ulm university universität ulm

Semantic Web Grundlagen

Ontology Editing

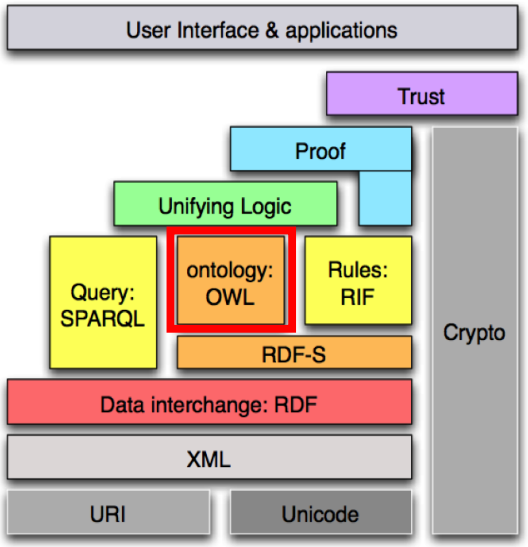
Birte Glimm
Institut für Künstliche Intelligenz | 23. Jan 2012

Organisatorisches: Inhalt

Einleitung und XML	17. Okt	Hypertableau II	12. Dez
Einführung in RDF	20. Okt	Übung 4	15. Dez
RDF Schema	24. Okt	SPARQL Syntax & Intuition	19. Dez
fällt aus	27. Okt	SPARQL Semantik	22. Dez
Logik – Grundlagen	31. Okt	SPARQL 1.1	9. Jan
Übung 1	3. Nov	Übung 5	12. Jan
Semantik von RDF(S)	7. Nov	SPARQL Entailment	16. Jan
RDF(S) & Datalog Regeln	10. Nov	SPARQL Implemetierung	19. Jan
OWL Syntax & Intuition	14. Nov	Ontology Editing	23. Jan
Übung 2	17. Nov	Übung 6	26. Jan
OWL & BLs	21. Nov	Ontology Engineering	30. Jan
OWL 2	24. Nov	Linked Data	2. Feb
Tableau	28. Nov	SemWeb Anwendungen	6. Feb
Übung 3	1. Dez	Übung 7	9. Feb
Blocking & Unravelling	5. Dez	Wiederholung	13. Feb
Hypertableau	8. Dez	Übung 8	16. Feb

Abfragen und RIF wurde gestrichen

Ontology Engineering



Agenda

- Wie können wir programmatisch mit einer Ontologie arbeiten?
- Wie können wir über eine grafische Oberfläche mit einer Ontologie arbeiten?

OWL Implementieren

Was heisst es “OWL zu implementieren”?

- Modellierung – Bereitstellung von Datenstrukturen, die eine OWL Ontologie repäsentieren
- Parsen – Konvertierung der textuellen Repräsentation einer Ontologie (z.B. in RDF/XML) in entsprechenden Datenstrukturen
- Serialisierung – Erzeugung einer entsprechenden textuellen Darstellung (z.B. in RDF/XML) einer Ontologie in den vorliegenden (internen) Datenstrukturen
- Manipulation – Bereitstellung von Methoden zum Manipulieren der Datenstrukturen bzw. zum Erzeugen neuer ontologischer Elemente
- Schlussfolgerungen – Berücksichtigung der formalen Semantik von OWL

Aspekte einer Implementierung

- Identifizierung der Funktionalitäten und Zuständigkeiten
- Repräsentation
  - Syntax vs. Datenmodell
  - Interface vs. Implementierung
  - Lokalität von Informationen
- Parsing/Serialization
  - Abstraktion der zugrunde liegenden konkreten Repräsentation, z.B. als Tripel
- Manipulation
  - Granularität
  - Abhängigkeiten
  - Benutzerintention
  - Strategien
- Schlussfolgerungen
  - Unterscheidung von expliziten und impliziten Konsequenzen
  - Externe Implementierungen

Was ist eine Ontology (für den Implementierer)

- Eine bestimmte syntaktische Repräsentation?
- Die Fakten, die durch die syntaktische Repräsentation repräsentiert werden?
- Die Informationen, die aus den repräsentierten Fakten folgen?

- Derartige Fragen werden wichtig im Zusammenhang mit OWL
- Welche Antworten soll ich erwarten, durch Abfragen an die Strukturen?

Das OWL API

- Das OWL API ist eine derartige OWL Implementierung
- Gedacht für OWL DL
- Eine Ontology ist repräsentiert durch eine Menge von Axiomen, welche Informationen über Klassen, Rollen und Individuen modellieren
- Wann ist eine Klasse oder Rolle “in” einer Ontologie?
  - Nicht in der OWL Spezifikation definiert
  - Etwas Abhängig von der Implementierung

Datenstrukturen des OWL API

- Stellt Datenstrukturen zur Repräsentation von Ontologien zur Verfügung
- Hilfsklassen für
  - die Erzeugung,
  - die Manipulation,
  - das Parsen,
  - das Rendern/Darstellen und
  - das Schlussfolgern über diese Strukturen
- Die grundlegenden Datenstrukturen repräsentieren die Objekte in einer Ontology und orientieren sich an der funktionalen Syntax von OWL

Das Ontologie Objekt

- Ein `OWLOntology` Objekt repräsentiert eine Ontologie
- Enthält eine Menge von Axiomen
- Weiss welche Klassen und Rollen in der Ontologie verwendet werden
- Kann eine IRI als Namen/ID haben und kann von IRIs geladen werden
  - Die ID der Ontologie muss nicht übereinstimmen mit der IRI von der die Ontologie geladen wird
  - Gleiches gilt für importierte Ontologien

Laden einer Ontologie

```
OWLOntologyManager manager
    =OWLManager.createOWLOntologyManager();
OWLOntologyIRIMapper aMapper
    =new AutoIRIMapper(new File("/ont/"), false);
manager.addIRIMapper(aMapper);
IRI rem=IRI.create("http://www.ex.org/ex.owl");
IRI loc=IRI.create("file:/ont/ex.owl");
OWLOntologyIRIMapper sMapper
    =new SimpleIRIMapper(rem, loc);
manager.addIRIMapper(sMapper);
OWLOntology ont
    =m.loadOntologyFromOntologyDocument(
        IRI.create("file:/ont/ont.owl"));
```


Objekt für Entitäten

- Die Klasse `OWLClass` repräsentiert eine (OWL) Klasse
- Enthält keine Informationen darüber in welchen Axiomen die Klasse verwendet wird
- Axiome bzgl. einer Klasse gehören zu einem `OWLOntology` Objekt
- Methoden in `OWLClass` geben Zugriff auf Informationen bzgl. der Klasse im Kontext der verwendeten Ontologie (convenience methods)
- Nicht-atomare Klassen verwenden das Interface `OWLClassExpression`
- Analog für Properties: `OWLObjectProperty` und `OWLDataProperty`

Hinzufügen von Axiomen

```
OWLDataFactory df=m.getOWLDataFactory();
OWLClass student=df.getOWLClass(IRI.create("..."));
OWLClass course=df.getOWLClass(IRI.create("..."));
OWLObjectProperty takes
    =df.getOWLObjectProperty(IRI.create("..."));
OWLClassExpression sup
    =df.getOWLObjectSomeValuesFrom(takes, course);
OWLAxiom ax=df.getOWLSubClassOfAxiom(student, sup);
manager.addAxiom(ont, axiom);
```


Nutzung eines Reasoners

- Die Klasse `OWLReasoner` ist ein Interface, welches die meisten OWL Reasoner implementieren
- Enthält Methoden, um mit dem Reasoner zu interagieren

```
import org.semanticweb.owlapi.reasoner.OWLReasoner;
import org.semanticweb.HermiT.Configuration;
import org.semanticweb.HermiT.Reasoner.ReasonerFactory;
...

OWLReasonerFactory reasonerFactory
    =new Reasoner.ReasonerFactory(); // HermiT factory
OWLReasoner reasoner
    =reasonerFactory.createReasoner(ont);
System.out.println(reasoner.isConsistent());
System.out.println(reasoner.isSatisfiable(student));
hermit.precomputeInferences(
    InferenceType.CLASS_HIERARCHY);
```


Besuchermuster im OWL API

- **Besucher**: Kapsle eine auf den Elementen einer Objektstruktur auszuführende Operation als ein Objekt. Das Besuchermuster ermöglicht es Ihnen, eine neue Operation zu definieren, ohne die Klassen der von ihr bearbeiteten Elemente zu verändern. [Entwurfsmuster, Gamma et al.]
- Erlaubt verschiedenste Operationen über die Datenstrukturen, ohne dass diese mit applikations-spezifischem Kode beladen werden
- Gut solange die Datenstrukturen sich nicht ändern, aber Änderungen teuer
- Änderungen am OWL Standard selten

Beispiel ClassExpressionVisitor

- Besucher implementieren das Interface `OWLClassExpressionVisitor` welches pro OWL Klassen Objekt eine `visit()` Methode erfordert
- Alle Implementierungen des Interfaces `OWLClassExpression` müssen eine Methode `accept(OWLClassExpressionVisitor visitor)` implementieren

```
public interface OWLClassExpressionVisitor {
    void visit(OWLObjectUnionOf ce);
    ...
}
public interface OWLObjectUnionOf extends OWLClassExpression {
    public void accept(OWLClassExpressionVisitor v) {
        visitor.visit(this);
    }
    ...
}
```

Anwendungen für Besucher

Erlauben das Arbeiten mit beliebigen OWL Objekten, zum Beispiel

- Umwandlung in Negationsnormalform von Klassen oder Axiomen
- Vereinfachung von Axiomen
- Serialisierung in verschiedenen Formaten

Übung: Transformation von Konzepten in Negationsnormalform

Agenda

- Wie können wir programmatisch mit einer Ontologie arbeiten?
- Wie können wir über eine grafische Oberfläche mit einer Ontologie arbeiten?

Ontology Editing mit Protege

- Laden Sie mad-cow-start.owl in Protege
- Machen Sie sich mit der Ontologie etwas vertraut

Ontology Editing mit Protege

Gegeben der folgende Text, ergänzen Sie die Ontologie um die Konzepte pet+owner, dog+owner und vegetarian:

Pet owners are persons who own a pet. Dog owners are special pet owners in that they own a dog.

Vegetarians are animals that do not eat other animals and neither do they eat parts of other animals.

Ontology Editing mit Protege

- Klassifizieren Sie die Ontologie
- Sie sollten eine unerfüllbare Klasse mad+cow erhalten