

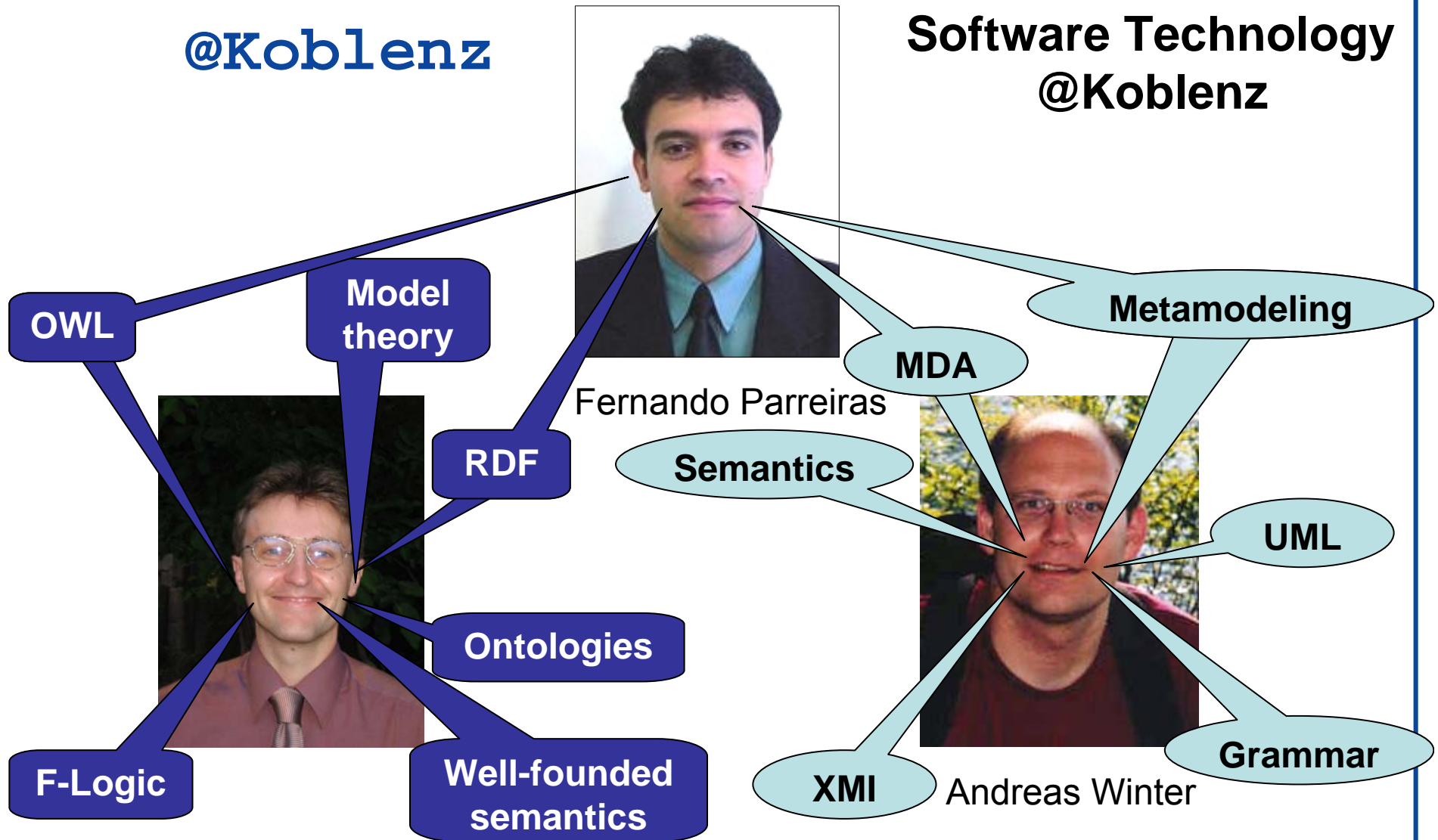
Joint Metamodels for UML and OWL

Steffen Staab



<isweb>
@Koblenz

IST – Institute for
Software Technology
@Koblenz

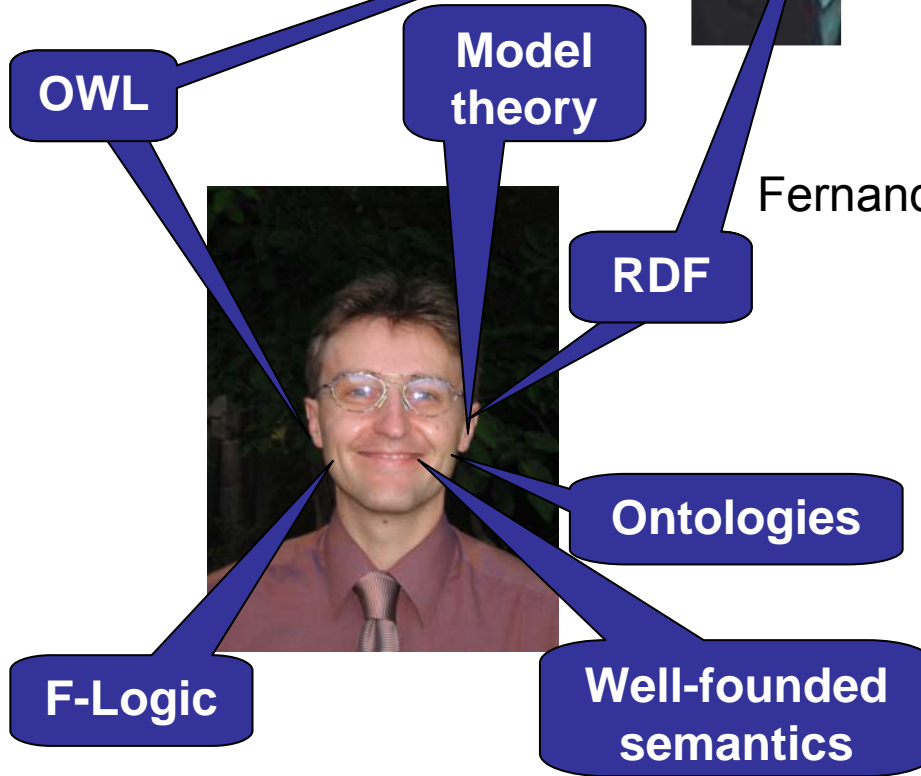


Schism?

<isweb>
@Koblenz



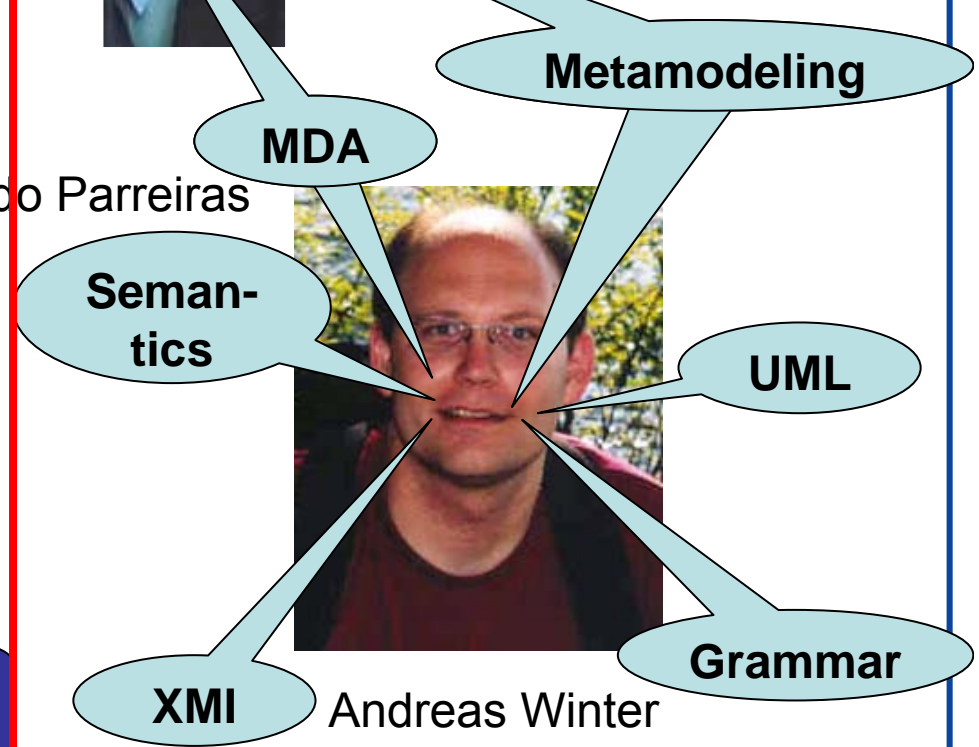
Fernando Parreiras



IST – Institute for
Software Technology
@Koblenz



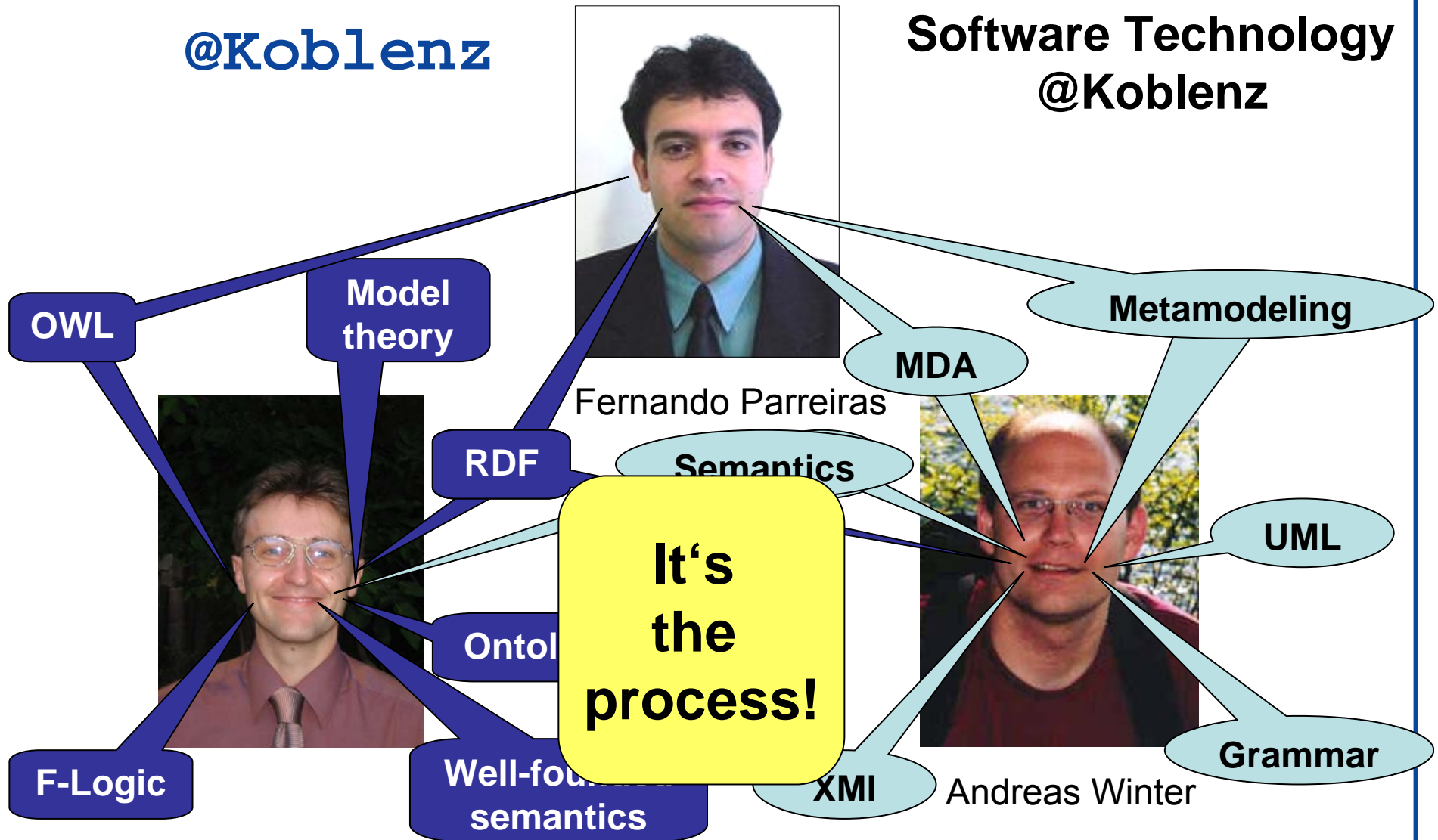
Andreas Winter



Synthesis!

<isweb>
@Koblenz

IST – Institute for
Software Technology
@Koblenz

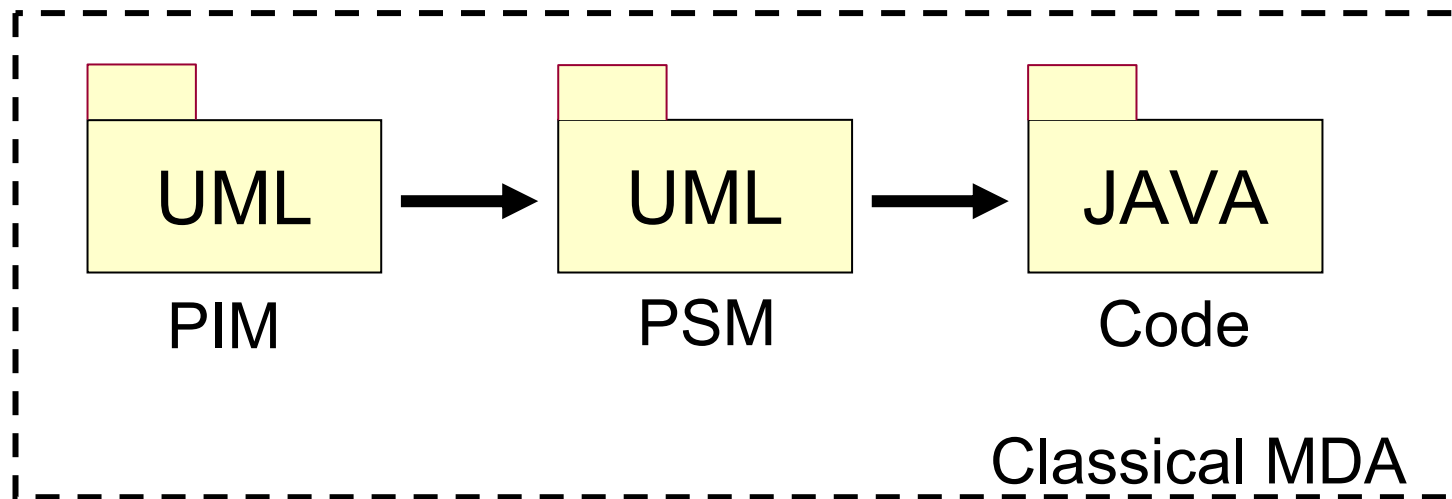


- ◆ **Model-driven Engineering**
- ◆ Ontology
- ◆ A Use Case of Ontology Technology in MDE
- ◆ Joint Metamodels
- ◆ Case 1:
Use Ontology Technology in a Design Pattern
- ◆ Case 2:
Using MDE for Translating between Ontologies

MDA is an instance of Model-Driven Engineering

Transformations:

- ◆ Adapt to the target platform
- ◆ Add additional modeling



- ◆ Model-driven Engineering
- ◆ **Ontology**
- ◆ A Use Case of Ontology Technology in MDE
- ◆ Joint Metamodels
- ◆ Case 1:
Use Ontology Technology in a Design Pattern
- ◆ Case 2:
Using MDE for Translating between Ontologies

Definition: What is an ontology? (in computer science)

Based on Gruber 93:

An Ontology is a

formal specification

⇒ Executable, Discussable

of a shared

⇒ Group of persons

conceptualization

⇒ About concepts

of a domain of interest ⇒ Between application
and „unique truth“

To make domain assumptions **explicit**

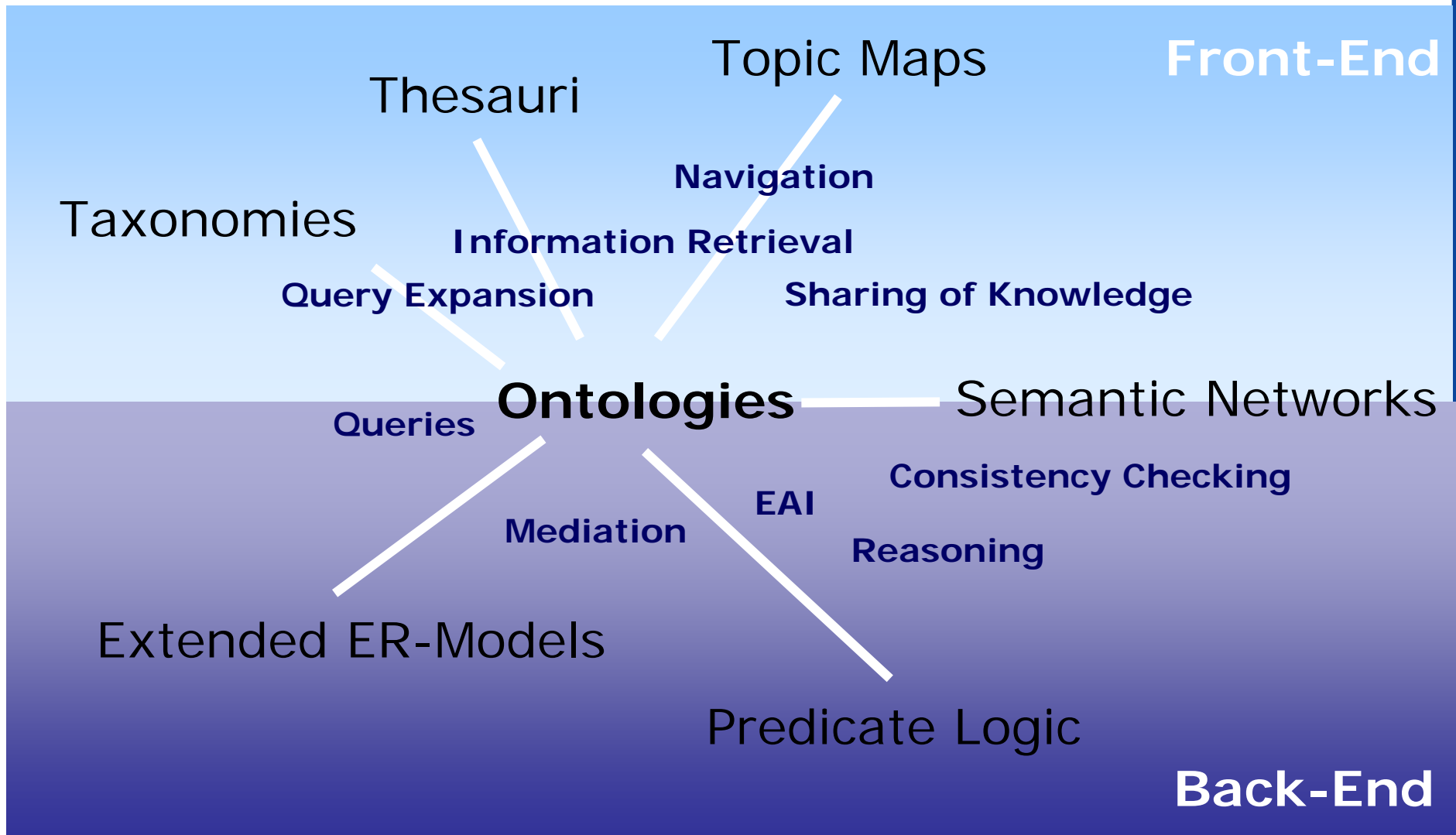
- ◆ Easier to change domain assumptions
- ◆ Easier to understand and update legacy data

To separate **domain knowledge** from operational knowledge

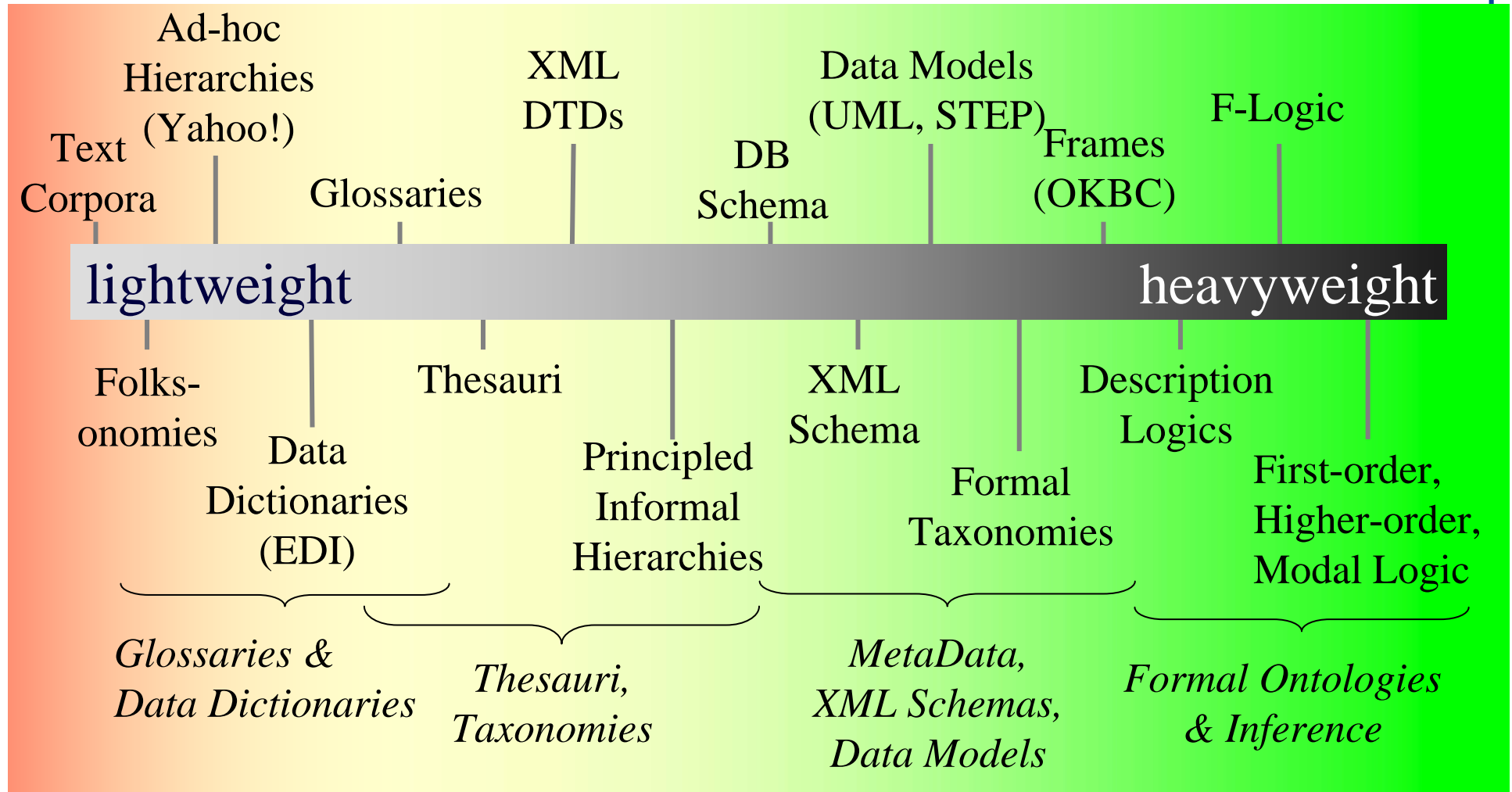
- ◆ Re-use domain and operational knowledge separately

A **community reference** for applications

To **share a consistent understanding** of what information means



Formality: What is an ontology?

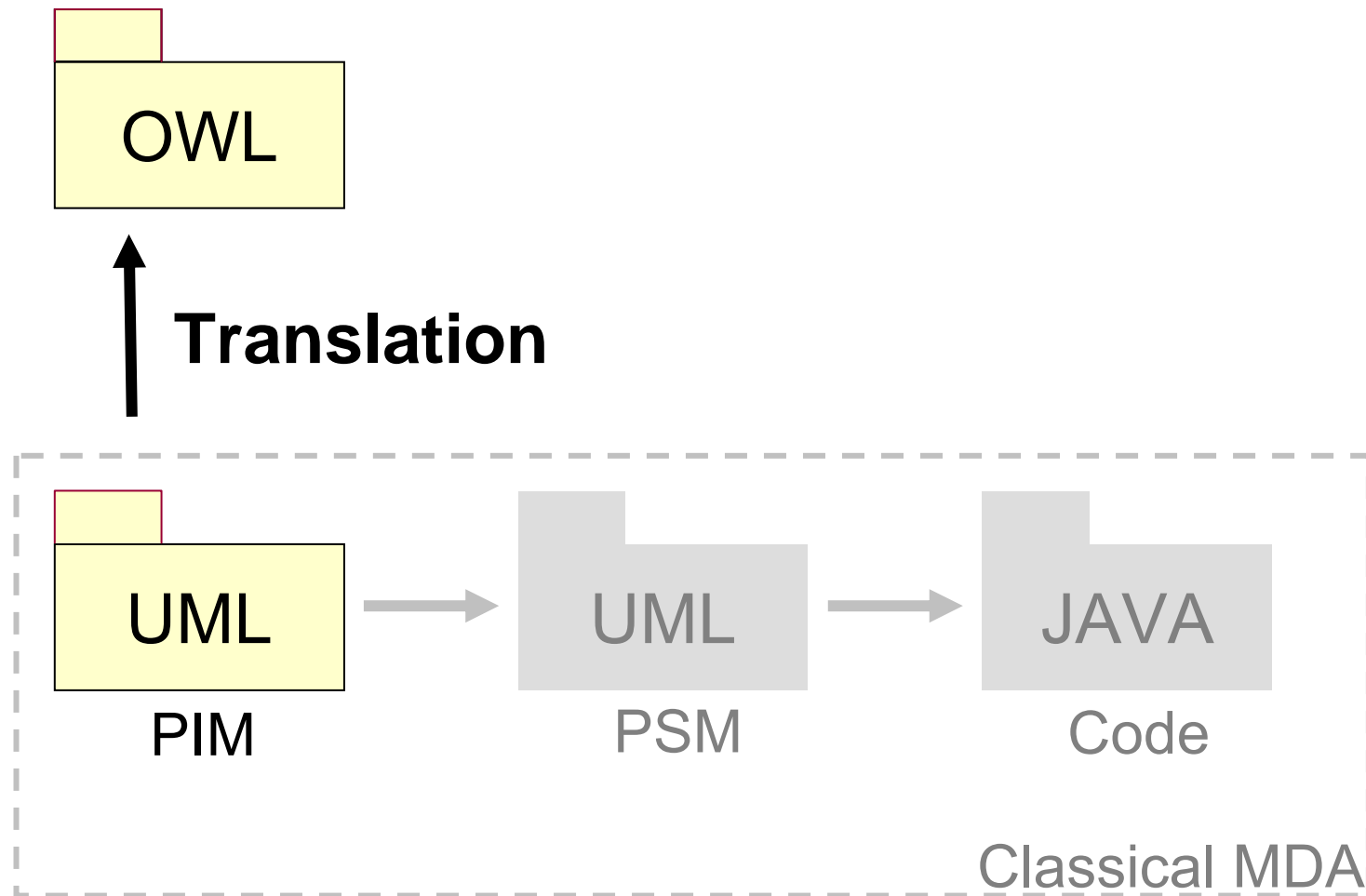


Foundational Model of Anatomy

- ◆ Represents structures ranging from macromolecular complexes to body parts
- ◆ Contains
 - ◆ ~70,000 distinct concepts
 - ◆ ~ 110,000 terms
 - ◆ 140 relations
 - ◆ Metaclasses to define class-level properties
 - ◆ Attributed relations
 - ◆ Different types of part-whole, location, and other spatial relations
 - ◆ Synonyms

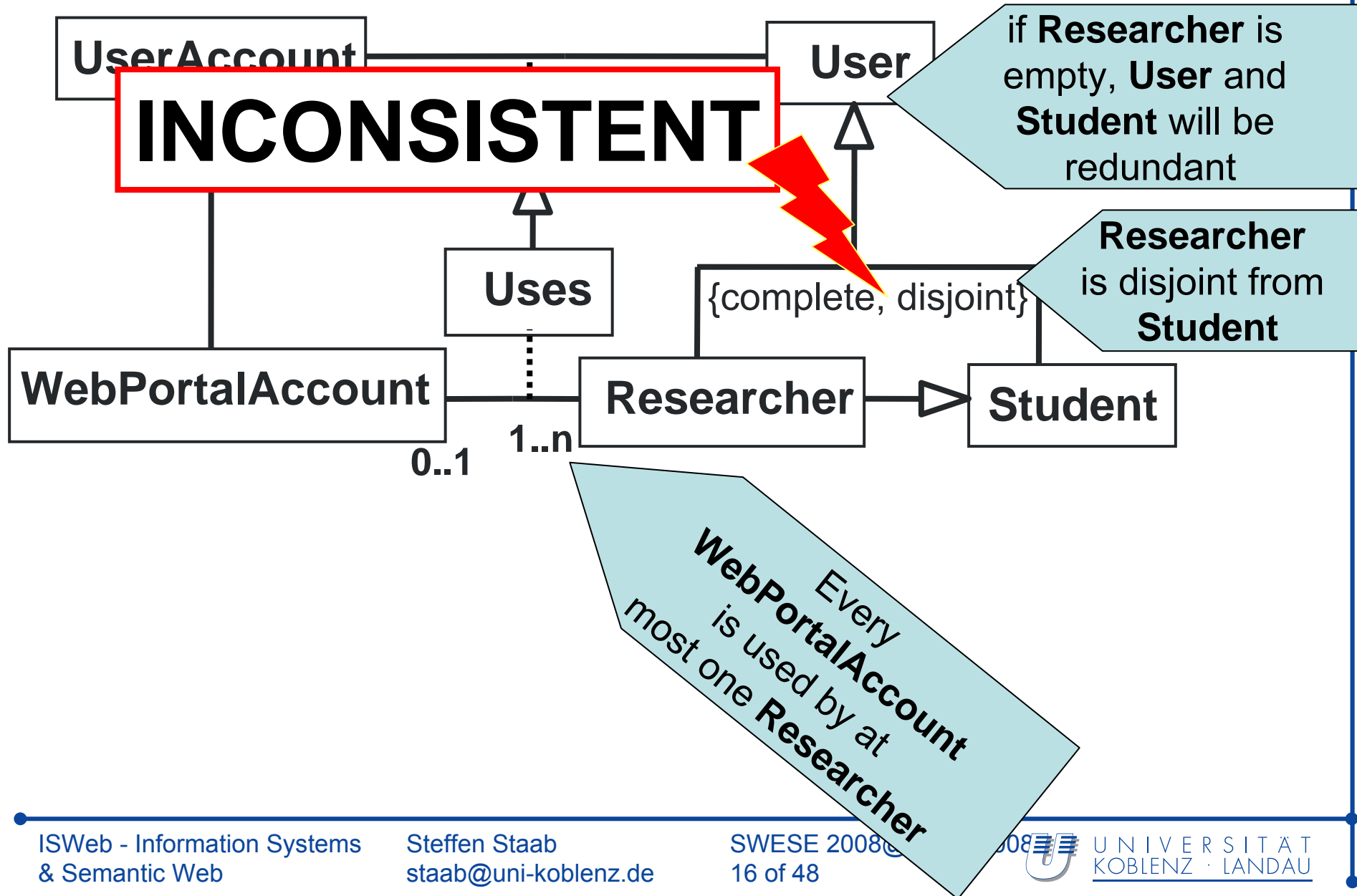
- ◆ Model-driven Engineering
- ◆ Ontology
- ◆ **A Use Case of Ontology Technology in MDE**
- ◆ Joint Metamodels
- ◆ Case 1:
Use Ontology Technology in a Design Pattern
- ◆ Case 2:
Using MDE for Translating between Ontologies

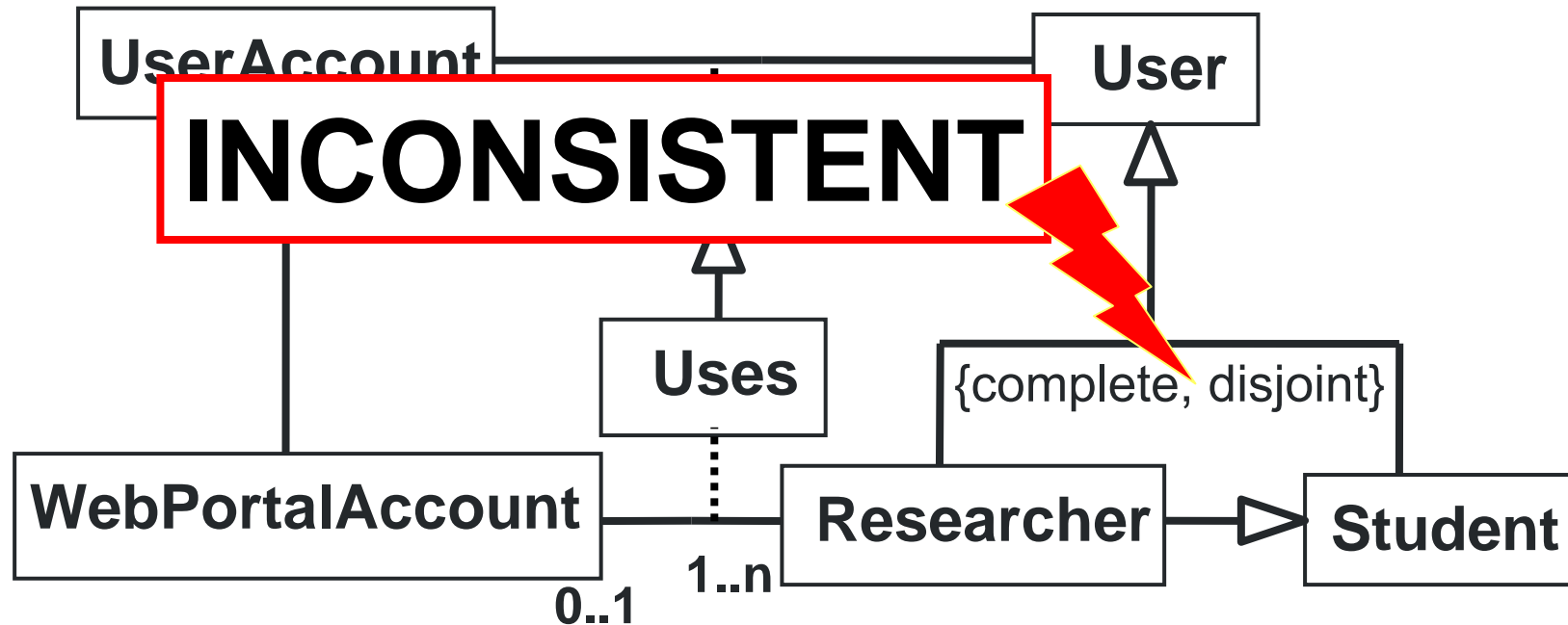
Reasoning on UML class diagrams



Reasoning on UML class diagrams allows for checking:

- ◆ **Consistency of the diagram:**
Can the classes be populated?
- ◆ **Classification to** identify the possible omission of an explicit generalization.
- ◆ **Equivalence among classes** to discover redundancy.
- ◆ **Refinement of properties** to apply stricter multiplicities or typing than the ones explicitly specified in the diagram.

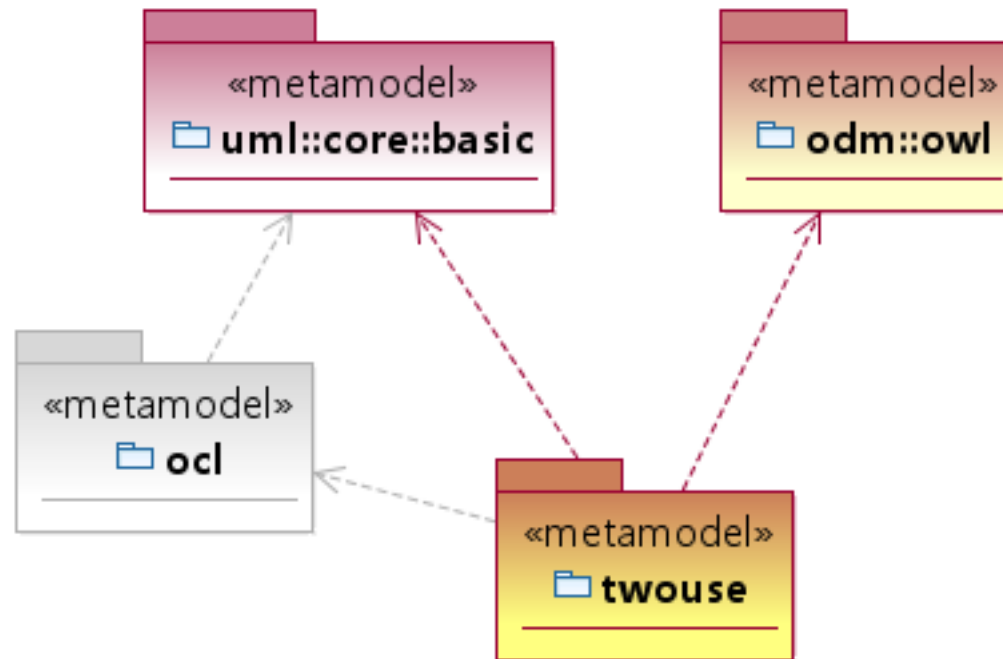


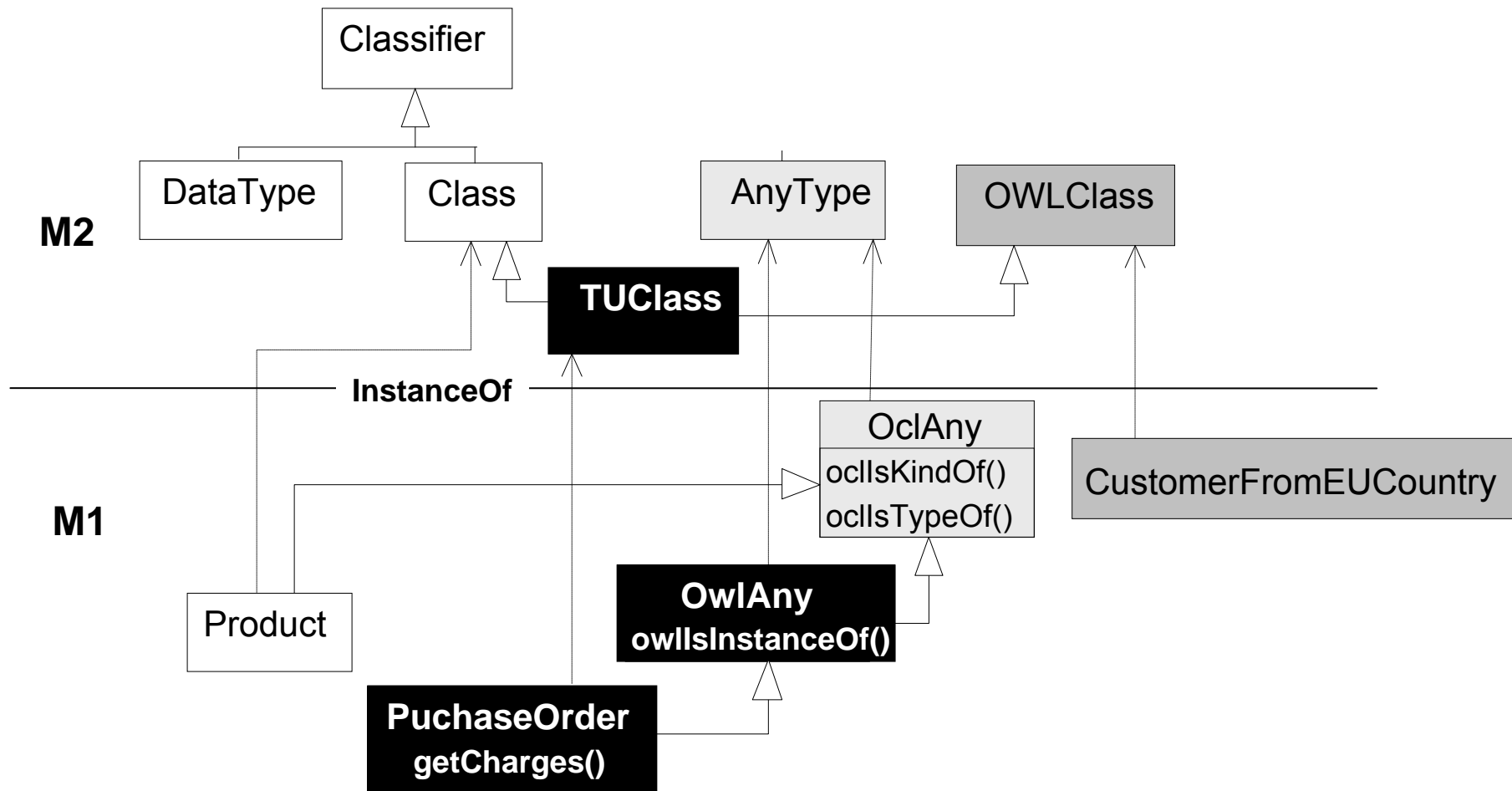


Advantage for SE:
Models with provably higher quality

- ◆ Model-driven Engineering
- ◆ Ontology
- ◆ A Use Case of Ontology Technology in MDE
- ◆ **Joint Metamodels: TWOUSE**
- ◆ Case 1:
Use Ontology Technology in a Design Pattern
- ◆ Case 2:
Using MDE for Translating between Ontologies

TWOUSE: Transforming and Weaving Ontologies and UML for Software Engineering

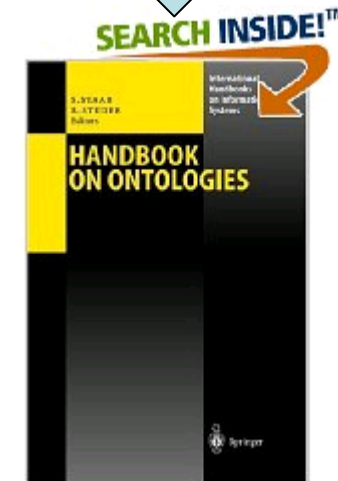
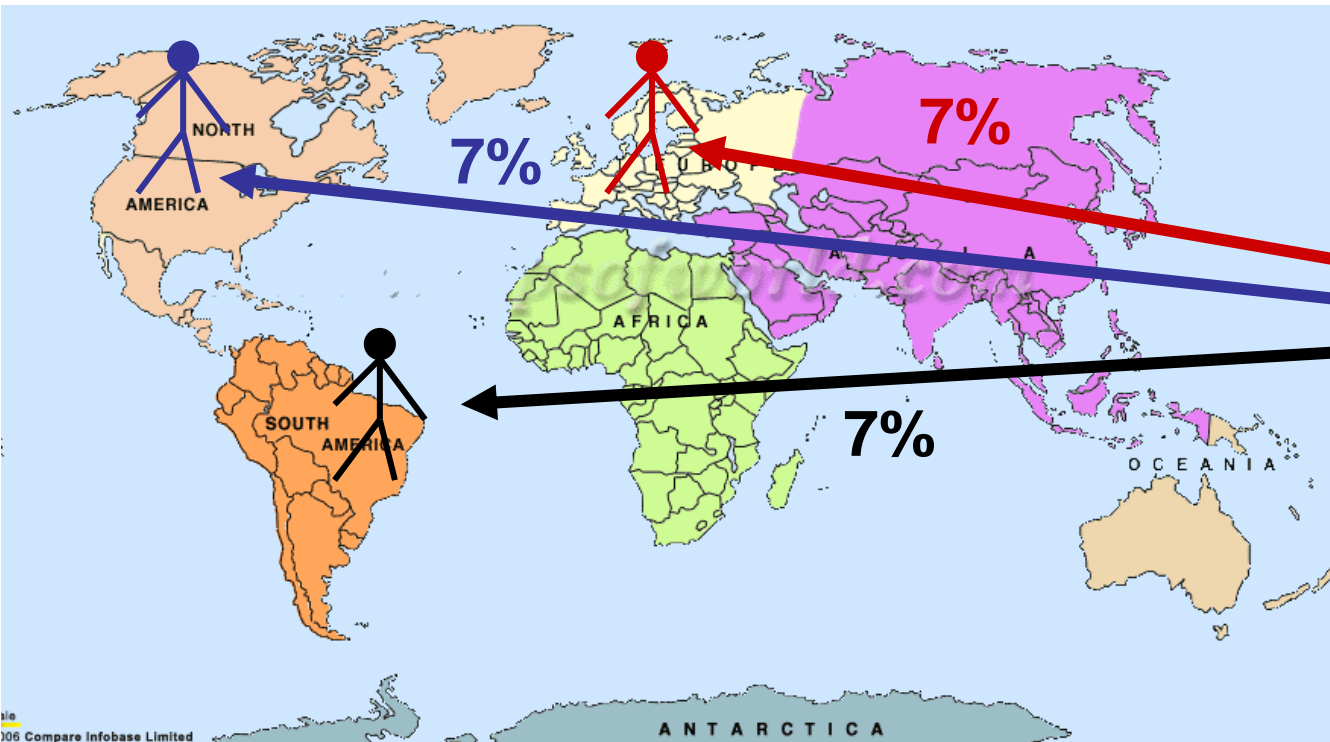
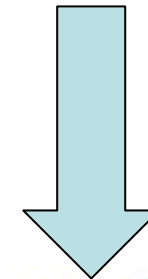
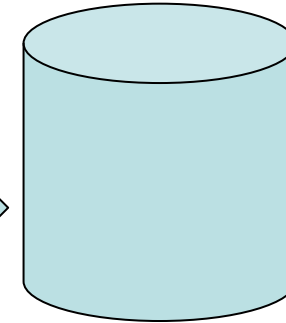
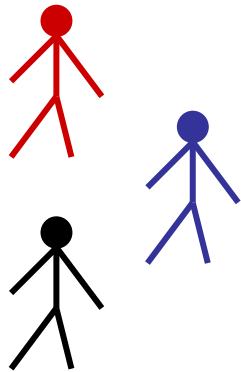




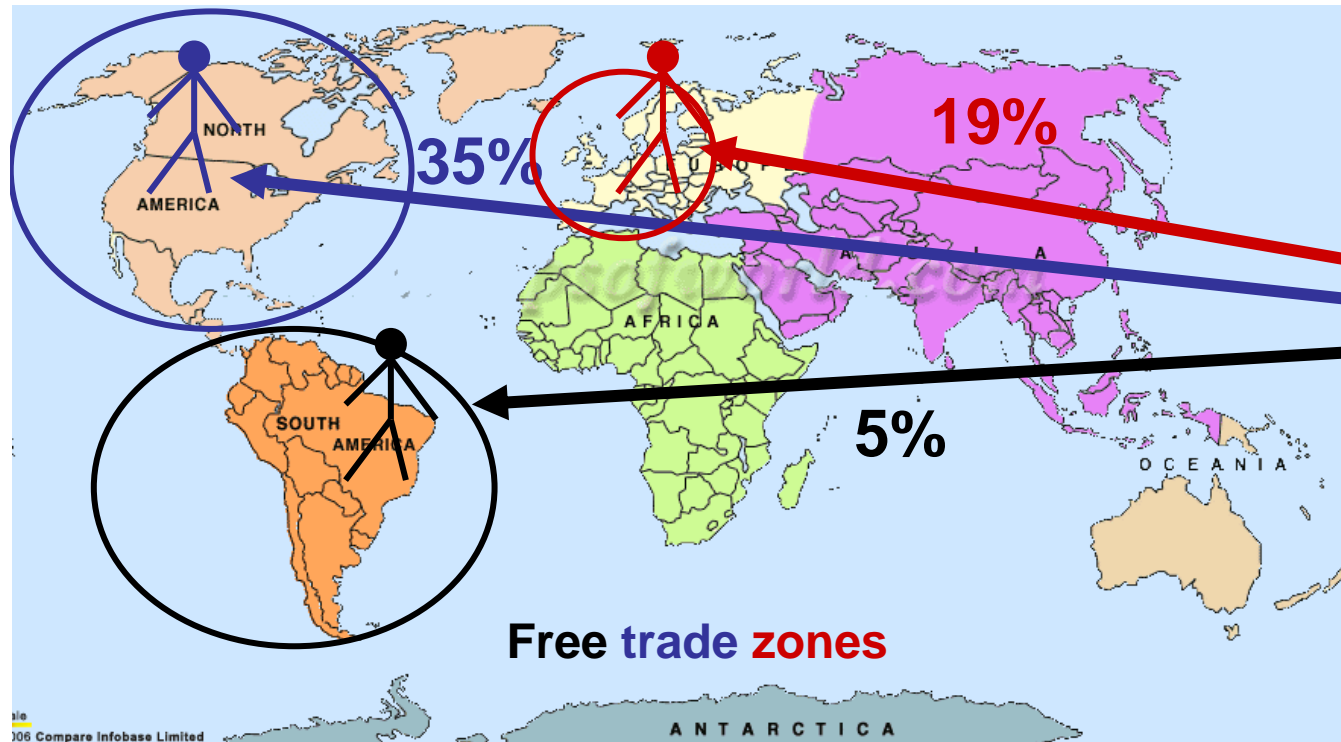
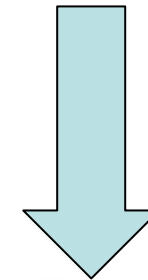
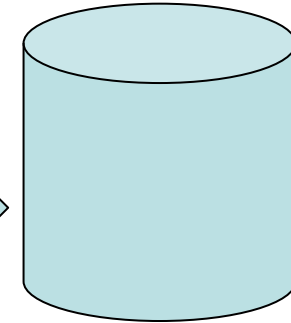
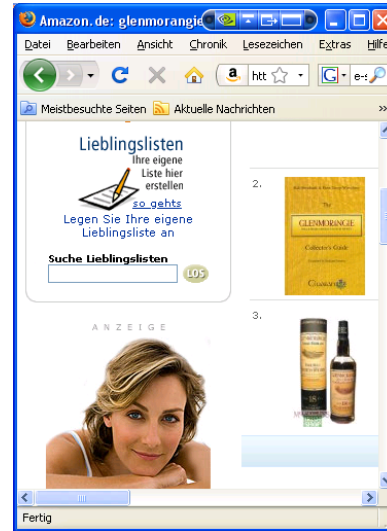
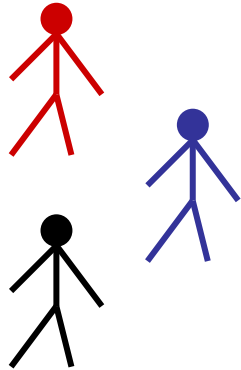
```
context PurchaseOrder::getCharges() : Real
body: if self.owlIsInstanceOf(DutyFreeOrder)
then 0.00 else 0.60 endif}
```

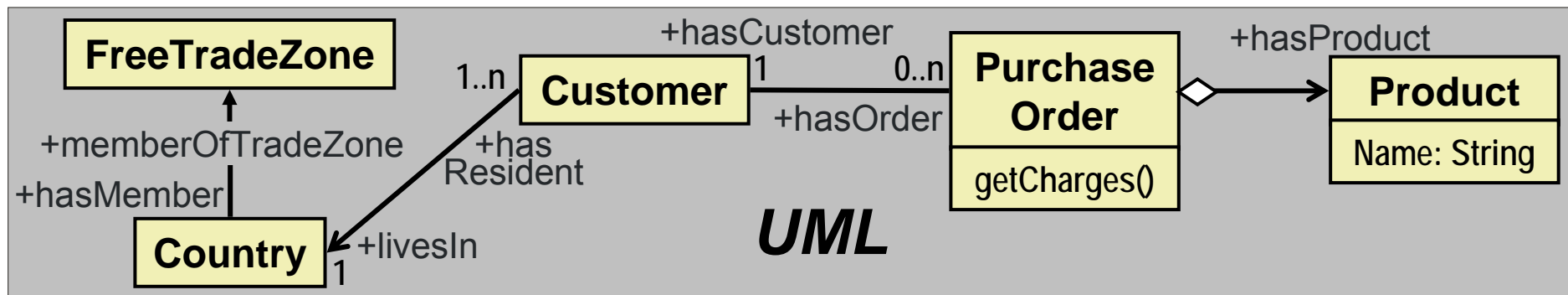
- ◆ Model-driven Engineering
- ◆ Ontology
- ◆ A Use Case of Ontology Technology in MDE
- ◆ Joint Metamodels: TWOUSE
- ◆ **Case 1:**
Use Ontology Technology in a Design Pattern
- ◆ **Case 2:**
Using MDE for Translating between Ontologies

Running Example: E-Commerce Shop

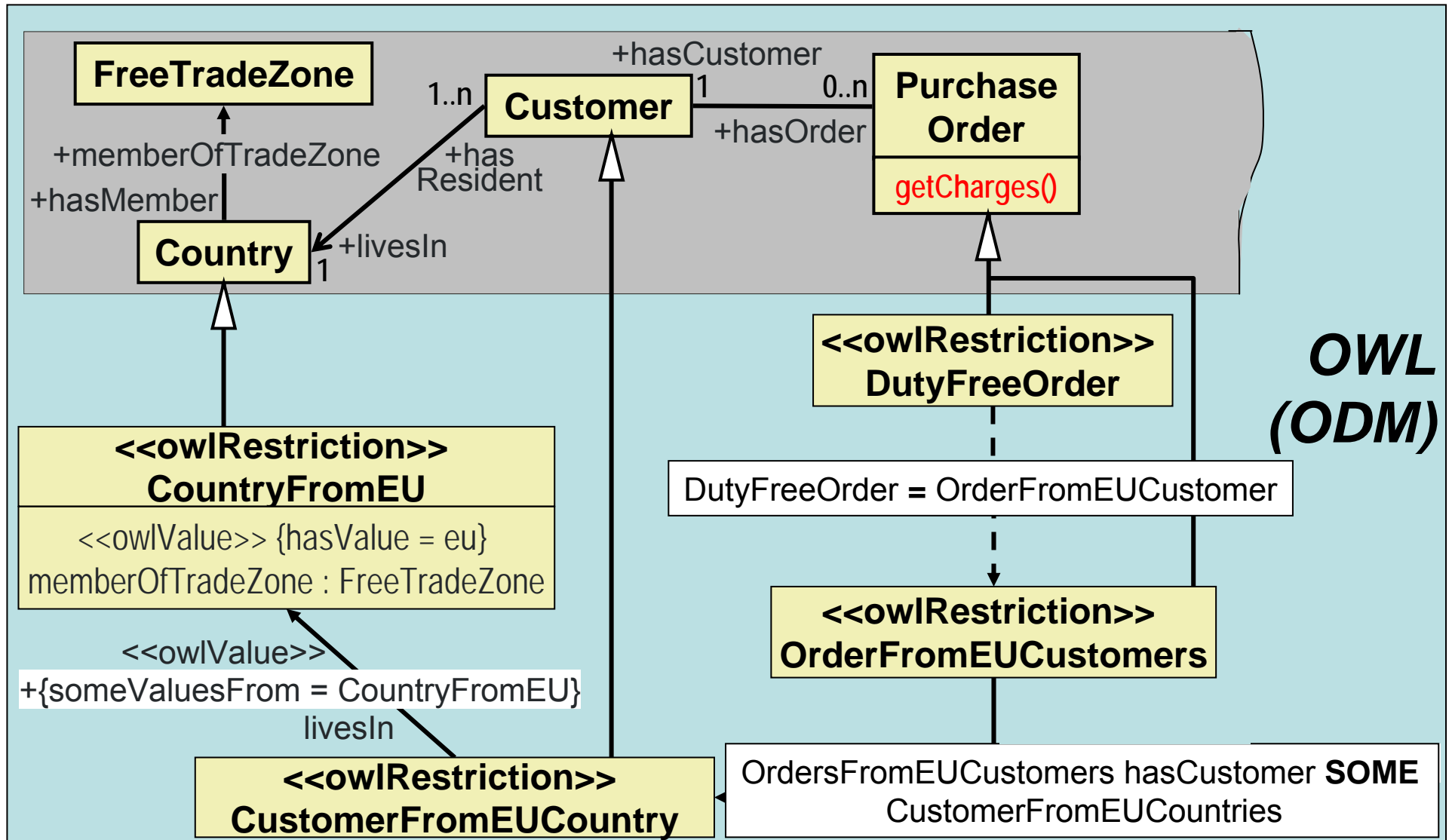


Running Example: E-Commerce Shop

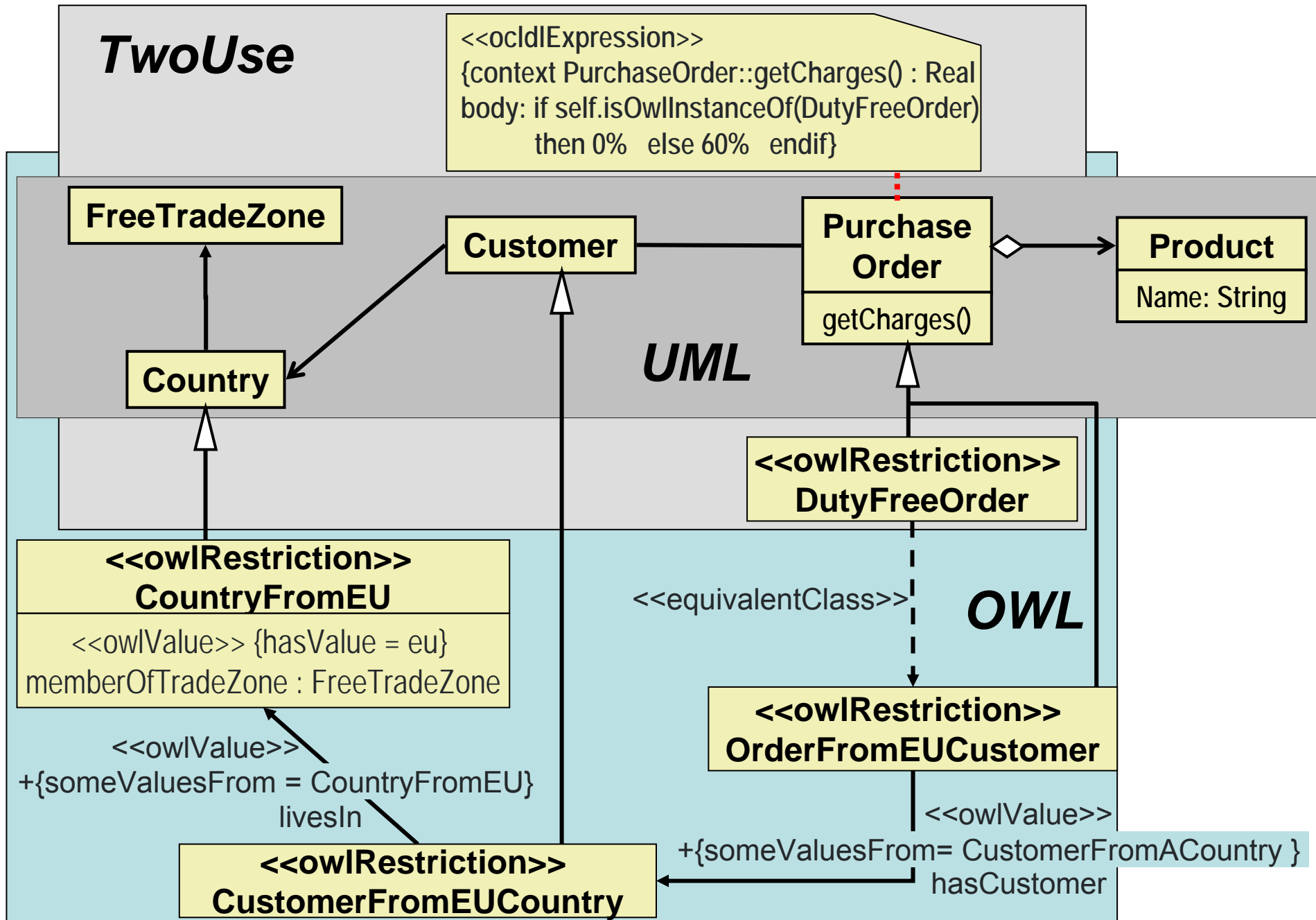




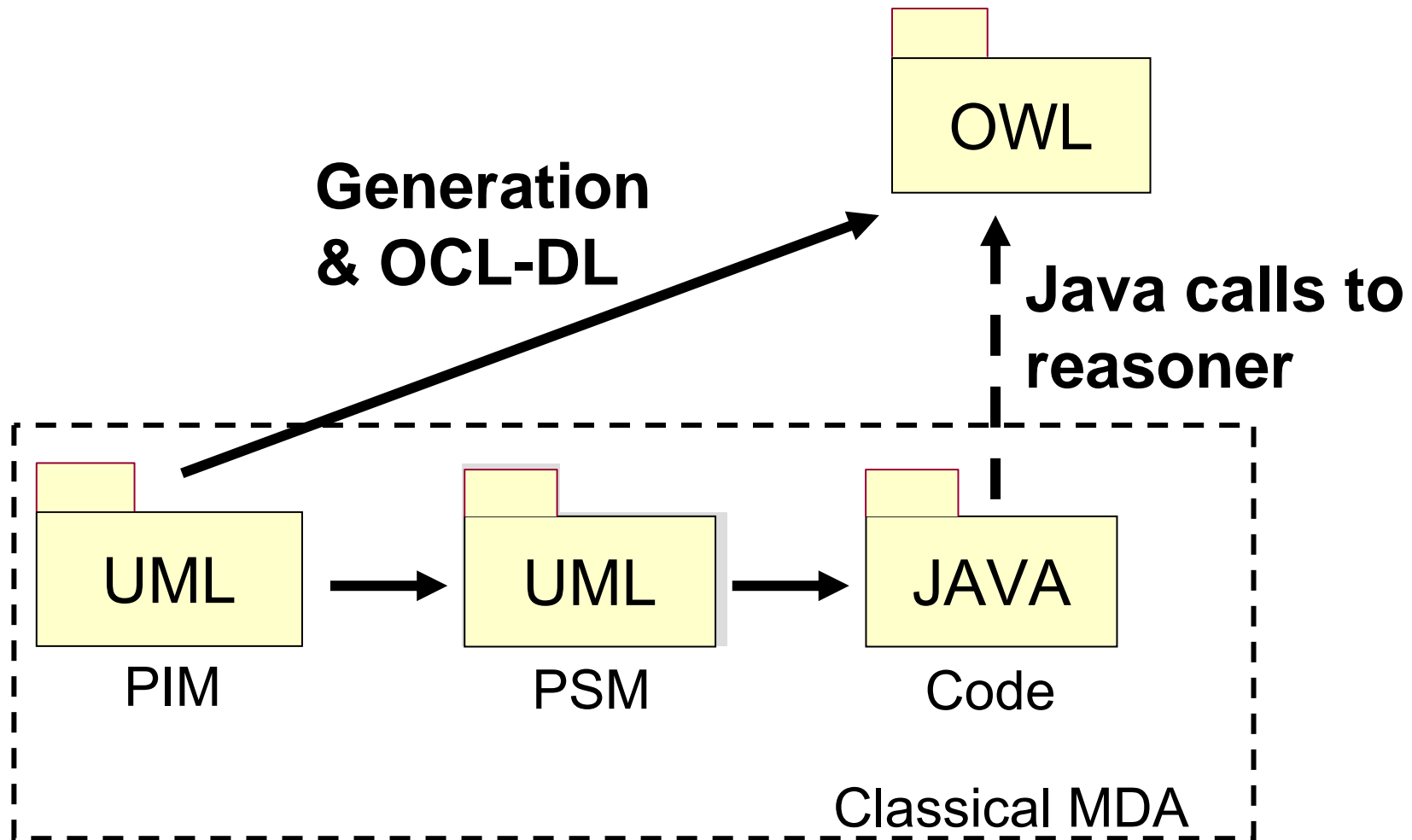
Hybrid Model: Example OWL



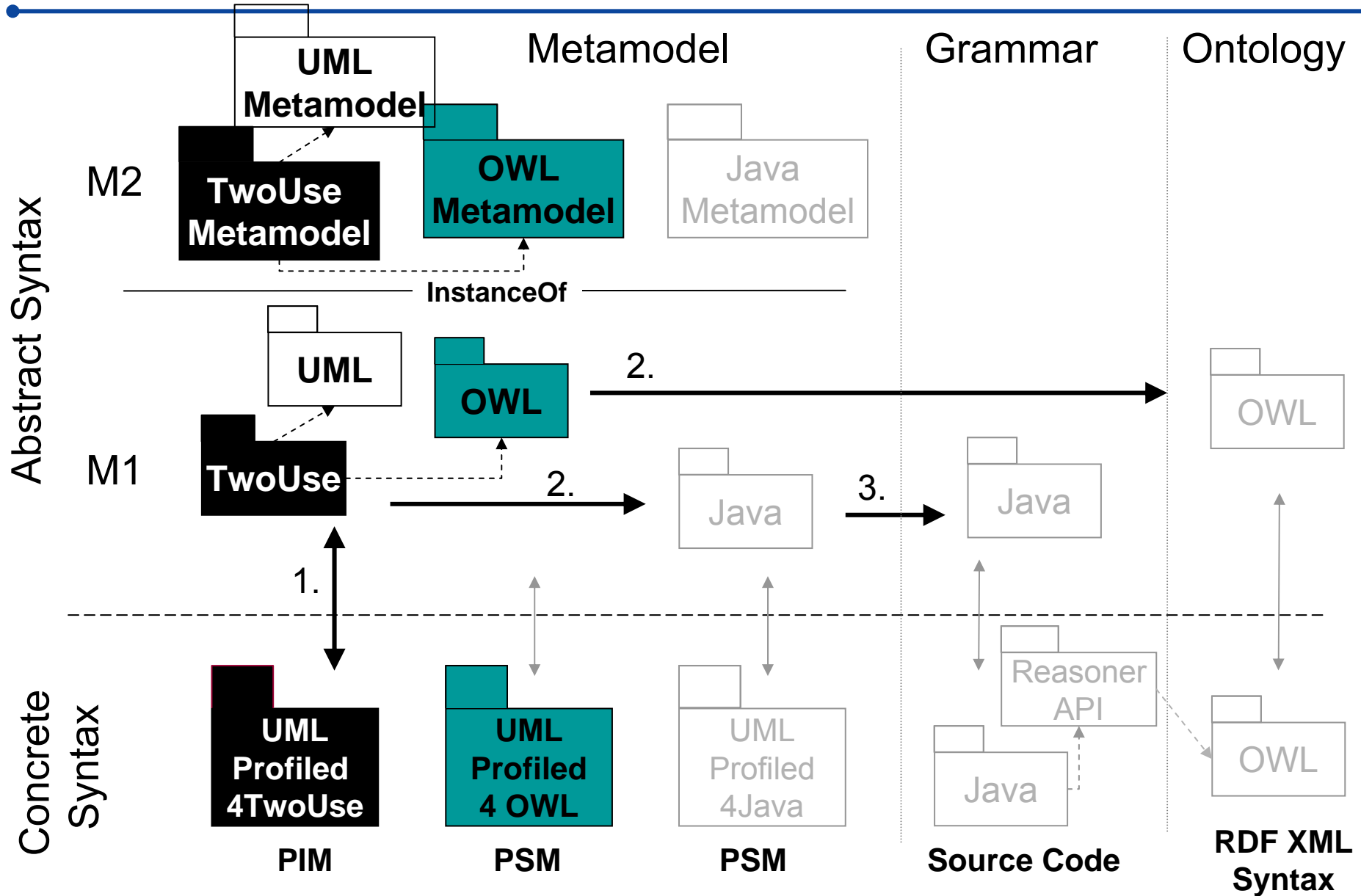
Hybrid Model: Example TwoUse

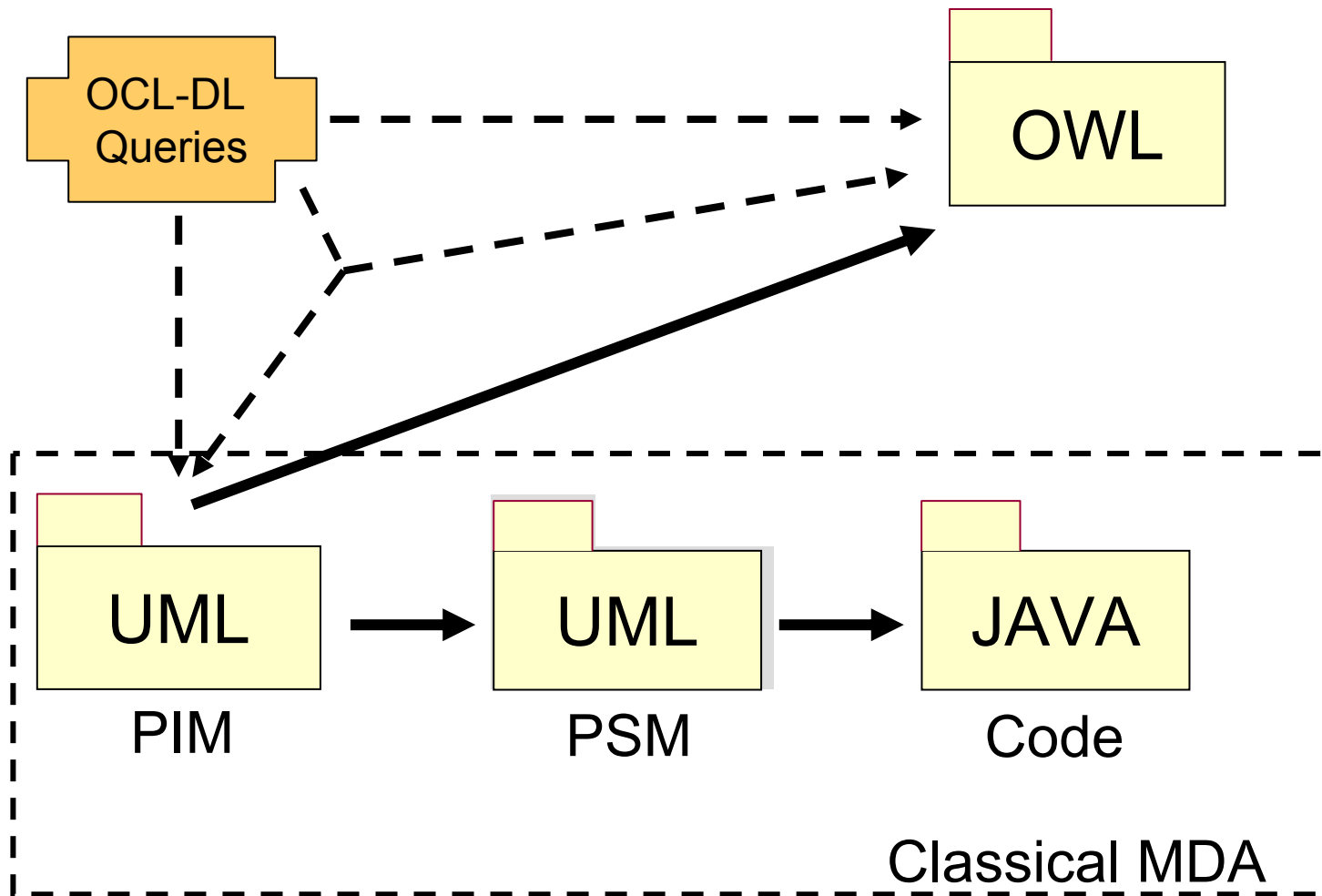


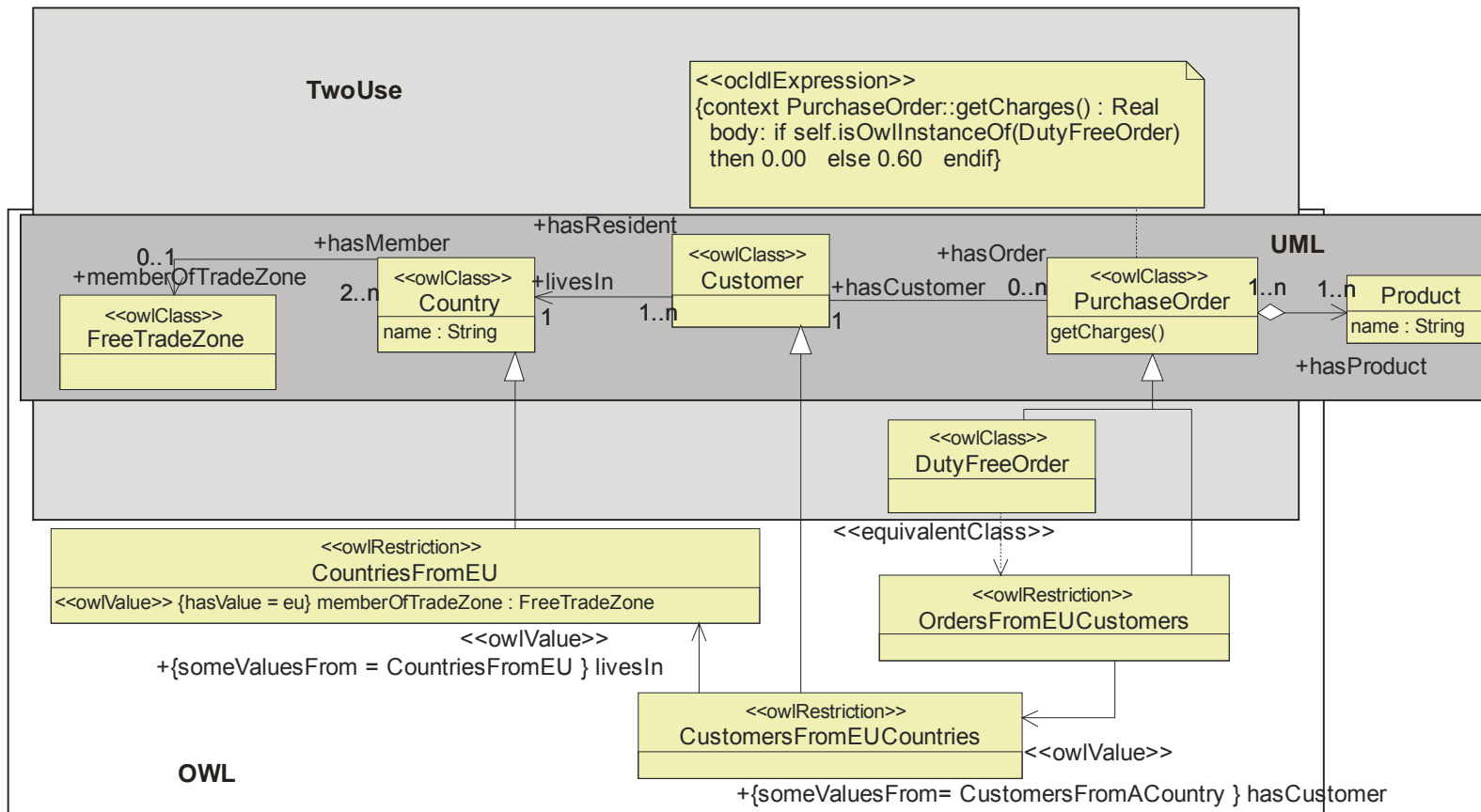
Co-generation of OWL models from UML



Transformation Process







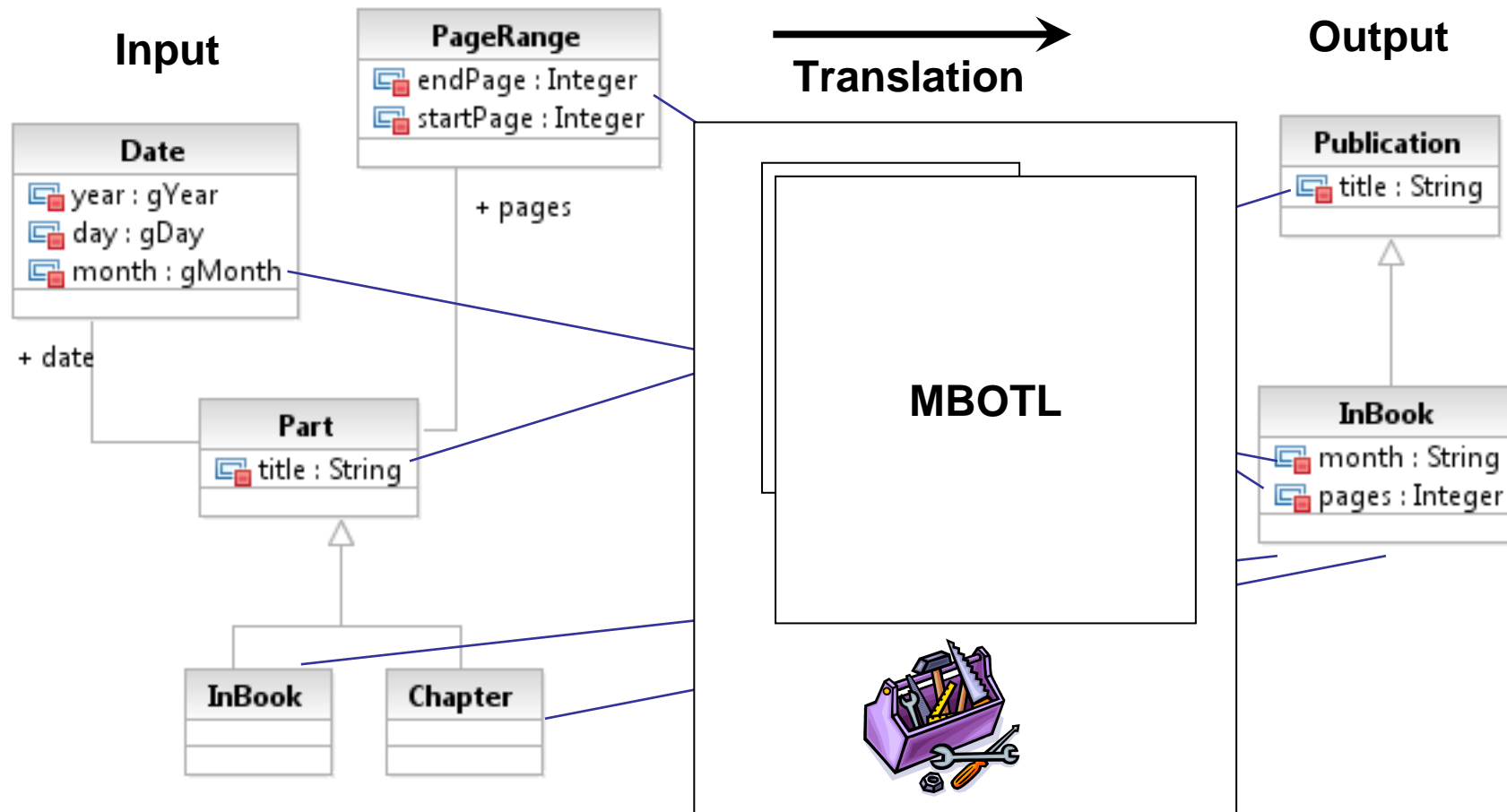
self.owlIsInstanceOf(DutyFreeOrder)
 self.owlAllTypes()
 self.owlAllInstances()

Type (?self, DutyFreeOrder)
 Type (?self, ?x)
 Type (?self,?x), Type (?y, ?x)

- ◆ “Dialect” of OCL
- ◆ Specification of
 - ◆ Queries
 - ◆ Invariants
 - ◆ Guards
- ◆ OCL Library with new operations
 - ◆ `owlIsInstanceOf(typespec: OclType)`
 - ◆ `owlAllTypes()` `owlAllInstances()`
- ◆ OWL Support
 - ◆ OWL-DL
 - ◆ OWL2, DL Rules

- ◆ Model-driven Engineering
- ◆ Ontology
- ◆ A Use Case of Ontology Technology in MDE
- ◆ Joint Metamodels
- ◆ Case 1:
Use Ontology Technology in a Design Pattern
- ◆ **Case 2:**
Using MDE for Translating between Ontologies

Translation of ontology datasets



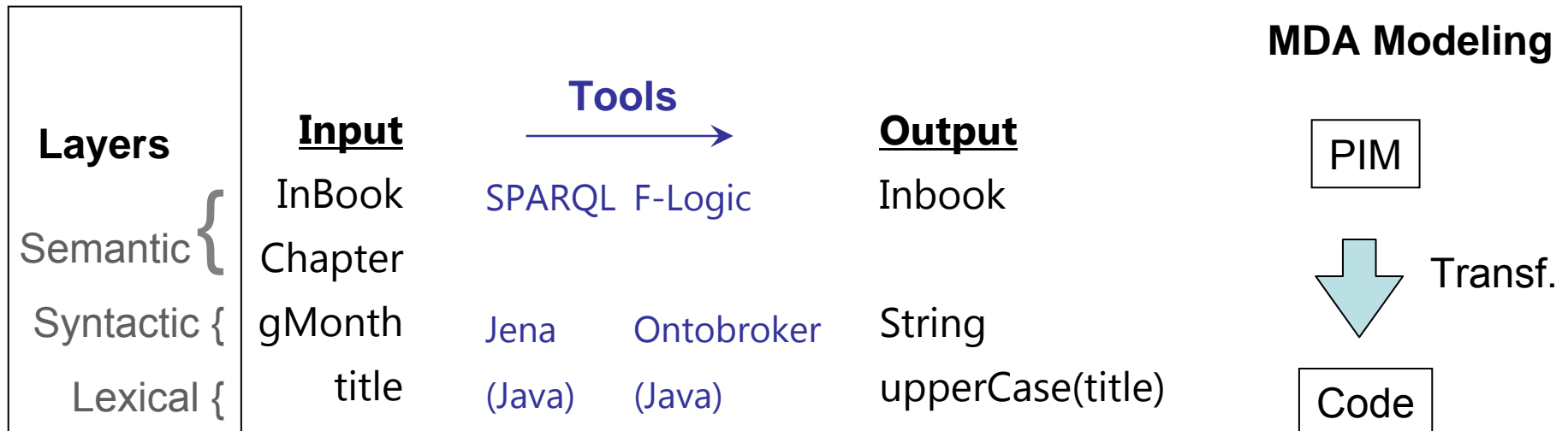
The screenshot displays the Neon Toolkit's 'Integration' workspace. It features a central 'Mapping' view where two ontologies are being compared. On the left, the 'Input' ontology lists classes like Book, Informal, Misc, MotionPicture, Part, Article, Chapter, and InBook. On the right, the 'Output' ontology lists classes like Person, Product, Project, Publication, Article, Book, Booklet, and InBook. A central diagram shows a mapping from 'InBook' in the input to 'InBook' in the output. Below this, the 'Attributes/Relations' section shows 'InBook' with properties like 'book', 'Properties from Part', and 'Properties from Reference' in the input, and 'InBook' with 'Properties from Publication' in the output. At the bottom, the 'Instances' section is empty, and the 'General Mapping Info' section provides details about the 'InBook -> InBook' mapping, including a filter configuration.

Input

Output

visual mapping of ontologies

Plug-ins must be written separately

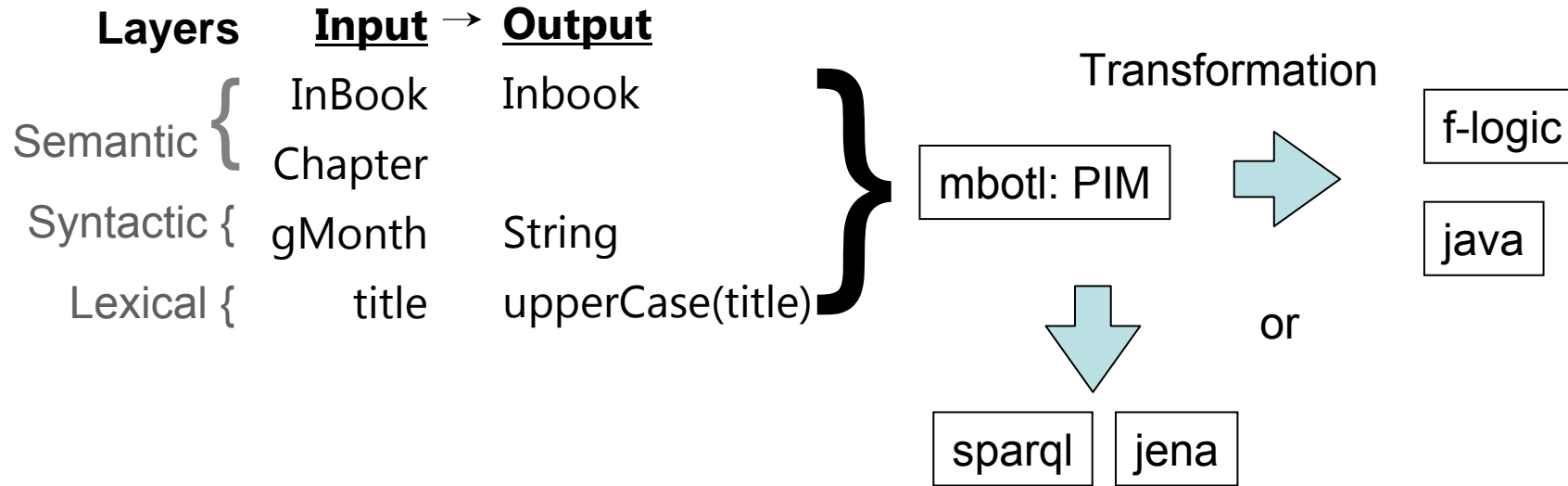


A **PIM** focus on the operation of a system and **hides** the **details** necessary for a particular **platform** (MDA Guide)

All three layers are **platform independent**

Problem:
 Why to use two languages?
 Instead of general purpose programming language, specific language?

MDA Modeling



Advantages:

- Productivity:** focus on business and not on platform details
- Portability:** Same PIM can be automatically transformed into multiple PSMs for different platforms
- Maintenance:** higher level of abstraction than code

(MDA)

Usage of OCL-like expressions to formulate queries.

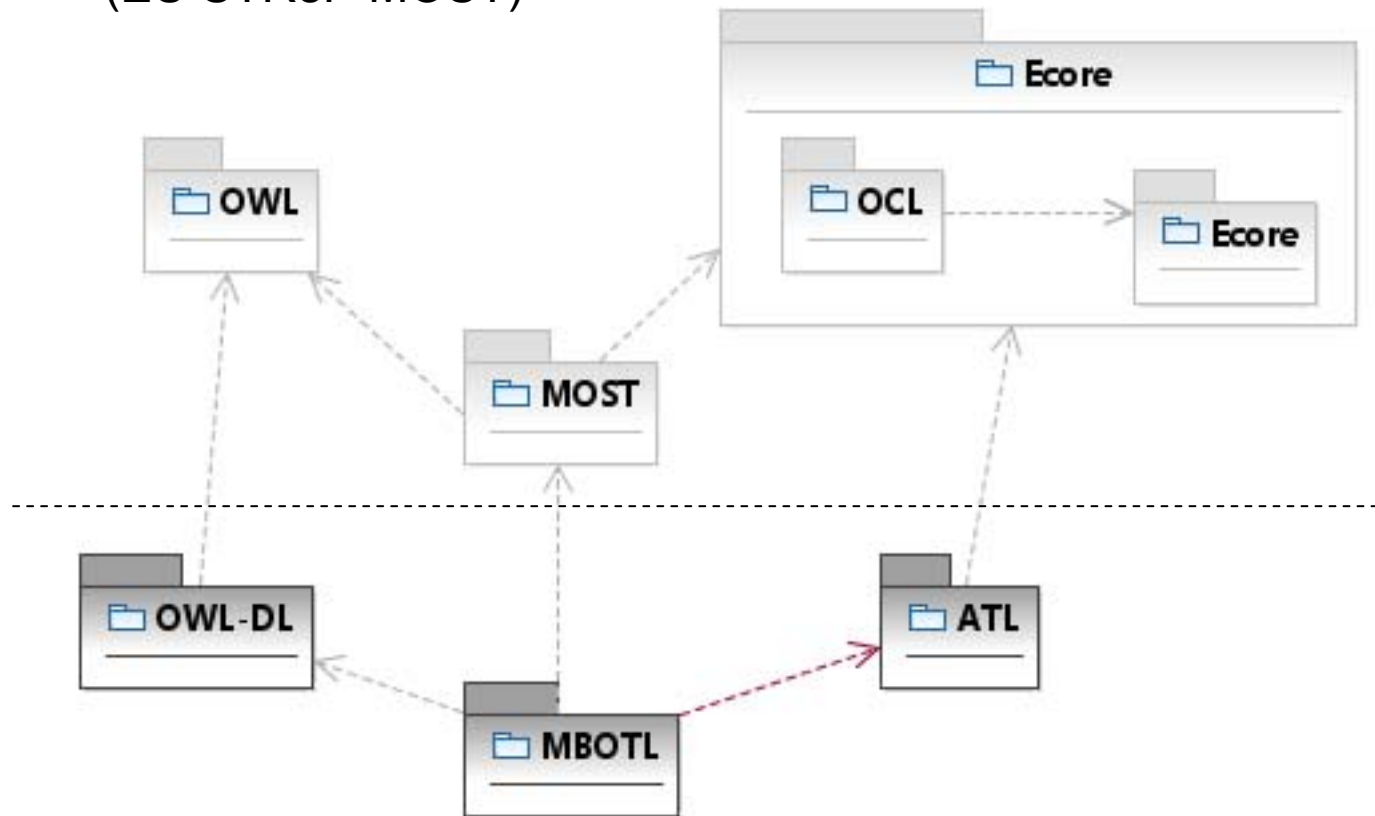
```
rule ChapterInBook2Inbook {
  from
    s : _endnote!Part (s.owlIsInstanceOf(Chapter) or
                      s.owlIsInstanceOf(Inbook))
10  to
    t : _bibtex!Inbook (
      title <- s.title.toUpper(),
      pages <- s.pages.endPage - s.pages.startPage,
      month <- s.date.month.notShortened(),
15  )
}
```

Application of predefined operations or helpers

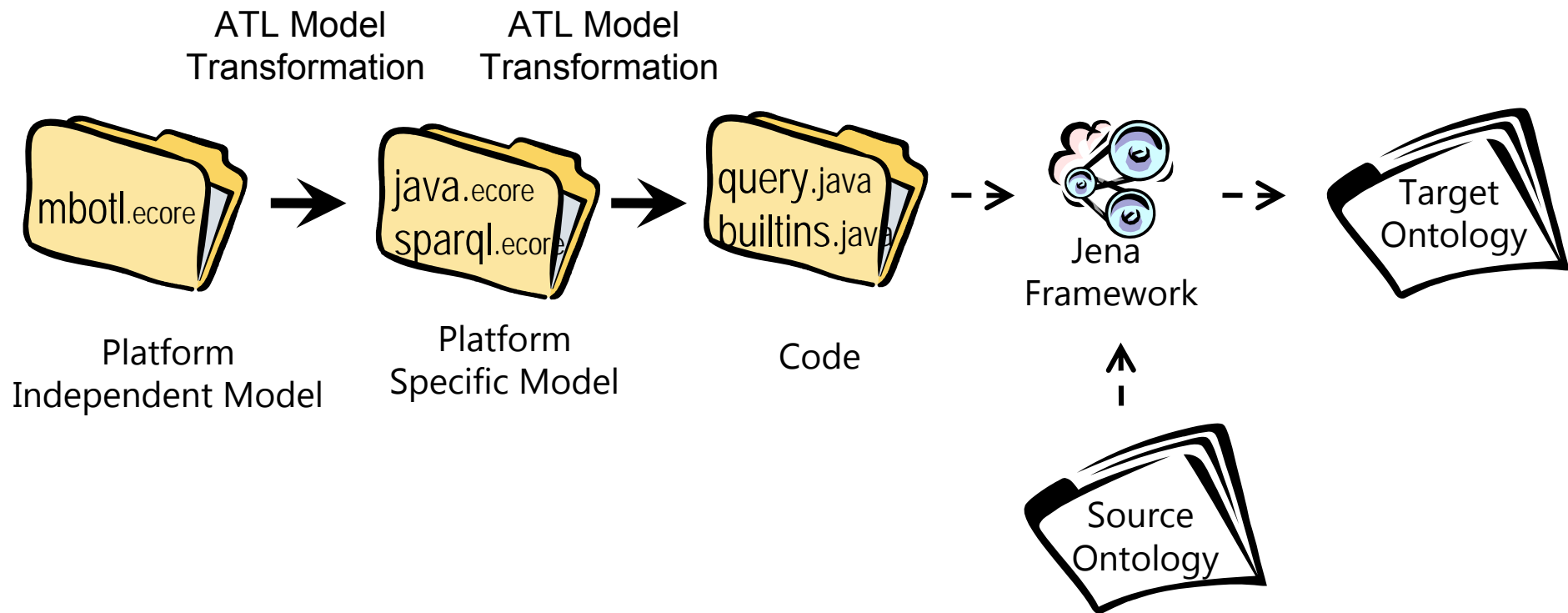
```
1 helper context _endnote!gMonth
  def: notShortened() : String =
    Sequence{'January', 'February', 'March'}->at(
      Sequence{'--01', '--02', '--03'}->indexOf(self.toString()))
5
rule ChapterInBook2Inbook {
  from
    s : _endnote!Part (s.owlIsInstanceOf(Chapter) or
                      s.owlIsInstanceOf(Inbook))
10  to
    t : _bibtex!Inbook (
      title <- s.title.toUpperCase(),
      pages <- s.pages.endPage - s.pages.startPage,
      month <- s.date.month.notShortened(),
15  )
}
```

Unified Representation

Reference Layer
(EU STReP MOST)



MBOTL Extension

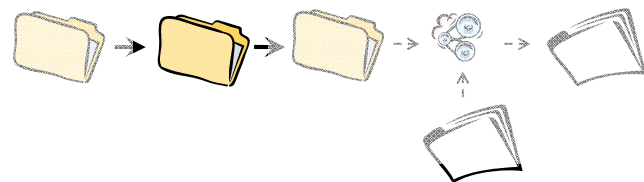


```
module OntoA2OntoB;
create OUT : OntoB from IN : OntoA;

helper context _101!gMonth
def: notShortened() : String =
  Sequence{'January', 'February', 'March'}->at(
    Sequence{'--01', '--02', '--03'}->indexOf(self.toString()));

rule ChapterInBook2Inbook {
  from
    s : _101!Part (s.owlIsInstanceOf(Chapter) or
                  s.owlIsInstanceOf(Inbook))
  to
    t : _303!Inbook (
      title <- s.title.toUpper(),
      pages <- s.pages.endPage - s.pages.startPage,
      month <- s.date.month.notShortened()
    )
}
```

Screenshot: Generated SPARQL Model



Helpers.ocl.xmi.ocl Transformation2.mbot queries.sparql.xmi.s simpleSPARQL.km3 simpleSPARQL.tcs Transformal

```
PREFIX ont: <"http://www.testontology.com/">
PREFIX ont2: <"http://www.testontology2.com/">
PREFIX userdef: <"java:propertyfunction.">
```

simpleSPARQL

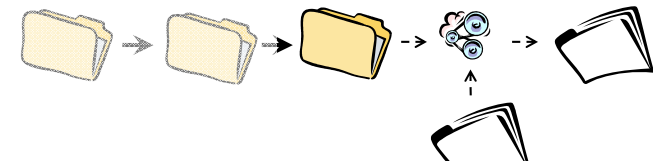
- Metamodel
 - simpleSPARQL.ann
 - simpleSPARQL.ecore
 - simpleSPARQL.km3
- plugin
- Samples
 - queries.sparql.xmi
 - queries.sparql.xmi.sparql
 - Transformation2.sparql
- Syntax
 - simpleSPARQL_ANTLR3.g
 - simpleSPARQL.tcs
 - simpleSPARQL-parser.jar
- TGF
 - simpleSPARQL.ecore
 - simpleSPARQL.km3
- plugin
- Samples
 - queries.sparql.xmi

```
kSubject ?ChapterInBook2 InBookPredicate ont2:Inbook.
kSubject ont2:title ?ChapterInBook2 InBook_title.
kSubject ont2:month ?ChapterInBook2 InBook_notShortened.
kSubject ont2:pages ?ChapterInBook2 InBook_endPage

kSubject ?ChapterInBook2 InBookPredicate ont:Part.
kSubject ont:date ?ChapterInBook2 InBook_date.
k_date ont:month ?ChapterInBook2 InBook_month.
k_hasSize userdef:notShortened ?ChapterInBook2 InBook_month.
kSubject ont:pages ?ChapterInBook2 InBook_pages.
k_pages ont:endPage ?ChapterInBook2 InBook_endPage
```

Screenshot: Java Code with Jena API

<isweb>



```
Java - Main.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Octopus Run Window Help

Package Explorer Hierarchy Main.java Sizemapping.java Helper.ocl minimodelquery.sparql MBOTLHelpers.uml Helper.java NotShortene

public static void main( String[] args ) {

    // String saving the SPARQL-query
    String queryString = "";

    // File variables
    String sourceModel = "file:./res/minimodel.owl";
    String targetModel = "./res/testtarget.owl";
    String sparqlFile = "./res/minimodelquery.sparql";
    //String sparqlFile = "./res/minimodelquery.sparql";

    // Model variables
    OntModel mA = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
    OntModel mB = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);

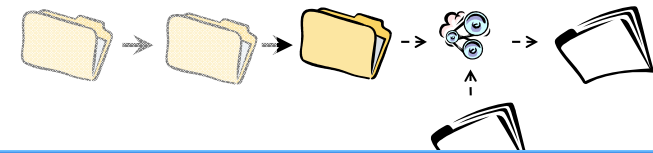
    // Read the source document
    mA.read(sourceModel);

    try
    {
        BufferedReader in = new BufferedReader(new FileReader(sparqlFile));
        String line = null;

        while ((line = in.readLine()) != null)
    }
}
```

Screenshot: Java Code of Built-In

<is web>



Java - Helper.java - Eclipse SDK

File Edit Source Refactor Navigate

Package Explorer Hierarchy

MBOTLHelpers

- src
 - engine
 - Main.java
 - MbotQueryEngine.java
 - MBOTLHelpers
 - Helper.java
 - propertyfunction
 - NotShortened.java
 - Sizemapping.java
 - UpperCase.java
 - utilities
- JRE System Library [jre1.6.0]
- antlr-2.7.5.jar
- arq.jar
- arq-extra.jar
- commons-logging-1.1.1.jar
- concurrent.jar
- icu4j_3_4.jar
- iri.jar
- jena.jar
- jenatest.jar
- json.jar
- junit.jar
- log4j-1.2.12.jar
- lucene-core-2.2.0.jar
- stax-api-1.0.jar
- wstx-asl-3.0.0.jar
- xercesImpl.jar

```
/** Implements Sequence('January', 'February', 'March')
 *
 * @param s
 */
private List collectionLiteral1(String s) {
    List /*(String)*/ myList = new ArrayList( /*String*/);
    myList.add( "January" );
    myList.add( "February" );
    myList.add( "March" );
    return myList;
}

/** Implements Sequence('01', '02', '03')
 *
 * @param s
 */
private List collectionLiteral2(String s) {
    List /*(String)*/ myList = new ArrayList( /*String*/);
    myList.add( "01" );
    myList.add( "02" );
    myList.add( "03" );
    return myList;
}

List /*(String)*/ myList = new ArrayList( /*String*/);
myList.add( "01" );
myList.add( "02" );
```

MBOTL provides an **unified language** to model different layers of ontology dataset translation problems.

At semantic level: OCL as query language.

At syntactic and lexical levels: OCL predefined operations and user-defined helpers.

Improvements: productivity, portability, maintenance.

Implementation: Eclipse, ATL Transformation Language, Jena.

Future Work: plug-in, SPARQL-like syntax, evaluation.

Download and test: isweb.uni-koblenz.de/Research/MBOTL

Joint metamodels

- ◆ allow for joint querying of UML & OWL
- ◆ provide synchronized reasoning calls
- ◆ extend the power of UML with UWL
- ◆ extend the power of SPARQL (or other ontology services) by programming language competencies

Thank You!

<http://isweb.uni-koblenz.de/Projects/twouse>

<http://isweb.uni-koblenz.de/Projects/MOST>

Fernando Silva Parreiras, Steffen Staab, Andreas Winter: *Improving Design Patterns by Description Logics: A Use Case with Abstract Factory and Strategy*. in Modellierung 2008, 12.-14. März 2008, Berlin. 2008. GI. LNI. 127.

Fernando Silva Parreiras, Steffen Staab, Simon Schenk, Andreas Winter: *Model Driven Specification of Ontology Translations*. In: Conceptual Modeling - ER 2008, 27th International Conference on Conceptual Modeling, 2008. Springer.

Fernando Silva Parreiras, Steffen Staab, Andreas Winter: *On Marrying Ontological and Metamodeling Technical Spaces*. 2007. ACM Press. Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2007.

Fernando Silva Parreiras, Steffen Staab, Andreas Winter: *TwoUse: Integrating UML Models and OWL Ontologies*. Universität Koblenz-Landau, Fachbereich Informatik. 2007. Arbeitsbericht.

