Model Driven Architecture
Model Transformation

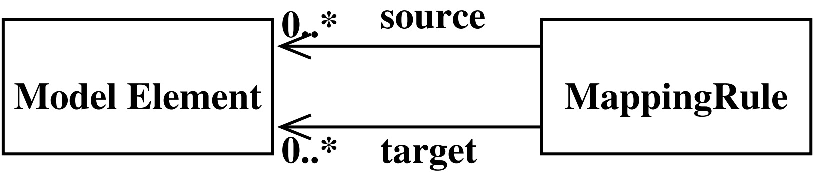
Prof. Dr. Peter Thiemann

Universität Freiburg

14.06.2006

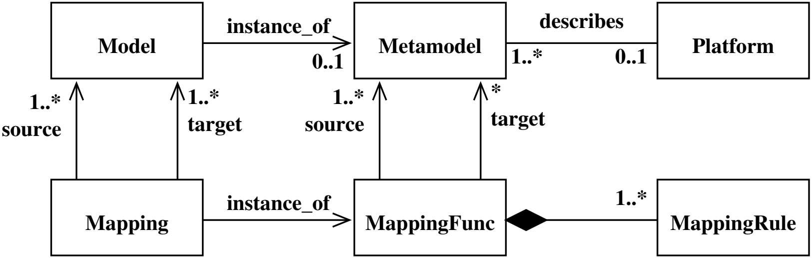
Model Transformation

Abstract Setting



- Relate elements in the source model to elements in the target model
- Express by relating metaclasses

Model Transformation
Mapping Definition



- Tunability for special cases
  - exceptions from the general rule
  - additional information
- Traceability
  links from target model to source model
- Preservation of extra information attached to generated models
- Bidirectionality

Tunability

- Manual control — not feasible
- Transformation parameters
  - name prefixes
  - database tuning
- Conditional transformations
  - depending on (combination of) stereotypes
  - depending on name conventions
  - depending on parameters

Traceability

- Each generated target element “knows” the source elements it depends on
- Changes to the generated code
  - Warning if edited code is no longer an image
  - Edit in target model are propagated back to the source model
- Testing and Debugging
- Impact analysis if requirements change

Preservation

- Without 100% code generation, edits needed in target **Model Elaboration**; “filling in the method bodies”
- Change of source model and regeneration should preserve the edits
- Typical concept: **Protected Region**
  - Hole in the generated code
  - Uniquely identified to enable preservation
  - Mix of generated and handwritten code potentially introduces dependencies

```
public int showSummary() throws RemoteException {
  //PROTECTED REGION ID(3CF1E93C037E) START
  try {
    // ...
  } catch (Exception e) {
    throw new RemoteException(
      "$EntityObject ’Account’ : couldn’t execute operation ’showS
  }
  //PROTECTED REGION END
}
```

Bidirectionality

- Not achievable in general, but interesting for reverse engineering
- Definition
  - Both transformations from one definition
  - Two transformation definitions with proof that they are inverses
- Further problems
  - edits in the target model
  - target model only may only reflect one facet of the source model
  - different levels of abstraction

Transformation Parameters

Placement Options

- source model
  - may break abstraction
  - may lead to clutter
- target model
  - not available before first transformation
  - obsolete information (clutter)
- transformation object

Query, Views, and Transformations (QVT)

- Transformations should be automatized
  - turn-around time
  - scalability (large models)
  - improved quality (fewer errors)
- Formal definition of transformations needed
- QVT is OMG's RFP (request for proposals) for a language for defining transformations
  no standardization, yet
- Basic choices for QVT
  - Imperative/procedural programming (not reversible)
  - Template-based approach (text generation, reversible?)
  - Declarative specification (often rule-based, improved chance for reversibility)
- More to follow

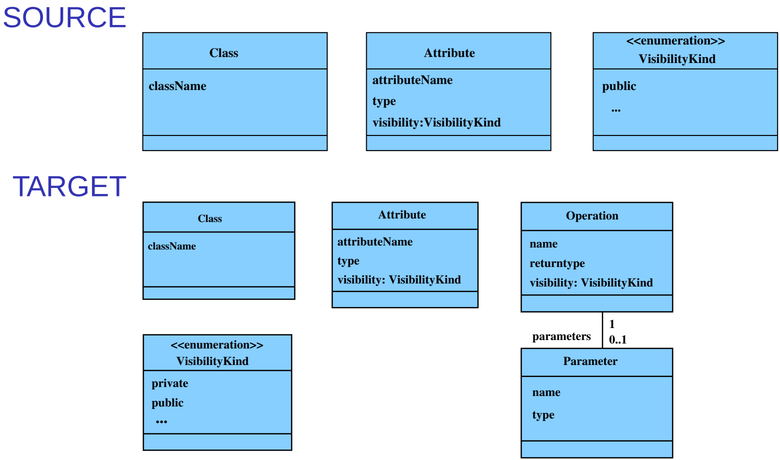
Example Transformation: Getter and Setter

Textual Description (Declarative)

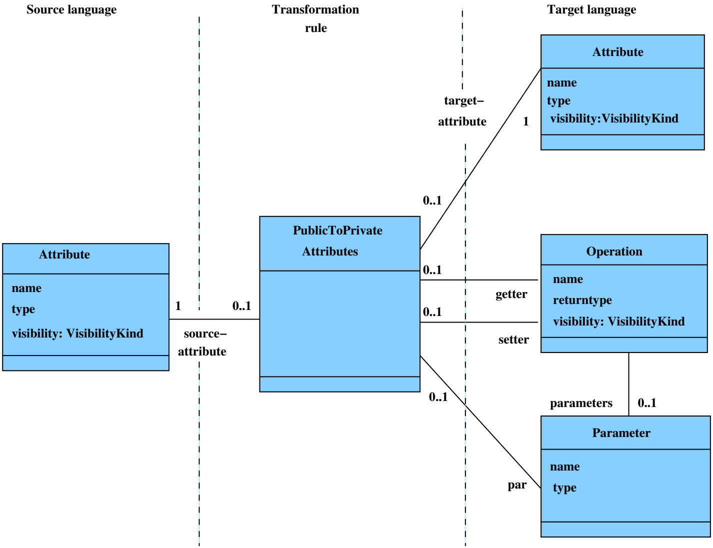
- For each class `className` in SOURCE there is a class `className` in the TARGET
- For each public attribute `attName : Type` of class `className` in the SOURCE there are the following aggregates to class `className` in the TARGET:
  - a private attribute `attName : Type`,
  - a public operation `getAttName () : Type`,
  - a public operation `setAttName (att : Type)`

Example Transformation

Source and Target Metamodels



Example Transformation
Transformation Rule



Example Transformation

Specification of “MDA Explained”

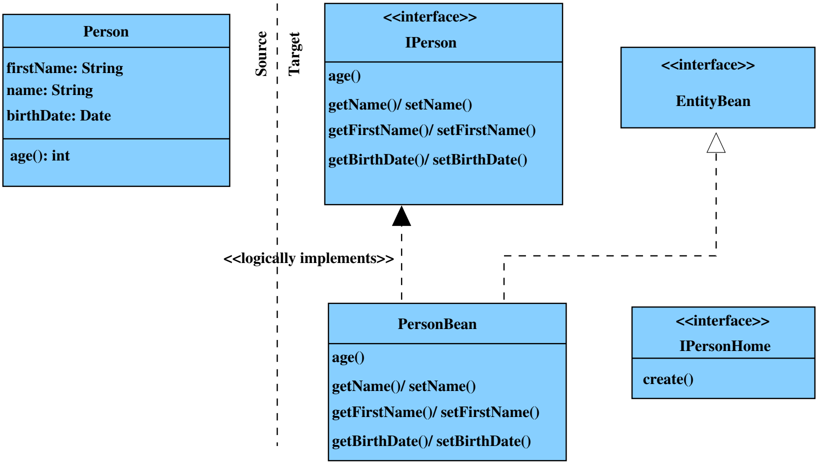
```
TRANSFORMATION PublicToPrivateAttributes (UML, UML)
  PARAMS
    setterprefix : String = 'set';
    getterprefix : String = 'get';
  SOURCE
    sourceAttr : UML::Attribute;
  TARGET
    targetAttr : UML::Attribute;
    getter     : UML::Operation;
    setter     : UML::Operation;
  BIDIRECTIONAL;
  MAPPING
    sourceAttr.name <~> targetAttr.name;
    sourceAttr.type <~> targetAttr.type;
```

Example Transformation

Specification of “MDA Explained”/2

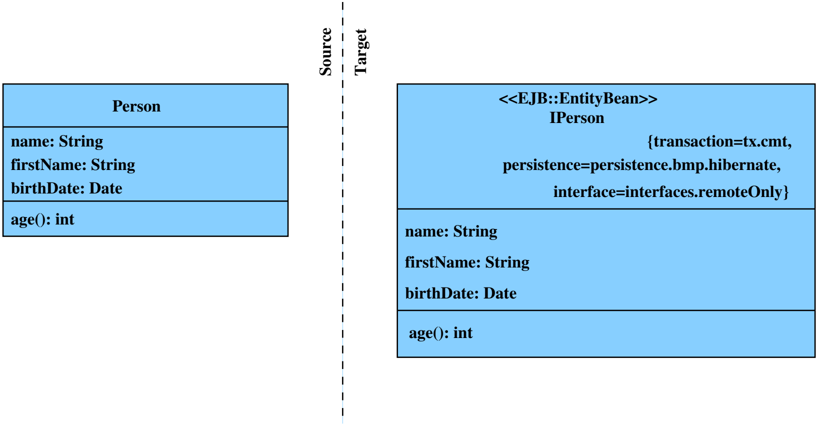
```
  SOURCE CONDITION
    sourceAttr.visibility = VisibilityKind::public;
  TARGET CONDITION
    targetAttr.visibility = VisibilityKind::private     and
    setter.name = setterprefix.concat(targetAttr.name)  and
    setter.parameters->exists(p |
                             p.name = targetAttr.name
                             and
                             p.type = targetAttr.type) and
    setter.type = OclVoid                               and
    getter.name = getterprefix.concat(targetAttr.name)  and
    getter.parameters->isEmpty()                        and
    getter.type = targetAttr.type                       and
    targetAttr.class = setter.class                     and
    targetAttr.class = getter.class;
END TRANSFORMATION
```

Example Transformation: EJB
Goal in Terms of UML

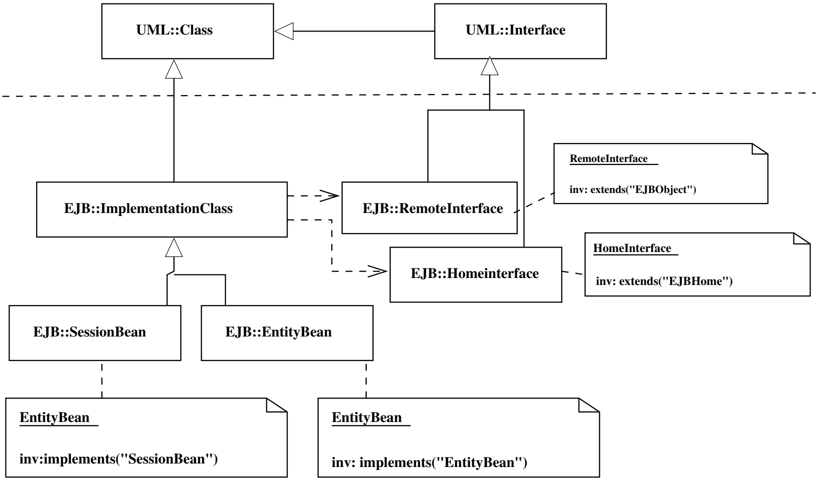


Example Transformation: EJB

Goal in Terms of an EJB Metamodel



Example Transformation: EJB
Simplified Target Metamodel



Example Transformation: EJB

Imperative Programming

```
Model createEJBModel(Model source) {
  Model target = new Model();

  for (Class c : source.classes) {

    ImplementationClass
      implClass = new ImplementationClass();
    implClass.setName( c.getName() + "Bean");
    target.addClass( implClass );

    Dependencies.define( implClass, c );
```

Example Transformation: EJB

Imperative Programming/2

```
    RemoteInterface ri = new RemoteInterface();
    ri.setName( "I" + c.getName() );
    target.addClass( ri );

    HomeInterface hi = new HomeInterface();
    hi.setName( c.getName() + "Home");
    target.addClass( hi );

    for (Operation o : c.operations ) {
      ri.addOperation( new Operation( o.clone() ));
      implClass.addOperation(
        new Operation( o.clone() ));
    }
  }
}
```

- JMI $=$ Java Metadata Interface
- API for creation, storage, access, discovery, and exchange of metadata
- based on MOF (provides “standard interface”)
- access to metadata at design and runtime (!)
- metamodel and metadata interchange using XMI (XML Metadata Interchange)
- implementations: UniSys (reference), NetBeans (open source), SAP NetWeaver

Implementation in Java: openArchitectureWare

- open source mde framework supported by software companies
  see `openarchitectureware.org`
- contains Java implementation of
  - UML Class Metamodel
  - Activity Core Metamodel
- each metaclass is represented by a Java class
- specialized metaclasses (*e.g.*, stereotypes) by subclasses of metaclasses

```
public class EJB.ImplementationClass
  extends UML.Class {}
public class EJB.SessionBean
  extends EJB.ImplementationClass {}
public class EJB.EntityBean
  extends EJB.ImplementationClass {}
```

- OAW generates for each model element an instance of the implementation of the corresponding metaclass
- each model element is represented by a Java object
- tagged values are instance variables
- transformation traverses model to generate a new model
- service methods for transformation into the objects

```
public class EJB.EntityBean
  extends EJB.ImplementationClass {
  protected ElementSet keyList = null;
  public ElementSet key() throws DesignException {
    if (keyList == null) {
      keyList = new ElementSet();
      for( Attribute att : attributes() ) {
        if (att instanceof Key) {
          keyList.add(att);
        }
      }
    return keyList;
  }
}
```

- if UML tool does not check design constraints
- OAW invokes all `CheckConstraints()` methods after instantiation

```
public class EJB.EntityBean
  extends EJB.ImplementationClass {
  public String CheckConstraints() throws DesignExc
    if(key().isEmpty()) {
      throw new DesignException("Constraint violati
            +"no key found for entity "
            + this.Name());
    }
  return "";
  }
}
```

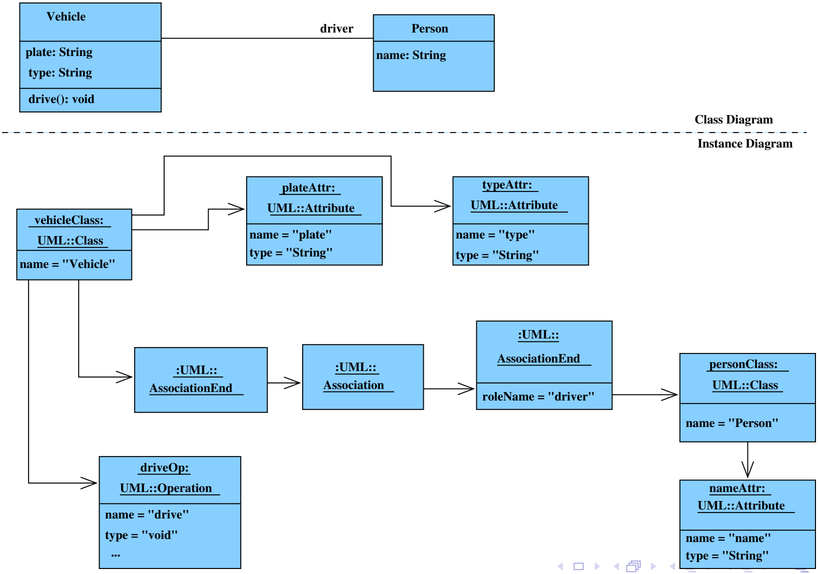
Example Transformation: EJB

Graphical Specification

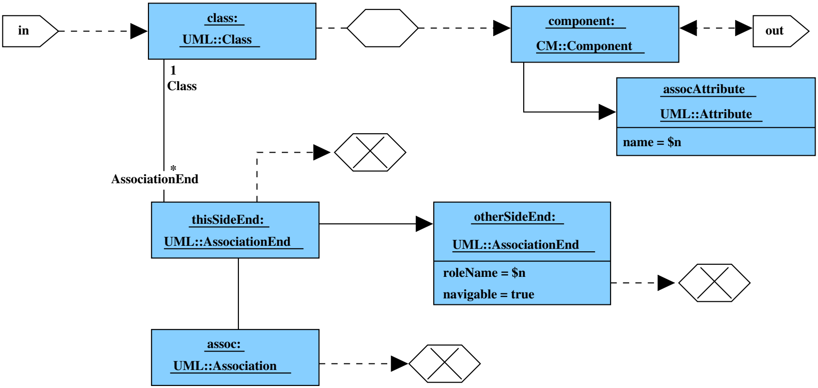
- UMLX: pattern matching against instance diagram
- graphical notation based on class diagrams viewed as instance diagrams wrt their metamodel
- extensions to specify
  - input
  - output
  - deleted elements
  - new elements

Example Transformation: EJB

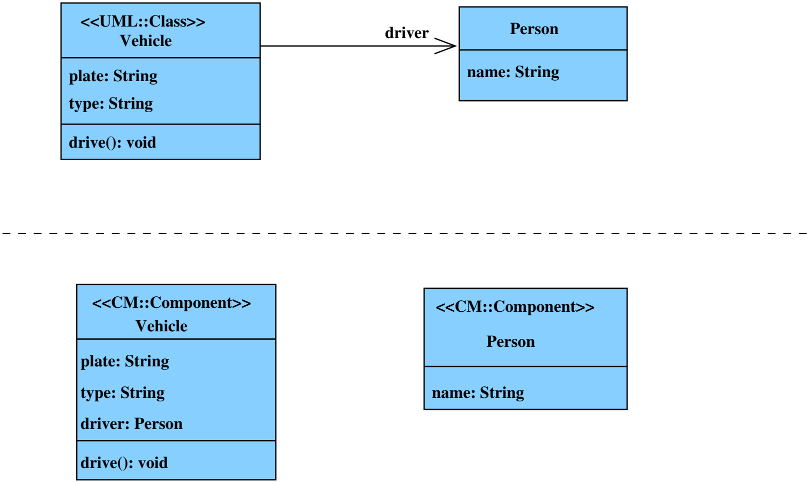
Reminder: Class Diagram as Instance Diagram



Example Transformation: EJB
Transformation Rule in UMLX

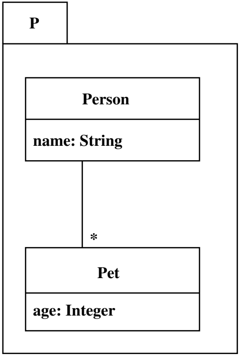


Example Transformation: EJB
Result for Transformation



Declarative Transformation

Textual Notation: Class Diagrams




```
(Package)[
  name= 'P' ,
  contents = {
    (Class, Person)[
     name = 'Person',
     attributes = {
       (Attribute)[
         name = 'name',
         type = String],
        (Attribute)[
         name = 'pet',
         type = (SetType)[
         elementType = Pet]]}],
(Class, Pet)[
     name = 'Pet',
     attributes = {
     (Attribute)[
          name = 'age',
          type = Integer]}]}]
```

Declarative Transformation
Textual Notation: Pattern for Relation

```
relation R {
  domain {
    pattern-1 [when condition-1]
  }
  domain {
    pattern-2 [when condition-2]
  }
  when {
    condition
  }
}
```

```
relation Method_To_XML {
  domain {
    (UML.Method)[name = n, body = b]
  }
  domain {
    (XML.Element)[
      name = "Method",
      attributes = {(XML.Attribute)[
                      name = "name",
                      value = n]},
      contents = {b}
    ]
  }
}
```