The Rough Guide to the OWL API: a tutorial
Version 3.2.3 for OWL 2

Ignazio Palmisano & the OWL API team

Department of Computer Science
University of Manchester

Contacts:
For praises: owlapi-developer@lists.sourceforge.net
For complaints, errors, etc: palmisai@cs.man.ac.uk
For feature requests & bugs: http://owlapi.sourceforge.net trackers

5 June 2011 / OWLED
Outline

1. Introduction: what’s what and what’s an ontology
2. Loading, modifying, saving, checking an ontology
3. Queries
   - Inspect asserted axioms
   - Using a reasoner
4. Outside the core OWL API: extra modules
   - Wait, who changed my ontology? Concurrent access
   - Modularization
5. Applications using the OWL API
   - Protégé
   - OPPL: OWL PreProcessing Language
Where was the API born? Where is it now?

- WonderWeb
  - http://wonderweb.semanticweb.org/
  - first incarnation of the API in this EU STREP project, dated 2003

- CO-ODE
  - http://www.co-ode.org/
  - further support and development in this UK JISC project, until 2009

- currently hosted on SourceForge at http://owlapi.sourceforge.net
  - available under LGPL and/or Apache license
  - a few developers (19 at last count) scattered around, highest concentration at University of Manchester
What’s OWL 2?

**OWL 2**

The OWL 2 Web Ontology Language, informally OWL 2, is an ontology language for the Semantic Web with formally defined meaning.

from: http://www.w3.org/TR/owl2-overview

Description Logics are the formal languages underlying OWL 2

**OWL 2 Profiles**

Not all DLs are created equal:
OWL 2 EL, OWL 2 QL, OWL 2 RL,
OWL 1 DL (slightly enriched in OWL 2 DL)

from: http://www.w3.org/TR/owl2-profiles/#Computational_Properties
What’s an ontology?

I’m not answering THAT...
What’s an ontology? Take two

For the purposes of the OWL API:

- An OWL ontology is a specification of a conceptualization (as defined by Gruber)
- An OWL ontology is structured as described in the OWL 2 specs

I’m a Java developer (get me out of here)

In the OWL API, an OWLOntology is an interface, modelling a set of logical and nonlogical OWL Axioms, with a name (an IRI), an (optional) physical location and convenience methods to retrieve such axioms.
OWL Axioms, Classes, Properties, Individuals and Entities...

- **OWLEntity**: anything that can be identified with an IRI, i.e., class names, data and object properties (and annotation properties) and named individuals

- **OWLAnonymousIndividual, OWLClassExpression, OWLPropertyExpression**: unnamed individuals, class expressions such as restrictions, property expressions such as the inverse of a property

- **OWLAnnotation**: an annotation for any entity, ontology, expression or axiom; characterized by an OWLAnnotationProperty and an OWLAnnotationValue
OWL Axioms, Classes, Properties, Individuals and Entities...

- **OWL Axiom**: the basic unity
  - TBox axioms describe relations between classes and class expressions (equivalence, subsumption, disjointness)
  - ABox axioms (assertions) describe relations between individuals and between individuals and classes/class expressions
  - RBox axioms describe relations between properties
How do I build an object of type...?

- **OWL Ontologies** are created by **OWL Ontology Managers**
- All other interfaces are built using **OWL Data Factory**
  - **OWL Data Factory** is an interface itself
  - A few implementations available: with and without cache, and experiments with threadsafe/memory friendly versions
- Binding to an implementation
  - Only binding needed: **OWL Ontology Manager**
  - **OWL Manager** in the apibinding **package**
  - **OWL Data Factory** is bound in **OWL Manager** for convenience
A Visitor to visit them all

- All important interfaces accept two kinds of visitor
  - `ClassNameVisitor`: visitor stores a value or performs an action
  - `ClassNameVisitorEx`: visitor returns a value
- Most Visitor interfaces have a base implementation
  - `VisitorAdapter`
  - all methods implemented as empty ones
  - Developers only need to override methods they need
Loading or creating an ontology

**OntologyCreation**

```java
OWLOntologyManager m = create();
OWLOntology o = m.createOntology(example_iri);
assertNotNull(o);
```

**OntologyLoading**

```java
OWLOntologyManager m = create();
OWLOntology o = m.loadOntologyFromOntologyDocument(pizza_iri);
assertNotNull(o);
```
A few helpers

- Code snippets from `TutorialSnippets.java`
- Real code (it runs, I promise)
- Box title corresponds to JUnit test name

**TutorialSnippets looks like this...**

```java
public class TutorialSnippets extends TestCase {
    public static final IRI pizza_iri = IRI
        .create("http://www.co-ode.org/ontologies/pizza/pizza.owl");
    public static final IRI example_iri = IRI
        .create("http://www.semanticweb.org/ontologies/ont.owl");
    OWLDataFactory df = OWLManager.getOWLDataFactory();
    public OWLOntologyManager create() {
        OWLOntologyManager m =
            OWLManager.createOWLOntologyManager();
        m.addIRIMapper(new AutoIRIMapper(
            new File("materializedOntologies"), true));
        return m;
    }
    ...
```

Ignazio Palmisano & the OWL API team

DON'T PANIC - The Rough Guide to the OWL API
Alternative loading methods...

**OntologyLoadingFromStringSource**

```java
OWLOntologyManager m = create();
OWLOntology o = m.loadOntologyFromOntologyDocument(pizza_iri);
assertNotNull(o);
StringDocumentTarget target = new StringDocumentTarget();
m.saveOntology(o, target);
m.removeOntology(o);
OWLOntology o2 = m
    .loadOntologyFromOntologyDocument(
        new StringDocumentSource(target.toString()));
assertNotNull(o2);
```

- **OWLOntologyDocumentSource** is an interface for document sources, e.g., readers
- **OWLOntologyDocumentTarget** is an interface for document destinations, e.g., writers
IRIMapper

```java
OWLOntologyManager m = OWLManager.createOWL OntologyManager();
// map the ontology IRI to a physical IRI (files for example)
File output = File.createTempFile("saved_pizza", "owl");
IRI documentIRI = IRI.create(output);
// Set up a mapping, which maps the ontology to the document IRI
SimpleIRIMapper mapper =
    new SimpleIRIMapper(example_save_iri, documentIRI);
m.addIRIMapper(mapper);
// set up a mapper to read local copies of ontologies
File localFolder = new File("materializedOntologies");
// the manager will look up an ontology IRI by checking
// localFolder first for a local copy
m.addIRIMapper(new AutoIRIMapper(localFolder, true));
// Now create the ontology using the ontology IRI (not the
// physical URI)
OWLOntology o = m.createOntology(example_save_iri);
// save the ontology to its physical location - documentIRI
m.saveOntology(o);
```
**Adding axioms to an ontology**

```java
OWLOntologyManager m = create();
OWLOntology o = m.createOntology(pizza_iri);
// class A and class B
OWLClass clsA = df.getOWLClass(IRI.create(pizza_iri + "#A"));
OWLClass clsB = df.getOWLClass(IRI.create(pizza_iri + "#B"));
// Now create the axiom
OWLAxiom axiom = df.getOWLSubClassOfAxiom(clsA, clsB);
// add the axiom to the ontology.
AddAxiom addAxiom = new AddAxiom(o, axiom);
// We now use the manager to apply the change
m.applyChange(addAxiom);
// remove the axiom from the ontology
RemoveAxiom removeAxiom = new RemoveAxiom(o, axiom);
m.applyChange(removeAxiom);
```
Various kinds of changes... SWRL rules

```java
OWLOntologyManager m = create();
OWLOntology o = m.createOntology(example_iri);
// Get hold of references to class A and class B.
OWLClass clsA = df.getOWLClass(
    IRI.create(example_iri + "#A"));
OWLClass clsB = df.getOWLClass(
    IRI.create(example_iri + "#B"));
SWRLVariable var = df.getSWRLVariable(
    IRI.create(example_iri + "#x"));
SWRLClassAtom body = df.getSWRLClassAtom(clsA, var);
SWRLClassAtom head = df.getSWRLClassAtom(clsB, var);
SWRLRule rule = df.getSWRLRule(Collections.singleton(body),
    Collections.singleton(head));
m.applyChange(new AddAxiom(o, rule));
```
Individual Assertions

```java
OWLOntologyManager m = create();
OWLOntology o = m.createOntology(example_iri);
// We want to state that matthew has a father who is peter.
OWLIndividual matthew = df.getOWLNamedIndividual(
    IRI.create(example_iri + "#matthew"));
OWLIndividual peter = df.getOWLNamedIndividual(
    IRI.create(example_iri + "#peter"));
// We need the hasFather property
OWLObjectProperty hasFather = df.getOWLObjectProperty(
    IRI.create(example_iri + "#hasFather"));
// matthew -> hasFather -> peter
OWLAxiom assertion = df.getOWLObjectPropertyAssertionAxiom(
    hasFather, matthew, peter);
// Finally, add the axiom to our ontology
AddAxiom addAxiomChange = new AddAxiom(o, assertion);
m.applyChange(addAxiomChange);
```
Various kinds of changes... Delete individuals

```java
// Delete individuals representing countries
OWL OntologyManager m = create();
OWL Ontology o = m.loadOntologyFromOntologyDocument(pizza_iri);
// Ontologies don’t directly contain entities but axioms
// OWLEntityRemover will remove an entity
// from a set of ontologies by removing all referencing axioms
OWLEntityRemover remover = new OWLEntityRemover(m,
        Collections.singleton(o));
int previous = o.getIndividualsInSignature().size();
// Visit all individuals with the remover
// Changes needed for removal will be prepared
for (OWLNamedIndividual ind : o.getIndividualsInSignature())
    ind.accept(remover);
    m.applyChanges(remover.getChanges());
assertTrue(previous > o.getIndividualsInSignature().size());
```
Various kinds of changes... Existential restrictions

```java
AddSomeRestriction

OWLOntologyManager m = create();
OWLOntology o = m.createOntology(example_iri);
// all Heads have parts that are noses (at least one)
// We do this by creating an existential (some) restriction
OWLObjectProperty hasPart = df.getOWLObjectProperty(
    IRI.create(example_iri + "#hasPart"));
OWLClass nose = df.getOWLClass(
    IRI.create(example_iri + "#Nose"));
// Now let’s describe the class of individuals that have at
// least one part that is a kind of nose
OWLClassExpression hasPartSomeNose =
    df.getOWLObjectSomeValuesFrom(hasPart, nose);
OWLClass head =
    df.getOWLClaas(IRI.create(example_iri + "#Head"));
// Head subclass of our restriction
OWLSUBClassOfAxiom ax =
    df.getOWLSUBClassOfAxiom(head, hasPartSomeNose);
m.applyChange(new AddAxiom(o, ax));
```
Various kinds of changes... Datatype restrictions

DatatypeRestriction

```java
OWLOntologyManager m = create();
OWLOntology o = m.createOntology(example_iri);
// Adults have an age greater than 18.
OWLDataProperty hasAge = df.getOWLDataProperty(
    IRI.create(example_iri + "#hasAge"));
// Create the restricted data range
OWLDatatypeRestriction greaterThan18 = df.getOWLDatatypeRestriction(
    df.getIntegerOWLDatatype(), OWLFacet.MIN_INCLUSIVE,
    df.getOWLLiteral(18));
// Now we can use this in our datatype restriction on hasAge
OWLClassExpression adultDefinition =
    df.getOWLDataSomeValuesFrom(hasAge, greaterThan18);
OWLClass adult = df.getOWLClass(IRI.create(
    example_iri + "#Adult"));
OWLSubClassOfAxiom ax =
    df.getOWLSubClassOfAxiom(adult, adultDefinition);
m.applyChange(new AddAxiom(o, ax));
```
Various kinds of changes... Add a comment (or any annotation)

```java
OWLOntologyManager m = create();
OWLOntology o = m.loadOntologyFromOntologyDocument(pizza_iri);
// We want to add a comment to the pizza class.
OWLClass pizzaCls = df.getOWLClass(
    IRI.create(pizza_iri + "#Pizza"));
// the content of our comment: a string and a language tag
OWLAnnotation commentAnno = df.getOWLAnnotation(
    df.getRDFSComment(),
    df.getOWLLiteral("A class which represents pizzas", "en"));
// Specify that the pizza class has an annotation
OWLAnnotation commentAnno = df.getOWLAnnotation(
    df.getRDFSComment(),
    df.getOWLLiteral("A class which represents pizzas", "en"));
// Add the axiom to the ontology
m.applyChange(new AddAxiom(o, ax));
```
Introduction: what’s what and what’s an ontology
Loading, modifying, saving, checking an ontology
Queries
Outside the core OWL API: extra modules
Applications using the OWL API

Various kinds of changes... Add version info

VersionInfo

```java
OWLOntologyManager m = create();
OWLOntology o = m.loadOntologyFromOntologyDocument(pizza_iri);
// We want to add a comment to the pizza class.
OWLLiteral lit =
        df.getOWLLiteral("Added a comment to the pizza class");
// create an annotation to pair a URI with the constant
OWLAnnotationProperty owlAnnotationProperty =
        df.getOWLAnnotationProperty(
            OWLRDFVocabulary.OWL_VERSION_INFO.getIRI());
OWLAnnotation anno =
        df.getOWLAnnotation(owlAnnotationProperty, lit);
// Now we can add this as an ontology annotation
m.applyChange(new AddOntologyAnnotation(o, anno));
```
Save changes to an ontology

```java
OWLOntologyManager m = create();
OWLOntology o = m.loadOntologyFromOntologyDocument(pizza_iri);
assertNotNull(o);
File output = File.createTempFile("saved_pizza", "owl");
IRI documentIRI2 = IRI.create(output);
// save in OWL/XML format
m.saveOntology(o, new OWLXMLOntologyFormat(), documentIRI2);
// save in RDF/XML
m.saveOntology(o, documentIRI2);
// print out the ontology on System.out
m.saveOntology(o, new SystemOutDocumentTarget());
// Remove the ontology from the manager
m.removeOntology(o);
```
Check OWL profile violations

```java
OWLObjectManager m = create();
OWLontology o = m.createOntology(pizza_iri);
// Available profiles:  DL, EL, QL, RL, OWL2 (Full)
OWL2DLProfile profile = new OWL2DLProfile();
OWLProfileReport report = profile.checkOntology(o);
for(OWLProfileViolation v:report.getViolations()) {
    System.out.println(v);
}
```
Explore classes

ShowClasses

```java
OWLOntologyManager m = create();
OWLOntology o = m.loadOntologyFromOntologyDocument(pizza_iri);
assertNotNull(o);
// Named classes referenced by axioms in the ontology.
for (OWLClass cls : o.getClassesInSignature())
    System.out.println(cls);
```

AssertedSuperclasses

```java
OWLClass clsA = df.getOWLClass(IRI.create(example_iri + "#A"));
Set<OWLClassExpression> superClasses = clsA.getSuperClasses(o);
// for each superclass there will be a corresponding axiom
// the ontology indexes axioms in a variety of ways
Set<OWLSubClassOfAxiom> sameSuperClasses = o.
    .getSubClassAxiomsForSubClass(clsA);
assertEquals(superClasses.size(), sameSuperClasses.size());
```
Walking an ontology

OntologyWalker

// How to walk the asserted structure of an ontology
OWLOntologyManager m = create();
OWL Ontology o = m.loadOntologyFromOntologyDocument(pizza_iri);
// Create the walker
OWL OntologyWalker walker =
    new OWLOntologyWalker(Collections.singleton(o));
// Now ask our walker to walk over the ontology
OWL OntologyWalkerVisitor<Object> visitor =
    new OWLOntologyWalkerVisitor<Object>(walker) {
        @Override
        public Object visit(OWLObjectSomeValuesFrom desc) {
            System.out.println(desc);
            System.out.println("  " + getCurrentAxiom());
            return null;
        }
    };
// Have the walker walk...
walker.walkStructure(visitor);
MergedOntology

OWL Ontology Manager `m = create();`
OWL Ontology `o1 = m.loadOntology(pizza_iri);`
OWL Ontology `o2 = m.loadOntology(example_iri);`

// Create our ontology merger
OWL Ontology Merger `merger = new OWL Ontology Merger (m);`

// Merge all of the loaded ontologies, specifying an IRI for the new ontology
IRI mergedOntologyIRI =
    IRI.create("http://www.semanticweb.com/mymergedont");
OWL Ontology `merged = merger.createMergedOntology (m, mergedOntologyIRI);`

assertTrue(merged.getAxiomCount() > o1.getAxiomCount());
assertTrue(merged.getAxiomCount() > o2.getAxiomCount());
Search for restrictions...

LookupRestrictions

```java
OWLOntologyManager m = create();
OWLOntology o = m.loadOntologyFromOntologyDocument(pizza_iri);
// We want to examine the restrictions on all classes
for (OWLClass c : o.getClassesInSignature()) {
    // collect properties used in existential restrictions
    RestrictionVisitor visitor =
        new RestrictionVisitor(Collections.singleton(o));
    for (OWLAxiom ax: o.getSubClassAxiomsForSubClass(c)) {
        ax.getSuperClass().accept(visitor);
    }
    // Our RestrictionVisitor has now collected all
    // properties that have been restricted in existential
    // restrictions - print them out.
    System.out.println("Properties for " + labelFor(c, o));
    for (OWLObjectPropertyExpression prop: visitor.getRestrictedProperties()) {
        System.out.println("    " + prop);
    }
}
```
**Search for restrictions...**

**RestrictionVisitor** extends an adapter class:

```java
private class RestrictionVisitor extends OWLClassExpressionVisitorAdapter {
    // A few internals omitted...

    public Set<OWLObjectPropertyExpression> getRestrictedProperties() {
        return properties;
    }

    public void visit(OWLClass desc) {
        if (!classes.contains(desc)) {
            classes.add(desc);
            for (OWLOntology ont : onts)
                for (OWLSubClassOfAxiom ax:
                    ont.getSubClassAxiomsForSubClass(desc))
                    ax.getSuperClass().accept(this);
        }
    }

    public void visit(OWLObjectSomeValuesFrom desc) {
        properties.add(desc.getProperty());
    }
}
```
Search annotations

ReadAnnotations

```java
OWLOntologyManager m = create();
OWLOntology o = m.loadOntologyFromOntologyDocument(pizza_iri);
for (OWLClass cls : o.getClassesInSignature()) {
    // Get the annotations on the class that use the label property
    for (OWLAnnotation annotation : cls.getAnnotations(o, df.getRDFSLabel())) {
        if (annotation.getValue() instanceof OWLLiteral) {
            OWLLiteral val = (OWLLiteral) annotation.getValue();
            // look for portuguese labels
            if (val.hasLang("pt"))
                System.out.println(cls + " labelled " + val.getLiteral());
        }
    }
}
```
Change default rendering formats...

Rendering

```java
// Read an ontology and then render it
OWLOntologyManager m = create();
OWLOntology o = m.loadOntologyFromOntologyDocument(pizza_iri);
// Register the ontology storer with the manager
m.addOntologyStorer(new OWLTutorialSyntaxOntologyStorer());
// Save using a different format
m.saveOntology(o, new OWLTutorialSyntaxOntologyFormat(),
    new SystemOutDocumentTarget());
```
Classes needed:

- **OWLTutorialSyntaxOntologyStorer**: the OWLOntologyStorer implementation
  - refers **OWLTutorialSyntaxObjectRenderer** and **OWLTutorialSyntaxRenderer**
  - renders an ontology as an HTML page¹

- **OWLTutorialSyntaxOntologyFormat**: a PrefixOWLOntologyFormat extension

¹Too long to turn into slides, but source available - ask to switch to Eclipse
Visiting labels

class LabelExtractor extends OWLObjectVisitorExAdapter<String> implements OWLAnnotationObjectVisitorEx<String> {
    @Override
    public String visit(OWLAnnotation annotation) {
        if (annotation.getProperty().isLabel()) {
            OWLLiteral c = (OWLLiteral) annotation.getValue();
            return c.getLiteral();
        }
        return null;
    }
}

Ignazio Palmisano & the OWL API team

DON'T PANIC - The Rough Guide to the OWL API
Looking for entity annotations

```java
private LabelExtractor le = new LabelExtractor();

private String labelFor(OWLEntity clazz, OWLOntology o) {
    Set<OWLAnnotation> annotations = clazz.getAnnotations(o);
    for (OWLAnnotation anno : annotations) {
        String result = anno.accept(le);
        if (result != null) {
            return result;
        }
    }
    return clazz.getIRI().toString();
}
```
DL reasoners and the OWL API

- **OWLReasoner** and **OWLReasonerFactory**
- A few OWL DL reasoners available
  - Hermit
  - FaCT++
  - Pellet
  - Reasoners available through OWLLink (e.g., RacerPro)
  - New kid on the block: JFact (a port of FaCT++ to Java)
Introduction: what’s what and what’s an ontology
Loading, modifying, saving, checking an ontology
Queries
Outside the core OWL API: extra modules
Applications using the OWL API

Hierarchy printing...

```java
OWLOntologyManager m = create();
OWLOntology o = m.loadOntologyFromOntologyDocument(pizza_iri);
// Get Thing
OWLClass clazz = df.getOWLThing();
System.out.println("Class : " + clazz);
// Print the hierarchy below thing
printHierarchy(o, clazz, new HashSet<OWLClass>());
```
Hierarchy printing...

Helper method:

```java
public void printHierarchy(OWLReasoner r, OWLClass clazz,
    int level, Set<OWLClass> visited) throws OWLException {
    // Only print satisfiable classes to skip Nothing
    if (!visited.contains(clazz) && reasoner.isSatisfiable(clazz)) {
        visited.add(clazz);
        for (int i = 0; i < level * 4; i++) {
            System.out.print(" ");
        }
        System.out.println(labelFor(clazz, r.getRootOntology()));
    }
    // Find the children and recurse
    NodeSet<OWLClass> classes = r.getSubClasses(clazz, true);
    for (OWLClass child : classes.getFlattened()) {
        printHierarchy(r, child, level + 1);
    }
}
```
List unsatisfiable classes

```java
UnsatisfiableClasses

OWL Ontology Manager m = create();
OWL Ontology o = m.loadOntologyFromOntologyDocument(pizza_iri);
// Create a reasoner; it will include the imports closure
OWL Reasoner reasoner = reasonerFactory.createReasoner(o);
// Ask the reasoner to precompute some inferences
reasoner.precomputeInferences(InferenceType.CLASS_HIERARCHY);
// We can determine if the ontology is actually consistent
assertTrue(reasoner.isConsistent());
// get a list of unsatisfiable classes
Node<OWLClass> bottomNode = reasoner.getUnsatisfiableClasses();
System.out.println("Unsatisfiable classes: ");
// leave owl:Nothing out
for (OWLClass cls : bottomNode.getEntitiesMinusBottom())
    System.out.println(labelFor(cls, o));
```
Direct subclasses

```java
OWLOntologyManager m = create();
OWLOntology o = m.loadOntologyFromOntologyDocument(pizza_iri);
OWLReasoner r = reasonerFactory.createReasoner(o);
r.precomputeInferences(InferenceType.CLASS_HIERARCHY);
// Look up and print all direct subclasses for all classes
// a NodeSet represents a set of Nodes.
// a Node represents a set of equivalent classes
for (OWLClass c : o.getClassesInSignature()) {
    // the boolean argument specifies direct subclasses
    NodeSet<OWLClass> subClasses = r.getSubClasses(c, true);
    for (OWLClass subClass : subClasses.getFlattened())
        System.out.println(labelFor(subClass, o) + " subclass of " + labelFor(c, o));
}
```
Looking up instances and property values

PetInstances

```java
// reasoner from previous example...
// for each class, look up the instances
for (OWLClass c : o.getClassesInSignature()) {
    // the boolean argument specifies direct subclasses
    for (OWLNamedIndividual i :
        r.getInstances(c, true).getFlattened()) {
        System.out.println(labelFor(i, o) +":"+ labelFor(c, o));
    }
}
```

```java
// look up all property assertions
for (OWLObjectProperty op:
    o.getObjectPropertiesInSignature()) {
    NodeSet<OWLNamedIndividual> petValuesNodeSet =
        r.getObjectPropertyValues(i, op);
    for (OWLNamedIndividual value :
        petValuesNodeSet.getFlattened())
        System.out.println(labelFor(i, o) + " " +
            labelFor(op, o) + " " + labelFor(value, o));
}
```
Compute inferences

InferredOntology

// reasoner from previous example...
// Use an inferred axiom generators
List<InferredAxiomGenerator<? extends OWLAXiom>> gens =
    Collections.singletonList(
        new InferredSubClassAxiomGenerator());
OWLOntology infOnt = m.createOntology();
// create the inferred ontology generator
InferredOntologyGenerator iog =
    new InferredOntologyGenerator(r, gens);
iog.fillOntology(m, infOnt);
Necessary property assertions...

```java
// Margherita

// reasoner from previous example...
// For this ontology, we know that classes, properties, ...have
// IRIs of the form: ontology IRI + # + local name
String iri = pizza_iri + "#Margherita";
// Now we can query the reasoner
// to determine the properties that
// instances of Margherita MUST have
OWLClass margherita = df.getOWLClass(IRI.create(iri));
printProperties(m, o, r, margherita);
```
Necessary property assertions...helper

// Prints out the properties that instances must have
private void printProperties(
    OWLOntologyManager man, OWLOntology o,
    OWLReasoner reasoner, OWLClass cls) {
System.out.println("Properties of " + cls);
for (OWLObjectPropertyExpression prop :
    o.getObjectPropertiesInSignature()) {
    // To test if an instance of A MUST have a p-filler,
    // check for the satisfiability of A and not (some p Thing)
    // if this is unsatisfiable, then a p-filler is necessary
    OWLClassExpression restriction =
        df.getOWLObjectSomeValuesFrom(prop, df.getOWLThing());
    OWLClassExpression intersection =
        df.getOWLObjectIntersectionOf(cls,
            df.getOWLObjectComplementOf(restriction));
    if (!reasoner.isSatisfiable(intersection))
        System.out.println("Instances of " + cls + " must have " + prop);
}
}
Concurrent access: Default implementations

- **OWLOntology** contains maps
  - OWLAXioms indexed by OWLEntity in the signature
  - OWLAXioms indexed by AxiomType
  - ...and more

- **OWL OntologyManager** contains maps and sets
  - OWLOntologies indexed by IRI
  - OWLOntologies indexed by OWLOntologyFormat
  - ...and more

- **OWLDataFactory** uses caches to internalize OWLEntities

1. All these are weak spots
2. The list is not exhaustive
3. Transactions: a series of changes instead of a single change?
   Rollback if the last one fails?
Wait, who changed my ontology?

When multithread is the issue...

- Diagnosis can be hard
  - `ConcurrentModificationException` is common but not reliable
  - `NullPointerException` happens sometimes
  - Threading issues masquerading as parsing errors

- Fixes can slow things down
- Immutability a great help
Which solutions are available?

- Synchronize everything? S l o w w w
- Locks? Explicit or implicit? ReadWriteLocks?
- Caches are a vulnerability. Drop them?
- Transaction support. . . hard to figure out
The implConcurrent module

- Alternate implementation for OWLOntologyManager, OWLOntology, OWLDataFactory
- Alternate implementation binding: ThreadSafeOWLManager
  - Alternate implementations can be configured via OWLImplementationBinding
  - OWLDataFactory implementations: cacheless, with explicit locks, ConcurrentHashMap and LRU partial caches
How do I pick and mix?

ThreadSafeBinding

```java
public final class ThreadSafeBinding implements OWLImplementationBinding {
    public OWLOntologyManager getOWLOntologyManager(
            OWLDataFactory d) {
        return new LockingOWLOntologyManagerImpl(d);
    }
    public OWLOntology getOWLOntology(
            OWLOntologyManager oom, OWLOntologyID id) {
        return new LockingOWLOntologyImpl(oom, id);
    }
    public OWLDataFactory getOWLDataFactory() {
        return DataFactoryCSR.getInstance();
    }
}
```
Does it work? Is it fast? Where’s the catch?

- Concurrent implementation passes same tests as default
- Extra tests run same operations multiple times on multiple threads
- Speed varies, depending on choices - usually not much worse

Any catch?

- No transaction support
  - A sequence of changes won’t roll back if the last one fails
  - A thread cannot lock an ontology or a manager and call a sequence of methods
  - Threads can step on each other’s toes

- Protégé offers some support for this
Modularization

- Ontology modularization is a broad topic
- Locality based modularization
  - Many people at Manchester working on it
  - Start from a signature S (set of IRIs) from O
  - Compute a set of axioms M
  - Any expression built with elements from S has the same interpretation in O and in M
  - M is smaller than O → reasoning is faster(ish)
- The only challenge left to the user is how to choose the signature...
**Modularization example**

```java
OWLOntologyManager m = create();
OWLOntology o = m.loadOntologyFromOntologyDocument(pizza_iri);
// extract a module for all toppings
// start by creating a signature "PizzaTopping"
OWLClass topping =
    df.getOWLClass(IRI.create(pizza_iri + "#PizzaTopping"));
// We now add all subclasses of the chosen classes.
Set<OWLEntity> seedSig = new HashSet<OWLEntity>();
OWLReasoner reasoner = reasonerFactory.createReasoner(o);
seedSig.add(topping);
seedSig.addAll(reasoner
    .getSubClasses(ent.asOWLClass(), false).getFlattened());
// Extract a locality-based module
SyntacticLocalityModuleExtractor sme =
    new SyntacticLocalityModuleExtractor(m, o, ModuleType.STAR);
Set<OWLAxiom> mod = sme.extract(seedSig);
System.out.println("Module size "+ mod.size());
```
Protégé: You may have heard of it...

Protégé is a well known ontology editor

- [ ] http://protege.stanford.edu
- [ ] it tracks the latest OWL API developments very closely
- [ ] it provides a lot of useful bug reports
- [ ] Thanks, Timothy :-(
Introduction: what’s what and what’s an ontology
Loading, modifying, saving, checking an ontology
Queries
Outside the core OWL API: extra modules
Applications using the OWL API

Protégé
OPPL: OWL PreProcessing Language

Looks like this...
Introduction: what's what and what's an ontology
Loading, modifying, saving, checking an ontology
Queries
Outside the core OWL API: extra modules
Applications using the OWL API

Protégé
OPPL: OWL PreProcessing Language

OPPL: OWL PreProcessing Language (2)

- Add/remove axioms from ontologies
- Can be used with or without a reasoner
- Plugs into Protégé
A few OPPL scripts...

Declare matched classes disjoint

```oppl
?x:CLASS, ?y:CLASS
SELECT ?x subClassOf gender,
   ?y subClassOf gender
WHERE ?x != ?y
BEGIN
  ADD ?x disjointWith ?y
END;
```
Add restrictions

```oppl
?x: CLASS
SELECT ?x subClassOf person
BEGIN
  ADD ?x subClassOf has_age some int
END;
```
A few OPPL scripts...

Assertions can be changed too

```oppl
?country:INDIVIDUAL[instanceOf Country],
?adiacentCountry:INDIVIDUAL[instanceOf Country]
SELECT ?country adjacentTo ?adiacentCountry
BEGIN
    REMOVE ?country adjacentTo ?adiacentCountry,
    ADD ?country instanceOf
        hasLandBoundary some (LandBoundaryFragment
            and boundaryOf value ?adiacentCountry)
END;
```
Patterns

- OPPL scripts without a SELECT section
- Variable binding done manually
- Useful for more localized tasks
- Available in Protégé too
Question time

Questions?

Contacts:
For praises: owlapi-developer@lists.sourceforge.net
For complaints, errors, etc: palmisai@cs.man.ac.uk
For feature requests & bugs: http://owlapi.sourceforge.net trackers