# Ontologies and Software Language Engineering

Fernando Silva Parreiras
Tobias Walter
Dragan Gašević
University of Koblenz-Landau and Athabasca University

# (How) Are ontologies and software lanuages related?

# Topics

- Ontologies
  - Basics, langauges and reasoning services
- Existing efforts
  - Ontologies and software languages
- Ontology-enhanced software language engineering
  - Reasoning on software models
- Conclusion

# Part I

# Ontologies

## - Basics, languages, and reasoning -

# What is an ontology?

- Classic definitions
  (Gruber, 1993), (Guarino, 1994)
  - an explicit,
    formal, and
    declarative specification of
    a shared conceptualization

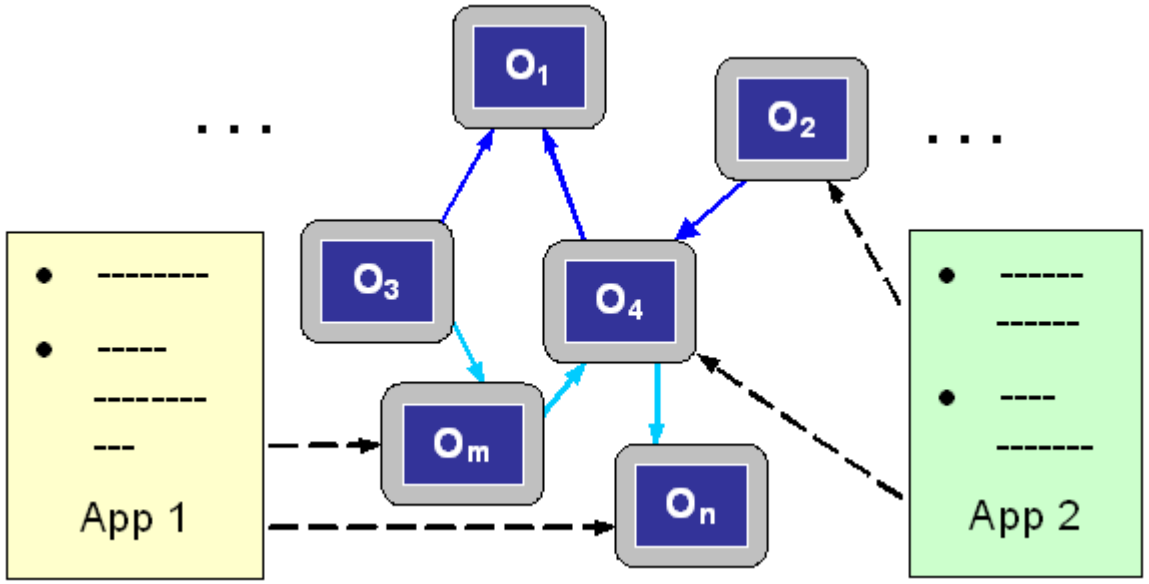
# What is an ontology?

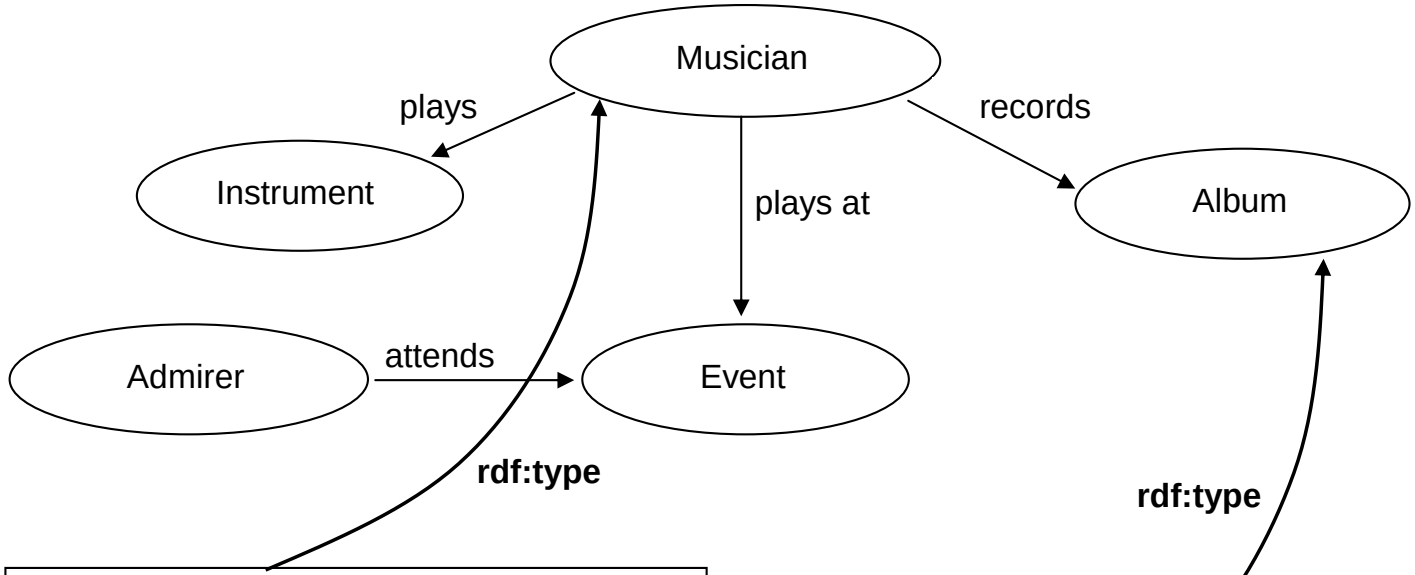
- Important definition (Hendler, 2001)
  - a set of knowledge terms, including
    - vocabulary
    - semantic interconnections
    - some simple rules of inference and logic for some particular topic

# Ontologies for knowledge sharing

# Semantic Web

- Ontologies: Interconnecting applications
  - Shared domain conceptualizations

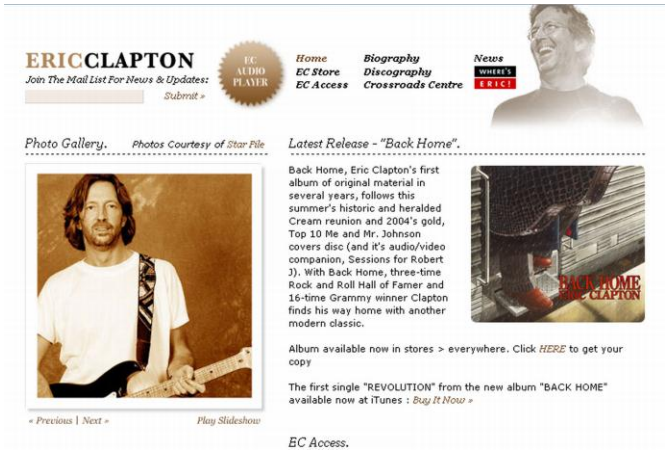





```
< musician:Musician
rdf:ID="urn:rdf:969914d5ca929194ea18787de32c665a-1">
    …
    <musician:name>Eric Clapton</musician:name>
    <musician:records  rdf:resource =
"http://www.guitar.org/legendaryrecordings/EC#urn:rdf:958804d5ca918084ea17676de21c887a-0"/>
    …
</musician:Musician>
```

**musician:records**

```
<album: Album
rdf:ID="urn:rdf:958804d5ca918084ea17676de21c887a-0">
    …
    <album:title>Unplugged</album:title>
    <album:year>1992</album:year>
    …
</album:Album>
```

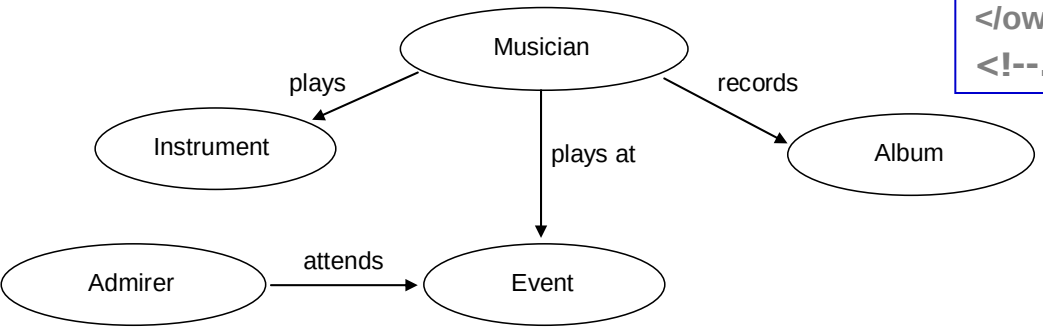



# Web Ontology Language – OWL 2

- Some language features
  - Classes, properties, and individuals
  - Equivalence and disjoints
  - Specific types of restrictions over properties
    - Cardinal, existential and universal
  - Properties
    - Object and data
    - Transitive, (inverse) functional, symmetric

# Web Ontology Language

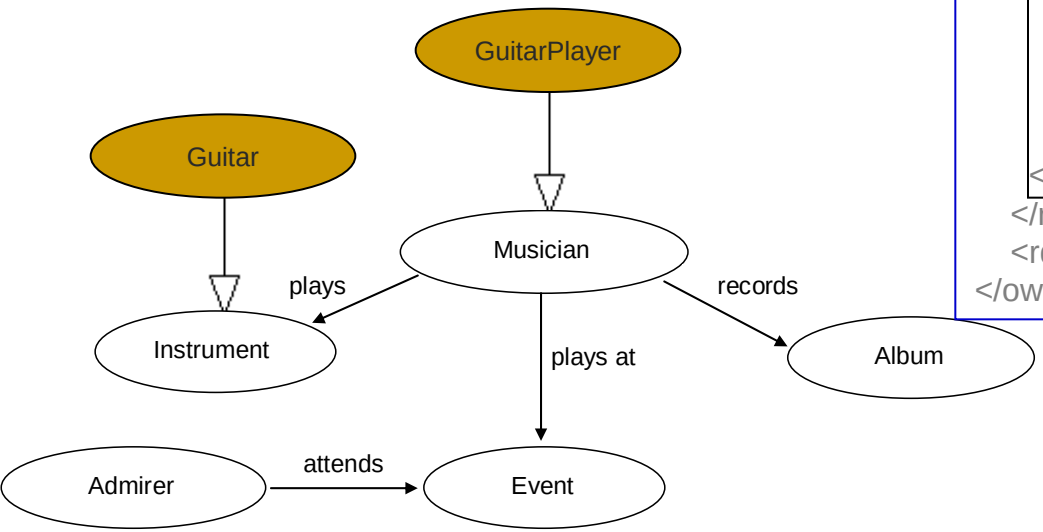
- Musician ontology




```
<owl:Class rdf:ID="Event"/>
<owl:Class rdf:ID="Album"/>
<owl:Class rdf:ID="Instrument"/>
<owl:Class rdf:ID="Musician"/>
<owl:Class rdf:ID="Admirer"/>
<owl:ObjectProperty rdf:ID="plays">
    <rdfs:range rdf:resource="#Musician"/>
    <rdfs:domain rdf:resource="#Instrument"/>
</owl:ObjectProperty>
<!--...-->
```

# Web Ontology Language

- Musician ontology




```
<owl:Class rdf:ID="Guitar“/>

<owl:Class rdf:ID="GuitarPlayer">
   <rdfs:subClassOf>
     <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID=“plays"/>
      </owl:onProperty>
      <owl:allValuesFrom>
        <owl:Class rdf:ID="Guitar"/>
      </owl:allValuesFrom>
     </owl:Restriction>
   </rdfs:subClassOf>
   <rdfs:subClassOf rdf:resource="#Musician"/>
</owl:Class>
```

# Ontology languages enable reasoning!

Not ontologies per se.

# Description Logics

- Designed to represent and reason over structured knowledge
- A domain of interest is structured in (TBox):
  - Concepts
    - correspond to classes (sets of individuals)
  - Roles
    - correspond to associations (binary relations on individuals)
- Knowledge is asserted through so-called assertions (ABox)

# Description Logics

- Provide formal semantics for ontology languages
- Basic reasoning problems
  - Satisfiability
  - Consistency
  - Subsumption
  - Instantiation
  - …

**http://www.cs.man.ac.uk/~horrocks/Slides/Innsbruck-tutorial/pt3-dlreasoning.pdf**

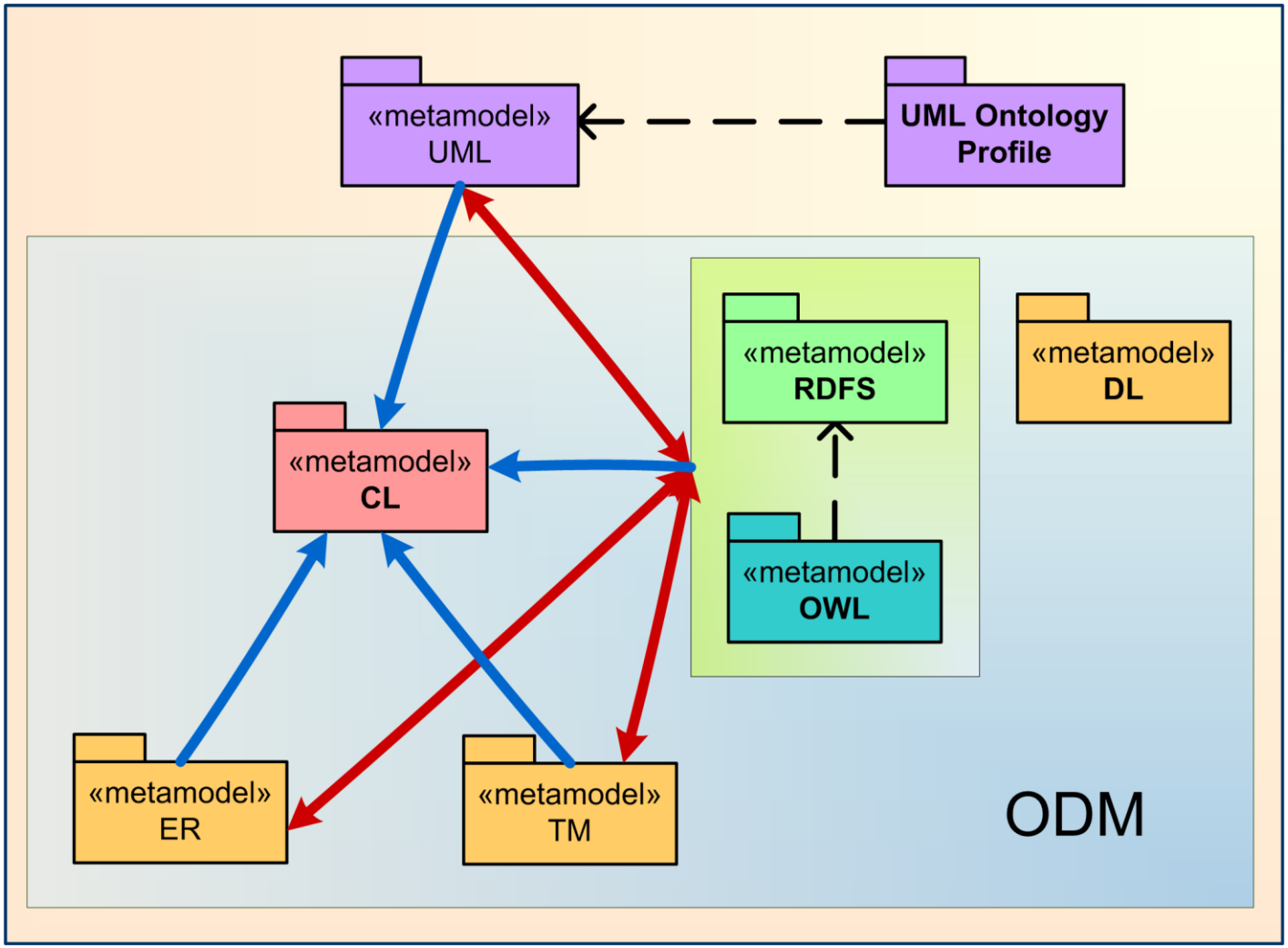
# Part II

# Ontologies and Software Languages

# Ontologies and software languages

- Existing transformations between
  - OWL and UML and MOF (Ecore)
    - IODT
  - OCL and OWL+SWRL
    - with ATL

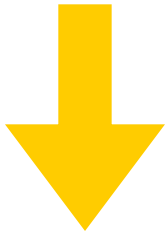
# Ontology Definition Metamodel



# Ontologies and software languages

- Existing transformations between
  - OWL and UML/MOF (Ecore)
  - OCL and OWL+SWRL
- Various languages described with OWL
  - OWL used instead of MOF (Ecore)

# Should OWL and UML/MOF be one language?



# OWL and UML/MOF will be one language

**[Atkinson, 2005]**

# Ontologies and software languages

- Existing transformations between
  - OWL and UML and MOF (Ecore)
  - OCL and OWL+SWRL
- Various languages described with OWL
  - OWL used instead of MOF (Ecore)
- Embedding ontologies in OO languages
  - Zhi#

# Ontologies and software languages

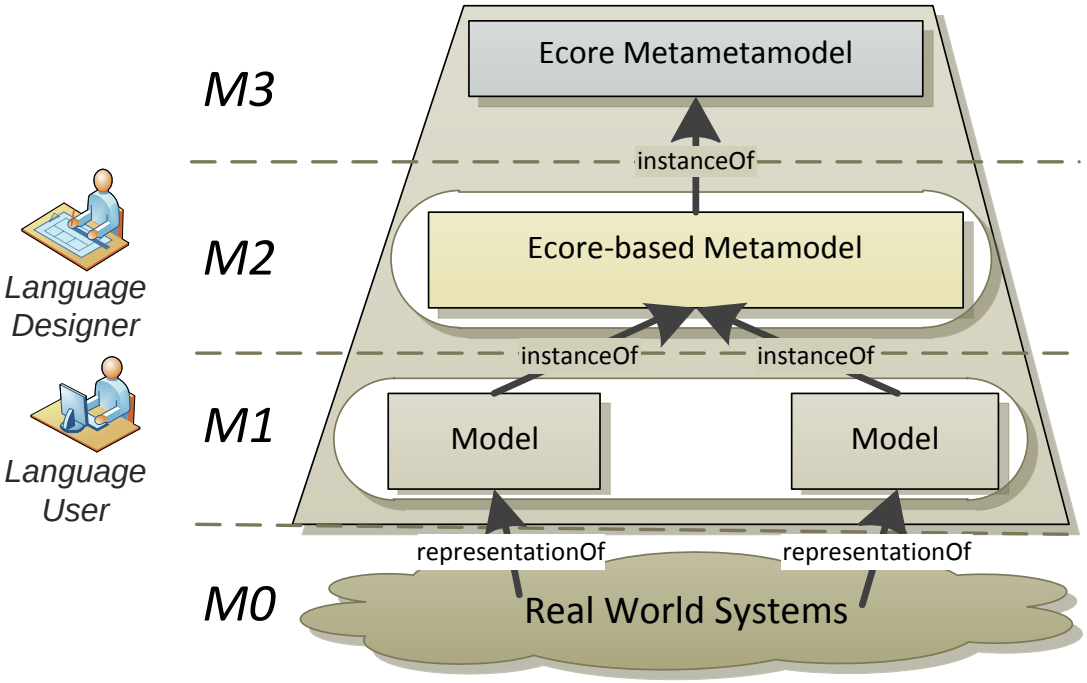
- Automated mapping between languages
  - Inferring mappings among languages
- Effective software knowledge management
  - Explicit traceability among software artifacts

# Part III

# Ontology-enhanced software language engineering

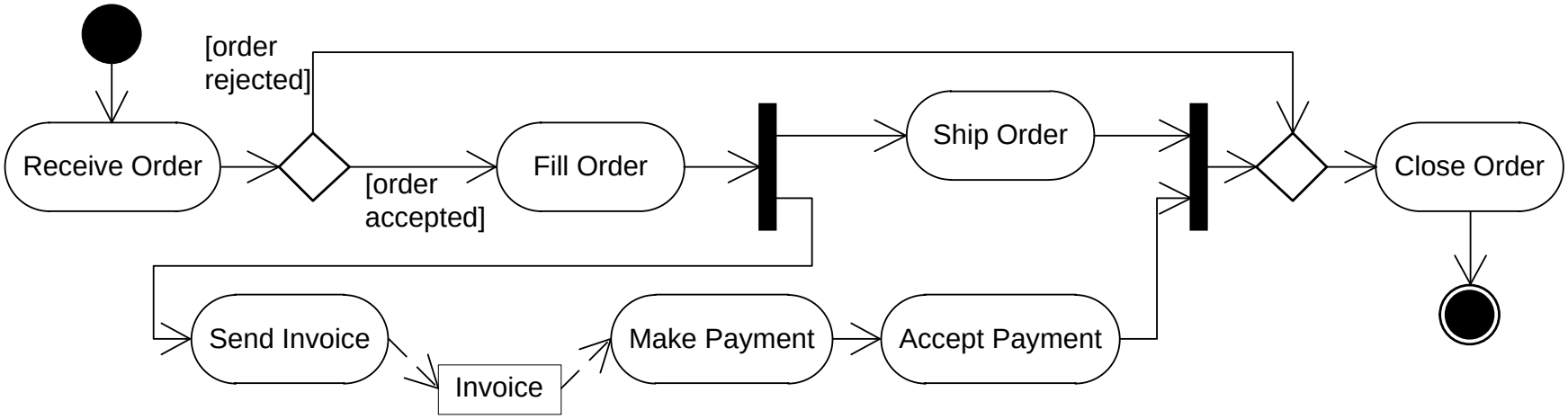
# Ecore Space

- Model hierarchy
  - M1: User models
  - M2: Language metamodels
  - M3: Ecore metamodeling language



# M1 User Model

- M1 user models (e.g process models)
  - designed by language user
  - conforms to an M2 metamodel
  - visualized by different concrete syntaxes



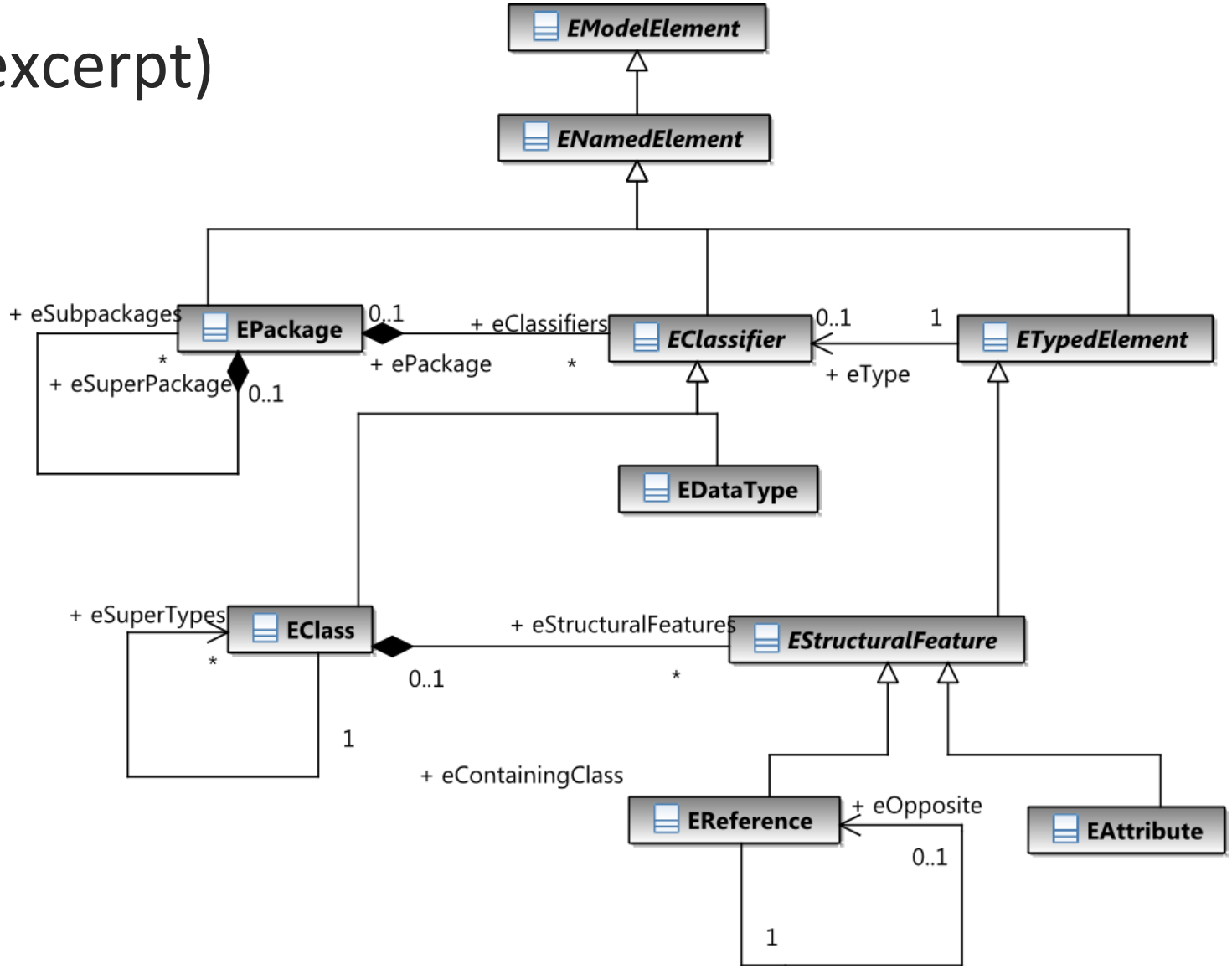
# M2 Metamodel

- conforms to Ecore metametamodel

```
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 {  }
```

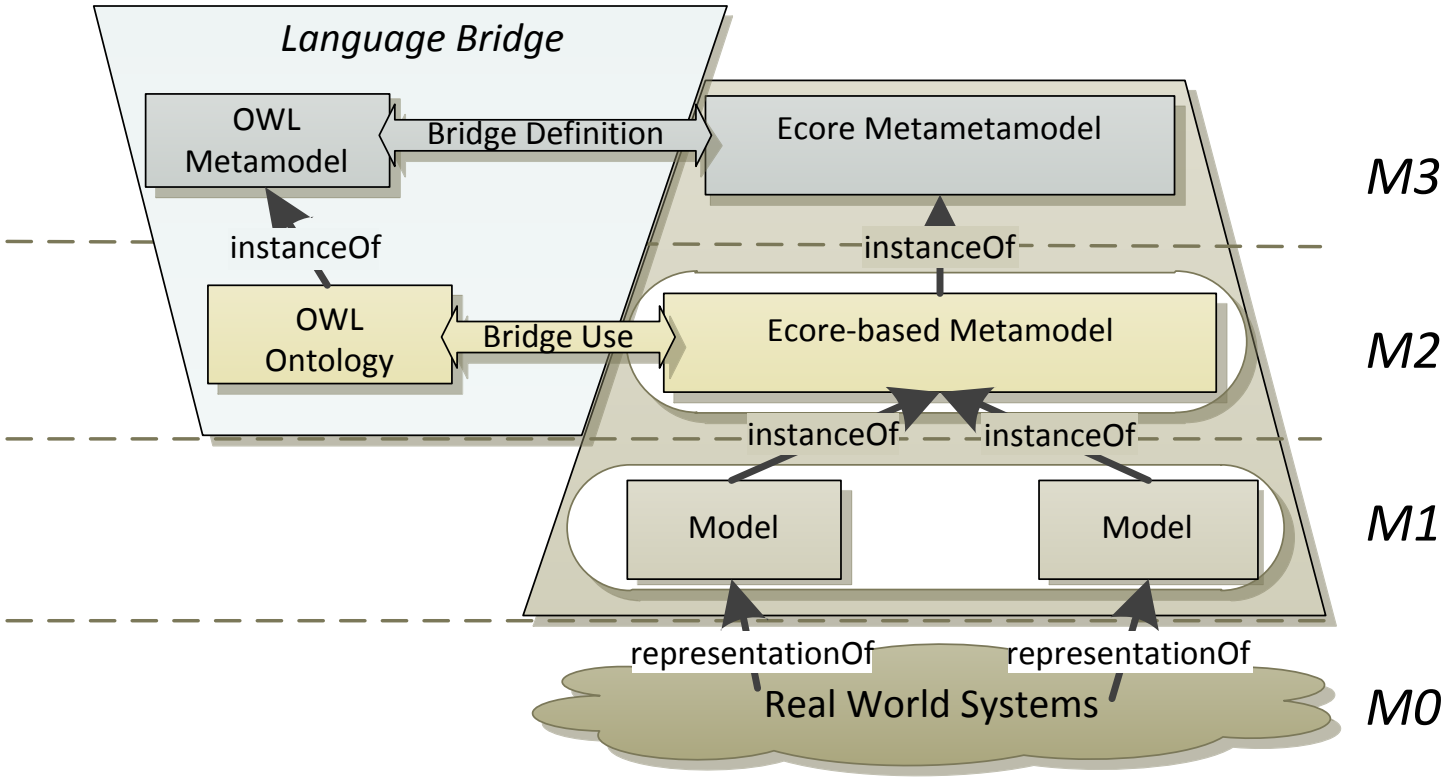
# M3 Metametamodel

- Ecore M3 metametamodel (excerpt)



# Bridging Ecore and OWL

- Integration of Ecore technical space with ontology language OWL2
  - Create Ecore-based metamodels with integrated
    - OWL2 axioms
    - OWL2 expressions



# Bridge Definition

Step 1: Mapping

| Ecore / EMOF | OWL |
|---|---|
| package | ontology |
| class | class |
| suptertype relation | subclass relation |
| reference, attribute | object property, data property |
| data types | data types |
| enumeration | enumeration |
| multiplicity | cardinality |
| opposite reference | inverse object properties |

Step 2: Based on Mapping:

Integrate/Merge concepts of Ecore and OWL (meta-) metamodels

Result: Integrated metametamodel

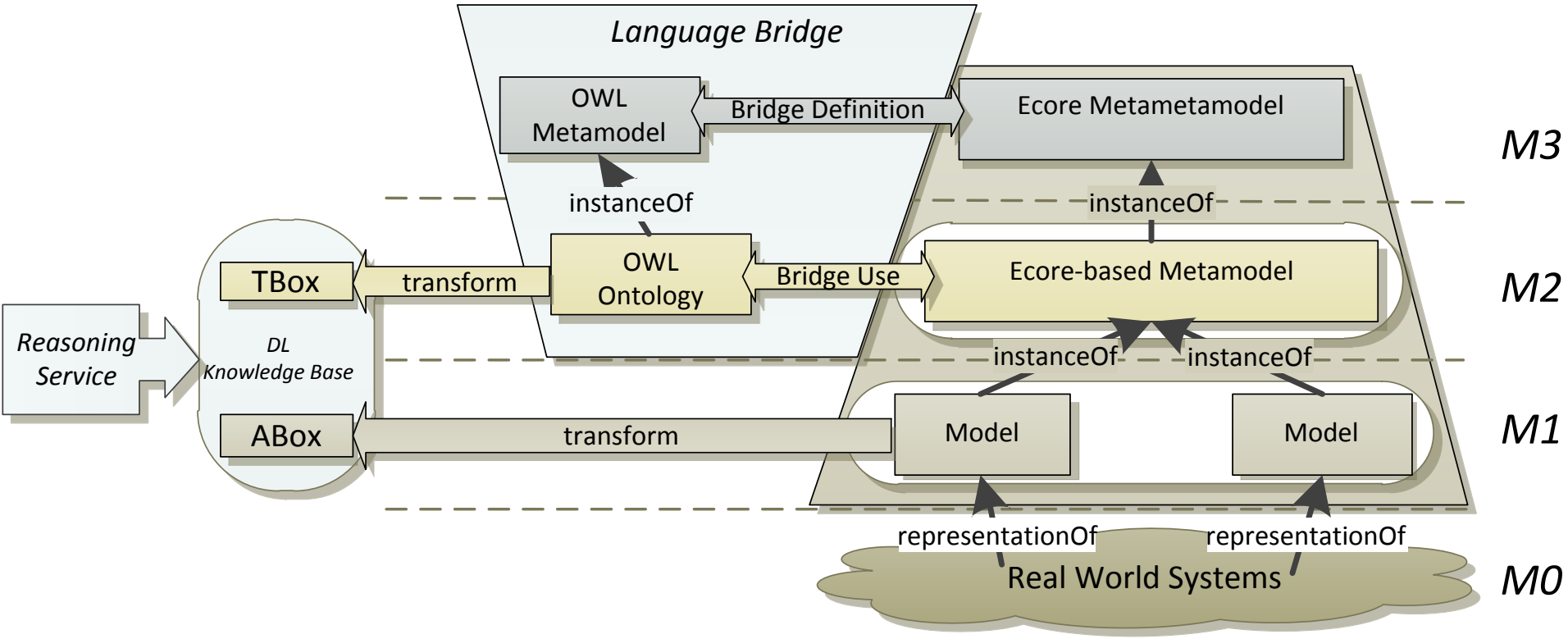
# Metamodel + OWL Annotations

- conforms to integrated metametamodel

```
abstract class ActivityNode {equivalentWith edge some Final {
  reference incoming [0-*] : ActivityEdge oppositeOf target;
  reference outgoing [0-*] : ActivityEdge oppositeOf source;

  transitive reference edge [0-*] : ActivityNode;isChain(outgoing, target);
}

abstract class ActivityEdge {
    reference source [1-1] : ActivityNode;
    reference target [1-1] : ActivityNode;
  }

class Initial extends ControlNode,
              subClassOf outgoing some (to some (Action or ControlNode))

{

}

...
```

# Bridge - Services

- Metamodel and model are transformed to DL knowlegde base (schema-aware transformation)
  - reasoning services



# Satisfiability Checking of Metamodels

- Accomplished Service
  - Finds unsatisfiable concepts in a metamodel

| 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<br>b = a set of unsatisfiable concepts otherwise |

# Satisfiability Checking (Example)

*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

# Consistency Checking of User Models

- Accomplished Service
  - Ensures that a model does not contain any contradictory facts with regard to its language metamodel

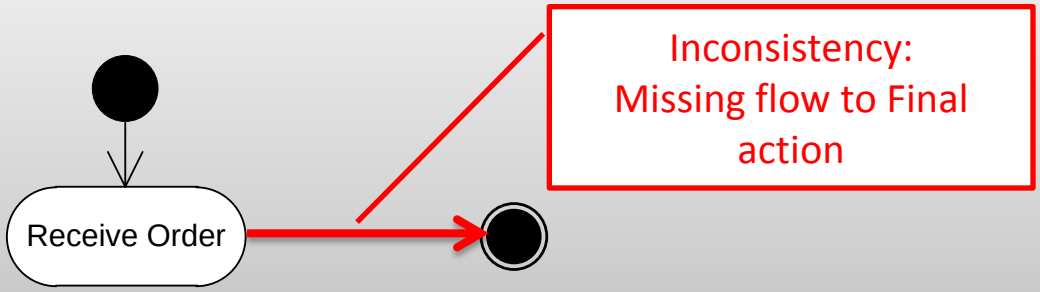
| | |
|---|---|
| *Name* | Consistency Checking |
| *Signature* | boolean consistency (Ontology O) |
| *Description* | Checks if the given ontology O is consistent, i.e. if there exists a model (a model-theoretic instance) for O. If ontology O is consistent, then return true. Otherwise return false.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

# Consistency Checking (Example)

*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*



# Classification of Elements in User Models

- Accomplished Service
  - Determines the most specific type an model element has
  - with respect to all attributes and properties in the context of the model element

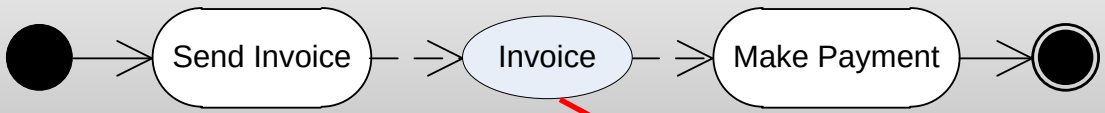
| Name | Classification |
|---|---|
| Signature | boolean classifiesAs (Ontology O, concept A, individual i) |
| Description | Checks if the given individual i is an instance of concept A in the ontology ref, then return true. Otherwise return false. |
| Pattern | b = classifiesAs ( O, A, i) |
| Input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

# Classification (Example)

*M2 Metamodel*

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

{  }
```

*M1 Model*



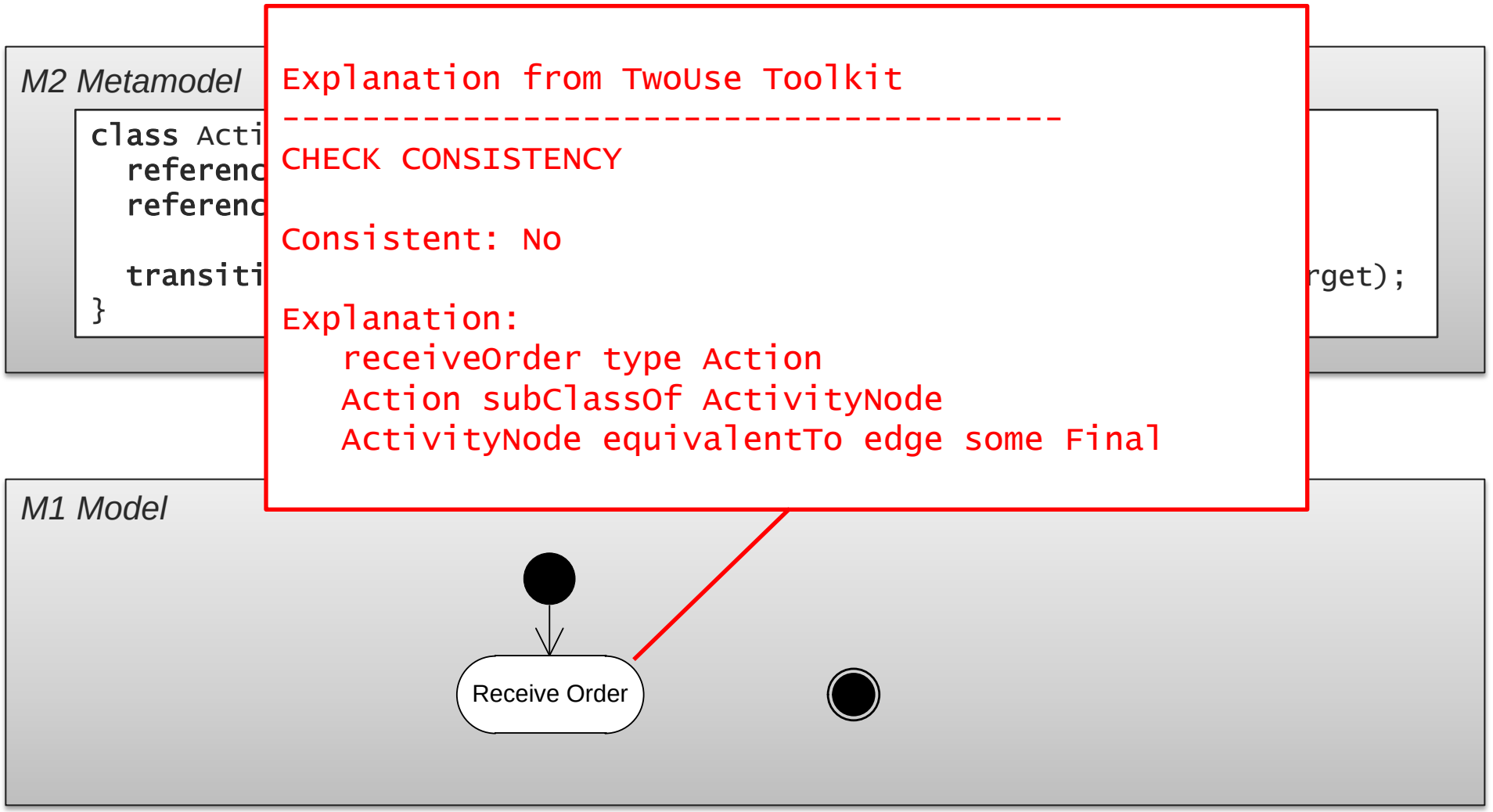

Classify Invoice Node
Result: It is of type
ObjectNode

# Explanations in User Models

- Accomplished Service
  - Explanations for subsumptions and unsatisfiable classes in metamodels
  - Explanations for inconsistencies in models
- Benefits for language users
  - Debugging of models

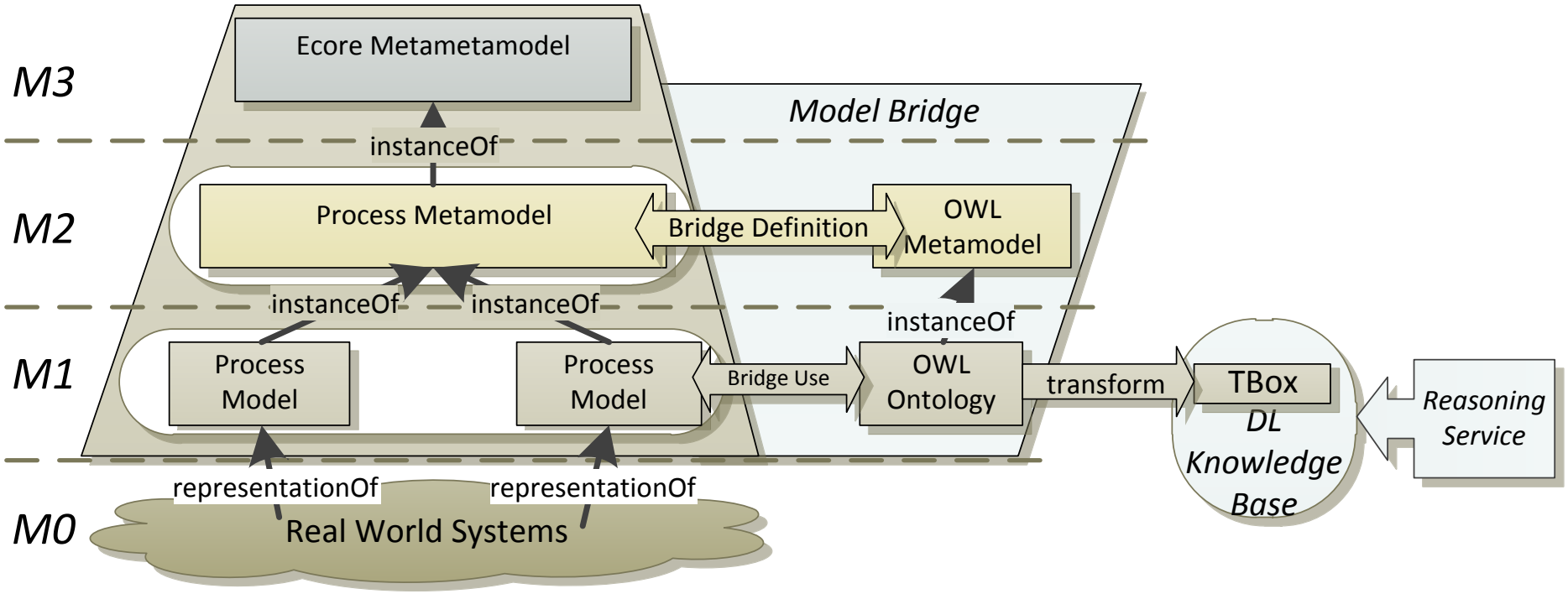
| Name | Explanation |
|---|---|
| Signature | Set<Axiom> getExplanation (Ontology O, axiom Ax) |
| Description | Retrieve the set of axiom that entail axiom Ax in the given ontology, then return them. |
| Pattern | b = getExplanation (O,Ax) |
| Input | An Ontology O and axiom Ax |
| Output | b = set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

# Explanation (Example Inconsistency)



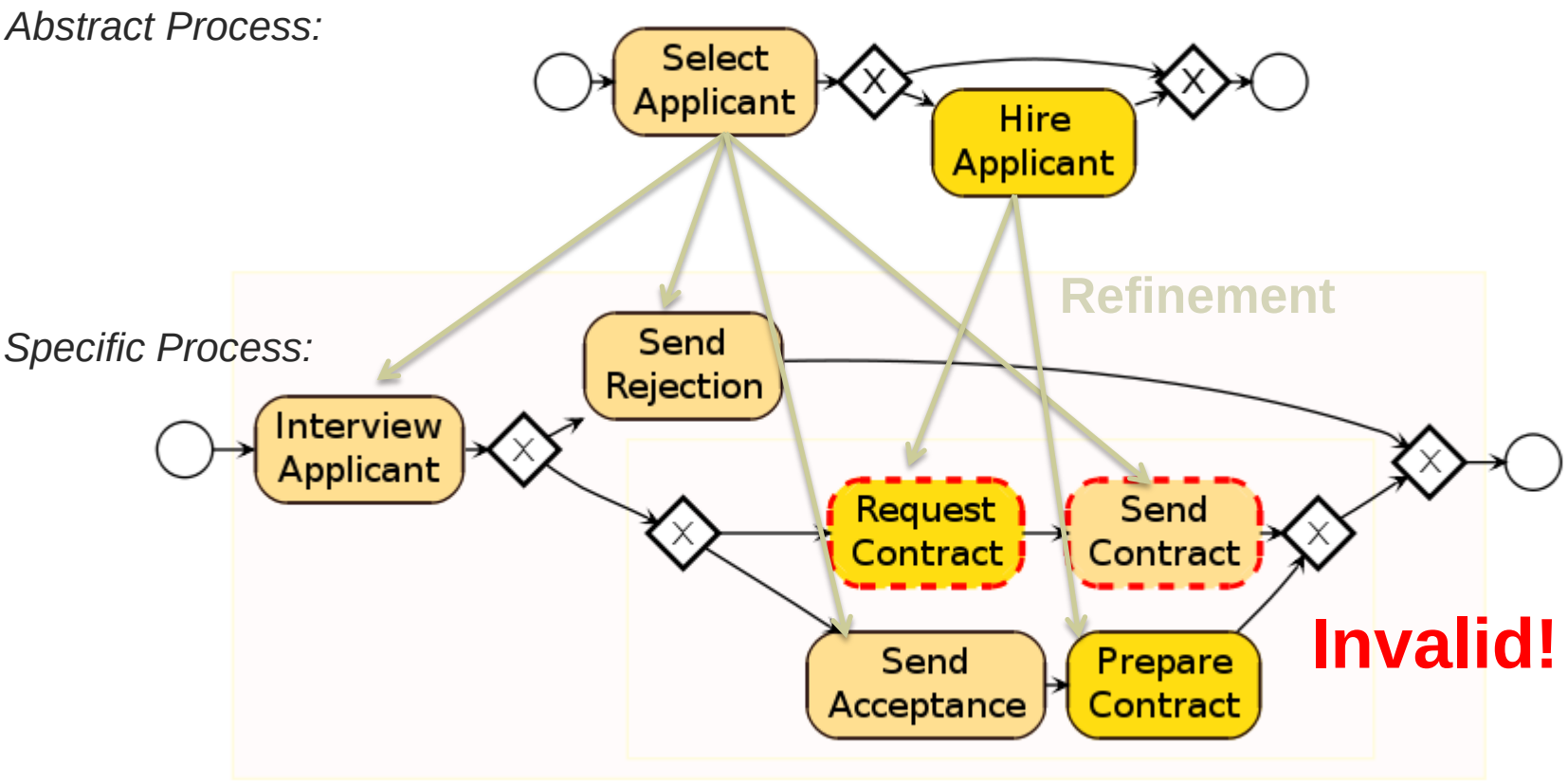
# Model Bridge

- Transforming only user models to DL knowlegde base
  - Services for reasoning on the semantics of the language



# Process Refinement

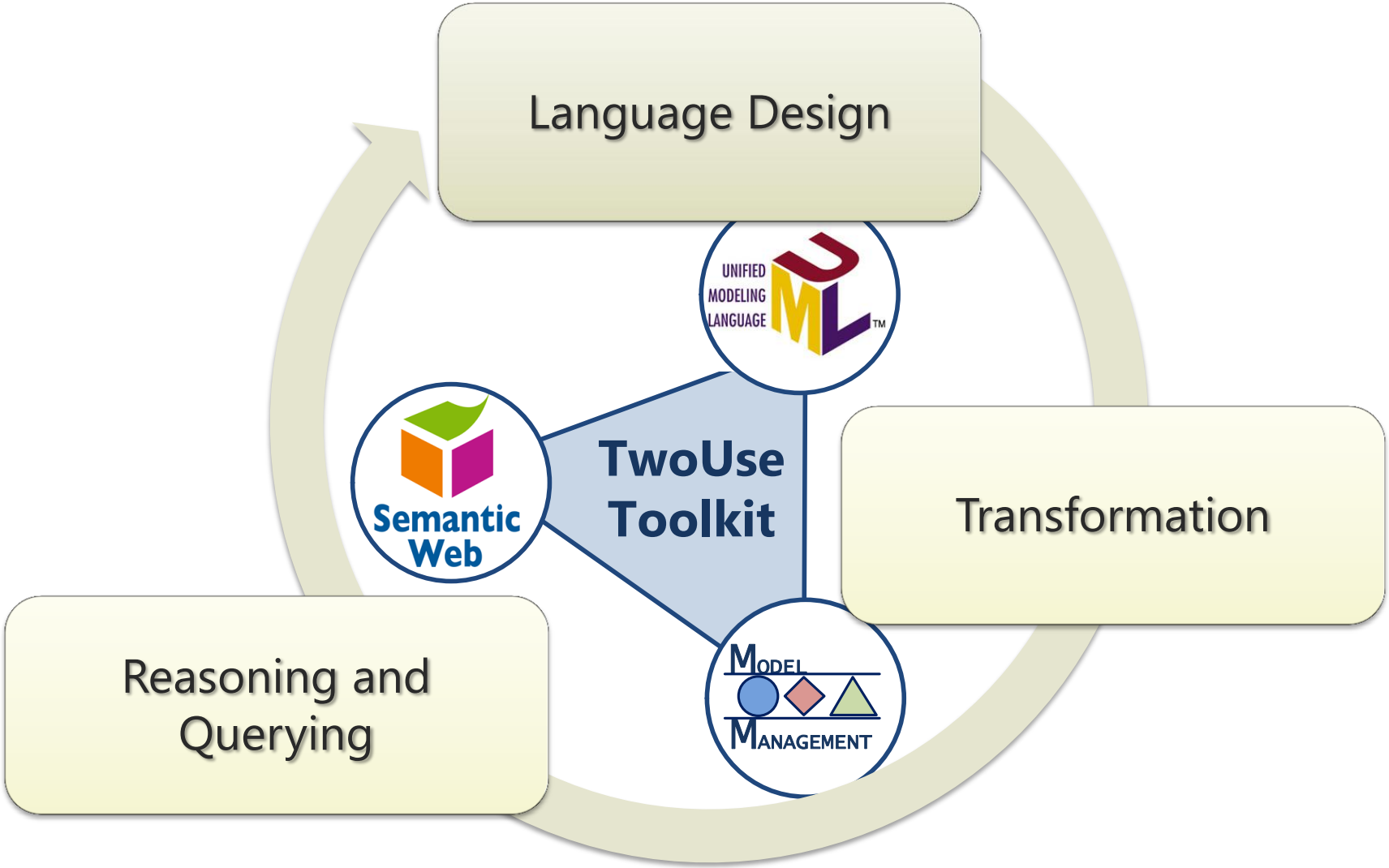
- Formalization of semantics of graph-based modeling languages
- Interpretation and validation of refinement constraints
- Ensuring the specific process preserving the intended meaning of the abstract process

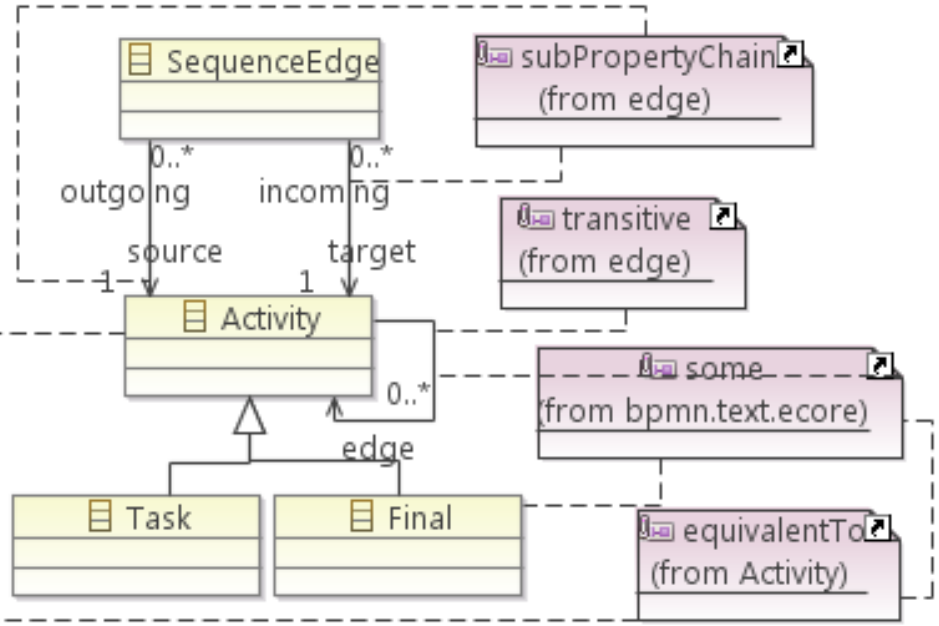


# Part IV

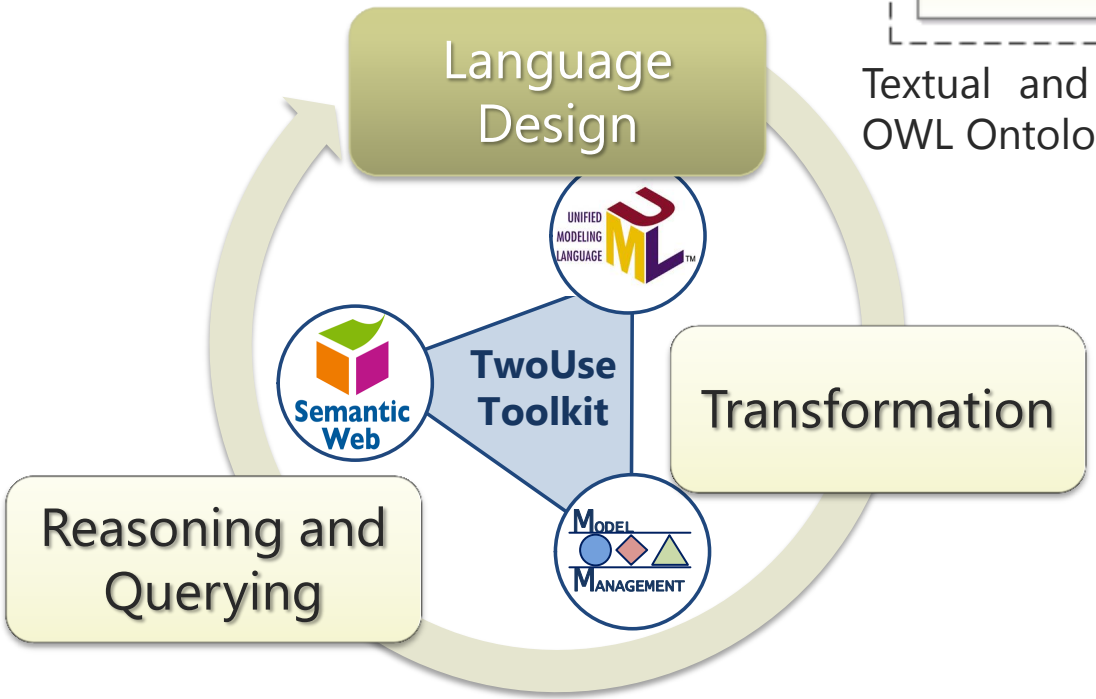
# Demo

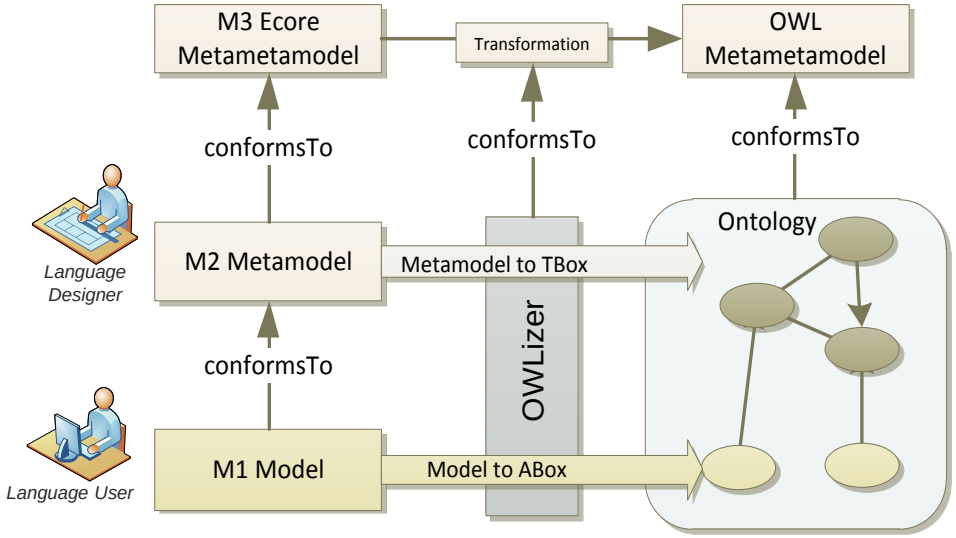
# The TwoUse Toolkit


Language Design
UNIFIED MODELING LANGUAGE
Semantic Web
TwoUse Toolkit
Transformation
Reasoning and Querying
Model Management

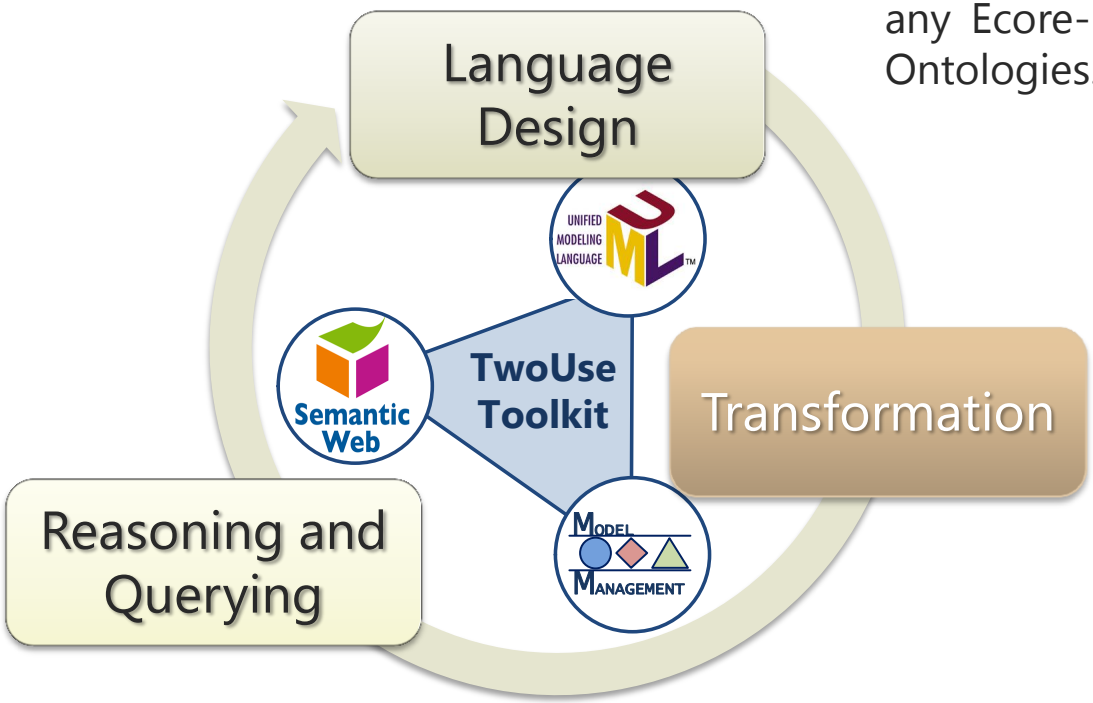


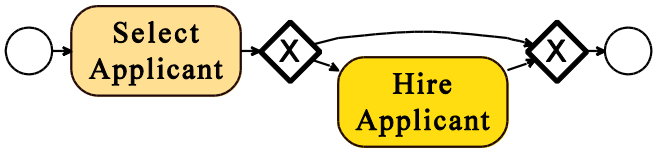

Textual and Graphical Notations for Integrating OWL Ontologies with Ecore and UML.



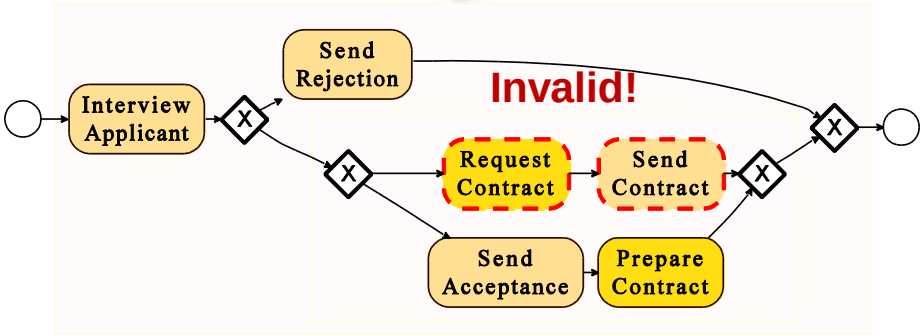



Model Transformations from UML, BPMN and any Ecore-Based Software Language into OWL Ontologies.

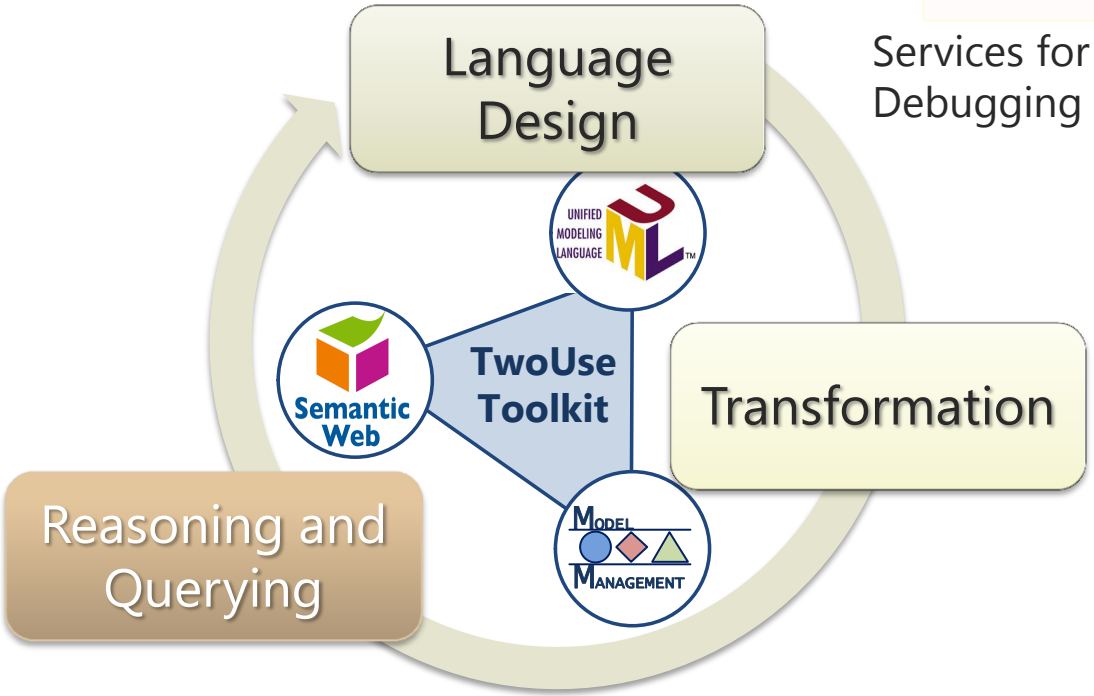





Refinement




Services for Validating, Querying, Integrating and Debugging Software Languages.



# Conclusion

- Ontology languages are well-defined
  - Allowing for reasoning
- Reasoning and ontologies are not magic
- OWL does not meaning using ontologies
- Some early and promising steps
- Many challenges still open

Ontologies and software languages ~~will~~ live happily together!

Thank you!

Questions?