

Semantic Model-driven Engineering

Steffen Staab

with

Gerd Gröner, Fernando Silva Parreiras, Tobias Walter



People and Knowledge Networks

Who we are

Prof. Dr. Staab

Prof. Dr. Sure



Semantic Web
Dr. Janik

Web Retrieval
Dr. Sizov

Interactive Web
Dr. Scherp

Multimedia Web
Dr. Grzegorzec

Software Web
F. Silva Parreiras

GESIS
Prof. Sure

EU NeOn
BMBF CollabCloud
EU Net2

DFG Multipla
HP Synth Docs
EU Tagora

EU WeKnowIt
EU WeGov

EU X-Media
EU kspace
EU aceMedia

EU Most
EU ASG

EU WeGov



MDA / MDE

Ingredients

- **Models:**
representing complementary views of a system
 - ◆ Structural
 - ◆ Behavioral
- **Metamodeling:**
Linguistic instantiation of syntactic class descriptions
- **Transformations:**
Towards target platform
Adding refinements

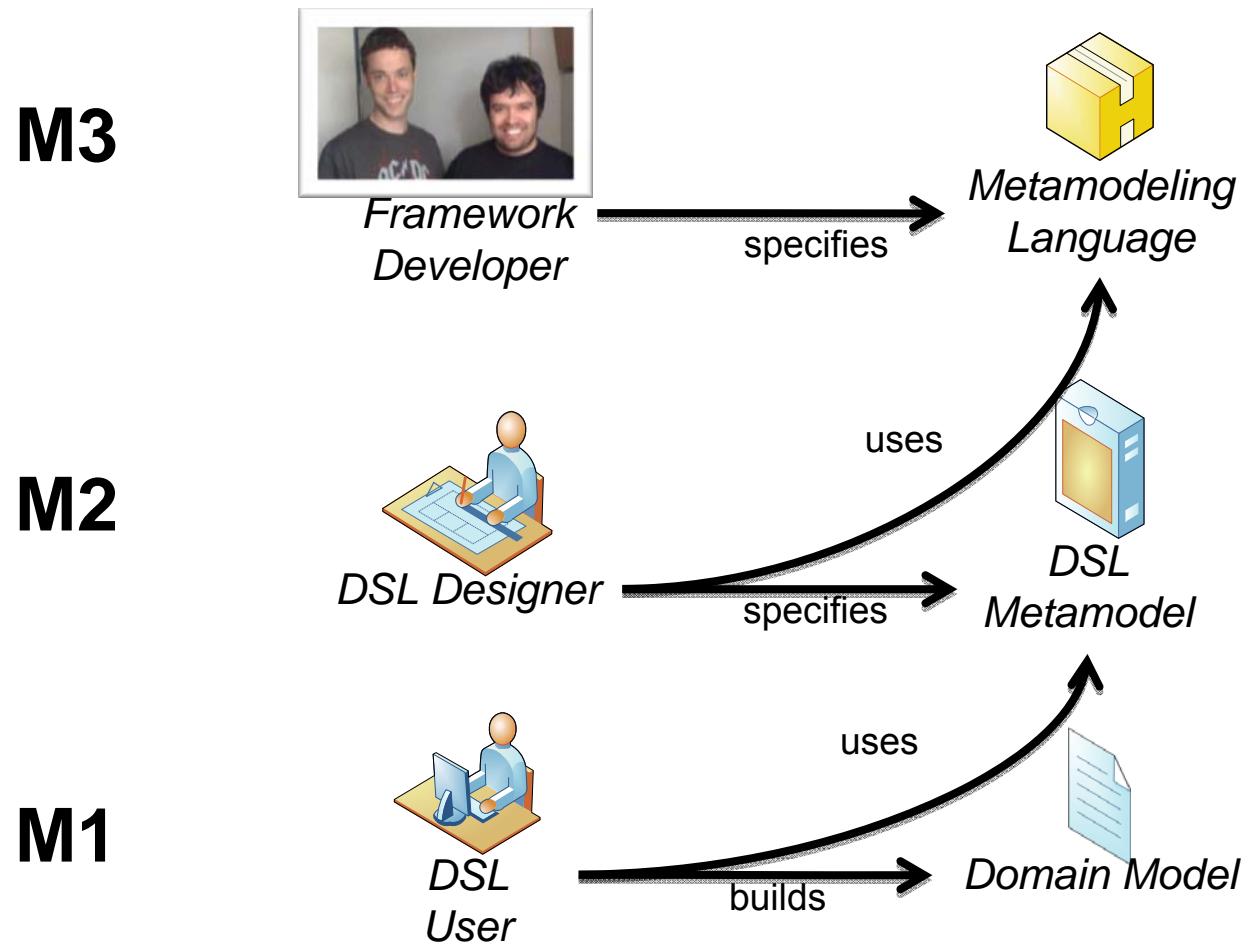
Description Logics / Ontologies

Ingredients

- **Ontologies:**
representing complementary views of a system
 - ◆ Structural
 - ◆ (Behavioral)
- **Metamodeling:**
Ontological instantiation of logical class descriptions
- **Transformations:**
Between knowledge bases/ontologies

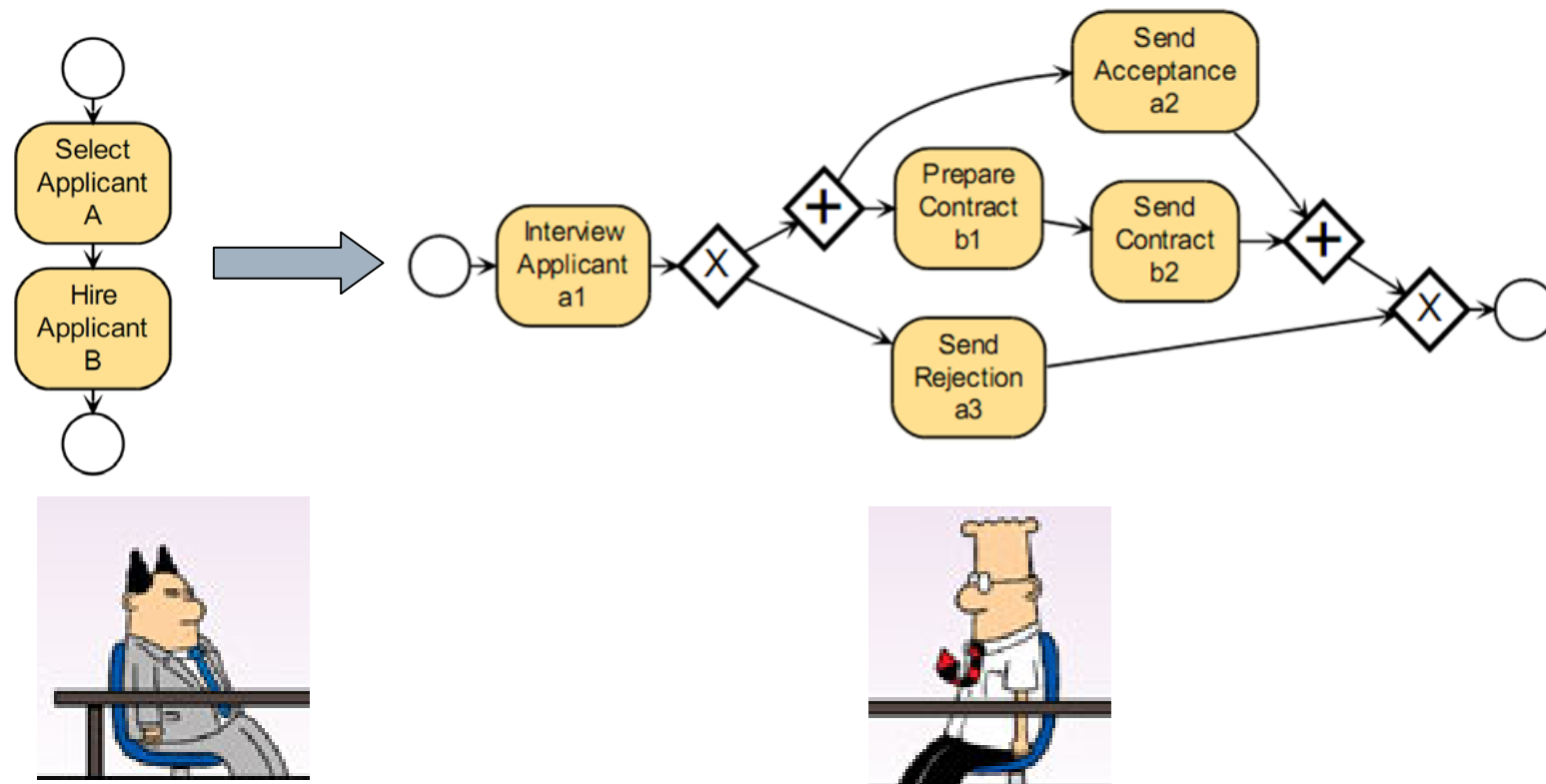
Refinements in several dimensions

- **Refinements along metamodeling levels**



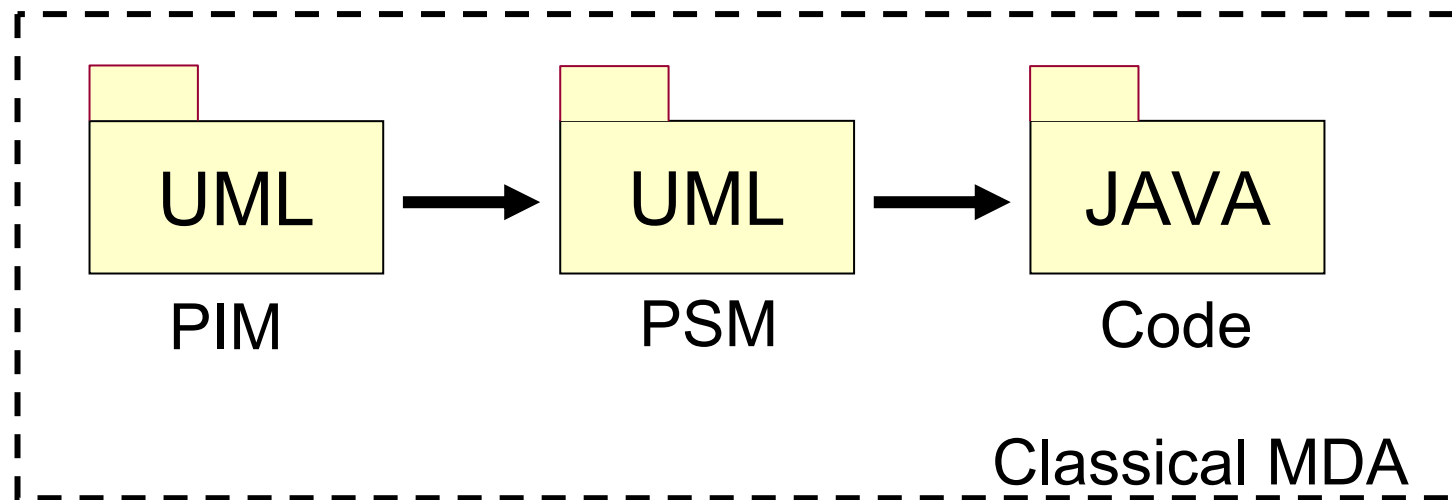
Refinements in several dimensions

- Refinements along metamodeling levels
- **Refinements along model specification**
 - ◆ **From business developer to software developer**



Refinements in several dimensions

- Refinements along model specification
 - ◆ From business developer to software developer
- Refinements along metamodeling levels
- **Refinements along platform specification**
 - ◆ **For Ontology Translations**



Refinements in several dimensions

- Refinements along metamodeling levels
- Refinements along model specification
 - ◆ From business developer to software developer
- Refinements along platform specification
 - ◆ For Ontology Translations
- **Refinements along time**
 - ◆ **Metamodel evolution**
 - **Suggesting changes to transformations by DL reasoning**
 - ◆ **Ontology API Co-evolution**

- **Description Logics Reasoning by Example**
- Model-driven engineering with OWL
- Refinements in several dimensions
 - ◆ Refinements along metamodeling levels [Models 2009, ECMFA 2010]
 - ◆ Refinements along model specification [DL 2009, EKAW 2010]
 - From business developer to software developer
 - ◆ Refinements along platform specification [ER 2008]
 - For Ontology Translations
 - ◆ Refinements along time
 - Metamodel evolution [ISWC2010]
 - Suggesting changes to transformations by DL reasoning
 - Ontology API Generation/Co-evolution [ICSC 2009]

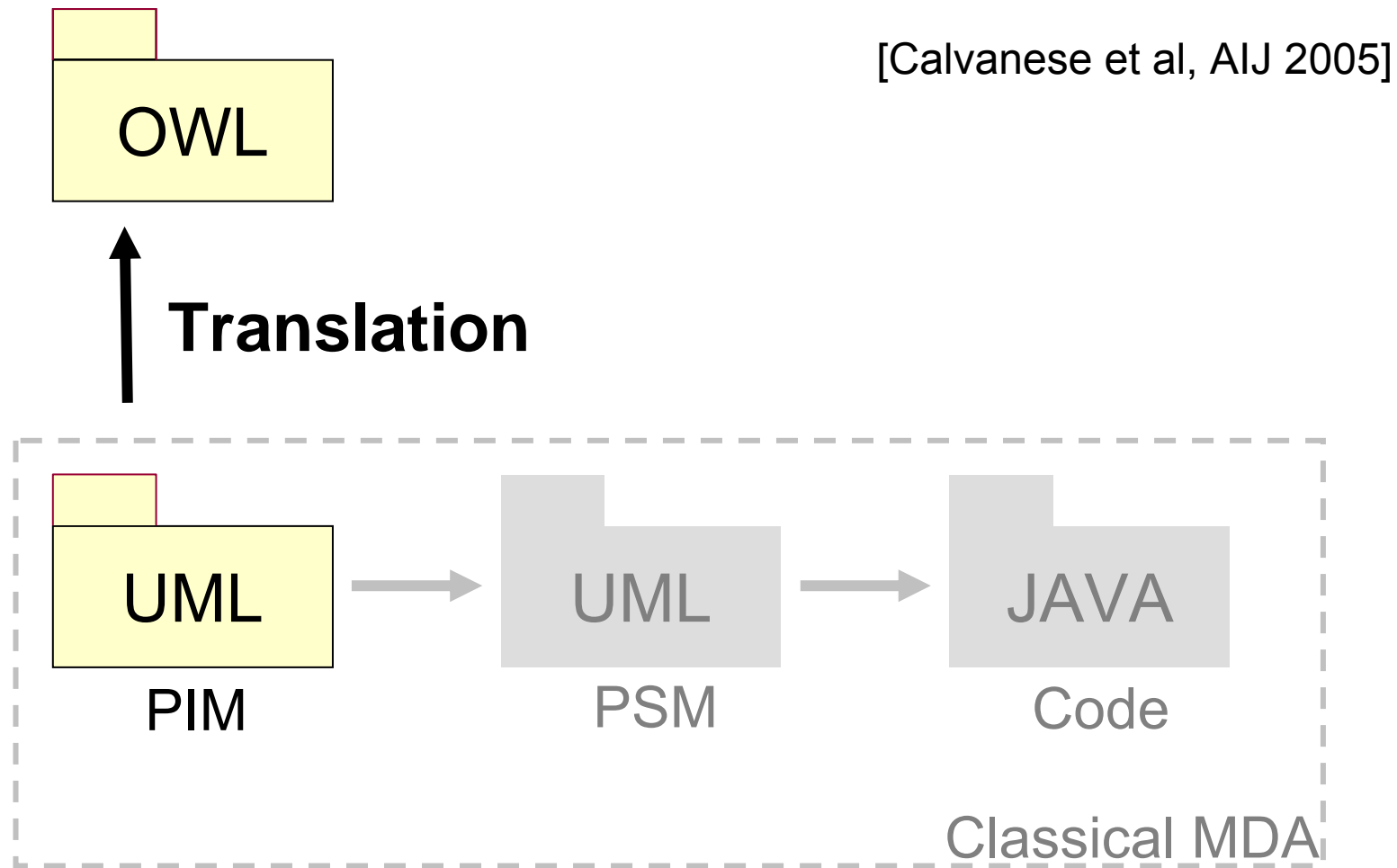
- Description Logics (DLs) are logics designed to represent and reason on structured knowledge
- The domain of interest is structured into (TBox):
 - ◆ concepts, which correspond to classes, and denote sets of individuals
 - ◆ roles, which correspond to associations, and denote binary relations on individuals
- The knowledge is asserted through so-called assertions (ABox)
- They provide formal semantics
- DLs provide the foundations for standard ontology languages, like OWL2
- **BUT: Why should a software engineer care?**

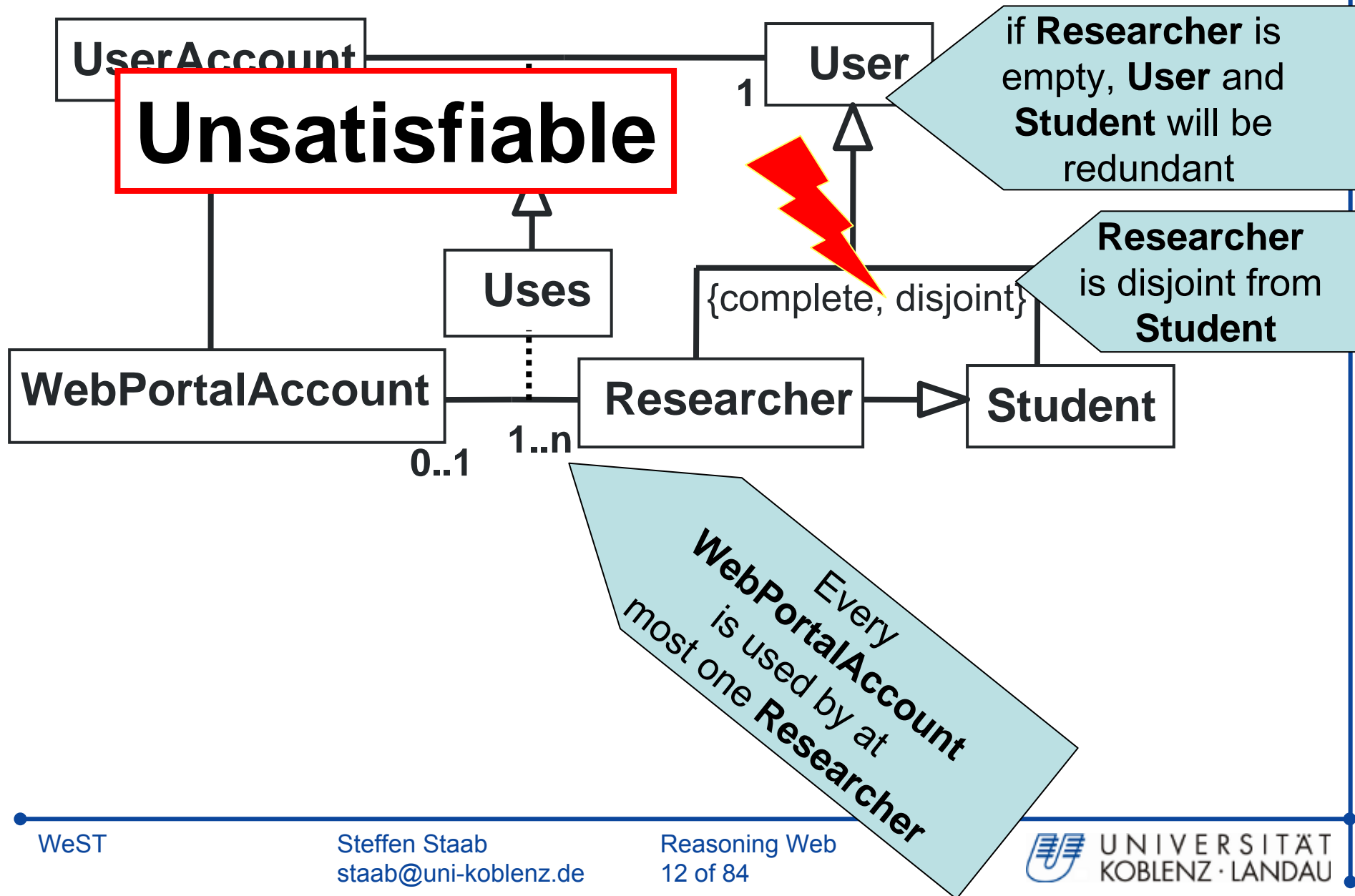
- General Problems from Knowledge Representation (KR) perspective
 - ◆ Representational View
 - Lack of formal semantics in modeling languages
 - metamodel to check syntax but not semantics of a model
 - If there is a formal semantics, modeling languages are often too expressive to have sound and complete, full-fledged, general purpose reasoner

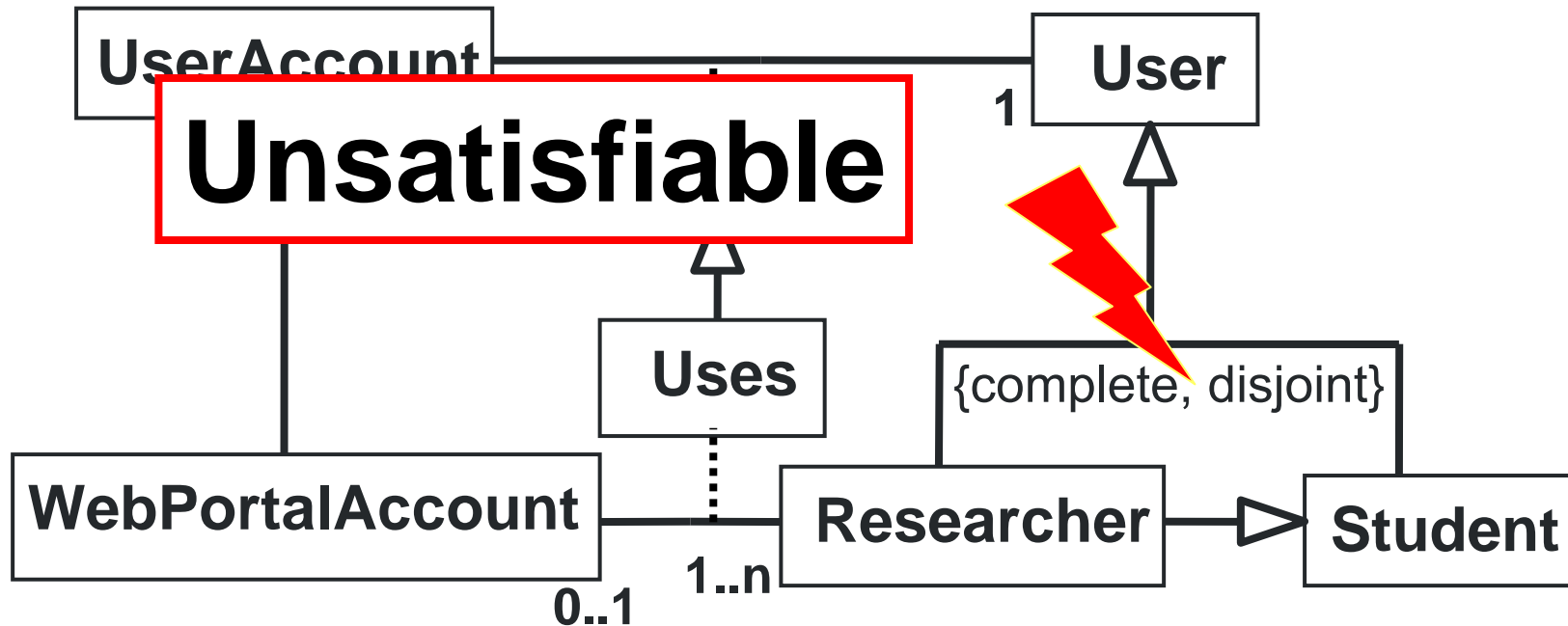
- Operational View
 - ◆ Reasoning Tasks are not well automated

- Transformations from Models to Description Logics as an asset, not as a silver bullet

- Reasoning on UML class diagrams







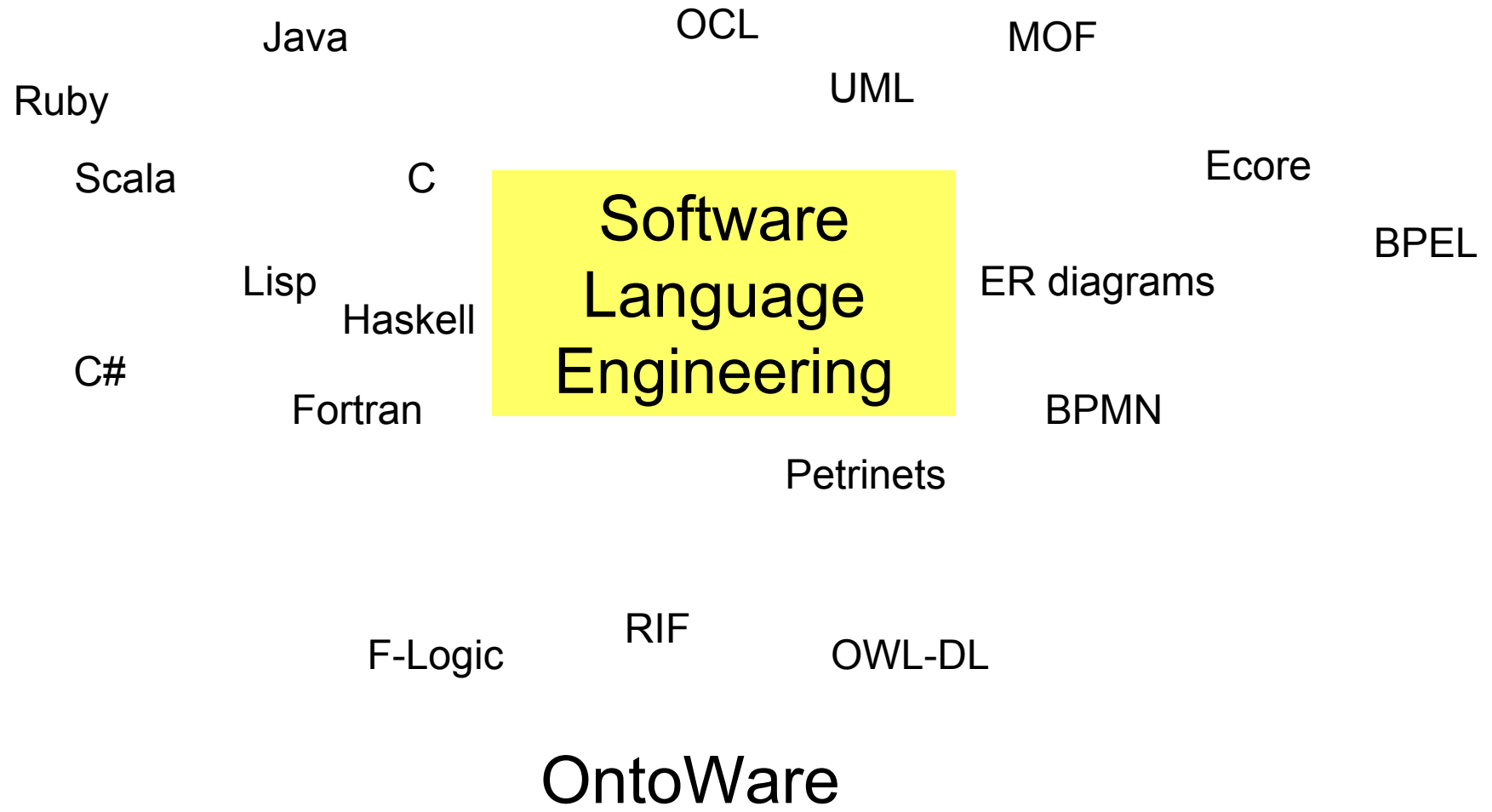
**Advantage for software engineering:
Models with provably higher quality**

- Using ontologies to formalize models
- Modeling relationships between artifacts
- Inferring relationships between artifacts, e.g. traceability relationships
- Ensuring semantic correctness by constraint validation
- Guiding the development by reasoning (reasoning in process/model development)

- Description Logics Reasoning by Example
- **Model-driven engineering (with OWL)**
- Refinements in several dimensions
 - ◆ Refinements along metamodeling levels [Models 2009, ECMFA 2010]
 - ◆ Refinements along model specification [DL 2009, EKAW 2010]
 - From business developer to software developer
 - ◆ **Refinements along platform specification** [ER 2008]
 - For Ontology Translations
 - ◆ Refinements along time
 - Metamodel evolution [ISWC2010]
 - Suggesting changes to transformations by DL reasoning
 - Ontology API Generation/Co-evolution [ICSC 2009]

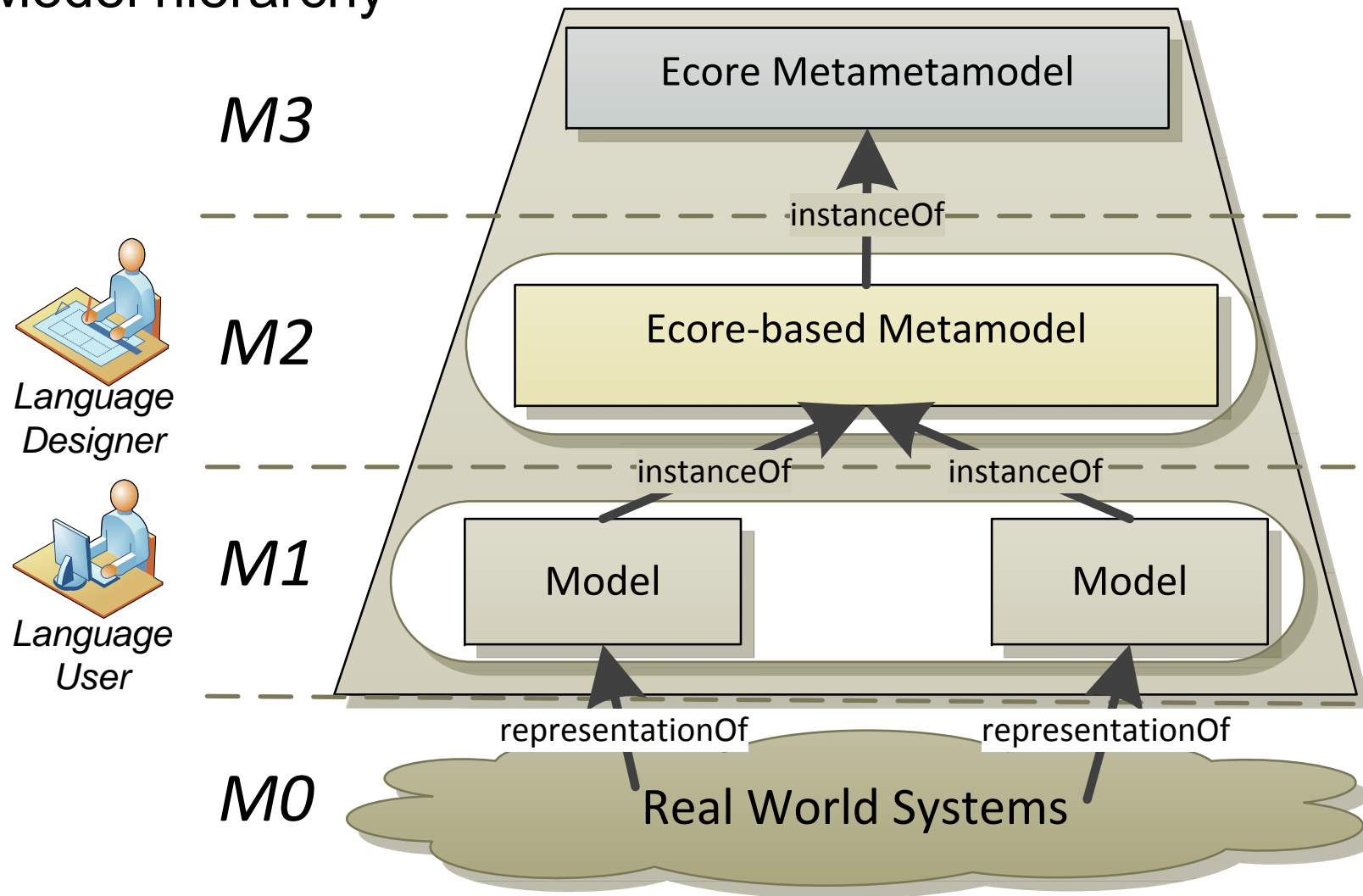
Grammarware

Modelware



- Context
 - ◆ Software Languages
 - ◆ Ontology Languages
- Language Bridges
 - ◆ Transforming Software Languages to OWL
 - ◆ Integrating Software Languages with OWL
- Services
 - ◆ Reasoning Services
 - ◆ Querying Services

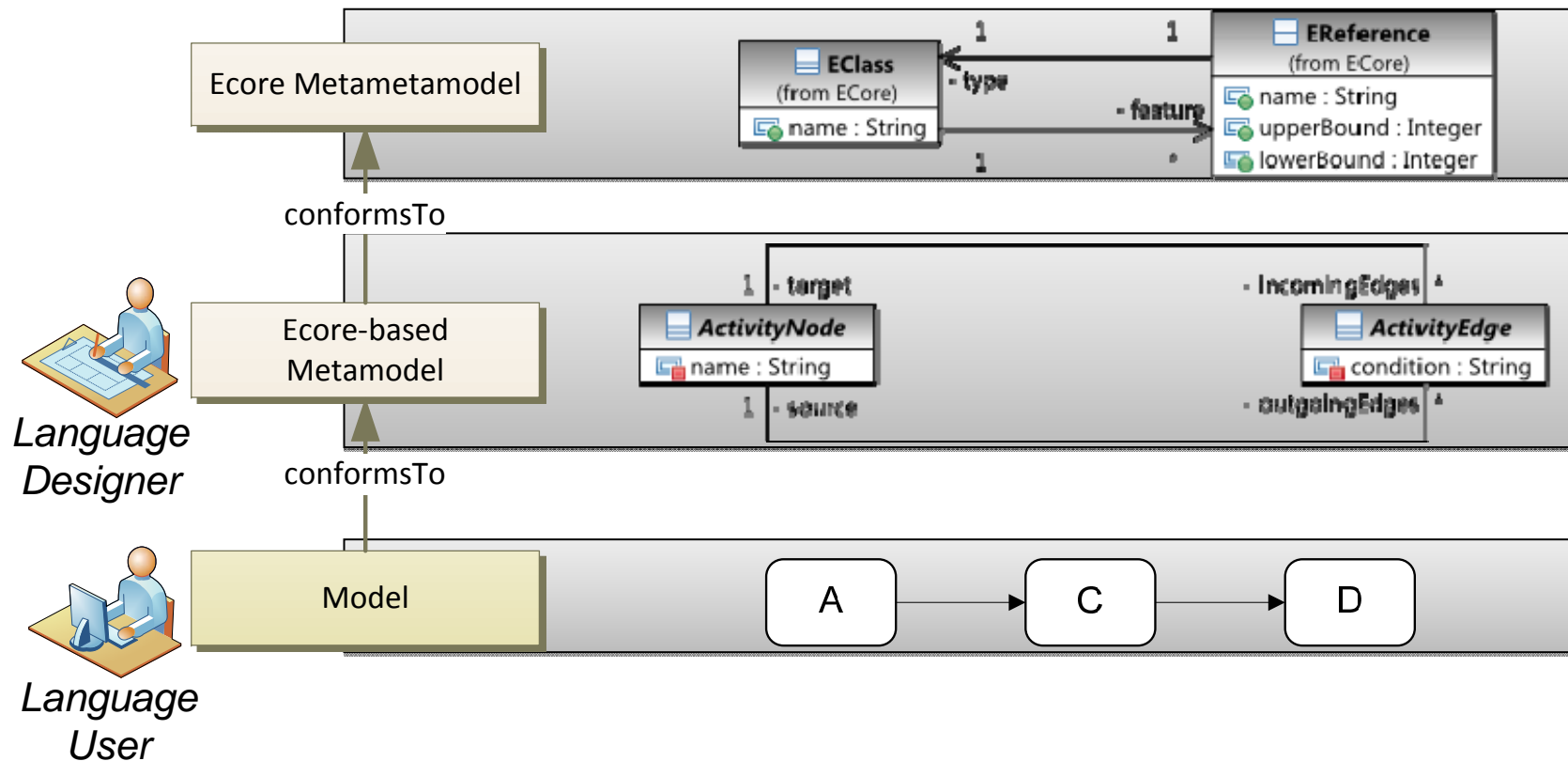
- Model hierarchy



- MOF simplifies UML Class Diagrams
 - ◆ No n-ary associations
 - ◆ No association classes
 - ◆ Used for metamodeling

- Ecore: two aspects
 - ◆ implements MOF as part of Eclipse
 - ◆ Specification language like MOF (minus some bugs)

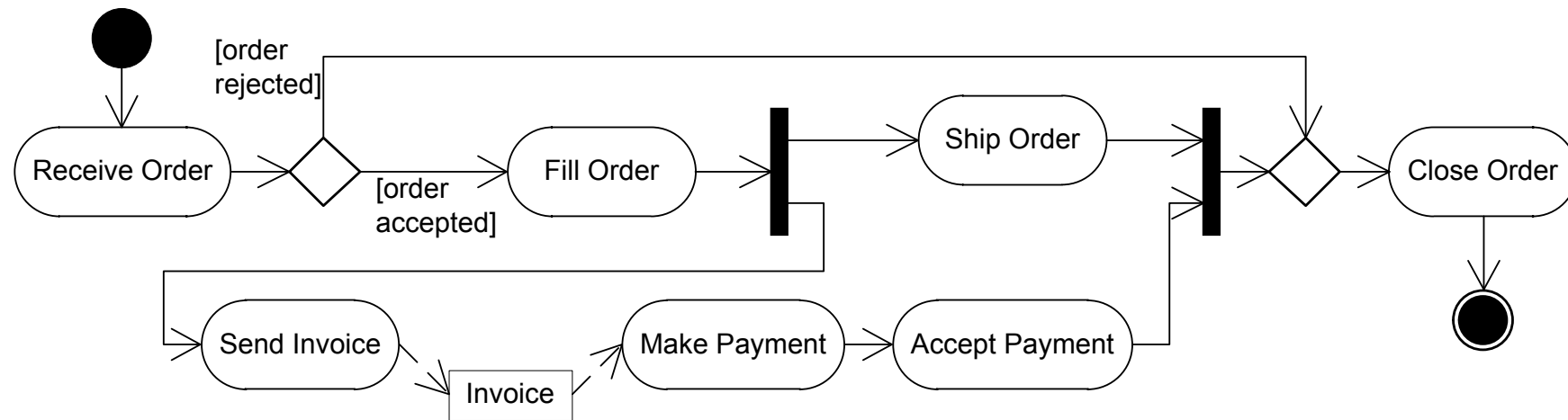
Example: Process modeling

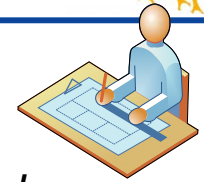




Language User

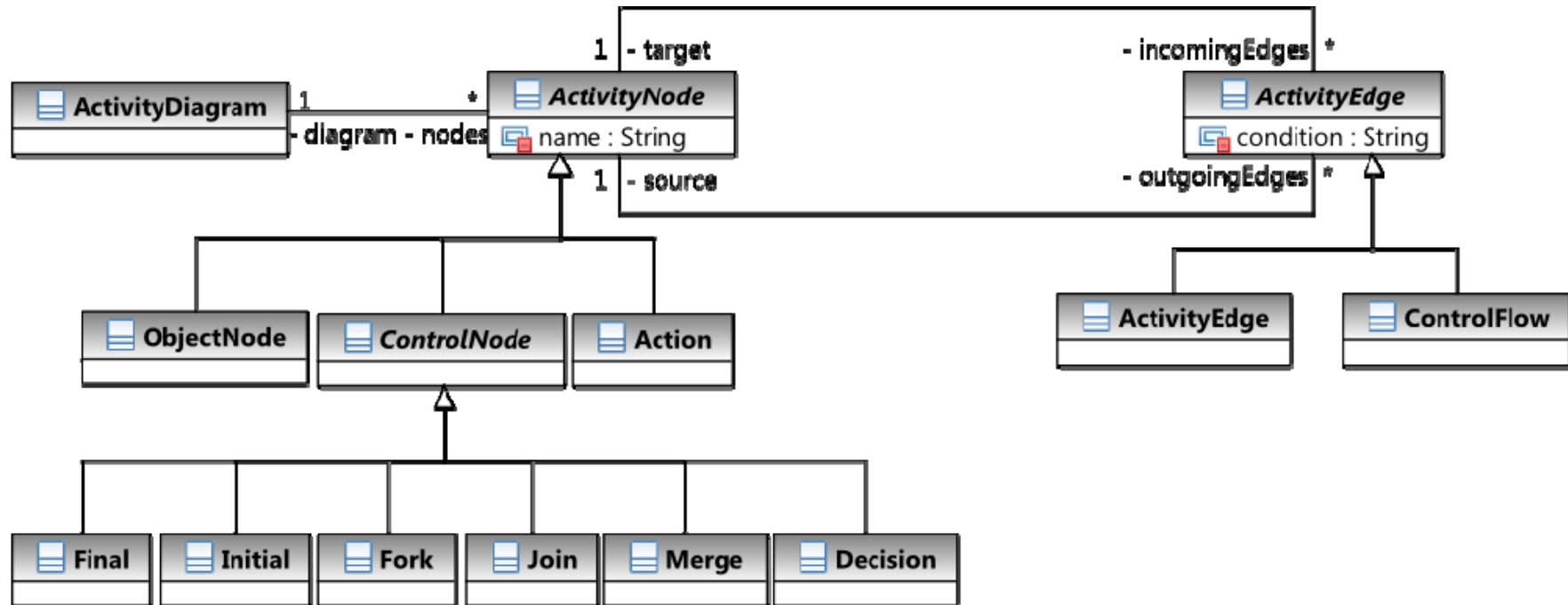
- Model (Activity Diagram)
 - designed by language user





Language Designer

- M2 metamodel (Activity Diagram)
 - ◆ conforms to Ecore M3 metamodel
 - ◆ visualized using concrete syntax of UML class diagrams



Example: Metamodel for Process Models (2)



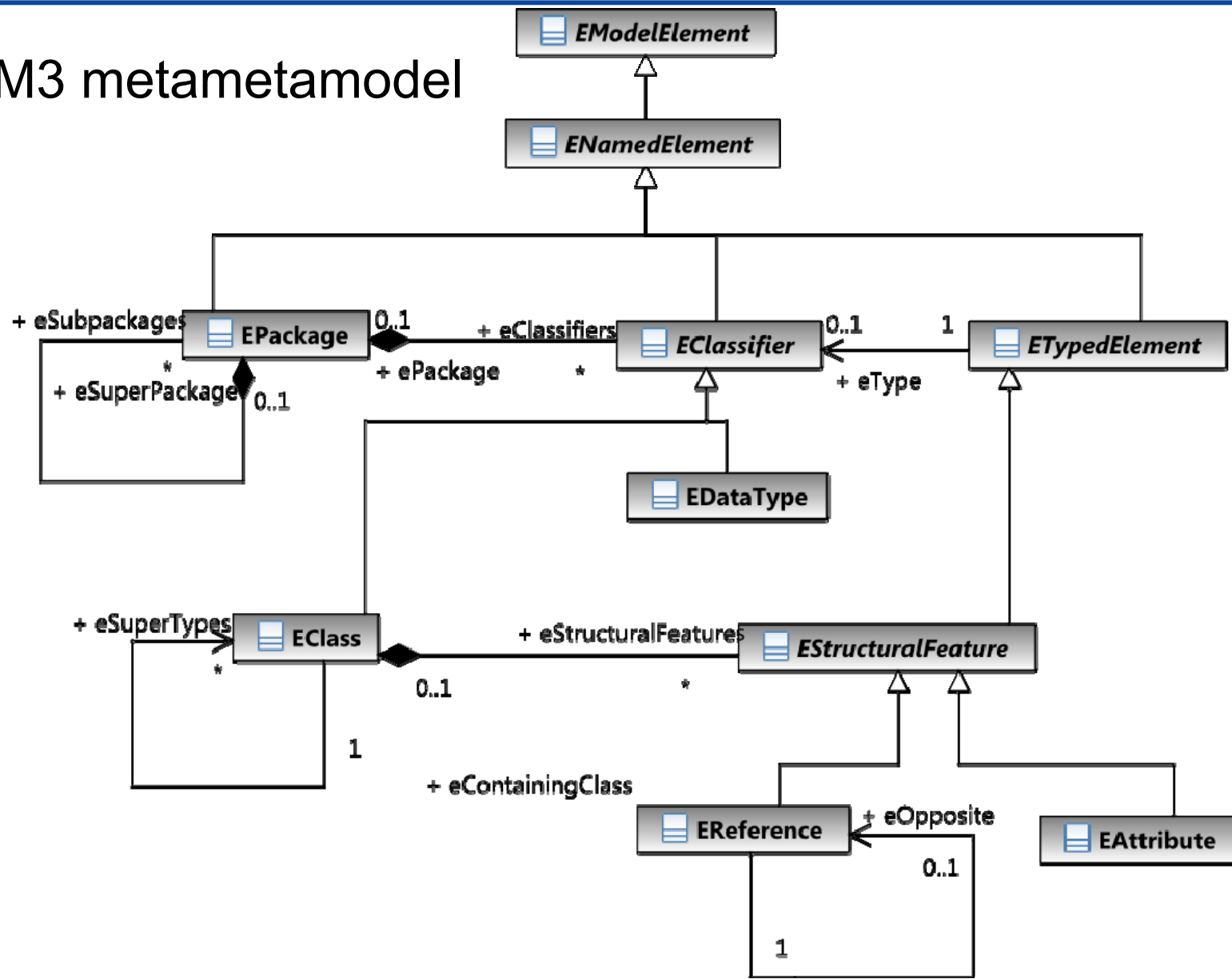
- Ecore-based metamodel (Activity Diagram)
 - ◆ conforms to Ecore M3 metamodel
 - ◆ visualized using **textual concrete syntax**

```
abstract class ActivityNode {
    reference incoming [0-*] : ActivityEdge oppositeOf target;
    reference outgoing [0-*] : ActivityEdge oppositeOf source;
}
class ObjectNode extends ActivityNode { }
class Action extends ActivityNode {
    attribute name : String;
}

abstract class ControlNode extends ActivityNode { }
class Initial extends ControlNode { }
class Final extends ControlNode { }
class Fork extends ControlNode { }
class Join extends ControlNode { }
class Merge extends ControlNode { }
class Decision extends ControlNode { }

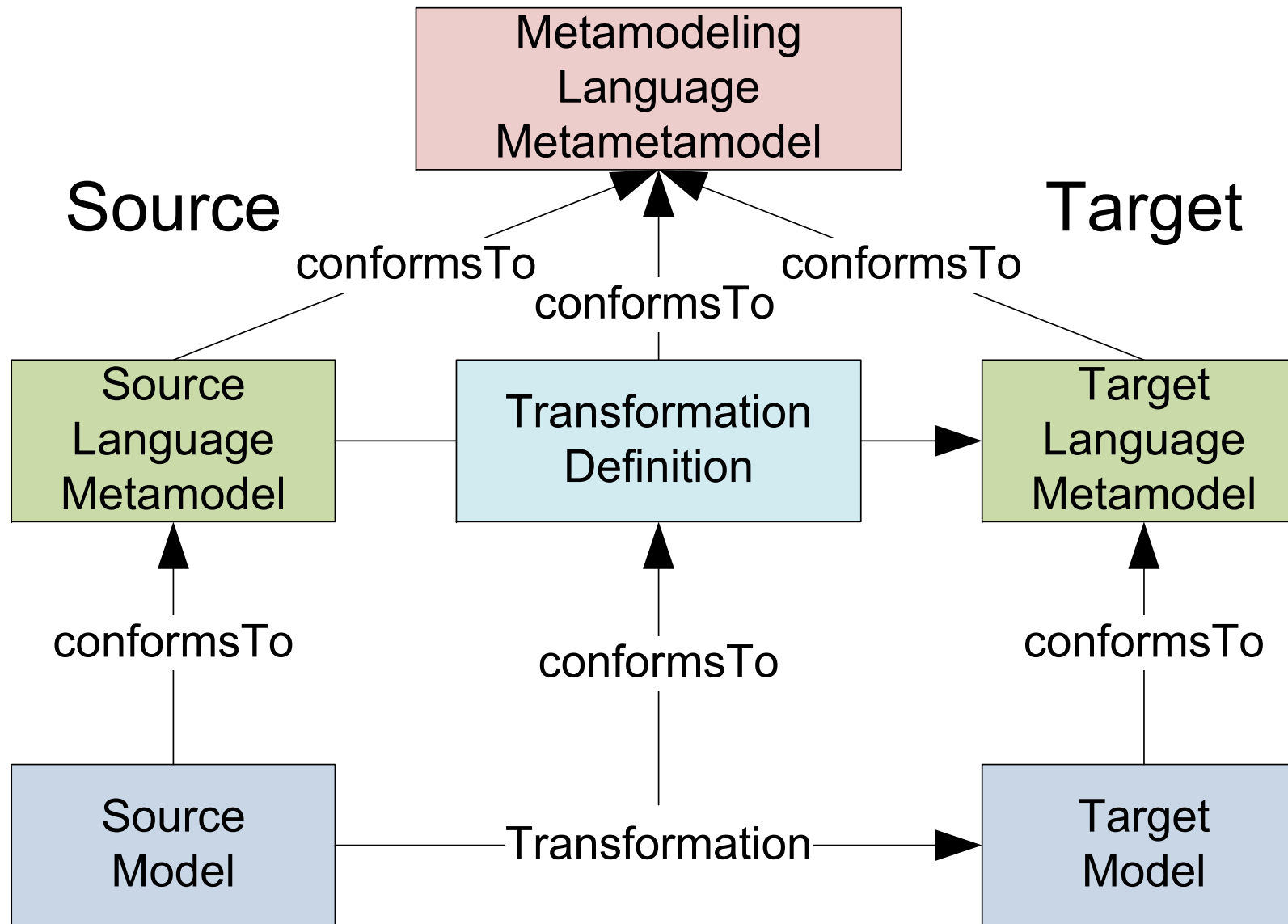
abstract class ActivityEdge {
    reference source [1-1] : ActivityNode;
    reference target [1-1] : ActivityNode;
}
class ObjectFlow extends ActivityEdge { }
class ControlFlow extends ActivityEdge { }
```

- Ecore M3 metamodel



- Take 10 minutes
- Write down a metamodel for Java method declarations (or for your other favorite language)
- Suggestion:
write down 3 example method declarations first

- Take 5 minutes
- Write down a metametamodel i.e. a metamodel language for the metamodel describing Java method declarations



- Metamodels as abstract grammar for our models
- Abstract grammar defines what is related to what
 - ◆ Undefined: word order
 - ◆ Undefined: semantics
 - ◆ Undefined: visual syntax
(e.g. textual or diagrammatic syntax)

- Now: ontologies are models, too!

Linguistic instantiation of the OWL2 metamodel in OWL2 Functional Style Syntax (axiom-based)

SubClassOf(Activi tyEdge owl : Thi ng)

SubClassOf(Activi tyNode owl : Thi ng)

ObjectPropertyDomain(to Activi tyEdge)

ObjectPropertyRange(to Activi tyNode)

ObjectPropertyDomain(outgoing Activi tyNode)

ObjectPropertyRange(outgoing Activi tyEdge)

ObjectPropertyDomain(from Activi tyEdge)

ObjectPropertyRange(from Activi tyNode)

ObjectPropertyDomain(incoming Activi tyNode)

ObjectPropertyRange(incoming Activi tyEdge)

ClassAssertion(acti on1 Acti on)

<http://www.w3.org/TR/owl2-syntax/>

OWL2 Manchester Style Syntax (frame-based) requires a different metamodel

Class: ActivityEdge
SubClassOf: owl:Thing

Class: ActivityNode
SubClassOf: owl:Thing

ObjectProperty: to
Domain: ActivityEdge
Range: ActivityNode

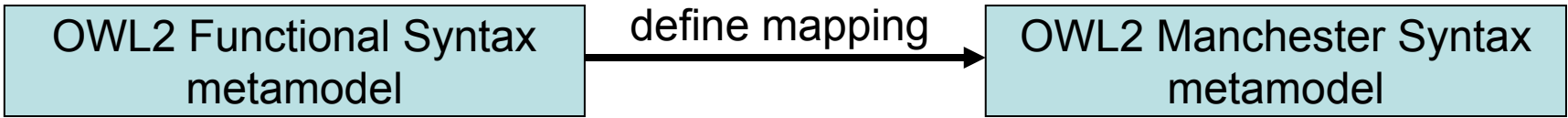
ObjectProperty: outgoing
Domain: ActivityNode
Range: ActivityEdge

ObjectProperty: from
Domain: ActivityEdge
Range: ActivityNode

ObjectProperty: incoming
Domain: ActivityNode
Range: ActivityEdge

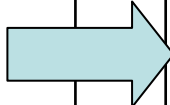
Individual: action1
Types: Action

<http://www.w3.org/TR/owl2-manchester-syntax/>



```
SubClassOf(ActivityEdge owl:Thing)
SubClassOf(ActivityNode owl:Thing)
ObjectPropertyDomain(to ActivityEdge)
ObjectPropertyRange(to ActivityNode)
ObjectPropertyDomain(outgoing
ActivityNode)
ObjectPropertyRange(outgoing
ActivityEdge)
ObjectPropertyDomain(from
ActivityEdge)
ObjectPropertyRange(from ActivityNode)
ObjectPropertyDomain(incoming
ActivityNode)
ObjectPropertyRange(incoming
ActivityEdge)
ClassAssertion(action1 Action)
```

derive
translation

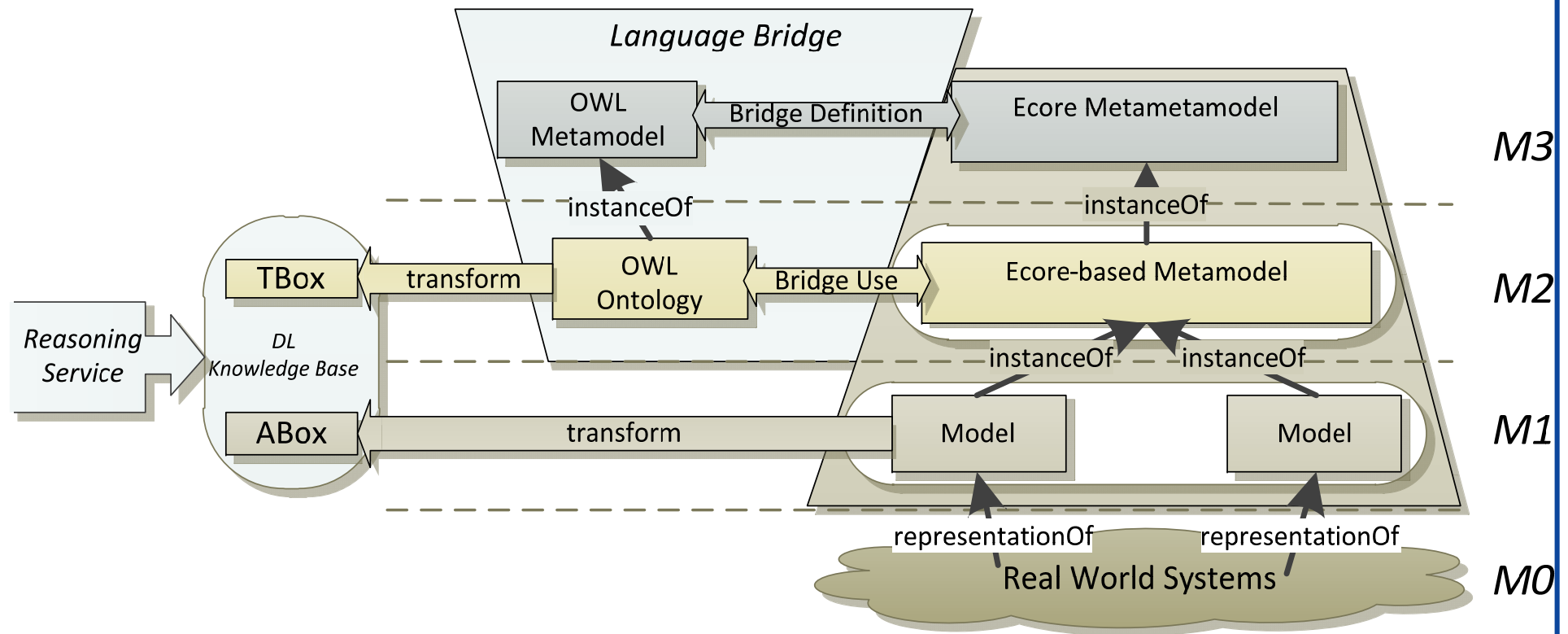


```
Class: ActivityEdge
SubClassOf: owl:Thing
Class: ActivityNode
SubClassOf: owl:Thing
ObjectProperty: to
Domain: ActivityEdge
Range: ActivityNode
ObjectProperty: outgoing
Domain: ActivityNode
Range: ActivityEdge
ObjectProperty: from
Domain: ActivityEdge
Range: ActivityNode
ObjectProperty: incoming
Domain: ActivityNode
Range: ActivityEdge
Individual: action1
Types: Action
```

- Many similar constructs

Ecore	OWL
package	ontology
class	class
instance and literals	individual and literals
reference, attribute	object, data property
data types	data types
enumeration	enumeration
multiplicity	cardinality
Opposite reference	Inverse object properties

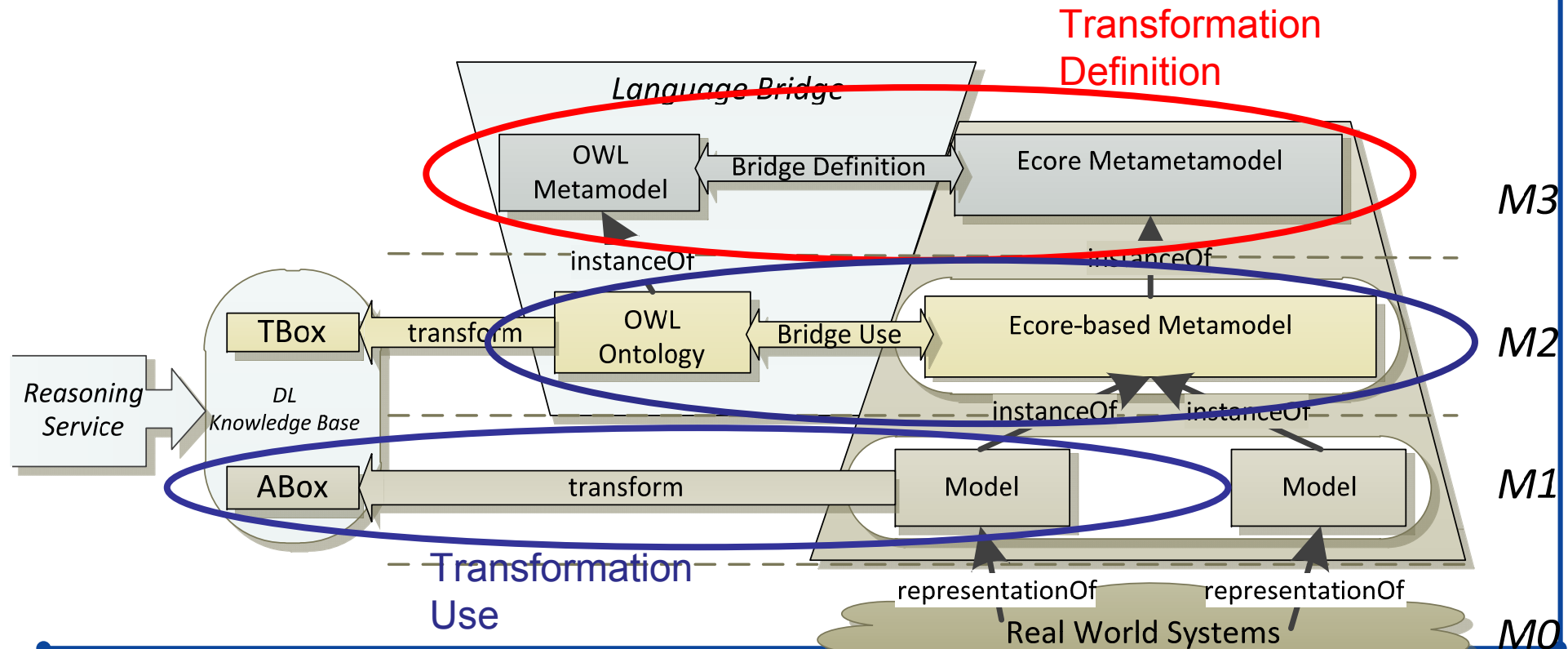
- Reasoners use logic-based language
- Representation of software models in a logic-based language
- We need bridges!



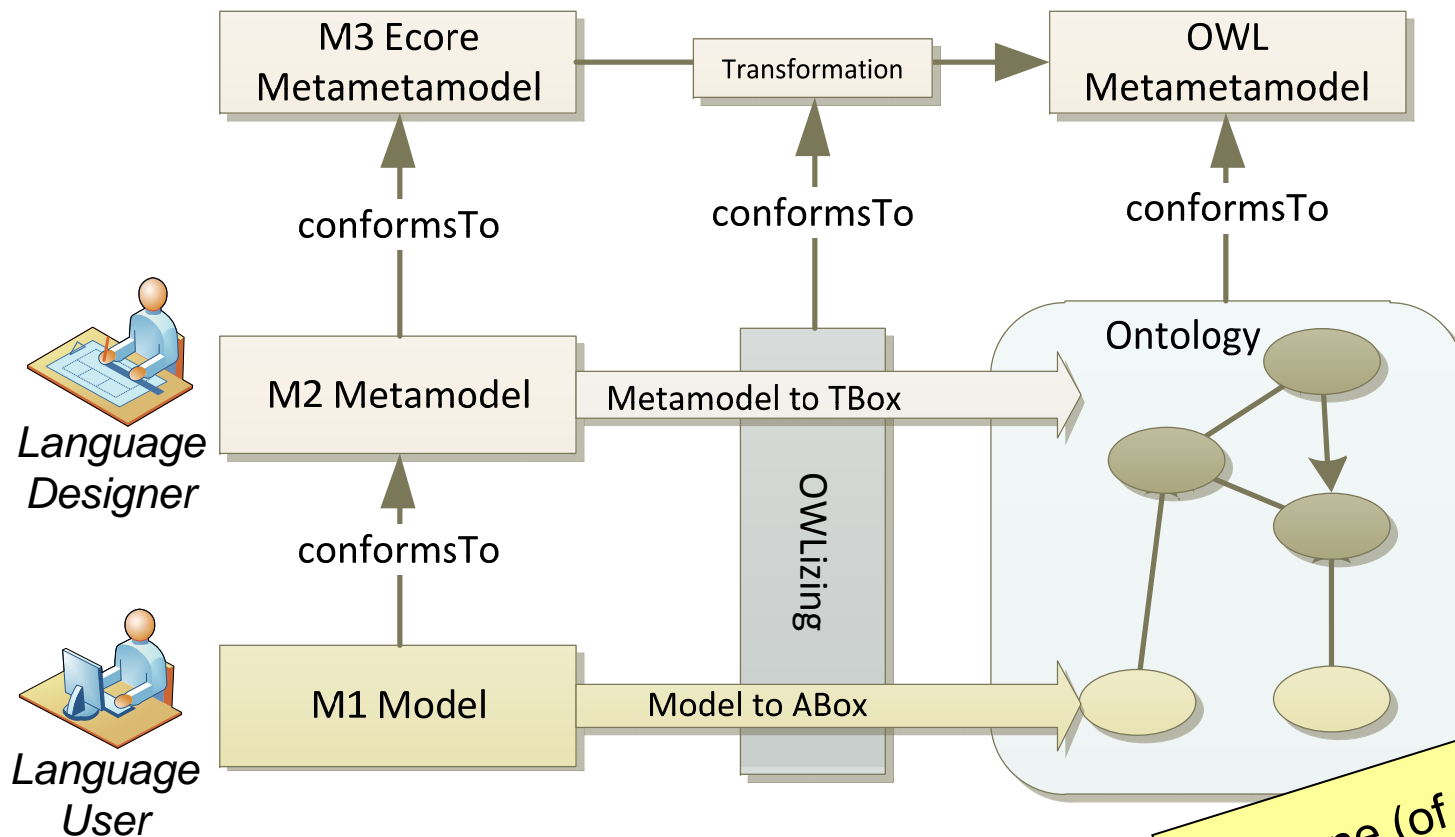
- 2 kinds of Language Bridges
 - ◆ M3 Transformation bridge
 - ◆ M3 Integration bridge

Just one (of many) possible pictures

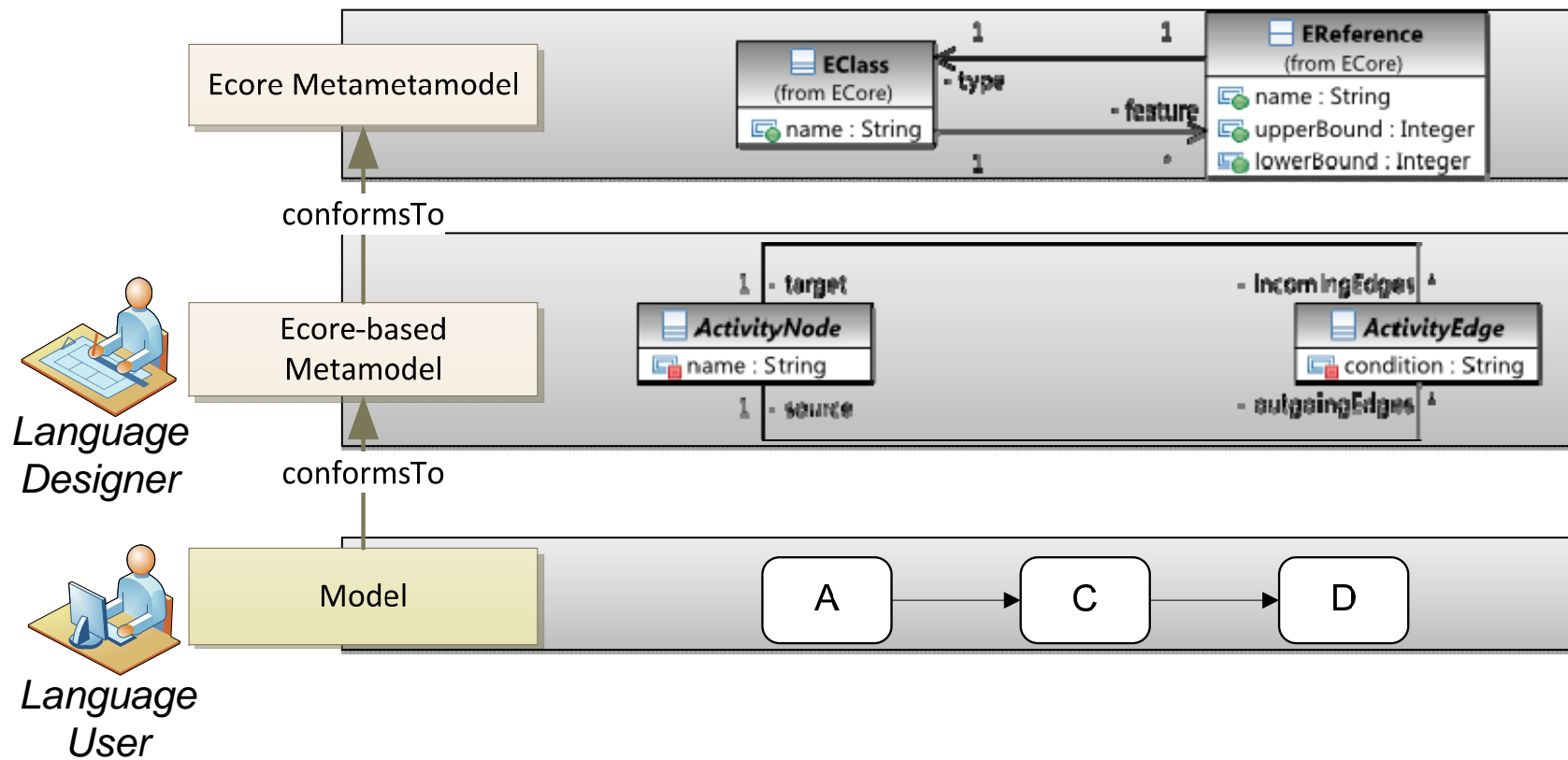
- Transformation of Ecore Technical Space with Ontology Language OWL2
 - ◆ Transformation of Ecore-based Metamodels
 - ◆ Transformation of conforming Models



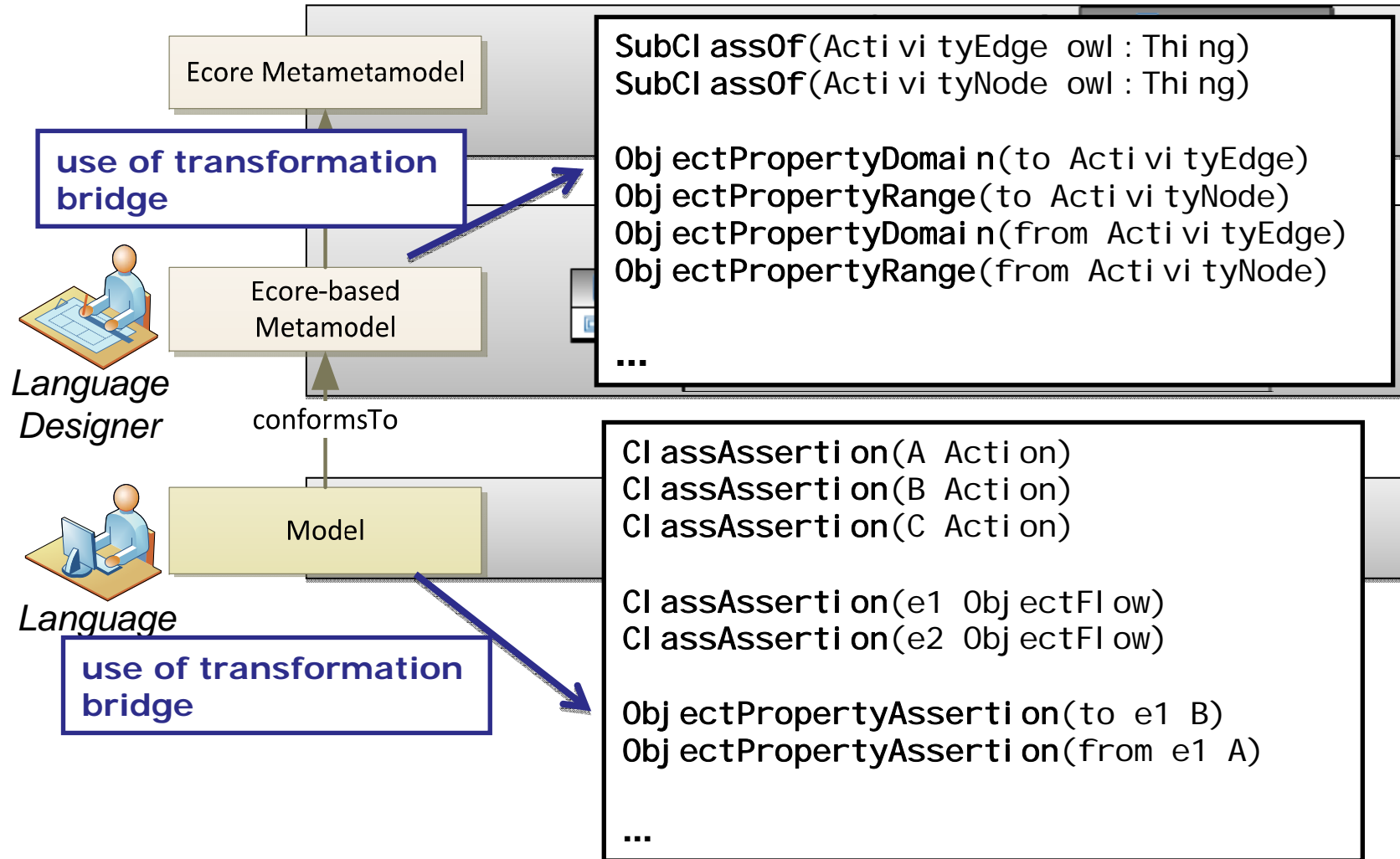
- OWLizer
 - ◆ Transforming Software Metamodels to Ontology TBox
 - ◆ Transforming Software Models to Ontology ABox



Just one (of many) possible pictures



- Definition of Transformation Bridge
 - Model transformation rules according to mappings of constructs similar in Ecore and OWL2



- Transforming Ecore Models and Metamodels into OWL using the OWLizer.mp4

- Transforming some class-like structures to OWL is sort of trivial

But:

- A metamodel-based approach saves work if multiple output formats are expected
- Reasoning on the output/input structures may be helpful
- The demonstrations neglect (so far) the distinction between:
 - ◆ Ecore: Linguistic instantiation
 - The way how things may be phrased in a sentence is constrained
 - ◆ OWL: Semantic instantiation
 - The way how things are factually related in the world is constrained

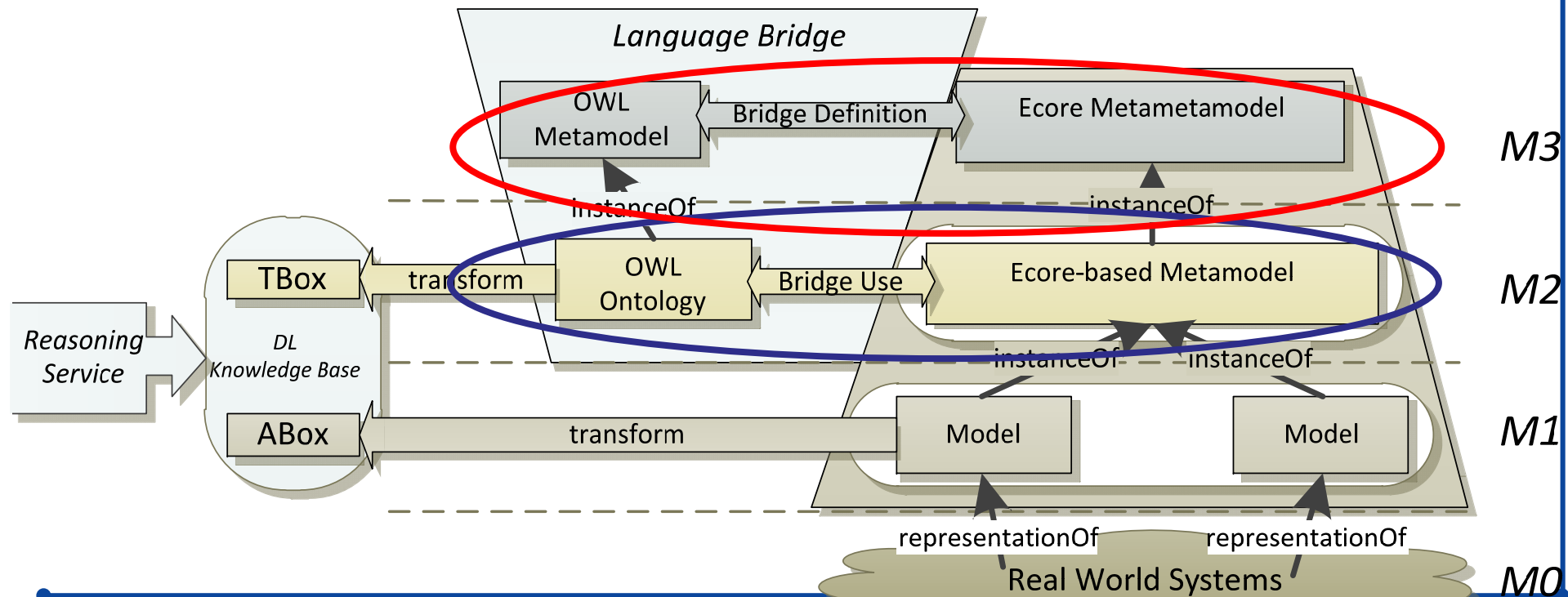
- Integration of Ecore Technical Space with Ontology Language OWL2

Integration of abstract Syntax

- OWL2 concrete Syntax expressions

Integration of concrete Syntax

- OWL2 concrete Syntax expressions



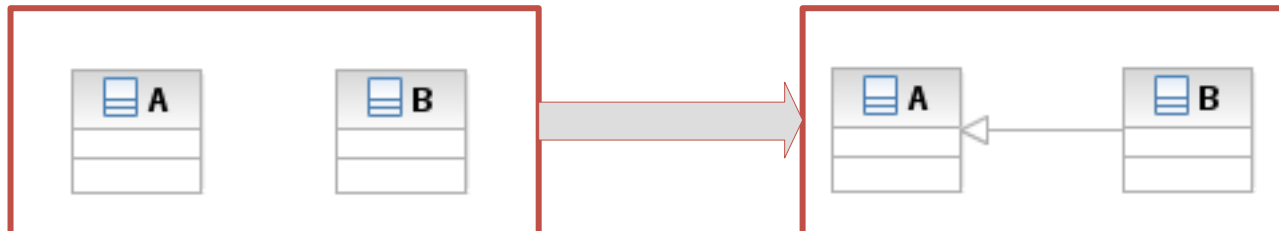
- Metamodel Integration by using Integration Operations

- ◆ Merge of concepts

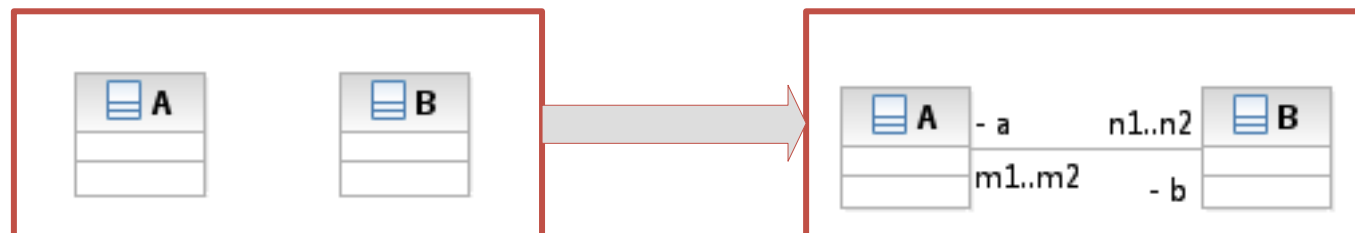


- ◆ Relation of concepts by

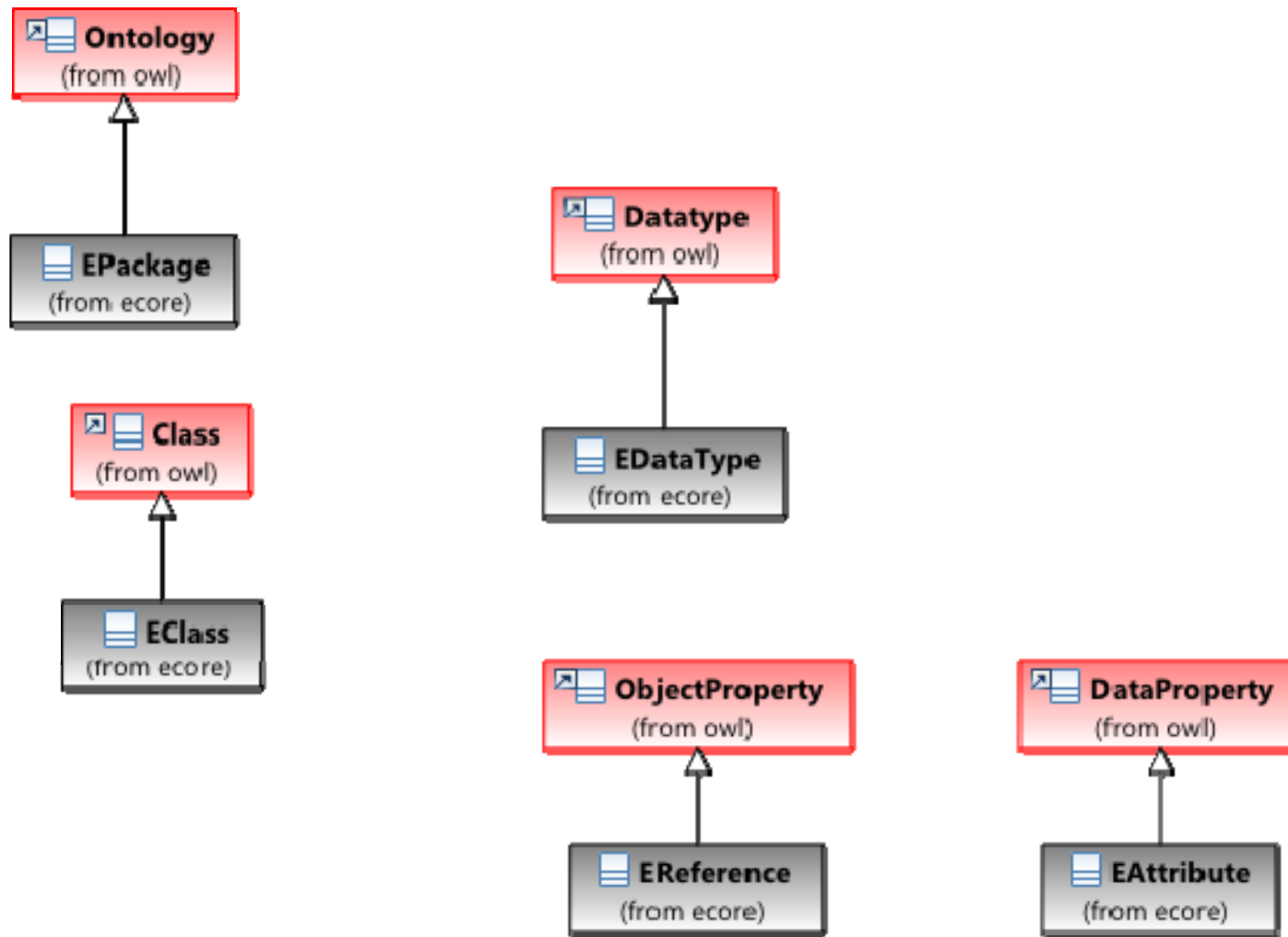
- Specialization Relationship



- Association



- Integration of abstract Syntax
 - Result: Integrated Metamodel



- Textual Concrete Syntax for Coding Metamodels
 - Simple Syntax, similar to Java
- Grammar (concrete Syntax):

```
class ::= ["abstract"] "class" name [supertypes] "{" features "}";  
...  
feature ::= attribute | reference;  
...  
attribute ::= "attribute" name multiplicity ":" typeref ";"  
...  
reference ::= "reference" name multiplicity [iscontainer] ":"  
             typeref ["oppositeOf" name] ";"  
...
```

▪ (textual) Activity Diagram Metamodel

```
abstract class ActivityNode {
    reference incoming [0-*] : ActivityEdge opposite target;
    reference outgoing [0-*] : ActivityEdge opposite source;
}
class ObjectNode extends ActivityNode { }
class Action extends ActivityNode {
    attribute name : String;
}

abstract class ControlNode extends ActivityNode { }
class Initial extends ControlNode { }
class Final extends ControlNode { }
class Fork extends ControlNode { }
class Join extends ControlNode { }
class Merge extends ControlNode { }
class Decision extends ControlNode { }

abstract class ActivityEdge {
    reference source [1-1] : ActivityNode;
    reference target [1-1] : ActivityNode;
}
class ObjectFlow extends ActivityEdge { }
class ControlFlow extends ActivityEdge { }
```

- New non-terminals:
 - `classAxioms`: produces a list of OWL Class Axioms
 - `objectPropertyAxioms`: produces a list of OWL Object Property Axioms
 - `dataPropertyAxioms`: produces a list of OWL Data Property Axioms

```
class ::= ["abstract"] "class" name [supertypes] [classAxioms] "{" features "}";
classAxioms ::= ClassAxiom { ", " ClassAxiom };

reference ::= "reference" name multiplicity iscontainer ":" typeref "oppositeOf"
            name [objectPropertyAxioms] ";";

objectPropertyAxioms ::= ObjectPropertyAxiom { ", " ObjectPropertyAxiom };

attribute ::= "attribute" name multiplicity ":" typeref [dataPropertyAxioms] ";";

dataPropertyAxioms ::= DataPropertyAxiom { ", " DataPropertyAxiom };
```

- Extended Metamodel of Activity Diagram (excerpt)

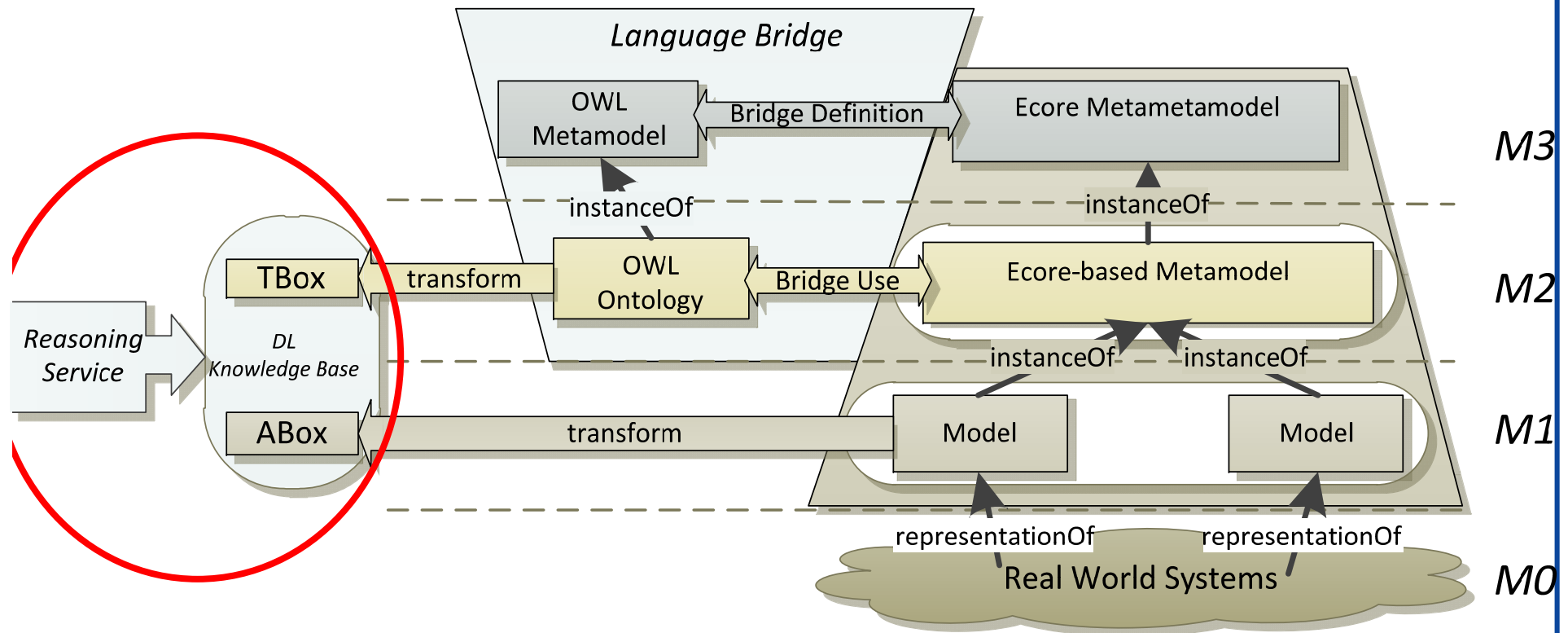
```
class ActivityNode equivalentWith restrictionOn edge with some Final {  
  reference incoming [0-*] : ActivityEdge oppositeOf target;  
  reference outgoing [0-*] : ActivityEdge oppositeOf source;  
  
  transitive reference edge [0-*] : ActivityNode isChain(outgoing, target);  
}  
  
...  
  
class Initial extends ControlNode,  
  subclassOf restrictionOn outgoing with some  
  (restrictionOn with some (Action or ControlNode))  
{  
  
}
```

Why?

Its the reasoning, stupid!

- extending ecore models with owl annotations.mp4

- Consistency Checking
- Satisfiability Checking
- Classification
- Subsumption Checking
- Explanation
- Querying



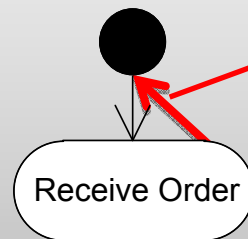
<i>Name</i>	Consistency Checking
<i>Signature</i>	boolean consistency (Ontology O)
<i>Description</i>	Checks if the given ontology O is consistent, i.e. if there exists a model (a model-theoretic instance) for O. If ontology O is consistent, then return true. Otherwise return false.
<i>Pattern</i>	$b = \text{consistency}(O)$
<i>Input</i>	An Ontology O
<i>Output</i>	$b = \text{true}$ iff O is consistent, $b = \text{false}$ otherwise

- Accomplished Service
 - ◆ Ensures that a Model does not contain any contradictory facts with regard to its Language Metamodel
 - ◆ Requirements for Language Designers
 - Possibility to define Constraints and Restrictions
 - Define Axioms
 - ◆ Benefits for Language User
 - Sound Models

M2 Metamodel

```
class ActivityNode equivalentWith restrictionOn edge with some Final {  
  reference incoming [0-*] : ActivityEdge oppositeOf target;  
  reference outgoing [0-*] : ActivityEdge oppositeOf source;  
  
  transitive reference edge [0-*] : ActivityNode isChain(outgoing, target);  
}
```

M1 Model



Inconsistency:
Disallowed Edge to
Initial Action

Name	Satisfiability checking
Signature	Set<Concept> GetUnsatisfiable (Ontology O)
Description	Find all unsatisfiable concepts in given ontology O. A concept in an ontology is unsatisfiable if it is an empty set. Return NULL if there is not any unsatisfiable concept.
Pattern	b = GetUnsatisfiable (O)
Input	An Ontology O
Output	b = NULL iff there is no unsatisfiable concept b = a set of unsatisfiable concepts otherwise

- Accomplished Service
 - ◆ Finds unsatisfiable classes in a metamodel

- Benefits for Language Designers
 - ◆ Higher quality metamodels, where all classes can be instantiated

- Benefits for Language Users
 - ◆ Less inconsistencies in models (because instances of unsatisfiable classes lead to inconsistencies)

M2 Metamodel

```
class ActivityNode equivalentWith restrictionOn edge with some Final {  
  reference incoming [0-*] : ActivityEdge oppositeOf target;  
  reference outgoing [0-*] : ActivityEdge oppositeOf source;  
  
  transitive reference edge [0-*] : ActivityNode isChain(outgoing, target);  
}  
  
class Final extends ControlNode  
  subclassOf (restrictionOn edge with some ActivityNode) and  
  not(restrictionOn edge with some ActivityNode)  
{ }
```

Unsatisfiable Class:
two contradictory
restrictions

Name	Classification
Signature	boolean classifiesAs (Ontology O, concept A, individual i)
Description	Checks if the given individual i is an instance of concept A in the ontology ref, then return true. Otherwise return false.
Pattern	$b = \text{classifiesAs} (O, A, i)$
Input	An Ontology O, Concept A and Individual i
Output	$b = \text{true}$ iff i is an instance of A, $b = \text{false}$ otherwise

- Accomplished Service
 - ◆ Determines the most specific type an Model Element belongs to
 - ◆ With respect to all Attributes and Properties in the Context of the Model Element

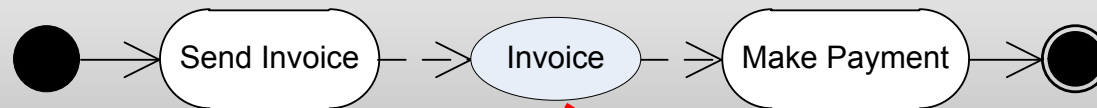
- Requirements for Language Designers
 - ◆ Define Axioms
 - ◆ Possibility to define Constraints and Restrictions

- Benefits for Language Users
 - ◆ Automatically Refinement of Model Elements
 - ◆ Suggestions of suitable domain concepts to be used

M2 Metamodel

```
class ObjectNode extends ActivityNode
    equivalentWith ((restrictionOn incoming with some ObjectFlow)
                    and (restrictionOn outgoing with some ObjectFlow))
{ }
```

M1 Model



Classify Invoice Node
Result: It is of type
ObjectNode

Name	Subsumption Checking
Signature	boolean subsumes (Ontology O, concept A, concept B)
Description	Checks whether the interpretation of A is a subset of the interpretation of B in the given ontology O. If the interpretation of A is a subset of the interpretation of B, then return true. Otherwise returns false.
Pattern	$b = \text{subsumes}(O, A, B)$
Input	An Ontology O and concepts A, B
Output	$b = \text{true}$ iff the interpretation of A is a subset of the interpretation of B in the ontology O, $b = \text{false}$ otherwise

- Accomplished Service
 - ◆ Computes a Subsumption Hierarchy of all Classes
 - ◆ based on all Class Expressions and Axioms in the Ontology

- Requirements for Language Designers
 - ◆ Define Axioms
 - ◆ Possibility to define Expressions and Restrictions

Name	Explanation
Signature	Set<Axiom> getExplanation (Ontology O, axiom Ax)
Description	Retrieve the set of axioms that entail axiom Ax in the given ontology, then return them.
Pattern	b = getExplanation (O,Ax)
Input	An Ontology O and axiom Ax
Output	b = set of axioms that entail the given axiom Ax. b = NULL otherwise

- Accomplished Service
 - ◆ Explanations for subsumptions and unsatisfiable classes in metamodels
 - ◆ Explanations for inconsistencies in models

- Benefits for Language Designers
 - ◆ Debugging of metamodels

- Benefits for Language Users
 - ◆ Debugging of models

M2 Metamodel

```
class ActivityNode equivalentWith restrictionOn edge with some Final {  
  reference incoming [0-*] : ActivityEdge oppositeOf target;  
  reference outgoing [0-*] : ActivityEdge oppositeOf source;  
  
  transitive reference edge [0-*] : ActivityNode isChain(outgoing, target);  
}  
  
class Final extends ControlNode  
  subclassOf (restrictionOn edge with some ActivityNode) and  
  not(restrictionOn edge with some ActivityNode)  
{ }
```

Explanation from TwoUse Toolkit

Unsatisfiability of Final:

Explanation:

Final equivalentTo not edge some ActivityNode
and edge some ActivityNode

Name	Query answering
Signature	Set answering (Ontology O, query q)
Description	checks the answer sets of a query q to ontology O.
Pattern	res = answering (Ontology O, query q)
Input	An Ontology O, and a query q
Output	res = a set of answers of the query to the ontology iff there is a answer set res = NULL if there is not any answer

- Accomplished Service
 - ◆ Query for model elements and metamodel element
 - ◆ Use of OWL2 entailment regime

- Benefits for Language Designers
 - ◆ Retrieving information on concepts

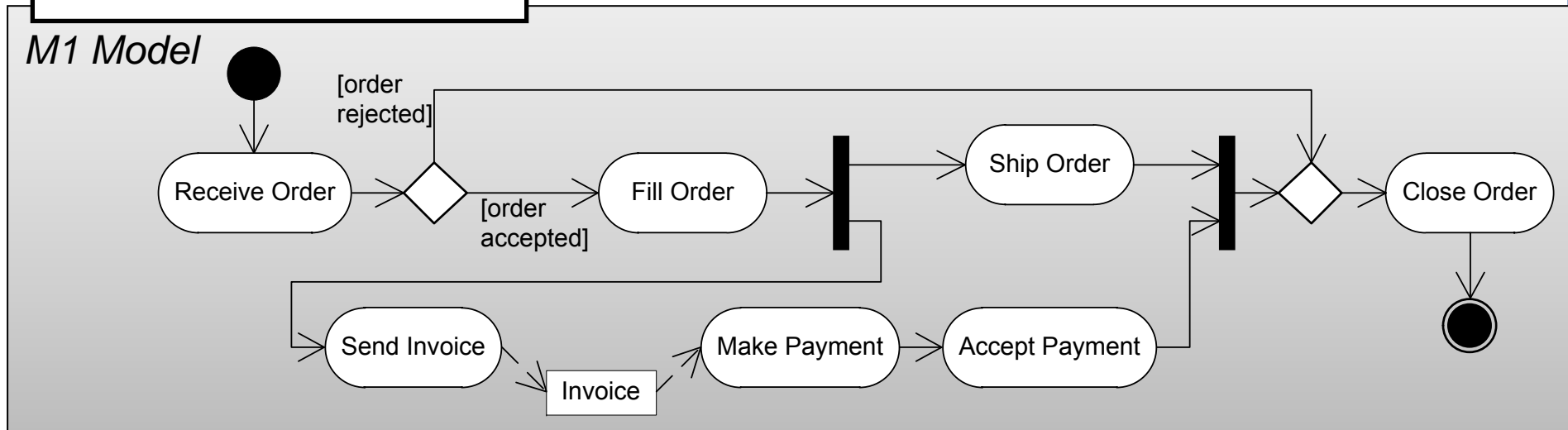
- Benefits for Language Users
 - ◆ Retrieving existing and complex parts of models
 - ◆ E.g. [Tappolet-2010].

- Find all actions, that are executed before „Ship Order“:

```
Query ( ClassAssertion(?i Action),  
ObjectPropertyAssertion(InverseInverseObjectProperties(edge)  
?i shi porder))
```

Results:

```
-----  
| c |  
=====  
| fill order |  
| receiveorder |  
-----
```



- Find all unsatisfiable classes:

```
Query ( SubClassOf(?c owl : Nothing) )
```

Results:

```
-----  
| c |  
=====  
| Nothing |  
| Final |  
-----
```

- Find all concepts that do not go via edge to Final:

```
Query ( EquivalentClasses(?c  
                          Not (ObjectSomeValuesFrom(edge Final))))
```

Results:

```
-----  
| c |  
=====  
| ActivityDiagram |  
-----
```

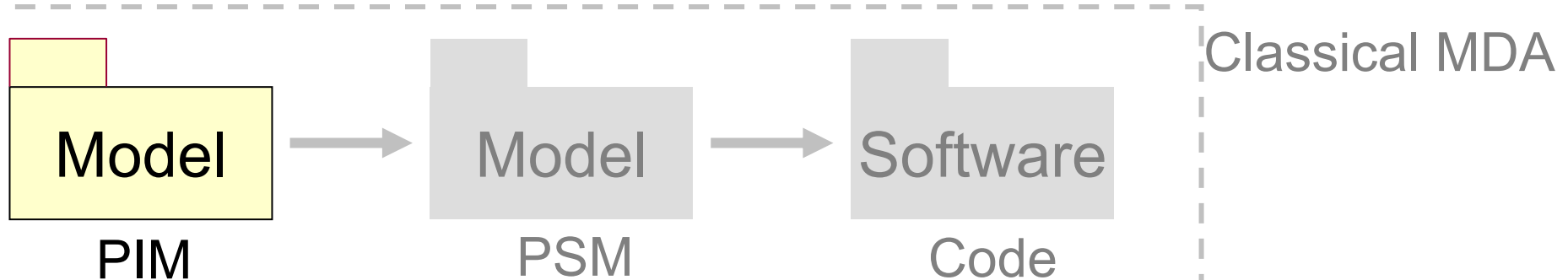
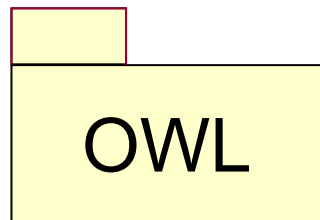
Mapping to:

- Abox: Incomplete vs. Complete models
- Tbox: no problem

Tools support:

- Open World yes
- Closed Domain yes
- Closed World not really

- Ontologies are software models
- Metamodeling facilitates life:
 - ◆ Use of established modeling tools
 - ◆ definition of translations
- Reasoning not restricted to UML Class Diagrams



... Questions?

Semantic Model-driven Engineering

Improving Design Patterns by Description Logics: An integration bridge

Steffen Staab

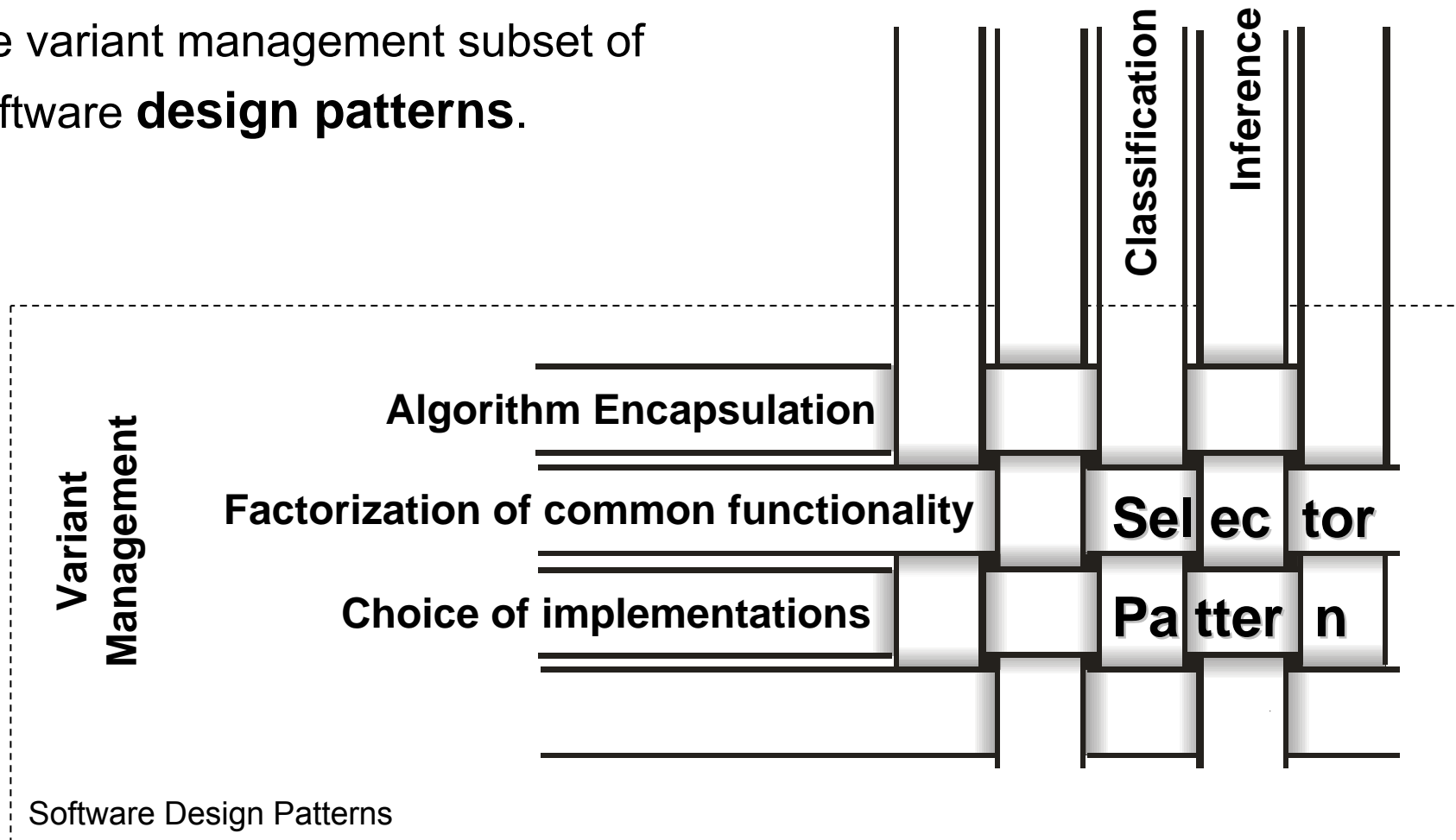
with

Gerd Gröner, Fernando Silva Parreiras, Tobias Walter

- Description Logics Reasoning by Example
- Model-driven engineering with OWL
- Refinements in several dimensions
 - ◆ Refinements along metamodeling levels [Models 2009, ECMFA 2010]
 - ◆ Refinements along model specification [DL 2009, EKAW 2010]
 - From business developer to software developer
 - ◆ **Refinements along platform specification** [ER 2008, DKE2010]
 - For Ontology Translations
 - ◆ Refinements along time
 - Metamodel evolution [ISWC2010]
 - Suggesting changes to transformations by DL reasoning
 - Ontology API Generation/Co-evolution [ICSC 2009]

Weaving features of **Description Logics** into the Strategy Pattern under the variant management subset of software **design patterns**.

Description Logics (OWL)

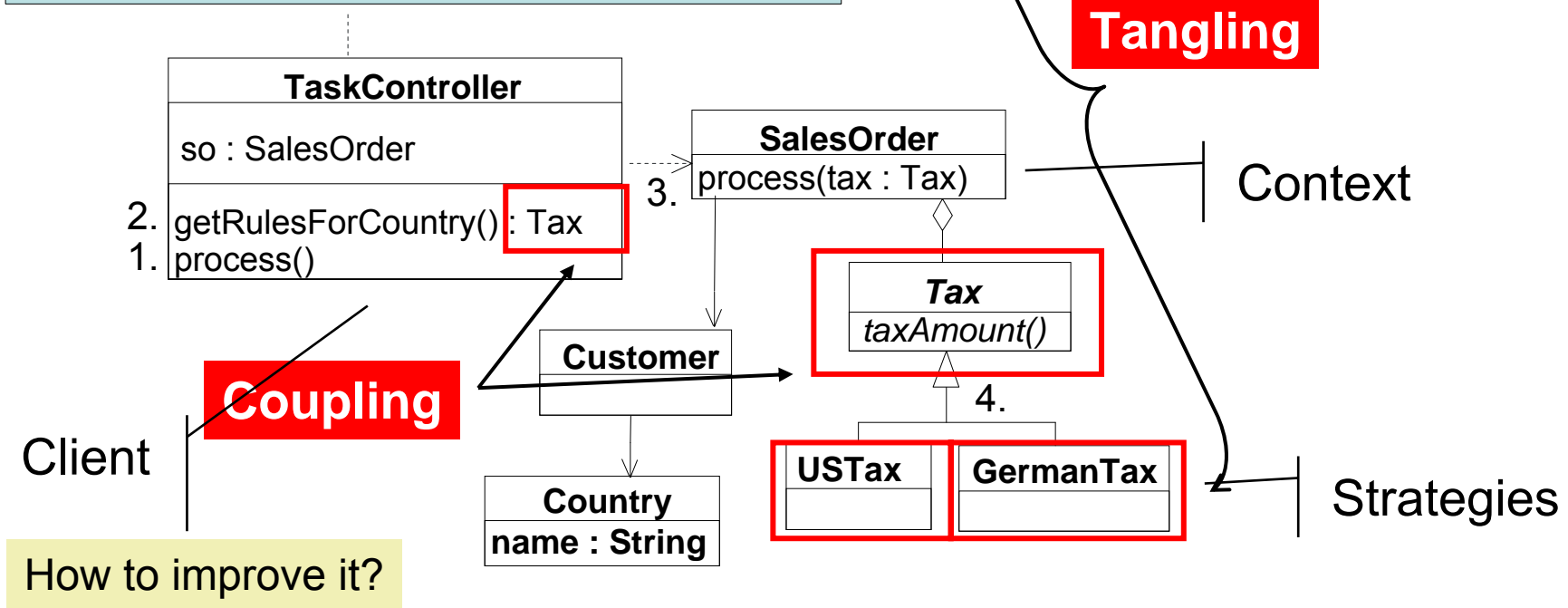


- An **order-processing system** for an international e-commerce company in the United States
- This system must be able to process **sales orders** in many different countries, like the **USA and Germany**, and handle **different tax calculations**.

Drawbacks:

```

context TaskController::getRulesForCountry():Tax
body.
if so.customer.country.name = 'USA' then
  USTax.new()
else
if so.customer.country.name = 'GERMANY' then
  GermanTax.new()
endif
endif
endif
    
```

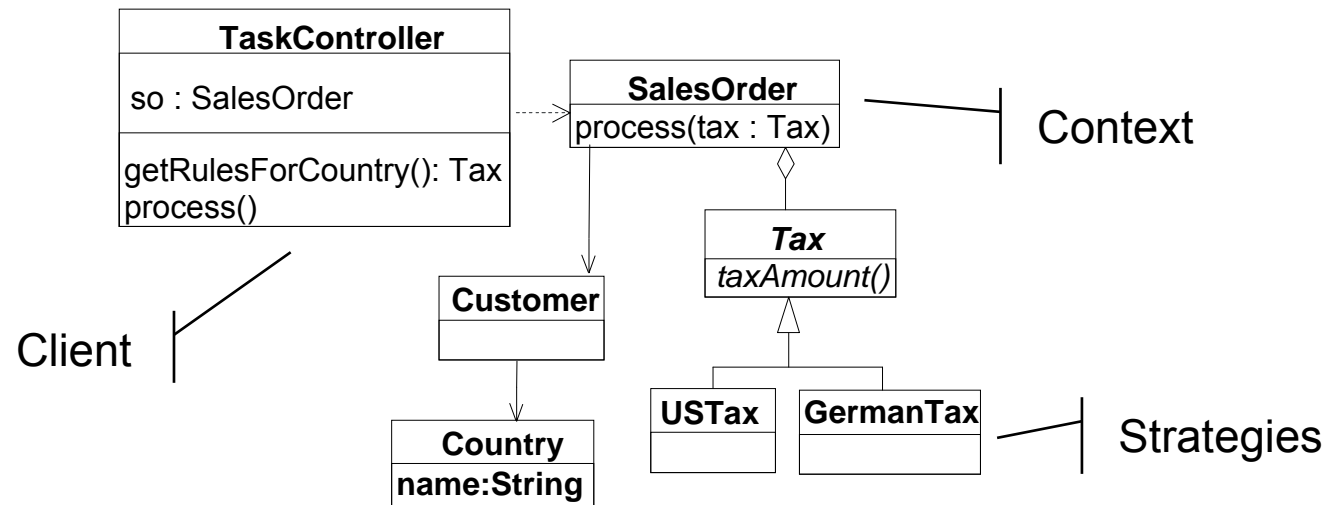


- The **TwoUse** solution

Uses an Ontology to describe **Context** and **Strategies**

Classifies dynamically the Context

- Hybrid Diagram
- Metamodel
- Transformation Process

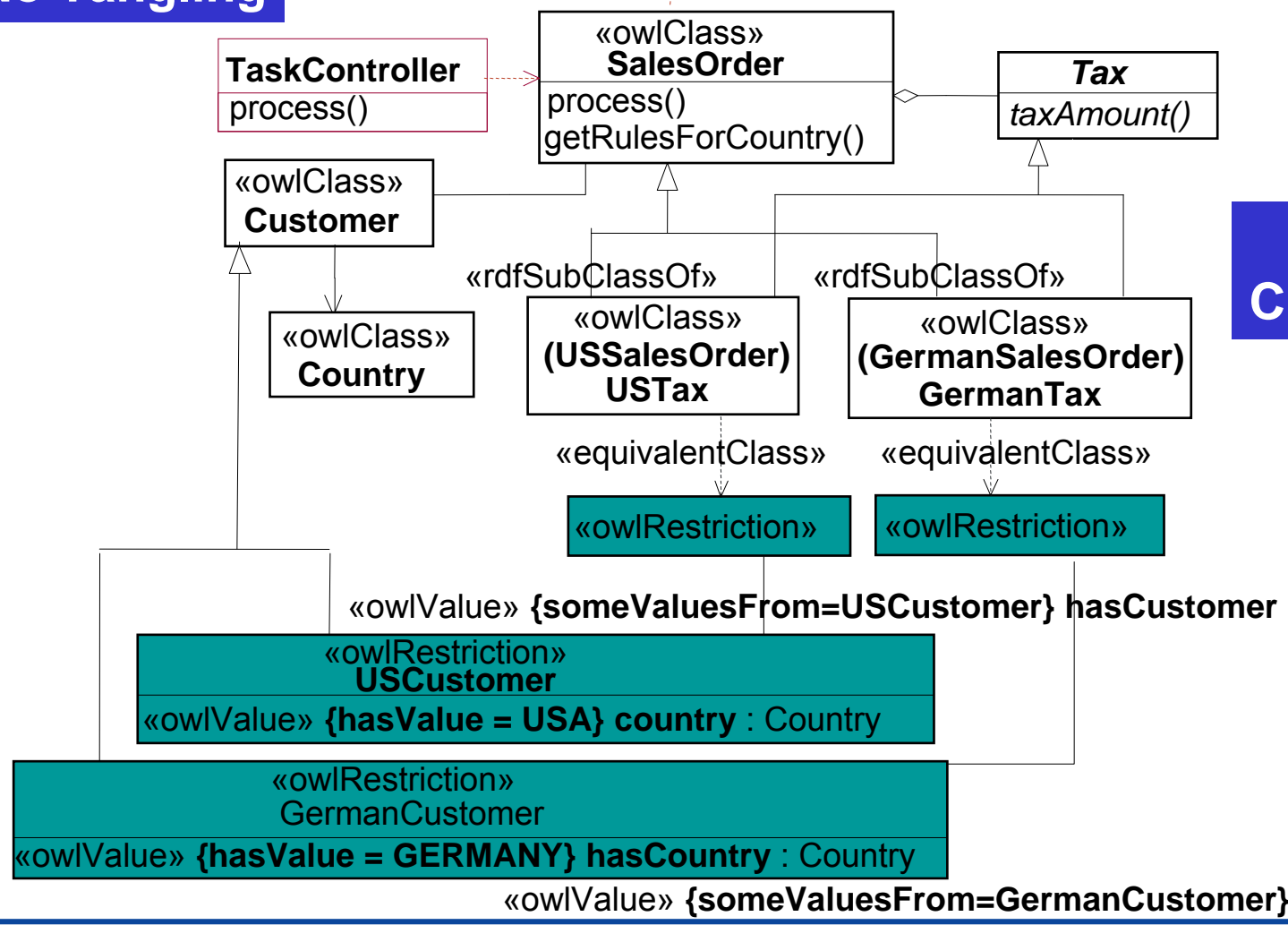


Hybrid Diagram: Strategy Pattern + OWL

No Coupling

No Tangling

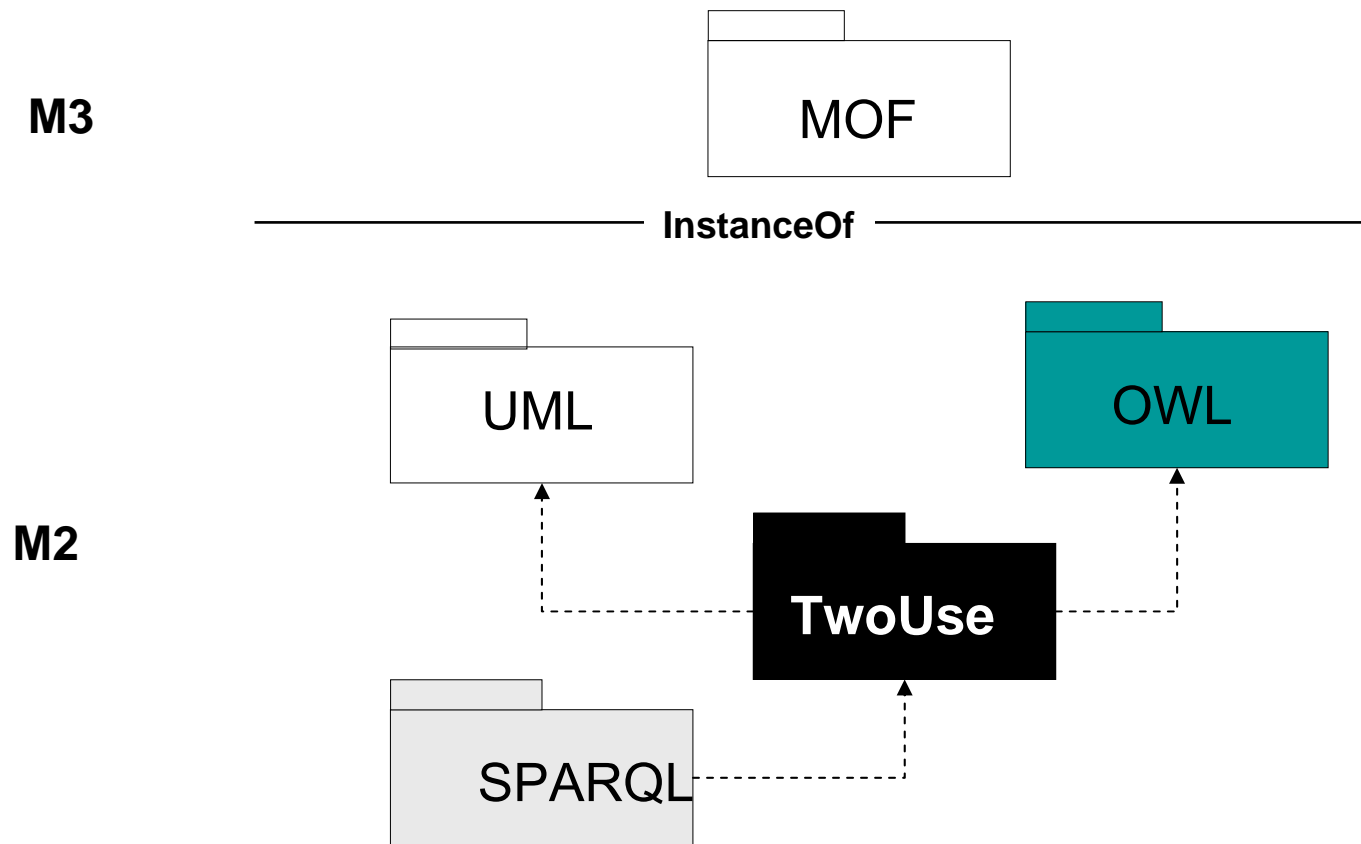
```
context SalesOrder::getRulesForCountry():OclTy
body:
  Select ?T where ?self directType ?T
```

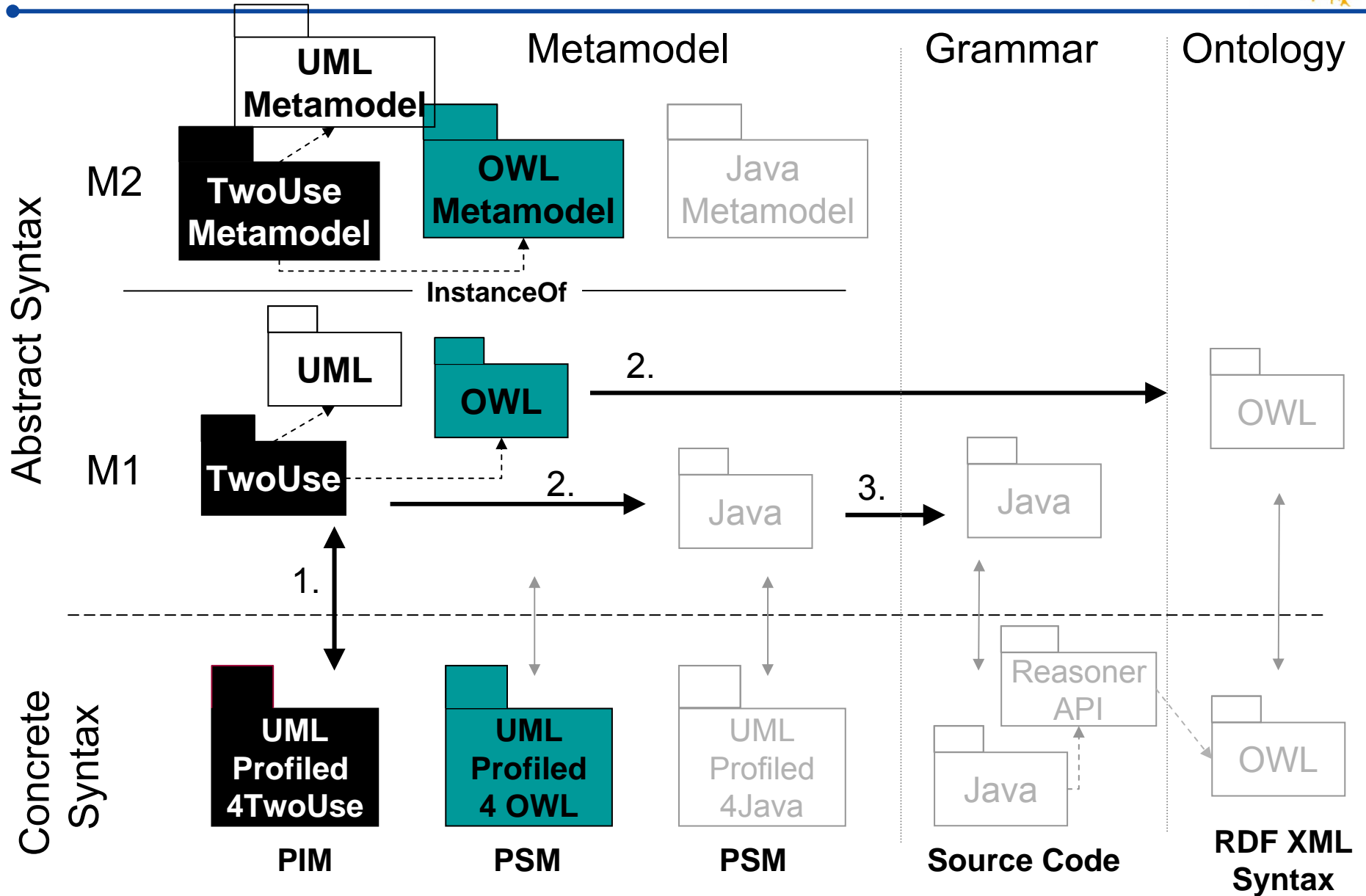


Dynamic Classification

Reuse

Flexibility





- The Ontology can be:
 - ♦ **reused** independently of platform;
 - ♦ **modeled and evolved** independently of the execution logic;
 - ♦ **tested automatically** by logical unit tests.
- ♦ **Changes** required for adoption **are minor**.

References

Software Language Engineering, 3rd Int. Conf, October 2010, Eindhoven

Applying model-driven engineering for the Semantic Web

- S. Brockmans, R. Volz, A. Eberhart, and P. Löffler. Visual modeling of OWL DL ontologies using UML. In *Proc. of ISWC 2004*, pages 198–213, 2004.
- Gasevic, D., Djuric, D., Devedzic, V.: *Model Driven Engineering and Ontology Development*, 2nd Ed. Springer (2009)
- F. S. Parreiras, G. Gröner, T. Walter, S. Staab. A model-driven approach for using templates in OWL ontologies. In: *Proc. of the European Conference on Knowledge Acquisition and Management – EKAW-2010*. Lisbon, Portugal, October 2008, LNCS, Springer.
- F. Silva Parreiras, C. Saathoff, T. Walter, T. Franz, S. Staab: APIs `a gogo: Automatic Generation of Ontology APIs. In: *IEEE Int. Conference on Semantic Computing*, IEEE Press, 2009.
- F. Silva Parreiras, S. Staab, S. Schenk, A. Winter. Model Driven Specification of Ontology Translations. *ER 2008 – 27th International Conference on Conceptual Modeling*, LNCS, Springer, 2008.

Applying RDF/OWL querying for software engineering

- Jonas Tappolet, Christoph Kiefer, Abraham Bernstein, Semantic web enabled software analysis, *Journal of Web Semantics: Science, Services and Agents on the World Wide Web* 8, July 2010
- G. Gröner, S. Staab. Modeling and Query Patterns for Process Retrieval in OWL. In: *Proc. of ISWC-2009 – International Semantic Web Conference*, Westfield,

References

Applying ontology reasoning for software engineering

- G. Gröner, S. Staab. Specialization and Validation of Statecharts in OWL. In: *Proc. of the European Conference on Knowledge Acquisition and Management – EKAW-2010*. October 2008, LNCS, Springer.
- F. Silva Parreiras, S. Staab, A. Winter. Improving Design Patterns by Description Logics: An Use Case with Abstract Factory and Strategy. In: *Proc. of Modellierung 2008*. LNI, Gi e.V, März 2008.
- Wang, H., Li, Y., Sun, J., Zhang, H., Pan, J.: Verifying Feature Models using OWL. *J of Web Semantics* 5(2) (2007) 117-129
- Y. Ren, G. Gröner, J. Lemcke, T. Rahmani, A. Friesen, Y. Zhao, J. Z. Pan, S. Staab. Validating Process Refinement with Ontologies. In: *The 22nd International Workshop on Description Logics (DL2009)*. 27 to 30 July 2009, Oxford, United Kingdom.

Integrating software and ontology engineering practices:

- F. Silva Parreiras, S. Staab, A. Winter. On Marrying Ontological and Metamodeling Technical Spaces. In: *ESEC/ACM FSE-2007 – Proceedings of the 6th joint meeting of the European software engineering conference and the 14th ACM SIGSOFT symposium on Foundations of software engineering*, September 03 - 07, 2007, Dubrovnik, Croatia. ACM 2007, pp. 439 - 448
- F. S. Parreiras, S. Staab. Using Ontologies with UML Class-based Modeling: The TwoUse Approach. In: *Data & Knowledge Engineering*, Elsevier, to appear.

Integrating software and ontology engineering practices:

Creating Domain-specific languages with OWL

- T. Walter, F. S. Parreiras, S. Staab, J. Ebert. Joint Language and Domain Engineering. In: B. Selic & T. Kühne (eds.). *Proc. of ECMFA-2010 – 6th European Conference on Modelling Foundations and Applications (formerly ECMDA-FA)*, LNCS, Springer, Paris, France, June 15-18.
- T. Walter, F. Silva Parreiras, S. Staab. OntoDSL: An Ontology-based Development Environment for Domain-specific Languages. In: *Proc. of Models-2009 - ACM/IEEE 12th Int. Conference on Model Driven Engineering Languages and Systems*. Denver, Colorado, October 4-9, 2009.

Upcoming books

- Uwe Assmann, Jeff Pan, Steffen Staab, Yuting (eds.) *Marrying ontology and software technologies*, book, by Springer, 2011
- F. Parreiras, *TwoUse book*, by Wiley, early 2011