Building Ontology-based Applications using Pellet

Evren Sirin
Clark & Parsia, LLC

evren@clarkparsia.com
Tutorial Webpage

http://clarkparsia.com/pellet/tutorial

Unless otherwise noted, tutorial materials are available under the CC Attribution-Share Alike 3.0 United States License.

Code bundled in the tutorial is available under AGPL v. 3 terms.
What is Clark & Parsia?

- Small R&D firm in Washington, DC
- Provides software development and integration services
- Specializing in Semantic Web, web services, and advanced AI technologies for federal and enterprise customers

http://clarkparsia.com/
Twitter: @candp
What is Pellet?

- Pellet is an OWL-DL reasoner
  - Supports nearly all of OWL 1 and OWL 2
  - Sound and complete reasoner
- Written in Java and available from [http://clarkparsia.com/pellet](http://clarkparsia.com/pellet)
- Dual-licensed
  - AGPL license for open-source applications
  - Proprietary license available for commercial applications
Tutorial Schedule

- Introduction and orientation (20 min)
- Basic of OWL reasoning (20 min)
- Ontology development with Pellet (25 min)
- Break (15 min)
- Ontology alignment (20 min)
- Programming with Pellet (45 min)
- Break (15 min)
- Closed-world instance validation (20 min)
- Advanced Pellet programming (45 min)
- Wrap-up (15 min)
Running Example: POPS

● Expertise location in a large organization
  ○ Based on POPS application in NASA
  ○ Multiple sources containing personnel data: contact information, work history, evidence of skills, publications, etc.
  ○ Find people that satisfy certain conditions

● Several challenges
  ○ Integrate data from multiple sources
  ○ Ensure data consistency
  ○ Query with inferencing
  ○ Faceted browser user interface
    ■ Not covered in this talk; see jSpace
    ■ Soon to be rebranded as Pelorus
JSpace - POPS
Let's build it!

<table>
<thead>
<tr>
<th>Organization (367)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Petru Maior&quot; University of Targu Mures</td>
</tr>
<tr>
<td>A.I. Group, Faculty of Technology, Bielefeld</td>
</tr>
<tr>
<td>Accenture Technology Labs</td>
</tr>
<tr>
<td>Adobe Systems Inc.</td>
</tr>
<tr>
<td>AgroParisTech</td>
</tr>
<tr>
<td>AIIR</td>
</tr>
<tr>
<td>Aristotle University of Thessaloniki</td>
</tr>
<tr>
<td>Asian Institute of Technology</td>
</tr>
<tr>
<td>Atos Origin</td>
</tr>
<tr>
<td>Bard College</td>
</tr>
<tr>
<td>Basque Country University</td>
</tr>
<tr>
<td>BBN</td>
</tr>
<tr>
<td>BEA Systems, Inc.</td>
</tr>
<tr>
<td>Beijing University</td>
</tr>
<tr>
<td>Bibliothèque nationale de France</td>
</tr>
<tr>
<td>Birkbeck, University of London</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project (2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIANCA</td>
</tr>
<tr>
<td>POPS</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Person (1310)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Min Tjoa</td>
</tr>
<tr>
<td>Aaron Kershenbaum</td>
</tr>
<tr>
<td>AbdelMounaam Rezgui</td>
</tr>
<tr>
<td>Abhita Chugh</td>
</tr>
<tr>
<td>Abir Qasem</td>
</tr>
<tr>
<td>Abraham Bernstein</td>
</tr>
<tr>
<td>Achille Fokoue</td>
</tr>
<tr>
<td>Adam Funk</td>
</tr>
<tr>
<td>Adina Sirbu</td>
</tr>
<tr>
<td>Aditya Kalyanpur</td>
</tr>
<tr>
<td>Adrian Mocan</td>
</tr>
<tr>
<td>Adriana Budura</td>
</tr>
<tr>
<td>Afraz Jaffri</td>
</tr>
<tr>
<td>Agata Filipowska</td>
</tr>
<tr>
<td>Ahlem Ben Hassine</td>
</tr>
<tr>
<td>Ahmad Iqbal</td>
</tr>
<tr>
<td>Aidan Hogan</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Publication (490)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[facett: A Browser for Heterogeneous Semantic Web Repositories]</td>
</tr>
<tr>
<td>A Bootstrapping Architecture for Integration of Relational Databases to the Semantic Web</td>
</tr>
<tr>
<td>A Caching Mechanism for Semantic Web Service Discovery</td>
</tr>
<tr>
<td>A cognitive support framework for ontology mapping</td>
</tr>
<tr>
<td>A Collaborative Semantic Web Layer to Enhance Legacy Systems</td>
</tr>
<tr>
<td>A Constraint-based Approach to Horizontal Web Service Composition</td>
</tr>
<tr>
<td>A Core Ontology for Business Process Analysis</td>
</tr>
<tr>
<td>A Core Ontology for Knowledge Acquisition</td>
</tr>
<tr>
<td>A Domain Ontology Construction Method Supported by an Ontology Search Engine</td>
</tr>
</tbody>
</table>
Building the Example

- Author ontology schemas
  - Validate and debug schema definitions
- Connect multiple schemas
  - Simple ontology alignment
- Validating instance data
  - Identify and resolve inconsistencies in the data
  - Closed world data validation with Pellet Integrity Constraints
- Reasoning with instance data
  - Answer queries over combined data using Pellet
  - Scalability and performance considerations
OWL and Reasoning
OWL in 3 Slides (1)

ENTITIES

- Class: Person, Organization, Project, Skill, ...
- Datatype: string, integer, date, ...

- Individual: Evren, C&P, POPS, ...
- Literal: "Evren Sirin", 5, 5/26/2008, ...

- Object Property: worksAt, hasSkill, ...
- Data property: name, proficiencyLevel, ...
OWL in 3 Slides (2)

EXPRESSIONS

● Class expressions
  ○ and, or, not
  ○ some, only, min, max, exactly, value, Self
  ○ { ... }

● Datatype definitions
  ○ and, or, not
  ○ <, <=, >, >=
  ○ { ... }
OWL in 3 Slides (3)

AXIOMS

- Class axioms
  - subClassOf, equivalentTo, disjointWith

- Property axioms
  - subPropertyOf, equivalentTo, inverseOf, disjointWith, subPropertyChain, domain, range

- Property characteristics
  - Functional, InverseFunctional, Transitive, Symmetric, Asymmetric, Reflexive, Irreflexive

- Individual assertions
  - Class assertion, property assertion, sameAs, differentFrom
OWL Example

- Employee equivalentTo (CivilServant or Contractor)
- CivilServant disjointWith Contractor
- Employee subClassOf
  employeeID some integer[>= 100000, <= 999999]
- Employee subClassOf employeeID exactly 1
- worksOnProject domain Person
- worksOnProject range Project
- Person0853 type CivilServant
- Person0853 employeeID 312987
- Person0853 worksOnProject Project2133
**OWL Example**

- **Employee** `equivalentTo (CivilServant or Contractor)`
- **CivilServant** `disjointWith Contractor`
- **Employee** `subClassOf` `employeeID` `some integer[>= 100000, <= 999999]`
- **Employee** `subClassOf` `employeeID` `exactly 1`
- **worksOnProject** `domain` Person
- **worksOnProject** `range` Project
- **Person0853** `type` CivilServant
- **Person0853** `employeeID` 312987
- **Person0853** `worksOnProject` Project2133

---

**Schema (TBox)**

**Data (ABox)**
Reasoning in OWL

1. Check the consistency of a set of axioms
   ○ Verify the input axioms do not contain contradictions
Inconsistency Examples

● Example 1
  ○ CivilServant disjointWith Contractor
  ○ Person0853 type CivilServant, Contractor

● Example 2
  ○ ActiveProject subclassOf endDate max 0
  ○ Project2133 type ActiveProject
  ○ Project2133 endDate "1/1/2008"^^xsd:date
Unsatisfiability

● Unsatisfiable class cannot have any instances
  ○ Consistent ontologies may contain unsatisfiable classes
  ○ Declaring an instance for an unsatisfiable class causes inconsistency

● Example
  ○ CivilServant disjointWith Contractor
  ○ CivilServantContractor subClassOf ( CivilServant and Contractor )
Reasoning in OWL

1. Check the consistency of a set of axioms
   ○ Verify the input axioms do not contain contradictions
   ○ Mandatory first step before any other reasoning service
   ○ Fix the inconsistency before reasoning
     - Why?
     - Because any consequence can be inferred from inconsistency
Inference Examples

● Input axioms
  1. Employee equivalentTo ( CivilServant or Contractor )
  2. CivilServant disjointWith Contractor
  3. isEmployeeOf inverseOf hasEmployee
  4. isEmployeeOf domain Employee
  5. Person0853 type CivilServant
  6. Person0853 isEmployeeOf Organization5349

● Some inferences
  ○ CivilServant subClassOf Employee { 1 }
  ○ Person0853 type Employee { 1, 5 }, { 4, 6 }
  ○ Person0853 type not Contractor { 2, 5 }
  ○ Organization5349 hasEmployee Person0853 { 3, 6 }
Reasoning in OWL

1. Check the consistency of a set of axioms
   ○ Verify the input axioms do not contain contradictions
   ○ Mandatory first step before any other reasoning service
   ○ Fix the inconsistency before reasoning
     ■ Any consequence can be inferred from inconsistency

2. Infer new axioms from a set of axioms
   ○ Truth of an axiom is logically proven from asserted axioms
   ○ Infinitely many inferences for any non-empty ontology
   ○ Inferences can be computed as a batch process or as required by queries
Common Reasoning Tasks

- Classification
  - Compute subClassOf and equivalentClass inferences between all named classes
- Realization
  - Find most specific types for each instance
  - Requires classification to be performed first
Asserted Ontology
Inferred Subclasses
Classification Tree

owl:Thing

A

C

B

D

x

z

y

Class
instance

subClassOf (transitive reduction)
Instance Realization

Inferences

\[
x \text{ rdf:type } A
\]
\[
z \text{ rdf:type } D
\]
SPARQL Queries

- Retrieve subclasses
  ```sparql
  SELECT ?C WHERE {
    ?C rdfs:subClassOf :Employee .
  }
  ```

- Retrieve instances
  ```sparql
  SELECT ?X WHERE {
    ?X rdf:type :Employee .
  }
  ```

- Retrieve subclasses and their instances
  ```sparql
  SELECT ?X ?C WHERE {
    ?C rdfs:subClassOf :Employee .
  }
  ```
Ontology Development
CLI Demo

- Incrementally build the ontology
  - Basic modeling and reasoning
- Go through Pellet CLI features
  - Consistency, explanation, lint
- See the tutorial distribution file for the versions of the ontology we are building
  - data/README.txt - general instructions
  - data/commands.txt - CLI commands used
Ontology Alignment
Data Integration

● Integrate data from multiple sources
● Sources use different vocabularies
● Establish a common vocabulary to enable uniform access to all data sources
● Goal for our running example
  ○ Integrate POPS data with FOAF data
  ○ Align POPS and FOAF vocabularies
  ○ Use a single query to retrieve instances from both data sets
Simple Alignment

- `pops:Employee` subClassOf `foaf:Person`
- `pops:Project` equivalentTo `foaf:Project`
- `pops:Organization` equivalentTo `foaf:Organization`

- `pops:hasEmployee` subPropertyOf `foaf:member`
- `pops:mbox_sha1sum` equivalentTo `foaf:mbox_sha1sum`
Alignment with SWRL

- Mapping sometimes not straight-forward
  - POPS defines `firstName` and `lastName`
  - FOAF defines `name`
  - Concat first and last names to get the full name
- SWRL rule with a built-in function

```swrl
pops:firstName(?person, ?first) ^
pops:lastName(?person, ?last) ^
?name = swrlb:concat(?first " " ?last)
=>
foaf:name(?person, ?name)
```
More SWRL Mapping

- Another example
  - POPS uses `worksOnProject` property for both current and previous projects
  - FOAF distinguishes `currentProject` and `pastProject`
- Solution: POPS also defines `ActiveProject` class
- SWRL rule to encode conditional subproperty:

```swrl
pops:worksOnProject(?person, ?project) ^
pops:ActiveProject(?project)
=>
foaf:currentProject(?person, ?project)
```
Performance Tuning

- For best Pellet performance minimize class atoms and maximize property atoms in rules
- With a modeling trick we can remove the class atom from the rule
  - Instead of this pattern
  - We want this pattern

![Diagram]

- We want this pattern

![Diagram]
New Mapping Rule

pops:ActiveProject subClassOf pops:activeProject Self

pops:worksOnProject(?person, ?project) ^
pops:activeProject(?project, ?project) =>
foaf:currentProject(?person, ?project)
Final Mapping Rule

pops:ActiveProject subclassOf pops:activeProject Self

foaf:currentProject propertyChainAxiom
  ( pops:worksOnProject pops:activeProject )
Programming with Pellet
APIs for accessing Pellet

- Pellet can be used via three different APIs
  - Internal Pellet API
  - Manchester OWLAPI
  - Jena API
- Each API has pros and cons
  - Choice will depend on your applications’ needs and requirements
Pellet Internal API

- API used by the reasoner
  - Designed for efficiency, not usability
  - Uses ATerm library for representing terms
  - Fine-grained control over reasoning
  - Misses features (e.g. parsing & serialization)

- Pros: Efficiency, fine-grained control
- Cons: Low usability, missing features
Manchester OWLAPI

- API designed for OWL
  - Closely tied to OWL structural specification
  - Support for many syntaxes (RDF/XML, OWL/XML, OWL functional, Turtle, ...)
  - Native SWRL support
  - Integration with reasoners
  - Support for modularity and explanations

- Pros: OWL-centric API
- Cons: Not as stable, no SPARQL support (yet)
- More info: [http://owlapi.sf.net](http://owlapi.sf.net)
Jena API

- RDF framework developed by HP labs
  - An RDF API with OWL extensions
  - In-memory and persistent storage
  - Built-in rule reasoners and integrated with Pellet
  - SPARQL query engine
- Pros: Mature and stable and ubiquitous
- Cons: Not great for handling OWL, no specific OWL 2 support
- More info: http://jena.sf.net
Jena Basics

- **Model** contains set of **Statements**
- **Statement** is a triple where
  - Subject is a **Resource**
  - Predicate is a **Property**
  - Object is an **RDFNode**
- **InfModel** extends **Model** with inference
- **OntModel** extends **InfModel** with ontology API
Creating Inference Models

// create an empty non-inferencing model
Model rawModel = ModelFactory.createDefaultModel();

// create Pellet reasoner
Reasoner r = PelletReasonerFactory.theInstance().create();

// create an inferencing model using the raw model
InfModel model = ModelFactory.createInfModel(r, rawModel);
Creating Ontology Models

// create an empty non-inferencing model
Model rawModel = ModelFactory.createDefaultModel();

// create an ontology model using Pellet spec and raw model
OntModel model = ModelFactory.createOntologyModel(
    PelletReasonerFactory.THE_SPEC, rawModel);
Which Model to Use?

- Ontology API may introduce some overhead
  - Additional object conversions (from RDF API objects to OWL API objects)
  - Additional queries to the underlying reasoner
Data Validation
Consistency Checking

// create an inferencing model using Pellet reasoner
InfModel model = ModelFactory.createInfModel(r, rawModel);

// get the underlying Pellet graph
PelletInfGraph pellet = (PelletInfGraph) model.getGraph();

// check for inconsistency
boolean consistent = pellet.isConsistent();
Explaining Inconsistency

// IMPORTANT: The option to enable tracing should be turned on before the ontology is loaded to the reasoner!
PelletOptions.USE_TRACING = true;

// create an inferencing model using Pellet reasoner
InfModel model = ModelFactory.createInfModel(r, rawModel);
PelletInfGraph pellet = (PelletInfGraph) model.getGraph();

// create an inferencing model using Pellet reasoner
if(!pellet.isConsistent()) {
    // create an inferencing model using Pellet reasoner
    Model explanation = pellet.explainInconsistency();
    // print the explanation
    explanation.write(System.out);
}

Dealing with Inconsistency

● Inconsistencies are unavoidable
  ○ Distributed data, no single point of enforcement
  ○ Expressive modeling language
● Classical logical formalisms are not good at dealing with inconsistency
  ○ Reasoners refuse to reason with inconsistent ontologies
● Paraconsistent logics not practical
  ○ Complexity, tool support, etc.
● What can we do?
An Automated Solution

• Typical process for solving a contradiction
  ○ Use Pellet to find which axioms cause contradiction
  ○ Domain expert (human) inspects the axiom set
  ○ Expert edits/deleted incorrect axioms

• An automated (and cautious) solution
  ○ Use Pellet to find which axioms cause contradiction
  ○ Delete all reported axioms (WIDTIO)

• When to use the automated solution
  ○ Pros: Completely automated, guaranteed to retain only consistent information
  ○ Cons: May remove too much information
Resolving Inconsistencies

// continue until all inconsistencies are resolved
while (!pellet.isConsistent()) {
    // get the explanation for current inconsistency
    Graph explanation = pellet.explainInconsistency();
    // iterate over the axioms in the explanation
    for (Triple triple : explanation.find(Triple.ANY).toList()) {
        // remove any individual assertion that contributes
        // to the inconsistency (assumption: all the axioms
        // in the schema are believed to be correct and
        // should not be removed)
        if (isIndividualAssertion(triple))
            graph.remove(triple);
    }
}
Closed vs. Open World

- Two different views on truth
  - CWA: Any statement that is not known to be true is false
  - OWA: A statement is false only if it is known to be false

- Used in different contexts
  - Databases use CWA because (typically) you have complete information
  - Ontologies use OWA because (typically) you have incomplete information

- Data validation results significantly different when using CWA instead of OWA
Example (1)

- **Input axioms**
  - Employee subclassOf employeeID some integer
  - Person0853 type Employee

- **OWA**
  - Consistent: true
  - Reason: Person0853 has an employeeID but we don't know the exact value

- **CWA**
  - Consistent: false
  - Reason: Person0853 does not have an employeeID
Example (2)

- Input axioms
  - `isEmployeeOf range Organization`
  - `Person0853 isEmployeeOf Organization5349`

- OWA
  - Consistent: true
  - Inference: `Organization5349 type Organization`

- CWA
  - Consistent: false
  - Reason: `Organization5349 type Organization` is not explicitly asserted
Example (3)

- **Input axioms**
  - hasManager Functional
  - Organization5349 hasManager Person0853
  - Organization5349 hasManager Person1735

- **OWA**
  - Consistent: true
  - Inference: Person0853 sameAs Person1735

- **CWA**
  - Consistent: false
  - Reason: Organization5349 has more than one value for hasManager
Should I use CWA or OWA?
- Of course use both!
- In the application domain there is complete information about some parts but not others

In POPS application we have...
- Complete knowledge about employees
- Incomplete information about external publications
  - Retrieved from conference proceedings, etc

An axiom can be interpreted with...
- OWA - regular OWL axiom
- CWA - integrity constraint (IC)
How to use ICs in OWL

● Two easy steps
  1. Specify which axioms should be ICs
  2. Validate ICs with Pellet

● Ontology developer
  ○ Develop ontology as usual
  ○ Separate ICs from regular axioms
    ■ Annotation, separation of files, named graphs, ...

● Pellet IC validator
  ○ Translates ICs into SPARQL queries automatically
  ○ Execute SPARQL queries with Pellet
  ○ Query results show constraint violations

● Download: http://clarkparsia.com/pellet/download/oicv-0.1.1
IC Validation

// create an inferencing model using Pellet reasoner
InfModel dataModel = ModelFactory.createInfModel(r);
// load the schema and instance data to Pellet
dataModel.read("file:data.rdf");
dataModel.read("file:schema.owl");

// Create the IC validator and associate it with the dataset
JenaICValidator validator = new JenaICValidator(dataModel);

// Load the constraints into the IC validator
validator.getConstraints().read("file:constraints.owl");
// Get the constraint violations
Iterator<ConstraintViolation> violations =
    validator.getViolations();
Resolving IC Violations

- IC violations are similar to logical inconsistencies but not exactly same
  - Lack of information may cause IC violation
- ICs do not cause new inferences
  - Used to detect violations
- Resolving IC violations
  - Add more information
    - Example: Add the missing employee ID info
  - Delete existing information
    - Example: Remove the employee
Query Answering
Querying via RDF API

// Get the resource we want to query about
Resource Employee = model.getResource(
    NS + "Employee" );

// Retrieve subclasses
Iterator subClasses = model.listSubjectsWithProperty(
    RDFS.subClassOf, Employee);

// Retrieve direct subclasses
Iterator directSubClasses = model.listSubjectsWithProperty(
    ReasonerVocabulary.directSubClassOf, Employee);

// Retrieve instances
Iterator instances = model.listSubjectsWithProperty(
    RDF.type, Employee);
// Get the resource we want to query about
OntClass Employee = ontModel.getResource(NS + "Employee");

// Retrieve subclasses
Iterator subClasses = Employee.listSubClasses();

// Retrieve direct subclasses
Iterator supClasses = Employee.listSubClasses(true);

// Retrieve instances
Iterator instances = Employee.listInstances();
Querying with SPARQL

```
Query query = Query.create(
    PREFIXES +
    "SELECT ?X ?C " +
    "WHERE {
    " +
    "      ?X rdf:type ?C ." +
    "      ?C rdfs:subClassOf :Employee ." +
    "}" );

    // Create a query execution engine with a Pellet model
    QueryExecution qe =
    QueryExecutionFactory.create(query, model);

    // Run the query
    ResultSet results = qe.execSelect();
```
Query query = Query.create(
    PREFIXES +
    "SELECT ?X ?C " +
    "WHERE {
    " +
    "    ?X sparqldl:directType ?C ." +
    "    ?C rdfs:subClassOf :Employee ." +
    "}" );

    // Create a query execution engine with a Pellet model
QueryExecution qe =
    SparqlDLQueryExecutionFactory.create(query, model);

    // Run the query
ResultSet results = qe.execSelect();
SPARQL Engines

- **ARQ query engine (comes with Jena)**
  - ARQ handles the query execution
  - Calls Pellet with single triple queries
  - Supports all SPARQL constructs
  - Does not support OWL expressions

- **Pellet query engine**
  - Pellet handles the query execution
  - Supports only Basic Graph Patterns
  - Supports OWL expressions

- **Mixed query engine**
  - ARQ handles SPARQL algebra, Pellet handles Basic Graph Patterns
  - Supports all OWL and SPARQL constructs
Advanced Pellet Programming
Under the Hood

Main processing/reasoning steps
1. Loading data from Jena to Pellet
2. Consistency checking
3. Classification [Optional]
   ■ Compute subClassOf and equivalentClass inferences between all named classes
4. Realization [Optional]
   ■ Compute instances for all named classes

Steps should be performed in the given order
No need to repeat any of the steps unless the underlying data changes
Processing Steps

● Loading and consistency checking mandatory
  ○ Pellet performs

● Classification and realization optional
  ○ Performed only if required by a query
  ○ Queries triggering classification
    ■ Querying for equivalent classes
    ■ Querying for (direct or all) sub/super classes
    ■ Querying for disjoint/complement classes
  ○ Queries triggering realization
    ■ Querying for direct instances of a class
    ■ Querying for (direct or all) types of an individual
// Create objects as usual
InfModel model = ModelFactory.createInfModel(r, rawModel);
PelletInfGraph pellet = (PelletInfGraph) model.getGraph();

// Load data to Pellet
model.rebind();
// Check consistency
boolean consistent = pellet.isConsistent();
// Trigger classification
pellet.classify();
// Trigger realization
pellet.realize();
public class ClassificationMonitor extends AbstractProgressMonitor {
    private JProgressBar progressBar;

    public ClassificationMonitor(JProgressBar progressBar) {
        this.progressBar = progressBar;
    }

    public void setProgressLength(int length) {
        progressBar.setMaximum(length);
    }

    protected void updateProgress() {
        progressBar.setValue(getProgress());
    }
}
JProgressBar progressBar = new JProgressBar(JProgressBar.HORIZONTAL);
PelletInfGraph pellet = (PelletInfGraph) model.getGraph();

progressBar.setIndeterminate(true);
pellet.isConsistent();
progressBar.setIndeterminate(false);

TaxonomyBuilder taxonomyBuilder = pellet.getKB().getTaxonomyBuilder();
taxonomyBuilder.setProgressMonitor(new ClassificationMonitor(progressBar));

pelletGraph.classify();
Multi-threaded Query

● Pellet is not really thread-safe
  ○ But you can run multiple queries concurrently if you are careful

● What you need to do
  ○ Perform consistency checking first
  ○ Perform classification or don't ask queries that triggers classification - cls.listSubClasses()
  ○ Perform realization or don't ask queries that triggers realization - cls.listIndividuals(true)

● More details
Log Configuration

handlers = java.util.logging.ConsoleHandler

# Modify the following level property for more or less verbose console logging
java.util.logging.ConsoleHandler.level = FINEST

# Modify the following property to select a different log record formatter
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter

# The log level for specific loggers can be configured
# Turn off warnings displayed during loading
org.mindswap.pellet.jena.graph.loader.DefaultGraphLoader.level = SEVERE
// create an ontology model using Pellet spec
OntModel model = ModelFactory.createOntologyModel(
    PelletReasonerFactory.THE_SPEC);

// Add sub models
model.addSubModel( dataModel1 );
model.addSubModel( dataModel2 );

// Remove sub models
model.removeSubModel( dataModel2 );
// Create an ontology model and load the data
OntModel model = ModelFactory.createOntologyModel(
    PelletReasonerFactory.THE_SPEC);
model.read(ontologyURI);

// Get an existing class from the ontology
// (Triggers load and consistency checking because
// getOntClass queries the reasoner)
OntClass cls = model.getOntClass(classURI);

// Create an instance (modifies the model so reasoner status
// becomes out of sync)
Individual ind = cls.createIndividual(individualURI);

// Run a query (requires another consistency check)
Iterator i = model.listStatements(...)
Update Non-inference Model

// Create a non-inferencing ontology model and load the data
OntModel rawModel = ModelFactory.createOntologyModel(
    OntModelSpec.OWL_MEM);
rawModel.read(ontologyURI);

// Create a Pellet model on top of the raw model
OntModel model = ModelFactory.createOntologyModel(
    PelletReasonerFactory.THE_SPEC, model);

// Get an existing class from the raw model
OntClass cls = rawModel.getOntClass(classURI);

// Create an instance in the raw model
Individual ind = cls.createIndividual(individualURI);

// Query the inference model (updates automatically detected)
Iterator i = model.listStatements(...);
Demo Application

- Log configuration
- Inconsistency detection and automated resolution
- Multi-threaded query execution
- Automated query generation and execution
- Class hierarchy visualization
- Handling updates (addition/removal)
- Handling sameAs inferences