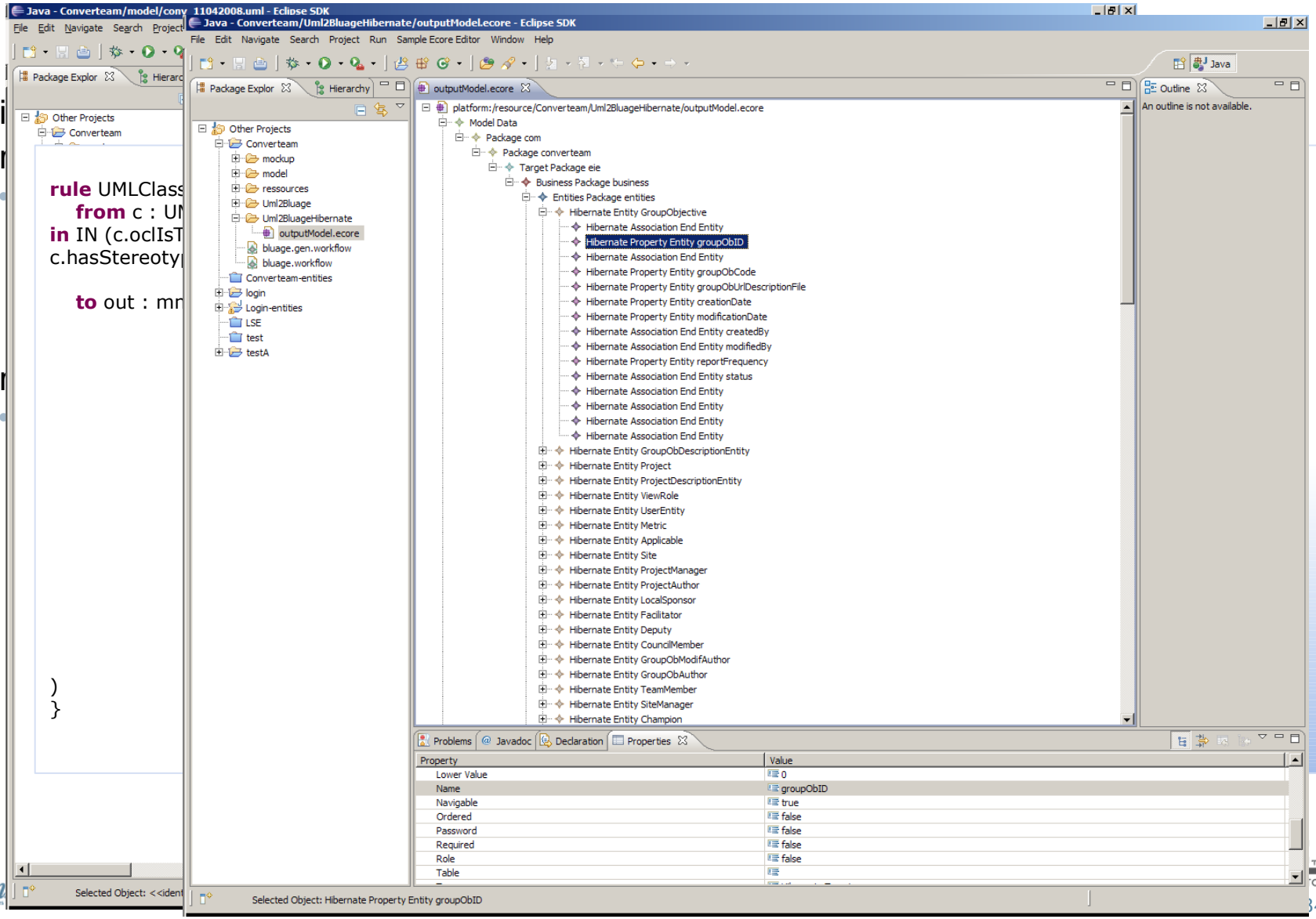
nstances

IV.I Tagging the XHTML to make it dynamique



- n data &
- letion
- models

IV.I UML to PSM

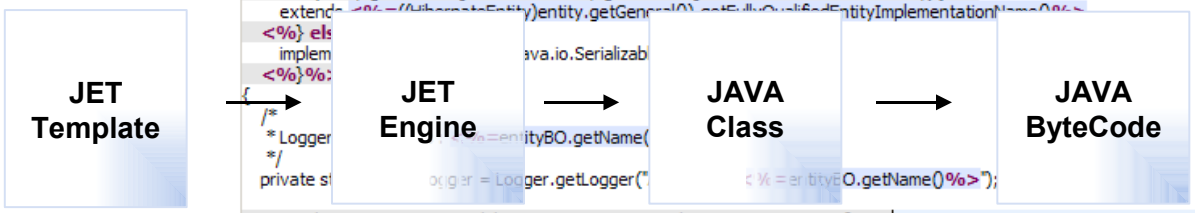


IV.II JET


Document Microsoft Word

- Is a template eng
- Templates:
  - Are composed
  - Generate ready
  - Insert code into

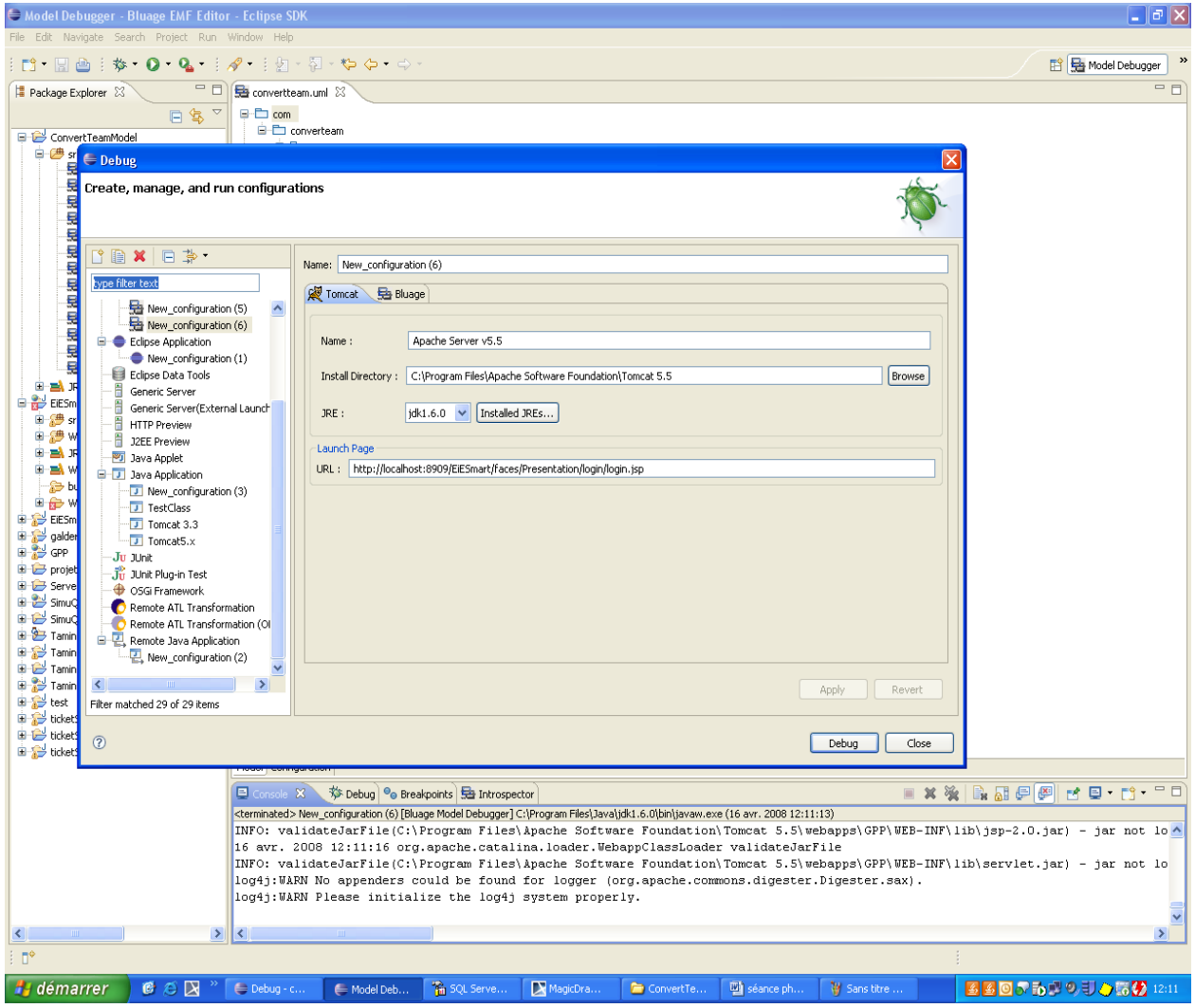
```
   String packageImport = entityBO.getPackageName();
   String entityPackageImport = entity.getPackageName();
%>
package <%=packageImport%>;
<%
   ArrayList<Set<String>> arr = HibernateUtils.getListeimport(entity);
   Set<String> sTypePrim =  arr.get(0);
   Set<String> sType =  arr.get(1);
   for(Iterator<String> it = sTypePrim.iterator(); it.hasNext();) {%>
import <%=(String) it.next()%>;
   <%} for(Iterator<String> it = sType.iterator(); it.hasNext();) {%>
import <%=(String) it.next()%>;
   <%}%>
import java.io.Serializable;
import java.util.List;
import java.util.Hashtable;
import java.util.Iterator;
import java.util.Set;

import org.hibernate.*;

import <%=com.netfective.bluage.bsp.j2ee.hibernate.utilities.Properties.fwkPackage%>.core.persistence.HibernateUtil;
import <%=com.netfective.bluage.bsp.j2ee.hibernate.utilities.Properties.fwkPackage%>.core.exception.ApplicationException;
import <%=com.netfective.bluage.bsp.j2ee.hibernate.utilities.Properties.fwkPackage%>.core.business.IBusinessObject;
import <%=com.netfective.bluage.bsp.j2ee.hibernate.utilities.Properties.fwkPackage%>.core.translation.ocl.validation.*;

import <%=packageImport%>.daoFinder.<%=entityBO.getName()%>DAOMethodFinderImpl;
import <%=entityPackageImport%>.daoFinder.<%=entity.getName()%>DAOFinderImpl;

import org.apache.log4j.Logger;

/**
<%//=entityBO.getDocumentation("* ")%>
*/
public class <%=entityBO.getName()%>
   <% if(entity.getGeneral()!=null && entity.getGeneral() instanceof HibernateEntity){%>
      extend
   <%} el
      implem                                    ava.io.Serializab
   <%}%>
{
   /*
    * Logger                                    ityBO.getName(
    */
   private st                                   ger = Logger.getLogger("                    O.getName()%>");

   <% HibernateEntityMacros hibernateEntity = new HibernateEntityMacros();%>
      <%=hibernateEntity.generate(entity)%>

<%
      EList<Property> props = entityBO.getAttribute();
      for(int j =0; j<props.size();j++) {
         Property p = (Property)props.get(j);%>
   <%= p.getVisibility()%> <%=HibernateUtils.getTypeName(p)%> <%=p.getName()%> ;
   /**
    * Getter for <%= p.getName()%>
    * @return <%= p.getType().getName()%>
```



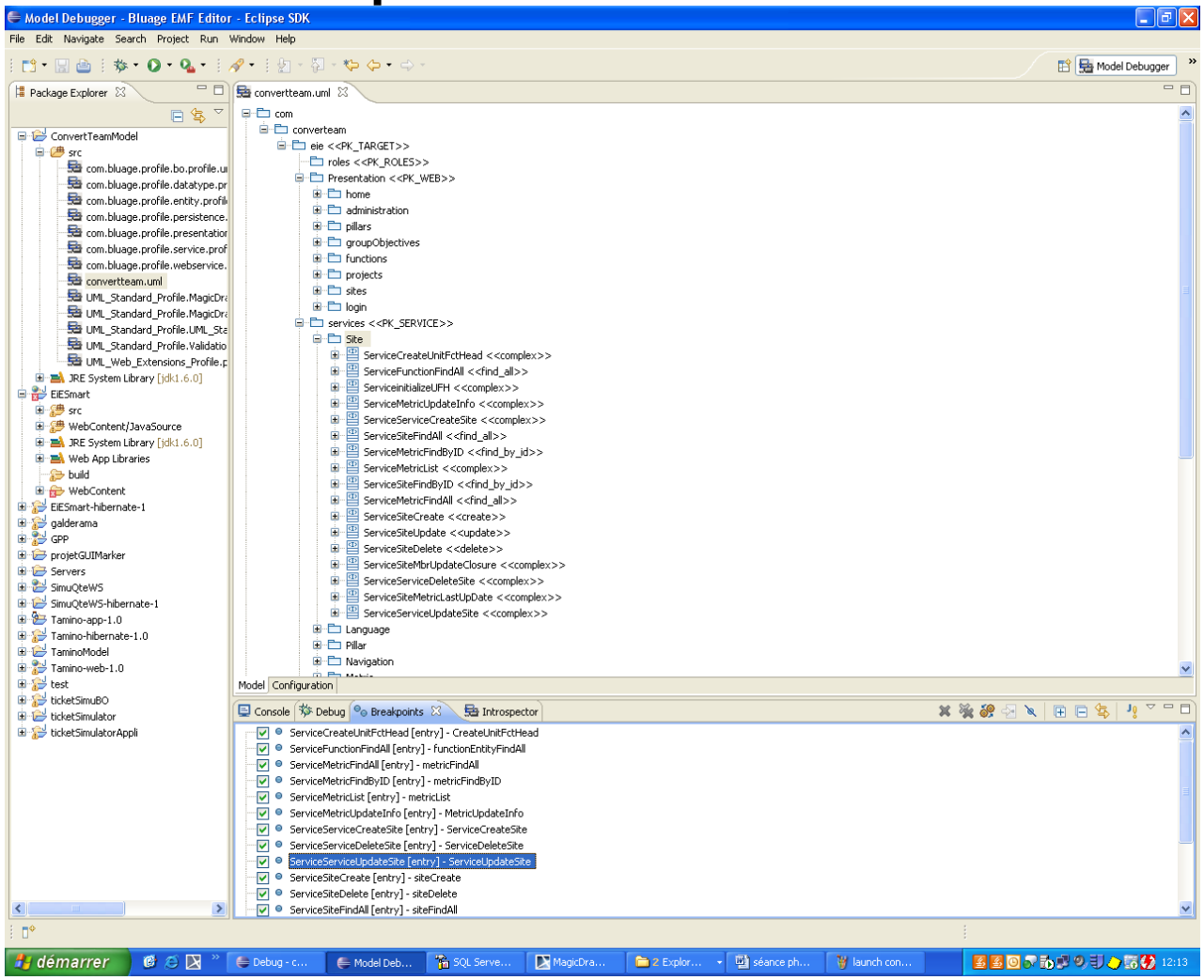
IV.II Data type mapping

- Templating is the key to support different technology, but its not enough
- Mapping are used for simple transformation : FROM <type> TO <type>
- Example of mappings UML type to :
  - JAVA™ types
  - C# types
  - SQL types
    - Oracle 10g types
    - MS-SQL 2000 types
    - MS-SQL 2005 types
    - UDB-DB2 8.x types
    - And so on
  - Wrapper types
    - Transforming primitive type into object type

IV.III Model debugging launch from Eclipse

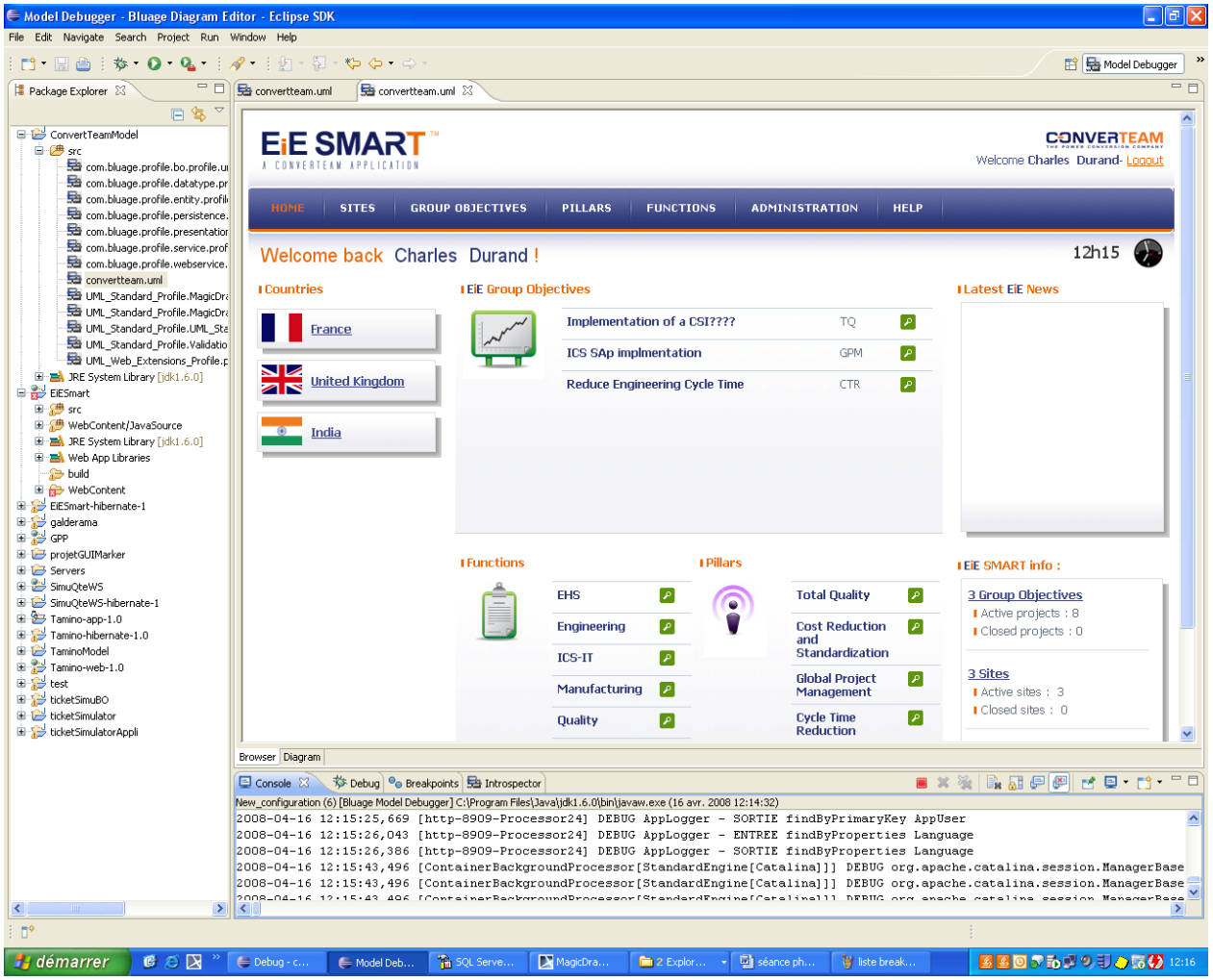


IV.III Tree view of models to define breakpoints



IV.III While executing the application,

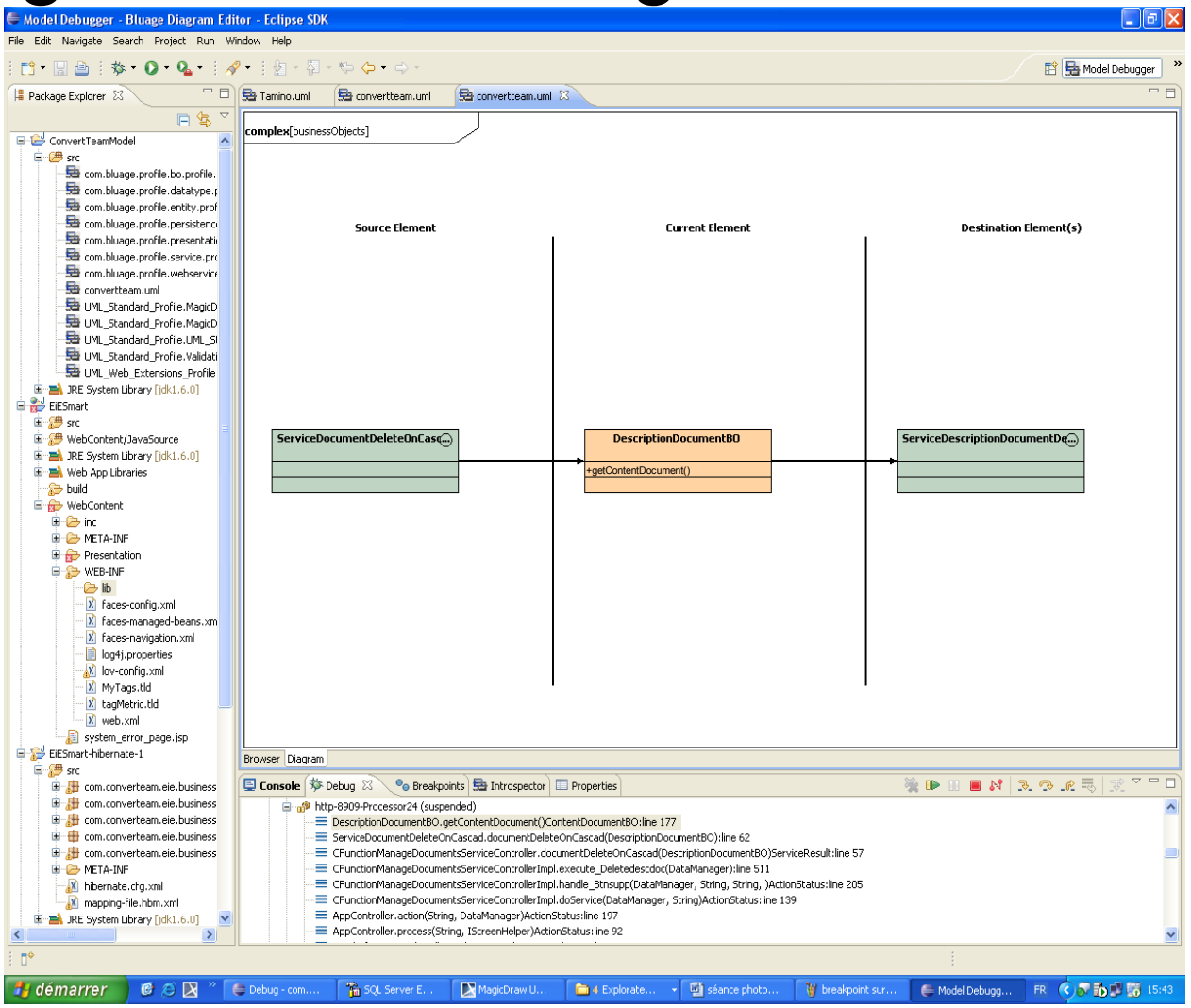
you stop the execution to jump into model debugger



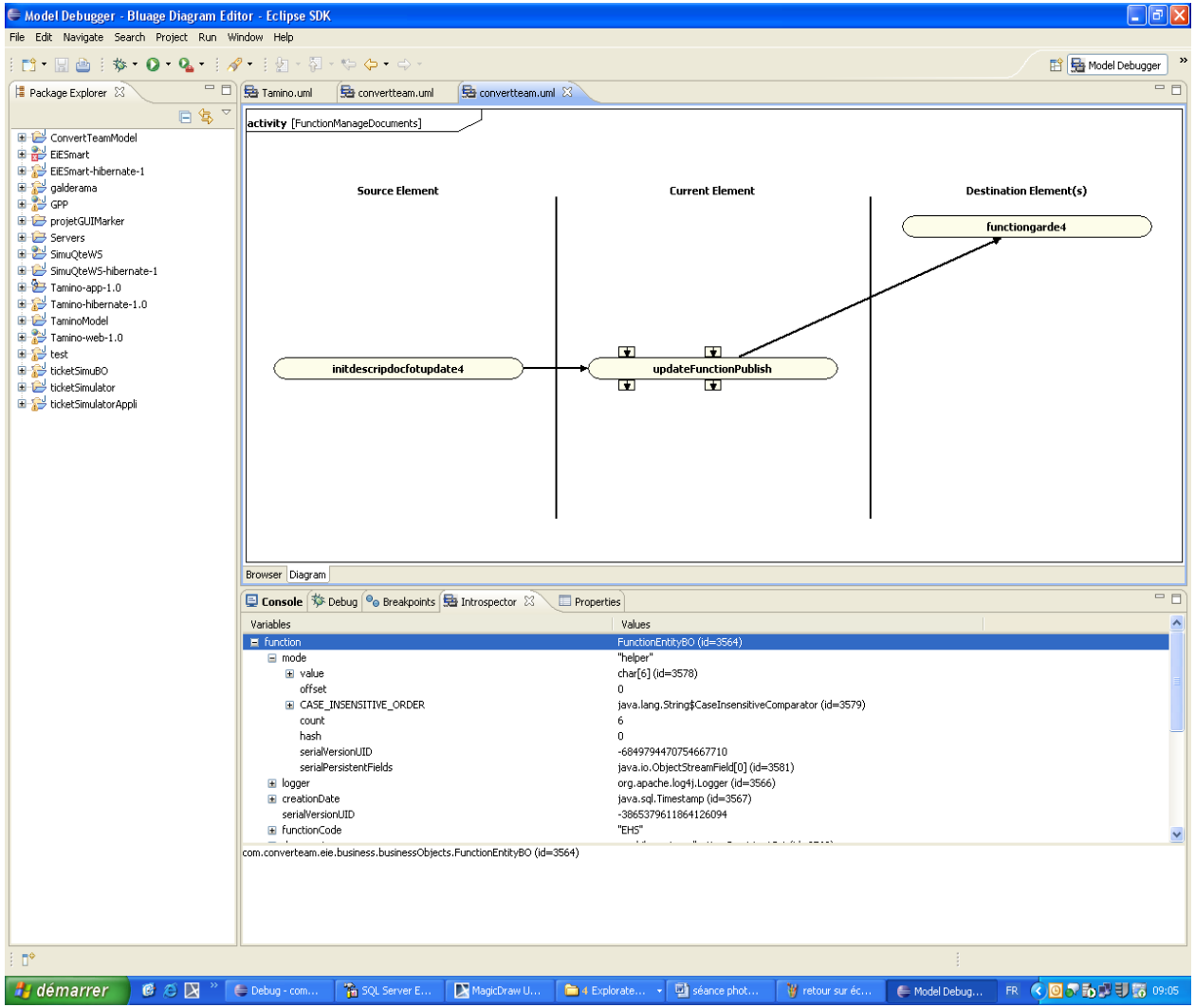
IV.III Browsing the related diagram

If need be, the user can navigate into model elements and step in for more details

Otherwise the user may trigger a "step return" to the application and resume execution



Advanced feature: instance introspection



For More Information

- Blu Age and Netfective:
  http://www.bluage.com
  Booth 118
- MDA, UML, MOF and XMI specification:
  http://www.omg.org/mda
- EMF:
  http://www.eclipse.org/modeling/emf/
- AMMA:
  http://www.sciences.univ-nantes.fr/lina/atl/
- ATL:
  http://www.eclipse.org/m2m/atl/
- JET:
  http://www.eclipse.org/emft/projects/jet/

THANK YOU

Alexis HENRY
Engineering Director

BLU AGE™
JAVA EE .NET APPLICATION GENERATOR