

Designing Tools for Conceptual Modelling and Code Generation

Vicente Pelechano

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia



Contents

- Automated Software Production Methods. Elements and Technology
- Defining Modelling Languages and Repositories
- Designing and Implementing Visual Editors
- Model to Model Transformation
- Model to Code Transformation

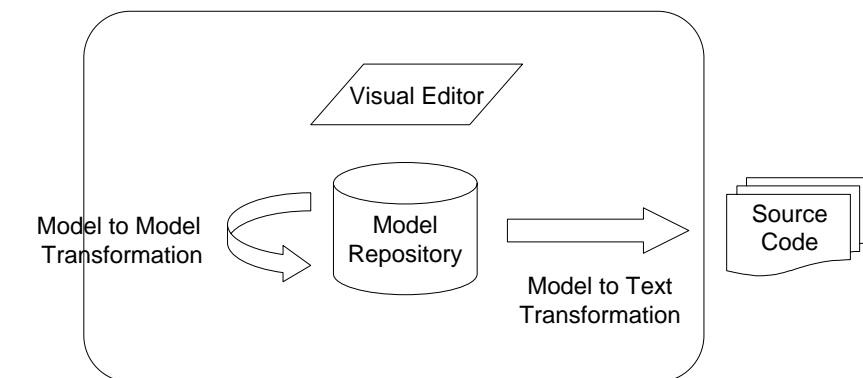


Automated Software Production Methods

- MDA and Software Factories promote the MDSD.
- The development and wide adoption of CASE Tools that provide Code Generation from Models is the most important requirement to confirm the success of the MDSD.
- Main actors in the Software Development Industry are aware that they must provide technology and tools that allow building and/or adapting advanced CASE Tools to completely support MDSD.
- In this context, IBM and Borland, and other companies, (on the one hand) and Microsoft (on the other hand) are building advanced tools that are going to make easy the development and deployment of MDSD.



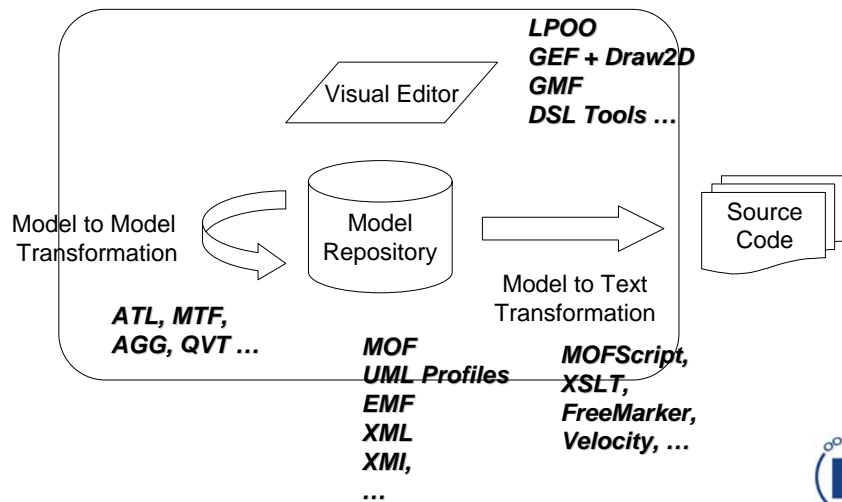
Automated Software Production Methods Elements



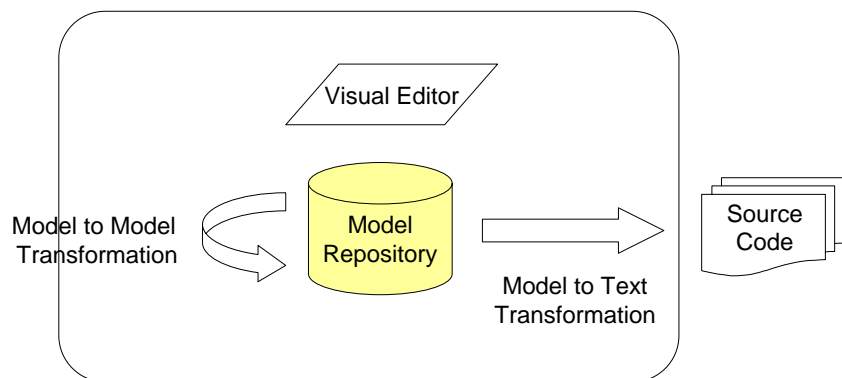
*Supporting MDA Transformations
PIM to PIM or PIM to PSM*




Automated Software Production Methods Technology



Defining Modelling Languages and Repositories





Defining Modelling Languages and Repositories

- MOF
- UML “Profiles”
- EMF (eCORE)
- XMI
- DSL Tools. Domain Model Editor

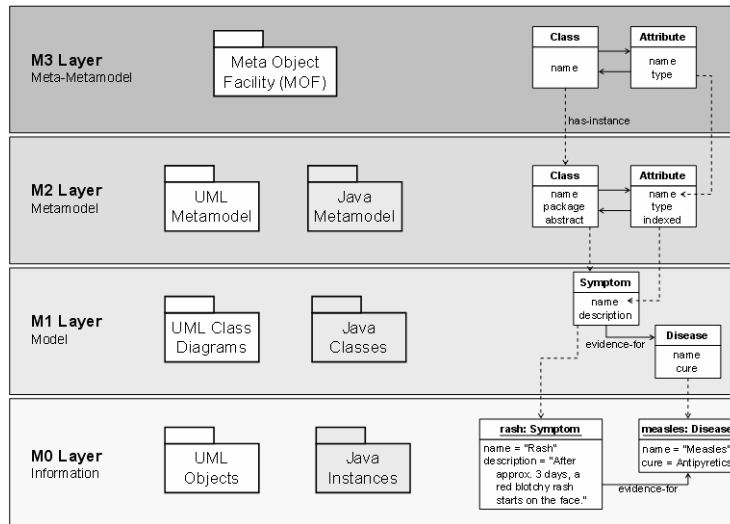


Meta Object Facility (MOF)

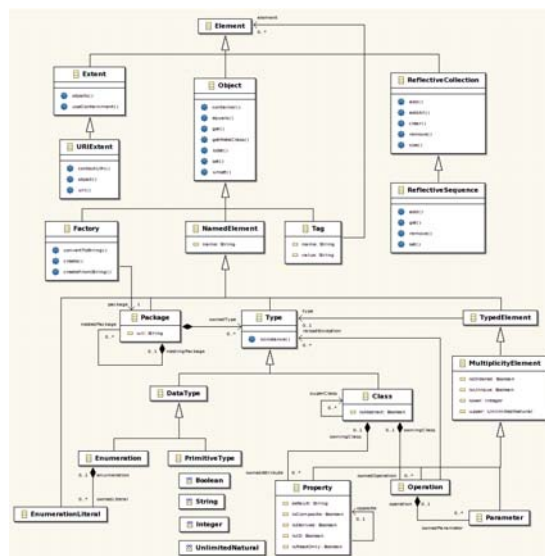
- **Objective:** “Standard” Language to specify metamodels
- Subset of UML
- Used to define OMG metamodels
- Current Version: 2 *(based on UML 2)*



Meta Object Facility (MOF)



Meta Object Facility (MOF)



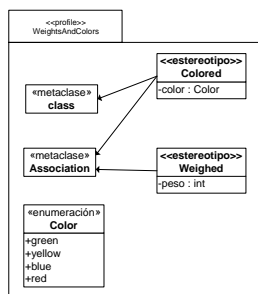
MOF. EMOF (Essential MOF) and CMOF (Complete MOF)

- MOF 2.0 is built based on a subset of the UML 2.0 infrastructure (the package `Core::Basic`) that provides the main concepts and the graphical notation for its models.
- MOF 2.0 is divided in two main packages, **Essential MOF (EMOF)** and **Complete MOF (CMOF)**.
- The goal of EMOF is to allow the definition of simple metamodels using simple concepts without losing the possibility of expressing more complex models.
- To express complex models CMOF is provided. CMOF is built from EMOF and the `Core::Constructs UML2` package.



UML Profiles

A **UML profile** is a grouping of UML modeling elements that have been adapted for a specific purpose.





UML Profiles

- The main **goal** of the profiles is:
 - Provide a direct mechanism to **adapt** an existing metamodel with constructors that are specific of a concrete domain, platform or method.
- **It is not possible** to **delete** any constraint in the UML metamodel, but it is **possible** to **add** new constraints that are specific of the profile.
- The semantics of UML elements cannot be changed.
- The **<<Profiles>>** UML **package** contains mechanisms that allow extending the metaclasses of existing metamodels to adapt them. In this package the following elements are defined:
 - stereotype
 - extension



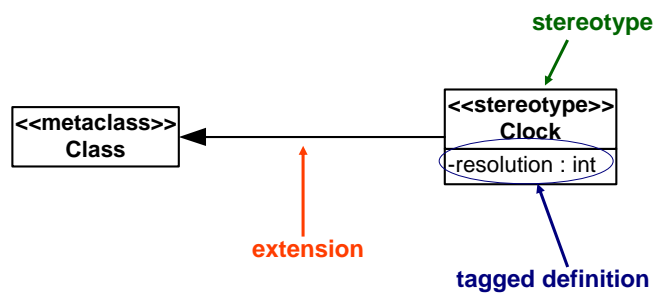
UML Profiles

- Elements used in the definition of profiles:
 - **Stereotype**. Defines how a UML metaclass can be extended.
 - **Extension**. It is used to indicate that the properties of a metaclass are extended through a stereotype.
 - **Tag Definition**. Just like a class, a stereotype may have properties, which may be referred to as tag definitions.
 - **Constraint**. Can be attached to any modeling element in order to redefine its semantics.



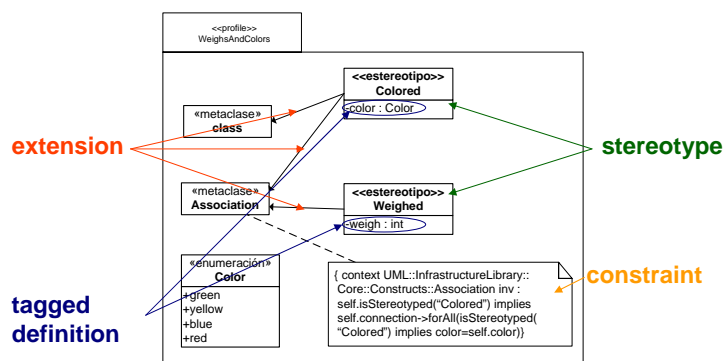
UML Profiles

■ Simple Example of a UML Profile



UML Profiles

■ A more complex Example



UML Profiles

■ Stereotypes:

- **<<Colored>>** Provides a color to a UML element. Only classes and associations can be colored.

- **Tagged Definition:** **color**, of Color type (Enumeration defined in the profile). Indicates the color of the classes and associations that have been labeled as Colored.

- **<<Weighed>>** Provides a weigh to a UML element. Only associations can be attached a weigh.

- **Tagged Definition :** **weigh**, of integer type. Indicates the weigh of each association that has been stereotyped as Weighed.

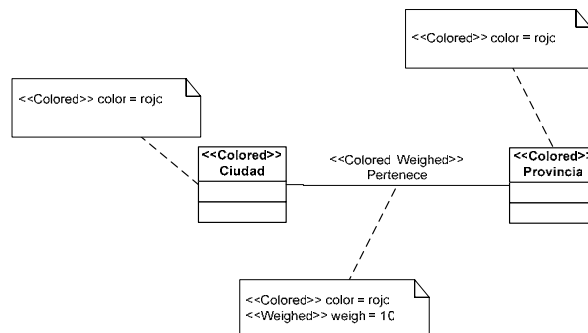
■ Constraint over the Association:

- *"If two or more classes are related through a colored association, the color of both classes must be the same as the one of the association".*



UML Profiles

■ WeighsAndColors Profile Instantiation





Eclipse Modelling Framework

- EMF is a modeling and code generation framework to build tools and other applications that are based on structured data model. Too generic?
- EMF is a set of Eclipse plugins.



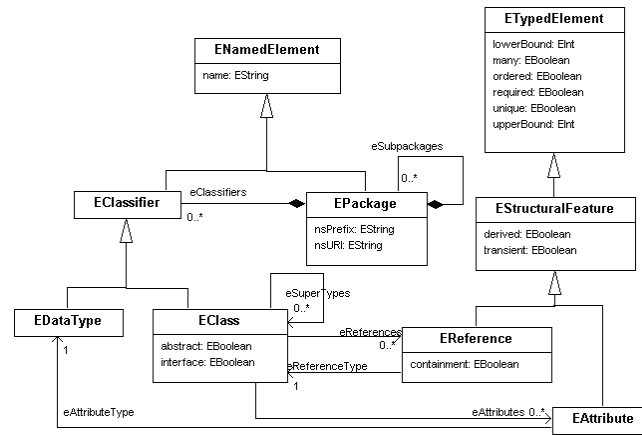
(... If look inside)

- EMF is Java implementation of the **Ecore** metamodel
 - **Ecore** is a light version of MOF. Ecore is oriented to simplicity and practicality.
 - **Ecore** designers have contributed to the specification of **Essential MOF**. This is the reason because both are similar.

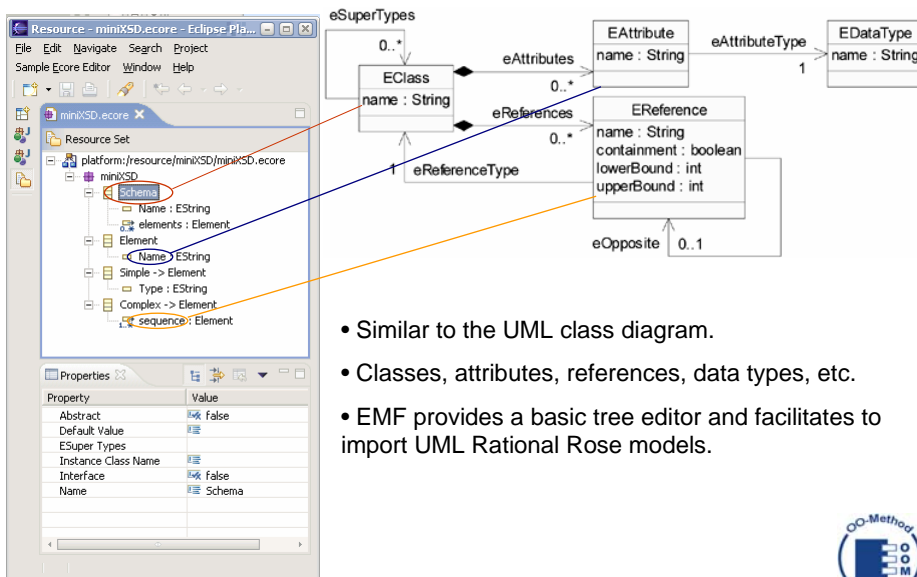


Ecore

ECORE



Basic Editor of Ecore Metamodels

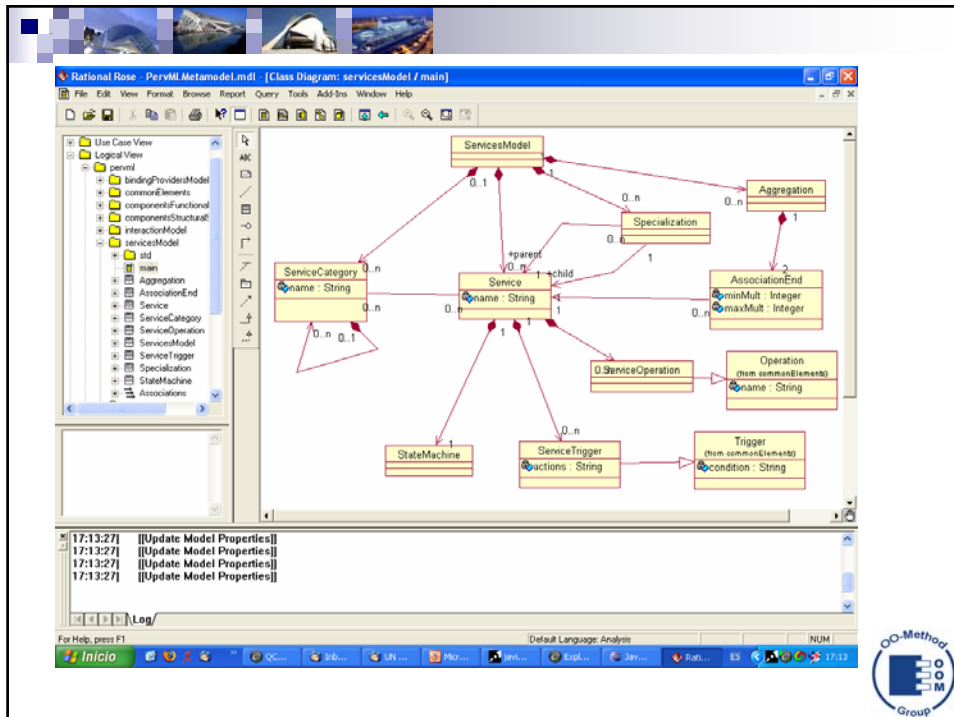


- Similar to the UML class diagram.
- Classes, attributes, references, data types, etc.
- EMF provides a basic tree editor and facilitates to import UML Rational Rose models.



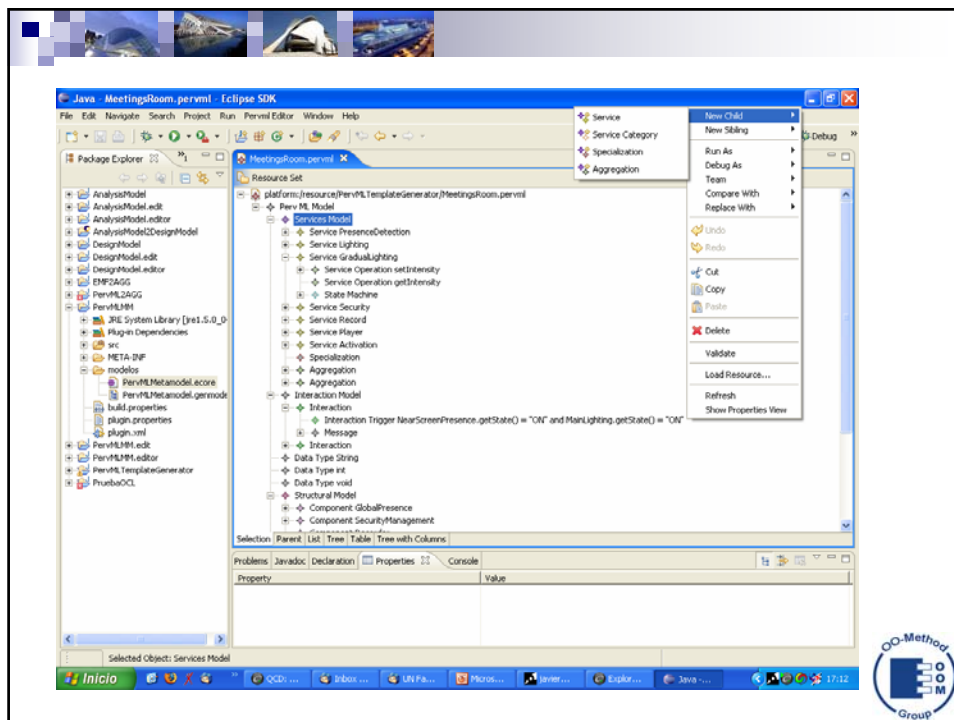
EMF. Building an Editor

1. Specify the Metamodel using:
 1. Rational Rose
 2. XML Schema
 3. Annotated Java
2. Import to EMF



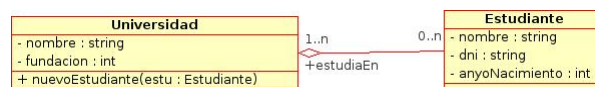
Code Generation

- Given a Metamodel definition EMF provides tools and execution support.
 - For Implementing the model
 - Structured Java code
 - Model-in-code
 - XMI and XML Schema Serializers
 - Classes that facilitate the development of (textual or graphical) editors
 - Generates a Basic Editor in a Tree View Form



XML Metadata Interchange (XMI)

- Rules to save MOF models using XML
- **Objective:** Share Models
- Example:



XML Metadata Interchange (XMI)

```
<UML:Class visibility="public" xmi.id="11" isAbstract="false" name="Universidad" >
  <UML:Classifier.feature>
    <UML:Attribute visibility="private" xmi.id="13" initialValue="" type="10" isAbstract="false"
      name="nombre" />
    <UML:Attribute visibility="private" xmi.id="14" initialValue="" type="2" isAbstract="false"
      name="fundacion" />
    <UML:Operation visibility="public" xmi.id="15" isAbstract="false" name="nuevoEstudiante" >
      <UML:BehavioralFeature.parameter>
        <UML:Parameter visibility="private" xmi.id="17" value="" type="12"
          isAbstract="false" name="estu" />
      </UML:BehavioralFeature.parameter>
    </UML:Operation>
  </UML:Classifier.feature>
</UML:Class>
```



XML Metadata Interchange (XMI)

```
<UML:Association visibility="public" xmi.id="21" name="" >
  <UML:Association.connection>
    <UML:AssociationEnd visibility="public" isNavigable="true" xmi.id="22" aggregation="shared"
      type="11" name="estudiaEn" multiplicity="1..n" />
    <UML:AssociationEnd visibility="public" isNavigable="true" xmi.id="23" aggregation="none"
      type="12" name="" multiplicity="0..n" />
  </UML:Association.connection>
</UML:Association>
```



XML Metadata Interchange (XMI)

- Several versions (1.X, 2.X)
- A model is saved using:
 - A specific XMI version
 - A specific metamodel version
- Example: XMI 1.2 / UML 1.4
- ¡Interoperability problems!



XMI 1.1 corresponds to MOF 1.3
XMI 1.2 corresponds to MOF 1.4
XMI 1.3 (added Schema support) corresponds to MOF 1.4
XMI 2.0 (adds Schema support and changes document format) corresponds to MOF 1.4
XMI 2.1 corresponds to MOF 2.0

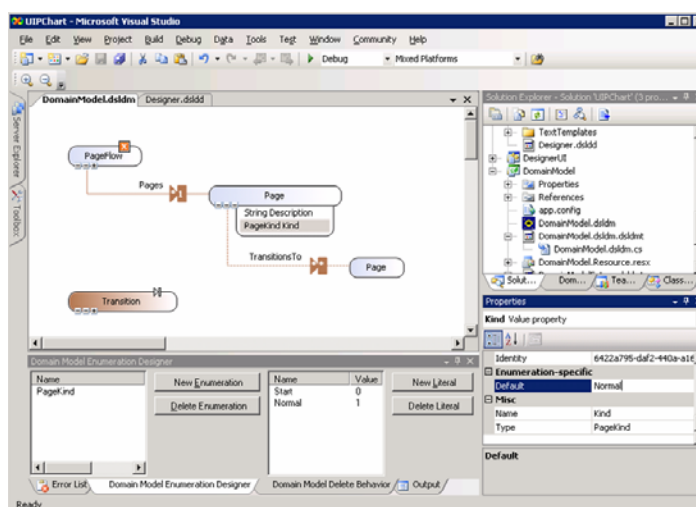


DSL Tools. Domain Model Editor

- Microsoft DSL Tools provide a graphical designer to define domain languages.
- Allows defining the metamodel in a proprietary notation and uses XML files as the persistence mechanism.



DSL Tools. Domain Model Editor

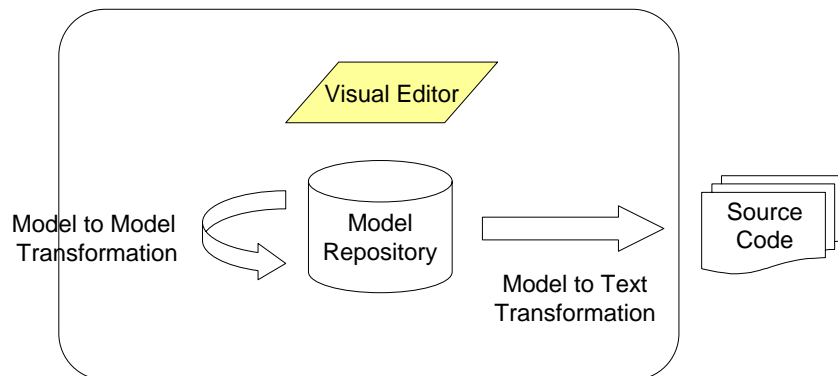



DSL Tools. Domain Model Editor

The screenshot displays the Domain Model Editor interface. The main window shows XML metadata for a domain model, including concepts like PageFlow and Page, and their relationships. The Domain Explorer on the right shows a tree structure of the model, including classes like PageFlow, Page, and PageFlowHasPages, and their properties and roles.

```
<mdfmetadata xsi:type="rolemdfmetadata" accessmodifier="public" category="" descr...>
<source>i0f6e65d0z95b3z47d6zbd11z2470fb37ee28</source>
<type>i0f6e65d0z95b3z47d6zbd11z2470fb37ee28</type>
<generatedProperty name="Source" identity="5a6c4c2a-3b62-4a9f-9f38-5284435dd226">
  <referenceType>i0f6e65d0z95b3z47d6zbd11z2470fb37ee28</referenceType>
</generatedProperty>
</role>
</roles>
</relationship>
</relationships>
<trees>...
<concepts>
  <concept name="PageFlow" identity="5a0416f2-879f-4903-a9ec-845c980f6ad3" namespace="Fal...
  <mdfmetadata xsi:type="conceptorshapemdfmetadata" accessmodifier="public" category="...
  </concept>
  <concept name="Page" identity="0f6e65d0-95b3-47d6-bd11-2470fb37ee28" namespace="Fabrik...
  <mdfmetadata xsi:type="conceptorshapemdfmetadata" accessmodifier="public" category="...
  <valueProperties>
    <property name="Description" identity="6833aced-ad8a-4fe4-867f-0ca14d26c4d9" id="i...
    <mdfmetadata xsi:type="propertymdfmetadata" accessmodifier="public" category="...
    <propertyInfo xsi:type="StringProperty...
  </property>
  </valueProperties>
</concept>
```

Designing and Implementing Visual Editors





Designing and Implementing Visual Editors

- Manual Programming
- GEF+Draw2D
- GMF
- DSL Tools. Model Designer



Manual Programming

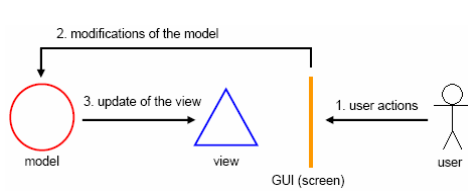
- A lot of LOC
- Low productivity at a High Cost
- Commercial Components and Frameworks exist



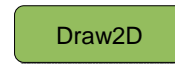
Graphical Editing Framework (GEF)



- Eclipse extension to develop graphical editors of modeling languages
- Infrastructure to develop the *Controller* component in a Model-View-Controller Framework



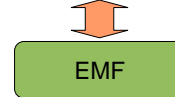
View



Controller



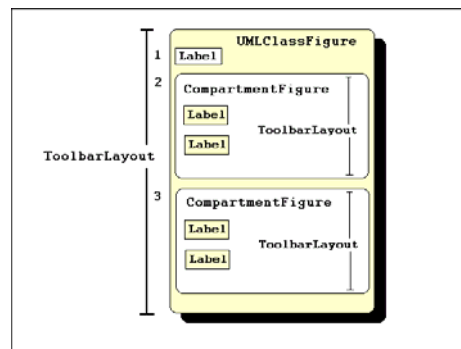
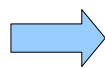
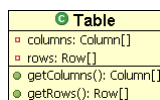
Model



Draw2D



- Integrated in GEF
- Library to draw graphical elements
 - Figures, Labels, Layout Managers, Borders, etc.

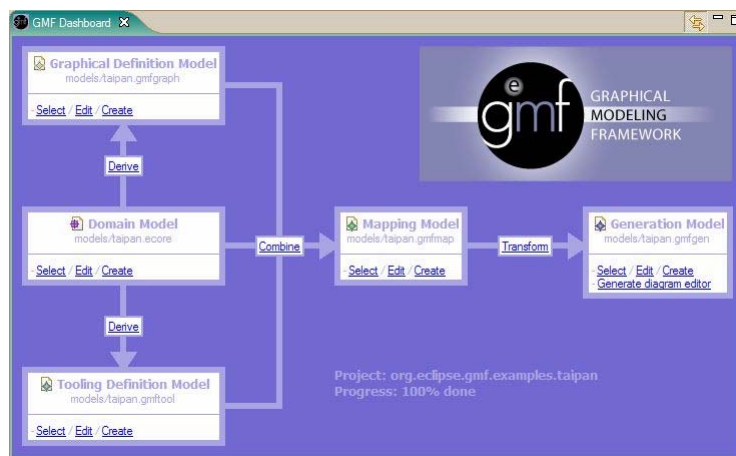


GMF

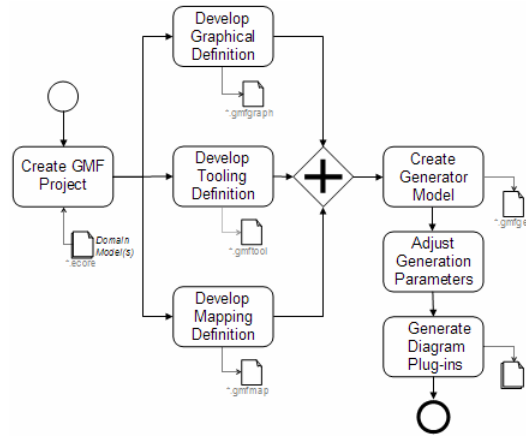
- Graphical Modeling Framework (GMF) is an Eclipse project that pretends to provide a bridge between EMF and GEF. GMF generates automatically the GEF code that allows to build a graphical editor of EMF models.



GMF



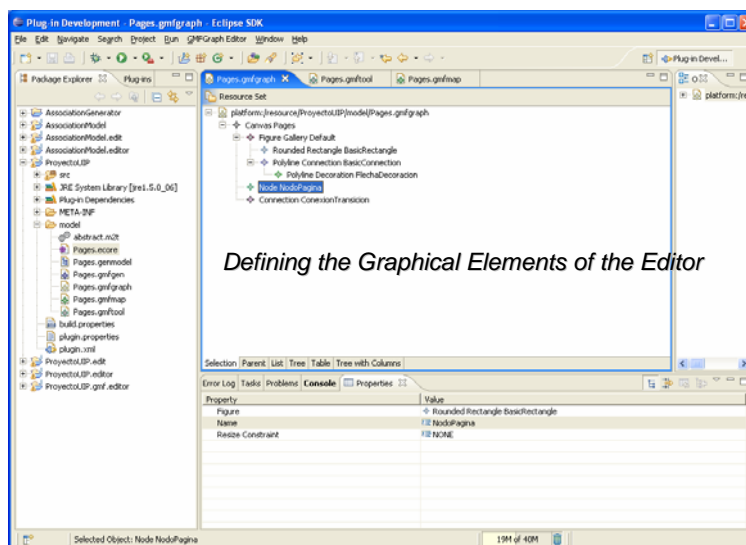
GMF



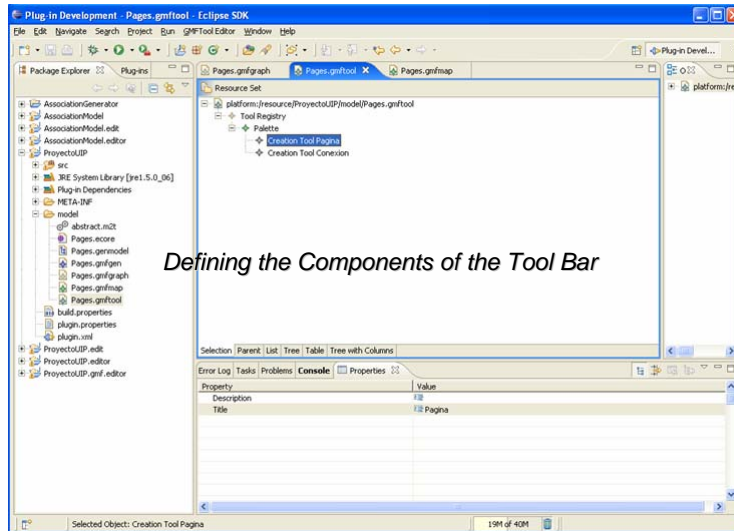
*Three Basic Steps:
Graphical Definition, Tooling Definition and Mapping Definition*



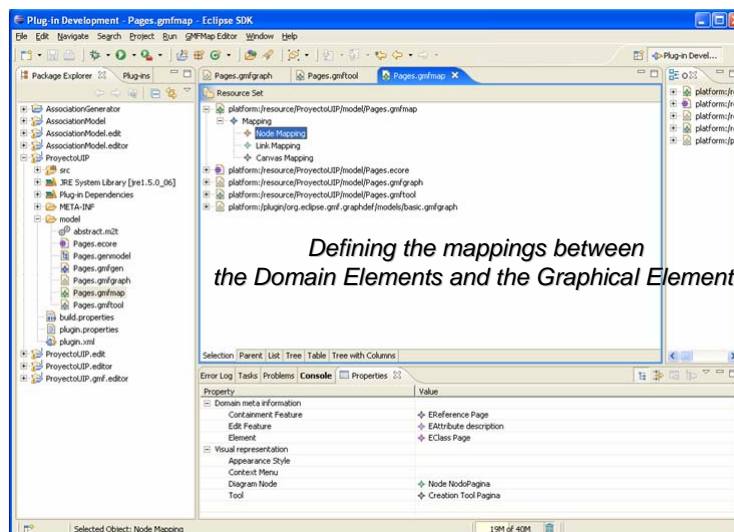
GMF



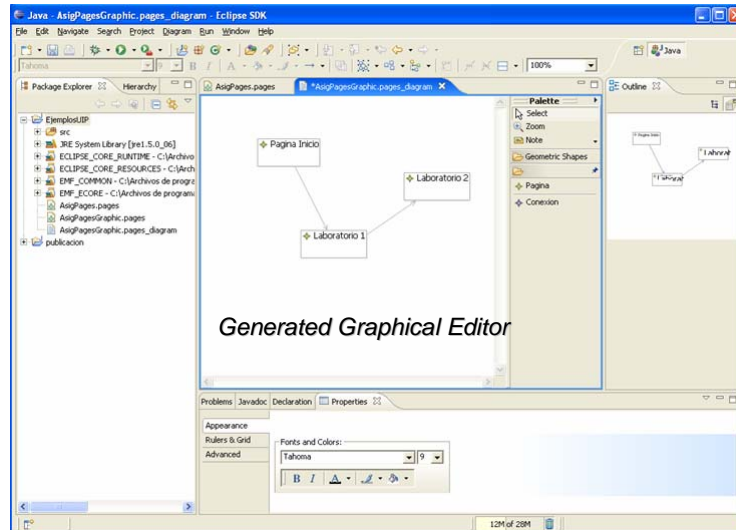
GMF



GMF



GMF



DSL Tools. Designer Definition

- A “designer definition” is used with a domain model (language) to build a “designer” or graphical editor of the DSL primitives.
- Designer Definitions are specified in a XML format
- The XML document includes the definition of:
 - The diagrams, including graphical shapes and connectors.
 - Mappings of the diagrams (shapes and connectors) to the elements of the underlying domain model (or DSL).



DSL Tools. Model Designer

```
DomainModel.dslm Designer.dsldd
<?xml version="1.0"?>
<designerDefinition namespace="Fabrikam.TestUseCase.TestUC.Designer" name="TestUC">
  <explorer>...
  <notation>...
  <ObjectModels>...
  <propertiesWindow>...
  <validation open="false" save="false" menu="false" custom="false"/>
  <validationBehavior>...
</designerDefinition>
```

XML Structure



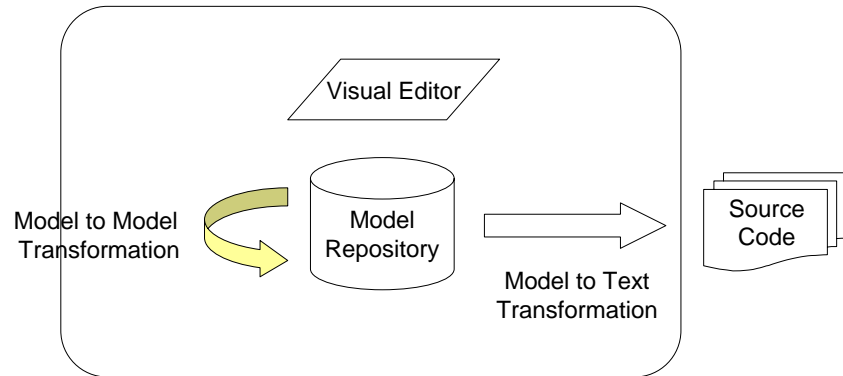
DSL Tools. Model Designer

```
<shapes>
  <imageShape name="ActorShape" imageId="Actor">
    <decorators>
      <shapeText name="Name" position="InnerBottomCenter"
        defaultTextId="ActorNameDecorator"/>
    </decorators>
  </imageShape>
  <geometryShape name="CommentShape" initialWidth="1.5"
    initialHeight="0.3" geometry="Rectangle">
    <decorators>
      <shapeText name="Comment" position="Center"
        defaultTextId="CommentShapeCommentDecorator"/>
    </decorators>
    <fillColor color="khaki" variability="User"/>
    <outlineColor color="brown" variability="Fixed"/>
  </geometryShape>
  <geometryShape geometry="Ellipse" initialHeight="0.6"
    initialWidth="1.4" name="UseCaseShape">
    <decorators>
      <shapeText name="Name" position="Center"
        defaultTextId="UseCaseNameDecorator"/>
    </decorators>
    <fillColor color="White" variability="User"/>
    <outlineColor color="Gray" variability="Fixed"/>
  </geometryShape>
  <geometryShape geometry="Rectangle" initialHeight="3"
    initialWidth="2"
    name="ContainerShape">
    <decorators>
      <shapeText name="Name" position="InnerTopCenter"
        defaultTextId="ContainerNameDecorator"/>
    </decorators>
    <fillColor color="AliceBlue" variability="User"/>
    <outlineColor color="Gray" variability="Fixed"/>
  </geometryShape>
```

Shapes Definition



Model to Model Transformations



Model to Model Transformations


- Manual Programming
- MTF
- AGG
- ATL
- QVT





Manual Programming

- Read the XML or XMI File or DB Tables etc... and load into memory + apply transformations + save the results in the format of the target models
Or...
- Use a library generated by EMF to:
 1. Load the source Model
 2. Navigate, create, delete and modify elements
 3. Save the target model



MTF (Model Transformation Framework)



- IBM participated in the Request For Standards de OMG about MOF 2.0 Query/View/Transformation (QVT) (2004).
- IBM has developed MTF as a prototype that implements some concepts of QVT and it is based on EMF.
- Provides:
 - A declarative language to define mappings between models
 - A transformation engine that can interpret the mappings and apply the transformations
 - An Eclipse Plugin

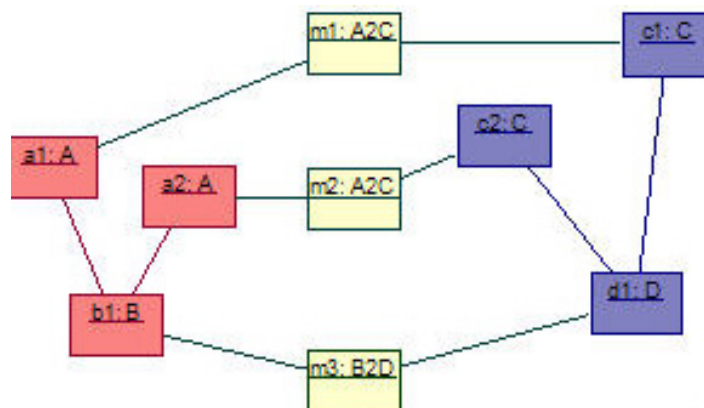


MTF (Model Transformation Framework)

- The general idea behind the term model transformation is to produce models from other models given as input, according to predefined relationships between elements.
- It assumes that these models are described by compatible meta-models, in order to express these correspondences in a consistent way. In the scope of QVT, model transformation relates to MOF models. MTF applies these concepts to EMF models.



MTF (Model Transformation Framework)



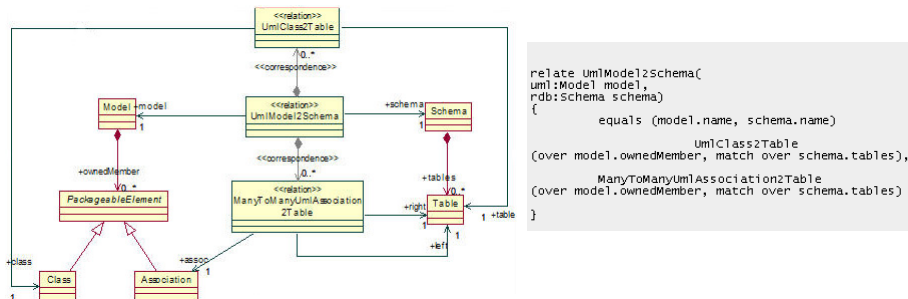
MTF (Model Transformation Framework)

- Transformations in MTF are defined in a declarative way: you specify a set of relations between model classes, and then let the MTF engine perform the transformation actions using these relations as input.
- The relations that drive the transformations are expressed in a language called the **Relation Definition Language (RDL)**.
- RDL allows the definition and application of relations between classes, based on correspondences between structural feature.



MTF. Example: Transforming the UML Class Diagram to a Relational Design

- Mapping a model to a schema



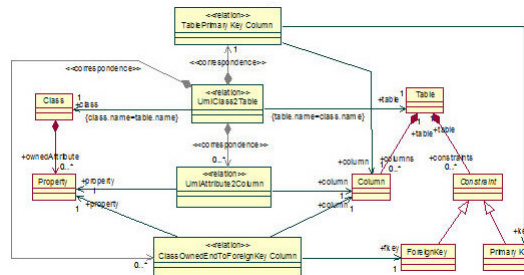
```

relate UmlModel2Schema(
  uml:Model model,
  rdb:Schema schema)
{
  equals (model.name, schema.name)
  UmlClass2Table
  (over model.ownedMember, match over schema.tables),
  ManyToManyUmlAssociation2Table
  (over model.ownedMember, match over schema.tables)
}
  
```



MTF. Example: Transforming the UML Class Diagram to a Relational Design

■ Mapping a class to a table



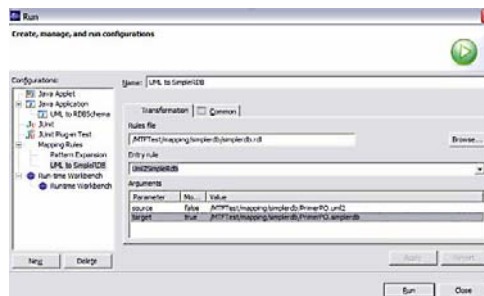
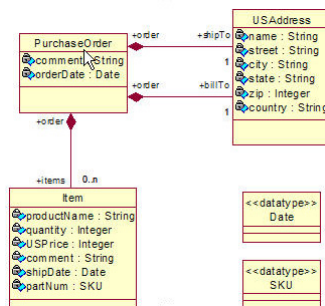
```

relate UmlClass2Table(
    uml:Class class,
    rdb:Table table)
when equals(class.name, table.name)
{
    TablePrimaryKeyColumn [1] (match over table.columns,
over table.constraints),
    UmlAttribute2Column(over class.ownedAttribute,
match over table.columns),
    ClassOwnedEnd2ForeignKeyColumn(over class.ownedAttribute,
match over table.columns, over table.constraints)
}
    
```



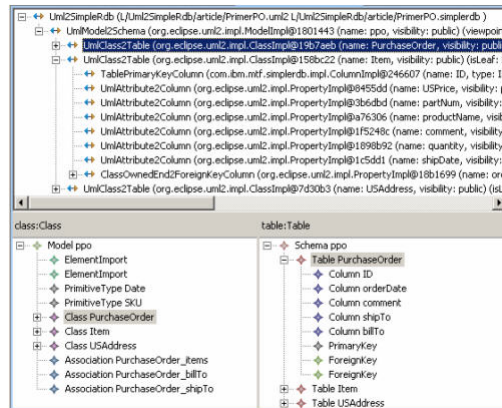
MTF. Example: Transforming the UML Class Diagram to a Relational Design

■ Running the Transformations



MTF. Example: Transforming the UML Class Diagram to a Relational Design

■ Mappings Generated by MTF



AGG

■ The Attributed Graph Grammar System

□ <http://tfs.cs.tu-berlin.de/agg/>

■ Allows defining model transformations

■ Based on graph grammars

■ Advantages:

- Intuitive Editor (Implemented in Java)
- Transformations are graphically defined
- Provides libraries to implement our own generator

■ Drawbacks:

- The Source model must be represented as a graph in a concrete format
- The Target model is obtained as a graph
- The transformation algorithm is inefficient (slow)



AGG

■ Graph Grammar:

- Set of transformation rules
 - Left Hand Side (LHS)
 - Non Application Condition (NAC)
 - Right Hand Side (RHS)
- Source Graph (Host Graph)

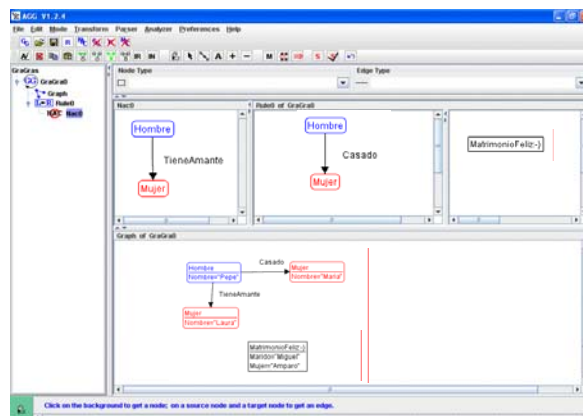
- When a LHS is detected in the source graph and the NAC does not hold, the LHS is substituted by the RHS



AGG. Example

Source Graph: Relationships between men and women

Transformation: Married Couples are happy if other Lovers do not exist



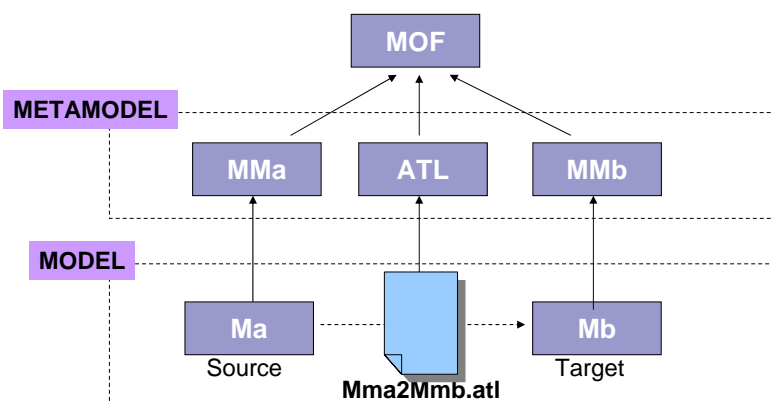
ATL

<http://www.sciences.univ-nantes.fr/lina/atl/atlProject/>

- ATL (Atlas Transformation Language)
- Hybrid Language (declarative + imperative)
- Suitable to implement PIM to PSM transformations
- It is based on the following principles:
 1. Models are first class entities
 2. The transformations are also Models
 3. Allows specialization and composition of transformations



ATL Approach



Language Definition (1/3)

■ 4 sections:

HEADER

Defines the transformation module name and declares the input and output modules.

```
module SimpleClass2SimpleRDBMS;  
create OUT : SimpleRDBMS from IN : SimpleClass;
```

- IN and OUT are variables
- More than one model can be declared as an input or output.

IMPORT

Declares libraries that are going to be imported.

```
uses strings;
```



Language Definition(2/3)

HELPER

(Global) Variables and functions can be defined.

```
helper context SimpleClass!Class def :  
  allAttributes : Sequence(SimpleClass!Attribute) =  
    self.attrs->union(  
      if not self.parent.oclIsUndefined() then  
        self.parent.allAttributes->select(attr |  
          not self.attrs->exists(at | at.name = attr.name)  
        )  
      else Sequence {}  
      endif  
    )->flatten();  
  } ATTRIBUTE  
  
helper context Book!Book def :  
  getAuthors() : String =  
    self.chapters->collect(e | e.author)->asSet()  
    ->iterate(authorName; acc : String = '' |  
      acc +  
      if acc = '' then authorName  
      else ' and ' + authorName  
      endif  
    );  
  } OPERATION
```



Language Definition(3/3)

TRANSFORMATION RULES

Defines the transformation between an input model and an output model by relating their metamodels.

```
rule Book2Publication {
  from
    b : Book!Book (
      b.getNbPages() > 2
    )
  to
    out : Publication!Publication (
      title <- b.title,
      authors <- b.getAuthors(),
      nbPages <- b.getNbPages()
    )
}
```



Transformation Types

- **Module:** Classical Transformations between models.
- **Query:** Allows querying model primitives and calculating output that shouldn't necessarily be a model. Useful to generate code or text from a model.
- **Library:** Groups a set of ATL functions that can be used in several and distinct places.



ATL + Eclipse

The screenshot displays the Eclipse IDE interface for ATL development. Red arrows point to various components: 'Debugger Variable Location' points to the 'Variables' view; 'Code Editor' points to the ATL rule editor; 'Automatic Builder Errors' points to the 'Problems' view; 'Project' points to the 'Navigator' view; and 'Outline' points to the 'Outline' view. The ATL rule editor shows a rule named 'PersistentClassTable' with a body containing ATL expressions. The 'Variables' view shows the state of variables during execution, including 'self', 'C', 'scopeName', 'name', and 'isPersistent'. The 'Navigator' view shows the project structure, including the 'SimpleClassSimpleRDEPS' project and its sub-projects.


Debugger Variable Location

Code Editor

Automatic Builder Errors

Project

Outline



MOF 2.0 Query/View/Transformation

- Defines three transformation languages:
 - **Relations** (Declarative)
 - **Core** (Declarative). Same expressive power than Relations but simpler (atomic).
 - **Operational Mappings** (Imperative)
- Core and Relations provide the same semantics but at different levels of abstraction
- Depends on two OMG specifications:
 - MOF 2.0
 - OCL 2.0

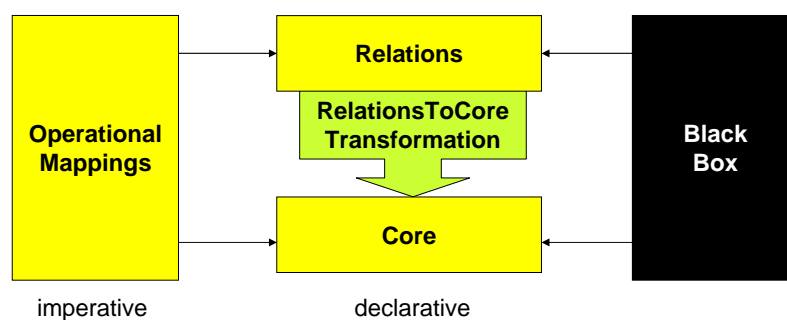


MOF 2.0 Query/View/Transformation

- **Queries** take a model as input and select specific elements from that model
- **Views** are models that are derived from other models
- **Transformations** take a model as input and update it or create a new model



QVT Metamodels Relationships



The Relations Language

- Declarative (defines the **what** and not the **how**)
- Support complex pattern-matching of objects
- Transformations are specified as a set of relationships
- Transformations can be invoked for:
 - 1) Checking the consistency between models (Checkonly Domains)
 - 2) Modifying a model to enforce the consistency (Enforced Domains)



The Relations Language

```
transformation umlRdbms (uml : SimpleUML, rdbms : SimpleRDBMS) {  
  relation ClassToTable /* maps classes to tables */ {  
    domain uml c:Class {  
      namespace = p:Package {},  
      kind='Persistent',  
      name=cn }  
    domain rdbms t:Table {  
      schema = s:Schema {},  
      name=cn,  
      column = cl:Column {  
        name=cn+'_tid',  
        type='NUMBER'},  
      primaryKey = k:PrimaryKey {  
        name=cn+'_pk',  
        column=cl }  
      }  
    when { PackageToSchema(p, s); }  
    where { AttributeToColumn(c, t); }  
  }  
  ...  
}
```

- Transformations
- Model Types
- Relations
- Domains
- When and where clauses



The Relations Language

```
transformation umlRdbms (uml : SimpleUML, rdbms : SimpleRDBMS) {
  top relation PackageToSchema {...}
  top relation ClassToTable {...}
  relation AttributeToColumn {...}
}
```

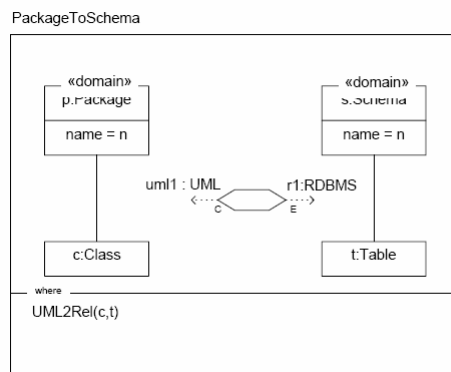
- top-level and non-top-level relations
- checkonly and enforced

```
relation PackageToSchema /* map each package to a schema */{
  checkonly domain uml p:Package {name=pn}
  enforce domain rdbms s:Schema {name=pn}
}
```



The Relations Language

- Proposes a graphical notation:



The Operational Mappings Language

- Imperative (defines the **how** and not the **what**)
- Represents the definition of a **unidirectional** transformation expressed imperatively.
- It is used to implement *Relations* when it is difficult to give a purely declarative specification of how to populate a *Relation*.



The Operational Mappings Language

```
transformation Bpmm_To_Uml;
metamodel 'http://www.borland.com/together/2005/bpmm';
metamodel 'http://www.borland.com/together/uml';
metamodel 'http://www.borland.com/together/uml20';

mapping main(in model: bpmm::BpmmProcessPool): uml::together::Model {
  init {
    var actors := model.lanes->select(lane | lane.oclIsKindOf(bpmm::BpmmLane))->asOrderedSet();
    var tasks := model.lanes.flowObjects->select(lane | lane.oclIsKindOf(bpmm::BpmmTask))->asOrderedSet();
  }
  object {
    ownedMembers := actors->collect(a | makeActor(a))->asOrderedSet();
    ownedMembers += tasks->collect(t | makeUseCase(t))->asOrderedSet();
  }
}

mapping makeActor(in lane: bpmm::BpmmLane): uml20::usecases::Actor {
  object {
    name := lane.name;
    description := lane.documentation;
  }
}

mapping makeUseCase(in task: bpmm::BpmmFlowObject): uml20::usecases::UseCase {
  object {
    name := task.name;
    description := task.documentation;
  }
}
```



Operational Mappings - Together 2006

```

transformation AddScaffoldingCode:
  umlmodel 'http://www.borland.com/together/uml';
  metamodel 'http://www.borland.com/together/uml20';

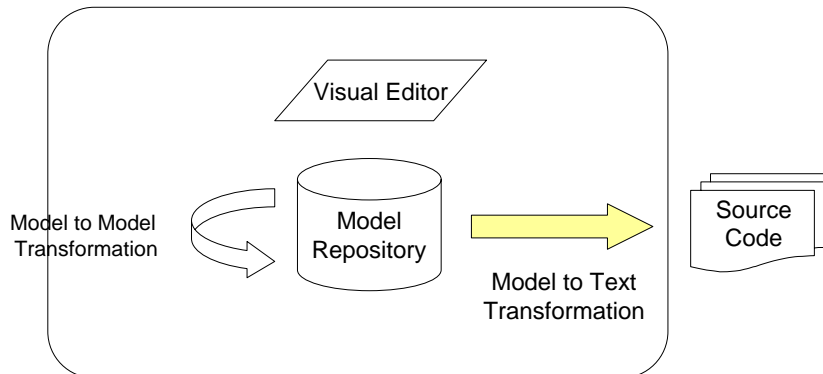
  mapping main(in model: uml::together::Model): uml::together::Model {
    object {
      ownedMembers := model.ownedMembers->select(it.it.ocIsType);
      description := model;
    };
  };


  mapping makeClass(in cl: uml::Class): uml::Class {
    init {
      var attrs := cl.attributes;
    };
    object {
      name := cl.name;
      description := 'put your';
      attributes := attrs->collect(a => {
        name := a.name;
        ownedOperations := a.ownedOperations->collect(a => {
          makeSetOperation(a);
          makeGetOperation(a);
        });
      });
    };
  };

  mapping makeAttribute(in attr: uml20::kernel::Property): uml20::kernel::Property {
    object {
      name := attr.name;
      visibility := uml::kernel::VisibilityKind::PRIVATE;
    };
  };
  
```



Model to Text (Code) Transformations





Model to Text (Code) Transformations

- Manual Programming
- XSLT
- Template Languages (Engines)
 - Velocity
 - FreeMarker
- MOF2Text (MOFScript)



Manual Programming

- Read the XML or XMI File or DB Tables etc... and load into memory + apply transformations (implemented in any PL) + generate the code
- Use the EMF generated library to:
 1. Load the model
 2. Navigate through or Query the model + println()



XSL Transformations

- XSL Transformations (XSLT)
 - <http://www.w3.org/TR/xslt>
- XML Document Transformation Language
- Defined by W3C
- Advantages:
 - Many tools exist to define and to apply XSL Transformations
 - Supported by Web Browsers
 - Efficient transformation algorithm
- Drawbacks:
 - Textual Definition
 - Complex Transformations need very large XSLT documents.



XSL Transformations. Example

Input Document:

```
<ejemplo>
  <HolaMundo />
</ejemplo>
```

Output Document:

```
<html>
<body>
  <p> Hola Mundo </p>
</body>
</html>
```

XSLT:

```
<xsl:template match="ejemplo">
<html>
<body>
  <xsl:apply-templates>
</body>
</html>
</xsl:template>
```

```
<xsl:template match="HolaMundo">
<p> Hola Mundo </p>
</xsl:template>
```



Apache Velocity



The Apache Jakarta Project
<http://jakarta.apache.org/>



- 'Lightweight' Template Engine
- Can be embedded in any kind of applications
- Open source and Implemented in Java
- Uses its own template language (called VTL)
- VTL allows including references to objects defined in Java code.



Basic Example

```
<HTML>
<BODY> Hello $customer.Name!
<table>
#foreach( $mud in $mudsOnSpecial )
  #if ( $customer.hasPurchased($mud) )
    <tr> <td>
      $flogger.getPromo($mud)
    </td></tr>
  #end
#end
</table>
```

Customer
- name : String
+ getName() : String
+ setName(newName : String)
+ hasPurchased (mud : Mud) : Bool

Mud

```
<HTML>
<BODY> Hello Pepe!
<table>
  <tr> <td>
    MUD 1 special price: 100$!
  </td></tr>
  <tr> <td>
    MUD 2 special price: 100$!
  </td></tr>
  <tr> <td>
    MUD x special price: 100$!
  </td></tr>
</table>
```



VTL Template Language

- Directives (#...)
 - Control Structures: `#if`, `#foreach`
 - Variable Assignment
`#set($list = [1,2,3])`
 - Basic Extensibility (Macros):
`#macro(maybe $variable $alt)`
`#if("$variable" == "")`
`$alt`
`#else`
`$variable`
`#end`
`-----> #maybe($person.name "Pepe")`
 - Advanced Extensibility (new directives can be defined) through an API
- Variables (\$...)
 - Can be passed through the context or declared by means of `#set`
 - Support to JavaBeans properties
`$person.name` \equiv `person.getName()`
`#set($person.name = "Pepe")`
 - Support to method invocation
`$mi_string.trim()`
- Dynamic evaluation of the VTL code (`eval()`)
`#set($list = [1,2,3])`
`#set($object = '$list')`
`#set($method = 'size()')`
`$render.eval($ctx, "${object}.${method}")`



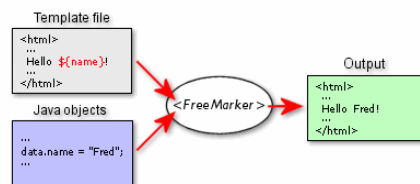
Invocation from Java

- Invocation of Templates
 1. Initialization and Configuration of Velocity
`Velocity.init();`
 2. Context instance creation (< HashTable)
`VelocityContext context = new VelocityContext();`
 3. Inserting data into the context
`Model miModelo = obtainModel();`
`context.put("model", miModelo);`
`context.put("nombre", "prueba");`
`context.put ...`
 4. Interpret the template in the context
`Template template = Velocity.getTemplate("mytemplate.vm");`
`String result = template.merge(context);`



FreeMarker (<http://freemarker.sourceforge.net>) <FreeMarker>

- Java Template Engine
- How does it work?:
 - Java Data Structures +
 - Templates +
 - Program =
 - Template filled with the input data



FreeMarker (<http://freemarker.sourceforge.net>) <FreeMarker>

- Own Template Language

```
<code><html>
<head>
  <title>Welcome!</title>
</head>
<body>
  <h1>Welcome ${user}</h1>
  <p>Our latest product:
  <a href="${latestProduct.url}">${latestProduct.name}</a>!
</body>
</html></code>
```

- Provides directives:
 - **Control structures:** if, else, elseif; switch; list, break
 - **functions:** macro, nested, return, function ...
 - **and much more:** flush, stop, ftl, t, lt, rt, nt, attempt, recover, visit, recurse, fallback.....



FreeMarker (ejemplo)

<FreeMarker>

```
package ${activator.javaPackage.name};

public class ${activator.fullName} {
  <#list activator.getMethods() as meth>
    ${method.visibility} ${meth.type} ${meth.name} (
      <#list meth.getFeatureParameters() as param>
        ${arg.type} ${arg.name}
        <#if param_has_next>,</#if>
      </#list>
    ) { ${meth.body} }
  </#list>

  <#import "JavaClass.ftl" as classTemplate>
  <#list activator.getNestedClasses() as nested>
    <@classTemplate.classNoPackage class=nested>
  </#list>
```



FreeMarker (Example)

<FreeMarker>

```
public class Activator implements BundleActivator {

  public void start(BundleContext context)
  { Properties props = new Properties();
    props.put("Language", "English");
    context.registerService(
      DictionaryService.class.getName(),
      new DictionaryImpl(), props
    );
  }

  public void stop(BundleContext context)
  { // NOTE: The service is automatically unregistered. }
}
```

Continues in the following slide...



FreeMarker (Example)



```
private static class DictionaryImpl
    implements DictionaryService
{
    String[] m_dictionary;
    public boolean checkWord(String word)
    {
        m_dictionary = { "welcome", "to",
            "the", "osgi", "tutorial" };
        word = word.toLowerCase();

        for (int i = 0; i < m_dictionary.length; i++)
        {
            if (m_dictionary[i].equals(word))
            { return true; }
        }
        return false;
    }
}
```

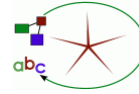


MOF2Text

- Nowadays, a little “gap” exists in the MDA specifications.
 - MOF provides a mechanism to define modeling languages.
 - MOF QVT defines a standard to transform a MOF model to another.
 - Standard Mappings like the MOF to XMI allow serializing models.
 - **However, a specification that defines how to produce text from models does not exist.**
- **MOF2Text** RFP pretends to convert models to code.
- Developers need a way to specify and control how the source code and documentation is produced.
- **MOF2Text** wants to provide a standard and practical way to write specifications for generating text (code) from level M2 MOF models.



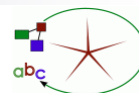
MOFScript



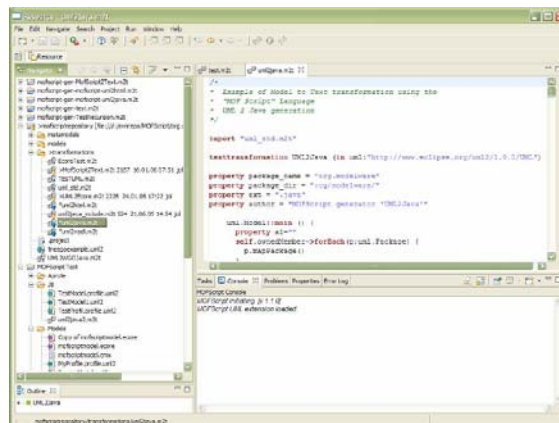
- MOFScript is an Eclipse plugin that allows generating text from MOF based models.
- It is being developed in the MODELWARE European Project and it is available as open source.
- MOFScript is prototype implementation that is based in some of the recommendations included in the [OMG MOF Model to Text RFP](#).



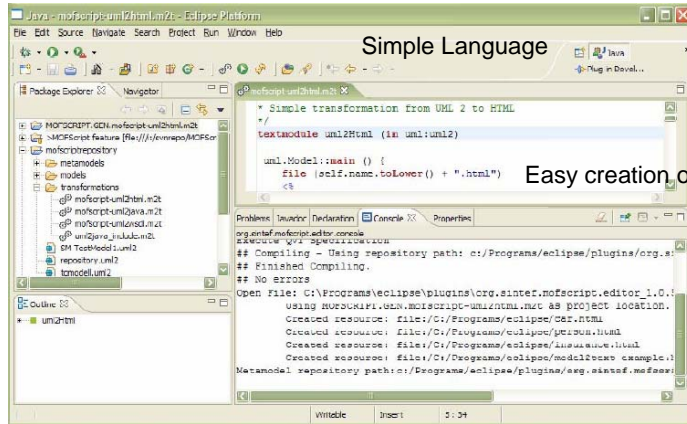
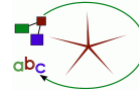
MOFScript



- MOFScript
(<http://www.modelbased.net/mofscript/>)
Eclipse Plugin.

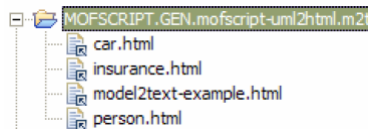


MOFScript



Easy creation of Output Files

Code Generation



MOFScript

```
import "ParticipantAbstract.m2t"
import "Participant.m2t"
import "DecoratorAbstract.m2t"
import "DecoratorConcrete.m2t"
```

Modular: Transformations can be included

Easy Access to the EMF Metamodel Repository

```
texttransformation Association2CSharp (in asso: http://associationmodel.ecore" ) {
  asso.ClassDiagram: :main () {
    self.ParticipantClass->forEach(c:asso.ParticipantClass) {
      file (c.name+"Abstract.java")
      c.generateParticipantAbstractClass ()

      file (c.name+".java")
      c.generateParticipantClass ()

      file ("Decorator"+c.name+"Abstract.java")
      c.generateDecoratorAbstractClass ()

      self.AssociationEnd->forEach(end:asso.AssociationEnd) {
        file ("Decorator" + c.name + end.association.name )
        c.generateDecoratorConcreteClass ()
      }
    }
  }
}
```



MOFScript

Complete Environment



MOFScript

Visual Access to the Source Metamodel



MOFScript

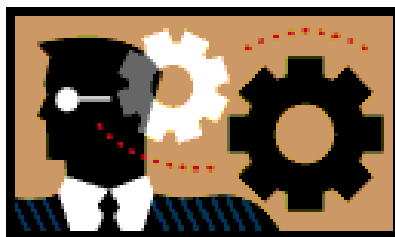
Example of C# Code Generation

```
texttransformation ParticipantAbstract (in asso:"http://associationmodel.ecore" ) {  
  asso.ParticipantClass::generateParticipantAbstractClass() {  
    <using System;  
    namespace org.oomethod.publications  
    {  
      public abstract class %> self.name <Abstract {  
%>  
      //Attribute Mapping  
      self.Attribute->forEach(a:asso.Attribute) {  
        a.generateAttribute()  
      }  
      //Identification Projection Mapping  
      self.AssociationEnd->forEach( end:asso.AssociationEnd | end.identificationProjection  
        end.association.conexion->forEach(conx:asso.AssociationEnd | conx<> end) {  
        conx.participant.Attribute->forEach(atrib:asso.Attribute | atrib.isIdentifier  
          atrib.generateAttribute()  
        }  
      }  
    }  
    //Constructor  
    <<  
    public %> self.name <Abstract () {  
      }  
    }  
  }  
}
```



That's all...

...but it's only the Beginning of MDSD



Let's have a look to the most representative Tools for supporting MDD

