Distributed Intelligent Systems

Laboratorio
Ontologías: SPARQL JENA

Ulises Cortés

Ignasi Gómez-Sebastià

Luis Oliva

Sergio Alvarez

SID2022

{ia, igomez, loliva, salvarez}@cs.upc.edu

Introducción

- **S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage
- Lenguaje de consultas *de facto* para la web semántica
- Funciona bajo RDF
  - Consultas basadas en tripletas
- Permite consultas en datos estructurados y semi-estructurados
- Permite consultas sobre las estructuras de datos de forma natural

Introducción

- Permite *joins* sobre bases de conocimiento no homogéneas
- Dispone de algunas extensiones
  - GeoSPARQL
  - SPARUL
    - SPARQL con DML
      - INSERT
      - UPDATE
- Versión 1.0 de 2008
  - Versión 1.1 de 2013

Estructura de una consulta

- **Prefijos: Permiten abreviar URIs que de otra forma serían muy largas (opcionales)**
- **Definición de modelos: Sobre qué modelos RDF hacemos la consulta**
- **Resultado: Información que estamos obteniendo**
- **Patrón de consulta: Cómo obtenemos dicha información**
- **Modificadores de consulta: División, ordenado, etc**

```
# prefix declarations
PREFIX foo: <http://example.com/resources/>
...
# dataset definition
FROM ...
# result
                    clause
SELECT ...
# query pattern
WHERE {
    ...
}
# query modifiers
ORDER BY ...
```

Ejemplo de consulta I

```
select distinct ?Concept where {[] a ?Concept} LIMIT 100

PREFIX dbpedia: <http://dbpedia.org/property/birthName>
PREFIX actor: <http://dbpedia.org/ontology/Actor>
SELECT *
WHERE {
    ?actor a <http://dbpedia.org/ontology/Actor>.
?actor <http://dbpedia.org/property/birthName> ?nombre

} LIMIT 50
```

- ?Variable
- . Al final de cada línea en el WHERE
- WHERE va entre corchetes, no paréntesis
- Uso de tripletas
  - Notad como saltamos entre nodos

Ejercicio I

- Usad el motor SPARQL del Universal Protein Resource: https://sparql.uniprot.org/
- Investigad las consultas que hay disponibles para entender cómo se estructura la ontología
- Obtened los acrónimos (`up:mnemonic`) y nombres (`skos:prefLabel`) de 100 enfermedades

Ejemplo de consulta II

```
PREFIX actor: <http://dbpedia.org/ontology/Actor>
SELECT ?nombreActor, ?nombrePeli
WHERE {
    ?actor a <http://dbpedia.org/ontology/Actor>.
?peli <http://dbpedia.org/ontology/starring> ?actor.
?actor <http://dbpedia.org/property/name> ?nombreActor.
?peli <http://xmlns.com/foaf/0.1/name> ?nombrePeli.
} LIMIT 50
```

```
PREFIX actor: <http://dbpedia.org/ontology/Actor>
SELECT ?nombreActor, ?nombrePeli
WHERE {
    ?actor a <http://dbpedia.org/ontology/Actor>.
?peli <http://dbpedia.org/ontology/starring> ?actor.
?actor <http://dbpedia.org/property/name> ?nombreActor.
?peli <http://xmlns.com/foaf/0.1/name> ?nombrePeli.
?actor <http://dbpedia.org/property/name> "Bruce Lee"@en.
} LIMIT 50
```

```
PREFIX actor: <http://dbpedia.org/ontology/Actor>
SELECT ?nombreActor, ?nombrePeli
WHERE {
    ?actor a <http://dbpedia.org/ontology/Actor>.
?peli <http://dbpedia.org/ontology/starring> ?actor.
?actor <http://dbpedia.org/property/name> ?nombreActor.
?peli <http://xmlns.com/foaf/0.1/name> ?nombrePeli.
FILTER regex(?nombreActor, "^Bruce")
} LIMIT 50
```

Modificadores útiles

- a (rdf:type)
- LIMIT
- ORDER BY
  - ORDER BY ?X DESC ?Y
- OFFSET
- DISTINCT

Modificadores útiles

- **CONSTRUCT (Grafo RDF como resultado)**
- **DESCRIBE (Descripción de un grafo RDF), e.g.:**

```
DESCRIBE <http://purl.uniprot.org/core/>
```

Modificadores útiles II

- FILTER, MINUS
  - `SELECT * {?s ?p ?o FILTER NOT EXISTS {?s ?y ?z}}`
  - `SELECT * {?s ?p ?o FILTER REGEX(?s, STRING, "i")}`
    - `FILTER REGEX funciona como LIKE en SQL`
  - `SELECT * {?s ?p ?o MINUS {?x ?y ?z}}`
  - `FILTER(LANG(?label) = "" || LANGMATCHES(LANG(?label), "en"))`
- BIND

```
SELECT ?nombre ?precioFinal
{?x precio ?precio . ?x descuento ?descuento.
 ?x nombre ?nombre .
 BIND (?precio * (1 - ?descuento) AS ?precioFinal)}
```

Modificadores útiles II

- ASK (boolean como resultado)
- OPTIONAL (Bases heterogeneas)
- SameTerm (?X, ?Y)
- Comentarios

```
# Esto es un comentario
```

Ejercicio II

- Usad el motor SPARQL del Universal Protein Resource: https://sparql.uniprot.org/
- Obtened las enfermedades cuyo acrónimo sea *3KTD*

DML

- **INSERT**

```
INSERT {
  <http://example/libro> titulo “Snowcrash";
                         Autor “Neal Stephenson”;
                         Precio “Un huevo”} WHERE{}
```

- **DELETE**

```
DELETE{?x} WHERE {?x titulo “SnowCrash”}
```

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
WITH <http://example/addresses>
DELETE { ?person foaf:givenName 'Bill' }
INSERT { ?person foaf:givenName 'William' }
WHERE { ?person foaf:givenName 'Bill’ }
```

DML II

- **UPDATE**

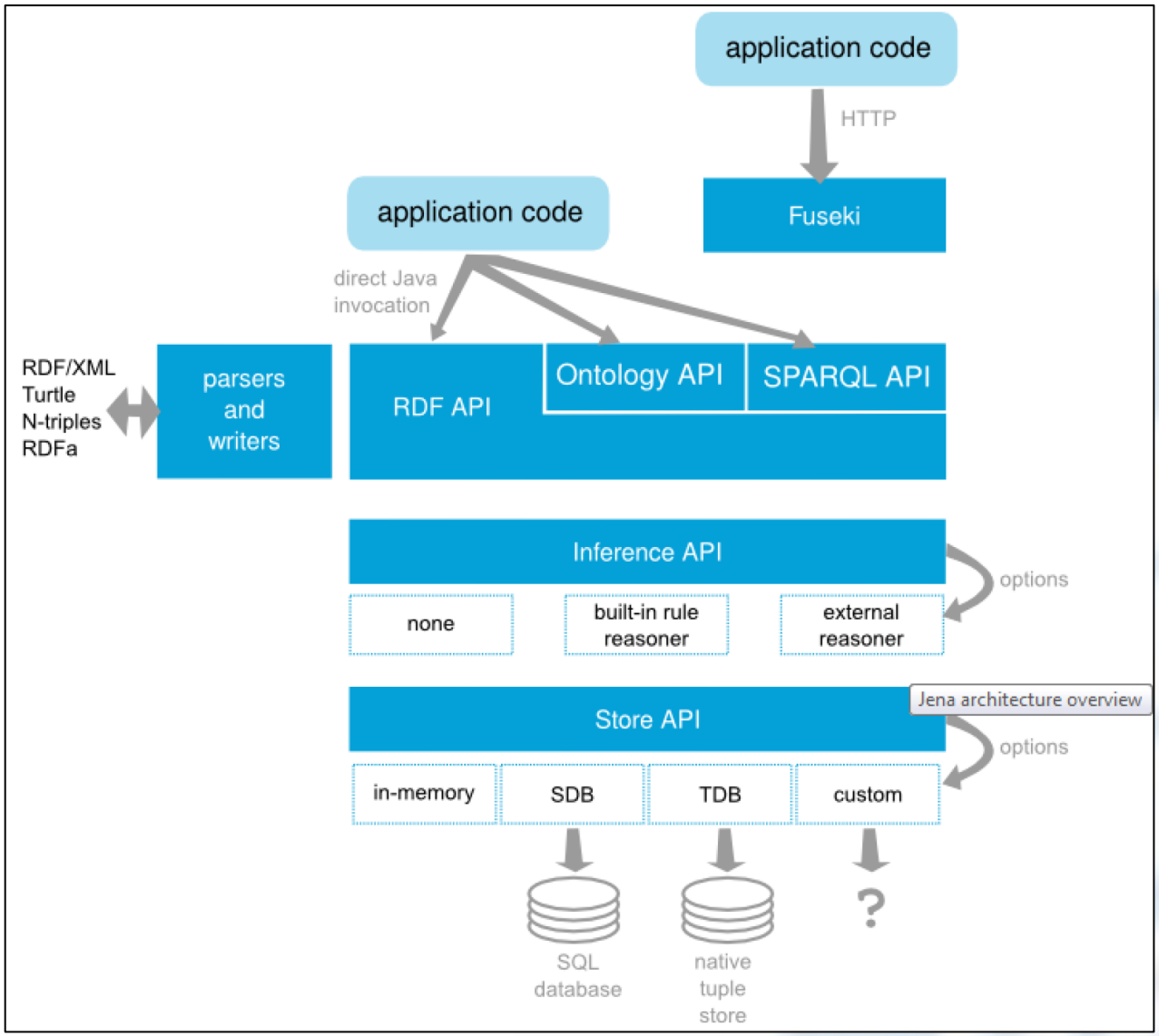
```
INSERT DATA {
  <http://example/libro> PrecioReal “Muy caro” }
DELETE DATA {
  <http://example/libro> PrecioReal “Muy caro” }
```

- **Otras operaciones (según configuración del motor):**
  - **LOAD**
  - **COPY**
  - **MOVE**
  - **CLEAR**

Ejercicio III

- Probad las siguientes consultas en la DBPedia
  - http://dbpedia.org/sparql
  - Podéis investigar sobre la ontología en https://dbpedia.org/fct/
- Buscar películas de “Hayao Miyazaki”
  - https://dbpedia.org/page/Hayao_Miyazaki
- Buscar nombres de actores nacidos en Barcelona y películas en las que han participado, con su director
  - http://dbpedia.org/ontology/birthPlace
  - http://dbpedia.org/ontology/starring
  - http://dbpedia.org/property/occupation
  - http://dbpedia.org/resource/Actor
- ¡No olvidéis limitar a 50 las búsquedas!

Jena: Arquitectura



Jena

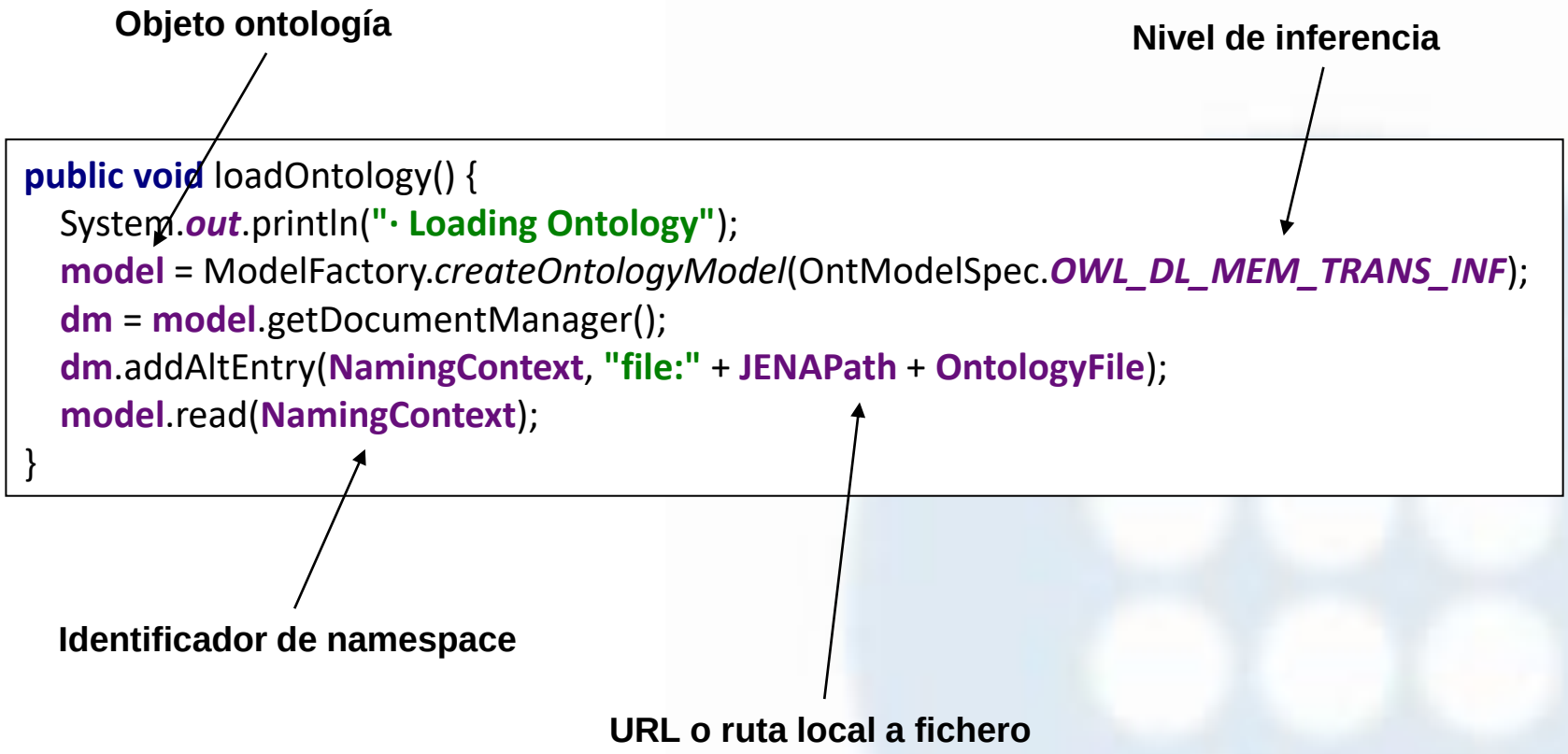
- Proyecto Apache Jena
  - RDF + OWL APIs
  - SPARQL Server + UI (Fuseki)
- Descargad las librerías de las APIs y el código de ejemplo:
  - https://dlcdn.apache.org/jena/binaries/apache-jena-4.4.0.zip
  - https://gitlab.fib.upc.edu/sergio.alvarez-napagao/material-sid/raw/master/jena/JenaTester.java
- Importad los siguientes .jar al CLASSPATH (JDK 11):
  - jena-core-4.4.0
  - jena-base-4.4.0
  - jena-arq-4.4.0
  - jena-iri-4.4.0
  - jena-shaded-guava-4.4.0
  - libthrift-0.15.0
  - slf4j-api-1.7.35
  - commons-lang3-3.12.0

Ejemplos con Jena: JenaTester

```
public static void main(String args[]) throws FileNotFoundException {
   System.out.println("Starting tester...");
   String path = "./";
   String owlFile = "pizza.owl";
   String namespace = "pizza";
   JenaTester tester = new JenaTester(path, owlFile, namespace);

   System.out.println("---------------\n\nLoading ontology");
   tester.loadOntology();
   System.out.println("---------------\n\nGet classes");
   tester.getClasses();
   System.out.println("---------------\n\nGet properties");
   tester.getPropertiesByClass();
   System.out.println("---------------\n\nGet all individuals");
   tester.getIndividuals();
   System.out.println("---------------\n\nGroup individuals by class");
   tester.getIndividualsByClass();
   System.out.println("---------------\n\nRun a SPARQL query about a data property");
   tester.runSparqlQueryDataProperty();
   System.out.println("---------------\n\nRun a SPARQL query about an object property");
   tester.runSparqlQueryObjectProperty();
   System.out.println("---------------\n\nRun a modification using the JENA Ontology API");
   tester.addInstances("FourSeasons");
   System.out.println("---------------\n\nRun a modification via SPARQL");
   tester.runSparqlQueryModify();
   System.out.println("---------------\n\nRe-run the modification via SPARQL");
   tester.runSparqlQueryModify();
   System.out.println("---------------\n\nRelease and save ontology");
   tester.releaseOntology();
   System.out.println("---------------\n\nCheck equivalent class inference");
   tester.testEquivalentClass();
}
```

Cargar una ontología



Niveles de inferencia

OntModelSpec	Language profile	Storage model	Reasoner
OWL_MEM	OWL full	in-memory	none
OWL_MEM_TRANS_INF	OWL full	in-memory	transitive class-hierarchy inference
OWL_MEM_RULE_INF	OWL full	in-memory	rule-based reasoner with OWL rules
OWL_MEM_MICRO_RULE_INF	OWL full	in-memory	optimised rule-based reasoner with OWL rules
OWL_MEM_MINI_RULE_INF	OWL full	in-memory	rule-based reasoner with subset of OWL rules
OWL_DL_MEM	OWL DL	in-memory	none
OWL_DL_MEM_RDFS_INF	OWL DL	in-memory	rule reasoner with RDFS-level entailment-rules
OWL_DL_MEM_TRANS_INF	OWL DL	in-memory	transitive class-hierarchy inference
OWL_DL_MEM_RULE_INF	OWL DL	in-memory	rule-based reasoner with OWL rules
OWL_LITE_MEM	OWL Lite	in-memory	none
OWL_LITE_MEM_TRANS_INF	OWL Lite	in-memory	transitive class-hierarchy inference
OWL_LITE_MEM_RDFS_INF	OWL Lite	in-memory	rule reasoner with RDFS-level entailment-rules
OWL_LITE_MEM_RULES_INF	OWL Lite	in-memory	rule-based reasoner with OWL rules
DAML_MEM	DAML+OIL	in-memory	none
DAML_MEM_TRANS_INF	DAML+OIL	in-memory	transitive class-hierarchy inference
DAML_MEM_RDFS_INF	DAML+OIL	in-memory	rule reasoner with RDFS-level entailment-rules
DAML_MEM_RULE_INF	DAML+OIL	in-memory	rule-based reasoner with DAML rules
RDFS_MEM	RDFS	in-memory	none
RDFS_MEM_TRANS_INF	RDFS	in-memory	transitive class-hierarchy inference
RDFS_MEM_RDFS_INF	RDFS	in-memory	rule reasoner with RDFS-level entailment-rules

Niveles de inferencia

Constructs	Supported by	Notes
rdfs:subClassOf, rdfs:subPropertyOf, rdf:type	all	Normal RDFS semantics supported including meta use (e.g. taking the subPropertyOf subClassOf).
rdfs:domain, rdfs:range	all	Stronger if-and-only-if semantics supported
owl:intersectionOf	all	
owl:unionOf	all	Partial support. If C=unionOf(A,B) then will infer that A,B are subclasses of C, and thus that instances of A or B are instances of C. Does not handle the reverse (that an instance of C must be either an instance of A or an instance of B).
owl:equivalentClass	all	
owl:disjointWith	full, mini	
owl:sameAs, owl:differentFrom, owl:distinctMembers	full, mini	owl:distinctMembers is currently translated into a quadratic set of owl:differentFrom assertions.
Owl:Thing	all	
owl:equivalentProperty, owl:inverseOf	all	
owl:FunctionalProperty, owl:InverseFunctionalProperty	all	
owl:SymmetricProperty, owl:TransitiveProperty	all	
owl:someValuesFrom	full, (mini)	Full supports both directions (existence of a value implies membership of someValuesFrom restriction, membership of someValuesFrom implies the existence of a bNode representing the value). Mini omits the latter "bNode introduction" which avoids some infinite closures.
owl:allValuesFrom	full, mini	Partial support, forward direction only (member of a allValuesFrom(p, C) implies that all p values are of type C). Does handle cases where the reverse direction is trivially true (e.g. by virtue of a global rdfs:range axiom).
owl:minCardinality, owl:maxCardinality, owl:cardinality	full, (mini)	Restricted to cardinalities of 0 or 1, though higher cardinalities are partially supported in validation for the case of literal-valued properties. Mini omits the bNodes introduction in the minCardinality(1) case, see someValuesFrom above.
owl:hasValue	all	

Niveles de inferencia

- La decisión dependerá de las necesidades
  - TRANS_INF es mucho más rápido que RULE_INF
  - MICRO es mucho más rápido que MINI o DL
- Otra opción es usar un razonador externo (e.g. Pellet)
- Documentación
  - https://jena.apache.org/documentation/inference/index.html
  - https://jena.apache.org/documentation/ontology/

Guardar la ontología

```
public void releaseOntology() throws FileNotFoundException {
   System.out.println("· Releasing Ontology");
   if (!model.isClosed()) {
      model.write(new FileOutputStream(JENAPath + File.separator + MODIFIED_PREFIX + OntologyFile, false));
      model.close();
   }
}
```


URL o ruta local a fichero

Listar instancias

```
public void getIndividuals() {
   //List of ontology properties
   for (Iterator i = model.listIndividuals().toList().iterator(); i.hasNext(); ) {
      Individual dummy = (Individual) i.next();
      System.out.println("Ontology has individual: ");
      System.out.println("   " + dummy);
      Property nameProperty = model.getProperty(PIZZA_BASE_URI + "#hasPizzaName");
      RDFNode nameValue = dummy.getPropertyValue(nameProperty);
      System.out.println("   hasPizzaName = " + nameValue);
   }
}
```

```
Get all individuals
Ontology has individual:
   http://www.co-ode.org/ontologies/pizza/pizza.owl#England
   hasPizzaName = null
Ontology has individual:
   http://www.co-ode.org/ontologies/pizza/pizza.owl#France
   hasPizzaName = null
Ontology has individual:
   http://www.co-ode.org/ontologies/pizza/pizza.owl#America
   hasPizzaName = null
Ontology has individual:
   http://www.co-ode.org/ontologies/pizza/pizza.owl#Italy
   hasPizzaName = null
Ontology has individual:
   http://www.co-ode.org/ontologies/pizza/pizza.owl#Germany
   hasPizzaName = null
---------------
```

Agrupar instancias por clase

```
public void getIndividualsByClass() {
   Iterator<OntClass> classesIt = model.listNamedClasses().toList().iterator();
   while (classesIt.hasNext()) {
      OntClass actual = classesIt.next();
      System.out.println("Class: '" + actual.getURI() + "' has individuals:");
      OntClass pizzaClass = model.getOntClass(actual.getURI());
      for (Iterator i = model.listIndividuals(pizzaClass).toList().iterator(); i.hasNext(); ) {
         Individual instance = (Individual) i.next();
         System.out.println("    · " + instance);
      }
   }
}
```

```
Class: 'http://www.co-ode.org/ontologies/pizza/pizza.owl#Country' has individuals:
    · http://www.co-ode.org/ontologies/pizza/pizza.owl#Italy
    · http://www.co-ode.org/ontologies/pizza/pizza.owl#Germany
    · http://www.co-ode.org/ontologies/pizza/pizza.owl#France
    · http://www.co-ode.org/ontologies/pizza/pizza.owl#England
    · http://www.co-ode.org/ontologies/pizza/pizza.owl#America
```

Consultar propiedades

```
public void getPropertiesByClass() {
   Iterator<OntClass> classesIt = model.listNamedClasses().toList().iterator();
   while (classesIt.hasNext()) {
      OntClass actual = classesIt.next();
      System.out.println("Class: '" + actual.getURI() + "' has properties:");
      OntClass pizzaClass = model.getOntClass(actual.getURI());
      //List of ontology properties
      Iterator<OntProperty> itProperties =
pizzaClass.listDeclaredProperties().toList().iterator();

      while (itProperties.hasNext()) {
         OntProperty property = itProperties.next();
         System.out.println("    · Name :" + property.getLocalName());
         System.out.println("      · Domain :" + property.getDomain());
         System.out.println("      · Range :" + property.getRange());
         System.out.println("      · Inverse :" + property.hasInverse());
         System.out.println("      · IsData :" + property.isDatatypeProperty());
         System.out.println("      · IsFunctional :" + property.isFunctionalProperty());
         System.out.println("      · IsObject :" + property.isObjectProperty());
         System.out.println("      · IsSymetric :" + property.isSymmetricProperty());
         System.out.println("      · IsTransitive :" + property.isTransitiveProperty());
      }
   }
}
```

```
Name :hasBase
      · Domain :…#Pizza
      · Range :…#PizzaBase
      · Inverse :false
      · IsData :false
      · IsFunctional :true
      · IsObject :true
      · IsSymetric :false
      · IsTransitive :false
```

Añadir elementos a la ontología

Crear instancia

```
private void addInstances(String className) {
   System.out.println("   Adding instance to '" + className + "'");
   OntClass pizzaClass = model.getOntClass(PIZZA_BASE_URI + "#" + className);
   Individual particularPizza = pizzaClass.createIndividual(PIZZA_BASE_URI + "#" + className + "Instance");

   // Data properties (create and use)
   Property nameProperty = model.createDatatypeProperty(PIZZA_BASE_URI + "#hasPizzaName");
   particularPizza.addProperty(nameProperty, "A yummy" + className);

   // Object property (retrieve and use)
   Individual italy = model.getIndividual(PIZZA_BASE_URI + "#Italy");
   Property countryProperty = model.getObjectProperty(PIZZA_BASE_URI + "#hasCountryOfOrigin");
   particularPizza.addProperty(countryProperty, italy);
}
```

Crear función

Crear relación

Crear tripleta

Consultar con SPARQL

```
public void runSparqlQueryModify() {
   String queryString = "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> " +
         "PREFIX pizza: <http://www.co-ode.org/ontologies/pizza/pizza.owl#> " +
         "SELECT ?Pizza ?eaten where {" +
         "?Pizza a ?y. " +
         "?y rdfs:subClassOf pizza:Pizza. " +
         "Optional {?Pizza pizza:eaten ?eaten}}";

   Query query = QueryFactory.create(queryString);

   QueryExecution qe = QueryExecutionFactory.create(query, model);
   ResultSet results = qe.execSelect();

   for (Iterator iter = results; iter.hasNext(); ) {
      ResultBinding res = (ResultBinding) iter.next();
      Object Pizza = res.get("Pizza");
      Object eaten = res.get("eaten");
      if (eaten == null) {
         System.out.println("Pizza = " + Pizza + " <-> false");
         Individual actualPizza = model.getIndividual(Pizza.toString());
         Property eatenProperty = model.createDatatypeProperty(PIZZA_BASE_URI + "#eaten");
         Literal rdfBoolean = model.createTypedLiteral(Boolean.valueOf("true"));
         actualPizza.addProperty(eatenProperty, rdfBoolean);
      } else {
         System.out.println("Pizza = " + Pizza + " <-> " + eaten);
      }
   }
   qe.close();
}
```

```
Pizza = …#FourSeasonsInstance <-> false
Pizza = …#FourSeasonsInstance <-> true
```

Usar diferentes niveles de inferencia

```
private void testEquivalentClass() {
   System.out.println("· Loading Ontology");
   model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM_MICRO_RULE_INF);
   dm = model.getDocumentManager();
   dm.addAltEntry(NamingContext, "file:" + JENAPath + File.separator + MODIFIED_PREFIX + OntologyFile);
   model.read(NamingContext);
   Individual instance = model.getIndividual(PIZZA_BASE_URI + "#FourSeasonsInstance");
   boolean isRealItalian = instance.hasOntClass(PIZZA_BASE_URI + "#RealItalianPizza");
   boolean isSpicy = instance.hasOntClass(PIZZA_BASE_URI + "#SpicyPizza");
   System.out.println("FourSeasonsInstance classifies as RealItalianPizza?: " + isRealItalian);
   System.out.println("FourSeasonsInstance classifies as SpicyPizza?: " + isSpicy);
   model.close();
}
```

```
FourSeasonsInstance classifies as RealItalianPizza?: true
FourSeasonsInstance classifies as SpicyPizza?: false
```

Ejercicio IV

- Probad que JenaTester funciona correctamente y os genera una ontología modificada
  - Abrid la ontología modificada en Protégé y comprobad los cambios
- Programáticamente, añadir clases a la ontología:
  - Añadir queso Sistemas
  - Añadir base Inteligente
  - Añadir carne Distribuidos
- Añadir instancias a la ontología:
  - De la base, carne y quesos nuevos
  - De una pizza SID que usa esos ingredientes
- Usando una inferencia “RULE”, comprobad a qué clases equivalentes clasifica
  - Podéis usar `Individual.listOntClasses(false)`

Referencias

- **https://www.w3.org/TR/2013/REC-sparql11-query-20130321/**
- https://www.w3.org/TR/2013/REC-sparql11-update-20130321/
- https://jena.apache.org/documentation/inference/
- https://jena.apache.org/documentation/ontology/