The OWL API:
An Introduction

Sean Bechhofer and Nicolas Matentzoglu
University of Manchester
sean.bechhofer@manchester.ac.uk
OWL allows us to describe a domain in terms of:

- **Individuals**
  - Particular objects in our domain

- **Classes**
  - Collections of objects (usually sharing some common characteristics)

- **Properties**
  - Binary relationships between individuals.

- Plus a collection of axioms describing how these classes, individuals, properties are related
OWL

- OWL has a number of operators that allow us to describe the classes and the characteristics that they have.
- Boolean operators
  - and, or, not
- Quantification over properties/relationships
  - universal, existential.
- A clear and unambiguous semantics for the operators and composite class expressions.
Why build an OWL API?

- The use of a higher level data model can help to
  - insulate us from the vagaries of concrete syntax.
  - make it clear what is happening in terms of functionality.
  - increase the likelyhood of interoperating applications.
Assumptions

- Primarily targeted at **OWL-DL**
  - This does not mean that we cannot handle OWL-Full ontologies, but a number of design decisions reflect this assumption.

- Java based
  - Interfaces
  - Java reference implementation
    - Main memory based
What is an “OWL Implementation”? 

- **Modelling**
  - Provide data structures that represent OWL ontologies/documents.

- **Parsing**
  - Taking some syntactic presentation, e.g. OWL-RDF and converting it to some [useful] internal data structure.

- **Serializing**
  - Producing a syntactic presentation, e.g. OWL-XML from a local data structure.

- **Manipulation/Change**
  - Being able to manipulate the underlying objects.

- **Inference**
  - Providing a representation that implements/understands the formal semantics of the language.
• Provides a definition of the language in terms of the constructs and assertions allowed.

• **Semantics** are then defined in terms of this abstract syntax.

• Our OWL API data model is based largely on this abstract presentation.
  – Conceptually cleaner.
  – Syntax doesn’t get in the way
Considerations

- Clear identification of functionalities and a separation of concerns
- Representation
  - Syntax vs. Data Model
  - Interface vs. Implementation
  - Locality of Information
- Parsing/Serialization
  - Insulation from underlying concrete presentations
  - Insulation from triples
Considerations

- **Manipulation/Change**
  - Granularity
  - Dependency
  - User Intention
  - Strategies

- **Inference**
  - Separation of explicit assertions from inferred consequences
  - External reasoning implementations
Caveats

• Primarily designed to support manipulation of T-Box/schema level ontologies
  – Large amounts of instance data may cause problems.
• Designed to support OWL (not RDF)
• This isn’t industrial production level quality code
  – It’s not bad though :-) 
• We can’t promise to answer all your questions
• We can’t promise to fix all your bugs
• But we’ll try……
Where’s it used?

- Pellet
  - OWL reasoner
- SWOOP
  - OWL editor
- Protégé 4
  - OWL editor
- ComparaGrid
- CLEF
- OntoTrack
  - OWL Editor
- DIP Reasoner
- BT
  - SNOMED-CT support
- BioPAX
  - Lisp bindings (!!!)
Other, Related Work

• Jena
  – Provides OWL Ontology interfaces, layered on the RDF structures of Jena

• Protégé API
  – Protégé 3 provided OWL API layered on Protégé model
  – Mixture of frames, RDF and OWL
  – Evolution to support a UI

• KAON2
  – Support for OWL
  – Not open source
References

- **Matthew Horridge, Sean Bechhofer.** *The OWL API: A Java API for OWL Ontologies.* Semantic Web Journal 2(1), Special Issue on Semantic Web Tools and Systems, pp. 11-21, 2011

- *Cooking the Semantic Web with the OWL API*, ISWC2003
- *Parsing OWL DL: Trees or Triples?*, WWW2004
- *Patching Syntax in OWL Ontologies*, ISWC 2004
- *The Manchester OWL Syntax*, OWLEd 2006
- *Igniting the OWL 1.1 Touch Paper: The OWL API*, OWLEd 2007
- *The OWL API: A Java API for Working with OWL 2 Ontologies*, OWLEd 2009
Programming to the OWL API
What is an Ontology?
OWL API Philosophy

• An Ontology is represented as
  – a collection of axioms
  – that assert information about the classes, properties and individuals

• OWL API provides a uniform view on the ontology
• More or less direct implementation of the OWL 2 specification
• Helpful Resources!
Basic Data Structures

• At its heart, the OWL API provides data structures representing OWL ontologies, like their axioms, classes and relations

• Plus classes to help
  – Create;
  – Manipulate;
  – Parse;
  – Render; and
  – Reason about those structures
Main Building Blocks

- OWLOntology
- OWLOntologyManager
- OWLAXiom
  - SubclassOf
  - EquivalentClasses
  - DisjointClasses
- OWLEntity
  - OWLClass
  - OWLObjectProperty
  - OWLDataProperty
  - OWLIndividual
OWLOntology

Signature contains OWLEntities

Axiom-centric view
Names and URIs

- Ontologies in OWL are named using URIs
- Entities in OWL are identified using URIs

Ontology: <http://owl.cs.manchester.ac.uk/ontologies/sushi.owl>

Class: <http://owl.cs.manchester.ac.uk/ontologies/sushi.owl#Sushi>
• OWLEntity is the fundamental building block of the ontology
  – Classes
  – Properties
  – Individuals
  – Datatypes
• Named using URIs

Class: <http://owl.cs.manchester.ac.uk/ontologies/sushi.owl#Sushi>
• Represents an OWL Class.
• The Class itself is a relatively lightweight object
  – A Class doesn’t hold information about definitions that may apply to it.
• Axioms relating to the class are held by an OWLOntology object
  – E.g. a superclass axiom must be stated within the context of an OWLOntology
  – Thus alternative characterisations/perspectives can be asserted and represented for the same class.
**OWLClass**

- Methods are available on OWLClass that give access to the information within a particular ontology

```java
java.util.Set<OWLDescription> getDisjointClasses(OWLOntology ontology)
java.util.Set<OWLDescription> getEquivalentClasses(OWLOntology ontology)
```

- But these are simply *convenience* methods.
• OWL makes a distinction between
  – Object Properties: those that relate two individuals
    • E.g. hasBrother
  – Data Properties: those that relate an individual to a concrete data value
    • E.g. hasName

• There is a strict separation between the two and two explicit classes representing them
  – OWLObjectProperty
  – OWLDataProperty
• Properties can have associated domains and ranges
• There is also a property hierarchy
  – Super properties
  – Property equivalences
  – Disjoint Properties (OWL2)
• Assertions about properties are made in the context of an Ontology.
  – E.g functional properties


**OWLObjectProperty**

- Represents an Object Property that can be used to relate two individuals

- Object properties can have additional characteristics
  - Transitivity
  - Inverses
• Represents an Data Property that can be used to relate an individual to some concrete data value
• Data properties can also have additional characteristics
  – Functional
Project Setup and Task 1

- Let's get our hands dirty.
The structure of axioms
An ontology contains a collection of OWLAxioms.
Each axiom represents some fact that is explicitly asserted in the ontology.
There are a number of different kinds of axiom:
- Annotation Axioms
- Declaration Axioms
- Import Axioms
- Logical Axioms
Logical Axioms

• The subclasses of OWLLogicalAxiom represent the logical assertions contained in the ontology
  – Supers (of classes and properties)
  – Equivalences (of classes and properties)
  – Property Characteristics
    • Functionality, transitivity etc.
  – Facts about particular individuals
    • Types
    • Relationships
    • Values
Annotation Axioms

• An OWLAnnotationAxiom is used to associate arbitrary pieces of information with an object in the ontology
  – Labels or natural language strings
  – Dublin core style metadata, e.g. author or creator information

• Annotation Axioms have no logical significance
  – They do not affect the underlying semantics of the ontology
Change
Changes

• The API takes an “axiom-centric” view

• There are a limited number of change objects
  – Add an Axiom
  – Remove an Axiom
  – Set the Ontology URI

• Trade off between simplicity and power
  – Change from original API, which had a number of different change objects encapsulating different changes.
  – Change object describes what happened, e.g. add/remove
  – Wrapped axiom describes the change
• The OWLOntologyFormat class represents a format used for concrete serialisation
  – E.g. OWL RDF/XML

• The format may also contain information about the particular serialisation
  – E.g. namespace declarations
  – Ordering
  – Structural information
  – Helps in addressing problems with round-tripping

• If an ontology was parsed, the Manager maintains information about the original format of the ontology
Task 2+3

- Creating Entities and Axioms
- Saving the ontology
This is an axiom!

**A statement about the student class.**

**A class expression** in logical terms is a complex concept (such as an “attends some Course”) or a class name (such as “Person”) and is **used in** axioms

Axioms can be **true or false**, class expressions have **instances**
Task 4

• Working with more complicated class expressions and individuals
Inference
Inference

- OWL’s semantics allows the possibility to perform inference or reasoning over an ontology.
- A reasoner may be able to determine additional facts that follow from the information in the ontology.
- How best do we expose this information?
  - Add it into the structure?
  - Provide alternative interfaces?

\[
\begin{align*}
    &a \text{ subClassOf } b \\
    &b \text{ subClassOf } c \\
    &a \text{ subClassOf } (b \text{ and } (\text{some r x})) \\
    &c \text{ equivalentClass } (\text{some r Thing}) \\
    &a \text{ subClassOf } c \\
\end{align*}
\]
Reasoner Implementations

- OWLReasoner and OWLReasonerFactory
- Pellet and HermiT
  - Pure Java implementation
  - Implements OWL API reasoner interfaces
- FaCT++
  - C++ Implementation
  - Java wrapper
  - OWLAPI wrapper implementing OWL API interfaces
Nuts and Bolts

• OWL API code is available from github:
  https://github.com/owlcs/owlapi/releases

• Get the Examples.java!
  – Click on Examples for 3.x → Examples.java

• Latest versions are in the git repository.
Task 5-7

• Some advanced things to look at:
  – Using a reasoner
  – Generating class annotations