Model-to-model Transformation with ATL

Frédéric Jouault (INRIA)
Brahim-Khalil LOUKIL (INRIA)
William Piers (Obeo)
Agenda

• Introduction
  Š Model-To-Model transformation in the MDE field
  Š M2M vs. M2T

• ATL Overview
  Š Available resources: wiki, zoo, newsgroup, use cases, etc.
  Š ATL language description

• First exercise: Ecore-To-UML2.1

• Architecture
  Š Overview
  Š The new virtual machine: EMF-VM

• Second exercise: GMF Diagram Refactoring

• Third exercise: Public-To-Private
Agenda

• Introduction
  - Model-To-Model transformation in the MDE field
  - M2M vs. M2T

• ATL Overview
  - Available resources: wiki, zoo, newsgroup, use cases, etc.
  - ATL language description

• First exercise: Ecore-To-UML2.1

• Architecture
  - Overview
  - The new virtual machine: EMF-VM

• Second exercise: GMF Diagram Refactoring

• Third exercise: Public-To-Private
ATL - Description

• ATL: ATLAS Transformation Language
• ATL is a language and a virtual machine dedicated to model transformation
• ATL is an Eclipse Model-to-Model (M2M) component, inside of the Eclipse Modeling Project (EMP)
Definitions

- A **model transformation** is the automatic creation of target models from source models.

- Model transformation is not only about M1 to M1 transformations:
  - M1 to M2: promotion,
  - M2 to M1: demotion,
  - M3 to M1, M3 to M2, etc.
Operational context: small theory

Metametamodel

conformsTo

G 2 P

Rule

Ma

conformsTo

Green

Class

Red

Class

MMa

conformsTo

ATL

Rule

MMa2MMb.atl

conformsTo

conformsTo

Rule

R 2 B

Rule

G 2 P

MMa2MMb.atl

conformsTo

conformsTo

conformsTo

conformsTo

Mb

Blue

Class

Pink

Class

MMb
Operational context of ATL

MOF

MMa is the source metamodel

Ma is the source model

ATL

MMa2MMb.atl

Mb is the target model

MMb is the target metamodel
M2M vs. M2T

- **M2M**: Model To Model transformation
  - Abstract syntax to abstract syntax
  - Languages: ATL, QVT Operational, QVT Relations

- **M2T**: Model To Text transformation
  - Abstract syntax to concrete syntax
  - Languages: JET, xPand

- **TMF**: Textual Modeling Framework
  - Abstract syntax to and from concrete syntax
  - Languages: TCS, xText

- This tutorial is focused on M2M
Agenda

• Introduction
  • Model-To-Model transformation in the MDE field
  • M2M vs. M2T
• ATL Overview
  • Available resources: wiki, zoo, newsgroup, use cases, etc.
  • ATL language description
• First exercise: Ecore-To-UML2.1
• Architecture
  • Overview
  • The new virtual machine: EMF-VM
• Second exercise: GMF Diagram Refactoring
• Third exercise: Public-To-Private
ATL history

• **1990**: first works on model transformation
• **1998**: initial publication for a Ph.D. thesis at the University of Nantes
• **1998 - 2004**: Implementation
  - CARROLL/MOTOR project (CEA, Thales, INRIA)
  - Collaborative projects: ModelWare, ModelPlex, OpenEmbeDD
• **2004**: Eclipse GMT integration
• **2006**: Industrial solution inside of the Eclipse M2M project
ATL community

• Active community
  ◆ Newsgroup: news://news.eclipse.org/eclipse.modeling.m2m
  ◆ Wiki: http://wiki.eclipse.org/ATL
  ◆ Bugzilla
  ◆ Use cases: http://www.eclipse.org/m2m/atl/usecases/
  ◆ Transformations zoo (i.e., a library):
    http://www.eclipse.org/m2m/atl/atlTransformations/

• Other links :
  ◆ Project page: http://www.eclipse.org/m2m/atl/
ATL transformation zoo

ATL Transformations

There are currently 101 model transformations scenarios in this zoo.

ATL Transformations list

-> Ant to Maven: Documentation, Source files
-> Assertion Modification: Documentation, Source files
-> ATL to BindingDebugger: Documentation, Source files
-> ATL to Problem: Documentation, Source files
-> ATL to Tracer: Documentation, Source files
-> BibTeX to DocBook: Documentation, Source files
-> Book to Publication: Documentation, Source files
-> CatalogueModelTransformations: Documentation, Source files
-> Class to Relation: Documentation, Source files
-> Code Clone Tools to SVG: Documentation, Source files
-> CPL to SPL: Documentation, Source files
-> Disaggregation: Documentation, Source files
-> DSL to EMF: Documentation, Source files
-> EliminateRedundantInheritance: Documentation, Source files

Model-To-Model Transformation with ATL | © 2008 by INRIA, Obeo; made available under the EPL v1.0
<table>
<thead>
<tr>
<th>List of use cases</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSLs coordination for Telephony</td>
<td>This work presents a case study of implementing two telephony languages: SPL and CPL. They are partially based on similar vocabularies. However, they are very different and have been designed to be used by different people. The use case shows how M2M transformations may be used to map programs conforming to SPL or CPL at different abstraction levels.</td>
</tr>
<tr>
<td>Models Validation through Petri nets</td>
<td>This work presents a use case of model transformation using ATL rules to validate MDO's models. This use case considers a simplified process description language, SimplePDL. It then presents a property-driven approach in which SimplePDL process models are translated into Petri nets. SimplePDL behavioral properties are expressed on corresponding Petri nets in LTL (Linear Temporal Logic). The Tina toolkit and, in particular, its model-checker, are used to validate process models by checking the expressed properties. This use case has been done by Benoit Combemale (IRIT). This is a collaboration between IRIT and LAAS labs in Toulouse, France.</td>
</tr>
<tr>
<td>Sharing Rules Between OCL/UML and SWRL/OWL</td>
<td>This work presents an implementation of sharing rules between two rule languages from different domains: OCL (Object Constraint Language) together with UML and SWRL (Semantic Web Rule Language) together with OWL. For this integration we used the R2ML (REWERSE 11 Rule Markup Language) metamodel as pivotal metamodel. The R2ML is a general Web rule markup language and it can represent different rule types: integrity, reaction, derivation and production. This work has been done by Milan Milanovic in collaboration between the GOOD OLD AI Laboratory at University of Belgrade, School of Interactive Arts and Technology at Simon Fraser University Surrey in Canada and Chair of Internet Technology at Brandenburg University of Technology at Cottbus in Germany.</td>
</tr>
</tbody>
</table>
ATL overview

• Source models and target models are distinct:
  - Source models are read-only (they can only be navigated, not modified),
  - Target models are write-only (they cannot be navigated).
• The language is a declarative-imperative hybrid:
  - Declarative part:
    - Matched rules with automatic traceability support,
    - Side-effect free navigation (and query) language: OCL 2.0
  - Imperative part:
    - Called rules,
    - Action blocks.
• Recommended programming style: declarative
ATL overview (continued)

• A declarative rule specifies:
  • a source pattern to be matched in the source models,
  • a target pattern to be created in the target models for each match during rule application.

• An imperative rule is basically a procedure:
  • It is called by its name,
  • It may take arguments,
  • It can contain:
    ▪ A declarative target pattern,
    ▪ An action block (i.e. a sequence of statements),
    ▪ Both.
ATL overview (continued)

• Applying a declarative rule means:
  * Creating the specified target elements,
  * Initializing the properties of the newly created elements.

• There are three types of declarative rules:
  * **Standard** rules that are applied **once** for each match,
    ▪ A given set of elements may only be matched by one standard rule,
  * **Lazy** rules that are applied **as many times** for each match as it is referred to from other rules (possibly never for some matches),
  * **Unique lazy** rules that are applied at most once for each match and only if it is referred to from other rules.
Declarative rules: source pattern

- The source pattern is composed of:
  - A labeled set of types coming from the source metamodels,
  - A guard (Boolean expression) used to filter matches.

- A match corresponds to a set of elements coming from the source models that:
  - Are of the types specified in the source pattern (one element for each type),
  - Satisfy the guard.
Declarative rules: target pattern

• The target pattern is composed of:
  • A labeled set of types coming from the target metamodels,
  • For each element of this set, a set of bindings.
  • A binding specifies the initialization of a property of a target element 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.
Execution order of declarative rules

- Declarative ATL frees the developer from specifying execution order:
  - The order in which rules are matched and applied is not specified.
    - Remark: the match of a lazy or unique lazy rules must be referred to before the rule is applied.
  - The order in which bindings are applied is not specified.

- The execution of declarative rules can however be kept **deterministic**:
  - The execution of a rule cannot change source models
    - It cannot change a match,
  - Target elements are not navigable
    - The execution of a binding cannot change the value of another.
Example: Class to Relational - Overview

• The source metamodel Class is a simplification of class diagrams.
• The target metamodel Relational is a simplification of the relational model.

⇒ ATL declaration of the transformation:

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

• The transformation excerpts used in this presentation come from:

http://www.eclipse.org/m2m/atl/atlTransformations/#Class2Relational
Example: Class to Relational - Source Metamodel

- **NamedElt**
  - name: String

- **Classifier**
  - type

- **DataType**

- **Class**
  - attr

- **Attribute**
  - multivalued: Boolean

The diagram shows the relationships between the concepts and their attributes.
Example: Class to Relational – Target Metamodel

```
Named
+ name: String

Table
+ keyOf: 0..1
+ owner
  {ordered}

Column
+ col
+ key

Type
* + type
```
Example: Class to Relational, overview

- Informal description of rules
  - **Class2Table:**
    - A table is created from each class,
    - The columns of the table correspond to the single-valued attributes of the class,
    - A column corresponding to the key of the table is created.
  - **SingleValuedAttribute2Column:**
    - A column is created from each single-valued attribute.
  - **MultiValuedAttribute2Column:**
    - A table with two columns is created from each multi-valued attribute,
    - One column refers to the key of the table created from the owner class of the attribute,
    - The second column contains the value of the attribute.
Example: Class to Relational - Rule Class2Table (1 Of 4)

- For each Class, create a Table:

```plaintext
rule Class2Table {
  from -- source pattern
  c : Class!Class
  to -- target pattern
  t : Relational!Table
}
```
Example: Class to Relational - Rule Class2Table (2 Of 4)

• The *name* of the Table is the *name* of the Class:

```
rule Class2Table {
  from
c : Class!Class
  to
t : Relational!Table (name <- c.name -- a simple binding )
}
```
Example: Class to Relational - Rule Class2Table (3 Of 4)

- The columns of the table correspond to the single-valued attributes of the class:

```atl
rule Class2Table {
  from
    c : Class!Class
  to
    t : Relational!Table (name <- c.name,
                          col <- c.attr->select(e | -- a binding
                                       not e.multiValued -- using
                          complex navigation
                         )
                    }
  
  Remark: attributes are automatically resolved into columns by automatic traceability support.
```
Example: Class to Relational - Rule Class2Table (4 Of 4)

- Each Table owns a key containing a unique identifier:

``` ATL
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’
    -- another target
  )
  -- pattern element

  -- for the key
}
```
Example: Class to Relational - Rule SingleValuedAttribute2Column

• For each single-valued Attribute create a Column:

```plaintext
rule SingleValuedAttribute2Column {
    from -- the guard is used for selection
    a : Class!Attribute (not a.multiValued)
    to
    c : Relational!Column (name <- a.name
                           )
}
```
Example: Class to Relational - Rule MultiValuedAttribute2Column

• For each multi-valued Attribute create a Table, which contains two columns:
  • The identifier of the table created from the class owner of the Attribute
  • The value.

```plaintext
rule MultiValuedAttribute2Column {
  from
  a : Class!Attribute (a.multiValued)
  to
  t : Relational!Table (  
    name <- a.owner.name + '_' + a.name, 
    col <- Sequence {id, value} 
  ),

  id : Relational!Column (  
    name <- 'Id'  
  ),
  value : Relational!Column (  
    name <- a.name  
  )
}```
Object Constraint Language (OCL)

- Originally intended to express constraints over UML models, for instance:
  
  context Person inv: self.age > 0

- Extended to query any model

- Used in several transformation languages (e.g., ATL, QVT) to compute values from the source models

- Specification:
  - The version on which ATL is based is available from:
    
  - Section 7: language overview
  - Section 11: standard library
    - Section 11.8: iterator expressions
Other ATL features: rule inheritance

- Rule inheritance, to help structure transformations and reuse rules and patterns:
  - A child rule matches a subset of what its parent rule matches,
    - All the bindings of the parent still make sense for the child,
  - A child rule specializes target elements of its parent rule:
    - Initialization of existing elements may be improved or changed,
    - New elements may be created,
  - Syntax:

    ```
    abstract rule R1 {
      -- ...
    }
    rule R2 extends R1 {
      -- ...
    }
    ```
Copy class inheritance without rule inheritance

--- Source metamodel: MMA

```java
class A1 { attribute v1 : String; }
class A2 extends A1 {
    attribute v2 : String;
}
```

```java
module MMAtoMMB;
create OUT : MMB from IN : MMA;
rule A1toB1 {
    from
    s : MMA!A1
    to
    t : MMB!B1 (v1 <- s.v1)
}
```

--- Target metamodel: MMB

```java
class B1 { attribute v1 : String; }
class B2 extends B1 {
    attribute v2 : String;
}
```

```java
rule A2toB2 {
    from
    s : MMA!A2
    to
    t : MMB!B2 (v1 <- s.v1,
                v2 <- s.v2)
}
```
Copy class inheritance with rule inheritance

-- Source metamodel: MMA

class A1 { attribute v1 : String; }
class A2 extends A1 {
    attribute v2 : String;
}

module MMAtoMMB;
create OUT : MMB from IN : MMA;
rule A1toB1 {
    from
    s : MMA!A1
    to
    t : MMB!B1 ( v1 <- s.v1 )
}

-- Target metamodel: MMB

class B1 { attribute v1 : String; }
class B2 extends B1 {
    attribute v2 : String;
}

rule A2toB2 extends A1toB1 {
    from
    s : MMA!A2
    to
    t : MMB!B2 ( v2 <- s.v2 )
}
Other ATL features: refining mode

• Refining mode for transformations that need to modify only a small part of a model:
  - Since source models are read-only target models must be created from scratch,
  - This can be done by writing copy rules for each elements that are not transformed,
    ➔ This is not very elegant,
  - In refining mode, the ATL engine automatically copies unmatched elements.

• The developer only specifies what changes.
  ➔ An (optimized) engine may modify source models in-place but only commit the changes in the end.

• Syntax: replace from by refining
  
  module A2A; create OUT : MMA refining IN : MMA;
Model-To-Model Transformation with ATL | © 2008 by INRIA, Obeo; made available under the EPL v1.0
ATL content assist

• ADT (ATL Development Tools) has recently been improved with basic content assist
  - Type completion
  - Left-side bindings completion
  - Basic code templates
Launching ATL using ANT

```xml
<project name="Families2Persons" default="main">
    <property name="input" value="sample-Families.ecore"/>
    <property name="output" value="sample-Persons.ecore"/>

    <target name="main" depends="loadModels">
        <am3.loadModel modelHandler="EMF"
            name="myFamilies" metamodel="Families"
            path="${input}"/>

        <am3.atl path="Families2Persons.atl">
            <inmodel name="Families" model="Families"/>
            <inmodel name="IN" model="myFamilies"/>
            <inmodel name="Persons" model="Persons"/>
            <outmodel name="OUT" model="myPersons" metamodel="Persons"/>
        </am3.atl>

        <am3.saveModel model="myPersons" path="${output}"/>
    </target>
</project>
```
Agenda

• Introduction
  - Model-To-Model transformation in the MDE field
  - M2M vs. M2T

• ATL Overview
  - Available resources: wiki, zoo, newsgroup, use cases, etc.
  - ATL language description

• First exercise: Ecore-To-UML2.1

• Architecture
  - Overview
  - The new virtual machine: EMF-VM

• Second exercise: GMF Diagram Refactoring

• Third exercise: Public-To-Private
First exercise: Ecore-To-UML2.1

- In Java
  - Already implemented by eclipse UML2 API.
  - Ecore2UMLConverter.java
    - ~2330 code lines

- In ATL
  - Ecore2UML.atl
    - ~230 code lines
First exercise: Quick overview

- **Java**

```java
public Object caseETypedElement(ETypedElement eTypedElement) {
    Object element = eModelElementToElementMap.get(eTypedElement);
    if (element != null) {
        if (element instanceof TypedElement) {
            ((TypedElement) element).setType(getType(eTypedElement));
        }
    }
    ...
    return element;
}
```

```java
public Object caseEAttribute(EAttribute eAttribute) {
    EClass eContainingClass = eAttribute.eContainingClass();
    if (eContainingClass != null) {
        Property property = UMLFactory.eINSTANCE.createProperty();
        eModelElementToElementMap.put(eAttribute, property);
        Classifier classifier = (Classifier) doSwitch(eContainingClass);
        getOwnedAttributes(classifier).add(property);
        property.setName(eAttribute.getName());
        property.setIsReadOnly(!eAttribute.isChangeable());
        property.setIsDerived(eAttribute.isDerived());
        property.setVisibility(VisibilityKind.PUBLIC_LITERAL);
        caseETypedElement(eAttribute);
        defaultCase(eAttribute);
    }
    return super.caseEAttribute(eAttribute);
}
```

- **ATL**

```java
abstract rule ETypedElement2TypedElement {
    from et: Ecore!ETypedElement
to f: UML!TypedElement(
    name<-et.name,
    type<-thisModule.getMappingType(et),
    lower<-et.lowerBound,
    upper<-et.upperBound
    )
}

rule EAttribute2Property extends ETypedElement2TypedElement {
    from et: Ecore!EAttribute
to f: UML!Property(
    isReadOnly<- not et.changeable,
    isDerived<- et.derived
    )
}
```
Agenda

• Introduction
  - Model-To-Model transformation in the MDE field
  - M2M vs. M2T
• ATL Overview
  - Available resources: wiki, zoo, newsgroup, use cases, etc.
  - ATL language description
• First exercise: Ecore-To-UML2.1
• Architecture
  - Overview
  - The new virtual machine: EMF-VM
• Second exercise: GMF Notation-To-Notation
• Third exercise: Public-To-Private
ATL main plugins organization

Eclipse ui integration

ATL core

ATL Regular VM (with drivers)

New ATL VM EMF-dedicated
ATL architecture

- Intermediate file format: ASM
- Modular VM dedicated to M2M support
  - A complete specification describes the VM (http://www.eclipse.org/m2m/atl/doc/)
  - Can be implemented on different platforms (Java, .Net)
ATL execution process

- 2 steps
  - Compilation
  - Execution
EMF-VM

- A new ATL Virtual Machine is now available
  - Increased performance
  - EMF-only (no wrapping of EObjects)
- At this time...
  - Some missing features (decreasing in number quickly)
  - No UML profile support
- Non-regression evaluated with a new test plugin
<table>
<thead>
<tr>
<th>Test case</th>
<th>Status</th>
<th>Time (omfv)</th>
<th>Time (Regular VM)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace2PerformanceMetrics</td>
<td>PASS</td>
<td>0.015s</td>
<td>0.0s</td>
<td></td>
</tr>
<tr>
<td>BibTeX2DocBook</td>
<td>PASS</td>
<td>0.031s</td>
<td>0.047s</td>
<td></td>
</tr>
<tr>
<td>FlattenTest</td>
<td>PASS</td>
<td>0.047s</td>
<td>0.032s</td>
<td>Collections implementation completed on 12/12/2007.</td>
</tr>
<tr>
<td>XML2Book</td>
<td>PASS</td>
<td>0.0s</td>
<td>0.016s</td>
<td></td>
</tr>
<tr>
<td>Families2Persons</td>
<td>PASS</td>
<td>0.0s</td>
<td>0.0s</td>
<td></td>
</tr>
<tr>
<td>Ecore2Class</td>
<td>PASS</td>
<td>0.0s</td>
<td>0.016s</td>
<td></td>
</tr>
<tr>
<td>DSL2KM3</td>
<td>PASS</td>
<td>0.032s</td>
<td>0.031s</td>
<td></td>
</tr>
<tr>
<td>Book2Publication</td>
<td>PASS</td>
<td>0.016s</td>
<td>0.015s</td>
<td></td>
</tr>
<tr>
<td>Class2Relational</td>
<td>PASS</td>
<td>0.0s</td>
<td>0.0s</td>
<td></td>
</tr>
<tr>
<td>UML2Relational</td>
<td>FAIL</td>
<td>0.046s</td>
<td>0.0s</td>
<td>Unsupported model elements containing &quot;::&quot;.</td>
</tr>
<tr>
<td>DSL2XML</td>
<td>PASS</td>
<td>0.094s</td>
<td>0.156s</td>
<td></td>
</tr>
<tr>
<td>AssertionModification</td>
<td>PASS</td>
<td>0.078s</td>
<td>0.063s</td>
<td></td>
</tr>
<tr>
<td>XML2DSL Model</td>
<td>PASS</td>
<td>0.938s</td>
<td>10.203s</td>
<td></td>
</tr>
<tr>
<td>XML2Ant</td>
<td>PASS</td>
<td>0.25s</td>
<td>2.053s</td>
<td></td>
</tr>
<tr>
<td>DSLModel2KMZ</td>
<td>PASS</td>
<td>0.016s</td>
<td>0.031s</td>
<td></td>
</tr>
<tr>
<td>Ant2Maven</td>
<td>PASS</td>
<td>0.015s</td>
<td>0.031s</td>
<td></td>
</tr>
<tr>
<td>KM32DSL</td>
<td>PASS</td>
<td>0.032s</td>
<td>0.031s</td>
<td></td>
</tr>
<tr>
<td>Maven2XML</td>
<td>PASS</td>
<td>0.031s</td>
<td>0.047s</td>
<td></td>
</tr>
<tr>
<td>ATL2Problem</td>
<td>PASS</td>
<td>0.046s</td>
<td>0.078s</td>
<td>EnumLiteral support added on 25/01/2008.</td>
</tr>
<tr>
<td>XML2DSL</td>
<td>PASS</td>
<td>0.078s</td>
<td>0.453s</td>
<td></td>
</tr>
<tr>
<td>SpreadsheetMLSimplified2Trace</td>
<td>FAIL</td>
<td>0.015s</td>
<td>0.0s</td>
<td>Transformation fails without errors.</td>
</tr>
</tbody>
</table>
## EMF-VM Benchmarks

<table>
<thead>
<tr>
<th></th>
<th>Benchmarks</th>
<th>ATL</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Regular VM</td>
<td>emfvm</td>
</tr>
<tr>
<td>RSM</td>
<td>to TOPCASED</td>
<td>v0: 45 min</td>
<td>v0: 75 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>v1: ? min</td>
<td>v1: 45 s</td>
</tr>
<tr>
<td></td>
<td>UML2 API to</td>
<td>v0: 92 min 37 sec</td>
<td>v0: 29 min 53 sec</td>
</tr>
<tr>
<td></td>
<td>Platform Ontologies</td>
<td>v1: ?min</td>
<td>v1: ?min</td>
</tr>
</tbody>
</table>

*EMF-VM Benchmarks*

EMF-VM Benchmarks

<table>
<thead>
<tr>
<th></th>
<th>Benchmarks</th>
<th>ATL</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Regular VM</td>
<td>emfvm</td>
</tr>
<tr>
<td>RSM</td>
<td>to TOPCASED</td>
<td>v0: 45 min</td>
<td>v0: 75 s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>v1: ? min</td>
<td>v1: 45 s</td>
</tr>
<tr>
<td></td>
<td>UML2 API to</td>
<td>v0: 92 min 37 sec</td>
<td>v0: 29 min 53 sec</td>
</tr>
<tr>
<td></td>
<td>Platform Ontologies</td>
<td>v1: ?min</td>
<td>v1: ?min</td>
</tr>
</tbody>
</table>

*EMF-VM Benchmarks*
Agenda

• Introduction
  • Model-To-Model transformation in the MDE field
  • M2M vs. M2T
• ATL Overview
  • Available resources: wiki, zoo, newsgroup, use cases, etc.
  • ATL language description
• First exercise: Ecore-To-UML2.1
• Architecture
  • Overview
  • The new virtual machine: EMF-VM
• Second exercise: GMF Diagram Refactoring
• Third exercise: Public-To-Private
Second Exercise: GMF Diagram Refactoring

• Context:
  • GMF (Graphical Modeling Framework) enables graphical concrete syntaxes for models
  • Graphical properties (e.g., position, size, color) are stored in a separate *Notation* model, for instance:
    ▪ default.uml conforming to the *UML* metamodel
    ▪ default.umlclass_diagram conforming to the *Notation* metamodel

• Objective:
  • Mirroring a *Notation* model horizontally and vertically

• Approach:
  • Using an ATL transformation from *Notation* to *Notation*
GMF Diagram Refactoring: Sample Source
GMF Diagram Refactoring: Sample Target
GMF Diagram Refactoring

• Possible extensions:
  - Bendpoints for edges
  - Styles (e.g., so that lines appear in the same shade of gray)
  - Other kinds of refactorings
Agenda

• Introduction
  ◦ Model-To-Model transformation in the MDE field
  ◦ M2M vs. M2T
• ATL Overview
  ◦ Available resources: wiki, zoo, newsgroup, use cases, etc.
  ◦ ATL language description
• First exercise: Ecore-To-UML2.1
• Architecture
  ◦ Overview
  ◦ The new virtual machine: EMF-VM
• Second exercise: GMF Diagram Refactoring
• Third exercise: Public-To-Private
Third exercise: Public To Private

• Context
  - Simplified Java metamodel: classes, fields, methods
  - Sample Java model: PetriNet

• Objective
  - Convert public fields into:
    - Private fields
    - A getter method
    - A setter method (if the attribute is changeable, i.e. non-final)

• Sample
  - A classic PetriNet model encoded using the simplified Java metamodel
Input model (with public fields)
Output model (privatized)
Legal Notices

• Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both
• Other company, product, or service names may be trademarks or service marks of others
END

• Questions or Comments?