Joint Metamodels for UML and OWL

Steffen Staab
Schism?

OWL
Model theory
RDF
Ontologies
Well-founded semantics
F-Logic

IST – Institute for Software Technology @Koblenz

Fernando Parreiras

Metamodelling
MDA
UML
Grammar
XMI

Semantics

Andreas Winter

ISWeb - Information Systems & Semantic Web
Steffen Staab
staab@uni-koblenz.de
SWSE 2008@ISWC 2008
3 of 48
Synthesis!

<isweb>

@Koblenz

IST – Institute for Software Technology @Koblenz

Fernando Parreiras

It’s the process!

Andreas Winter

OWL

Model theory

RDF

Ontology

Well-founded semantics

F-Logic

MDA

Metamodelling

UML

Grammar

RDF Semantics

It’s the process!

ISWeb - Information Systems & Semantic Web

Steffen Staab
staab@uni-koblenz.de

SWSE 2008@ISWC 2008

4 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
Model-driven Architecture

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
of a shared
conceptualization
of a domain of interest
⇒ Executable, Discussable
⇒ Group of persons
⇒ About concepts
⇒ 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?

Front-End

- Thesauri
- Topic Maps
- Navigation
- Information Retrieval
- Sharing of Knowledge
- Query Expansion

Back-End

- Ontologies
- Semantic Networks
- EAI
- Mediation
- Reasoning
- Consistency Checking
- Sharing of Knowledge

- Extended ER-Models
- Queries
- Predicate Logic

- Taxonomies

Information systems & semantic web

Steffen Staab
staab@uni-koblenz.de

SWESE 2008@ISWC 2008
Formality: What is an ontology?

Ad-hoc Hierarchies (Yahoo!)

Text Corpora

Glossaries

Data Dictionaries (EDI)

Folks-onomies

Thesauri

Principled Informal Hierarchies

Glossaries & Data Dictionaries

Thesauri, Taxonomies

XML DTDs

DB Schema

Formal Taxonomies

XML Schema

Description Logics

Formal Ontologies & Inference

Data Models (UML, STEP)

Frames (OKBC)

First-order, Higher-order, Modal Logic

F-Logic

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
MDA & Ontology Case: Model Checking

Reasoning on UML class diagrams

Translation

OWL

UML

PIM

Classical MDA

PIM → PSM → JAVA

Code

JAVA

UML

Classical MDA

PIM

Translation

OWL

UML

JAVA
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

Every WebPortalAccount is used by at most one Researcher.

If Researcher is empty, User and Student will be redundant.

Researcher is disjoint from Student.

Every WebPortalAccount is used by at most one Researcher.

UserAccount

Uses

WebPortalAccount

Researcher

Student

{complete, disjoint}
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
context PurchaseOrder::getCharges() : Real

body: if self.owlIsInstanceOf(DutyFreeOrder)
then 0.00 else 0.60 endif}
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 UML

- **FreeTradeZone**
- **Country**
- **Customer**
- **Order**
- **Product**

**UML Diagram:***

- `+hasMember` relationship from `Country` to `Customer`.
- `+livesIn` relationship from `Country` to `Customer`.
- `+hasMemberOfTradeZone` relationship from `FreeTradeZone` to `Customer`.
- `+hasOrder` relationship from `Customer` to `Order`.
- `+hasCustomer` relationship from `Order` to `Customer`.
- `+hasProduct` relationship from `Order` to `Product`.
- `+hasResident` relationship from `Country` to `Order`.
- `getCharges()` method in `Order` class.

**Classes:**
- `FreeTradeZone`
- `Country`
- `Customer`
- `Order`
- `Product`
Hybrid Model: Example OWL

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

<<owlRestriction>>
CustomerFromEUCountry

<<owlRestriction>>
DutyFreeOrder
DutyFreeOrder = OrderFromEUCustomer

<<owlRestriction>>
OrderFromEUCustomers

OrdersFromEUCustomers hasCustomer SOME CustomerFromEUCountries
```

```
+hasResident
+memberOfTradeZone
+hasMember
+hasOrder
+hasCustomer
```

```
FreeTradeZone
Country
Customer
Purchase
Order
getCharges()
```

```
1..n
1
0..n
1
0..n
1
0..n
1
```
Hybrid Model: Example TwoUse

TwoUse

FreeTradeZone

Country

Customer

Purchase Order

getCharges()

Product

Name: String

OWL

DutyFreeOrder

<<owlRestriction>>
OrderFromEUCustomer

<<owlRestriction>>
CountryFromEU

<<owlRestriction>>
CustomerFromEUCountry

<<owlValue>>
{hasValue = eu}
memberOfTradeZone : FreeTradeZone

<<owlValue>>
{someValuesFrom = CountryFromEU}
livesIn

<<owlRestriction>>

<<owlValue>>
{someValuesFrom = CustomerFromACountry}
hasCustomer

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

Co-generation of OWL models from UML

Generation & OCL-DL

Java calls to reasoner

UML

PIM

UML

PSM

JAVA

Code

Classical MDA
OCL-DL can do more!

OCL-DL Queries → OWL

PIM → UML → PSM → JAVA

Classical MDA
OCL-DL Example

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

Part
- title : String

InBook

Chapter

Translation

Output

PageRange
- endPage : Integer
- startPage : Integer + pages

Publication
- title : String

InBook
- month : String
- pages : Integer

MBOTL

Translation of ontology datasets
State of the Art: Neon Toolkit

Input

Output

visual mapping of ontologies

Plug-ins must be written separately
Problem

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**
- InBook
- Chapter
- gMonth
- title

**Output**
- Inbook
- String
- upperCase(title)

**Transformation**
- mbotl: PIM
- f-logic
- java
- or
- sparql
- jena

**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)
Addressing the Semantic Layer

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.toUpper(),
    pages <- s.pages.endPage - s.pages.startPage,
    month <- s.date.month.notShortened(),
  }
}
```
Addressing the syntax and lexical layers

Application of predefined operations or helpers

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

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

ATL Model Transformation

Platform Independent Model

Platform Specific Model

Code

Jena Framework

Target Ontology

Source Ontology
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()}
}
Screenshot: Generated SPARQL Model

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

CONSTRUCT {  
  ?subject ont2:month ?ChapterInBook2InBook_notShortened.
  ?hasSize userdef:notShortened ?ChapterInBook2InBook_month.
}
```
```java
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)
```
/** 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](http://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
Thank You!

http://isweb.uni-koblenz.de/Projects/twouse
http://isweb.uni-koblenz.de/Projects/MOST


Expressiveness of OCL, UML and DLs

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

ISWeb - Information Systems & Semantic Web
Steffen Staab
staab@uni-koblenz.de
SWESE 2008@ISWC 2008