

# MODEL TRANSFORMATIONS? TRANSFORMATION MODELS!

JEAN BÉZIVIN, FABIAN BUTTNER, MARTIN  
GOGOLLA, FREDERIC JOUALT, IVAN KURTEV,

PRESENTED BY: RAFAEL OLAECHEA

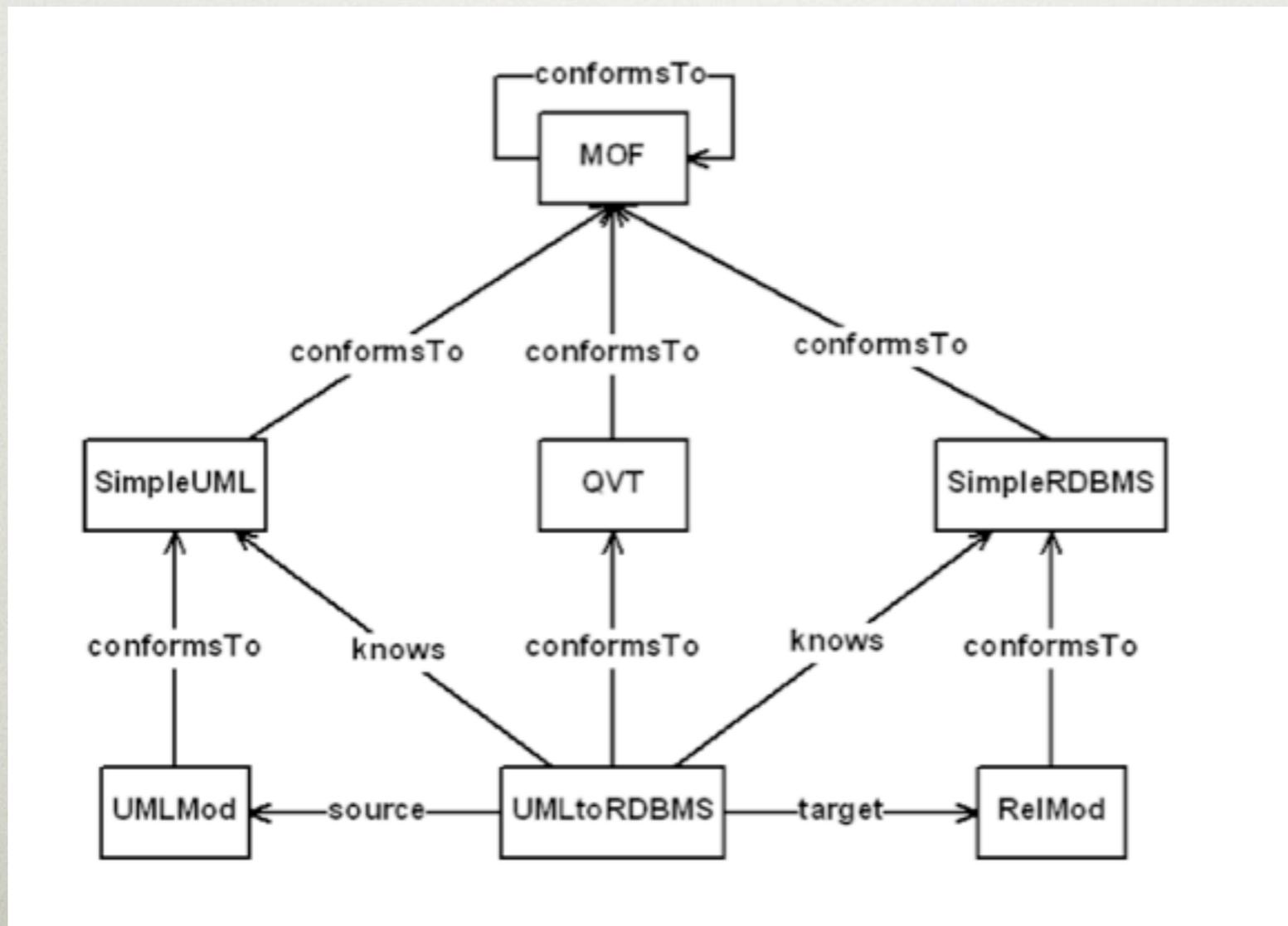
# WHY USE TRANSFORMATION MODELS?

---

- Focus on the properties of transformations instead of an operational description of them.
  - Undirected.
  - Higher Order Transformations.
  - Validations and Completions.

# OPERATIONAL VIEW OF MODEL TRANSFORMATIONS

## Query View Transformation (QVT) Standard



# OPERATIONAL VIEW OF MODEL TRANSFORMATIONS

---

```
map packageToSchema in umlRdbms {  
  
    uml () {  
        p:Package  
    }  
  
    rdbms () {  
        s:Schema  
    }  
    where () {  
  
        p2s:PackageToSchema |  
        p2s.umlPackage = p;  
        p2s.schema = s;  
    }  
    map {  
        where () {  
            p2s.name := p.name;  
            p2s.name := s.name;  
            p.name := p2s.name;  
            s.name := p2s.name;  
        }  
    }  
}
```

# OPERATIONAL VIEW OF MODEL TRANSFORMATIONS

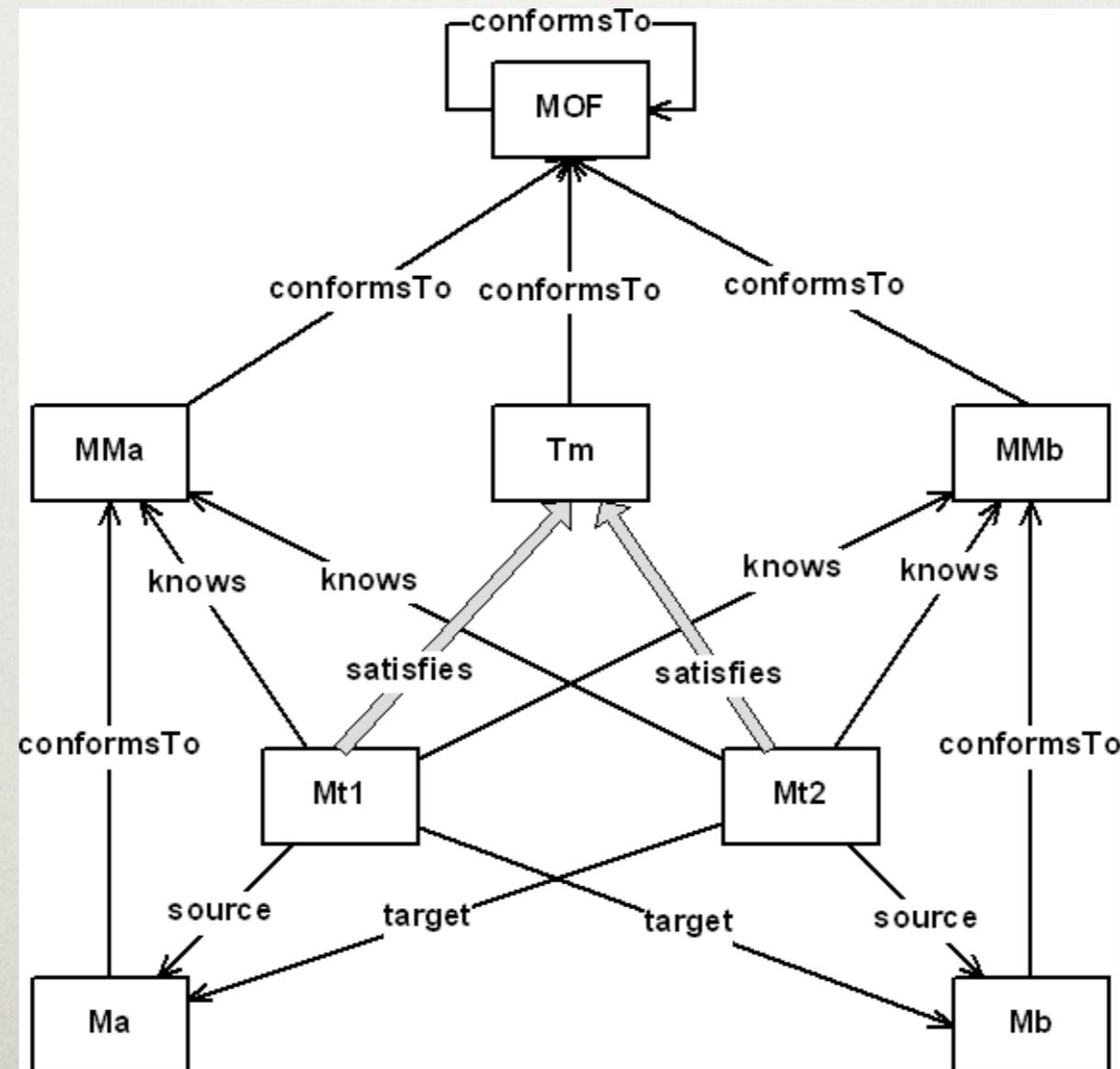
---

```
class PrimitiveToName {
  owner : PackageToSchema opposites primitivesToNames;
  name : String;
  -- uml
  primitive : PrimitiveDataType;
  --rdbms
  typename: String;
}

map primitiveToName in umlRdbms {
  uml (p:Package) {
    prim:PrimitiveDataType |
    prim.owner = p;
  }
  check enforce rdbms () {
    sqlType:String
  }
  where (p2s:PackageToSchema | p2s.umlPackage=p) {
    realize p2n:PrimitiveToName |
    p2n.owner := p2s;
    p2n.primitive := prim;
    p2n.typeName := sqlType;
  }
  map {
    where () {
      p2name := prim.name + '2' + sqlType
    }
  }
}

map integerToNumber in umlRdbms
refines primitiveToName {
  uml () {
    prim.name = 'Integer';
  }
  check enforce rdbms () {
    sqlType := 'NUMBER';
  }
}
```

# MODELLING VIEW OF MODEL TRANSFORMATIONS

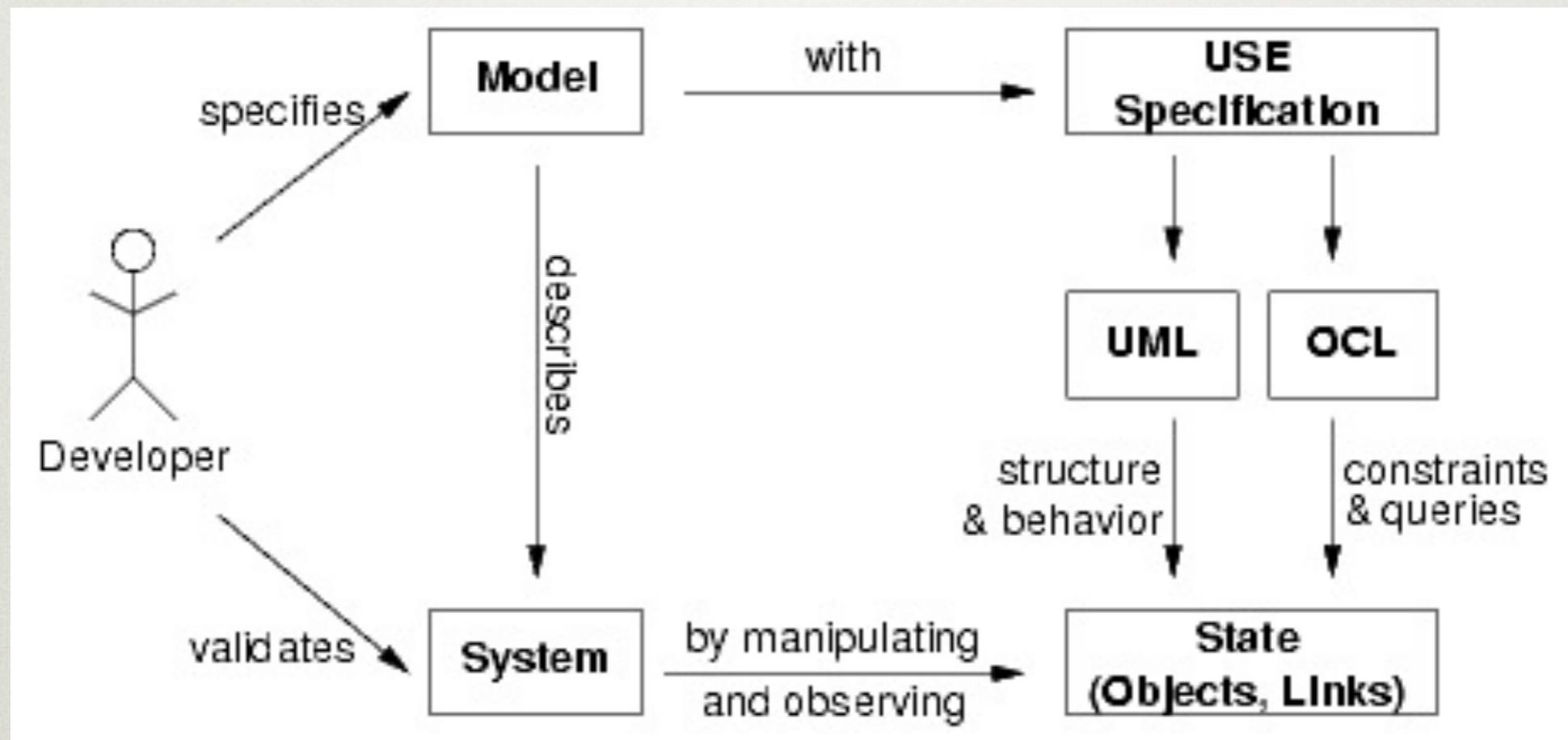


# WHAT ARE TRANSFORMATION MODELS?

---

- Transformation models express transformations as a MOF model relating source and target elements , including OCL constraints.

# UML-BASED SPECIFICATION ENVIRONMENT (USE)



Transformation model of Entity Relationship into a Relational model , is specified and validated using USE.

# EXAMPLE: ER TO REL TRANSFORMATION MODEL

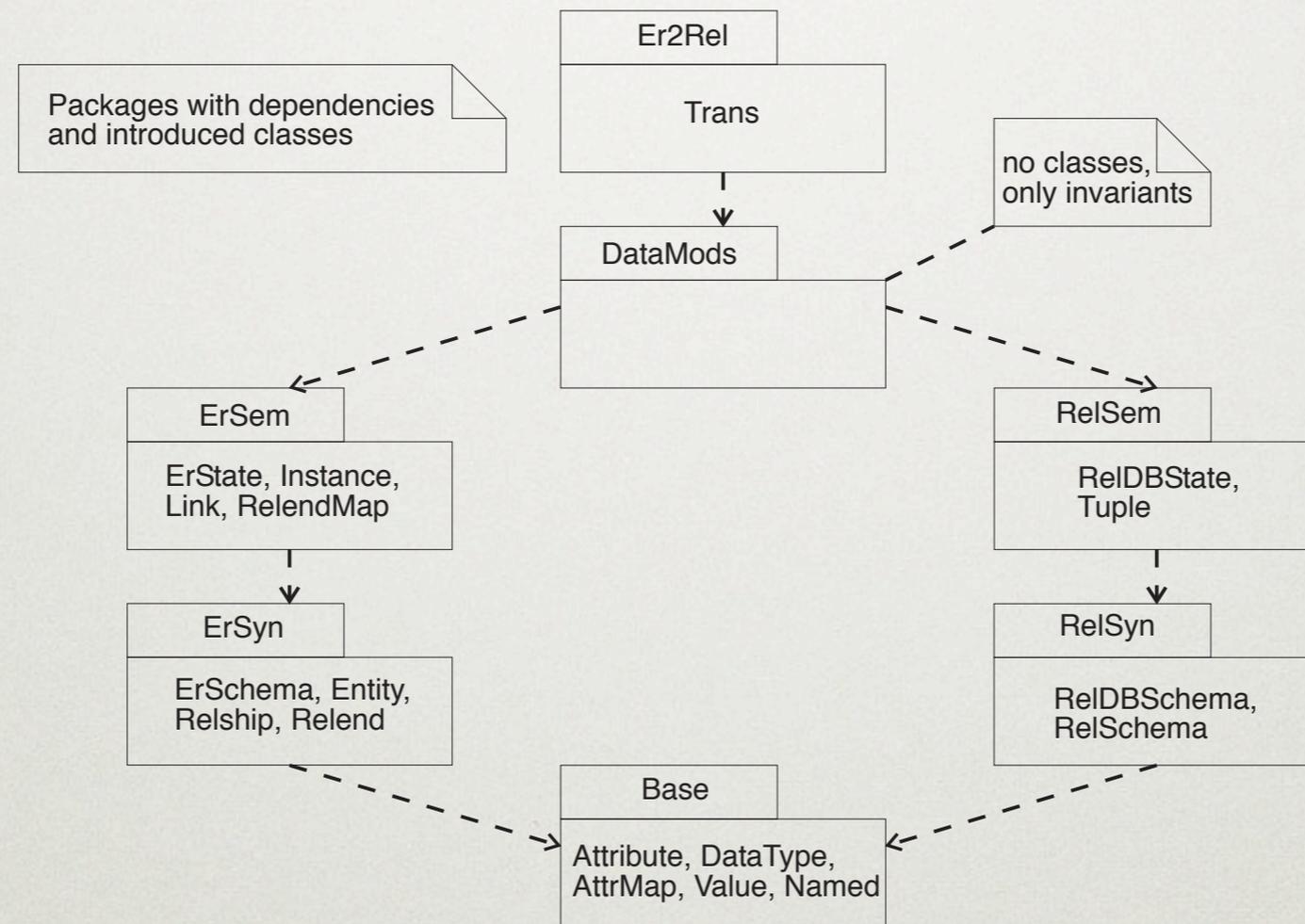
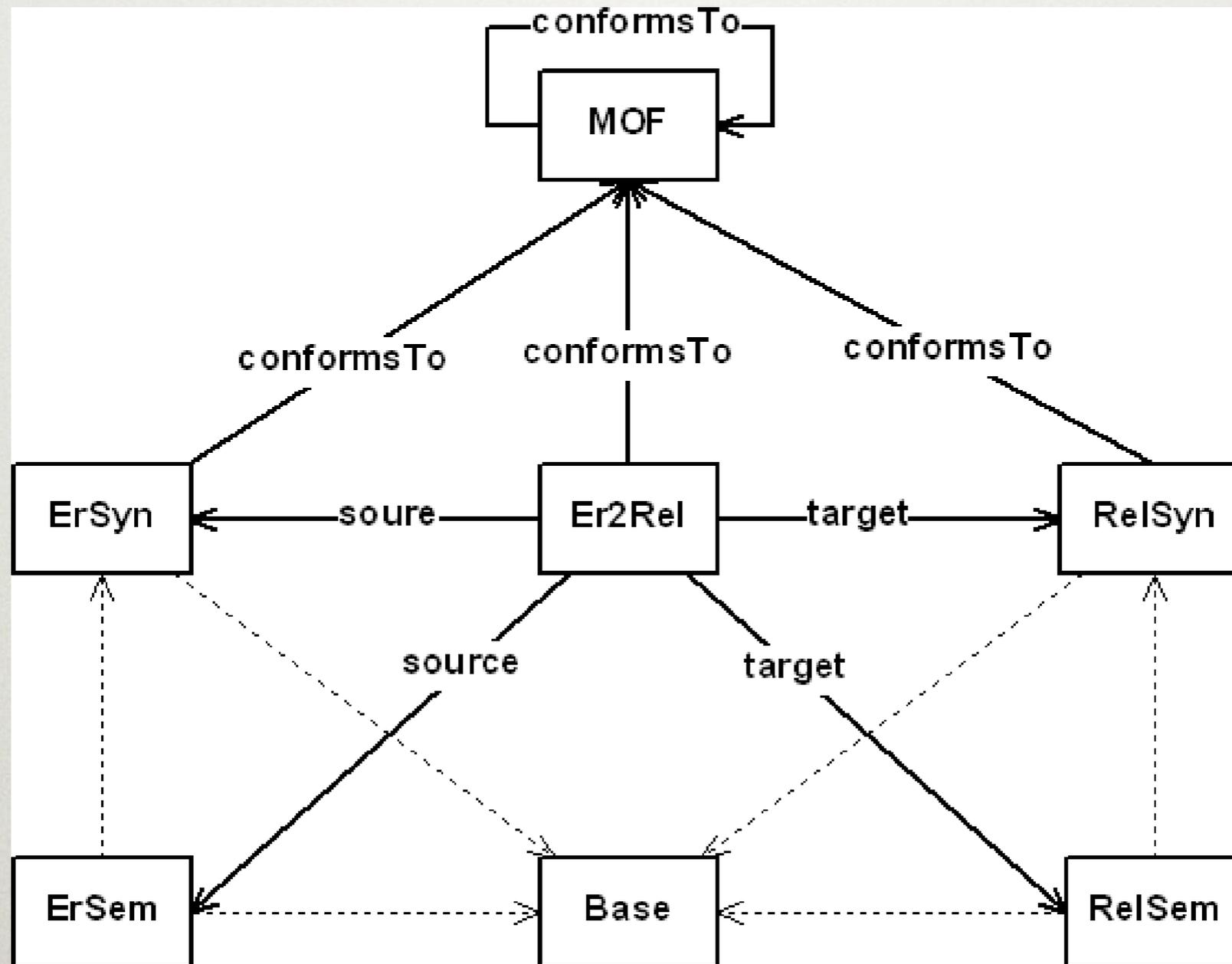
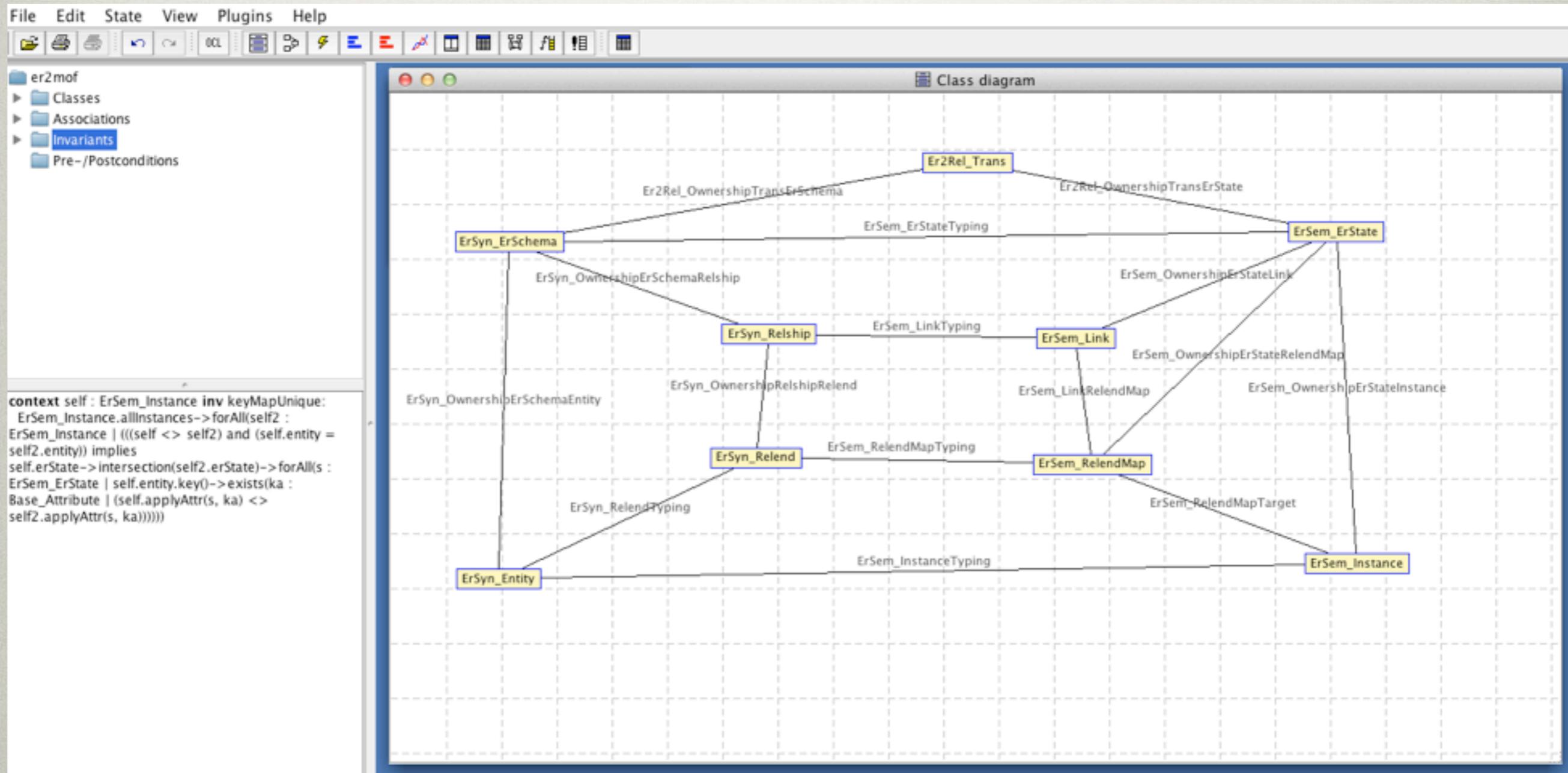


Figure 2.1: Package Diagram with Class Names

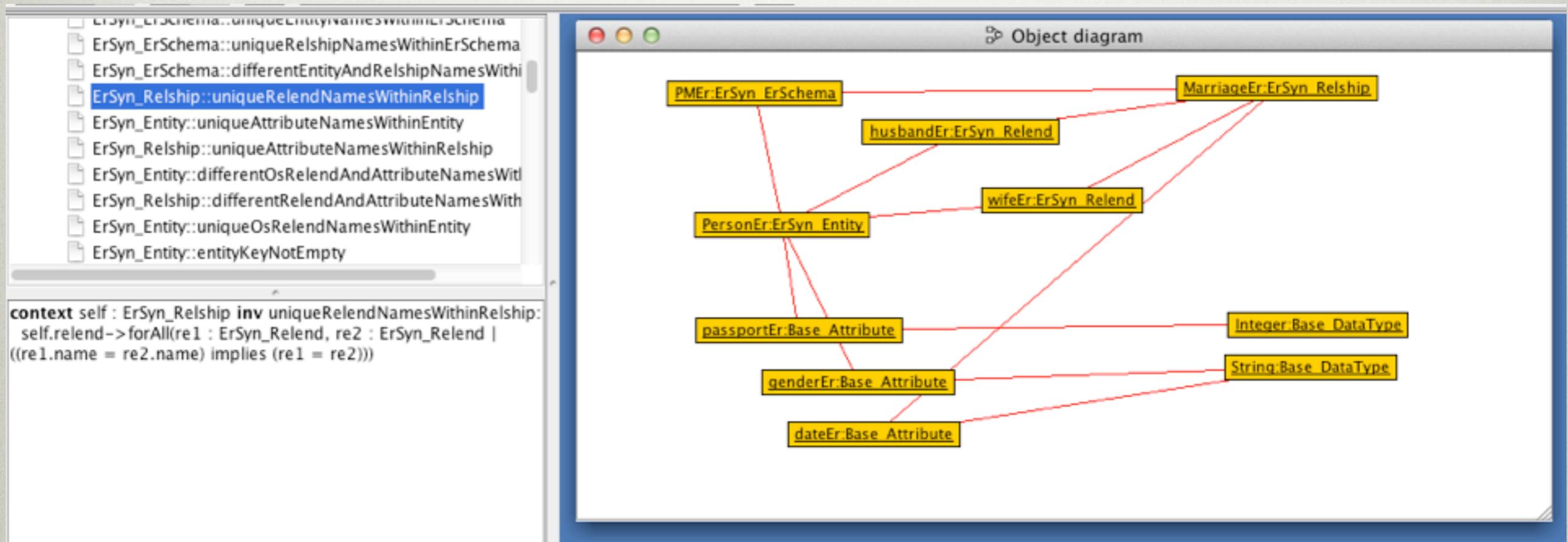
# EXAMPLE: ER TO REL TRANSFORMATION MODEL



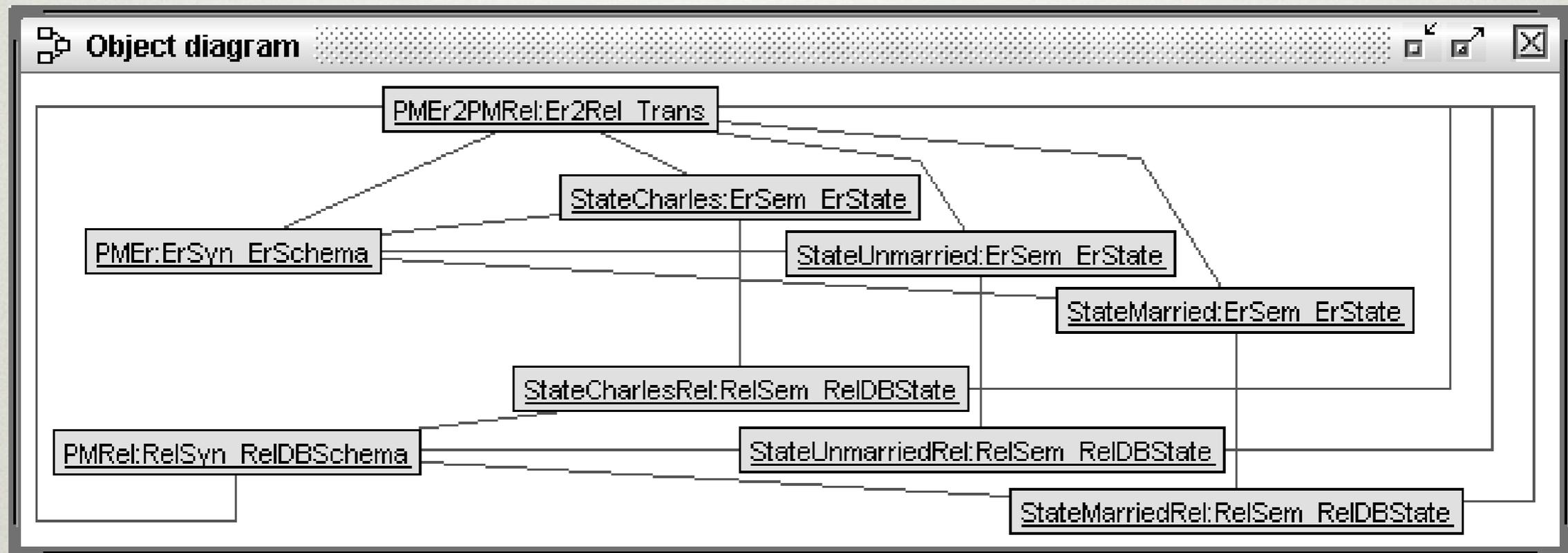
# SYNTAX AND SEMANTICS METAMODELS FOR THE ER MODEL



# SYNTAX FOR THE ER MODEL

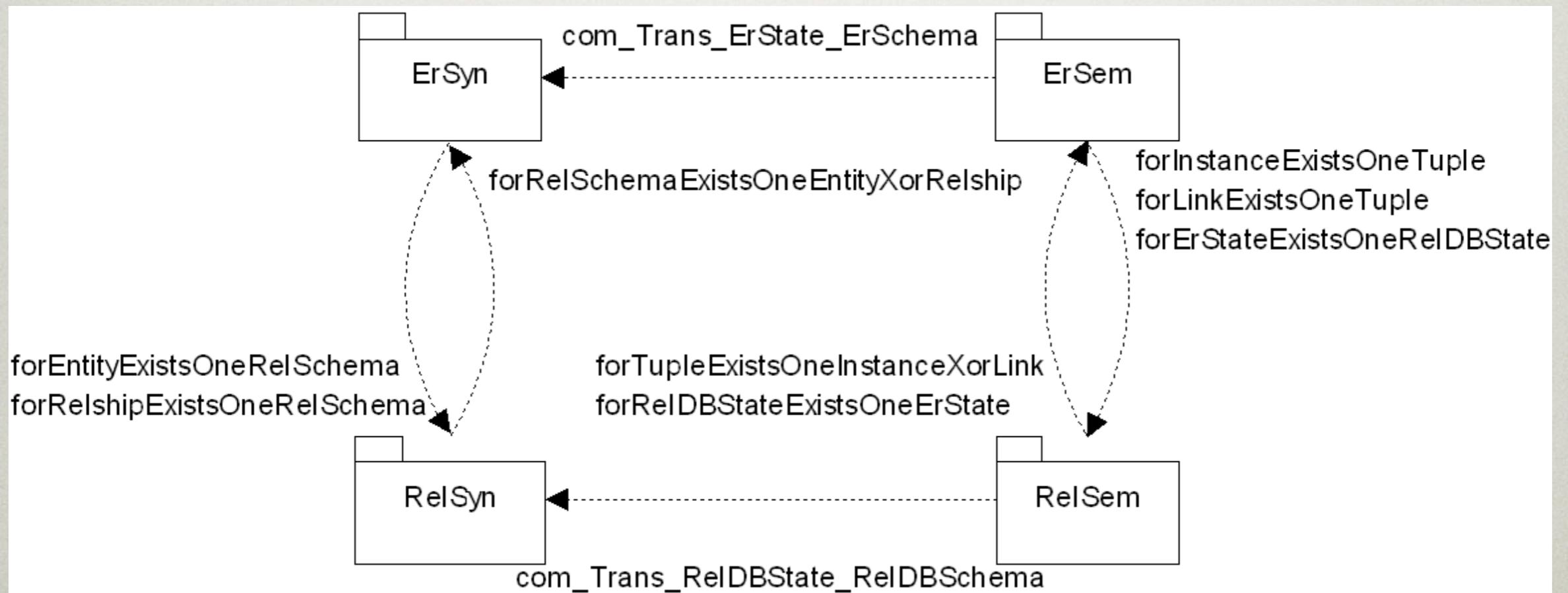


# TRANSFORMATION MODEL OBJECT DIAGRAM



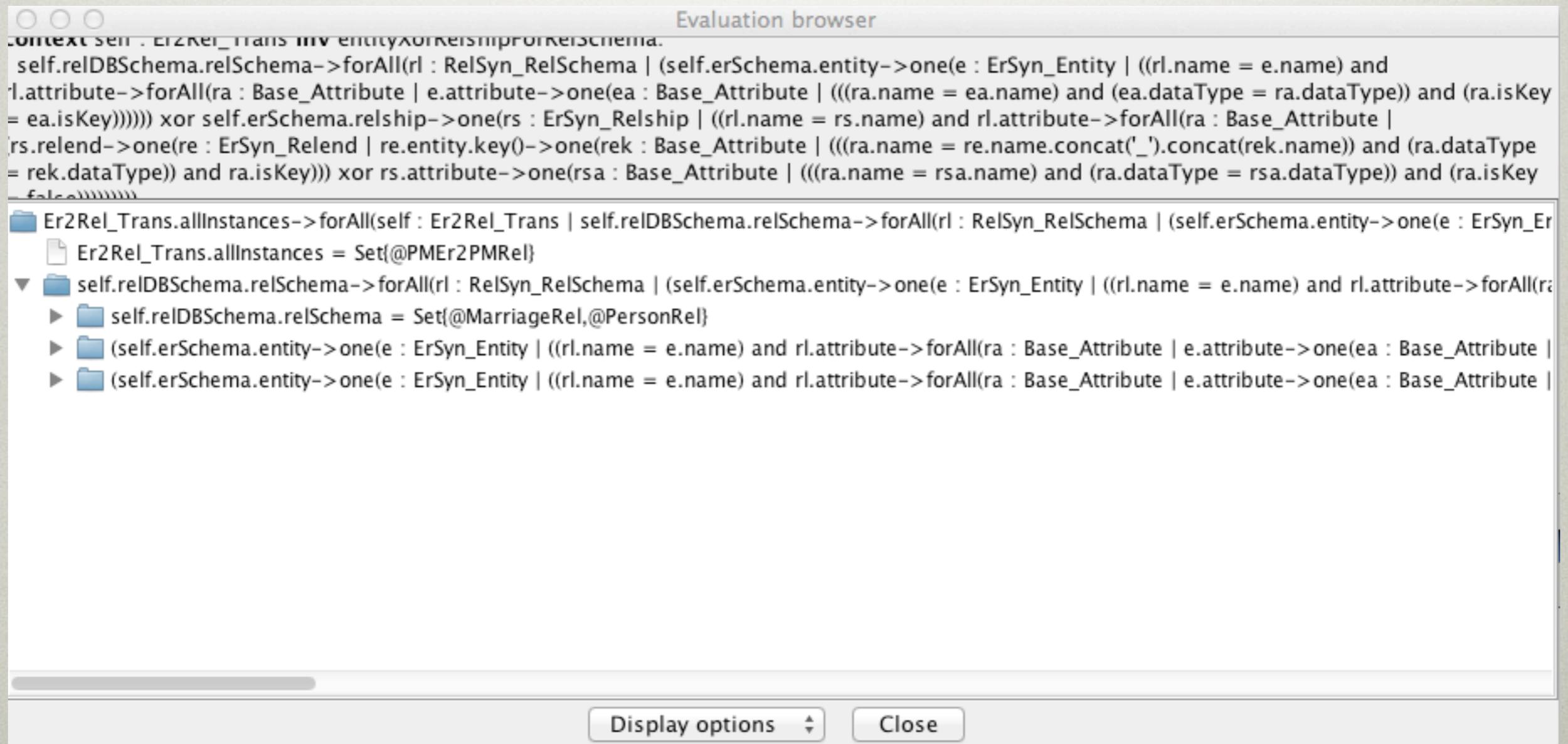
# CONSTRAINTS CAPTURED IN THE ER2REL TRANSFORMATION MODEL

- Er2Rel captures 10 constraints:



# CONSTRAINT I

- forRelSchemaExistsOneEntityXorRelship



The screenshot shows an "Evaluation browser" window. The top part displays a logical expression in a query language, likely OCL, defining a constraint. The expression is: `self.relDBSchema.relSchema->forAll(rl : RelSyn_RelSchema | (self.erSchema.entity->one(e : ErSyn_Entity | ((rl.name = e.name) and rl.attribute->forAll(ra : Base_Attribute | e.attribute->one(ea : Base_Attribute | (((ra.name = ea.name) and (ea.dataType = ra.dataType)) and (ra.isKey = ea.isKey)))))) xor self.erSchema.relship->one(rs : ErSyn_Relship | ((rl.name = rs.name) and rl.attribute->forAll(ra : Base_Attribute | (rs.relend->one(re : ErSyn_Relend | re.entity.key()->one(rek : Base_Attribute | (((ra.name = re.name.concat('_').concat(rek.name)) and (ra.dataType = rek.dataType)) and ra.isKey))) xor rs.attribute->one(rsa : Base_Attribute | (((ra.name = rsa.name) and (ra.dataType = rsa.dataType)) and (ra.isKey = false))))))`

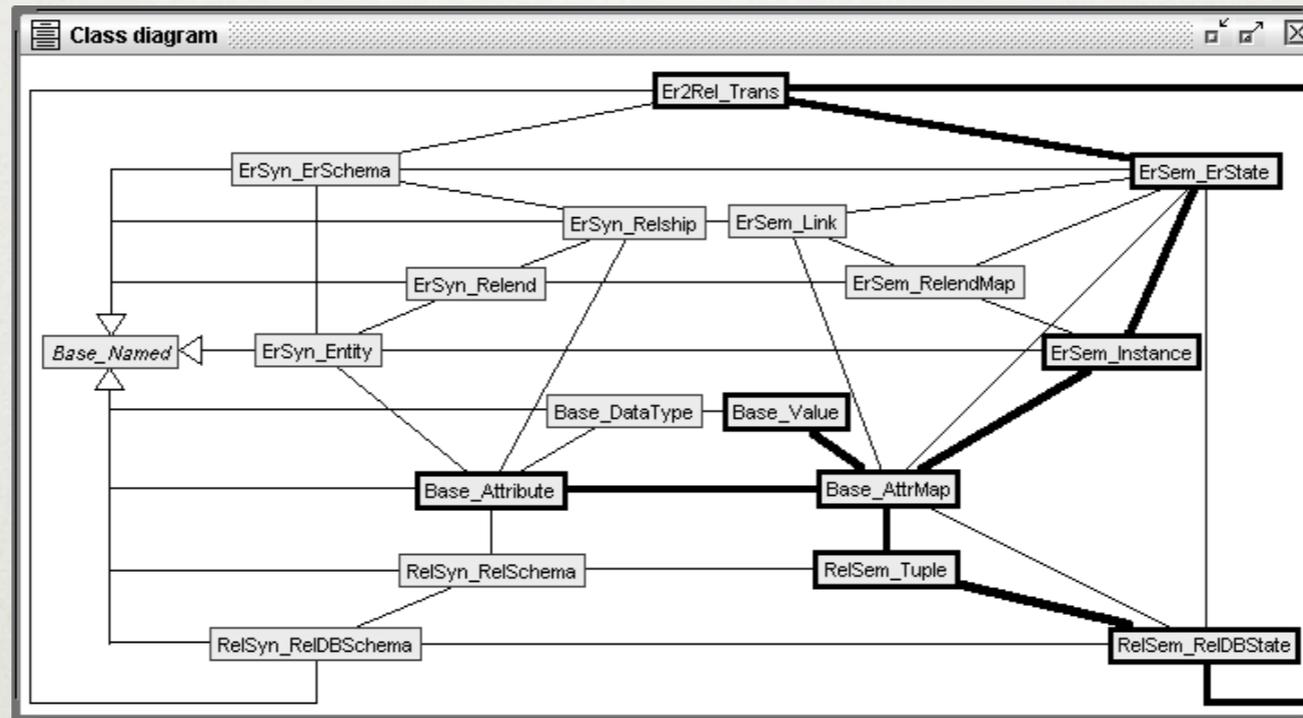
The bottom part of the window shows the evaluation tree for this expression. The root node is `Er2Rel_Trans.allInstances->forAll(self : Er2Rel_Trans | self.relDBSchema.relSchema->forAll(rl : RelSyn_RelSchema | (self.erSchema.entity->one(e : ErSyn_Entity | ...))`. The tree is expanded to show the following structure:

- Er2Rel\_Trans.allInstances = Set{@PMEr2PMRel}
- self.relDBSchema.relSchema = Set{@MarriageRel, @PersonRel}
- (self.erSchema.entity->one(e : ErSyn\_Entity | ((rl.name = e.name) and rl.attribute->forAll(ra : Base\_Attribute | e.attribute->one(ea : Base\_Attribute | ...))
- (self.erSchema.entity->one(e : ErSyn\_Entity | ((rl.name = e.name) and rl.attribute->forAll(ra : Base\_Attribute | e.attribute->one(ea : Base\_Attribute | ...))

At the bottom of the window, there are two buttons: "Display options" and "Close".

# CONSTRAINT II

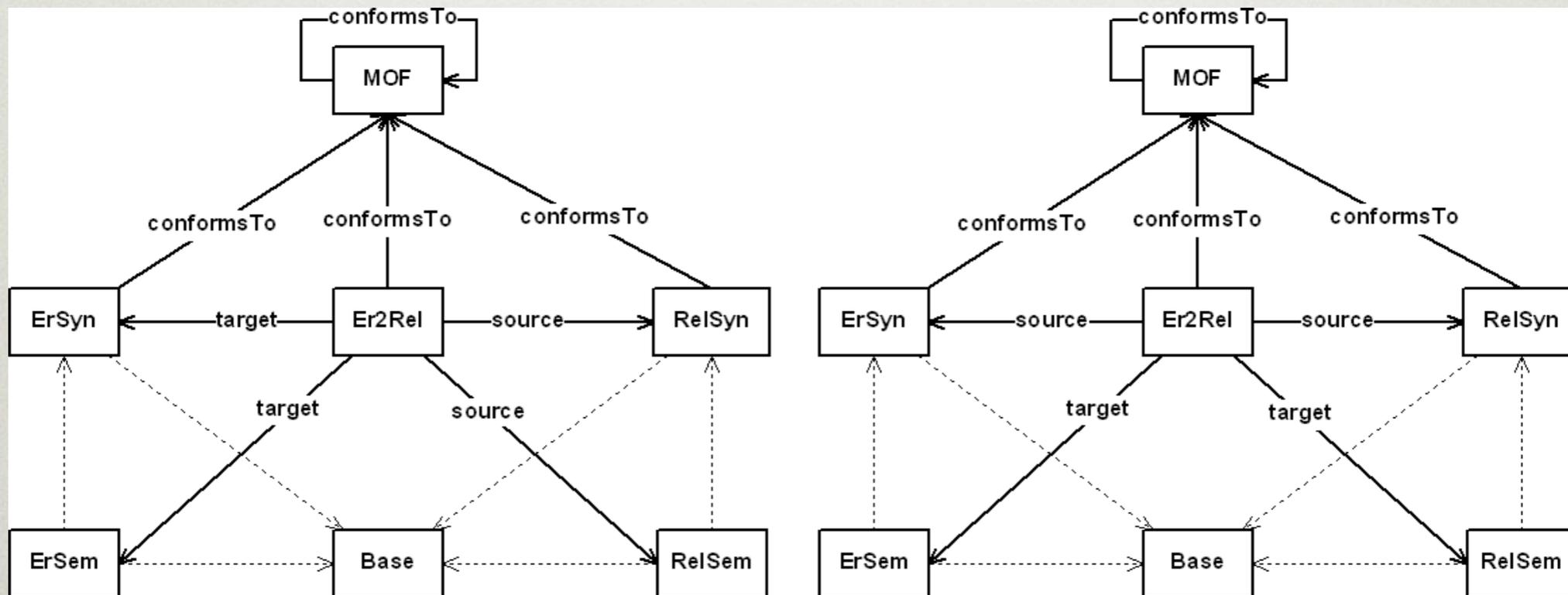
- forInstanceExistsOneTuple



```
context self:Er2Rel_Trans inv forInstanceExistsOneTuple:
  self.erState->forAll(erSt | self.relDBState->one(relSt |
    erSt.instance->forAll(i | relSt.tuple->one(t |
      i.attrMap->forAll(amEr |
        t.attrMap->one(amRel |
          amEr.attribute.name=amRel.attribute.name and
          amEr.value=amRel.value))))))
```

# ADVANTAGES OF TRANSFORMATION MODELS

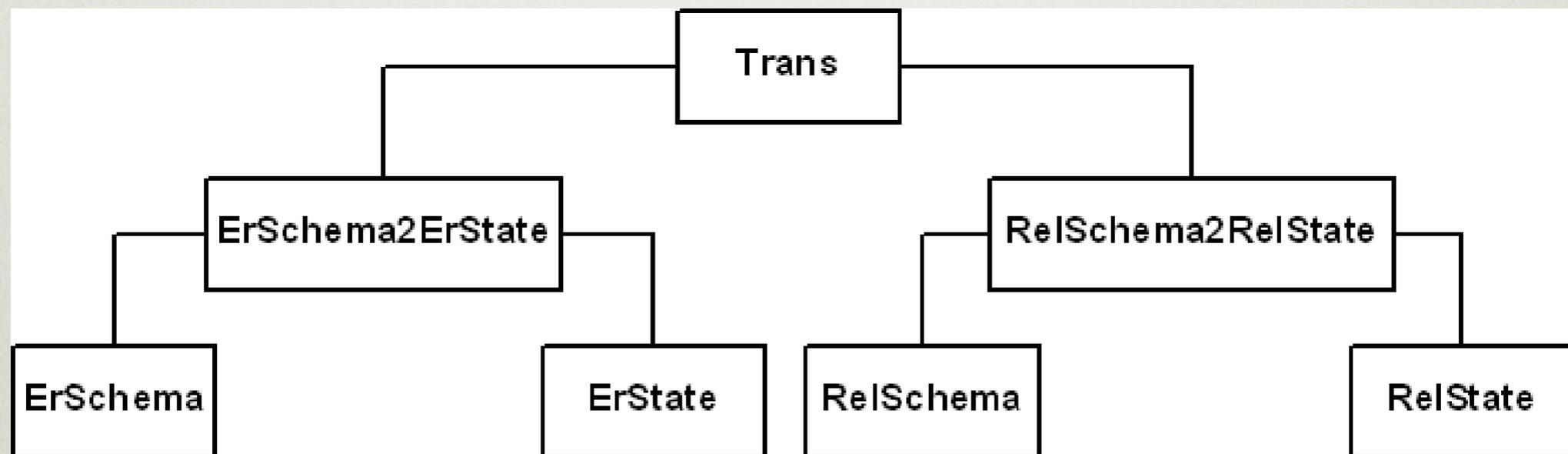
- Direction Neutral



# ADVANTAGES OF TRANSFORMATION MODELS

---

- Higher Order Transformations



# ADVANTAGES OF TRANSFORMATION MODELS

---

- Uniformity of Development
  - Avoiding QVT and using only UML and OCL.
- Transformations of Transformations
  - E.g Refactorings

# ADVANTAGES OF TRANSFORMATION MODELS

- Validations and Completions.
- “Analysis of Model Transformations via Alloy”.

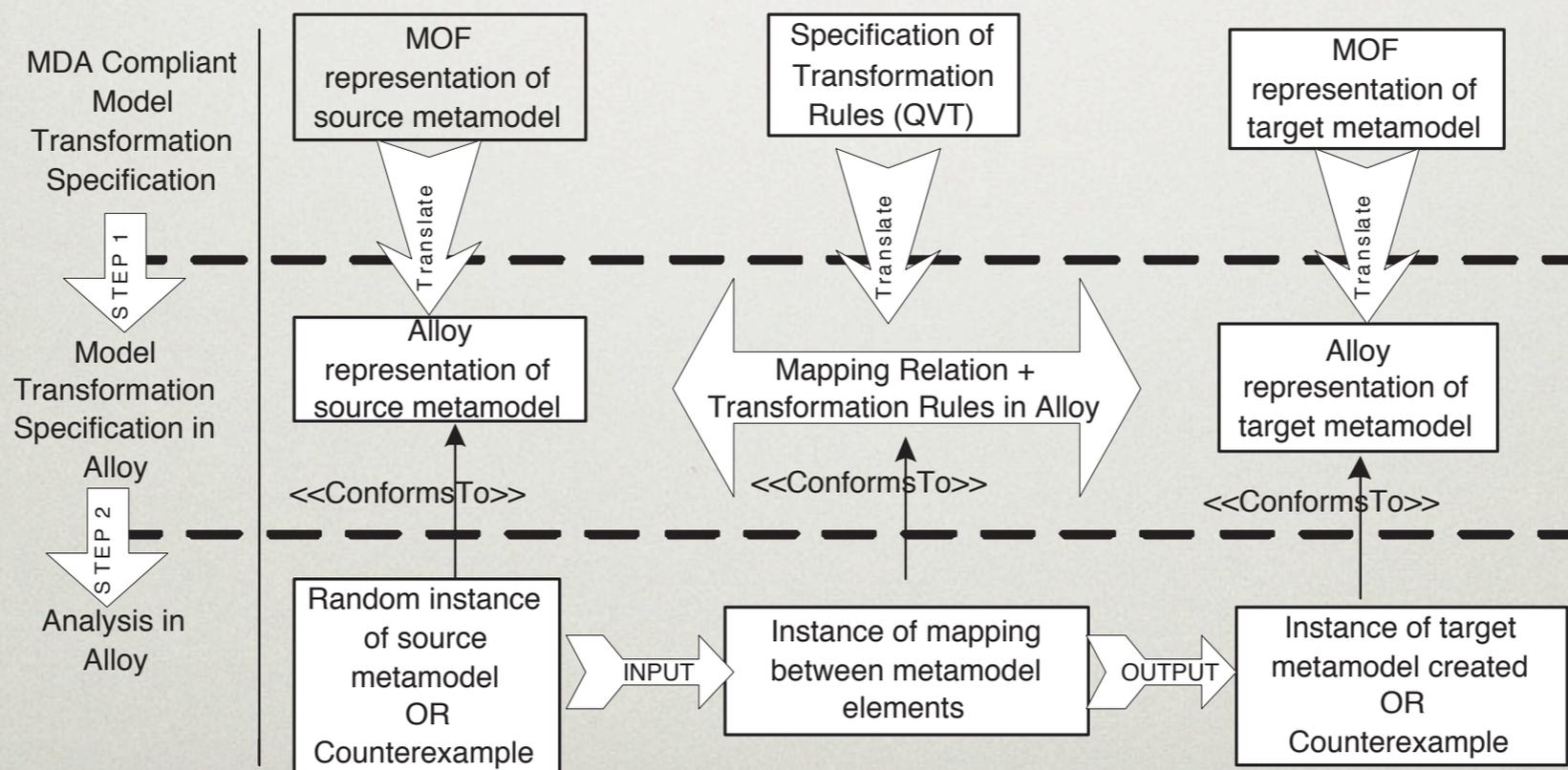


Fig. 1. An outline of our approach

# ADVANTAGES OF TRANSFORMATION MODELS

---

- Complete Language Description.
- Syntax and Semantics part.

# DISCUSSION QUESTIONS

---

- Why would you prefer a transformation model instead of standard QVT to describe transformation?
- Would it be possible to automatically complete transformations defined by the transformation model?
- What are the advantage of partitioning the models into semantic and syntax part?