Servicios Web con Eclipse, II

Introducción

El proyecto Eclipse Web Tools Platform tiene por objetivo extender la funcionalidad del ambiente integrado de desarrollo Eclipse con herramientas para desarrollar aplicaciones basadas en Java EE. Dentro de las herramientas disponibles, WTP proporciona algunas herramientas específicas para la generación, publicación y consumo de servicios Web.
En la versión ECLIPSE que estamos manejando ya está integrada.

Objetivos

• Crear un Servicio Web mediante el método **BOTTOM-UP** capaz de interactuar con una "base de conocimiento", que reside en la Web. (ver anexo).

•Incluir nuevas librerías dentro de Eclipse.

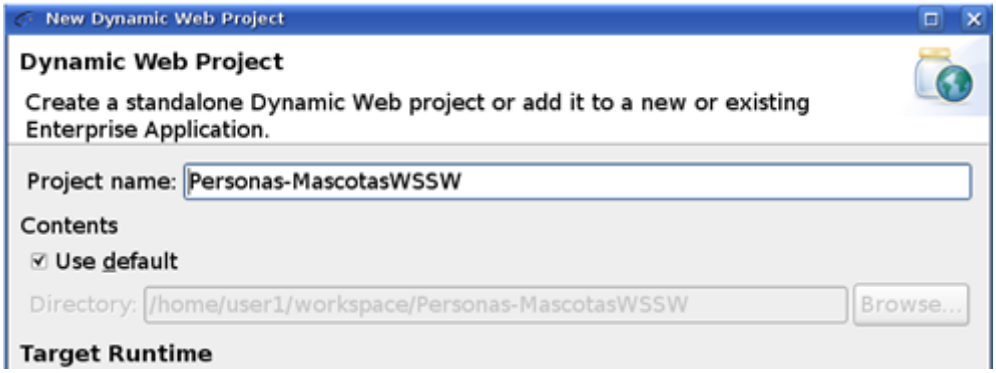
• Establecimiento de la lógica de negocio mediante un lenguaje de consultas.

• Probar el servicio web y monitorizar el intercambio de mensajes (SOAP).

Desarrollo de la práctica

1 En primer lugar, se crea un nuevo proyecto en Eclipse del tipo "Dynamic Web Project" con el nombre "**Personas-MascotasWSSW**".

Tened en cuenta que deberá estar seleccionada la perspectiva Java EE en le opción del menú Window / Open Perspective , para que sea posible seleccionar esta opción en la opción File / New.



Se pulsa en "Finish" y nos creará un proyecto vacío. A continuación, creamos un paquete denominado **wssw** (Botón derecho en Java Resources: src > new Package) en el que añadiremos un fichero **Persona** (botón derecho encima del paquete creado > new Class) con el siguiente código fuente.

```
package wssw;


import com.hp.hpl.jena.ontology.OntModel;

import com.hp.hpl.jena.ontology.OntModelSpec;

import com.hp.hpl.jena.query.Query;

import com.hp.hpl.jena.query.QueryExecution;

import com.hp.hpl.jena.query.QueryExecutionFactory;

import com.hp.hpl.jena.query.QueryFactory;

import com.hp.hpl.jena.query.QuerySolution;

import com.hp.hpl.jena.query.ResultSet;

import com.hp.hpl.jena.rdf.model.ModelFactory;

import com.hp.hpl.jena.rdf.model.RDFNode;
```

```
public class Persona {

  public String obtenerNombreMascota(String nombreDuenyo) {
        OntModel m = null;
        String resultado = "";
        // Utilizaremos razonamiento OWL-DL
        m = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM_MINI_RULE_INF );
        m.read("http://protege.cim3.net/file/pub/ontologies/people.pets/people+pets.owl");

        // Realizamos la consulta: Obtener el nombre de la mascota de la persona que se indica
        String queryString =
                        "PREFIX po: <http://cohse.semanticweb.org/ontologies/people#>" +
                        "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>" +
                        "SELECT ?nombre " +
                        "WHERE " +
                        "{ " +
                        " po:" + nombreDuenyo + " po:has_pet ?mascota ." +
                        " ?mascota rdfs:label ?nombre ." +
                        "}";

        Query query = QueryFactory.create(queryString);

        QueryExecution qe = QueryExecutionFactory.create(query, m);
        try{
                ResultSet results = qe.execSelect();
                StringBuilder result = new StringBuilder();

                while (results.hasNext()){
```

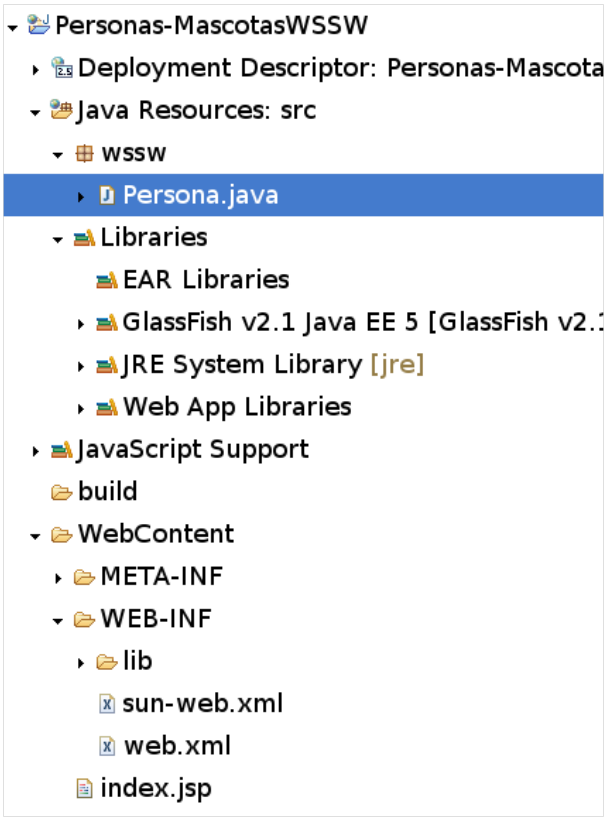
```
                    QuerySolution soln = results.nextSolution();
                    RDFNode title = soln.get("nombre");
                    result.append(title.toString());
                    result.append(" - ");
                }

                resultado += result;
            }finally{
                qe.close();
            }

            return resultado;
        }
}
```

En dicho código se utilizan las librerías de Jena, por lo que, se deben añadir al proyecto.

Dichas librerias (se encuentran dentro de un fichero jena-2.6.2.zip) se obtienen desde el sitio Web de Jena en **http://sourceforge.net/projects/jena/files/Jena/**. Tras descomprimir dicho fichero, y teniendo en cuenta que posteriormente se tiene que desarrollar un servicio web, una forma de proceder puede ser copiar todos los .jar desde ***directorio_de_descarga_de_Jena-2.6.2/lib*** a ***workspace_de_eclipse/Personas-MascotasWSSW/WebContent/WEB-INF/lib.***

Nota: otra posibilidad es incluir las librerías con el método habitual.

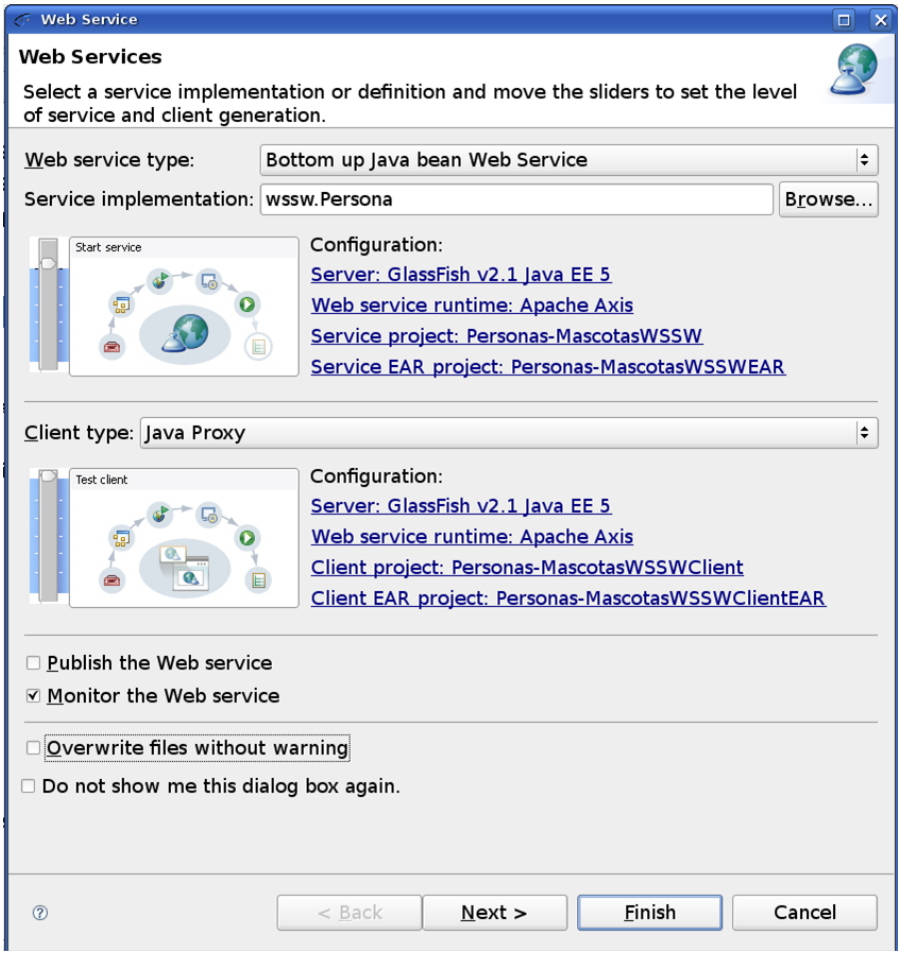
A continuación, situándonos en el proyecto, se realiza un 'Refresh' (F5), con lo que, desaparecerán los errores que se producían al no detectar las librerías de Jena.



En este momento, ya podemos proceder a la creación de un servicio Web, junto con un cliente Web para probarlo.

Nos situamos en el 'Project Explorer' de Eclipse sobre el fichero ***Persona.java***, y pulsando el botón derecho del ratón, seleccionamos la opción 'Web Services > Create Web service'. Se nos mostrará un cuadro de diálogo como el siguiente, en el cual, se indicarán las opciones tal y como se muestran en la ilustración. Finalmente, se pulsa en 'Finish', con lo que, se procederá, de forma automática, a la creación del servicio Web.

OJO: el Scroll del Cliente deberá estar en lo alto, para que la parte del cliente se cree.



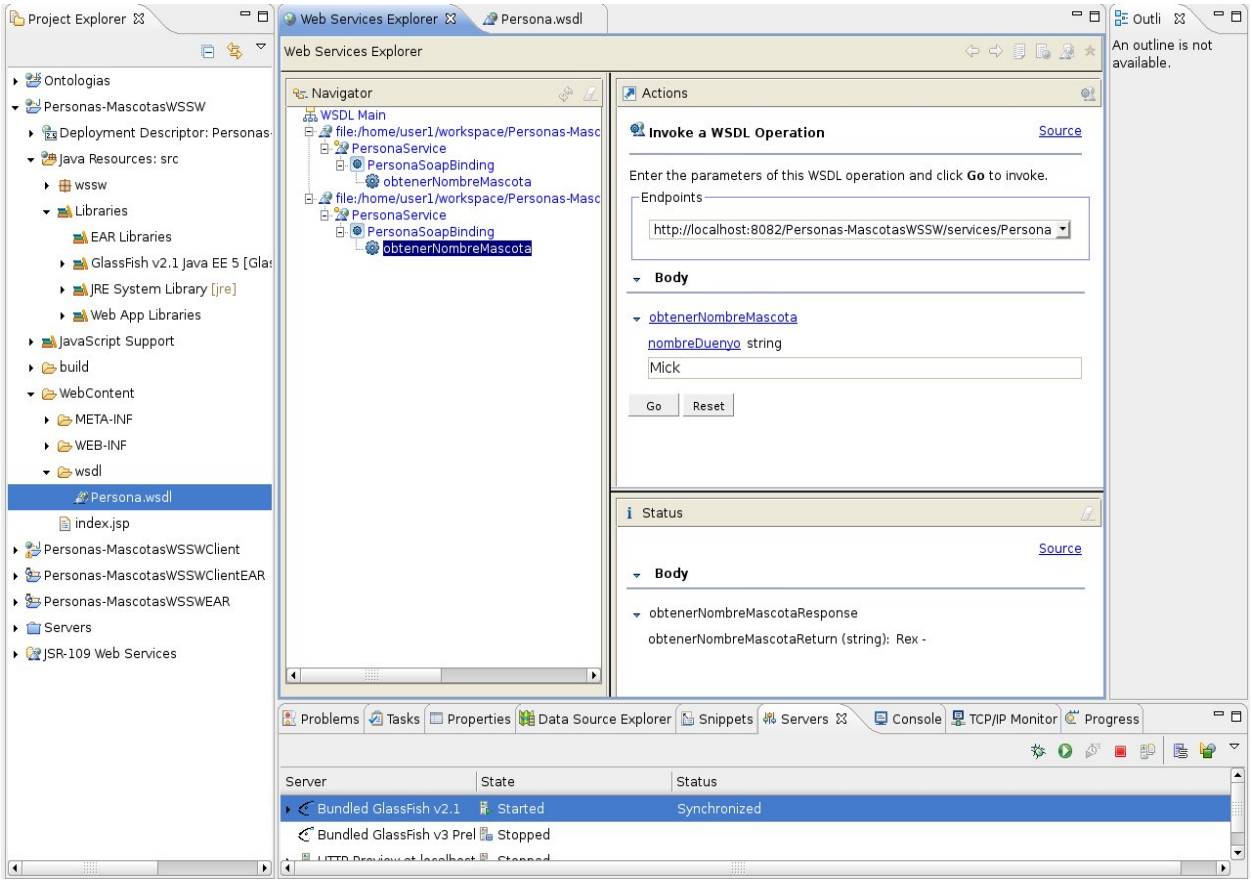
Como resultado se tendrá creado el servicio Web en el proyecto **'Personas-MascotasWSSW'**, y el cliente Web en **'Personas-MascotasWSSWClient'**.

2 TESTEO DEL SERVICOS WEB

Para comprobar que el servicio se ha generado correctamente y está desplegado y disponible en el servidor de aplicaciones JBoss v5.0, se puede realizar la siguiente prueba:

Nos situamos sobre el fichero descriptor del contrato (**Persona.wsdl**) y seleccionamos 'Boton derecho > Web Services > Test with Web Services Explorer'. En ese momento se abrirá el explorador de servicios Web, con lo que, podremos navegar entre los distintos servicios publicados y sus métodos. Para probar el método '**obtenerNombreMascota**', pulsamos sobre su enlace y le introducimos el nombre Mick y, a continuación, pulsamos en el botón 'Go'.

En la ventana de status podemos ver los mensajes SOAP que nos ha generado la acción, tanto el que se enviaría del cliente al servidor como el resultado que envía el servidor al cliente. Como se puede apreciar en este último viaja la respuesta del servidor: Rex.



Por último, también podemos ver donde se encuentra desplegado el servicio, ya que, el contrato del servicio (WSDL) lo especifica:

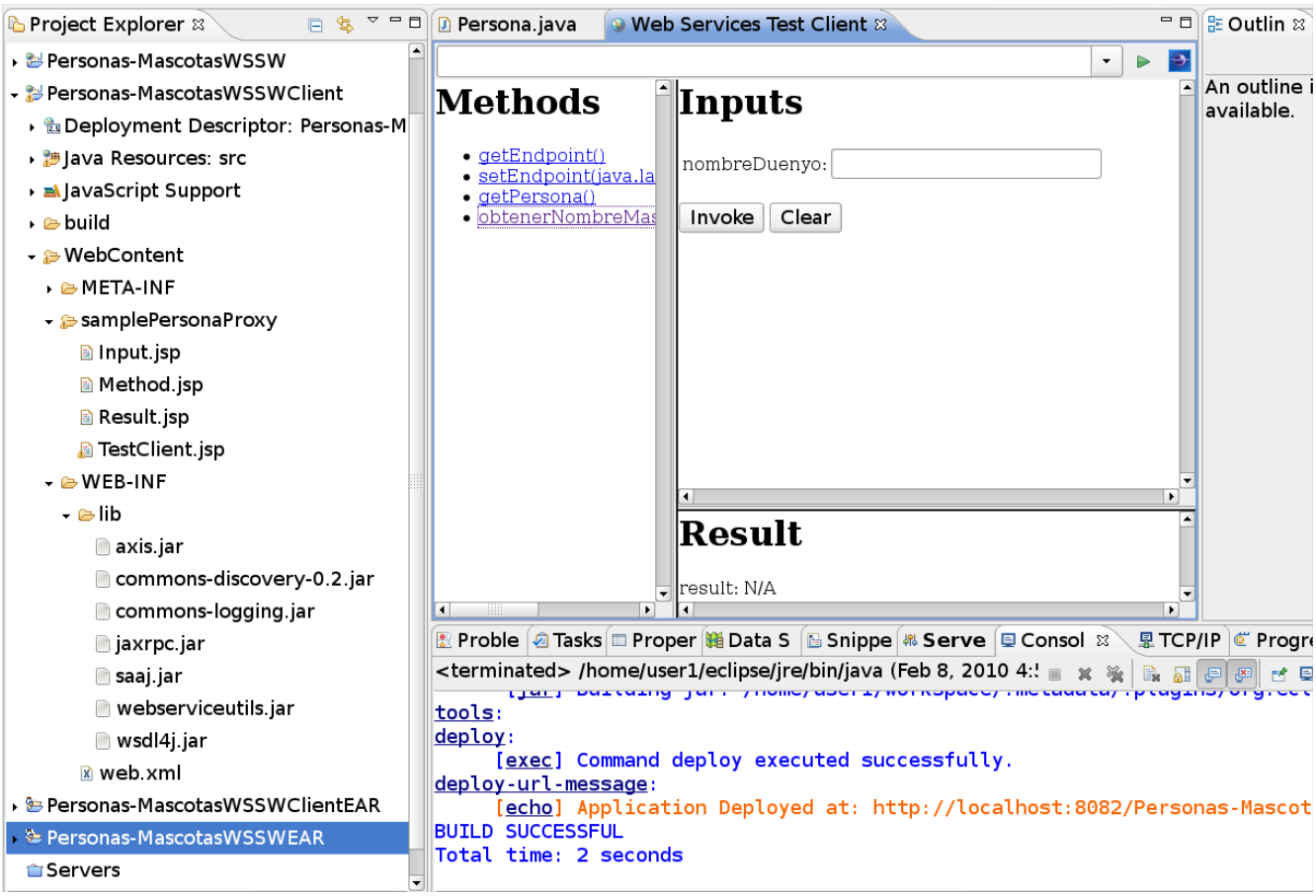
```
<wsdlsoap:address location="http://localhost:8082/Personas-MascotasWSSW/services/Persona"/>
```

Otra forma de probar nuestro servicio Web es mediante el método **'obtenerNombreMascota(nombre_del_dueño**' que se puede seleccionar a la izquierda de la interfaz que aparece en la dirección

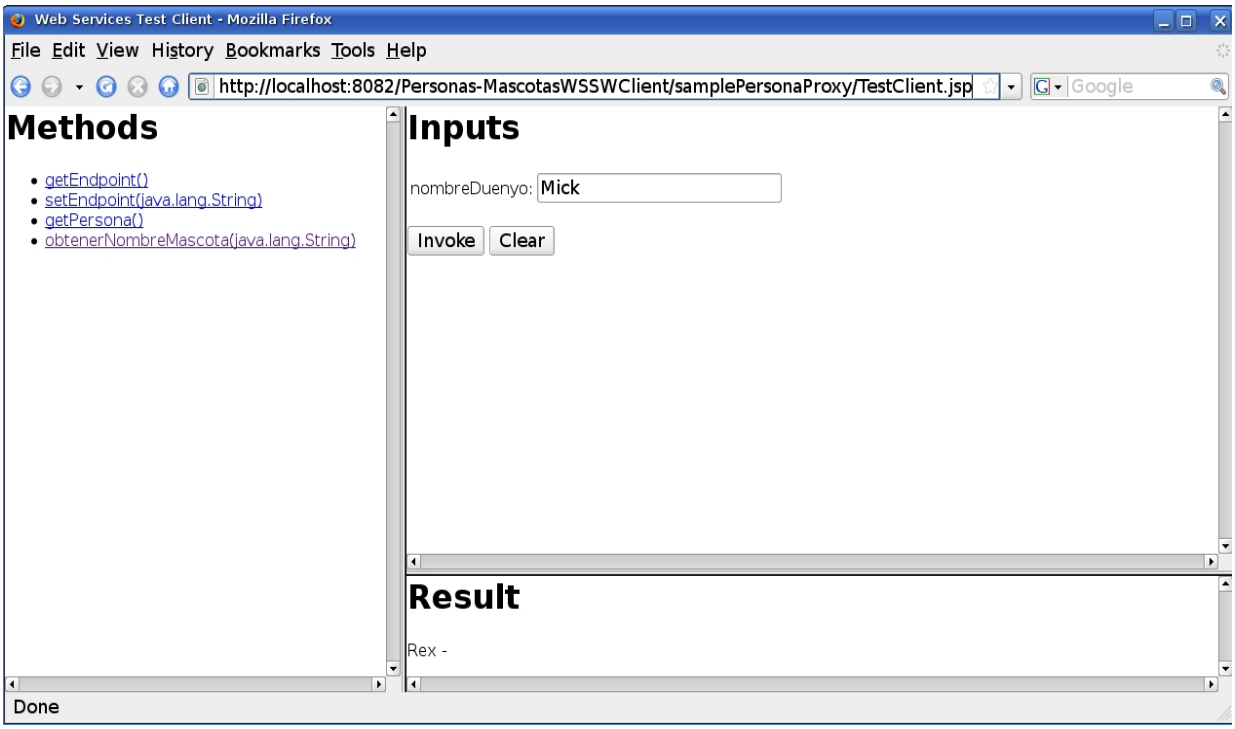
http://localhost:8082/Personas-MascotasWSSWClient/samplePersonaProxy/TestClient.jsp.

Si ponemos como nombre del dueño Mick y pulsamos en el botón etiquetado como 'Invoke', se tendrá como resultado Rex en la parte inferior de la interfaz.

Dicha interfaz se muestra tanto en el propio eclipse, como en la url anteriormente mencionada, y queda reflejado en las dos ilustraciones siguientes:



De no abrirse la pantalla con métodos (figura anterior) , seleccionar **TestClient.jsp** y con el botón derecho Run As > Run on Server.



Notar que se ha obtenido dicho resultado gracias a lo siguiente:

Se indica que "Rex es la mascota de Mick" (**Rex is_pet_of Mick**) (aunque no se tiene explícitamente que "Mick es el dueño de Rex" (**Mick has_pet Rex**)).

Se tienen definidas en la ontología que la propiedad "**is_pet_of**" es inversa de "**has_pet**".

Uso de un razonador OWL dentro del servicio Web (herramienta que aplica lógica y reglas a este tipo de conocimiento para obtener resultados, a menundo no explícitos).

Es decir, el razonador ha inferido la respuesta, de modo que hemos obtenido un conocimiento no explicito, a partir de otro¡¡¡¡¡¡¡¡

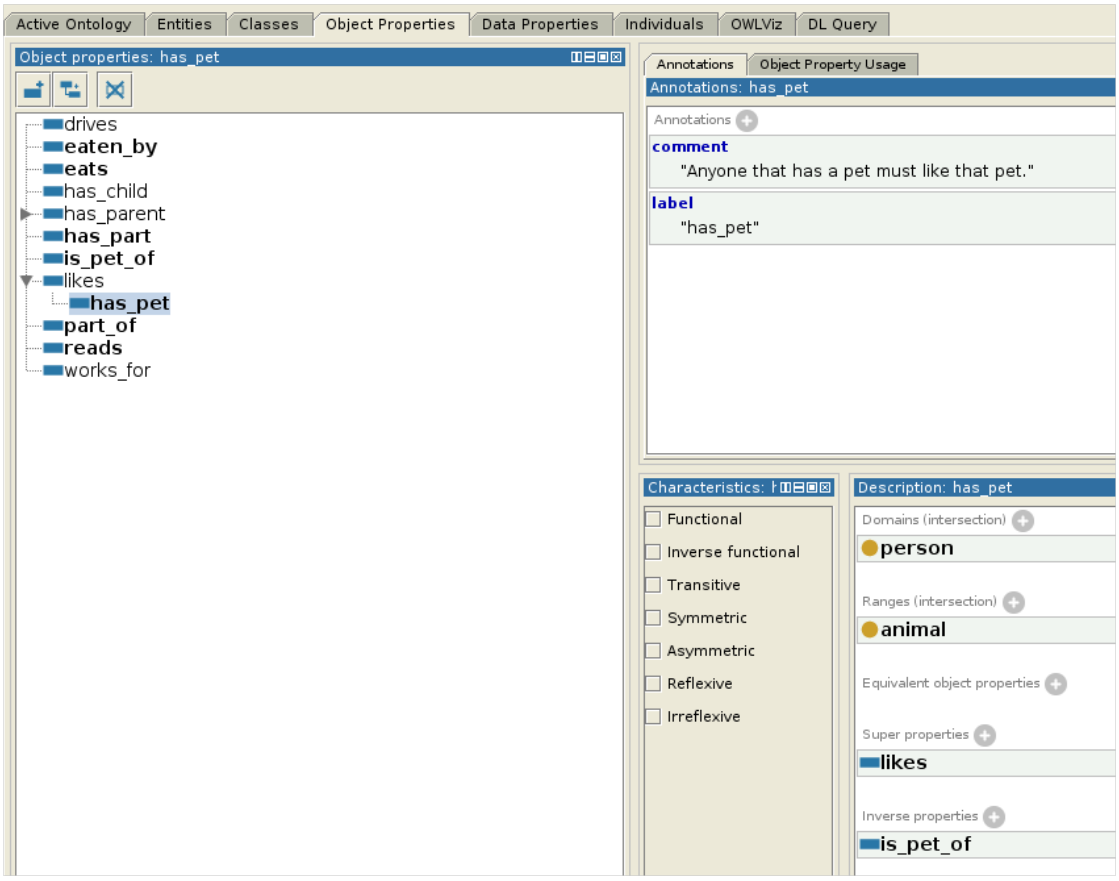
ANEXO: USO DE LA HERRAMIENTA PROTÉGÉ PARA VISUALIZAR LA BASE DE CONOCIMIENTO USADA EN EL SERVICIO WEB.

1. EDITOR DE ONTOLOGÍAS PROTÉGÉ

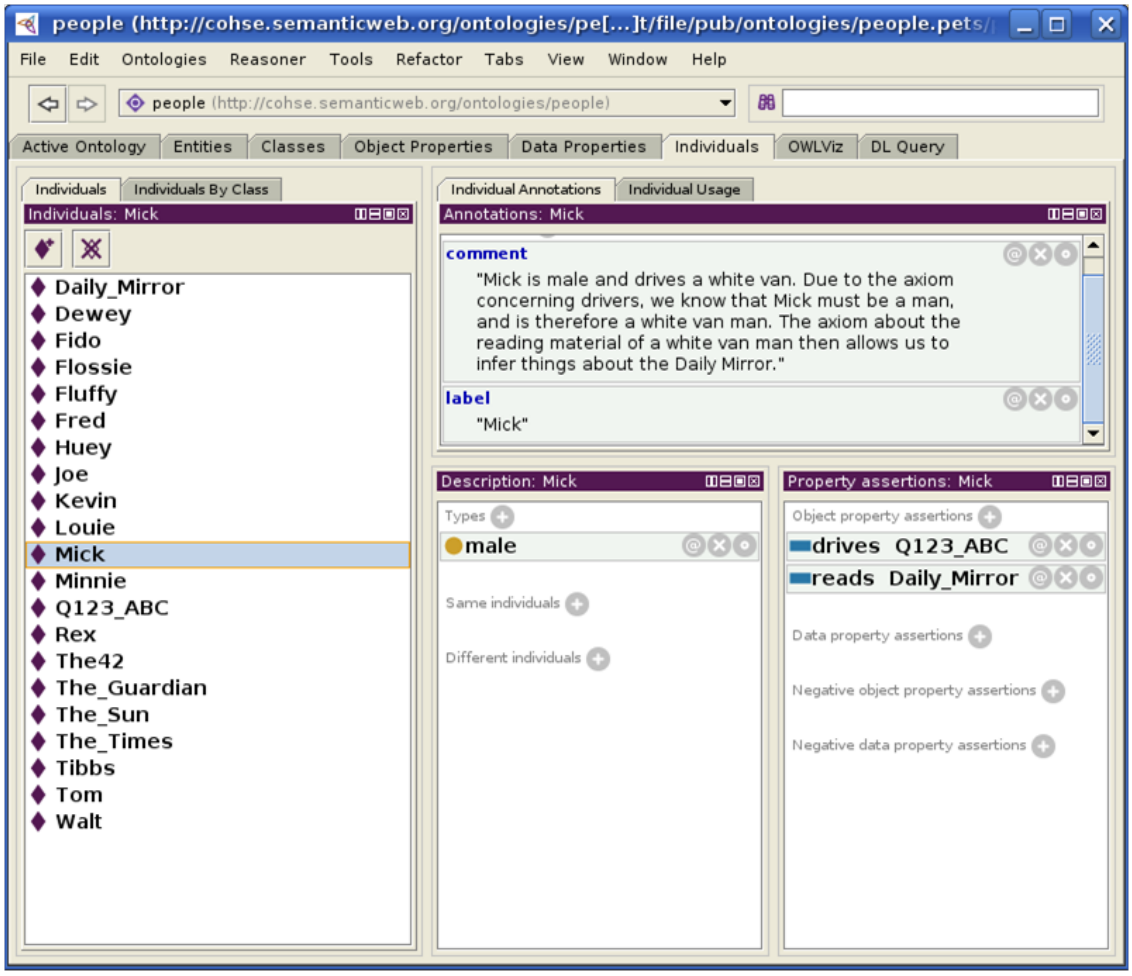
Dicho editor se encuentra en:
http://protege.stanford.edu/download/protege/4.0/installanywhere/

Una vez descargado, instalado y arrancado el editor, se procederá a cargar la ontología de ejemplo que se encuentra en http://protege.cim3.net/file/pub/ontologies/people.pets/people+pets.owl, mediante la opción 'Open OWL ontology from URI' e indicando dicha URI.

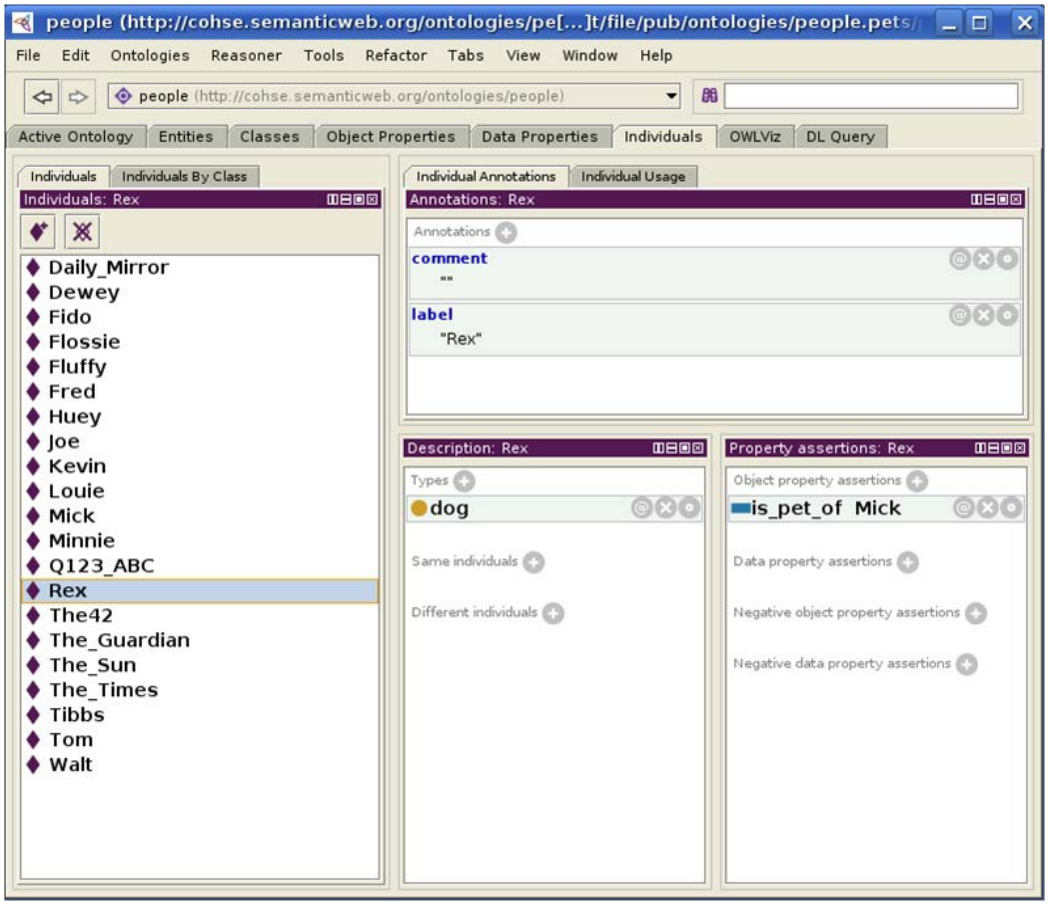
Veamos un par de pantallas que nos aparecerán. Por una parte, en la sección de definición de propiedades se tienen definida la propiedad "***is_pet_of***" como inversa de "***has_pet***"



Por otra parte, se tienen definidas 'Mick' y 'Rex' en la sección de ***Individuals***, tal y como se observa en las siguientes imágenes.



Notar que no se indica cual es la mascota de Mick. Sin embargo, cuando se define Rex sí que aparece como que es mascota de Mick (véase la siguiente imagen).


people (http://cohse.semanticweb.org/ontologies/pe[...]t/file/pub/ontologies/people.pets/
File Edit Ontologies Reasoner Tools Refactor Tabs View Window Help
people (http://cohse.semanticweb.org/ontologies/people)
Active Ontology Entities Classes Object Properties Data Properties Individuals OWLViz DL Query
Individuals Individuals By Class
Individuals: Rex
Daily_Mirror
Dewey
Fido
Flossie
Fluffy
Fred
Huey
Joe
Kevin
Louie
Mick
Minnie
Q123_ABC
Rex
The42
The_Guardian
The_Sun
The_Times
Tibbs
Tom
Walt
Individual Annotations Individual Usage
Annotations: Rex
Annotations
comment
""
label
"Rex"
Description: Rex
Types
dog
Same individuals
Different individuals
Property assertions: Rex
Object property assertions
is_pet_of Mick
Data property assertions
Negative object property assertions
Negative data property assertions