Ontologies and Model-driven Software Engineering

Steffen Staab
Ontologies & Software Tech: Starting Point

IST – Institute for Software Technology
@Koblenz

Fernando Parreiras

Andreas Winter

OWL

Model theory

RDF

Ontologies

Well-founded semantics

F-Logic

Metamodelling

MDA

Semantics

UML

XMI

Grammar

ISWeb - Information Systems & Semantic Web
Steffen Staab staab@uni-koblenz.de
Vienna 2 of 48
Schism?

IST – Institute for Software Technology @Koblenz

Fernando Parreiras

Andreas Winter

ISWeb - Information Systems & Semantic Web

Steffen Staab
staab@uni-koblenz.de

Vienna 3 of 48
Agenda of this Talk

- 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
Agenda of this Talk

- 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“
Purpose: What is an ontology?

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
Usage: What is an ontology?

Ontologies

Thesauri

Extended ER-Models

Predicate Logic

Semantic Networks

Topic Maps

Information Retrieval

Sharing of Knowledge

Navigation

Query Expansion

Query Expansion

Mediation

EAI

Reasoning

Consistency Checking
Formality: What is an ontology?

- Ad-hoc Hierarchies (Yahoo!)
- Text Corpora
- Folks-onomies
- Glossaries
- Data Dictionaries (EDI)
- Thesauri
- Principled Informal Hierarchies
- XML DTDs
- DB Schema
- Data Models (UML, STEP)
- Frames (OKBC)
- F-Logic

**lightweight**
- Glossaries & Data Dictionaries
- Thesauri, Taxonomies
- MetaData, XML Schemas, Data Models

**heavyweight**
- Description Logics
- First-order, Higher-order, Modal Logic
- Formal Ontologies & Inference

Formality: What is an ontology?
Example: 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
Agenda of this Talk

- 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

Classical MDA

PIM

OWL

Translation

UML

PSM

JAVA

Code
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.
Model Checking: Example

- INCONSISTENT
- WebPortalAccount
  - UserAccount
  - Researcher
  - Student

Every WebPortalAccount is used by at most one Researcher.

Researchers is disjoint from Student if Researcher is empty, User and Student will be redundant.

{complete, disjoint}

User

INCONSISTENT
Model Checking: Example

INCONSISTENT

Advantage for SE:
Models with provably higher quality
Agenda of this Talk

- 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
TwoUse Models (excerpt)

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

Diagram:
- M2
  - Classifier
  - DataType
  - Class
  - AnyType
  - OWLClass
  - TUClass
  - InstanceOf
  - CustomerFromEUCountry
  - PuchaseOrder
  - Product

M1
- Classifier
- Product
- PuchaseOrder
- getCharges()
Agenda of this Talk

- 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

Free trade zones

North America: 35%
South America: 5%
Europe: 19%
Hybrid Model: Example OWL

```
<<owlRestriction>>
CountryFromEU
<<owlValue>> {hasValue = eu}
memberOfTradeZone : FreeTradeZone

<<owlRestriction>>
CustomerFromEUCountry
<<owlValue>>
{livesIn} {someValuesFrom = CountryFromEU}
livesIn

<<owlRestriction>>
DutyFreeOrder

DutyFreeOrder = OrderFromEUCustomer

<<owlRestriction>>
OrderFromEUCustomers

OrdersFromEUCustomers hasCustomer SOME CustomerFromEUCountries
```
**Hybrid Model: Example TwoUse**

```
<<oclExpression>>
{context PurchaseOrder::getCharges() : Real
body: if self.isOwlInstanceOf(DutyFreeOrder)
then 0% else 60% endif}
```

```
<<owlRestriction>>
CountryFromEU
<<owlValue>> {hasValue = eu}
memberOfTradeZone : FreeTradeZone
```

```
<<owlRestriction>>
CustomerFromEUCountry
<<owlValue>>
+{someValuesFrom = CountryFromEU}
livesIn
```

```
<<owlRestriction>>
OrderFromEUCustomer
<<owlValue>>
+{someValuesFrom = CustomerFromACountry}
hasCustomer
```

```
<<owlRestriction>>
DutyFreeOrder
<<owlValue>>
+{someValuesFrom = CustomerFromACountry}
hasCustomer
```

```
<<owlRestriction>>
Product
Name: String
```

```
<<equivalentClass>>
OrderFromEUCustomer
```

```
<<equivalentClass>>
OrderFromEUCustomer
```

```
FreeTradeZone
```

```
Customer
```

```
Purchase
Order
getCharges()
```

```
Product
Name: String
```

```
TwoUse
```

```
Country
```

```
<<owlValue>>
<<ocldlExpression>>
{context PurchaseOrder::getCharges() : Real
body: if self.isOwlInstanceOf(DutyFreeOrder)
then 0% else 60% endif}
```

```
<<owlRestriction>>
CountryFromEU
<<owlValue>> {hasValue = eu}
memberOfTradeZone : FreeTradeZone
```

```
<<owlRestriction>>
CustomerFromEUCountry
<<owlValue>>
+{someValuesFrom = CountryFromEU}
livesIn
```

```
<<owlRestriction>>
DutyFreeOrder
<<owlValue>>
+{someValuesFrom = CustomerFromACountry}
hasCustomer
```

```
<<owlRestriction>>
OrderFromEUCustomer
<<owlValue>>
+{someValuesFrom = CustomerFromACountry}
hasCustomer
```

```
<<owlRestriction>>
Product
Name: String
```

```
<<equivalentClass>>
OrderFromEUCustomer
```

```
FreeTradeZone
```

```
Customer
```

```
Purchase
Order
getCharges()
```
Co-generation of OWL models from UML

Generation & OCL-DL

UML

PIM

UML

PSM

JAVA

Code

Classical MDA

Java calls to reasoner

OWL
OCL-DL can do more!

OCL-DL Queries -> OWL

PIM -> UML -> PSM -> JAVA

Classical MDA
OCL-DL Example

TwoUse

UML

OWL

self.owlIsInstanceOf(DutyFreeOrder)  Type (?self, DutyFreeOrder)
self.owlAllTypes()  Type (?self, ?x)
self.owlAllInstances()  Type (?self, ?x), Type (?y, ?x)
Summary of OCL-DL

- “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
Agenda of this Talk

- 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
Use Case 2: Using MDE for Translating between Ontologies

Translation of ontology datasets

Input

- Date
  - year : gYear
  - day : gDay
  - month : gMonth
- PageRange
  - endPage : Integer
  - startPage : Integer
- Part
  - title : String

Output

- Publication
  - title : String
- InBook
  - month : String
  - pages : Integer
- Chapter

Translation

MBOTL
State of the Art: Neon Toolkit

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?
The MBOTL Solution

Layers

Semantic

Syntactic

Lexical

Input → Output

Transformation

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))
    to
        t : _bibtex!Inbook {
            title ← s.title.toUpperCase(),
            pages ← s.pages.endPage - s.pages.startPage,
            month ← s.date.month.notNullShortened(),
        }
}
```
Addressing the syntax and lexical layers

Application of predefined operations or helpers

```java
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))
   to
      t : _bibtex!Inbook (title <- s.title.toUpperCase(),
                          pages <- s.pages.endPage - s.pages.startPage,
                          month <- s.date.month notShortened(),
                           }
```

Unified Representation
Metamodel

Reference Layer
(EU STReP MOST)

MBOTL Extension
Translation Process

Platform Independent Model

ATL Model Transformation

Platform Specific Model

ATL Model Transformation

Code

Jena Framework

Source Ontology

Target Ontology

mbotl.ecore

java.ecore

sparql.ecore

query.builtins.java
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.toUpperCase(),
      pages <- s.pages.endPage - s.pages.startPage,
      month <- s.date.month.notShortened();
    }
}
PREFIX ont: <http://www.testontology.com/>
PREFIX ont2: <http://www.testontology2.com/>
PREFIX userdf: <java:propertyfunction/>

CONSTRUCT

simpleSPARQL

Metamodel

simpleSPARQL.ann
simpleSPARQL.ecore
simpleSPARQL.km3

Samples

queries.sparql.xmi
queries.sparql.xmi.sparql
Transformation2.sparql

Syntax

simpleSPARQL_ANTLR3.g
simpleSPARQL.tcs
simpleSPARQL-parser.jar

TRANSFORM

kSubject ?ChapterInBook2InBookPredicate ont2:Inbook.
kSubject ont2:title ?ChapterInBook2InBook_title.
kSubject ont2:month ?ChapterInBook2InBook_notShortened.
kSubject ont2:pages ?ChapterInBook2InBook_endPage

kSubject ?ChapterInBook2InBookPredicate ont:Part.
kSubject ont:date ?ChapterInBook2InBook_date.
k_date ont:month ?ChapterInBook2InBook_month.
k_size userdf:notShortened ?ChapterInBook2InBook_month.
kSubject ont:pages ?ChapterInBook2InBook_pages.
k_pages ont:endPage ?ChapterInBook2InBook_endPage
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) {
            // Your code here
            String queryString += line + "\n";
        }
    }
    finally {
        in.close();
    }
}
/** 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;
}
Summary of Case 2

**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
Conclusion

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


Expressiveness of OCL, UML and DLs

- FOL
  - n-ary relations, functional dependencies on n-ary relations, and identification constraints on concepts
  - DLRifd (UML Class Diagrams)
  - limited complex role inclusion, reflexivity and irreflexivity, role disjointness
  - DL-Lite
    - role hierarchy
    - qualified cardinality restrictions
  - SHOIN(D) (OWL-DL)
    - nominals
  - SROIQ(D) (OWL 2)
    - transitive properties

- OCL
  - ALCQI
    - role inclusion, reflexivity, qualified cardinality restrictions
OCL-DL Dialect

- n-ary relations, functional dependencies on n-ary relations, and identification constraints on concepts
- limited complex role inclusion, reflexivity and irreflexivity, role disjointness
- n-ary relations, functional dependencies on n-ary relations, and identification constraints on concepts
- DLRifd (UML Class Diagrams)
- ALCQI
- DL-Lite
- role hierarchy
- cardinality restrictions
- SHOIN(D) (OWL-DL)
- nominals
- transitive properties
- SROIQ(D) (OWL 2)
- FOL
- OCL-DL
- OCL
- FOL
- OCL-DL
- OCL

OCLOCL-DL