Ontologies et Web Sémantique

Les Ontologies – API Jena Apache

Plan du cours

1. Introduction
2. Manipuler des triplets RDF
3. Lire et écrire des modèles RDF
4. Manipuler des graphes RDF
5. Utiliser SPARQL dans Jena API
6. Manipuler des ontologies

Introduction

- Jena est une API Java pour les applications du web sémantiques :
  - Manipuler des triplets RDF.
  - Lire et créer des documents RDF/XML.
  - Interprète SPARQL.
  - Gestion des ontologies: RDF-Schema, DAML + OIL, OWL.
  - …
- Open Source - The Apache Software Foundation, Licensed under the Apache License.
- A free and open source Java framework for building Semantic Web and Linked Data applications.
- Home : https://jena.apache.org/

Introduction

- **Compilation/Exécution**

- Pré-requis : Java JDK +IDE

- Télécharger Apache Jena : https://jena.apache.org/download/index.cgi

- Les fichiers jar requis pour compiler et exécuter un programme Java à l'aide de Jena sont disponibles dans le dossier lib.

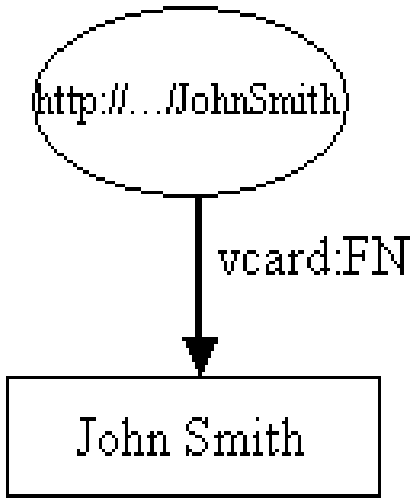
- Ils doivent être spécifiés dans le CLASSPATH du projet.

- Javadoc : http://jena.apache.org/documentation/javadoc/jena/

- Tutoriels : http://jena.apache.org/tutorials/index.html

Manipuler des triplets RDF

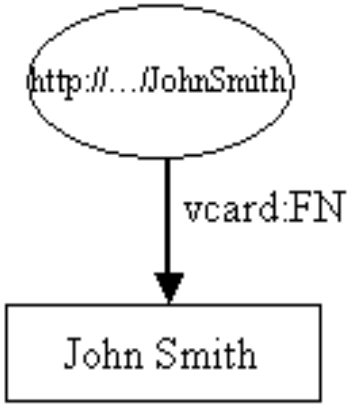
- Jena offre des classes pour représenter des triplets/graphes RDF, des ressources, des propriétés et des littéraux.
- Les interfaces représentant les ressources, les propriétés et les littéraux sont respectivement nommées **Resource**, **Property** et **Literal**.
- Dans Jena, un graphe est appelé un modèle et est représenté par l'interface **Model**.
- Exemple :



VCARD définit un modèle pour représenter les "cartes de visite": prénom, nom, téléphone, e-mail, etc.

Manipuler des triplets RDF

- Exemple :




```
// quelques définitions
static String personURI = "http://somewhere/JohnSmith";
static String fullName = "John Smith";

// créer un modèle vide
Model model = ModelFactory.createDefaultModel();

// créer la ressource
Resource johnSmith = model.createResource(personURI);

// ajouter la propriété
johnSmith.addProperty(VCARD.FN, fullName);
```

Manipuler des triplets RDF

- Créer un graphe RDF :

  ModelFactory est une fabrique à modèles (i.e. graphes RDF) .

  - *createDefaultModel* crée un graphe RDF standard en mémoire.
  - *createFileModelMaker* crée un graphique sur le disque.
  - *createOntologyModel* crée une ontologie (RDF Schema, etc.)
  - . . .

- Ajouter une ressource
  - *createResource* ajoute une ressource au modèle. La ressource créée est renvoyée.

- Ajouter une propriété
  - *addProperty* ajoute une propriété à une ressource. La ressource créée est renvoyée.

- Resource and Literal sont des subclasses de RDFNode.

Manipuler des triplets RDF

- Le code pour créer la ressource et ajouter la propriété peut être écrit d'une manière plus compacte avec un style en cascade :

```
// quelques définitions
static String personURI = "http://somewhere/JohnSmith";
static String fullName = "John Smith";

// créer un modèle vide
Model model = ModelFactory.createDefaultModel();


Resource johnSmith =
            model.createResource(personURI);
                  .addProperty(VCARD.FN, fullName);
                    .addProperty(VCARD.TITLE, "Officer");
```

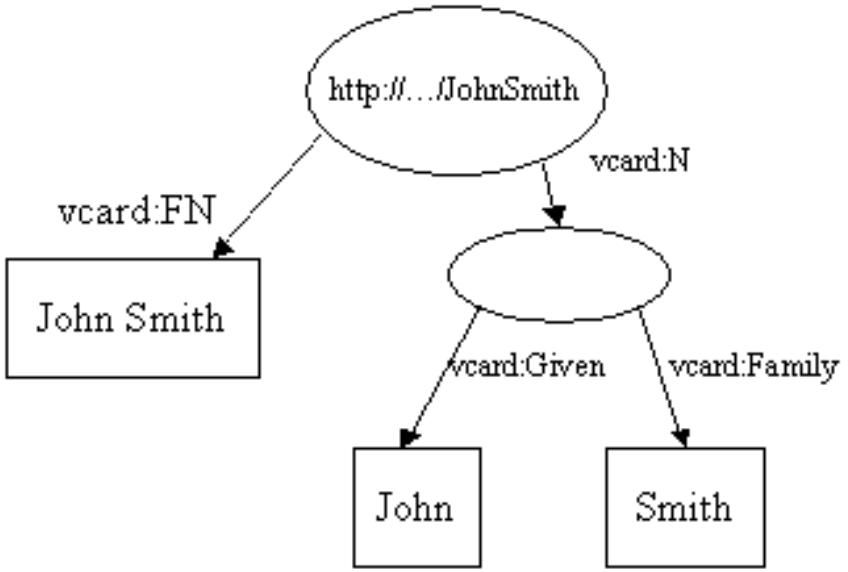
Manipuler des triplets RDF

- Jena fournit les propriétés (classes constantes) des schémas suivants:

  - **RDF** (com.hp.hpl.jena.vocabulary.RDF)
    - Ex: RDF.Bag, RDF.predicate
  - **RDF-Schema** (com.hp.hpl.jena.vocabulary.RDFS)
    - Ex: RDFS.Class, RDFS.subClassOf
  - **VCARD** (com.hp.hpl.jena.vocabulary.VCARD)
    - Ex: VCARD.FN, VCARD.BDAY
  - **Dublin Core** (com.hp.hpl.jena.vocabulary.DC)
    - Ex: DC.creator, DC.description

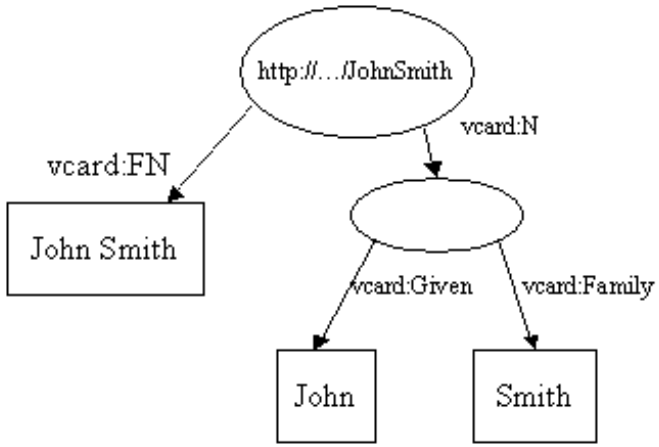
Manipuler des triplets RDF

- Exemple : - Nœud vide



Manipuler des triplets RDF

- Exemple : - Nœud vide




```
// quelques définitions
String personURI = "http://somewhere/JohnSmith";
String fullName = "John Smith";
String familyName = "Smith";
String fullName = givenName + " " + familyName;

// créer un modèle vide
Model model = ModelFactory.createDefaultModel();

Resource johnSmith = model.createResource(personURI)
         .addProperty(VCARD.FN, fullName)
         .addProperty(VCARD.N, model.createResource()
                  .addProperty(VCARD.Given, givenName)
                  .addProperty(VCARD.Family, familyName));
```

Manipuler des triplets RDF

- Un modèle RDF est représenté comme un ensemble de déclarations, appelées *Statement*.
- Chaque appel à *addProperty* ajoute un autre statement au modèle.
- Les interfaces Model de Jena définissent une méthode listStatements(), qui retourne un StmtIterator, un sous-type de la classe Java Iterator sur toutes les déclarations dans un modèle.
- *StmtIterator* possède une méthode nextStatement(), qui retourne la déclaration suivante de l'itérateur.
- L'interface *Statement* fournit des méthodes d'accès au sujet, au prédicat et à l'objet du statement.
- getSubject(), getPredicate(), et getObject().

Manipuler des triplets RDF

```
// lister des statements dans le modèle
StmtIterator iter = model.listStatements();

// afficher l'objet, le prédicat et le sujet
while (iter.hasNext()) {
        Statement stmt = iter.nextStatement();
        Resource subject = stmt.getSubject();
        Property predicate = stmt.getPredicate();
        RDFNode object = stmt.getObject();

        System.out.print(subject.toString());
        System.out.print(" " + predicate.toString() + " ");

        if (object instanceof Resource) {
                System.out.print(object.toString());
        } else {
          System.out.print(" \"" + object.toString() + "\"");
        }
        System.out.println(" .");
}
```

Manipuler des triplets RDF

```
// liste des déclarations dans le modèle
StmtIterator iter = model.listStatements();

// affiche l'objet, le prédicat et le sujet de chaque déclaration
while (iter.hasNext()) {
    Statement stmt      = iter.nextStatement();  // obtenir la prochaine déclaration
    Resource  subject   = stmt.getSubject();     // obtenir le sujet
    Property  predicate = stmt.getPredicate();   // obtenir le prédicat
    RDFNode   object    = stmt.getObject();      // obtenir l'objet

    System.out.print(subject.toString());
    System.out.print(" " + predicate.toString() + " ");
    if (object instanceof Resource) {
       System.out.print(object.toString());
    } else {
        // l'objet est un littéral
        System.out.print(" \"" + object.toString() + "\"");
    }

    System.out.println(" .");
}
```

Lire et écrire des modèles RDF

- Jena dispose de méthodes pour lire et écrire du RDF/XML.
- Elles peuvent être utilisées pour sauvegarder un modèle RDF dans un fichier et le relire plus tard.
- **Ecrire** - La méthode *write(…)* dans la classe Model :

```
write(java.io.OutputStream out, java.lang.String lang, java.lang.String base)
```

- out : le flux de sortie dans lequel le RDF est écrit.
- lang : le language vers lequel le RDF devrait être écrit. RDF/XML (par défaut), RDF/XML-ABBREV, N3.
- base : L'uri de base à utiliser lors de l'écriture des URI relatives. *null* signifie n'utiliser que des URI absolus.

Lire et écrire des modèles RDF

- Exemple : flux de sortie standard

```
model.write(System.out);
```

- Output

```
<rdf:RDF
   xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
   xmlns:vcard='http://www.w3.org/2001/vcard-rdf/3.0#' >

 <rdf:Description rdf:about='http://somewhere/JohnSmith'>
       <vcard:FN>John Smith</vcard:FN>
       <vcard:N rdf:nodeID="A0"/>
 </rdf:Description>
 <rdf:Description rdf:nodeID="A0">
       <vcard:Given>John</vcard:Given>
       <vcard:Family>Smith</vcard:Family>
 </rdf:Description>
</rdf:RDF>
```

Lire et écrire des modèles RDF

- Exemple : flux de sortie standard

```
model.write(System.out, "RDF/XML-ABBREV");
```

- Résultat :

```
<rdf:RDF
   xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
   xmlns:vcard='http://www.w3.org/2001/vcard-rdf/3.0#' >

 <rdf:Description rdf:about='http://somewhere/JohnSmith'>
       <vcard:FN>John Smith</vcard:FN>
       <vcard:N rdf:parseType="Resource">
              <vcard:Given>John</vcard:Given>
              <vcard:Family>Smith</vcard:Family>
       </vcard:N>
 </rdf:Description>
</rdf:RDF>
```

Lire et écrire des modèles RDF

- Jena dispose de méthodes pour lire et écrire du RDF/XML.
- Elles peuvent être utilisées pour sauvegarder un modèle RDF dans un fichier et le relire plus tard.
- **Lire** - La méthode *read(…)* dans la classe Model :
  - read (java.lang.String url) : lit le modèle à partir d'un document XML.
  - read (java.lang.String url, java.lang.String lang) : lit le modèle à partir d'un document XML dans un langage spécifique.
  - read (java.io.InputStream in, java.lang.String base, java.lang.String lang) : lit le modèle à partir d'un flux d'entrée dans un langage spécifique, utilisant une URI de base.

Lire et écrire des modèles RDF

- Exemple 1:

```
Model model = ModelFactory.createDefaultModel();

model.read("file:///home/moi/example.rdf", "RDF/XML");
```

- Exemple 2 :

```
Model model = ModelFactory.createDefaultModel();

InputStream in = FileManager.get().open("example.rdf");

if (in != null)
   model.read(in, null);
```

Manipuler des graphes RDF

- Créer / accéder aux ressources RDF :
- *Model.createResource* (String uri) et *Model.getResource* (String uri) renvoient la ressource créée ou la ressource de l'URI spécifié.
- *Resource.getProperty* (Property) renvoie un Statement (ou NULL). il n'en renvoie qu'un seul même s'il en existe plusieurs.
- Exemple :

```
String johnSmithURI = "http://somewhere/JohnSmith/";
Resource vcard=model.createResource(johnSmithURI);
Resource name=vcard.getProperty(VCARD.N).getResource();
```

- - Statement.getResource renvoie l'objet Ressource.
- - Statement.getString renvoie l'objet Literal.

Manipuler des graphes RDF

- Accéder aux propriétés :
- Il est possible qu'une propriété puisse apparaître plusieurs fois.
- La méthode *Resource.listProperties*(Property p) peut être utilisée pour renvoyer un Itérateur qui les liste toutes.
- Exemple :

```
// lister tous les emails
StmtIterator iter = vcard.listProperties(VCARD.EMAIL);
while (iter.hasNext()) {
       System.out.println(" " +
            iter.nextStatement().getObject().toString());
}
```

Manipuler des graphes RDF

- Parcourir un graphe RDF:

- *Model.listStatements* renvoie un itérateur qui liste tous les statements du modèle.

- *Model.listStatements*(Ressource s, Propriété p, RDFNode o) renvoie un itérateur qui liste certains statements spécifiques dans le modèle.

- *listSubjects* renvoie un ResIterator qui liste toutes les ressources qui sont le sujet d'un certain statement.

- *listSubjectsWithProperty* (Propriété p, RDFNode o) renvoie un ResIterator sur toutes les ressources ayant la propriété p avec la valeur o.

Manipuler des graphes RDF

- Parcourir un graphe RDF:
- Méthode générique pour sélectionner des statements :
- *listStatements*(Selector). Le **Selector** définit les statements à parcourir.
- *SimpleSelector*(sujet, prédicat, objet), une implémentation de l'interface Selector, permet de sélectionner certains statements.
- Exemple: sélectionner toutes les ressources avec une propriété VCARD.FN dont la valeur se termine par "Smith".

```
StmtIterator it = model.listStatements(
    new SimpleSelector(null, VCARD.FN, (RDFNode) null) {
        public boolean selects(Statement s){
            return s.getString().endsWith("Smith");
        }
    });
```

Manipuler des graphes RDF

- Opérations sur les Models:

- *Model.union*(Model) renvoie un nouveau modèle, le modèle transmis en tant que paramètre fusionné avec le modèle actuel.

- Les ressources ayant le même URI seront fusionnées.

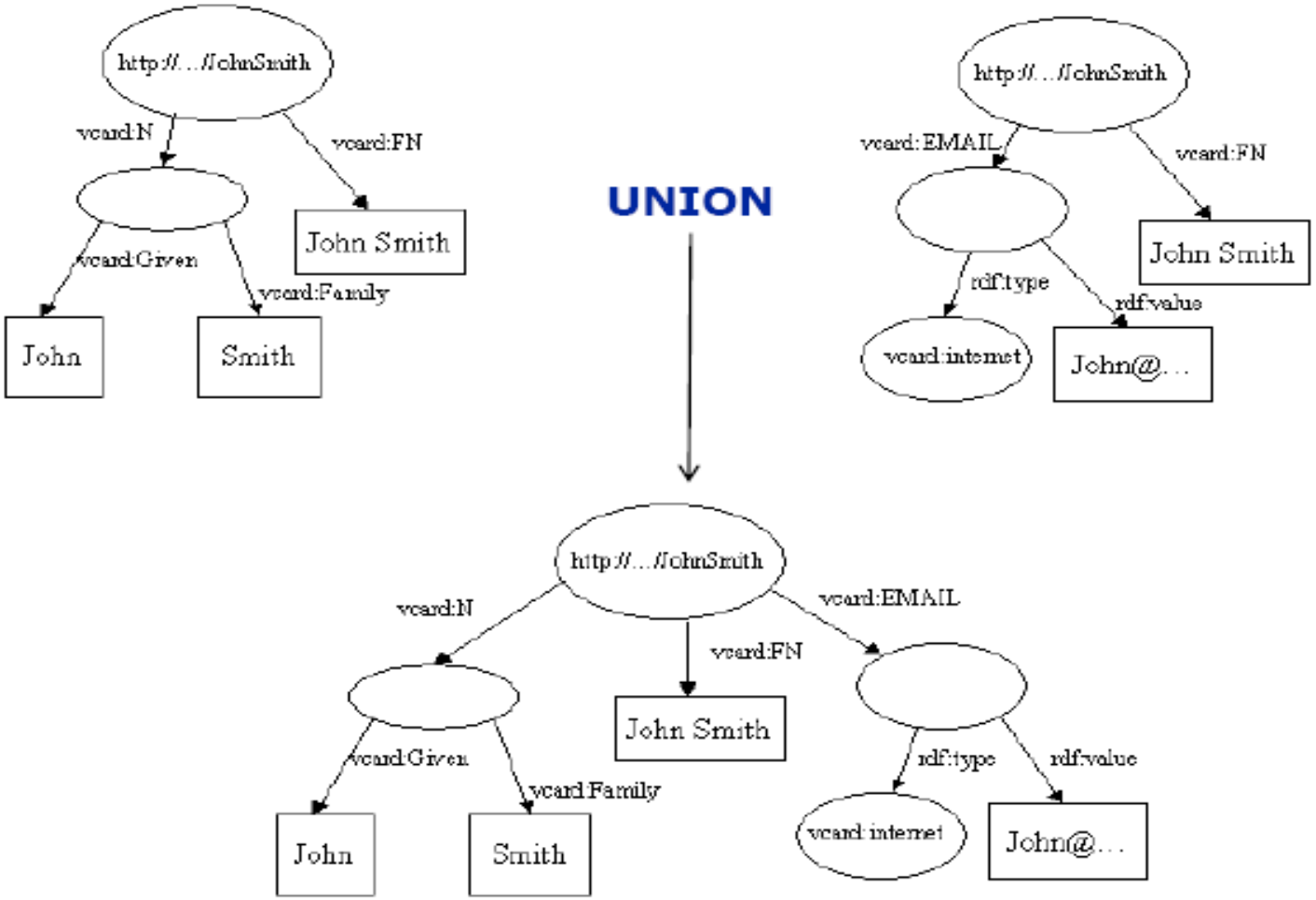
- *Model.intersection*(Model) renvoie un nouveau modèle contenant uniquement des déclarations communes.

- *Model.difference*(Model) renvoie un nouveau modèle contenant uniquement les déclarations qui se trouvent dans le modèle actuel mais pas dans celui qui a été passé en paramètre.

Manipuler des graphes RDF

- Opérations sur les Models:

- Exemple : Union



Manipuler des graphes RDF

- Containers:

- Jena prend en charge les trois types de conteneurs:
  - BAG: la méthode createBag
  - ALT: la méthode createAlt
  - SEQ: la méthode createSeq

- Ces méthodes peuvent être appelées:
  - Sans paramètres: Création d'un conteneur anonyme.
  - Avec un paramètre String: Création d'un conteneur identifié via une URI.

- Méthodes supportées sur les conteneurs: add, contains, size, etc.

Manipuler des graphes RDF

- Containers:

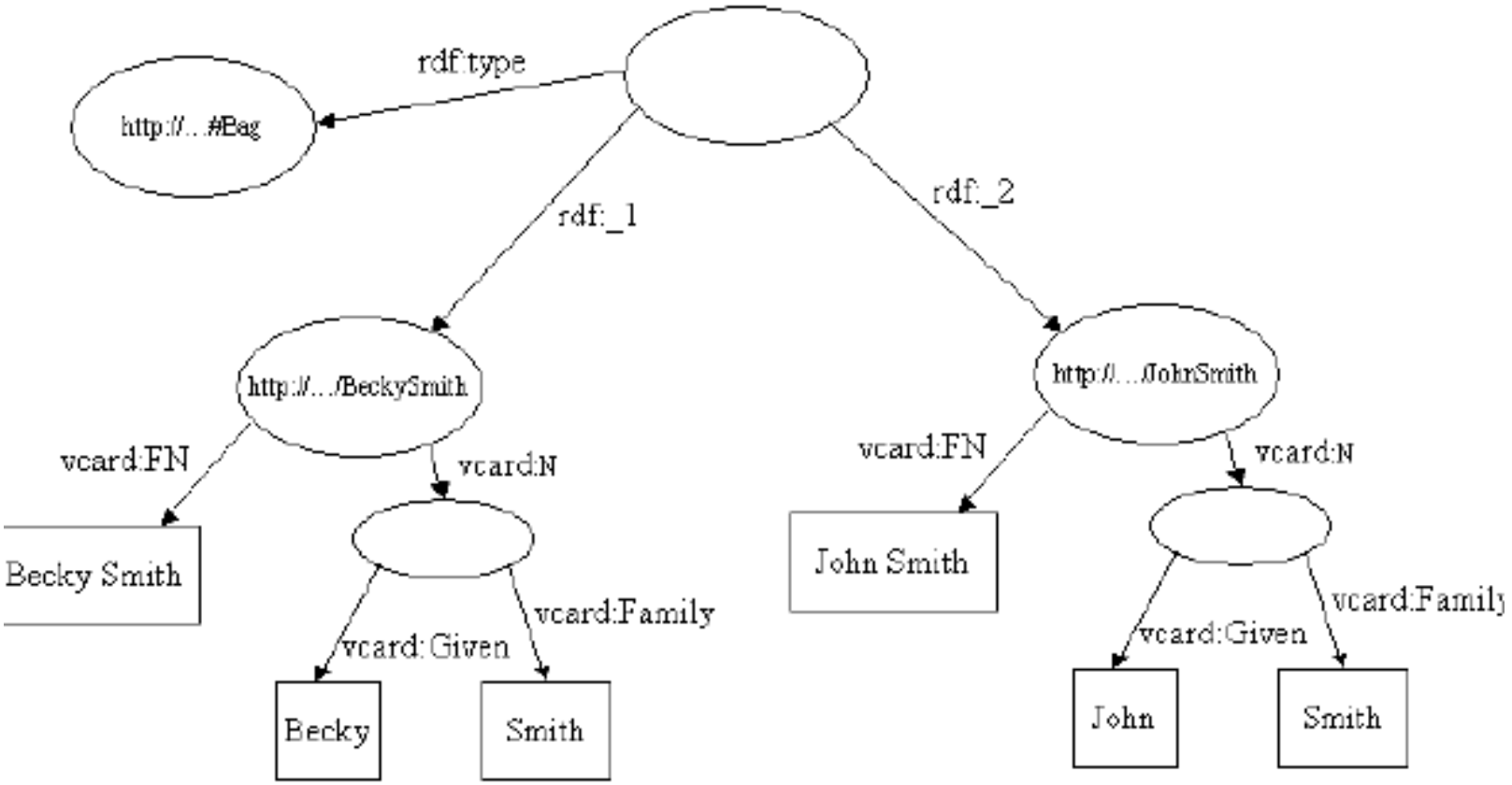
- Exemple : un Bag contenant les vcard des Smiths

```
Bag smiths = model.createBag();

// Select toutes les resources avec un VCARD.FN property
// pour qui sa valeur ends with "Smith"
StmtIterator iter = model.listStatements(
   new SimpleSelector(null, VCARD.FN, (RDFNode) null) {
         public boolean selects(Statement s) {
            return s.getString().endsWith("Smith");
         }
   });

// ajouter les Smiths au bag
while (iter.hasNext()) {
   smiths.add(iter.nextStatement().getSubject());
}
```

Manipuler des graphes RDF

- Containers:

- Exemple : un Bag contenant les vcard des Smiths



Utiliser SPARQL dans Jena API

1. Création d'une requête à l'aide de la méthode QueryFactory.create (String query), Où query est le texte de la requête SPARQL.

2. Création d'une *QueryExecution* à l'aide de la méthode QueryExecutionFactory.create(query, model).

3. Appeler execSelect sur l'exécution de requête créée. Cette méthode renvoie un *ResultSet*.

4. Parcourir le *ResultSet* en utilisant les méthodes hasNext et nextSolution, qui retournent une *QuerySolution*.

   QuerySolution propose les méthodes getResource et getLiteral avec les noms de variables SELECT comme paramètres.

5. Terminer l'exécution de *QueryExecution* en utilisant *close*.

Utiliser SPARQL dans Jena API

```
import java.util.*; import com.hp.hpl.jena.rdf.model.*;
import com.hp.hpl.jena.util.iterator.*;
import com.hp.hpl.jena.util.*; import com.hp.hpl.jena.query.*;
public class sparql {
    public static void main(String args[]) {
        Model model = FileManager.get().loadModel("file:personnes.n3");
1       String queryString = "PREFIX m: <http://exemple.fr/mv#> " +
                             "SELECT ?uri ?nom ?photo WHERE { "?uri m:nom ?nom . " +
                             "OPTIONAL { ?uri m:photo ?photo . } . }";
2       Query query = QueryFactory.create(queryString) ;
        QueryExecution qexec=QueryExecutionFactory.create(query,model);
        try {
3           ResultSet results = qexec.execSelect() ;
            for ( ; results.hasNext() ; ) {
4               QuerySolution soln = results.nextSolution() ;
                Resource uri = soln.getResource("uri") ;
                Resource photo = soln.getResource("photo") ;
                Literal nom = soln.getLiteral("nom") ;
                System.out.print(uri.toString()+" "+ nom.toString()+" ");
                if (photo != null) System.out.print(photo.toString());
                System.out.println();
            }
5       } finally { qexec.close(); };
```

Manipuler des Ontologies

- Jena peut gérer les ontologies (RDF-Schema, DAML + OIL, OWL) et nous pouvons raisonner en utilisant les connaissances offertes par l'ontologie.

- Une ontologie est un type spécial de modèle (c'est-à-dire OntModel) offrant certaines méthodes spécifiques aux ontologies:

- Créer une classe: la méthode *createClass* qui retourne une OntClass (OntClass est une spécialisation de Resource).

- Créer une propriété: la méthode createObjectProperty qui renvoie une ObjectProperty (ObjectProperty est une spécialisation de Resource).

Manipuler des Ontologies

- Créer une ontologie:

- La méthode *ModelFactory.createOntologyModel* utilisant comme paramètre:

✓ OntModelSpec.RDFS_MEM: Pour une ontologie RDF-S en mémoire sans raisonnement.
✓ OntModelSpec.RDFS_MEM_RDFS_INF: Pour une ontologie RDF-S en mémoire, avec raisonnement
✓ etc.

- Généralement, l'ontologie est chargée à partir d'un flux: en utilisant la méthode read.

- Les éléments d'une ontologie (OntClass, ObjectProperty) sont des spécialisations de la classe OntResource qui définissent les méthodes:

- getComment, setComment, getLabel, setLabel, getSeeAlso, setSeeAlso, getIsDefinedBy, setIsDefinedBy, etc.

Manipuler des Ontologies

- Exemple 1 : créer une classe

```
String exns = "http://www.exemple.com/voc#";
OntClass veh = m.createClass(exns + "Vehicle");
OntClass car= m.createClass(exns + "Car");
car.addSuperClass(veh);
```

- Exemple 2 : lire une classe

```
OntClass cl = m.getOntClass(exns + "Vehicle");
for (ExtendedIterator i =
                          cl.listSubClasses();i.hasNext();)
{
      OntClass c = (OntClass) i.next();
      System.out.print(c.getLocalName() + " ");
}
```

Manipuler des Ontologies

- ObjectProperty:
- Quelques méthodes :
- [has, list, set, add] : subProperty, superProperty, domain, range, etc.

```
OntClass engine= m.createClass(exns + "Engine");

ObjectProperty pcomp = m.createObjectProperty(exns
+"composant");
ObjectProperty pengine = m.createObjectProperty(exns
+"engine");
pengine.addSuperProperty(pcomp);
pengine.addDomain(veh);
pengine.addRange(engine);
```

- *createDatatypeProperty* pour les propriétés avec des littéraux dans le domaine.

Manipuler des Ontologies

- Ontologies et faits:

- Communément, l'ontologie est stockée dans un (ou plusieurs) fichiers (RDFS par ex.) Et les connaissances factuelles dans d'autres (RDF par exemple) ...

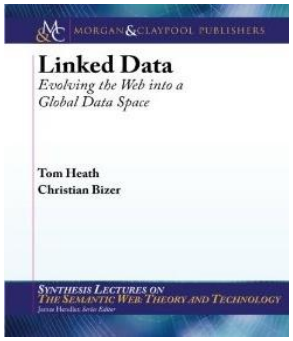
- Dans de tels cas, nous devons:
  - Créer un modèle pour l'ontologie.
  - Créer un modèle pour les faits.
  - Créer un modèle pour les inférences (une union des deux)
  - Utilisez le modèle qui permet l'inférence.

Manipuler des Ontologies

- Exemple:

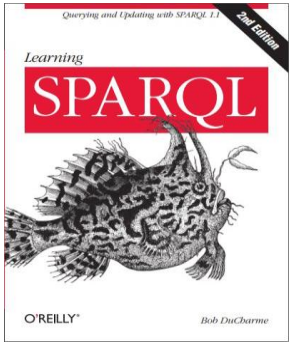
```
Model schema =
   FileManager.get().loadModel("file:explerdfs/rdfs.rdf");

Model data =
   FileManager.get().loadModel("file:explerdfs/rdf.rdf");

InfModel infmodel = ModelFactory.createRDFSModel(schema,
                                                 data);
Resource res =
     infmodel.getResource("http://www.exemple.com/smith");

System.out.println(res);
```

Références



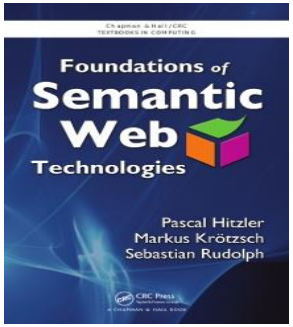
Linked Data: Evolving the Web into a Global Data Space

- ✓ Auteur : Christian Bizer, Tom Heath
- ✓ Éditeur : Morgan & Claypool Publishers
- ✓ Edition : Février 2011 - 136 pages - ISBN 9781608454310



Learning SPARQL : Querying and Updating with SPARQL

- ✓ Auteur : Bob DuCharme
- ✓ Éditeur : O'Reilly Media
- ✓ Edition: Juillet 2013– 386pages -ISBN : *9781449306595*



Foundations of Semantic Web Technologies

- ✓ Auteur : Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph
- ✓ Éditeur : CRC Press/Chapman and Hall
- ✓ Edition : 2009 - 455 pages - ISBN : 9781420090505

Références

- Jena Apache API
  - ✓ http://jena.apache.org/tutorials/rdf_api.html
- Cours - Knowledege Management
  - ✓ http://www-inf.it-sudparis.eu/~gaaloulw/KM/
- Introduction au RDF et à l'API RDF de Jena
  - ✓ https://web-semantique.developpez.com/tutoriels/jena/introduction-rdf/
- Noy et McGuinness - Ontology Development 101: A Guide to Creating Your First Ontology.
  - ✓ https://protege.stanford.edu/publications/ontology_development/ontology101.pdf/
- INRIA MOOC - Fabien Gandon– Web Sémantique et Web de Données
  - ✓ https://www.canal-u.tv/producteurs/inria/cours_en_ligne/web_semantique_et_web_de_donnees