Semantic Annotation for Persistence, the Reference Manual

by
Stephen L. Reed, Texai.org

Introduction

Semantically annotated Java domain entities can be persisted to, and loaded from the Sesame2 RDF store. Introduced in Java 1.5, annotations permit the tagging of classes, fields and methods with data that can be queried by the application at runtime. Java Enterprise Edition and the Java Persistence API use annotations to designate the object-relational mapping (ORM) between Java objects and a relational database. Texai elaborates a subset of these ORM annotations to perform mapping between Java objects and the Sesame2 RDF store.

Sesame

The Sesame RDF Store is available at OpenRDF.org. The Texai framework is compatible with Sesame version 2. The Texai download contains the support jars required to run a local Sesame server. To access the Sesame server through its web based RDF Workbench, the OpenRDF.org download for Sesame2 is required.

Semantic Annotation of a Simple POJO

Consider a simple Plain Old Java Object (POJO):

```
public class MyCar {
  String manufacturer;
  . . .
}
```

Let's apply semantic annotations that describe MyCar as a semantic web (RDF) class:

```
@RDFEntity(
namespaces={
  @RDFNamespace(prefix="cyc", namespaceURI=Constants.CYC_NAMESPACE)},
subject="cyc:MyCar", subClassOf="cyc:Automobile", context="cyc:MyContext")
public class MyCar {
  private String manufacturer;
  . . .
}
```

These annotations define MyCar as a class in the *Cyc* RDF ontology, having the superclass *Automobile* and the context *MyContext*. In addition to required semantic annotation at the class level, this framework also requires an ID field and annotation of each persistent field:

```
@RDFEntity(
namespaces={
  @RDFNamespace(prefix="cyc", namespaceURI=Constants.CYC_NAMESPACE)},
subject="cyc:MyCar", subClassOf="cyc:Automobile", context="cyc:MyContext")
public class MyCar {
  @Id
  private URI id;
```

```
   @RDFProperty
  private String manufacturerName;
  . . .
}
```

These annotations define *id* as the id field, and *manufacturerName* as a field to be persisted for instances of MyCar. RDFEntityManager is the framework class for persisting and loading RDF entities. To construct an RDFEntityManager instance we need a Sesame repository instance:

```
    Repository repository = new SailRepository(new NativeStore(
      new File("path-to-my-repository-directory"),
      "spoc,posc"));
    try {
      repository.initialize();
    } catch (RepositoryException ex) {
      ex.printStackTrace();
    }
    rdfEntityManager = new RDFEntityManager(repository);
```

Assuming that a constructor for MyCar accepts manufacturerName as its parameter, we can construct and persist a MyCar instance as follows:

```
   MyCar myCar = new MyCar(“Toyota”);
   rdfEntityManager.persist(myCar);
```

If cyc:MyCar is not already a defined RDF class in the Sesame RDF repository, then its defining statements will be automatically added from the @RDFEntity annotation on the MyCar Java class. Then statements will be added that define the myCar instance and the value of its manufacturerName field. During its first persistence operation, the Java instance will be given an Id value. RDF entities may be retrieved from the Sesame RDF store by ID, by a field value, or by an iterator over all instances of a given class. To retrieve by field value for example:

```
   URI predicate = rdfEntityManager.createURI(
     Constants.CYC_NAMESPACE + “manufacturerName”);
   Value rdfValue =  rdfEntityManager.getValueFactory()
     .createLiteral(“Toyota”);
   List<MyCar> someCars = rdfEntityManager.find(
      predicate,
      rdfValue,
      MyCar.class);
```

@RDFEntity Annotation

This annotation is applied to the java entity class or super class, and has the following properties:

- **namespace** - This optional property defines the default RDF namespace prefix for unqualified term names.
- **subject** - This optional property defines the name of the associated RDF class and defaults to the qualified java class name prefixed by the default namespace. It is used chiefly to map the annotated java class to an existing RDF class.
- **type** - This optional property defines the types of this entity. Each value must specify an existing RDF class. This property should be specified for a new java class.

- **subClassOf** - This optional property defines the RDF super classes of this entity. Each value must specify an existing RDF class. This property should be specified for a new java class.
- **context** – This optional property defines the context in which the RDF statements are stored. If omitted, the persistence statements are not associated with a context.

```
@RDFEntity(subject=Constants.TEXAI_NAMESPACE + "Friend",
subClassOf={"Person"}, namespace=Constants.FOAF_NAMESPACE,
context=Constants.TEXAI_NAMESPACE + "TestContext")
public class Friend extends Person { ...
```

@javax.persistence.Id Annotation
This annotation is adopted from the Java Persistence API and is applied to the field that contains the RDF entity identifier. The supported field types are:
- java.lang.String
- java.net.URI
- org.openrdf.model.URI

```
@Id
private String termId;
```

@RDFProperty Annotation
This annotation is applied to the java entity fields and specifies the manner of the mapping between values of a particular field and the RDF statements that persist the value that the field contains. Its properties are:
- **namespace** - This optional property defines the RDF namespace prefix for predicates derived from the name of the annotated field. When present, it takes precedence over the @RDFEntity namespace annotation.
- **name** - This optional property defines the RDF predicate that is mapped to this association. The default value is the name of the annotated field prefixed by the namespace.
- **subPropertyOf** - This optional property defines the RDF predicates for which this predicate is a specialization. Each value must be an existing RDF predicate.
- **domain** - This optional property defines the type of the subject for this predicate. The value must specify an existing class in the RDF store. The default value is the class of the domain object that contains the field, prefixed by the namespace.
- **range** - This optional property defines the type of the objects for this predicate. The value must specify an existing class in the RDF store. Unqualified values are automatically prefixed with the namespace. The default value of this property is the class of field's value prefixed by the namespace, or the corresponding XML schema datatype when applicable.
- **inverse** - This property defines whether the predicate is an inverse predicate with respect to the annotated field, in which case the field value is mapped to the domain of the predicate and the java object is mapped to the range of the predicate. The default value is "**false**".
- **fetch** - This optional property is adopted from the Java Persistence API and defines whether the value of the field or property should be lazily loaded or must be eagerly fetched. At present it is implemented only for collection valued fields. Because loading all the collection field values of a persisted java entity can be time consuming, lazy loading is the more efficient alternative if the the loading effort can be postponed until the application actually requires the field values. The allowed values are:
  - **javax.persistence.FetchType.LAZY**, which is the default
  - **javax.persistence.FetchType.EAGER**.
- **trueClass** – This optional property is only applicable for a boolean field. It defines the name of

the RDF class of which this object is an instance when the boolean association holds true.

- **falseClass** – This optional property is only applicable for a boolean field. It defines the name of the RDF class of which this object is an instance when the boolean association holds false.

Examples from the sample FOAF Agent class.

```
  /** the name of this agent  */
  @RDFProperty
  private String name;

  /** the birthday of this Agent, represented in mm-dd string form,
eg. '12-31'  */
  @RDFProperty
  private String birthday;

  /** the gender of this Agent (typically but not necessarily 'male'
or 'female')  */
  @RDFProperty
  private String gender;

  /** the unordered list of some things that were made by this agent
*/
  @RDFProperty(predicate="made")
  private List<Object> thingsMade;
```

Example from the sample Friend class.

```
 /** extends the FOAF ontology to provide date of birth */
  @RDFProperty(subPropertyOf=Constants.TEXAI_NAMESPACE +
"hasAttribute", range="Person")
  private Date dateOfBirth;
```

Persisting An RDF Entity

RDF entities are persisted and loaded by an instance of RDFEntityManager.

- **persist** – This method persists the given RDF entity instance as propositions in the KB. If the RDF entity class is new, then propositions are created to define it in the KB. The parameter is:
    - **rdfEntity** – an instance of Java **Object**; the RDF entity

This method returns void.

RDF Entity Persistence Example

Three Java classes are defined in this sample to illustrate common semantic annotations and their use in an inheritance hierarchy. Semantic annotations are highlighted with a bold font. First is an annotated RDF entity class that represents an Agent in the Friend Of A Friend (FOAF) ontology:

```
package org.texai.kb.persistence.sample;

import java.util.Set;
import javax.persistence.Id;
import org.texai.kb.Constants;
import org.texai.kb.persistence.RDFEntity;
import org.texai.kb.persistence.RDFNamespace;
import org.texai.kb.persistence.RDFProperty;

/** Provides a class to represent an agent in the FOAF ontology.
 *
 * @author reed
 */
@RDFEntity(
namespaces={
  @RDFNamespace(prefix="foaf", namespaceURI=Constants.FOAF_NAMESPACE)},
subject="foaf:Agent")
public class Agent {

  /** This is the URI string that identifies this person.  If null, then it
is automatically generated when
   * this instance is persisted.  When a subclass is instantiated, this
field is ignored. */
  @Id
  private String agentId;

  /** the name of this agent  */
  @RDFProperty(predicate="foaf:name")
  private String name;

  /** the birthday of this Agent, represented in mm-dd string form, eg.
'12-31'  */
  @RDFProperty(predicate="foaf:birthday")
  private String birthday;

  /** the gender of this Agent (typically but not necessarily 'male' or
'female')  */
  @RDFProperty(predicate="foaf:gender")
  private String gender;

  /** the set of some things that were made by this agent  */
  @RDFProperty(predicate="foaf:made")
  private Set<Object> thingsMade;

  /** Creates a new instance of Agent. RDF Entities must have a
   * default constructor.
   */
  public Agent() {
  }

  public String getAgentId() {
    return agentId;
  }

  public void setAgentId(final String agentId) {
    this.agentId = agentId;
  }

  public String getName() {
    return name;
  }
```

```
public void setName(final String name) {
  this.name = name;
}

public String getBirthday() {
  return birthday;
}

public void setBirthday(final String birthday) {
  this.birthday = birthday;
}

public String getGender() {
  return gender;
}

public void setGender(final String gender) {
  this.gender = gender;
}

public Set<Object> getThingsMade() {
  return thingsMade;
}

public void setThingsMade(final Set<Object> thingsMade) {
  this.thingsMade = thingsMade;
}

/** Returns whether the given object is equal to this object.
 *
 * @param obj the given object
 * @return whether the given object is equal to this object
 */
@Override
public boolean equals(final Object obj) {
  if (obj instanceof Agent) {
    final Agent that = (Agent) obj;
    return this.getAgentId().equals(that.getAgentId());
  } else {
    return false;
  }
}

/** Returns a hash code for this object.
 *
 * @return a hash code for this object
 */
@Override
public int hashCode() {
  if (getAgentId() == null) {
    return super.hashCode();
  } else {
    return getAgentId().hashCode();
    }
  }

}
```

The second Java class defines the FOAF Person which is a subclass of FOAF Agent:

```
package org.texai.kb.persistence.sample;

import org.texai.kb.Constants;
import org.texai.kb.persistence.RDFEntity;
import org.texai.kb.persistence.RDFNamespace;
import org.texai.kb.persistence.RDFProperty;

/** Provides a class to represent a person in the FOAF ontology.
 *
 * @author reed
 */
@RDFEntity(
namespaces={
  @RDFNamespace(prefix="foaf", namespaceURI=Constants.FOAF_NAMESPACE)},
subject="foaf:Person")
public class Person extends Agent {

  /** the first name of this person  */
  @RDFProperty(predicate="foaf:firstName")
  private String firstName;

  /** the family name of this person  */
  @RDFProperty(predicate="foaf:familyName")
  private String familyName;

  /** Creates a new instance of Person. The default constructor must be
    * present.
    */
  public Person() {
  }

  public String getFirstName() {
    return firstName;
  }

  public void setFirstName(final String firstName) {
    this.firstName = firstName;
  }

  public String getFamilyName() {
    return familyName;
  }

  public void setFamilyName(final String familyName) {
    this.familyName = familyName;
  }
  /** Returns whether the given object is equal to this object.
   *
   * @param obj the given object
   * @return whether the given object is equal to this object
   */
  @Override
  public boolean equals(final Object obj) {
    if (obj instanceof Person) {
      final Person that = (Person) obj;
      return this.getAgentId().equals(that.getAgentId());
    } else {
      return false;
    }
  }

  /** Returns a hash code for this object.
   *
   * @return a hash code for this object
   */
  @Override
  public int hashCode() {
    if (getAgentId() == null) {
      return super.hashCode();
    } else {
      return getAgentId().hashCode();
    }
  }

}
```

And the third Java class for Friend represents an extension to the FOAF ontology:

```
package org.texai.kb.persistence.sample;

import java.util.Date;
import java.util.Set;
import org.texai.kb.Constants;
import org.texai.kb.persistence.RDFEntity;
import org.texai.kb.persistence.RDFNamespace;
import org.texai.kb.persistence.RDFProperty;

/** Extends the FOAF ontology to represent a friend.
 *
 * @author reed
 */
@RDFEntity(
namespaces={
  @RDFNamespace(prefix="texai", namespaceURI=Constants.TEXAI_NAMESPACE),
  @RDFNamespace(prefix="foaf", namespaceURI=Constants.FOAF_NAMESPACE)},
subject="texai:Friend", subClassOf="foaf:Person",
context="texai:FriendContext")
// RDF entity classes must not have the final modifier.
public class Friend extends Person {
  /** extends the FOAF ontology to provide date of birth */
  @RDFProperty(predicate="texai:dateOfBirth",
subPropertyOf="texai:hasAttribute", range="foaf:Person")
  private Date dateOfBirth;

  /**
   * Creates a new Friend instance. RDF entities are required to have
   * a default (no-parameter) constructor among their constructors.
   */
  public Friend() {
  }

  /** Creates a new Friend instance. RDF entities are required to have
   * a no-parameter constructor among their constructors.
   */
  public Friend(
          final String name,
          final String birthday,
          final String gender,
          final Set<Object> thingsMade,
          final String firstName,
          final String familyName,
          final Date dateOfBirth) {
    setName(name);
    setBirthday(birthday);
    setGender(gender);
    setThingsMade(thingsMade);
    setFirstName(firstName);
    setFamilyName(familyName);
    setDateOfBirth(dateOfBirth);
  }

  /** Returns a string representation of this object.
   *
   * @return a string representation of this object
   */
  public String toString() {
    return "[friend: " + getName() + "]";
  }

  public Date getDateOfBirth() {
    return dateOfBirth;
  }

  public void setDateOfBirth(final Date dateOfBirth) {
    this.dateOfBirth = dateOfBirth;
  }

  /** Returns whether the given object is equal to this object.
   *
   * @param obj the given object
   * @return whether the given object is equal to this object
   */
  @Override
  public boolean equals(final Object obj) {
    if (obj instanceof Friend) {
      final Friend that = (Friend) obj;
      return this.getAgentId().equals(that.getAgentId());
    } else {
      return false;
    }
  }

  /** Returns a hash code for this object.
   *
   * @return a hash code for this object
   */
  @Override
  public int hashCode() {
    if (getAgentId() == null) {
      return super.hashCode();
    } else {
      return getAgentId().hashCode();
    }
  }

}
```

Below is a Java code snippet that creates and persists an instance object of the Friend class.

```
    rdfEntityManager.setAutoCommit(false);
    // populate the Friend
    final String name = "Stephen Reed";
    final String birthday = "09-20";
    final String gender = "male";
    final Set<Object> thingsMade = new HashSet<Object>();
    thingsMade.add(rdfEntityManager.getValueFactory().createURI("http://tex
ai.org"));
    final String firstName = "Stephen";
    final String familyName = "Reed";
    final Date dateOfBirth = (new GregorianCalendar(1951, 9, 20, 0, 0,
0)).getTime();
    final Friend friend = new Friend(
            name,
            birthday,
            gender,
            thingsMade,
            firstName,
            familyName,
            dateOfBirth);

    // persist the RDF entity and commit the transaction
    rdfEntityManager.persist(friend);
    rdfEntityManager.commit();
```

As a result, here are the knowledge base assertions created when the Friend instance was persisted for the first time.

```
1. <texai:Friend> <rdf:type> <cyc:FirstOrderCollection>
2. <texai:Friend> <rdfs:subClassOf> <http://xmlns.com/foaf/0.1/Person>
3. <texai:Friend_7c990ce2-bda4-4031-b4ce-c4dd07aebe4c> <rdf:type>
   <texai:Friend>
4. <texai:Friend_7c990ce2-bda4-4031-b4ce-c4dd07aebe4c>
   <texai:domainEntityClassName> “org.texai.kb.persistence.sample.Friend”
5. <texai:dateOfBirth> <rdf:type> <owl:ObjectProperty>
6. <texai:dateOfBirth> <rdfs:subPropertyOf> <cyc:conceptuallyRelated>
7. <texai:dateOfBirth> <rdfs:domain> <texai:Friend>
8. <texai:dateOfBirth> <rdfs:range> <http://xmlns.com/foaf/0.1/Person>
9. <texai:Friend_7c990ce2-bda4-4031-b4ce-c4dd07aebe4c>
   <texai:dateOfBirth> “1951-10-20T00:00:00-06:00”
10.<texai:Friend_7c990ce2-bda4-4031-b4ce-c4dd07aebe4c>
   <http://xmlns.com/foaf/0.1/name> “Stephen Reed”
11.<texai:Friend_7c990ce2-bda4-4031-b4ce-c4dd07aebe4c>
   <http://xmlns.com/foaf/0.1/firstName> “Stephen”
12.<texai:Friend_7c990ce2-bda4-4031-b4ce-c4dd07aebe4c>
   <http://xmlns.com/foaf/0.1/birthday> “09-20”
13.<texai:Friend_7c990ce2-bda4-4031-b4ce-c4dd07aebe4c>
   <http://xmlns.com/foaf/0.1/familyName> “Reed”
14.<texai:Friend_7c990ce2-bda4-4031-b4ce-c4dd07aebe4c>
   <http://xmlns.com/foaf/0.1/made> <http://texai.org>
15.<texai:Friend_7c990ce2-bda4-4031-b4ce-c4dd07aebe4c>
   <http://xmlns.com/foaf/0.1/gender> “male”
```

RDF Entity Load Examples

RDF entities may be loaded by an iterator. Below is a Java code snippet that loads an instance object of the Friend class from an iterator over all the persisted instances in the Sesame RDF store. This snippet omits the transaction handling statements.

```
// load via an iterator
final Iterator friend_iter =
  rdfEntityManager.rdfEntityIterator(Friend.class);
final Friend friend1 = (Friend) friend_iter.next();
System.out.println("loaded via iterator: " + friend1);
```

RDF entities may be loaded by their term id. Below is a Java code snippet that loads an Friend instance object given its term id. This snippet omits the transaction handling statements.

```
// load via friend id
final URI friendURI rdfEntityManager.getValueFactory().createURI(
  “texai:Friend_7c990ce2-bda4-4031-b4ce-c4dd07aebe4c”);
final Friend friend2 =
  (Friend) rdfEntityManager.find(Friend.class, friendURI);
System.out.println("loaded via URI: " + friendId + " --> " + friend2);
```

RDF entities may be loaded by specifying a value object for an identifying property. Below is a Java code snippet that loads an Friend instance object given the String value for the myString identifying property. An identifying property is a functional property that has only one RDF entity associated with a unique value object. This snippet omits the transaction handling statements.

```
// load via identifying property value
```

```
    final URI predicate = rdfEntityManager.getValueFactory().createURI(
      "http://xmlns.com/foaf/0.1/name");
    final Value value =
      rdfEntityManager.getValueFactory().createLiteral("Stephen Reed");
    final List<Object> friends =
      rdfEntityManager.find(predicate, value, Friend.class);
    assert friends.size() == 1 : "friends must have size 1";
    final Friend friend3 = (Friend) friends.get(0);
    System.out.println(
      "loaded via an identifying property: " +
      value.toString() + " --> " + friend3);
```

RDF Entity Update Example

RDF entities may be updated after loading them. Below is a Java code snippet that loads an instance object of the Friend class from an iterator, and then persists the instance after modification.

```
    // load the friend
    final URI friendURI =
      rdfEntityManager.getValueFactory().createURI(friendId);
    Friend friend =
      (Friend) rdfEntityManager.find(Friend.class, friendURI);
    System.out.println("loaded via URI: " + friendId + " --> " + friend);
    friend.setName("Stephen L. Reed");

    // persist the friend
    rdfEntityManager.persist(friend);
    friendId = friend.getAgentId();

    // reload the friend
    friend = (Friend) rdfEntityManager.find(Friend.class, friendURI);
    System.out.println("reloaded via URI: " + friendId + " --> " + friend);
```

RDF Entity Removal Example

RDF entities may be removed after loading them. Below is a Java code snippet that persists a new instance object of the Friend class, and then removes the instance after reloading it.

```
    // load the friend
    final URI friendURI =
      rdfEntityManager.getValueFactory().createURI(friendId);
    Friend friend =
      (Friend) rdfEntityManager.find(Friend.class, friendURI);
    System.out.println("loaded via URI: " + friendId + " --> " + friend);
    friend.setName("Stephen L. Reed");

    // remove the friend and commit the transaction
    friendId = friend.getAgentId();
    rdfEntityManager.remove(friend);
    rdfEntityManager.commit();

    // attempt to load the friend
    friend = (Friend) rdfEntityManager.find(Friend.class, friendURI);
    if (friend == null) {
      System.out.println("as expected, cannot load a removed entity via
URI: " + friendId);
    } else {
      System.out.println("error, loaded a removed entity via URI: " +
friendId + " --> " + friend);
    }
```

References

OpenRDF Sesame - http://openrdf.org/

OpenCyc – http://www.opencyc.org

WordNet – WordNet a lexical database for the English language http://wordnet.princeton.edu

Java Persistence API - The Java Persistence API - A Simpler Programming Model for Entity Persistence http://java.sun.com/developer/technicalArticles/J2EE/jpa

Texai – http://sf.net/projects/texai

Java Source Code and Examples

Texai subversion (version control) repository - http://texai.svn.sourceforge.net/viewvc/texai