Java source files of project Keltsi

Antti Hätinen, Arno Aalto, Merja Jalava, Tuomo Kajava, Petri Lindgren

27th August 2002

Contents

1 fi.helsinki.cs.keltsi.jenardf

1.1 Advertisement

```
package fi.helsinki.cs.keltsi.jenardf;

import com.hp.hpl.mesa.rdf.jena.model.*;
import com.hp.hpl.jena.daml.*;

import java.util.LinkedList;
import java.util.Iterator;

/*
 *  Class to encapsulate an Advertisement,
 *  or perhaps an Advertiser.
 *  Holds zero or more services to advertise.
 */
public class Advertisement extends KeltsiResource {
    private String label;

    /**
     *  Constructor.
     *  Mostly accesed via Service.getAdvertisement().
     */
    Advertisement(DAMLInstance advertisement) {
        super(advertisement);
    }

    public ImageFile getImageFile() {
        try {
            DAMLInstance imageFileNode = (DAMLInstance)
                    getResource().getPropertyValue(Keltsi.imageFile);
            return new ImageFile(imageFileNode);
        } catch (Exception ex) {}
        return null;
    }

    public Iterator getServices() {
        LinkedList services = new LinkedList();
        NodeIterator iter = getResource()
                .getPropertyValues(Keltsi.services);
        try {
            while (iter.hasNext()) {
                services.add(getModel()
                        .getKeltsiResource((DAMLInstance)iter.next()));
            }
        } catch (Exception ex) {
            try {
                iter.close();
            } catch(Exception exx) {}
        }
        return services.iterator();
    }
```

```
    public String getLabel() {
        if (label == null) {
            label = getSingleLiteralValue(Keltsi.label);
        }
        return label;
    }

}
```

1.2 Contact

```
package fi.helsinki.cs.keltsi.jenardf;

import com.hp.hpl.mesa.rdf.jena.model.*;
import com.hp.hpl.jena.daml.*;

/*
 * Encapsulates a contact of a Service.
 * Should never be  constructed directly.
 * Accessed via method Service.getContacts().
 */
public class Contact extends KeltsiResource {
    private String address = "";
    private String email = "";
    private String fax = "";
    private String telephone = "";
    private String url = "";
    private String label = null;
    private String openingHours = null;

    Contact(DAMLInstance contact) {
        super(contact);
        address = getSingleLiteralValue(Keltsi.address);
        email = getSingleLiteralValue(Keltsi.email);
        fax = getSingleLiteralValue(Keltsi.fax);
        telephone = getSingleLiteralValue(Keltsi.telephone);
        url = getSingleLiteralValue(Keltsi.url);
        label = getSingleLiteralValue(Keltsi.label);
        openingHours = getSingleLiteralValue(Keltsi.openingHours);
    }

    public boolean hasLabel() {
        return (label != null);
    }

    public String getLabel() {
        return label;
    }

    public String getAddress() {
        return address;
    }

    public String getEmail() {
```

```
        return email;
    }

    public String getFax() {
        return fax;
    }

    public String getTelephone() {
        return telephone;
    }

    public String getUrl() {
        return url;
    }

    public String getOpeningHours() {
        return openingHours;
    }

}
```

1.3 ImageFile

```
package fi.helsinki.cs.keltsi.jenardf;

import com.hp.hpl.jena.daml.*;
import com.hp.hpl.mesa.rdf.jena.model.*;

/*
 *  Class to encapsulate an ImageFile.
 *  Now the properties are not very many.
 */
public class ImageFile extends KeltsiResource {
    private String path = "";

    /*
     * Constructor, don't call directly.
     * Acces via Advertisement.getImageFile()
     */
    ImageFile(DAMLInstance imageFile) {
        super(imageFile);
        path = getSingleLiteralValue(Keltsi.path);
    }

    public String path() {
        return path;
    }
}
```

1.4 Keltsi

```
package fi.helsinki.cs.keltsi.jenardf;
```

```
import com.hp.hpl.jena.daml.DAMLModel;
import com.hp.hpl.jena.daml.DAMLClass;
import com.hp.hpl.jena.daml.DAMLProperty;
import com.hp.hpl.jena.daml.DAMLInstance;
import com.hp.hpl.mesa.rdf.jena.model.RDFException;
import com.hp.hpl.mesa.rdf.jena.model.Resource;
import com.hp.hpl.jena.vocabulary.DAML_OIL;
import com.hp.hpl.mesa.rdf.jena.vocabulary.RDF;
import com.hp.hpl.mesa.rdf.jena.vocabulary.RDFS;

import java.util.Iterator;

/**
 *  Defines the vocabulary for the Keltsi-project.
 *  Static instances of resources and properies are provided.
 *  Adds statements to the model to implement
 *  the transitivity and reflexiveness of RDFS.subPropertyOf
 *  and the implications on RDF.type.
 *  The latter is done just to speed things.
 *  The former is done to be able to express queries in
 *  RDQL.
 */
public class Keltsi{

    protected static final String uri =
            "http://protege.stanford.edu/Keltsi#";

    /** returns the URI for this schema
     * @return the URI for this schema
     */
    public static String getURI()
    {
        return uri;
    }

    public static       KeltsiClass ServiceRoot;
    public static       KeltsiClass ProductRoot;

           static final String   sService = "Service";
    public static       DAMLClass Service;
           static final String   sProduct = "Product";
    public static       DAMLClass Product;
           static final String   sAdvertisement = "Advertisement";
    public static       DAMLClass Advertisement;
           static final String   sContact = "Contact";
    public static       DAMLClass Contact;
           static final String   sImageFile = "ImageFile";
    public static       DAMLClass ImageFile;

           static final String   smissingAd = "missingAd";
    public static       DAMLInstance missingAd;


    /* Properties of Service classes */
```

```
       static final String   ssynonym = "synonym";
public static       DAMLProperty synonym;

/* Properties of Service instances */
       static final String scontact = "contact";
public static       DAMLProperty contact;
       static final String   srecommends = "recommends";
public static       DAMLProperty recommends;
       static final String   sadditionalServices = "additionalServices";
public static       DAMLProperty additionalServices;
       static final String   sprovides = "provides";
public static       DAMLProperty provides;

/* Superpoperties of Service instances */
       static final String sproductClassProperty = "productClassProperty";
public static       DAMLProperty productClassProperty;
       static final String sserviceClassProperty = "serviceClassProperty";
public static       DAMLProperty serviceClassProperty;
       static final String sserviceProperty = "serviceProperty";
public static       DAMLProperty serviceProperty;
       static final String sliteralProperty = "literalProperty";
public static       DAMLProperty literalProperty;

/* Properties of Contact */
       static final String saddress = "address";
public static       DAMLProperty address;
       static final String stelephone = "telephone";
public static       DAMLProperty telephone;
       static final String semail = "email";
public static       DAMLProperty email;
       static final String sfax = "fax";
public static       DAMLProperty fax;
       static final String surl = "url";
public static       DAMLProperty url;
       static final String simageFile = "imageFile";
public static       DAMLProperty imageFile;
       static final String   sopeningHours = "openingHours";
public static       DAMLProperty openingHours;

/* Properties of ImageFile */
       static final String   spath = "path";
public static       DAMLProperty path;

/* Properties of Advertisement */
       static final String   sservices = "service";
public static       DAMLProperty services;
       static final String   slabel = "label";
public static       DAMLProperty label;
       static final String   sdescription = "description";
public static       DAMLProperty description;

/* Properties to implement transitivity and reflexiveness */
       static final String   ssubClassOf = "subClassOf";
public static       DAMLProperty subClassOf;
```

```
        static final String   ssubPropertyOf = "subPropertyOf";
public static       DAMLProperty subPropertyOf;
        static final String   stype = "type";
public static       DAMLProperty type;
        static final String   sinstanceCount = "instanceCount";
public static       DAMLProperty instanceCount;

public static void initialize(KeltsiDAML model){
    try {
    Service              = (DAMLClass)model.getDAMLValue(uri
                                + sService);
    Product              = (DAMLClass)model.getDAMLValue(uri
                                + sProduct);
    Advertisement        = (DAMLClass)model.getDAMLValue(uri
                                + sAdvertisement);
    Contact              = (DAMLClass)model.getDAMLValue(uri
                                + sContact);
    ImageFile            = (DAMLClass)model.getDAMLValue(uri
                                + sImageFile);
    synonym              = (DAMLProperty)model.getDAMLValue(uri
                                + ssynonym);
    contact              = (DAMLProperty)model.getDAMLValue(uri
                                + scontact);
    productClassProperty = (DAMLProperty)model.getDAMLValue(uri
                                + sproductClassProperty);
    serviceClassProperty = (DAMLProperty)model.getDAMLValue(uri
                                + sserviceClassProperty);
    serviceProperty      = (DAMLProperty)model.getDAMLValue(uri
                                + sserviceProperty);
    literalProperty      = (DAMLProperty)model.getDAMLValue(uri
                                + sliteralProperty);
    address              = (DAMLProperty)model.getDAMLValue(uri
                                + saddress);
    telephone            = (DAMLProperty)model.getDAMLValue(uri
                                + stelephone);
    email                = (DAMLProperty)model.getDAMLValue(uri
                                + semail);
    fax                  = (DAMLProperty)model.getDAMLValue(uri
                                + sfax);
    url                  = (DAMLProperty)model.getDAMLValue(uri
                                + surl);
    imageFile            = (DAMLProperty)model.getDAMLValue(uri
                                + simageFile);
    path                 = (DAMLProperty)model.getDAMLValue(uri
                                + spath);
    services             = (DAMLProperty)model.getDAMLValue(uri
                                + sservices);
    label                = (DAMLProperty)model.getDAMLValue(uri
                                + slabel);
    openingHours         = (DAMLProperty)model.getDAMLValue(uri
                                + sopeningHours);
    recommends           = (DAMLProperty)model.getDAMLValue(uri
                                + srecommends);
    provides             = (DAMLProperty)model.getDAMLValue(uri
```

```
                                + sprovides);
    additionalServices   = (DAMLProperty)model.getDAMLValue(uri
                                + sadditionalServices);
    description          = (DAMLProperty)model.getDAMLValue(uri
                                + sdescription);

    subClassOf           = model.createDAMLProperty(uri + ssubClassOf);
    subPropertyOf        = model.createDAMLProperty(uri + ssubPropertyOf);
    type                 = model.createDAMLProperty(uri + stype);
    instanceCount        = model.createDAMLProperty(uri + sinstanceCount);

    ServiceRoot = (KeltsiClass)model.getKeltsiResource(Service);
    ProductRoot = (KeltsiClass) model.getKeltsiResource(Product);

/**
 *  Suppose the following statements are in the initial model.
 *  <ul>
 *  <li>(&lt;anObject>, &lt;RDF.type>, &lt;aClass>)</li>
 *  <li>(&lt;aClass>, &lt;RDFS.subClassOf>, &lt;bClass>)</li>
 *  <li>(&lt;bClass>, &lt;RDFS.subClassOf>, &lt;cClass>)</li>
 *  </ul>
 *  The following statements are added:
 *  <ul>
 *  <li> subClassOf is reflexive:
 *  <ul>
 *  <li>(&lt;aClass>, &lt;Keltsi.subClassOf>, &lt;aClass>) <i>new!</i></li>
 *  <li>(&lt;bClass>, &lt;Keltsi.subClassOf>, &lt;bClass>) <i>new!</i></li>
 *  <li>(&lt;cClass>, &lt;Keltsi.subClassOf>, &lt;cClass>) <i>new!</i></li>
 *  </ul>
 *  <li> subClassOf is transitive:
 *  <ul>
 *  <li>(&lt;aClass>, &lt;Keltsi.subClassOf>, &lt;bClass>) <i>new!</i></li>
 *  <li>(&lt;aClass>, &lt;Keltsi.subClassOf>, &lt;cClass>) <i>copy</i></li>
 *  <li>(&lt;bClass>, &lt;Keltsi.subClassOf>, &lt;cClass>) <i>copy</i></li>
 *  </ul>
 *  <li> type follows transitivity of subClassOf
 *  <ul>
 *  <li>(&lt;anObject>, &lt;Keltsi.type>, &lt;aClass>) <i>copy</i></li>
 *  <li>(&lt;anObject>, &lt;Keltsi.type>, &lt;bClass>) <i>new!</i></li>
 *  <li>(&lt;anObject>, &lt;Keltsi.type>, &lt;cClass>) <i>new!</i></li>
 *  </ul>
 *  </ul>
 *
 */
    Iterator classes = model.listDAMLClasses();
    while(classes.hasNext()) {
        DAMLClass thisClass = (DAMLClass) classes.next();
        //System.err.println("Instance: " + thisClass);
        if (model.isAClass(thisClass)) {
            // subClassOf is a reflexive property
            model.add(model.createStatement(
                    thisClass, subClassOf, thisClass));
        }
        Iterator superClasses = thisClass.getSuperClasses(true);
```

```
            while(superClasses.hasNext()) {
                Resource superClass = (Resource) superClasses.next();
                if (model.isAClass(superClass)) {
                    // subClassOf is a transitive property
                    model.add(model.createStatement(
                            thisClass, subClassOf, superClass));
                }
            }
        }

        /* Add statements (i, k:type, s) if (i, type, s) or
                                            (i, type, c) and (c, subClassOf, s)
        */
        Iterator instances = model.listDAMLInstances();
        while(instances.hasNext()) {
            DAMLInstance thisInstance = (DAMLInstance) instances.next();
            //System.err.println("Instance: " + thisInstance);
            Iterator rdfTypes = thisInstance.getRDFTypes(true);
            while(rdfTypes.hasNext()) {
                Resource rdfType = (Resource) rdfTypes.next();
                if (model.isAClass(rdfType)) {
                    // subClassOf is a transitive property
                    model.add(model.createStatement(
                            thisInstance, type, rdfType));
                }
            }
        }

        missingAd = model.createDAMLInstance(Advertisement, uri + smissingAd);
        missingAd.addProperty(RDFS.label, "no advertiser");
        missingAd.addProperty(Keltsi.label, "no advertiser");

        }catch(Exception ex) {
            ex.printStackTrace();
        }
    }

}
```

1.5 KeltsiClass

```
package fi.helsinki.cs.keltsi.jenardf;

import com.hp.hpl.jena.daml.*;
import java.util.Iterator;
import com.hp.hpl.mesa.rdf.jena.model.*;
import com.hp.hpl.mesa.rdf.jena.vocabulary.RDF;
import com.hp.hpl.mesa.rdf.jena.vocabulary.RDFS;
import java.util.LinkedList;
import java.util.Collections;
import java.util.Comparator;
import com.hp.hpl.jena.vocabulary.DAML_OIL;

/**
```

```
 *  Class representing RDFS.Class -objects
 *  There are three distinct subclasses,
 *  (1) a KeltsiClass of Services
 *  (2) a KeltsiClass of Products
 *  (3) a KeltsiClass of an additional ontology.
 *  Since they behave identically,
 *  subclassing has been removed.
 *  Methods to make the distinction are provided.
 */
public class KeltsiClass extends KeltsiResource {
    private KeltsiClass top;
    private KeltsiDAML model;

    /**
     *  Constructor not to be called directly, use
     *  KeltsiDAML.getKeltsiResource instead.
     */
    KeltsiClass(DAMLClass cls) {
        super(cls);
        model = (KeltsiDAML) cls.getModel();
    }

    public String toString() {
        return getLabel();
    }

    /**
     *  Iterator over the subclasses of this class.
     *  The iterator holds either only the direct
     *  subclasses (close is false) or the transitive
     *  closure of the inverse of the property subClassOf.
     *  @param close Supply transitive closure if true.
     *  @return Iterator over subclasses of this class.
     */
    public Iterator getSubClasses(boolean close) {
        return classIterator(getSubClassNodes(close));
    }

    public Iterator getSubClassesSorted() {
        LinkedList subClassList = classList(getSubClassNodes(false));
        Collections.sort(subClassList); //, new ResourceComparator());
        return subClassList.iterator();
    }

    /**
     *  Iterator over the superclasses of this class.
     *  The iterator holds either only the direct
     *  superclasses (close is false) or the transitive
     *  closure of the property subClassOf.
     *  @param close Supply transitive closure if true.
     *  @return Iterator over superclasses of this class.
     */
    public Iterator getSuperClasses(boolean close) {
        /* could do
```

```
      return classIterator(getSuperClassNodes(close));
      but order is random
    */
    LinkedList superClasses = new LinkedList();
    KeltsiClass superC = getSuper();
    while (superC != null) {
        superClasses.addFirst(superC);
        superC = superC.getSuper();
    }
    return superClasses.iterator();
}

/**
 *  Is the supplied class a subclass of this class?
 *  @param cls The candidate to be a subclass.
 *  @return Boolean indicating wether the class is a
 *  subclass of this class.
 */
public boolean hasSubClass(KeltsiClass cls) {
    return cls.hasSuperClass(this);
}

/**
 *  Is the supplied class a superclass of this class?
 *  @param cls The candidate to be a superclass.
 *  @return Boolean indicating wether the class is a
 *  superclass of this class.
 */
public boolean hasSuperClass(KeltsiClass cls) {
    if (cls == null) {
        return false;
    }
    try {
        return model.contains(
                this.resource, Keltsi.subClassOf, cls.getResource());
    } catch(Exception ex) {
        return false;
    }
}

/**
 *  Get the direct superclass of this class.
 *  Assuming there is only one, as is the case in the
 *  current data.
 *  @return The direct superclass of this class.
 */
public KeltsiClass getSuper() {
    try {
        Resource superClass =
            this.resource.getProperty(RDFS.subClassOf).getResource();
        if (!model.isAClass(superClass)) {
            return null;
        }
        return (KeltsiClass) model.getKeltsiResource(superClass.getURI());
```

```
        } catch(Exception ex) {
            return null;
        }
    }

    /**
     *  Get the topmost superclass of this class
     *  before rdfs:Resource or daml:Thing or whatever.
     *  @return The topmost superclass of this class.
     */
    public KeltsiClass getTop() {
        if (top == null) {
            KeltsiClass cls = this;
            KeltsiClass superC = getSuper();
            while (superC != null) {
                cls = superC;
                superC = cls.getSuper();
            }
            top = cls;
        }
        return top;
    }

    /**
     *  Provide an iterator over all the possible properties
     *  that are immediate subproperties of property.
     *  @param property The superproperty whose subproperties
     *                  can only be returned
     *  @result An iterator.
     */
    public Iterator getDefinedSubPropertiesOf(DAMLProperty property) {
        LinkedList properties = new LinkedList();
        Iterator props = getDAMLClass().getDefinedProperties();
        while (props.hasNext()) {
            DAMLProperty subProperty = (DAMLProperty)props.next();
            PropertyAccessor pa = subProperty.prop_subPropertyOf();
            if (pa.hasValue(property)) {
                properties.add(subProperty);
            }
        }
        return properties.iterator();
    }

    /* *
     *  Get the count of instances of this class.
     *  If close is false, only the direct instances will be
     *  included. If close is true, instances will be gathered
     *  from the transitive closure of the inverse of
     *  rdfs:subPropertyOf and this class.
     *  @param close Supply transitive closure if true.
     *  @return The count of instances.
     */
    public int getInstanceCount(boolean close) {
        int count = 0;
```

```
        Iterator instanceNodes = getInstanceNodes(close);
        while(instanceNodes.hasNext()) {
            ++count;
            instanceNodes.next();
        }
        return count;
    }

    /**
     *  Get an iterator over instances of this class.
     *  If close is false, only the direct instances will be
     *  included. If close is true, instances will be gathered
     *  from the transitive closure of the inverse of
     *  rdfs:subPropertyOf and this class.
     *  @param close Supply transitive closure if true.
     *  @return An iterator over instances.
     */
    public Iterator getInstances(boolean close) {
        Iterator instanceNodes = getInstanceNodes(close);
        LinkedList instances = new LinkedList();
        while(instanceNodes.hasNext()) {
            instances.add(model.getKeltsiResource(
                    (DAMLCommon) instanceNodes.next()));
        }
        Collections.sort(instances, new ResourceComparator());
        return instances.iterator();
    }

    /**
     * Does this class have Services as instances?
     * In other words, is this class a subclass of ServiceRoot.
     * If a class is neither a ServiceClass nor a ProductClass
     * it is an auxiliary OntologyClass (Location etc).
     * @return True if this class has Services as instances.
     */
    public boolean isServiceClass() {
        return (getTop().equals(Keltsi.ServiceRoot));
    }

    /**
     * Does this class have Products as instances?
     * In other words, is this class a subclass of ProductRoot.
     * At the moment there are no Products as instances,
     * but that need not to be the case.
     * If a class is neither a ServiceClass nor a ProductClass
     * it is an auxiliary OntologyClass (Location etc).
     * @return True if this class has Products as instances.
     */
    public boolean isProductClass() {
        return (getTop().equals(Keltsi.ProductRoot));
    }

    /**
     * Is this class right below Thing or Resource or whatever?
```

```
     * @return True if this class is topmost in it's ontology.
     */
    public boolean isTop() {
        return (getTop().equals(this));
    }

    Iterator getInstanceNodes(boolean close) {

        Property type = close ? Keltsi.type : RDF.type;
        return model.select(null, type, this.resource);

/*
        if (close) {
            LinkedList instanceNodes = new LinkedList();
            Iterator subClasses = getSubClassNodes(true);
            while(subClasses.hasNext()) {
                Resource subClass = (Resource) subClasses.next();
                Iterator iterator = model.select(null, RDF.type, subClass);
                while (iterator.hasNext()) {
                    instanceNodes.add(iterator.next());
                }
            }
            return instanceNodes.iterator();
        } else {
            return model.select(null, RDF.type, this.resource);
        }
*/
/*
        String xQuery = "SELECT ?x WHERE ";
        Property subClassOf = model.subClassProperty(close);
        xQuery += model.triple("?x", RDF.type, "?class")
                + model.triple("?class", subClassOf, this.resource);
        return model.xQuery(xQuery);
*/
    }

    DAMLClass getDAMLClass() {
        return (DAMLClass)resource;
    }

    Iterator getSuperClassNodes(boolean close) {
        return model.select(
                this.resource, model.subClassProperty(close), null);
    }

    Iterator getSubClassNodes(boolean close) {
        return model.select(
                null, model.subClassProperty(close), this.resource);
    }

    LinkedList classList(Iterator classNodes) {
        LinkedList kClasses = new LinkedList();
        while (classNodes.hasNext()) {
            Resource classNode = null;
```

```
            classNode = (Resource) classNodes.next();
            if(model.isAClass(classNode)) {
                KeltsiClass kClass =(KeltsiClass)
                            getModel().getKeltsiResource((DAMLClass)classNode);
                kClasses.addFirst(kClass);
            }
        }
        return kClasses;
    }

    Iterator classIterator(Iterator classNodes) {
        return classList(classNodes).iterator();
    }

}

class ResourceComparator extends Object implements Comparator {
    public int compare(Object o1, Object o2) {
        return o1.toString().compareTo(o2.toString());
    }
}
```

1.6 KeltsiDAML

```
package fi.helsinki.cs.keltsi.jenardf;

import com.hp.hpl.jena.daml.common.DAMLModelImpl;
import com.hp.hpl.jena.daml.common.DAMLSelector;
import com.hp.hpl.jena.daml.*;
import com.hp.hpl.mesa.rdf.jena.model.*;
import com.hp.hpl.jena.vocabulary.DAML_OIL;
import com.hp.hpl.mesa.rdf.jena.vocabulary.RDF;
import com.hp.hpl.mesa.rdf.jena.vocabulary.RDFS;

import com.hp.hpl.mesa.rdf.jena.common.*;
import com.hp.hpl.jena.rdf.query.*;

import fi.helsinki.cs.keltsi.util.*;

import java.util.HashMap;
import java.util.HashSet;
import java.util.Collections;
import java.util.Iterator;
import java.util.LinkedList;
import java.io.*;

/**
 *  The model extends DAMLModelImpl for now. Better solution
 *  could be to throw all DAML-stuff away and use basic ModelMem
 *  or ModelRDB. DAMLModel seems to be very slow in some aspects.
 *
 */
public class KeltsiDAML extends DAMLModelImpl {
```

```
public static final String keltsiNS =
            "http://protege.stanford.edu/Keltsi#";

public static final String RDFSNS = DAML_OIL.label.getNameSpace();
public static final String RDFNS = DAML_OIL.type.getNameSpace();

public static final int SUBJECT = 1;
public static final int PREDICATE = 2;
public static final int OBJECT = 4;
public String error;
private HashMap inverse;
private HashMap ontology;

public String modelName = "KeltsiData";

/******* Methods to access the schema and instance models ********/

private String baseURL = "http://www.cs.helsinki.fi/group/keltsi/";
public void setBaseURL(String baseURL) {
    this.baseURL = baseURL;
}
public String getBaseURL() {
    return baseURL;
}

public void addFile(String fileName) {
    try {
        read(baseURL + fileName);
    } catch (Exception ex) {
        try {
            // Use the class loader to find the input file.
            InputStream in = fi.helsinki.cs.keltsi.jenardf.KeltsiDAML.class
                            .getClassLoader()
                            .getResourceAsStream(fileName);
            if (in == null) {
                throw new IllegalArgumentException(
                        "File: " + fileName + " not found");
            }
            read(new InputStreamReader(in), "");
        } catch (Exception ex2) {
            System.err.println("Exception: "+ex2);
            ex2.printStackTrace(System.err);
            error = ex2.toString();
        }
    }
}

/**
 * load model from file
 */
public void loadModel(String modelName) {
    this.modelName = modelName;
    addFile("restaurant.rdfs");
    addFile("Location.rdfs");
```

```
        addFile(modelName + ".rdfs");
        addFile(modelName + ".rdf");
        Keltsi.initialize(this);
        initOntologies();
    }

    public void init() {
        Keltsi.initialize(this);
        setUseEquivalence(false);
        initOntologies();
    }

    private void initOntologies() {
        Keltsi.ServiceRoot = (KeltsiClass) getKeltsiResource(Keltsi.Service);
        Keltsi.ProductRoot = (KeltsiClass) getKeltsiResource(Keltsi.Product);

        inverse = new HashMap();
        inverse.put(keltsiNS + "franchisee",
                    getDAMLValue(keltsiNS + "franchiseeOf"));
        ontology = new HashMap();

        String[] pverbs = {"retails", "rents", "maintains", "wholesales",
                           "manufactures "};
        ontology.put("productclass",
                new Ontology(Keltsi.ProductRoot, pverbs, "productclass",
                             Keltsi.ServiceRoot));

        String[] sverbs = {"franchises"};
        ontology.put("serviceclass",
                new Ontology(Keltsi.ServiceRoot, sverbs, "serviceclass",
                             Keltsi.ServiceRoot));

        String[] rverbs = {"restaurantType"};
        KeltsiClass restaurantType = (KeltsiClass) getKeltsiResource(
                "http://protege.stanford.edu/restaurant#RestaurantType");
        KeltsiClass restaurants = (KeltsiClass) getKeltsiResource(
                Keltsi.getURI() + "RestaurantsAndCatering");
        ontology.put("restauranttype",
                new Ontology(restaurantType, rverbs, "restauranttype",
                             restaurants));

        String[] averbs = {"ambiance"};
        KeltsiClass ambiance = (KeltsiClass) getKeltsiResource(
                "http://protege.stanford.edu/restaurant#Ambiance");
        ontology.put("ambiance",
                new Ontology(ambiance, averbs, "ambiance",
                             restaurants));

        String[] rtverbs = {"rating"};
        KeltsiClass rating = (KeltsiClass) getKeltsiResource(
                "http://protege.stanford.edu/restaurant#Rating");
        ontology.put("rating",
                new Ontology(rating, rtverbs, "rating",
                             restaurants));
```

```
        String[] lverbs = {"location"};
        KeltsiClass location = (KeltsiClass) getKeltsiResource(
                "http://protege.stanford.edu/Location#Area");
        ontology.put("location",
                new Ontology(location, lverbs, "location",
                             Keltsi.ServiceRoot));
    }

    public Iterator getOntologies () {
        return ontology.values().iterator();
    }

    public String[] getVerbs(String ontologyURI) {
        return (String[]) ontology.get(ontologyURI);
    }

    public DAMLProperty getInverseProperty(String propertyURI) {
        return (DAMLProperty) inverse.get(propertyURI);
    }

    /* Methods to keep account of resources */


    public KeltsiResource getKeltsiResource(String uri) {
        if (uri == null) {
            return null;
        }
        DAMLCommon jenaResource = getDAMLValue(uri);
        KeltsiResource keltsiResource = null;
        if (jenaResource instanceof DAMLInstance) {
            keltsiResource = new Service((DAMLInstance) jenaResource);
        } else if (jenaResource instanceof DAMLClass) {
            keltsiResource = new KeltsiClass((DAMLClass) jenaResource);
        }
        return keltsiResource;
    }

    public KeltsiResource getKeltsiResource(DAMLCommon jenaResource) {
        return getKeltsiResource(jenaResource.getURI());
    }

    /******* Methods to query the models ********/

    /**
     * execute RDQL-query
     */
    public QueryResults executeQuery(String queryString) {
        Query query = new Query(queryString) ;
        query.setSource(this);
        QueryExecution qe = new QueryEngine(query) ;
        return qe.exec();
    }
```

```
/**
 * Execute RDQL-query beginning "SELECT ?x WHERE"
 */
LinkedList xQueryList(String queryString) {
    LinkedList list = new LinkedList();
    System.err.println(queryString);
    QueryResults results = executeQuery(queryString);
    while(results.hasNext()) {
        ResultBinding resultBinding = (ResultBinding)results.next();
        list.add(resultBinding.get("x"));
    }
    return list;
}

public Iterator xQuery(String queryString) {
    return xQueryList(queryString).iterator();
}

/**
 * Execute RDQL-query beginning "SELECT ?x WHERE"
 * and add results to set. Makes possible to
 * create union of several queryresults.
 */
public void xQuerySet(String queryString, HashSet set) {
    QueryResults results = executeQuery(queryString);
    while(results.hasNext()) {
        ResultBinding resultBinding = (ResultBinding)results.next();
        set.add(resultBinding.get("x"));
    }
}

/**
 * Query with a selector
 */
public ResultIterator select(Resource subject,
                             Property predicate,
                             RDFNode object) {
    int select = 0;
    if (subject == null) {
        select += SUBJECT;
    }
    if (predicate == null) {
        select += PREDICATE;
    }
    if (object == null) {
        select += OBJECT;
    }
    DAMLSelector damlSelector =
            new DAMLSelector(subject, predicate, object);
    try {
        StmtIterator si =
                listStatements(damlSelector);
        return new ResultIterator(si, select);
    } catch (Exception ex) {
```

```
            return null;
        }
    }

    /**
     * Contruct query-node
     */
    private String qnode(Object node) {
        String result = "";
        if (node == null) {
            result = "?x";
        }else if (node instanceof java.lang.String) {
            result = (String) node;
            if (result.startsWith("http://")) {
                result = "<" + result + ">";
            }
        } else if (node instanceof Resource) {
            result = "<" + ((Resource) node).getURI() + ">";
        } else if (node instanceof KeltsiResource) {
            result = "<" + ((KeltsiResource) node).getURI() + ">";
        }
        return result;
    }

    /**
     * Contruct query-triple
     */
    public String triple(Object subject, Object predicate, Object object) {
        return "\n\t(" + qnode(subject) + ", " + qnode(predicate) + ", " + qnode(object) + ") ";
    }

    private String getNextName(Iterator names) {
        while(names.hasNext()) {
            String name = (String) names.next();
            if (ontology.containsKey(name)){
                return name;
            }
        }
        return null;
    }

    public Iterator mapQuery(HashMap map, boolean close) {
        LinkedList nodes = mapQueryList(map, close);
        for (int i = 0; i < nodes.size(); ++i) {
            KeltsiResource service = getKeltsiResource((DAMLCommon)nodes.get(i));
            if (service != null) {
                nodes.set(i, service);
            }
        }
        Collections.sort(nodes);
        return nodes.iterator();
    }

    public int mapQuerySize(HashMap map, boolean close) {
```

```
        return mapQueryList(map, close).size();
    }

    public LinkedList mapQueryList(HashMap map, boolean close) {
        Iterator names = map.keySet().iterator();
        if (!map.containsKey("serviceclassuri")) {
            return null;
        }
        Property subClassOf = subClassProperty(close);
        String sclassURI = (String) map.get("serviceclassuri");
        Property type = close ? Keltsi.type : RDF.type;
        String query = "SELECT ?x WHERE "
                + triple(null, type, sclassURI);
        int count = 1;
        while(names.hasNext()) {
            String name = (String) names.next();
            if (ontology.containsKey(name)){
                Ontology ont = (Ontology) ontology.get(name);
                String value = (String) map.get(name);
                String predicate = (String) map.get(name + "verb");
                if (predicate == null) {
                    predicate = ont.defaultVerb();
                }
                if (close) {
                    String link = "?link" + count;
                    query += triple(null, keltsiNS + predicate, link)
                           + triple(link, subClassOf, value);
                    ++count;
                } else {
                    query += triple(null, keltsiNS + predicate, value);
                }
            }
        }
        return xQueryList(query);
    }

    public String getSynonymClassURI(String synonym) {
        return getSynonymClassURI(synonym, Keltsi.ServiceRoot);
    }

    public String getSynonymClassURI(String synonym, KeltsiClass root) {
        Iterator results = select(null,
                                  Keltsi.synonym,
                                  new LiteralImpl(synonym));
        if (results.hasNext()) {
            return ((DAMLCommon) results.next()).getURI();
        } else {
            return searchClassLabels(synonym, root);
        }
    }

    public String searchClassLabels(String synonym, KeltsiClass root) {
        String query = "SELECT ?x WHERE "
                + triple("?x", Keltsi.subClassOf, root)
```

```
                + triple("?x", RDFS.label, "\"" + synonym + "\"");
                //+ "AND ?label LIKE '/^" + synonym + "/i'";
        Iterator results = xQuery(query);
        if (results.hasNext()) {
            return ((Resource) results.next()).getURI();
        } else {
            return null;
        }
    }

    Property subClassProperty(boolean close) {
        return close ? Keltsi.subClassOf : RDFS.subClassOf;
    }

    boolean isAClass(Resource resource) {
        return resource != null && resource instanceof DAMLClass
                && !resource.equals(DAML_OIL.Thing);
    }

    public String inverseName(String localName) {
        if (localName == null) {
            return null;
        }
        if (localName.endsWith("Of")){
            return localName.substring(0, localName.length() - 2);
        } else {
            return localName + "Of";
        }
    }

    public static void main(String args[])
    throws RDFException {
        KeltsiDAML model = new KeltsiDAML();
        model.setBaseURL("http://192.168.1.1/Keltsi/");
        model.loadModel(model.modelName);
    }
}
```

1.7 KeltsiResource

```
package fi.helsinki.cs.keltsi.jenardf;

import com.hp.hpl.jena.daml.*;
import java.util.Hashtable;
import java.util.LinkedList;
import com.hp.hpl.mesa.rdf.jena.model.*;
import com.hp.hpl.mesa.rdf.jena.vocabulary.RDFS;
import com.hp.hpl.mesa.rdf.jena.vocabulary.RDF;

public abstract class KeltsiResource extends Object implements Comparable {
    protected DAMLCommon resource;
    private String label;
    private KeltsiClass type;
```

```
KeltsiResource(DAMLCommon resource) {
    this.resource = resource;
}

protected KeltsiDAML getModel() {
    return (KeltsiDAML) resource.getModel();
}

public KeltsiClass getType() {
    if (type == null) {
        try {
            DAMLClass dClass = (DAMLClass) resource.
                    getPropertyValue(RDF.type);
            type = (KeltsiClass) getModel().getKeltsiResource(dClass);
        } catch(Exception ex) {
            return null;
        }
    }
    return type;
}

public DAMLCommon getResource() {
    return resource;
}

public String getURI() {
    return resource.getURI();
}

public String getNameSpace() {
    return resource.getNameSpace();
}

public String getLabel() {
    if (label == null) {
        Statement s = null;
        /* Keltsi.label, workaround
            Protege doesn't allow to write meaningful labels for instances
        */
        try {
            s = resource.getProperty(Keltsi.label);
            if(s != null) {
                label = s.getString();
                return label;
            }
        } catch(Exception ex) {}
        try {
            s = resource.getProperty(RDFS.label);
            if(s != null) {
                label = s.getString();
                return label;
            }
        } catch(Exception ex) {}
```

```
            try {
                label = resource.getLocalName();
                return label;
            } catch(Exception ex) {}
            label = resource.toString();
        }
        return label;
    }

    /**
     *  Is the object equal to this one.
     *  The decision is droped to the jena-layer,
     *  which makes it by comparing the URIs of
     *  the Resources.
     *  @param object The object to compare this one to.
     *  @return True if the object equals this one.
     */
    public boolean equals(Object object) {
        if (object instanceof KeltsiResource) {
            return getResource()
                    .equals(((KeltsiResource) object).getResource());
        } else {
            return false;
        }
    }

    /**
     * Override Object.hashCode() and use Resource.hashCode()
     */
    public int hashCode() {
        return this.resource.hashCode();
    }

    /**
     *  Override Object.toString()
     */
    public String toString() {
        return this.resource.getURI();
    }

    /**
     * Method to implement Comparable.
     */
    public int compareTo(Object other) {
        return this.toString().compareTo(other.toString());
    }

    /**
     * Return stringvalue of property. The range of the property is
     * expected to be rdf:Literal.
     */
    public String getSingleLiteralValue(Property property) {
        RDFNode node = null;
        try {
```

```
            node = resource.getProperty(property);
        }
        catch (Exception ex) {
            return null;
        }
        if (node == null) return
                "<null value (shouldn't happen according to Jena docs)>";
        if (!(node instanceof Statement)) return
                "<not a statement (shouldn't happen according to Jena docs)>";
        try {
            return ((Statement) node).getString();
        }
        catch (Exception ex) {
            return "<not a literal>";
        }
    }

}
```

1.8 Ontology

```
package fi.helsinki.cs.keltsi.jenardf;

import com.hp.hpl.jena.daml.*;
import java.util.Iterator;
import com.hp.hpl.mesa.rdf.jena.model.*;
import com.hp.hpl.mesa.rdf.jena.vocabulary.RDF;
import com.hp.hpl.mesa.rdf.jena.vocabulary.RDFS;
import java.util.LinkedList;

public class Ontology extends Object {
    KeltsiClass rangetop; //root;
    KeltsiClass domaintop;
    String verbs[];
    String label;

    public Ontology(KeltsiClass rangetop, String[] verbs, String label, KeltsiClass domaintop) {
        this.rangetop = rangetop;
        this.verbs = verbs;
        this.label = label;
        this.domaintop = domaintop;
    }

    public String defaultVerb () {
        if (verbs != null && verbs.length > 0) {
            return verbs[0];
        } else {
            return null;
        }
    }

    public KeltsiClass getRangeTop() {
        return rangetop;
    }
```

```
    public String[] getVerbs() {
        return verbs;
    }

    public String getLabel() {
        return label;
    }

    public KeltsiClass getDomainTop() {
        return domaintop;
    }

    public String getNameSpace() {
        return rangetop.getNameSpace();
    }

}
```

1.9 Service

```
package fi.helsinki.cs.keltsi.jenardf;

import fi.helsinki.cs.keltsi.util.*;
import com.hp.hpl.mesa.rdf.jena.vocabulary.RDF;
import com.hp.hpl.mesa.rdf.jena.vocabulary.RDFS;

import com.hp.hpl.mesa.rdf.jena.model.*;
import com.hp.hpl.jena.daml.*;
import java.util.LinkedList;
import java.util.Iterator;

/**
 *  Class to encapsulate a Service.
 *  Provides access to it's properties including
 *  advertisement and contacts.
 */
public class Service extends KeltsiResource {
    private KeltsiClass type;
    private String label;
    private Advertisement advertisement;
    /*
     * Constructor called only from KeltsiDAML.
     * Use KelstiDAML.getKeltsiResource to access Services.
     */
    Service(DAMLInstance service) {
        super(service);
    }

    /**
     *  Get the class (rdf:type) of the Service.
     *  @return The KeltsiClass object that corresponds to the Service's type.
     */
    public KeltsiClass getServiceClass() {
```

```
        if (type == null) {
            try {
                type = (KeltsiClass) getModel().getKeltsiResource((DAMLClass) getType());
            } catch(Exception ex) {}
        }
        return type;
    }

    /**
     * The label of the Service. First tries to retrieve Keltsi.label,
     * then the advertisements label. Best practice would be rdfs:label
     * but due to Protege this was not possible.
     * TODO: Language versions.
     * @return The label of the service as a String.
     */
    public String getLabel() {
        if (label == null) {
            label = getSingleLiteralValue(Keltsi.label);
            if (label == null) {
                Advertisement adv = getAdvertisement();
                if (adv != null) {
                    label = adv.getLabel();
                }
            }
        }
        if (label == null) {
            label = "no name";
        }
        return label;
    }

    /**
     * A short description of the Service. Perhaps better would be to
     * to use rdfs:comment, but that is not available for instances
     * in Protege.
     * TODO: Language versions.
     * @return The description of the Service.
     */
    public String getDescription() {
        return getSingleLiteralValue(Keltsi.description);
    }

    /**
     * Creates an iterator holding values of the supplied property,
     * (<this>, <property>, ?service).
     * Fails (and returns an empty or partial list) if a
     * non-Service value is encountered.
     * @return Iterator of property values of type Service.
     * @param property The property whose values are to be iterated.
     */
    public Iterator getServiceValues(DAMLProperty property) {
        NodeIterator ni = resource.getPropertyValues(property);
        LinkedList list = new LinkedList();
        try {
```

```
            while (ni.hasNext()) {
                Service service = (Service) getModel()
                        .getKeltsiResource((DAMLInstance)ni.next());
                if (service != null) {
                    list.add(service);
                }
            }
        } catch(Exception ex) {}
        return list.iterator();
    }

    /**
     * Creates an iterator holding the Service-values of the supplied
     * properties inverse, (?service, <property>, <this>).
     * Fails (and returns an empty or partial list) if a
     * non-Service value is encountered.
     * @return Iterator over inverse properties values of type Service.
     * @param property The property whose inverses values are to be iterated.
     */
    public Iterator getServiceInverseValues(DAMLProperty property) {
        ResultIterator ri = getModel().select(null, property, getResource());
        LinkedList list = new LinkedList();
        try {
            while (ri.hasNext()) {
                Service service = (Service) getModel()
                        .getKeltsiResource((DAMLInstance)ri.next());
                if (service != null) {
                    list.add(service);
                }
            }
        } catch(Exception ex) {ex.printStackTrace();}
        return list.iterator();
    }

    /**
     * Creates an iterator holding KeltsiClass-values of the supplied
     * property, (<this>, <property>, ?class).
     * Fails (and returns an empty or partial list) if a
     * non-KeltsiClass value is encountered.
     * @return Iterator of property values of type KeltsiClass.
     * @param property The property whose values are to be iterated.
     */
    public Iterator getClassValues(DAMLProperty property) {
        NodeIterator ni = resource.getPropertyValues(property);
        LinkedList list = new LinkedList();
        try {
            while (ni.hasNext()) {
                KeltsiClass cls = (KeltsiClass) getModel()
                        .getKeltsiResource((DAMLClass) ni.next());
                if (cls != null) {
                    list.add(cls);
                }
            }
        } catch(Exception ex) {ex.printStackTrace();}
```

```
        return list.iterator();
    }

    /**
     * Creates an iterator holding Literal values of the supplied
     * property, (<this>, <property>, ?literal).
     * Fails (and returns an empty or partial list) if a
     * non-Literal value is encountered.
     * Literals are transformed to Strings.
     * This is not good practice, if the xml:lang attribute
     * is of use.
     * @return Iterator of property values of type Literal,
     * represented as Strings in the list.
     * @param property The property whose values are to be iterated.
     */
    public Iterator getLiteralValues(DAMLProperty property) {
        NodeIterator ni = resource.getPropertyValues(property);
        LinkedList list = new LinkedList();
        try {
            while (ni.hasNext()) {
                try {
                    Literal literal = (Literal)ni.next();
                    list.add(literal.getString());
                } catch(Exception ex) {ex.printStackTrace();}
            }
        } catch(Exception ex2) {ex2.printStackTrace();}
        return list.iterator();
    }

    /**
     * Get the advertisement. There should be exactly one.
     * In case none is found, the missingAd is used.
     * @return The advertisement of this Service.
     */
    public Advertisement getAdvertisement() {
        if (advertisement == null) {
            DAMLInstance adNode = null;
            ResultIterator resIter = getModel().select(
                    null, Keltsi.services, (DAMLInstance) resource);
            if (resIter.hasNext()) {
                try {
                    adNode = (DAMLInstance)resIter.next();
                } catch (Exception ex) {}
                resIter.close();
            }
            if (adNode == null) {
                adNode = Keltsi.missingAd;
                try {
                    adNode.addProperty(Keltsi.services, this.resource);
                } catch(Exception ex) {}
            }
            advertisement = new Advertisement(adNode);
        }
        return advertisement;
```

```
    }

    /**
     * Get an iterator for contacts of this Service.
     * There should be one or more.
     * @return An iterator over the contacts of this service.
     */
    public Iterator getContacts() {
        LinkedList contacts = new LinkedList();
        NodeIterator iter = ((DAMLInstance) resource)
                .getPropertyValues(Keltsi.contact);
        try {
            while (iter.hasNext()) {
                contacts.add(new Contact((DAMLInstance)iter.next()));
            }
        } catch (Exception ex) {
            try {iter.close();} catch(Exception exx) {}
        }
        return contacts.iterator();
    }

    public String toString() {
        if (getServiceClass() != null) {
            return getServiceClass().getLabel() + " / " + getLabel();
        } else {
            return getLabel();
        }
    }

}
```

2 fi.helsinki.cs.keltsi.jenardf.test

2.1 TestAdvertisement

```
package fi.helsinki.cs.keltsi.jenardf.test;

import junit.framework.*;
import java.lang.*;
import java.io.*;
import java.util.*;
import fi.helsinki.cs.keltsi.jenardf.*;
import fi.helsinki.cs.keltsi.utils.*;

import com.hp.hpl.mesa.rdf.jena.mem.ModelMem;
import com.hp.hpl.mesa.rdf.jena.model.*;
import com.hp.hpl.mesa.rdf.jena.vocabulary.RDF;
import com.hp.hpl.mesa.rdf.jena.vocabulary.RDFS;
import com.hp.hpl.mesa.rdf.jena.common.*;
import com.hp.hpl.jena.rdf.query.*;

/**
 * Class to test Advertisement.
```

```
 */
public class TestAdvertisement extends TestCase {

public TestAdvertisement(String theName) {
super(theName);
}

/**
 * main
 * @param String[] arguments (not used)
 */
public static void main(String[] theArgs) {
junit.textui.TestRunner.main
(new String[] {TestAdvertisement.class.getName()});
}

/**
 * Build a test <code>suite</code>.
 */
public static Test suite() {
return new TestSuite(TestAdvertisement.class);
}

private static String kbNS = "http://protege.stanford.edu/kb#";
private KeltsiJenaQuery kjq;
public void setUp()
{
kjq = new KeltsiJenaQuery();
}

/**
 * Methods for convenience to create two instances
 */
private Advertisement adv1 = null;
private Advertisement adv1()
{
if (adv1 == null)
{
Resource res = (Resource)kjq.getResourceByURI
(kbNS + "UNSPSC_00101");
adv1 = new Advertisement(res, kjq);
}
return adv1;
}

private Advertisement adv2 = null;
private Advertisement adv2()
{
if (adv2 == null)
{
Resource res = (Resource)kjq.getResourceByURI
(kbNS + "UNSPSC_00418");
adv2 = new Advertisement(res, kjq);
}
```

```
return adv2;
}


/**
 * Test constructor
 */
public void testConstructor()
throws Exception
{
assertTrue(adv1() != null);
assertTrue(adv2() != null);
}

/**
 * Test constructor
 */
public void testProviderOfService()
throws Exception
{
assertTrue(adv1().providerOfService() != null);
assertTrue(!adv1().providerOfService().equals(""));
assertTrue(adv2().providerOfService() != null);
assertTrue(!adv2().providerOfService().equals(""));
}

/**
 * Test constructor
 */
public void testImageFileURL()
throws Exception
{
assertTrue(adv1().imageFileURL() != null);
assertTrue( adv1().imageFileURL().equals(""));
assertTrue(adv2().imageFileURL() != null);
assertTrue(!adv2().imageFileURL().equals(""));
}

public void testContact()
throws Exception
{
assertTrue(adv1().contact() != null);
assertTrue(adv2().contact() != null);
}

}
```

3 fi.helsinki.cs.keltsi.tags

3.1 DisplayAdvertisement

```
package fi.helsinki.cs.keltsi.tags;

import fi.helsinki.cs.keltsi.util.*;
```

```
import fi.helsinki.cs.keltsi.jenardf.*;

import com.hp.hpl.jena.daml.*;
import com.hp.hpl.mesa.rdf.jena.model.*;
import com.hp.hpl.mesa.rdf.jena.common.*;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.jsp.tagext.TagSupport;
import javax.servlet.jsp.JspException;
import java.util.*;
import javax.servlet.jsp.PageContext;

/**
 *  Display details of a service
 *  Author Tuomo Kajava
 */
public class DisplayAdvertisement extends TagSupport {
    private final static String imagePath="img/";
    private String serviceUri;
    private KeltsiDAML model;
    private Html html;
    private Link link;

    public int doStartTag() throws JspException {
        String advString = "";
        html = new Html(pageContext.getOut());
        html.write("<!-- DisplayAdvertisement starting -->\n");

        /* Get the model from scope. */
        try {
            model = (KeltsiDAML) pageContext.findAttribute(
                    "fi.helsinki.cs.keltsi.KeltsiDAML");
        } catch (Exception ex) {
            html.errorMsg(ex, "while getting the model from scope");
        }
        Service service=null;
        try {
            HttpServletRequest req =
                    (HttpServletRequest)pageContext.getRequest();
            // Get serviceuri.
            serviceUri = req.getParameter("serviceuri");
            // Get service object.
            service=(Service) model.getKeltsiResource(serviceUri);
            link = new Link(req);
        } catch(Exception ex) {
            html.errorMsg(ex, " DA 1");
        }

        Advertisement adv = null;
        try {
            adv = service.getAdvertisement();
```

```
    advString = adv.getLabel();
} catch(Exception ex) {
    html.errorMsg(ex, " DA 2");
}

try {
    ImageFile img = adv.getImageFile();
    if (img != null) {
        String imgURL = imagePath + img.path();
        html.write(
                "<a href=\"" + imgURL + "\">"
                + html.img(imgURL, 150, 0, "right")
                + "</a>\n");
    }
} catch(Exception ex) {
    html.errorMsg(ex,  " DA 4");
}

html.write("<table>\n");
html.write(html.bar(html.element("h2",
        html.superClassLink(service.getType(), link, "serviceclassuri", false),
        "servicetype", null)));

try {
    advString = service.getLabel();
} catch(Exception ex) {
    html.errorMsg(ex, " DA 3");
}

html.write(html.tr(html.td(html.element(
        "h2", advString, "advlabel", null), null, "colspan=\"2\"")));
Iterator contacts = null;
try {
    contacts = service.getContacts();
} catch(Exception ex) {
    html.errorMsg(ex, " DA 5");
}

String description = service.getDescription();
html.write(html.pair("", description, "description"));

while(contacts.hasNext()) {
    Contact con=(Contact)contacts.next();
    String http = con.getUrl();
    /* Both "www.x.dom" and "http://www.x.dom" occur.
     * Fix it here, use "www.x.dom".
     */
    if (http != null && http.startsWith("http://")) {
        http = http.substring(7, http.length());
    }
    html.write(html.pair("", con.getLabel(), "contactlabel"));
    html.write(html.pair("Opening hours", con.getOpeningHours()));
    html.write(html.pair("Address", con.getAddress()));
    html.write(html.pair("Telephone", con.getTelephone()));
```

```
            html.write(html.pair("Fax", con.getFax()));
            html.write(html.pairlink("Email", con.getEmail(), null, "mailto"));
            html.write(html.pairlink("", http, null, "http://"));
            if (contacts.hasNext()) {
                html.write(html.tr(html.td(html.hr(), null, "colspan=\"2\"")));
            }
        }
        html.write("</table>\n");
        html.write("\n<!-- DisplayAdvertisement done -->\n");
        return SKIP_BODY;
    }
}
```

3.2 DisplayServiceClass

```
package fi.helsinki.cs.keltsi.tags;

import fi.helsinki.cs.keltsi.jenardf.Keltsi;
import fi.helsinki.cs.keltsi.jenardf.KeltsiClass;
import fi.helsinki.cs.keltsi.jenardf.Service;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.jsp.tagext.TagSupport;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.JspWriter;
import java.util.Iterator;

/**
 *  Display  a serviceclass
 *  @author Arno Aalto
 */
public class DisplayServiceClass extends TagSupport {

    private static boolean displayInstances = false;
    private Html html;
    private KeltsiClass serviceClass;
    private Link link;
    private String synonym;

    public int doStartTag() throws JspException {
        init();
        html.write("<form action=\"" + link.url(null, null, null) + "\""
                       + "method=\"post\">"
                  + "<table class=\"servicetable\" "
                          + "cellspacing=\"0\" "
                          + "cellpadding=\"2\" "
                          + "width=\"100%\">\n");

        displaySearchBar();
        displaySuperClasses();
```

```
        displayThisClass();
        displaySubClasses();
        if (displayInstances) {
            displayInstanceList();
        }
        html.write("</table>\n</form>\n");
        return SKIP_BODY;
    }

    private void init() {
        try {
            HttpServletRequest req =
                    (HttpServletRequest)pageContext.getRequest();
            html = new Html(pageContext.getOut());
            serviceClass = (KeltsiClass) pageContext
                    .getAttribute("fi.helsinki.cs.keltsi.ServiceClass");
            if (serviceClass == null) {
                throw new JspException("No serviceclass found!");
            }
            link  = (Link) pageContext
                    .getAttribute("fi.helsinki.cs.keltsi.ServiceClassLink");
            synonym = req.getParameter("serviceclasssynonym");
            if (synonym != null) {
                link.removeParam("serviceclasssynonym");
            } else {
                synonym = "";
            }
        } catch(Exception ex) {
            html.errorMsg(ex);
        }
    }

    private void displaySearchBar() {
        html.write(
            html.bar("Service shortcut:"
                     + "<input type=\"text\"/ "
                     +   "size=\"12\""
                     +   "value=\"" + synonym + "\" "
                     +   "name=\"serviceclasssynonym\" />"
                     + "<input type=\"submit\" "
                     +   "name=\"synonym\" value=\"go!\" />"
                     ));
    }

    private void displaySuperClasses() {
        Iterator sClasses = serviceClass.getSuperClasses(true);
        while (sClasses.hasNext()) {
            KeltsiClass sClass = (KeltsiClass)sClasses.next();
            html.write(
                html.tr(
                    html.td(
                        html.superClassLink(sClass, link, "serviceclassuri",
                                            false))
                  + html.td("" + sClass.getInstanceCount(true),
```

```
                            null, "align=\"right\""));
        }
    }

    private void displayThisClass() {
        html.write(
            html.tr(
                html.td(html.thisClass(serviceClass))
              + html.td("" + serviceClass.getInstanceCount(true),
                        null, "align=\"right\"")));
    }

    private void displaySubClasses() {
        Iterator sClasses = serviceClass.getSubClassesSorted();
        while (sClasses.hasNext()) {
            KeltsiClass sClass = (KeltsiClass)sClasses.next();
            html.write(
                html.tr(
                    html.td(
                        html.subClassLink(sClass, link, "serviceclassuri"))
                  + html.td("" + sClass.getInstanceCount(true),
                            null, "align=\"right\"")));
        }
    }

    private void displayInstanceList() {
        Iterator services = serviceClass.getInstances(false);
        while (services.hasNext()) {
            Service service = (Service)services.next();
            html.write(
                html.tr(
                    html.td(
                        html.serviceLink(service, link),
                                    null, "colspan=\"2\"")));
        }
    }
}
```

3.3 Html

```
package fi.helsinki.cs.keltsi.tags;

import fi.helsinki.cs.keltsi.jenardf.Keltsi;
import fi.helsinki.cs.keltsi.jenardf.KeltsiClass;
import fi.helsinki.cs.keltsi.jenardf.Service;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.jsp.tagext.TagSupport;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.JspWriter;
```

```
import java.util.Iterator;
import java.net.URLEncoder;

/**
 *  Class to write html to JspWriter
 */
public class Html {
    // TODO: Change to relative links and include icons in distribution package
    public static final String serviceIcon =
            "icon/service.gif";
    public static final String thisClassIcon =
            "icon/thisclass.gif";
    public static final String subClassIcon =
            "icon/subclass.gif";
    public static final String superClassIcon =
            "icon/superclass.gif";

    protected JspWriter out;

    public Html(JspWriter out) {
        this.out = out;
    }

    public void errorMsg(String msg) {
        try {
            out.write("<strong>" + msg + "</strong>");
        } catch(Exception ex) {
        }
    }

    public void errorMsg(Exception ex) {
        try {
            PrintWriter pw = new PrintWriter(out, true);
            out.write("<xmp>");
            ex.printStackTrace(pw);
            out.write("</xmp>");
        } catch(Exception e) {
        }
    }

    public void errorMsg(Exception ex, String msg) {
        errorMsg(ex);
        errorMsg(msg);
        hr();
    }
    public void write(String string) {
        if (string != null) {
            try {
                out.write(string);
            } catch(Exception ex) {}
        }
    }

    public static String img(String url, int width, int height, String align) {
```

```
        String img =  "<img src=\"" + url + "\" ";
        if (height > 0) {
            img += "height=\"" + height + "\" ";
        }
        if (width > 0) {
            img += "width=\"" + width + "\" ";
        }
        if (align != null) {
            img += "align=\"" + align + "\" ";
        }
        img += "/>";
        return img;
    }

    public static String img(String url) {
        return img(url, 0, 0, null);
    }

    public static String hr() {
        return "<hr />\n";
    }

    public static String br() {
        return "<br />\n";
    }

    public static String tr(String text, String cssClass, String args) {
        return "\t" + element("tr", "\n\t\t" + text + "\t", cssClass, args) + "\n";
    }

    public static String tr(String text, String cssClass) {
        return tr(text, cssClass, null);
    }

    public static String tr(String text) {
        return tr(text, null, null);
    }

    public static String td(String text, String cssClass, String args) {
        return "\t\t" + element("td", text, cssClass, args) + "\n";
    }

    public static String td(String text, String cssClass) {
        return td(text, cssClass, null);
    }

    public static String td(String text) {
        return td(text, null, null);
    }

    public static String span(String text, String cssClass) {
        return element("span", text, cssClass, null);
    }
```

```
    public static String div(String text, String cssClass) {
        return element("div", text, cssClass, null);
    }

    public static String trimLabel(String labelString) {
        return labelString; // don't do anything!
/*        if (labelString == null) {
            return null;
        } else if (labelString.length() > 28) {
            return labelString.substring(0,27) + ".." ;
        } else {
            return labelString;
        }
*/
    }

    public static String element(String tag, String value, String cssClass, String args) {
        if (value == null || tag == null) {
            return null;
        }
        if (args == null || args.length() == 0) {
            args = "";
        } else {
            args = " " + args;
        }
        if (cssClass != null) {
            args  += " class=\"" + cssClass + "\"";
        }
        return "<" + tag + args + ">" + value +"</" + tag + ">";
    }

    public static String pair(String name, String value, String cssClass) {
        if (value == null) {
            return null;
        }
        String nameClass = null;
        String valueClass = null;
        if (cssClass != null) {
            nameClass  = cssClass + "Name";
            valueClass = cssClass + "Value";
        }
        return tr(  td(name, nameClass)
                  + td(value, valueClass));
    }

    public static String pair(String name, String value) {
        return pair(name, value, "");
    }

    public static String pairlink(String name, String value,
                        String cssClass, String protocol) {
        if (value == null) {
            return null;
        }
```

```
        value = "<a href=\"" + protocol + value + "\">" + value + "</a>";
        return pair(name, value, cssClass);
    }

    private static String anyLink(String icon, String link, String cssClass) {
        return span(img(icon) + " " + link, cssClass);
    }

    public static String superClassLink(KeltsiClass kClass, Link link,
                                        String name, boolean trim) {
        String label = kClass.getLabel();
        String label2 = trim ? trimLabel(label) : label;
        return anyLink(superClassIcon,
                       link.aHref("ontology", "search.jsp", name, kClass.getURI(),
                                  label2, "Move to upper class: " + label),
                        "superclasslink");
    }
    public static String superClassLink(KeltsiClass kClass, Link link,
                                        String name) {
        return superClassLink(kClass, link, name, true);
    }

    public static String subClassLink(KeltsiClass kClass, Link link,
                                      String name) {
        String label = kClass.getLabel();
        return anyLink(subClassIcon,
                       link.aHref("ontology", "search.jsp", name, kClass.getURI(),
                                  trimLabel(label), "Move to subclass: " + label),
                       "subclasslink");
    }

    public static String thisClass(KeltsiClass kClass) {
        return span(img(thisClassIcon) + " " + kClass.getLabel(), "thisclass");
    }

    public static String serviceLink(Service service, Link link) {
        String label = service.getLabel();
        String parent = "";
        KeltsiClass sClass = service.getType();
        if(sClass != null) {
            String parentLabel = sClass.getLabel();
            parent = link.aHref("ontology", "search.jsp", "serviceclassuri",
                                sClass.getURI(), trimLabel(parentLabel),
                                "Open class: " + parentLabel) + " / ";
        }
        return img(serviceIcon) + " "
               + parent
               + link.aHref("detail", "detail.jsp", "serviceuri",
                            service.getURI(), label,
                            "View service: " + label);
    }

    public static String optionList(Iterator classes, Link link, String name) {
        StringBuffer result = new StringBuffer();
```

```
        boolean empty = true;
        result.append("<select name=\"" + name + "\" >\n");
                        //+ "onChange=\"this.form.submit()\">\n");
        while (classes.hasNext()) {
            empty = false;
            KeltsiClass sClass = (KeltsiClass) classes.next();
            result.append("\t\t<option "
                            + "value=\"" + sClass.getURI()
                            + "\">"
                        + sClass.getLabel()
                        + "</option>\n");
        }
        result.append("\t</select><input type=\"submit\" value=\"go\"/>\n");
        if (empty) {
            return "";
        } else {
            return result.toString();
        }
    }

    public static String bar(String text) {
        return tr(td(text, null, "bgcolor=\"yellow\" colspan=\"2\""));
    }

    public static String bar(String text, String control) {
        return tr(td(text) + td(control, null, "align=\"right\""),
                  null, "bgcolor=\"yellow\"");
    }
}
```

3.4 Init

```
package fi.helsinki.cs.keltsi.tags;

import fi.helsinki.cs.keltsi.util.*;
import fi.helsinki.cs.keltsi.jenardf.*;

import com.hp.hpl.jena.daml.*;
import com.hp.hpl.mesa.rdf.jena.model.*;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.tagext.TagSupport;
import javax.servlet.jsp.*;
import java.util.*;
import javax.servlet.jsp.PageContext;

/**
 *  Initialize model (and perhaps some other things in application scope).
 */
public class Init extends TagSupport {
    private boolean noModel = true;
    private boolean reload = false;
```

```
    private String baseURL;

    /**
     *  To be able to reload model from a jsp.
     */
    public void setReload(String reload) {
        this.reload = (reload != null && reload.equals("on"));
    }

    /**
     *  TO be able to set the baseURL from a jsp.
     *  Some default value maybe set elsewhere (now in KeltsiDAML).
     */
    public void setBaseURL(String baseURL) {
        this.baseURL = baseURL;
    }

    public int doStartTag() throws JspException {
        KeltsiDAML kjq;
    JspWriter out = pageContext.getOut();
    Html html = new Html(out);
        try {
            /* Get session*/

            HttpServletRequest req =
                (HttpServletRequest)pageContext.getRequest();
            HttpSession session = req.getSession(true);

            /* Get keltsi-query-knowledge-base-object. */
            ServletContext sc = pageContext.getServletContext();
            kjq = (KeltsiDAML)sc.getAttribute(
                    "fi.helsinki.cs.keltsi.KeltsiDAML");
            if ((kjq == null || reload) ){
                kjq = new KeltsiDAML();
                if (baseURL != null) {
                    kjq.setBaseURL(baseURL);
                }
                kjq.loadModel("KeltsiData");
                sc.setAttribute("fi.helsinki.cs.keltsi.KeltsiDAML", kjq);
                out.write("<p>Model loaded succesfully!</p>");
            }
            pageContext.setAttribute(
                    "fi.helsinki.cs.keltsi.KeltsiDAML", kjq);
            noModel = false;
            try {
                out.write("<!-- Using model " + kjq + " from " + kjq.getBaseURL() + ". -->\n" );
            } catch (Exception ex2) {}
        } catch (Exception e) {
            noModel = true;
            try {
                out.write(
                      "<html>\n"
                    + "\t<head>\n"
                    + "\t\t<title>Error</title>\n"
```

```
                    + "\t</head>\n"
                    + "\t<body>\n"
                    + "\t\t<h1>Error loading model</h1>\n"
                    + "\t</body>\n"
                    + "</html>\n");
            } catch (Exception ex2) {}
        }
        return SKIP_BODY;
     }

    public int doEndTag() throws JspException {
        if (noModel) {
            return SKIP_PAGE;
        } else {
            return EVAL_PAGE;
        }
    }
}
```

3.5 Link

```
package fi.helsinki.cs.keltsi.tags;

import java.net.URLEncoder;
import java.util.HashMap;
import java.util.Enumeration;
import java.util.Iterator;

import javax.servlet.http.HttpServletRequest;

/**
 *  Class for holding and manipulating the name-value-pairs of the query
 *  and to construct html a-elements.
 *  @author Arno Aalto
 */
public class Link {
    private HashMap param;
    private String submit;
    private String page;

    public Link(HttpServletRequest request) {
        page = request.getRequestURI();
        param = new HashMap();
        Enumeration names = request.getParameterNames();
        while(names.hasMoreElements()) {
            // Only one value per name
            String name = (String) names.nextElement();
            // Special cases
            if ("submit".equals(name)) {
                submit = request.getParameter(name);
            } else {
                param.put(name, request.getParameter(name));
            }
        }
```

```
    }

    public String getPage() {
        return page;
    }

    public HashMap getMap() {
        return param;
    }

    public boolean isSubmit() {
        return (submit != null);
    }

    public String getSubmit() {
        return submit;
    }

    public String addParamStr(String paramStr, String name, String value) {
        if (paramStr.length() > 0) {
            paramStr += "&";
        }
        paramStr += URLEncoder.encode(name)  + "=" + URLEncoder.encode(value);
        return paramStr;
    }

    private String params(String name, String value) {
        String result = "";
        Iterator names = param.keySet().iterator();
        while(names.hasNext()) {
            String aName = (String) names.next();
            if (aName.equals(name)) {
                if (value != null) {
                    result = addParamStr(result, aName, value);
                }
            } else if (!aName.endsWith("ontonym") && !aName.endsWith("submit")){
                result = addParamStr(result, aName, (String) param.get(aName));
            }
        }
        if (name != null && value != null && !param.containsKey(name)) {
            result = addParamStr(result, name, value);
        }
        return result;
    }

    public void removeParam(String name) {
        param.remove(name);
    }

    public void addParam(String name, String value) {
        param.put(name, value);
    }

    public String getParam(String name) {
```

```
        return (String) param.get(name);
    }

    public String url(String page, String name, String value) {
        // page defaults to current page
        if (page == null) {
            page = getPage();
        }
        return page + "?" + params(name, value);
    }

    public String aHref(String target,
                        String page,
                        String name,
                        String value,
                        String label,
                        String title) {
        String result = "<a";
        if (target != null) {
            result += " target=\"" + target + "\" ";
        }
        result += " href=\"" + url(page, name, value) + "\"";
        if (title != null) {
            result += " title=\"" + title;
        }
        result += "\">" + label + "</a>";
        return result;
    }

    public String aHref(String target,
                        String page,
                        String name,
                        String value,
                        String label,
                        String title,
                        String value2) {
        String result = "<a";
        if (target != null) {
            result += " target=\"" + target + "\" ";
        }
            result += " href=\"" + url(page, name, value);
        if (target == "_top") {
            result = addParamStr(result, "serviceclassuri", value2);
        }
        result += "\"";

        if (title != null) {
            result += " title=\"" + title;
        }
        result += "\">" + label + "</a>";
        return result;
    }
}
```

3.6 OntologySearch

```
package fi.helsinki.cs.keltsi.tags;

import fi.helsinki.cs.keltsi.jenardf.Keltsi;
import fi.helsinki.cs.keltsi.jenardf.KeltsiDAML;
import fi.helsinki.cs.keltsi.jenardf.Ontology;
import fi.helsinki.cs.keltsi.jenardf.KeltsiClass;

import com.hp.hpl.jena.daml.DAMLClass;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.jsp.tagext.TagSupport;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.JspWriter;
import java.util.Iterator;

/**
 *  Control to browse an ontology.
 *  @author Arno Aalto
 */
public class OntologySearch extends TagSupport {
    protected Link link;
    protected KeltsiClass ontoClass;
    protected Html html;
    protected String[] verbs;
    protected String verb;
    protected KeltsiDAML model;
    protected KeltsiClass serviceClass;
    HttpServletRequest request;

    public int doStartTag() throws JspException {
        init();
        Iterator ontologies = model.getOntologies();
        while (ontologies.hasNext()) {
            Ontology ontology = (Ontology) ontologies.next();
            KeltsiClass ontoTop = ontology.getDomainTop();
            if(serviceClass.hasSuperClass(ontoTop)) {
                displayOntology(ontology);
            } else {
                link.removeParam(ontology.getLabel());
                link.removeParam(ontology.getLabel() + "verb");
            }
        }
        return SKIP_BODY;
    }

    private String searchControl(String name, String ontonym) {
        return "<input type=\"text\"/ "
                  +   "size=\"12\""
                  +   "value=\"" + ontonym + "\" "
                  +   "name=\"" + name + "ontonym\" />"
```

```
          + "<input type=\"submit\" "
               +   "name=\"" + name + "submit\" value=\"go!\" />";
    }

    private void displayOntology(Ontology ontology) {
        String[] verbs = null;
        verb = null;
        boolean hasOntoClass = false;
        KeltsiClass ontoClass = null;
        KeltsiClass ontoRoot;
        String ontoClassUri = null;
        String paramName = ontology.getLabel();

        html.write("<form action=\"" + link.url(null, paramName, null) + "\""
                       + "style=\"margin: 1 1 1 1 ; padding 1 1 1 1 \" "
                       + "method=\"post\">"
                 + "<table class=\"ontologytable\" "
                       + "cellspacing=\"0\" "
                       + "cellpadding=\"0\" "
                       + "border=\"0\" "
                       + "bgcolor=\"#eeeebb\" "
                       + "width=\"100%\">\n");

        try {
            /* Get value of parameter named for this ontology */
            ontoRoot = ontology.getRangeTop();
            ontoClassUri = link.getParam(paramName);

            /* Try to get corresponding KeltsiClass object */
            if (ontoClassUri  != null) {
                ontoClass = (KeltsiClass) model.getKeltsiResource(ontoClassUri);
            }
            verbs = ontology.getVerbs();

            hasOntoClass = (ontoClass != null);
            if (!hasOntoClass) {
                ontoClass = ontoRoot;
            }
            verb = link.getParam(paramName + "verb");
            if ((verb == null || verb.length() == 0)
                    && verbs != null
                    && verbs.length > 0) {
                // First verb is default
                verb = verbs[0];
                link.addParam(paramName + "verb", verb);
            }
        } catch(Exception ex) {
            html.errorMsg(ex);
            return;
        }

        if (hasOntoClass) {
            // write verbs as a list
            verbs = ontology.getVerbs();
```

```
            String verbList = "";
            if (verbs != null) {
                int verbCount = verbs.length;
                int index = 0;
                while(index < verbCount) {
                    String v = verbs[index];
                    verbList += verbLink(verbs[index], paramName + "verb");
                    ++index;
                }
            }
            html.write(
                html.bar(link.aHref(null, null, paramName, null, "x",
                                    "Close this ontology")
                         + " " + verbList));
            Iterator sClasses = ontoClass.getSuperClasses(true);
            KeltsiClass sClass;
            while (sClasses.hasNext()) {
                sClass = (KeltsiClass) sClasses.next();
                html.write(
                    html.tr(
                        html.td(
                            html.superClassLink(sClass, link, paramName),
                            null, "colspan=\"2\"")));
            }

            html.write(html.tr(html.td(html.thisClass(ontoClass),
                                       null, "colspan=\"2\"")));

            sClasses = ontoClass.getSubClassesSorted();
if (paramName == "location") {
            html.write(html.tr(html.td(html.optionList(sClasses, link, paramName),
                                       null, "colspan=\"2\"")));
} else {
                while (sClasses.hasNext()) {
                    sClass = (KeltsiClass) sClasses.next();
                    html.write(html.tr(html.td(
                            html.subClassLink(sClass, link, paramName))));
                }
            }
        } else {
            String label = ontoClass.getLabel();
            html.write(
                html.bar(link.aHref(null, null, paramName, ontoRoot.getURI(),
                                    label, "Open " + label),
                         searchControl(paramName, "")));
        }
        // Separator between ontologyviews
        // html.write("<tr><td colspan=\"2\"><hr /></td></tr>");
        html.write("</table>\n</form>\n");
    }

    private String verbLink(String value, String paramVerb)  {
        if (value.equals(verb)) {
            return "<b>" + value + "</b> ";
```

```
        } else {
            return link.aHref(null, null, paramVerb, value, value, null)
                   + " ";
        }
    }


    private void init() {
        try {
            request =
                    (HttpServletRequest)pageContext.getRequest();
            html = new Html(pageContext.getOut());
            link  = (Link) pageContext
                    .getAttribute("fi.helsinki.cs.keltsi.ServiceClassLink");

            model = (KeltsiDAML) pageContext
                    .findAttribute("fi.helsinki.cs.keltsi.KeltsiDAML");

            serviceClass = (KeltsiClass) pageContext
                    .findAttribute("fi.helsinki.cs.keltsi.ServiceClass");
            if(serviceClass == null) {
                throw new Exception("No serviceclass found!");
            }
        } catch(Exception ex) {html.errorMsg(ex);}
    }

}
```

3.7 OtherServices

```
package fi.helsinki.cs.keltsi.tags;

import fi.helsinki.cs.keltsi.util.*;
import fi.helsinki.cs.keltsi.jenardf.*;

import com.hp.hpl.jena.daml.*;
import com.hp.hpl.mesa.rdf.jena.model.*;
import com.hp.hpl.jena.vocabulary.DAML_OIL;
import com.hp.hpl.mesa.rdf.jena.vocabulary.RDF;
import com.hp.hpl.mesa.rdf.jena.vocabulary.RDFS;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.jsp.tagext.TagSupport;
import javax.servlet.jsp.JspException;
import java.util.*;
import javax.servlet.jsp.PageContext;

/*
 * Display links to other Services associated with
 * the one in focus.
 */
public class OtherServices extends TagSupport {
```

```
private Service service;
private Link link;
private KeltsiDAML model;
private Html html;
private Advertisement adv;

private void errorMsg(String msg) {
    try {
        pageContext.getOut().write("<pre>" + msg + "</pre>");
    } catch(Exception ex) {}
}

public int doStartTag() throws JspException {
    String aHref2instance = null;
    String instanceURI = "";

    html = new Html(pageContext.getOut());
    html.write("<!-- OtherServices starting -->\n");

    /* Get Instance URI as a parameter. */
    try {
        HttpServletRequest req =
          (HttpServletRequest)pageContext.getRequest();

        link = new Link(req);

        instanceURI = req.getParameter("serviceuri");
    } catch (Exception ex) {
        html.errorMsg(ex, "OtherServices initializing");
    }

    /* Get the model from scope. */
    try {
        model = (KeltsiDAML) pageContext.findAttribute(
                "fi.helsinki.cs.keltsi.KeltsiDAML");
    } catch (Exception ex) {
        html.errorMsg(ex, "getting model from scope");
    }

    /* Get service instance. */
    service = (Service) model.getKeltsiResource(instanceURI);
    if (service == null) {
        html.errorMsg("Instance " + instanceURI + " was not found.");
    } else {
        try {
            adv = service.getAdvertisement();
            if (adv == null) {
                throw new Exception("No advertisement found!");
            }
        } catch(Exception ex) {
            html.errorMsg(ex);
        }
        displayServices(adv.getServices(), "Other services by");
```

```
        }
        displayServices(service.getServiceValues(Keltsi.provides),
                    "Also provided by");
        displayServices(service.getServiceValues(Keltsi.additionalServices),
                    "Also available at");
        displayServices(service.getServiceValues(Keltsi.recommends),
                    "Recommended by");
        html.write("\n<!-- OtherServices done -->\n");
        return SKIP_BODY;
    }

    private void displayServices(Iterator iter, String label) {
        Service serviceInstance = null;
        boolean first = true;
        while (iter.hasNext()) {
            serviceInstance = (Service) iter.next();
            if (!service.equals(serviceInstance)) {
                if (first) {
                    html.write(
                            html.hr()
                            + label + " <b>"
                            + adv.getLabel() + ":</b>"
                            + html.br());
                    first = false;
                }
                html.write(html.serviceLink(serviceInstance, link) + html.br());
            }
        }
    }

}
```

3.8 Search

```
package fi.helsinki.cs.keltsi.tags;

import fi.helsinki.cs.keltsi.util.*;
import fi.helsinki.cs.keltsi.jenardf.*;

import com.hp.hpl.jena.daml.*;
import com.hp.hpl.mesa.rdf.jena.model.*;
import com.hp.hpl.mesa.rdf.jena.common.*;

import java.io.*;
import java.util.Iterator;
import java.util.HashMap;

import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.jsp.tagext.TagSupport;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.JspWriter;
```

```
/**
 *  Display search-link, count of results or resultlist
 *  @author Arno Aalto
 */
public class Search extends TagSupport {
    protected Link link;
    protected String page;
    protected Html html;
    protected KeltsiDAML model;

    public int doStartTag() throws JspException {
        init();
        html.write("<table class=\"servicetable\" "
                        + "cellspacing=\"0\" "
                        + "cellpadding=\"2\" "
                        + "width=\"100%\">\n");
        if (link.isSubmit()) {
            resultList();
        } else {
            html.write(html.tr(html.td(
                    submitControl()
                    + " (" + resultCount() + ")")));
        }
        html.write("</table>\n");
        return SKIP_BODY;
    }

    protected String submitControl() {
        return "<hr width=\"30\" /><font size='4' color='#ffff00'>"
               + link.aHref(null, page, "submit", "do", "list results",
                            "List the matching services")
               + "</font>";
    }

    protected void resultList() {
        Service service = null;
        String list = "";
        Iterator results = model.mapQuery(link.getMap(), true);
        if (results == null) {
            // TODO: Display something?
        } else {
            int count = 0;
            while(results.hasNext()) {
                ++ count;
                service = (Service) results.next();
                list += html.tr(html.td(
                        html.serviceLink(service, link)));
            }
            html.write(list);
        }
    }

    protected String resultCount() {
```

```
        int count = model.mapQuerySize(link.getMap(), true);
        return new Integer(count).toString();
    }

    protected void init() {
        try {
            HttpServletRequest req =
                    (HttpServletRequest)pageContext.getRequest();
            page = req.getRequestURI();
            html = new Html(pageContext.getOut());
            link  = (Link) pageContext
                    .getAttribute("fi.helsinki.cs.keltsi.ServiceClassLink");
            model = (KeltsiDAML) pageContext
                    .findAttribute("fi.helsinki.cs.keltsi.KeltsiDAML");
        } catch(Exception ex) {
            html.errorMsg(ex);
        }
    }
}
```

3.9 SemanticLinks

```
package fi.helsinki.cs.keltsi.tags;

import fi.helsinki.cs.keltsi.util.*;
import fi.helsinki.cs.keltsi.jenardf.*;

import com.hp.hpl.jena.daml.*;
import com.hp.hpl.mesa.rdf.jena.model.*;
import com.hp.hpl.jena.vocabulary.DAML_OIL;
import com.hp.hpl.mesa.rdf.jena.vocabulary.RDF;
import com.hp.hpl.mesa.rdf.jena.vocabulary.RDFS;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.jsp.tagext.TagSupport;
import javax.servlet.jsp.JspException;
import java.util.*;
import javax.servlet.jsp.PageContext;

public class SemanticLinks extends TagSupport {

    private KeltsiDAML model;
    private Link link;
    private Html html;
    private Service service = null;
    private String serviceClassURI = "";
    private static final String[][] propURIs = {
        {"service",         "http://protege.stanford.edu/Keltsi#franchiseeOf"},
        {"serviceinverse",  "http://protege.stanford.edu/Keltsi#franchisee"},
        {"productclass",    "http://protege.stanford.edu/Keltsi#rents"},
        {"productclass",    "http://protege.stanford.edu/Keltsi#retails"},
        {"productclass",    "http://protege.stanford.edu/Keltsi#maintains"},
```

```
        {"serviceclass",    "http://protege.stanford.edu/Keltsi#franchises"},
        {"location",        "http://protege.stanford.edu/Keltsi#location"},
        {"restauranttype",  "http://protege.stanford.edu/Keltsi#restaurantType"},
        {"ambiance",        "http://protege.stanford.edu/Keltsi#ambiance"},
        {"rating",          "http://protege.stanford.edu/Keltsi#rating"}};

    public DAMLProperty getKeltsiProperty(String uri) {
        return (DAMLProperty) model.getDAMLValue(uri);
    }

    public int doStartTag() throws JspException {
        DAMLProperty prop = null;

        String serviceURI = "";
        html = new Html(pageContext.getOut());
        html.write("<!-- SemanticLinks starting -->\n");

        /* Get Instance URI as a parameter. */
        try {
            HttpServletRequest req =
              (HttpServletRequest)pageContext.getRequest();

            link = new Link(req);

            serviceURI = req.getParameter("serviceuri");

        } catch (Exception ex) {
            html.errorMsg(ex, "while getting request params.");
        }

        /* Get the model from scope. */
        try {
            model = (KeltsiDAML) pageContext.findAttribute(
                    "fi.helsinki.cs.keltsi.KeltsiDAML");
            if (model == null) {
                throw new Exception("Model is null!");
            }
        } catch (Exception ex) {
            html.errorMsg(ex, "while getting the model from scope");
        }

        /* Get service instance. */
        service = (Service) model.getKeltsiResource(serviceURI);
        if (service == null) {
            html.errorMsg("Instance " + serviceURI + " was not found.");
        }
        else {
            /* Get Keltsi properties. */
            html.write("<table>\n");
            for(int i=0; i<propURIs.length; ++i) {
                if (propURIs[i][0] != "serviceinverse") {
                    prop = getKeltsiProperty(propURIs[i][1]);
                } else {
                    prop = model.getInverseProperty(propURIs[i][1]);
```

```
                }
                if (prop == null) {
                    html.errorMsg("Property " + propURIs[i][1] + " was not found.");
                } else {
                    /* Print Keltsi properties and their values. */
                    writeValues(prop, propURIs[i][0]);
                }
            }
            html.write("</table>\n");
        }
        html.write("\n<!-- SemanticLinks done -->\n");
        return SKIP_BODY;
    }

    private void writeValues(DAMLProperty prop, String propertyType) {
        Iterator valuesIter = null;
        Service serviceInstance = null;
        String list = "";
        KeltsiClass kClass = null;

        /* Get values of property. */
        if (propertyType == "service") {
            valuesIter = service.getServiceValues(prop);
        } else if(propertyType == "serviceinverse") {
            valuesIter = service.getServiceInverseValues(prop);
        } else {
            valuesIter = service.getClassValues(prop);
        }
        if (valuesIter != null) {
            String propertyName = prop.getLocalName();
            if (propertyType == "serviceinverse") {
                propertyName = model.inverseName(propertyName);
            }
            int rows = 0;
            while(valuesIter.hasNext()) {
                if (propertyType == "service" || propertyType == "serviceinverse") {
                    serviceInstance = (Service) valuesIter.next();
                    list += Html.td(Html.serviceLink(serviceInstance, link));
                } else if(propertyType == "serviceclass") {
                    kClass = (KeltsiClass) valuesIter.next();
                    list += Html.td(Html.superClassLink(kClass, link, "serviceclassuri"));
                } else {
                    kClass = (KeltsiClass) valuesIter.next();
                    list += Html.td(Html.superClassLink(kClass, link, propertyType));
                }
                rows++;
            }
            if (rows > 0) {
                html.write(html.tr(html.td(propertyName, null,
                                           "rowspan=\"" + rows + "\"")
                                   + list));
            }
        }
    }
```

```
}
```

3.10 ServiceClassInit

```
package fi.helsinki.cs.keltsi.tags;

import fi.helsinki.cs.keltsi.jenardf.KeltsiDAML;
import fi.helsinki.cs.keltsi.jenardf.Keltsi;
import fi.helsinki.cs.keltsi.jenardf.KeltsiClass;
import fi.helsinki.cs.keltsi.jenardf.Ontology;

import java.util.Iterator;
import java.io.PrintWriter;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.jsp.tagext.TagSupport;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.JspWriter;

/**
 *  Initialize ServiceClass
 *      to be used with classes such as DisplayServiceClass
 *  @author Arno Aalto
 */
public class ServiceClassInit extends TagSupport {

    public int doStartTag() throws JspException {
        KeltsiDAML model = null;
        try {
            HttpServletRequest request =
                    (HttpServletRequest)pageContext.getRequest();
            Link link = new Link(request);
            model = (KeltsiDAML) pageContext
                .findAttribute("fi.helsinki.cs.keltsi.KeltsiDAML");

            /* Get value of parameter 'serviceclassuri' */
            String serviceClassUri = null;

            /*   First try synonym */
            String synonym = request.getParameter("serviceclasssynonym");
            if (synonym != null && synonym.length() > 0) {
                serviceClassUri = model.getSynonymClassURI(synonym);
            }
            /* Then try direct parametervalue */
            if (serviceClassUri == null) {
                serviceClassUri = request.getParameter("serviceclassuri");
            }
            link.removeParam("synonym");

            /* Try to get corresponding ServiceClass object */
            KeltsiClass serviceClass = null;
            if (serviceClassUri  != null) {
                serviceClass = (KeltsiClass) model
```

```
                    .getKeltsiResource(serviceClassUri);
            }

            /* Missing or invalid value of parameter 'serviceclassuri'
            Default value is the root of the Servicetree.
            TODO: Check if it should be something else
            */
            if (serviceClass == null) {
                serviceClass = Keltsi.ServiceRoot;
                serviceClassUri = serviceClass.getURI();
            }
            link.addParam("serviceclassuri", serviceClassUri);

            Iterator ontologies = model.getOntologies();
            while (ontologies.hasNext()) {
                Ontology ontology = (Ontology) ontologies.next();
                String paramName = ontology.getLabel();
                KeltsiClass ontoRoot = ontology.getRangeTop();
                String ontonym = request.getParameter(paramName + "ontonym");
                String ontoClassUri = null;
                if (ontonym != null && ontonym.length() > 0) {
                    ontoClassUri = model.getSynonymClassURI(ontonym, ontoRoot);
                    if (ontoClassUri != null) {
                        link.addParam(paramName, ontoClassUri);
                    }
                    link.removeParam(paramName + "ontonym");
                }
            }


            pageContext.setAttribute("fi.helsinki.cs.keltsi.ServiceClassLink",
                                     link);
            pageContext.setAttribute("fi.helsinki.cs.keltsi.ServiceClass",
                                     serviceClass);
        } catch(Exception ex) {
            try {
                PrintWriter pw = new PrintWriter(pageContext.getOut(),true);
                ex.printStackTrace(pw);
            } catch(Exception ex2) {}
        }
        return SKIP_BODY;
    }
}
```

4 fi.helsinki.cs.keltsi.tags.test

4.1 TestDisplayAdvertisement

```
package fi.helsinki.cs.keltsi.tags.test;

import com.meterware.httpunit.*;

import fi.helsinki.cs.keltsi.jenardf.*;
import fi.helsinki.cs.keltsi.util.*;
```

```
import java.io.IOException;
import java.net.MalformedURLException;

import org.xml.sax.*;
import org.w3c.dom.*;

import junit.framework.*;

import java.util.*;

import com.hp.hpl.jena.daml.*;
import com.hp.hpl.mesa.rdf.jena.model.*;
import com.hp.hpl.mesa.rdf.jena.common.*;

/**
 * An example of testing servlets using httpunit and JUnit.
 */
public class TestDisplayAdvertisement extends TestCase {

   public static void main(String args[]) {
      junit.textui.TestRunner.run( suite() );
   }

   public static Test suite() {
      return new TestSuite( TestDisplayAdvertisement.class );
   }

   public TestDisplayAdvertisement(String name) {
      super(name);
   }

   /**
    * Verifies that the welcome page has exactly one form,
    * with the single parameter, "name"
    */
   public void testMainPage() throws Exception  {
      WebConversation conversation = new WebConversation();
      WebRequest request = new GetMethodWebRequest(
              "http://kuussaari.cs.helsinki.fi:5123/keltsi/main.jsp" );
      WebResponse response = conversation.getResponse(request);
      WebForm forms[] = response.getForms();
      String parameters = "";
      KeltsiDAML kd;
      DAMLProperty aPredicate;
      RDFNode aObject;
      ResultIterator instanceiter;
      Service service;
      Advertisement adv;

      /* NodeList nodelist = response.getDOM().getElementsByTagName("input");

      LinkedList ll = new LinkedList();
      for(int i=0; i<nodelist.getLength(); ++i) {
```

```
Node node = nodelist.item(i);
ll.add(node.getAttributes().getNamedItem("name").getNodeValue());

}*/

    /* Check parameter names. */
    parameters= forms[0].getParameterNames()[0]
            + forms[0].getParameterNames()[1]
            + forms[0].getParameterNames()[2];
    assertEquals( 3, forms[0].getParameterNames().length );
    assertTrue( parameters.indexOf("rdfPredicate") >= 0);
    assertTrue( parameters.indexOf("rdfSubject") >=0);
    assertTrue( parameters.indexOf("rdfObject") >=0);

    /* Query KeltsiDAML with form. */
    request.setParameter( "rdfPredicate", "Franchises" );
    request.setParameter( "rdfObject", "HairCuttingOrColorServices" );
    request = forms[0].getRequest();
    response = conversation.getResponse( request );

    /* Query direct from KeltsiDAML. */
    kd = new KeltsiDAML();;
    kd.loadModel("KeltsiData");

    aPredicate = (DAMLProperty) kd.getDAMLValue(
            kd.keltsiNS + "Franchises");

    aObject = (RDFNode) kd.getDAMLValue(
            kd.keltsiNS + "HairCuttingOrColorServices");

    instanceiter =
kd.select(null, (Property) aPredicate, aObject);

    /* Compare query results. */
    while(instanceiter.hasNext() == true) {
service = Service
    .getService((DAMLInstance) instanceiter.next());
adv = service.getAdvertisement();
       assertTrue(response.getText()
                           .indexOf(adv.getProviderOfService()) != -1);
    }

 /**
  * Verifies that submitting the login form without entering
  * a name results in a page containing the text "Login failed"
  */
  /*   public void testBadLogin() throws Exception {

    WebConversation     conversation = new WebConversation();
    WebRequest  request = new GetMethodWebRequest(
            "http://www.meterware.com/servlet/TopSecret" );

    WebResponse response = conversation.getResponse( request );
    WebForm loginForm = response.getForms()[0];
```

```
      request = loginForm.getRequest();
      response = conversation.getResponse( request );
      assertTrue( "Login not rejected", response.getText().indexOf(
             "Login failed" ) != -1 );
     }*/



   /**
    * Verifies that submitting the login form with the name "master" results
    * in a page containing the text "Top Secret"
    */
    /*   public void testGoodLogin() throws Exception {

      WebConversation     conversation = new WebConversation();
      WebRequest  request = new GetMethodWebRequest( "http://www.meterware.com/servlet/TopSecret"

      WebResponse response = conversation.getResponse( request );
      WebForm loginForm = response.getForms()[0];
      request = loginForm.getRequest();
      request.setParameter( "name", "master" );
      response = conversation.getResponse( request );
      assertTrue( "Login not accepted", response.getText().indexOf( "You made it!" ) != -1 );

      assertEquals( "Page title", "Top Secret", response.getTitle() );
   }
    */
   }
}
```

5 fi.helsinki.cs.keltsi.util

5.1 ResultIterator

```
package fi.helsinki.cs.keltsi.util;

import java.util.Iterator;
import fi.helsinki.cs.keltsi.jenardf.*;
import com.hp.hpl.mesa.rdf.jena.model.StmtIterator;
import com.hp.hpl.mesa.rdf.jena.model.Statement;

public class ResultIterator implements java.util.Iterator
{
    private StmtIterator sIter;
    private int select;

    public ResultIterator(StmtIterator sIter, int select)
    {
        this.sIter = sIter;
        this.select = select;
    }
```

```
public int getSelect() {
    return select;
}

public boolean hasNext()
{
    if (sIter == null) return false;
    try
    {
        return sIter.hasNext();
    }
    catch (Exception ex)
    {
        return false;
    }
}

public void remove()
{
    if (sIter != null)
    try
    {
        sIter.remove();
    }
    catch (Exception ex)
    {
    }
}

public Object next()
{
    try {
        Statement stmt = sIter.next();
        switch (select)
        {
            case KeltsiDAML.SUBJECT: return stmt.getSubject();
            case KeltsiDAML.PREDICATE: return stmt.getPredicate();
            case KeltsiDAML.OBJECT: return stmt.getObject();
            default: return stmt;
        }
    }
    catch (Exception ex) {
        return null;
    }
}

public void close()
{
    try {
        if (sIter != null) sIter.close();
    }
    catch (Exception ex) {
    }
```

```
    }

}
```