TFC - XML Y WEB SEMÁNTICA

MEMORIA

DISEÑO Y POPULACIÓN SEMIAUTOMÁTICO DE ONTOLOGÍAS

ALUMNO: SATURNINO FERNÁNDEZ VERDEJO
CONSULTOR: SINUHÉ ARROYO GÓMEZ

SEPTIEMBRE 2011

Índice de contenido

1. Introducción.

Uno de los usos principales de la Web por parte de los usuarios es el de la búsqueda de información. Como ya sabemos, el proceso para encontrar información en la Web, habitualmente consiste en introducir palabras clave, que de algún modo estén relacionadas con la información que queremos consultar, en alguno de los buscadores Web más populares. Sin embargo, no es fácil obtener de una forma precisa la información que esperamos encontrar, ya que el resultado de las búsquedas se presenta al usuario como enlaces a las páginas que quizás pueden contener la información que está buscando y éste, generalmente está ordenado de una forma determinada por el buscador que se haya utilizado, además en la mayoría de los casos, el buscador, devolverá mucha más información de la que una persona normal puede asimilar, así, no se facilita de ninguna forma que podamos encontrar las páginas que contienen la información más importante, ni tampoco se garantiza que éstas aparezcan en la lista de resultados de nuestra búsqueda, la información se pierde. En cualquier caso, independientemente de que la búsqueda tenga éxito, no nos quedará más remedio que invertir un tiempo considerable en acceder de forma secuencial a cada uno de los enlaces que se nos ha devuelto, y comprobar de forma visual, si contienen la información que estamos buscando, es decir, el resultado de las búsquedas está diseñado para que sea interpretado por personas.

Debemos tener en cuenta que el contenido de la Web actual, o tradicional, se basa en estructuras diseñadas por personas con el fin de que puedan ser entendidas por otras personas. El desarrollo de las páginas Web, hasta ahora, no ha seguido un patrón o un formato que permita clasificarlas en función de su contenido, sino más bien, uno orientado al posible usuario, al visitante de las páginas, y motivado como respuesta, en algunas ocasiones de forma bastante arbitraria, a sus posibles necesidades cuando accede a la información contenida en ellas.

El hecho de que las personas sean el objetivo directo del contenido de la Web impide el desarrollo de muchas de sus posibles aplicaciones. Debido a la complejidad del contenido que se comparte en la Web, es muy difícil mantener actualizada la información sobre algún tema concreto sin realizar algún tipo de intervención manual por parte de una persona. Aún así, es imposible procesar de forma manual el volumen de datos que se produce diariamente.

La Web contiene una gran cantidad de datos en diversos formatos, con estructura (documentos de texto), sin estructura (imágenes) y secuenciales (video, audio), y éstos están almacenados como datos Web gestionados por servidores Web o almacenados en sistemas de gestión de contenido o en sistemas de gestión bases de datos y se utilizan para generar el contenido dinámico facilitado por estos servidores Web. La información solicitada por un usuario o por una aplicación puede estar almacenada en múltiples formatos y dispersa por diferentes sitios y repositorios en la Web. Para satisfacer las peticiones de información de los usuarios habitualmente necesitamos la existencia de algún tipo de relación definida entre los distintos formatos de presentación de la misma. Así, es necesario haber realizado de antemano un análisis para establecer esta relación en las aplicaciones, o bien, realizar la misma de forma dinámica. En cualquiera de los casos lo que estamos haciendo al definir estas relaciones es definir el significado, o el sentido, de los datos sobre los que subyacen, o sea, su semántica.

Durante los últimos años, se han desarrollado distintas líneas de investigación sobre semántica que tienen como objetivo común encontrar formas de codificar el significado de la información de forma explícita y declarativa, para que ésta, por medio de sistemas automáticos creados con las herramientas y utilidades fruto de estas líneas concretas de desarrollo, pueda ser creada y utilizada de forma eficiente.

La Web Semántica es una colección de tecnologías y estándares que permiten a las aplicaciones informáticas procesar de forma eficiente la información contenida en la Web gracias a su contenido semántico, es decir, es una extensión de la Web tradicional en la que los recursos tienen un significado concreto y preciso comprensible por dichas aplicaciones. Dicho entendimiento se consigue gracias a la anotación de los

recursos mediante el uso de ontologías. Aunque la tecnología es relativamente nueva y no del todo madura, las comunidades y los sectores empresariales consideran que la tecnología de Semántica puede tener un gran impacto en el futuro desarrollo de la Web.

Una de las principales ventajas del uso de las nuevas aplicaciones desarrolladas con estas tecnologías es que muchas de las funciones que se están haciendo en la actualidad manualmente en la Web, en un futuro próximo, se podrán realizar de forma automática, de forma que, estas aplicaciones procesarán el contenido de la Web y serán capaces de encontrar y agregar la información que necesitemos de forma correcta, por un lado, respondiendo a nuestras necesidades como usuarios y por el otro, produciendo mas información con contenido semántico para su uso por otras aplicaciones.

El desarrollo de estas aplicaciones se fundamenta en el modelado de la información con en el uso de metadatos y en la estandarización, o normalización, de sus formatos de representación. Las especificaciones estándar incluyen el uso de RDF (Resource Definition Framework) y XML para la representación de metadatos , y además, que los términos utilizados en el vocabulario de estos metadatos estén recogidos en ontologías disponibles en la Web. Las especificaciones estándar para representar las ontologías incluyen, entre otras, RDF Schema y OWL (Ontology Web Language).

Los componentes[1] que intervienen en el modelado de información en la Web se pueden organizar en tres niveles funcionales.

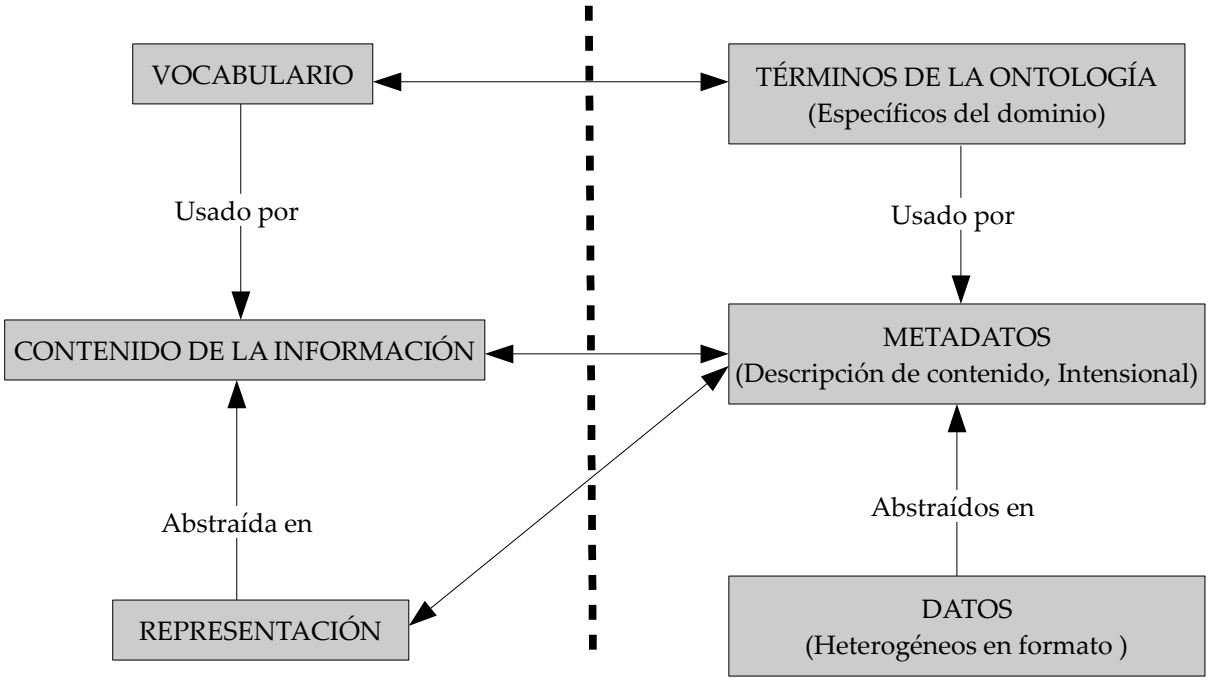


Figura 1 Elementos clave para el modelado de la información.

El nivel central, representa al componente de los metadatos, que implica el uso de los metadatos para representar la información contenida en los datos que están almacenados en los distintos repositorios y sitios Web. Las descripciones intensionales construidas a partir de los metadatos se usan para abstraerlos de su estructura y su organización, y especifican relaciones entre elementos de interés en el dominio de aplicación.

Una expresión intensional es aquella que facilita un significado o connotación. Por ejemplo, "Planetas del Sistema Solar" es la expresión intensional de la lista exhaustiva de todo aquello a lo cual aplica "Mercurio,

1 **Vipul Kashyap., et al. (2004):** *Information Modeling on the Web: The Role of Metadata, Semantics, and Ontologies* in *The Practical Handbook of Internet Computing. Munindar.* Edited by P. Singh. Chapman Hall and CRC Press, Baton Rouge. 2004.

Venus, Tierra, Marte, Júpiter, Saturno, Urano y Neptuno", siendo esta última una expresión extensional.

El nivel superior, representa al componente de la ontología, el cual engloba en dominios específicos los términos (conceptos, roles) que son usados para caracterizar las descripciones realizadas por los metadatos. En estos términos está autocontenido el conocimiento del dominio. Los términos se utilizan para describir las relaciones entre los datos contenidos en de los distintos repositorios estableciendo asociaciones entre ellos, siendo éstas, la base fundamental de la semántica.

Los metadatos, en sentido general, son datos o información sobre datos. El ejemplo mas común es el esquema de una base de datos estructurada. Los metadatos se pueden utilizar para guardar propiedades derivadas del medio en el que está almacenada la información con el fin de facilitar su acceso y su recuperación. Pueden realizar una descripción completa o un resumen de la información contenida en los datos de una forma intensional. También se pueden utilizar para representar las propiedades o relaciones entre tipos de objetos individuales y medios heterogéneos.

Como vemos en la figura 1, la función de las descripciones de los metadatos es doble:

- Permite la abstracción de los detalles figurativos de los datos, como su formato y su organización, así, capturan la información contenida en los datos subyacentes independientemente de sus detalles figurativos. Estas expresiones pueden ser usadas para representar relaciones útiles entre varias porciones de datos de repositorio o de un sitio Web.
- Permite la representación de un domino de conocimiento describiendo la información del dominio al que los datos subyacentes pertenecen. Este conocimiento puede ser usado entonces para realizar inferencias sobre los datos subyacentes para identificar relaciones entre datos almacenados en diferentes repositorios y sitios Web.

Como conclusión, gracias a la popularidad que hoy en día tiene Internet y a las facilidades que tenemos para publicar contenidos digitales heterogéneos, el volumen de información disponible, su formato y su dispersión han alcanzado unos niveles que impiden un uso eficiente de la información y dificultan la escalabilidad y el desarrollo de la Web. El modelado de la información puede cambiar este panorama. La creación y extracción de contenidos semánticos y la **integración** de diferentes ontologías es el punto clave de esta transformación.

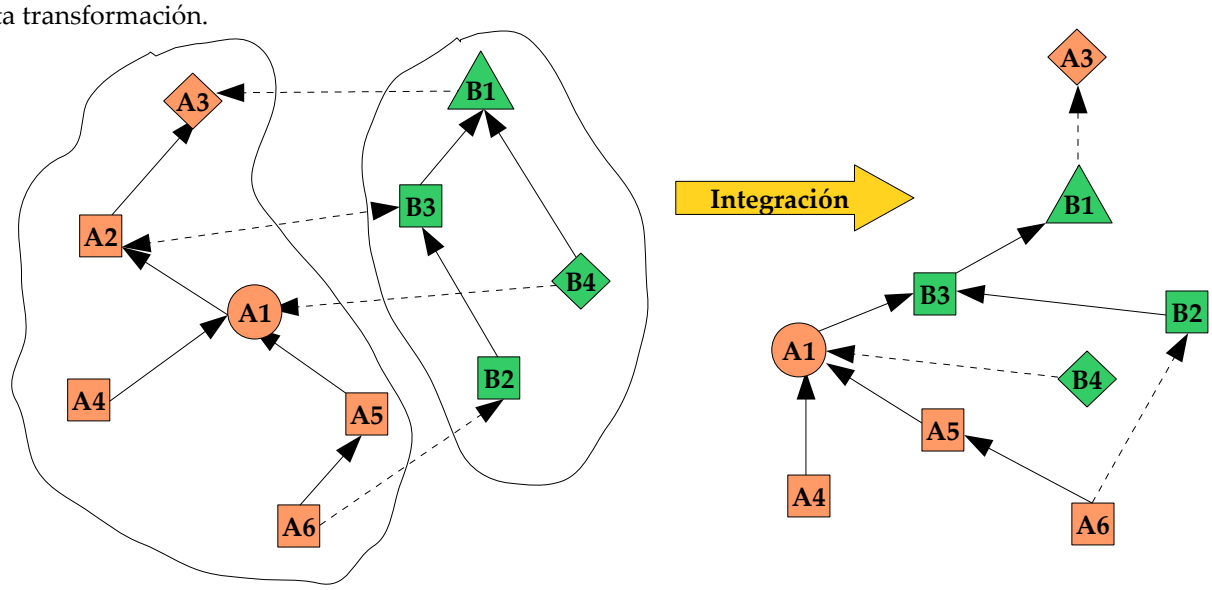


Figura 2 Expansión de la terminología a partir de la integración de ontologías[2].

2 **Vipul Kashyap., et al. (2004):** *Information Modeling on the Web: The Role of Metadata, Semantics, and Ontologies* in *The Practical Handbook of Internet Computing. Munindar.* Edited by P. Singh. Chapman Hall and CRC Press, Baton Rouge. 2004.

Uno de los principales problemas con los que la tecnología de Semántica se topa actualmente es la falta de contenidos semánticos. Esto es debido principalmente a las carencias de los métodos de anotación automáticos existentes. Un acercamiento plausible para solventar este problema se centra en esconder dichos procesos tras aplicaciones que las personas usan diariamente, tales como juegos on-line, de modo que sean los propios usuarios los que generen dichas anotaciones en lugar de los ordenadores. Otra alternativa es realizar esta tarea de forma semiautomática, por medio de programas, que a partir de la extracción de datos de la Web y una ontología determinada construyan estas anotaciones.

1.1 Objetivos del proyecto.

El objetivo de este proyecto es familiarizarse con las tecnologías de Semántica, entender que es una ontología y aprender a modelar una en un dominio elegido por nosotros. Realizar un parser que conectándose a la la Wikipedia y/o DBpedia rellene dicha ontología permitiendo al usuario navegar por sus conceptos y estudiar sus relaciones.

Dada la complejidad de la tarea de modelado, el dominio elegido será muy restringido. Además, trataremos de utilizar, tanto para el caso de la población como el del visualización, librerías open source existentes, siendo nuestra responsabilidad su correcta integración.

En resumen.

1. Conocer y familiarizarse con los conceptos básicos de la Web Semántica.
2. Conocer y familiarizarse con el concepto de ontología.
3. Modelado de una ontología en un dominio a elegir por nosotros.
4. Desarrollar un pequeño prototipo de un software que permita, conectándose a la Wikipedia y/o DBpedia rellenar con instancias la ontología creada.
5. Desarrollar un pequeño prototipo de un software que permita navegar la ontología creada junto con las instancias añadidas de una forma visual, a través de un navegador de Internet.

2. Técnicas de modelado de ontologías.

En este apartado nos familiarizaremos con el concepto de ontología, su motivación, y los contextos en los que se puede aplicar. También veremos cómo se puede modelar una ontología con RDF y RDF Schema y cómo se pueden realizar las consultas a los documentos RDF con el lenguaje SPARQL.

2.1 XML.

XML[3] (Extensible Markup Language) es un lenguaje que describe una clase de objetos llamados documentos XML y parcialmente el funcionamiento de los programas que procesan estos documentos. Es un derivado de SGML (Standard Generalized Markup Language, ISO 8879:1986) un estándar internacional para la definición de métodos de representación de información independientes de los dispositivos y sistemas. HTML es también un derivado de SGML desarrollado porque éste se consideró demasiado complejo para los fines relacionados con Internet. Posteriormente, para mejorar las carencias de HTML, se desarrolló XML.

XML[4] es un lenguaje de marcado que permite redactar contenido y ofrecer información sobre el papel que este desempeña. Este lenguaje organiza el contenido mediante etiquetas de apertura y cierre. El contenido encerrado y sus etiquetas de apertura y cierre se llama **elmento**. Por ejemplo, el título de una película puede estar definido con este elemento.

```
<titulo>Sleepy_Hollow</titulo>
```

Figura 3 Elemento XML.

Una de las características de XML es que ha sido diseñado para que pueda ser fácilmente interpretado por personas. Los programas de aplicación también pueden interpretarlo con facilidad ya que éste describe cada fragmento de información, y además, también define alguna de las formas que toman sus relaciones a través de la estructura anidada.

```
<Pelicula>
    <titulo>Sleepy_Hollow</titulo>
    <director>Tim_Burton</director>
</Pelicula>
```

Figura 4 Estructura anidada XML.

Como vemos en la figura anterior, ahora <titulo> aparece dentro de <Pelicula> describiendo que es una propiedad de la película. Un programa que procesara el documento XML de la figura 4 puede obtener como resultado que el elemento <titulo> hace referencia a al elemento <Pelicula> que lo contiene.

XML separa el contenido del tipo de formato, así la información que contiene un documento XML se puede visualizar de distintas formas sin tener que elaborar múltiples copias del mismo contenido, y además, éste puede ser utilizado para propósitos diferentes al de la presentación.

La información la podemos usar de varias formas, y es el usuario el que define el vocabulario apropiado para sus programas de aplicación, por eso XML es un metalenguaje para el marcado, no tiene un conjunto fijo de etiquetas sino que permite a los usuarios definir etiquetas propias. De esta forma, otro usuario puede

3 http://www.w3.org/TR/2008/REC-xml-20081126/#sec-xml-and-sgml

4 **Antoniou G., van Harmelen F.** (2004). *A semantic web primer.* The MIT Press.

representar el documento XML de la figura 4 tal y como aparece en la siguiente figura.

```
<Filme>
        <director>Tim_Burton</director>
        <nombre>Sleepy_Hollow</nombre>
</Filme>
```

Figura 5 Estructura anidada XML equivalente a la anterior.

Para que las aplicaciones que se utilizan en la Web puedan comunicarse entre si y colaborar entre ellas es necesario que utilicen un vocabulario común. Han surgido muchas iniciativas en este sentido, y ya existen muchos vocabularios especializados en dominios concretos. Muchas empresas de sectores muy diversos, (aeronáutica, prensa, finanzas, biotecnología, telecomunicaciones, automoción, hardware), están usando aplicaciones XML. El XML, en la actualidad se está utilizando como un formato uniforme de intercambio de datos entre aplicaciones.

Una de las principales debilidades de XML es que carece de medios para hablar de la semántica (el significado) de los datos. La anidación de etiquetas no tiene ningún significado predefinido y es responsabilidad de cada aplicación interpretar la anidación de las mismas. Volviendo al ejemplo que hemos visto en la figura 4.

```
<Pelicula titulo= “Sleepy_Hollow “>
        <director>Tim_Burton</director>
</Pelicula>

<Director nombre=”Tim_Burton”>
        <titulo>Sleepy_Hollow</titulo>
</Director>
```

Figura 6 Anidaciones XML opuestas.

Los dos documentos XML ofrecen anidaciones opuestas, aunque representan la misma información. Por lo tanto no existe una forma estándar para asignar significado a la anidación de etiquetas.

2.2 RDF.

Tal y como hemos visto en apartados anteriores, el contenido de la Web ha sido diseñado y construido para que pueda ser usado, leer y entender su contenido, por personas, y aunque está en un formato que algunas aplicaciones informáticas reconocen (navegadores), éste, dista mucho de poder ser interpretado, es decir de poder procesar de forma eficiente su contenido, por dichas aplicaciones. Por este motivo, mecanizar el tratamiento de información en la Web es muy complicado, y hacer esta tarea de forma manual es prácticamente imposible, debido al volumen de información que la Web contiene en la actualidad.

Para abordar este problema el W3C[5] realizó una propuesta basada en usar metadatos para describir los datos que están contenidos en la Web. Como los metadatos pueden ser procesados de una forma eficiente por una aplicación informática, tal y como hemos visto, entonces con ellos podemos procesar automáticamente el

5 Resource Description Framework (RDF) model and syntax specification, a W3CRecommendation, 22 February 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

contenido de los recursos Web que describen. En este contexto, el RDF (Resource Description Framework), que es un modelo de datos, es la base para crear y procesar estos metadatos.

Las principales características de RDF son:

- Define un mecanismo para describir recursos independientemente del domino de aplicación al que pertenecen, es decir, es independiente del domino, y por este motivo, se puede utilizar para describir información de cualquier dominio.
- Facilitar el intercambio de datos entre aplicaciones que pueden entender su contenido.
- Facilitar la conexión o el enlace de la información disponible sobre un recurso distribuida sobre la Web.
- Nos permite representar la información de una forma estructurada.
- Su desarrollo está basado en XML.

Un recurso en este contexto es "cualquier cosa sobre la que cualquier persona quiere decir algo". Por ejemplo, una cámara de fotos digital, un escritor, una obra de teatro, un televisor LED, un animal prehistórico, una película, un político, etc.

El hecho de que cualquier persona pueda aportar algún tipo de conocimiento o de información sobre cualquier cosa en la Web, implica, que la información sobre los recursos se encuentre ampliamente distribuida sobre la propia Web, y que el modelo de datos que usemos para describir estos recursos deba contemplar esta característica de la información.

2.3 RDF modelo abstracto y conceptual.

Con RDF podemos representar la información de una forma estructurada dividiéndola en piezas elementales de información. Estas piezas elementales de información se llaman "*sentencias*" y se componen de tres elementos ordenados (triples o ternas) que se denominan "*sujeto*", "*predicado*" y "*objeto*". Una sentencia RDF se puede representar como un grafo dirigido.

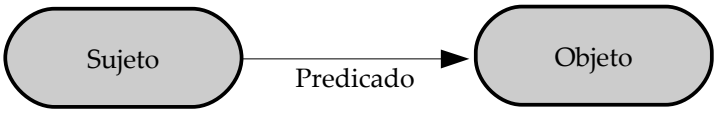


Figura 7: Sentencia RDF representada como grafo dirigido.

Donde el sujeto y el objeto son nombres simples de "alguna cosa concreta o abstracta" y el predicado es el nombre de la relación que conecta las dos "cosas". Por ejemplo, en el dominio del cine, y con respecto a la película Sleepy Hollow podríamos escribir estas sentencias, cada una de las cuales es una pieza de información, o un hecho, de la misma:

Sujeto	**Predicado**	**Objeto**
Sleepy Hollow	Es	Película
Sleepy Hollow	Director	Tim Burton
Sleepy Hollow	Protagonista	Johnny Depp
Sleepy Hollow	Protagonista	Christina Ricci

Figura 8: Sentencias RDF representadas de forma tabular.

Tomadas en conjunto, las cuatro sentencias nos facilitan información sobre Sleepy Hollow, y ésta tiene un significado que podemos entender, "*Sleepy Hollow es una película dirigida por Tim Burton y protagonizada por Johnny Depp y Christina Ricci*". Las cuatro sentencias se pueden representar en forma de grafo dirigido, y en este caso, hablamos de un grafo RDF.

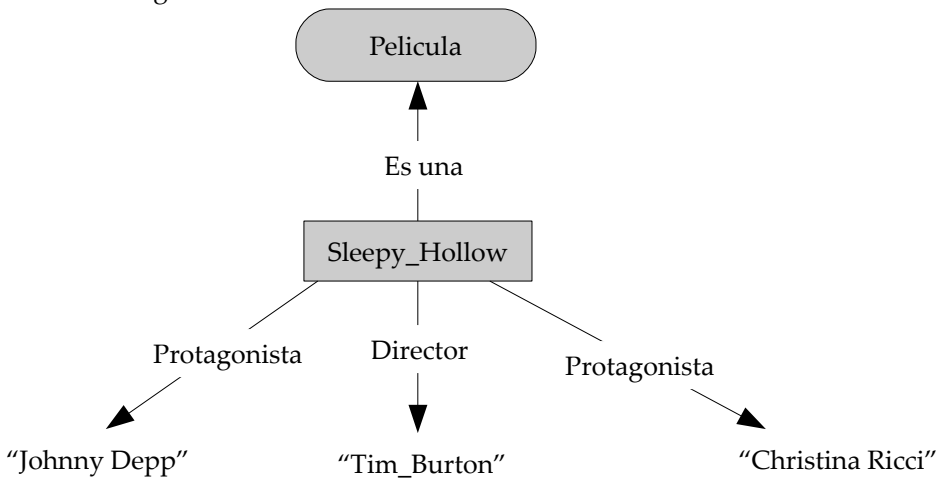


Figura 9 Grafo RDF.

Es importante resaltar que este modelo nos permite representar cualquier conocimiento sobre cosas concretas o abstractas, y además se puede ampliar fácilmente añadiendo mas sentencias al grafo. En el contexto en el que estamos, técnicas de Web semántica, el sujeto y/o el objeto de una sentencia, independientemente de que sean cosas concretas o abstractas se denomina "*recurso*", y así decimos, que una sentencia RDF describe un recurso.

Los nombres de los recursos no deben ser elegidos de forma arbitraria. Deben estar identificados por un Uniform Resource Identifier (URI). El motivo es que de esta forma evitamos que sean ambiguos. Por ejemplo, los siguientes conjuntos de caracteres, "Tim Burton", "Burton, Tim", "Tim_Burton", "tim burton", "Timothy William Burton", ¿se refieren a una persona?, ¿si es una persona, son términos que se refieren a la misma persona?, ¿si queremos hacer referencia a un científico que se llame así, cómo lo vamos a poder diferenciar del director de cine para saber de quien estamos hablando?. Si en lugar de usar un conjunto arbitrario de caracteres utilizamos un URI como http://dbpedia.org/resource/Tim_Burton que es global y único evitamos la ambigüedad en el nombre de los recursos. Todas las sentencias que utilizan este recurso se refieren a la misma persona, sin ninguna duda.

En la medida de lo posible, y para evitar ambigüedades, debemos reutilizar URI existentes para referirnos a un recurso, en vez de usar uno propio.

Los nombres de recurso pueden ser muy largos, dependiendo de lo complejo que sea el URI que lo forma. Para simplificarlos, aprovechando que la sintaxis de RDF se basa en XML, podemos reemplazarlos por su nombre abreviado en forma de "XML qualified name" o QName[6]. Este, se forma con un "prefijo" que corresponde al espacio de nombres al que pertenece el URI, seguido por ":" y por el "nombre local".

prefijo : nombre

Así, si definimos un prefijo que se llame dbpedia y le hacemos corresponder el espacio de nombres http://debpedia.org/resource/ entonces podemos simplificar el nombre de recurso correspondiente a "Tim Burton" como:

dbpedia : Tim_Burton

Esta abreviatura es equivalente a http://dbpedia.org/resource/Tim_Burton

6 http://www.w3.org/TR/2006/REC-xml-names-20060816/

Los nombres de los predicados y los objetos deben seguir las mismas reglas que los nombre de los recursos, y por tanto una nomenclatura similar. Se debe utilizar un URI en lugar de un nombre arbitrario. Un objeto puede ser un recurso, un literal (un conjunto de caracteres) y también puede ser anónimo[7].

De forma alternativa el grafo de la Figura 7 se puede representar de esta forma.

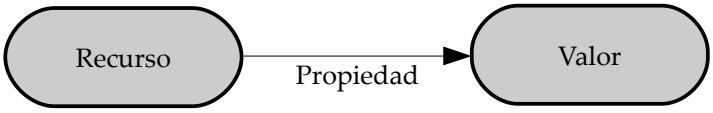


Figura 10: Sentencia RDF representada como grafo dirigido.

Y la tabla de la Figura 8 de esta otra forma:

Recurso	**Propiedad**	**Valor**
dbpedia : Sleepy_Hollow	prop : Es	"Película"
dbpedia : Sleepy Hollow	prop : Director	dbpedia : Tim Burton
dbpedia : Sleepy Hollow	prop : Protagonista	dbpedia : Johnny Depp
dbpedia : Sleepy Hollow	prop : Protagonista	dbpedia : Christina Ricci

Figura 11: Sentencias RDF representadas de forma tabular.

Un literal puede ir acompañado de una etiqueta que indique el idioma en el que está escrito, por ejemplo "Película"@es indica que el literal está en idioma Español. También puede ir acompañado de un URI que indica el tipo de dato contenido en el literal y este puede ser interpretado por un parser de RDF. Los literales se pueden expresar de esta forma:

"Película"

"Película"@es

"Película"^^<http://www.w3.org/2001/XMLSchema#string>

Figura 12: Etiquetas de lenguaje y tipo datos en literales RDF

2.4 RDF /XML.

El formato RDF/XML permite crear documentos RDF. El formato es adecuado para el procesamiento mecanizado y eficiente de su contenido mediante una aplicación informática, pero no es muy legible para las personas. Este formato es una especificación del W3C[8] y permite representar un grafo RDF como un documento XML y que denominamos, documento RDF.

Básicamente, sin entrar en profundidad[9], un documento RDF se compone de un elemento **rdf:RDF** que tiene como contenido un conjunto de descripciones. Continuando en el dominio del cine y el ejemplo anterior,

7 Ver concepto nodo anónimo en http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/#higherorder

8 http://www.w3.org/TR/rdf-syntax-grammar/

9 Podemos profundizar en el modelo en la bibliografía recomendada. **Antoniou G., van Harmelen F.** (2004). *A semantic web primer.* The MIT Press.

describimos los elementos mas comunes.

```
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" ( 1)
        xmlns:j.0="http://www.sfernandezve.org/cine#" > (2)
        <rdf:Description rdf:about="http://dbpedia.org/resource/Sleepy_Hollow"> (3)
                <j.0:Director rdf:resource="http://dbpedia.org/resource/Tim_Burton"/> (4)
                <j.0:Protagonista rdf:resource="http://dbpedia.org/resource/Christina_Ricci"/> (4)
                <j.0:Protagonista rdf:resource="http://dbpedia.org/resource/Johnny_Depp"/> (4)
                <rdf:type rdf:resource="http://www.sfernandezve.org/cine#Pelicula"/> (5)
                <j.0:Titulo>Sleepy Hollow</j.0:Titulo> (6)
        </rdf:Description>
</rdf:RDF>
```

Figura 13 Ejemplo de documento RDF para describir un recurso.

(1) y (2) Como hemos indicado RDF usa URI en vez de palabras para nombrar recursos y propiedades. Estas dos URIs permiten abreviar el vocabulario que empleamos para hacer referencia a las propiedades particulares de RDF y a las propiedades que nos hemos inventado para definir las relaciones entre la película Sleepy Hollow, su director y sus protagonistas.

(3) El elemento **rdf:Description** indica que comienza la descripción de un recurso y utiliza el atributo **rdf:about** para especificar el recurso que se está describiendo. También podemos utilizar **rdf:ID** en lugar de rdf:about como veremos mas adelante cuando describamos el dominio específico de aplicación por medio de su ontología.

(4) Son elementos de propiedad, representan parejas propiedad + valor del recurso descrito. El director y los protagonistas de la película. Tanto la propiedad como el valor son URI, la primera abreviada y la segunda sin abreviar. El valor de la propiedad es otro recurso y se define con **rdf:resource**.

(12) El elemento **rdf:type** es uno de los mas importantes. Indica de que tipo es el recurso que acabamos de describir.

(6) Es un elemento de propiedad, también representa una pareja propiedad + valor del recurso descrito, pero en este caso la propiedad es un URI y el valor un literal, el título de la película.

¿Que ventajas aporta RDF sobre XML?. Aunque XML es probablemente el mejor formato para compartir información y para realizar intercambio de datos entre aplicaciones en diferentes plataformas no tiene suficiente capacidad para expresar significado semántico.

2.5 RDF Schema.

RDF Schema es una recomendación del W3C[10]. Basándonos en la publicación indicada;

"RDF Schema es un lenguaje de representación de conocimiento extensible que podemos utilizar para crear un vocabulario para describir clases, subclases, y propiedades de los recursos RDF".

10 http://www.w3.org/TR/rdf-schema/

Tal y como hemos visto en el apartado anterior, los elementos propiedad de RDF se pueden considerar como atributos de los recursos descritos y se corresponden a parejas propiedad + valor. Y además, que estas propiedades también pueden representar relaciones entre recursos de un dominio elegido por el usuario, por ejemplo, el dominio del Cine. Sin embargo, ¿cómo se describen los propios elementos propiedad (Director, Protagonista,...)? ¿dónde se definen y a qué recursos se puede aplicar una propiedad?, ¿cómo se describe un dominio en particular (Cine)?. Esta, es la finalidad del lenguaje de descripción de vocabulario RDF, el RDF Schema.

RDF Schema es una extensión semántica de RDF. Facilita los mecanismos necesarios para describir grupos de recursos relacionados, así como las relaciones existentes entre ellos, es decir, facilita una colección de términos que podemos utilizar para definir las clases y las propiedades de un dominio de aplicación específico. Estos términos se identifican con un URI predefinido:

```
http://www.w3.org/2000/01/rdf-schema#
```

Y este se asocia habitualmente al prefijo de espacio de nombres rdfs :

Los términos RDF Schema mas comunes se pueden agrupar de esta forma, dependiendo de su función:

- **Clases.** Son los que utilizamos para definir clases. Entre otros tenemos, rdfs : Resource, rdfs : Class, rdfs : Literal, rdfs : Datatype.
- **Propiedades.** Son los que utilizamos para definir propiedades. Entre otros tenemos, rdfs : range, rdfs : domain, rdfs : subClassOf, rdfs : subPropertyOf, rdfs : label y rdfs : comment.
- **Utilidades.** Nos permiten definir enlaces con otras URL que contiene definiciones relacionadas con el recurso que se describe. Por ejemplo rdfs:seeAlso y rdfs:isDefinedBy.

RDF Schema es un lenguaje de ontologías primitivo que permite definir la semántica de dominios muy concretos[11].

2.6 Ontología.

En el contexto de técnicas de Web semántica, una ontología comprende estos aspectos.

- Una ontología aplica a un dominio específico, y se utiliza para describir y representar un área concreta de conocimiento, como puede ser la Medicina, la Fotografía, la Enología, la Pintura o el Cine.
- Una ontología contiene conceptos o clases y relaciones entre ellas. Estas relaciones se expresan en forma de jerarquías, en las cuales, los conceptos mas generales se encuentran en la zonas mas altas (superclases) y los conceptos mas específicos en las zonas mas bajas (subclases).
- Una ontología contiene, además de las relaciones indicadas en el párrafo anterior, otro nivel de relaciones expresadas usando un grupo especial de términos, las propiedades. Éstas describen características y atributos de los conceptos y pueden ser usadas para agrupar diferentes clases, expresar disyunción entre ellas, y aplicar restricciones de valor. Así las relaciones entre clases pueden ser de dos tipos, relación superclase – subclase y relación expresada en términos de propiedad.

Estos aspectos son los que contienen el conocimiento sobre un dominio de tal forma que éste, el conocimiento, pueda se procesado de forma eficiente por una aplicación informática.

11 **Antoniou G., van Harmelen F.** (2004). *A semantic web primer.* The MIT Press.

Una ontología nos aporta las siguientes ventajas.

- Nos facilita la comprensión de las ideas clave de un dominio concreto de aplicación.
- Nos aporta los términos que podemos emplear cuando creamos documentos RDF sobre un dominio.
- Nos permite reutilizar el conocimiento existente sobre el dominio.
- Codificada usando un lenguaje de descripción de ontologías, como RDF Schema tal y como lo vamos a realizar en el siguiente apartado, nos permite que desde un programa de aplicación podamos procesar este conocimiento y su semántica de una forma eficiente.

El modelado de ontologías es una disciplina que, dependiendo del dominio al que aplique, puede llegar a ser muy compleja. Para hablar con propiedad deberíamos referirnos a esta actividad como Ingeniería Ontológica y la podemos definir como el conjunto de actividades que se refieren al proceso de desarrollo de una ontología, su ciclo de vida, así como las metodologías, herramientas y lenguajes necesarios para la construcción de ontologías.

Es importante hacer notar que podemos construir sentencias RDF sin apoyarnos en ninguna ontología. Pero debemos tener en cuenta que los documentos RDF que obtenemos contienen descripciones de recursos que carecen de un contenido semántico definido, y por lo tanto, éste tampoco podrá ser usado para inferir ningún conocimiento. Como conclusión, cuando se usan sentencias RDF, estas se deben construir usando como base una ontología que sea consistente, de uso general, formalmente correcta y extensible.

Hay dos lenguajes de ontologías, RDF Schema, que es primitivo (limitado) y OWL que es mas expresivo que el anterior.

Para desarrollar una ontología, uno de los métodos mas populares es el creado por Noy y MCGuinness[12] (Natalya F. Noy and Deborah L. McGuinness, Stanford University). Siguiendo sus recomendaciones, los pasos que debemos seguir son:

1. Determinar el dominio y el alcance de la ontología.

   Para realizar esta tarea, debemos responder a estas preguntas:

   - ¿Cual es el dominio con el que está relacionado la ontología?.
   - ¿Para que propósito se crea la ontología?.
   - ¿Qué preguntas debe responder la ontología?
   - ¿Quien va a usar y mantener la ontología?.

2. Intentar reutilizar ontologías existentes.

   Reutilizar una ontología existente es una buena elección, pero solo cuando esta es adecuada para nuestro propósito. Sin embargo este es un punto muy importante a considerar, sobre todo si nuestra aplicación tiene que interaccionar con otras que ya están usando una ontología.

3. Enumerar los términos importantes en la ontología.

   Este paso es un requisito necesario antes de definir las clases y la jerarquía de clases. Los autores sugieren que se construya una lista de términos del dominio sin preocuparse del solapamiento de los conceptos que estos representan, de las relaciones entre los términos o de cualquier propiedad que los conceptos puedan tener. El objetivo es asegurarse de que todos los términos importantes están incluidos y así facilitar la definición de las clases y de la jerarquía de clases.

12 http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html

4. Definir las clases y la jerarquía de clases.

   Se proponen tres enfoques, a saber.

   - Top-down, consiste en definir las clases mas generales y a partir de ellas las mas especializadas.
   - Botton-up., consiste en hacer lo contrario al anterior. Empezar por las clases mas especializadas hasta llegar a las mas generales.
   - Combinar los otros dos métodos, cambiando de uno al otro a medida que se definen las clases y su jerarquía.

   Hay que tener en cuenta que una jerarquía de clases es una relación "is-a", en la que la relación entre subclases es siempre transitiva.

5. Definir las propiedades de las clases.

   En este punto los autores sugieren que se consideren los siguientes tipos de propiedades.

   - Intrínsecas, son aquellas inherentes o esenciales de una clase.
   - Extrínsecas, son aquellas que son externas o no esenciales de una clase.
   - Partes, elementos que son partes del objeto que representa la clase.
   - Relaciones con otros individuos, con otras clases.

6. Añadir restricciones a las propiedades.

   Por ejemplo de tipo de valor que puede admitir, su cardinalidad, su dominio, su rango, etc... RDF Schema no admite tantas restricciones como OWL, ya que es mucho mas limitado y primitivo.

7. Crear las instancias.

   Este paso es el último que realizamos, y puede ser opcional, ya que el documento que contiene la definición ontología no necesita tener incluidas instancias.

2.7 Definición de una ontología. Cine.

Lo primero que debemos tener en cuenta es de que cosas queremos hablar, y así determinar el domino y el alcance da la ontología. Naturalmente, vamos a seguir hablando de Cine, pero con un dominio muy restringido. Como ejemplo vamos a representar películas, directores, actores y productores y las relaciones que existen entre ellos. El dominio no va a incluir ni críticas ni calificaciones.

La ontología se crea con el fin de hacer una prueba de concepto de la tecnología. Y debe contestar a estas preguntas.:

- ¿Quienes son los directores, protagonistas y productores de una película?.
- ¿Cuáles son las películas que ha dirigido , producido y/o protagonizado un director de cine?.
- ¿Cuáles son las películas que ha dirigido, producido y/o protagonizado un actor de cine?.
- ¿Cuáles son las películas que ha dirigido, producido y/o protagonizado un productor de cine?.

No reutilizaremos ontologías existentes por la naturaleza del proyecto, pero no debemos olvidar la importancia de este punto si queremos interaccionar con otras aplicaciones que estén usando ya una ontología.

Los términos importantes en la ontología son:

Película, Director, Actor, Protagonista, Productor. Una película es una obra y tiene un título, una sinopsis, Directores, Productores, Protagonistas. Un Director es una Persona que tiene un nombre y que ha dirigido, protagonizado y producido películas. Un Productor es una Persona que tiene un nombre y que ha producido, protagonizado y dirigido películas. Un Actor es una Persona que tiene un nombre y que ha protagonizado, producido y dirigido películas.

A partir de los términos importantes de la ontología determinamos los elementos de este dominio. Podemos plantear una estructura jerárquica de clases como la que se indica a continuación.

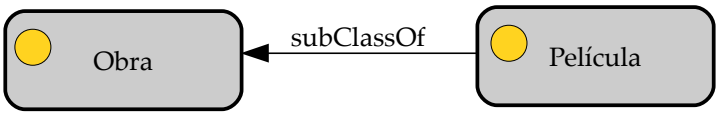


Figura 14: Definición de jerarquía de clases en el dominio Cine.

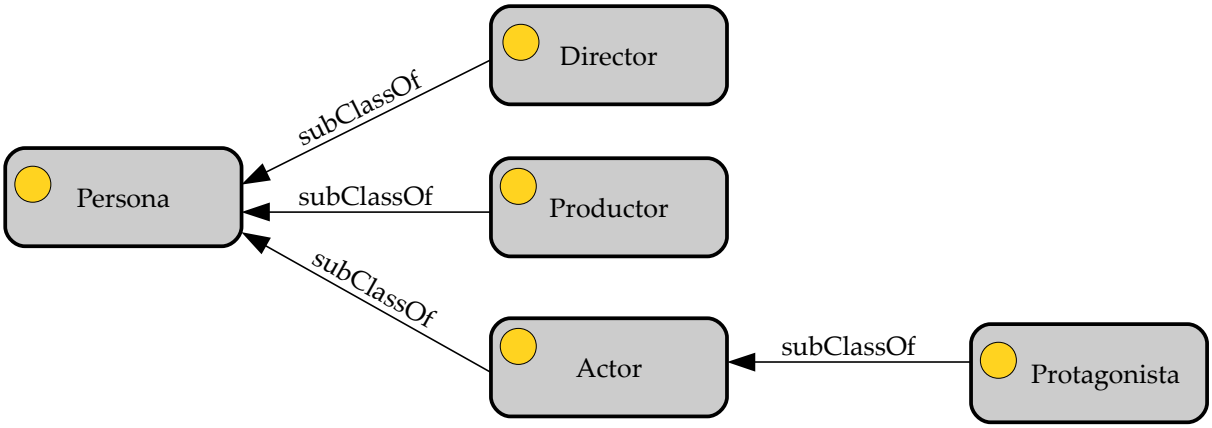


Figura 15: Definición de jerarquía de clases en el dominio Cine.

Ahora, debemos definir las propiedades de las clases y de los datos. Dado lo restringido y simple que es el dominio, aprovechamos también para definir las restricciones de las propiedades. Las obras tendrán un título y una sinopsis y las personas un nombre. Todas las propiedades son de tipo carácter (string).

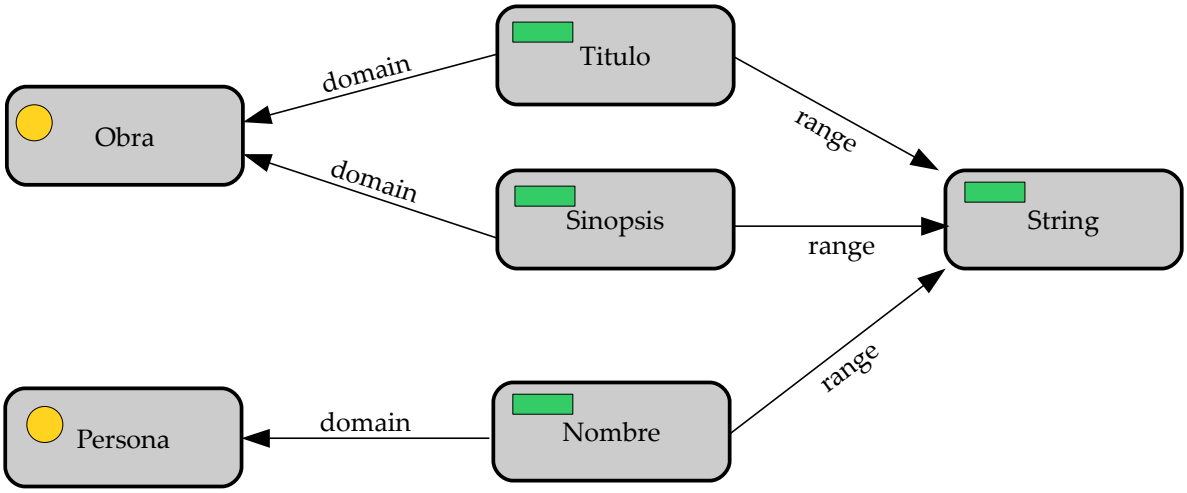


Figura 16: Definición de propiedades de clase y de datos en el dominio Cine.

Finalmente, definimos las propiedades entre los objetos, en función de qué es lo que queremos averiguar de ellos. Así:

- De una Película, queremos conocer su título, su sinopsis, sus directores, protagonistas y productores.
- De un Director, las películas que ha dirigido y además las que ha producido o protagonizado.
- De un Actor, las películas que ha protagonizado y además las que ha producido o dirigido.
- De un Productor, las películas que ha producido y además las que ha protagonizado o dirigido.

También especificamos las restricciones de dominio y de rango de estas propiedades.

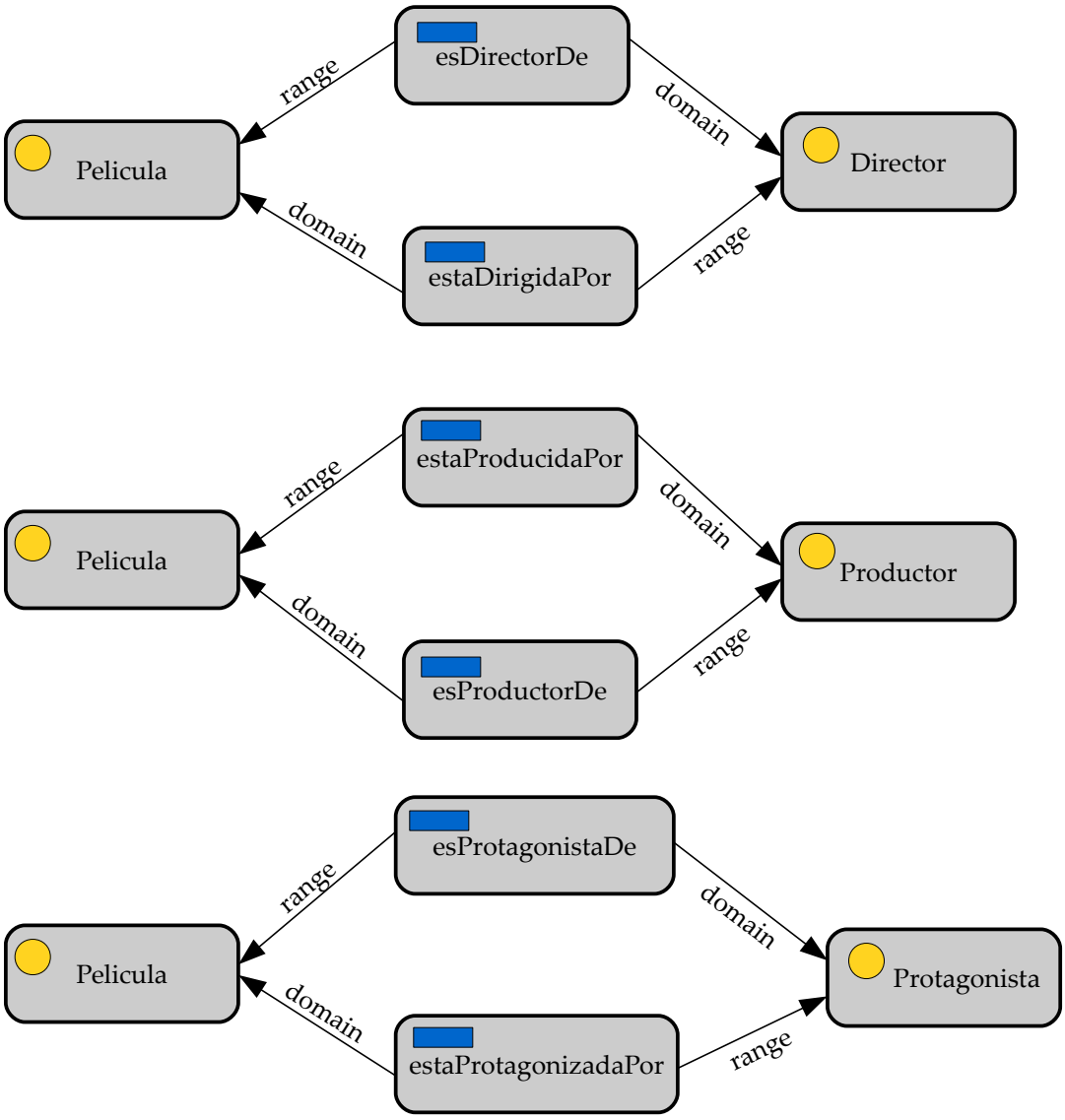


Figura 17: Definición de propiedades de objetos en el dominio Cine.

Pasamos ahora a describir la ontología con los términos RDF Schema.

```
<?xml version="1.0"?>
<rdf:RDF
        xml:base="http://www.sfernandezve.org/cine"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
        <!--
        // Clases
        // Al usar rdf:ID para describirlas el URI de la clase se forma
        // con URI base [13]+ "#" + rdf:ID valor, como base="http://www.sfernandezve.org/cine"
        // el recurso que describimos es "http://www.sfernandezve.org/cine#Pelicula"
        -->
        <rdfs:Class rdf:ID="Obra">
                </rdfs:Class>
        <rdfs:Class rdf:ID="Pelicula">
                        <rdfs:subClassOf rdf:resource="#Obra"/>
                </rdfs:Class>
        <rdfs:Class rdf:ID="Persona">
                </rdfs:Class>
        <rdfs:Class rdf:ID="Actor">
                        <rdfs:subClassOf rdf:resource="#Persona"/>
                </rdfs:Class>
        <rdfs:Class rdf:ID="Director">
                         <rdfs:subClassOf rdf:resource="#Persona"/>
                </rdfs:Class>
        <rdfs:Class rdf:ID="Productor">
                        <rdfs:subClassOf rdf:resource="#Persona"/>
                </rdfs:Class>
        <rdfs:Class rdf:ID="Protagonista">
                        <rdfs:subClassOf rdf:resource="#Actor"/>
                </rdfs:Class>
        <!--
        // Propiedades de los datos.
        -->
        <rdf:Property rdf:ID="Titulo">
                        <rdfs:domain rdf:resource="#Obra"/>
                        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                </rdf:Property>
```

13 http://www.w3.org/TR/2009/REC-xmlbase-20090128/

```
        <rdf:Property rdf:ID="Sinopsis">
                        <rdfs:domain rdf:resource="#Obra"/>
                        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                </rdf:Property>
        <rdf:Property rdf:ID="Nombre">
                        <rdfs:domain rdf:resource="#Persona"/>
                        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                </rdf:Property>
        <!--
        // Propiedades de los objetos.
        -->
        <rdf:Property rdf:ID="esDirectorDe">
                        <rdfs:domain rdf:resource="#Director"/>
                        <rdfs:range rdf:resource="#Pelicula"/>
                </rdf:Property>
        <rdf:Property rdf:ID="estaDirigidaPor">
                        <rdfs:domain rdf:resource="#Pelicula"/>
                        <rdfs:range rdf:resource="#Director"/>
                </rdf:Property>
        <rdf:Property rdf:ID="esProductorDe">
                        <rdfs:domain rdf:resource="#Productor"/>
                        <rdfs:range rdf:resource="#Pelicula"/>
                </rdf:Property>
        <rdf:Property rdf:ID="estaProducidaPor">
                        <rdfs:domain rdf:resource="#Pelicula"/>
                        <rdfs:range rdf:resource="#Productor"/>
                </rdf:Property>
        <rdf:Property rdf:ID="esProtagonistaDe">
                        <rdfs:domain rdf:resource="#Protagonista"/>
                        <rdfs:range rdf:resource="#Pelicula"/>
                </rdf:Property>
        <rdf:Property rdf:ID="estaProtagonizadaPor">
                        <rdfs:domain rdf:resource="#Pelicula"/>
                        <rdfs:range rdf:resource="#Protagonista"/>
                </rdf:Property>
</rdf:RDF>
```

Figura 18: Definición ontología Cine en RDF Schema.

La clases[14] definen conjuntos de elementos, y los objetos individuales que construiremos usando estas clases serán instancias de una clase. Las instancias las crearemos usando RDF y en cada una de ellas deberemos especificar con **rdf:type** la clase a la que pertenece la instancia que hemos creado. Así es como se hace el ***tipado*** de una instancia como veremos mas adelante. En los lenguajes de programación el ***tipado*** se usa para impedir que se redacten cosas sin sentido, y en RDF también es necesario. Así en nuestro ejemplo con las declaraciones de tipo evitaremos decir:

Sleepy Hollow	estaDirigidaPor	Sleepy Hollow
Stanley Kubrick	esDirectorde	Kirk Douglas
Steven Spielberg	estaDirigidaPor	Tim Burton

Figura 19: Restricciones de dominio y rango.

El primer caso no tiene sentido, porque las películas sólo pueden estar dirigidas por Directores. Hemos impuesto una restricción en la propiedad estaDirigidaPor al indicar que el rango de la propiedad son instancias de la clase Director.

En el segundo caso, aunque para una persona tiene un significado correcto, un Director dirige a un Actor, tampoco tiene sentido en nuestro dominio, porque hemos impuesto también otra restricción en la propiedad esDirectorDe al indicar que el rango de la propiedad son instancias de la clase Película.

El tercer caso tampoco tiene sentido, porque un Director dirige películas. Hemos impuesto una restricción en la propiedad estaDirigidaPor al indicar que el dominio de la propiedad son instancias de la clase Película.

Las clases que hemos declarado en el esquema, restringen lo que podemos utilizar, de que podemos hablar, en un documento RDF que utilice el esquema que acabamos de definir.

Con las propiedades de los datos ocurre lo mismo. Hemos impuesto restricciones de dominio y de rango de forma que una Película no puede tener la propiedad Nombre, ni Nombre puede tener un valor “integer”.

Cuando especificamos el dominio de una propiedad, utilizando RDF Schema, estamos indicando de forma indirecta que cualquier recurso que tenga esa propiedad es una instancia de las clases del dominio.

Cuando especificamos el rango de una propiedad, utilizando RDF Schema, estamos indicando de forma indirecta que los valores de una propiedad son instancias de la clase del rango.

Por ejemplo:

```
<rdf:Property rdf:ID="Titulo">
                    <rdfs:domain rdf:resource="#Obra"/>
                    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
                </rdf:Property>
```

Figura 20: Definición de propiedad, dominio y rango.

Indica que un recurso que tenga la propiedad Titulo, **por inferencia**, será una instancia de la clase Obra o de una de sus subclases (Película) y su valor será un string de caracteres. Una propiedad puede tener varios dominios o varios rangos definidos. En caso de tener varios dominios definidos, indica que la propiedad sólo puede ser utilizada por instancias que sean, **al mismo tiempo**, de las clases definidas en esos dominios, es decir si indicamos dominio A y dominio B, la propiedad sólo puede ser usada por alguna instancia que sea de la clase A y de la clase B al mismo tiempo. Análogamente, en el caso de especificar varios rangos, indica que la propiedad puede tomar un valor que sea instancia de las clases definidas en el rango, pero tiene que ser instancia de todas las clases especificadas **al mismo tiempo**.

14 **Antoniou G., van Harmelen F.** (2004). *A semantic web primer.* The MIT Press.

El uso de clases, herencia y propiedades es muy amplio en áreas de informática relacionadas con la programación orientada a objetos. Sin embargo, existen diferencias importantes entre ésta y RDF Schema. En RDF Schema, las propiedades de las clases se definen de forma global y no están encapsuladas como atributos en las definiciones de las clases. Es posible añadir nuevas propiedades a las clases existentes sin necesidad de modificar las clases, mientras que en los lenguajes de programación orientados a objetos, esto no es posible.

La jerarquía de clases juega un papel que es muy importante. No debemos olvidar que estamos aplicando restricciones de dominio y de rango sobre clases, y RDF Schema hace que estas restricciones sean heredadas por las clases hijas, porque todas las instancias de una clase son también instancias de su superclase, y además hay que tener en cuenta que con RDF Schema una clase también puede ser subclase de mas de una clase, es decir, existe la herencia múltiple. Por otro lado, las propiedades de las clases también se heredan. En nuestro ejemplo, como se puede comprobar, sólo hemos definido propiedades a las clases Obra y Persona, y éstas son heredadas por sus clases hijas.

La propiedad subClassOf es transitiva por definición. Así, si Actor es una subclase de Persona y Protagonista es una subclase de Actor, entonces Protagonista es una subclase de Persona. RDF Schema fija la semántica de "es subclase de" y no es tarea de nuestra aplicación hacer establecer estas interpretaciones.

En este punto, disponemos de un vocabulario simple del dominio del Cine expresado con términos RDF Schema. Este vocabulario puede ayudar a las aplicaciones a realizar inferencias, una forma de razonamiento deductivo, basándose en el conocimiento expresado en el vocabulario. El conocimiento lo podemos obtener en base a:

- Las propiedades que pueden ser usadas
- Los tipos de objetos que pueden ser usados como valores de las propiedades.

Por ejemplo. Película:

- Es una clase
- Podemos emplear la propiedad Titulo con un valor de string.
- Podemos emplear la propiedad Sinopsis con un valor de string.
- Podemos emplear la propiedad estaDirigidaPor con un valor de instancia Director.
- Podemos emplear la propiedad estaProducidaPor con un valor de instancia Productor.
- Podemos emplear la propiedad estaProtagonizadaPor con un valor de instancia Protagonista.

2.8 Creación de instancias de la ontología.

Las instancias las creamos desde el prototipo que hemos desarrollado. No están mezcladas con la ontología. Tanto la ontología como las instancias se almacenan en Derby separados.

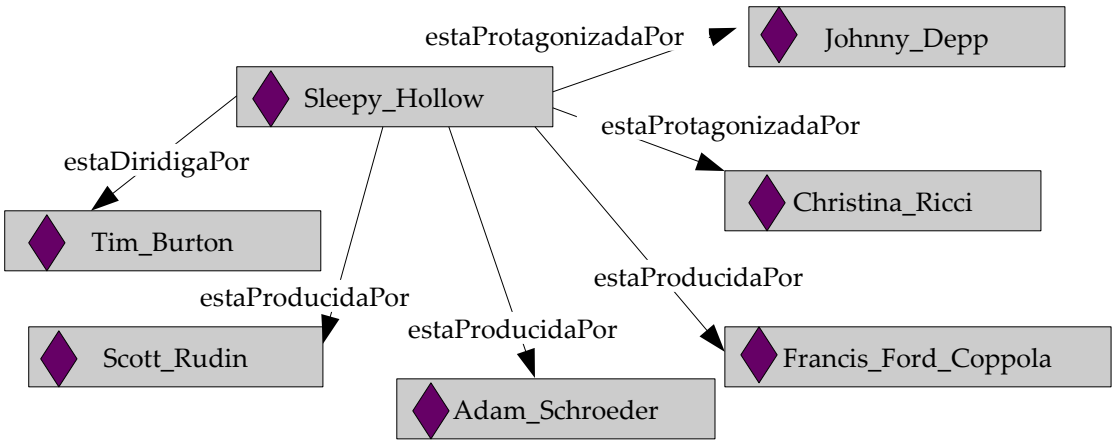


Figura 21 Ejemplo de instancias creadas con la ontología.

Las sentencias RDF que hemos creado para este recurso son la que se indican a continuación.

```
<rdf:Description rdf:about="http://dbpedia.org/resource/Sleepy_Hollow_%28film%29">
  <j.0:Sinopsis>Sleepy Hollow is a 1999 American period horror film directed by Tim Burton. It is a film adaptation loosely inspired by the 1820 short story "The Legend of Sleepy Hollow" by Washington Irving and stars Johnny Depp, Christina Ricci, Miranda Richardson, Marc Pickering, Michael Gambon, Jeffrey Jones, Casper Van Dien, Ian McDiarmid, Michael Gough, and Christopher Walken. The plot follows police constable Ichabod Crane (Depp) sent from New York City to investigate a series of murders in the village Sleepy Hollow by a mysterious Headless Horseman. Development for Sleepy Hollow began in 1993 at Paramount Pictures with Kevin Yagher originally set to direct Andrew Kevin Walker's script as a low-budget slasher film. Disagreements with Paramount resulted in Yagher being demoted to prosthetic makeup designer, and Burton was hired to direct in June 1998. Filming took place from November 1998 to May 1999, and Sleepy Hollow was released to generally favorable reviews from critics, and grossed approximately $207 million worldwide. Production designer Rick Heinrichs and set decorator Peter Young won the Academy Award for Best Art Direction.</j.0:Sinopsis>
  <j.0:Titulo>Sleepy Hollow</j.0:Titulo>
  <j.0:estaDirigidaPor rdf:resource="http://dbpedia.org/resource/Tim_Burton"/>
  <j.0:estaProducidaPor rdf:resource="http://dbpedia.org/resource/Adam_Schroeder"/>
  <j.0:estaProducidaPor rdf:resource="http://dbpedia.org/resource/Francis_Ford_Coppola"/>
  <j.0:estaProducidaPor rdf:resource="http://dbpedia.org/resource/Scott_Rudin"/>
  <j.0:estaProtagonizadaPor rdf:resource="http://dbpedia.org/resource/Christina_Ricci"/>
  <j.0:estaProtagonizadaPor rdf:resource="http://dbpedia.org/resource/Johnny_Depp"/>
</rdf:Description>
```

Ficgura 22 Sentencias RDF que describen la película Sleepy Hollow.

Para el director Tim Burton, las sentencia son estas:

```
<rdf:Description rdf:about="http://dbpedia.org/resource/Tim_Burton">
   <j.0:Nombre>Tim Burton</j.0:Nombre>
   <j.0:esDirectorDe rdf:resource="http://dbpedia.org/resource/The_Island_of_Doctor_Agor"/>
   <j.0:esDirectorDe
rdf:resource="http://dbpedia.org/resource/Sweeney_Todd:_The_Demon_Barber_of_Fleet_Street_
%282007_film%29"/>
   <j.0:esDirectorDe rdf:resource="http://dbpedia.org/resource/Stalk_of_the_Celery_Monster"/>
   <j.0:esDirectorDe rdf:resource="http://dbpedia.org/resource/Sleepy_Hollow_%28film%29"/>
   <j.0:esDirectorDe rdf:resource="http://dbpedia.org/resource/Planet_of_the_Apes_%282001_film%29"/>
   <j.0:esProductorDe rdf:resource="http://dbpedia.org/resource/Stalk_of_the_Celery_Monster"/>
   <rdf:type rdf:resource="http://www.sfernandezve.org/cine#Director"/>
   <rdf:type rdf:resource="http://www.sfernandezve.org/cine#Productor"/>
 </rdf:Description>
```

Figura 23 Sentencias RDF que describen al director de cine Tim Burton.

Es importante hacer notar que, de forma deliberada, en el prototipo hemos omitido el tipado en las sentencias que describen las instancias de la clase Película.

```
<rdf:type rdf:resource="http://www.sfernandezve.org/cine#Pelicula"/>
```

Mientras que en el resto de instancias, como podemos ver en la del director Tim Burton en la figura 23, si que se han añadido.

```
<rdf:type rdf:resource="http://www.sfernandezve.org/cine#Director"/>
<rdf:type rdf:resource="http://www.sfernandezve.org/cine#Productor"/>
```

El tipado es muy importante, identifica la clase o clases a la que pertenece una instancia. Si el fichero que contiene las sentencias RDF lo usamos sin una ontología, además de los inconvenientes que ya hemos visto, no podremos determinar la clase a la que pertenece la instancia. Es decir, no podemos obtener directamente las instancias de la clase Película porque no se puede determinar cuales son. Sin embargo, si que podemos obtener todas las instancias de la clase Director, Productor o Protagonista, ya que todas tiene su tipo definido.

Usado con una ontología y por medio de la inferencia, si se cumplen determinados criterios que ahora veremos, se puede determinar la clase de una instancia y se le aplica el tipado correspondiente a las instancias que no lo tengan definido. Hemos hecho este ejercicio precisamente para ver como funciona la inferencia.

Con el proyecto se adjuntan los siguientes ficheros.

cine.rdfs

Contiene la ontología que hemos diseñado en este documento.

cine.rdf

Contiene el fichero con todas las instancias que se crean en el prototipo. El contenido es fiel reflejo de lo que hemos encontrado en DBpedia.org en el momento de hacer este documento.

cine_resultado_modelo_inferencia.txt

Contiene la transformación del fichero cine.rdf después de utilizar inferencia con un reasoner de RDF Schema y aplicando éste sobre el fichero que contiene la ontología.

Si comparamos los ficheros cine.rdf y cine_resultado_modelo_inferencia.txt veremos que el segundo, tiene más sentencias que el primero, (1468 líneas el primero frente a 2388 líneas el segundo). El motivo es que este fichero contiene las sentencias que hemos declarado de forma explícita y las que se han inferido por medio del reasoner RDF Schema.

2.9 Inferencia.

Usando un reasoner de RDF Schema que combine el fichero RDF y la ontología se derivan los resultados que hemos obtenido en nuestra aplicación. Para obtenerlos, hemos construido las instancias de la clase Película omitiendo su tipado y las instancias de la clase Actor, las hemos tipado sólo como instancias de la clase Protagonista, así se cumplen algunos requisitos mas que nos permiten ver cómo funciona la inferencia tal y como indicamos a continuación.

A) Se deduce el tipo de clase de un recurso a partir de las propiedades de la etiqueta **rdfs:domain** que hemos definido en la ontología.

Cuando hemos definido las propiedades hemos usado rdfs:domain para indicar de forma específica cuales son las clases que pueden utilizar esta propiedad para describirla. Por ejemplo, la propiedad Título, tal y como la hemos definido indica que su dominio son las clases del tipo Obra.

```
<rdf:Property rdf:ID="Titulo">
        <rdfs:domain rdf:resource="#Obra"/>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</rdf:Property>
```

El reasoner RDF Schema, si identifica que un recurso está descrito por esta propiedad, entonces deduce que el recurso pertenece a la clase que esta propiedad tiene definida en el dominio. Así todos los recursos descritos con la propiedad Título son de clase Obra. Lo mismo ocurre con las propiedades estaDirigidaPor, estaProducidaPor, estaProtagonizdaPor, en estos casos el dominio indica:

<rdfs:domain rdf:resource="#Pelicula"/>

Entonces, de la misma forma, todos los recursos descritos por estas propiedades son de la clase Película.

B) Se deduce el tipo de clase de un recurso a partir de las propiedades de la etiqueta **rdfs:range** que hemos definido en la ontología.

Cuando hemos definido las propiedades, hemos usado rdfs:range para indicar de forma específica los valores que estas propiedades pueden tomar. Así, la propiedad Título puede tomar valores de tipo String,

<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>

Las propiedades esDirectorDe, esProductorDe, esProtagonistaDe pueden tomar estos valores:

<rdfs:range rdf:resource="#Pelicula"/>

Si un recurso está descrito por esta propiedad, y el valor de esta propiedad se usa para describir otro recurso representado por una URI específica, entonces este recurso es de la clase indica en rdfs:range.

Así se puede deducir el tipado de las instancias de la clase Película.

C) Se deduce la **superclase** a la que pertenece el recurso a partir de la jerarquía de clases definida en la ontología.

Como consecuencia de los dos casos anteriores, una vez que se identifica la clase de una instancia, se puede hacer una búsqueda sobre la jerarquía de clases, y así determinar que la instancia también es del tipo de todas sus superclases.

<rdfs:Class rdf:ID="**Pelicula**">
<**rdfs:subClassOf rdf:resource="#Obra**"/>
</rdfs:Class>

Así una vez identificada una instancia y tipada como Película, entonces también se hace el tipado como Obra.

Con las instancias Director, y Productor, se procede al tipado como Persona. Con las instancias de Protagonista, se procede al tipado primero como Actor y después como Persona.

Si comparamos el contenido de los ficheros que hemos indicado al final del apartado anterior, podemos apreciar las diferencias existentes.

Descripción de la película The Emperor en el fichero cine.rdf.

```
<rdf:Description rdf:about="http://dbpedia.org/resource/The_Emperor_%28short_film%29">
  <j.0:Sinopsis>The Emperor is a short film by George Lucas about a radio DJ.</j.0:Sinopsis>
  <j.0:Titulo>The Emperor</j.0:Titulo>
  <j.0:estaDirigidaPor rdf:resource="http://dbpedia.org/resource/George_Lucas"/>
  <j.0:estaProducidaPor rdf:resource="http://dbpedia.org/resource/Gary_Kurtz"/>
  <j.0:estaProducidaPor rdf:resource="http://dbpedia.org/resource/George_Lucas"/>
  <j.0:estaProtagonizadaPor rdf:resource="http://dbpedia.org/resource/Robert_Hudson"/>
 </rdf:Description>
```

Descripción de la película The Emperor después de aplicar el modelo de inferencia en el fichero cine_resultado_modelo_inferencia.txt.

```
<rdf:Description rdf:about="http://dbpedia.org/resource/The_Emperor_%28short_film%29">
  <j.0:Sinopsis>The Emperor is a short film by George Lucas about a radio DJ.</j.0:Sinopsis>
  <j.0:Titulo>The Emperor</j.0:Titulo>
  <j.0:estaDirigidaPor rdf:resource="http://dbpedia.org/resource/George_Lucas"/>
  <j.0:estaProducidaPor rdf:resource="http://dbpedia.org/resource/Gary_Kurtz"/>
  <j.0:estaProducidaPor rdf:resource="http://dbpedia.org/resource/George_Lucas"/>
  <j.0:estaProtagonizadaPor rdf:resource="http://dbpedia.org/resource/Robert_Hudson"/>
  <rdf:type rdf:resource="http://www.sfernandezve.org/cine#Pelicula"/>
  <rdf:type rdf:resource="http://www.sfernandezve.org/cine#Obra"/>
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdf:Description>
```

Descripción del actor Tom Hanks en cine.rdf.

```
<rdf:Description rdf:about="http://dbpedia.org/resource/Tom_Hanks">
  <j.0:Nombre>Hanks, Tom</j.0:Nombre>
  <j.0:esProtagonistaDe rdf:resource="http://dbpedia.org/resource/The_Terminal"/>
  <rdf:type rdf:resource="http://www.sfernandezve.org/cine#Protagonista"/>
 </rdf:Description>
```

Descripción del actor Tom Hanks después de aplicar el modelo de inferencia en el fichero cine_resultado_modelo_inferencia.txt.

```
<rdf:Description rdf:about="http://dbpedia.org/resource/Tom_Hanks">
  <j.0:Nombre>Hanks, Tom</j.0:Nombre>
  <j.0:esProtagonistaDe rdf:resource="http://dbpedia.org/resource/The_Terminal"/>
  <rdf:type rdf:resource="http://www.sfernandezve.org/cine#Protagonista"/>
  <rdf:type rdf:resource="http://www.sfernandezve.org/cine#Persona"/>
  <rdf:type rdf:resource="http://www.sfernandezve.org/cine#Actor"/>
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
 </rdf:Description>
```

El funcionamiento de la inferencia con un reasoner RDF Schema es simple y fácil de entender con los ejemplos que hemos mostrado. El resultado que obtenemos se ha enriquecido con información que antes no teníamos, lo que nos permite obtener mas conocimiento a partir de ésta.

En el prototipo, nos aprovechamos de la inferencia, y probamos cómo funciona la tecnología, para identificar las instancias del tipo Obra de las cuales hemos extraído información durante el proceso de descarga. También la usamos para identificar las instancias del tipo Película que tenemos y mostrar por pantalla un listado de las mismas para que podamos consultar su ficha.

2.10 SPARQL.

Podríamos decir que SPARQL es a RDF lo que SQL es a una base de datos relacional, aunque es cierto que SPARQL todavía no está a la altura del SQL. SPARQL se ha diseñado para realizar consultas sobre documentos que tienen formato RDF y es una recomendación del W3C[15], así, es un lenguaje de consultas y un protocolo para el acceso de datos en la Web. La motivación de SPARQL es la gran cantidad de información en formato RDF que ha ido apareciendo en la Web, gracias a la difusión y el uso de las técnicas de web semántica, y la necesidad de procesarla de forma local o accediendo a ficheros remotos cuyo contenido esté en formato RDF. Con un lenguaje de estas características podemos:

- Hacer consultas sobre documentos RDF para obtener algún tipo de información.
- Dirigir las consultas a un servidor RDF remoto y obtener información devuelta por él.
- Ejecutar consultas programadas contra ficheros RDF para generar informes.
- Mejorar nuestras aplicaciones permitiendo que trabajen con los datos devueltos por las consultas en vez de con los datos en formato RDF directamente.

Un documento en formato RDF puede estar almacenado en un fichero o en una base de datos. Cuando está almacenado en una base de datos, hablamos de bases de datos RDF o también de "Triple Store" o RDF data stores. Estas bases de datos se pueden crear en gestores de bases de datos de propósito general (DBMS) como MySQL, Derby, PostgreSQL, ORACLE, etc. Pero dada la naturaleza del formato de los datos RDF, se han desarrollado unos gestores de bases de datos específicos y optimizados para realizar el almacenamiento y la recuperación de sentencias RDF de forma efectiva. Entre ellos, destacan Joseki[16] y Virtuoso[17]. Para ver las funcionalidades de cada uno de ellos es recomendable visitar los sitios indicados en los enlaces.

Para poder realizar nuestras consultas usando SPARQL se han desarrollado unas interfaces que pueden ser usadas desde programas de aplicación o directamente por personas y se denominan "***SPARQL enpoints***". Su función es aceptar y validar las consulta y devolver el resultado de las mismas. Para su uso por personas, un SPARQL endpoint puede ser una aplicación instalada de forma local o una aplicación basada en Web que podemos utilizar desde nuestro navegador, como la que ofrece DBpedia[18]. Para uso desde programas, disponemos de un conjunto de API destinados a tal efecto.

Un SPARQL endpoint puede ser genérico o específico. Un endpoint genérico puede trabajar con cualquier fichero RDF independientemente de que esté almacenado localmente o de que sea accesible a través de la Web. Un endpoint específico está vinculado a un único fichero RDF y este no se puede cambiar.

15 http://www.w3.org/TR/rdf-sparql-query/
16 http://www.joseki.org
17 http://virtuoso.openlinksw.com
18 http://dbpedia.org/snorql/

En los prototipos del proyecto emplearemos SPARQL para extraer información de DBpedia usando el endpoint que facilita y con los datos obtenidos poblaremos las entidades de la ontología que hemos diseñado. Posteriormente, usando SPARQL, realizaremos consultas sobre las entidades que hemos creado y almacenado de forma persistente en un gestor de bases de datos (que no es obligatorio, podemos utilizar ficheros para hacer la persistencia, pero un gestor de bases de datos nos aporta mas ventajas como veremos mas adelante), navegando por la ontología y sus relaciones.

SPARQL, básicamente, dispone de cuatro tipos de consultas.

- SELECT.
- ASK.
- DESCRIBE.
- CONSTRUCT.

La más común es SELECT. Para realizar las consultas con SPARQL debemos tener en cuenta que se basa en dos patrones, el de terna (triple) y el de grafo.

El modelo de terna de RDF se compone de, tal y como hemos visto, sujeto + predicado + objeto. SPARQL se basa en el mismo modelo de terna, pero la diferencia es que cualquiera de los tres elementos puede ser una variable. Además las ternas tienen se tienen que finalizar con un punto "." como delimitador. Las variables se forman con el símbolo ? seguido de un nombre, (?variable). El nombre de la variable es indiferente.

El modelo de terna, cuando se aplica a un documento RDF funciona de esta forma.

1. Crea un "resultSet" vacío.
2. Recupera la siguiente terna del documento RDF. Si no quedan mas ternas, entonces devuelve el "resultSet".
3. Compara las elementos que no son variables de la terna de la consulta con la terna del documento RDF, si los elementos que no son variables coinciden entonces cada elemento definido como variable toma el valor del elemento correspondiente del documento RDF formando así una nueva terna que se añade al "resultSet " creado en el punto 1.
4. Volver al punto 2.

Como ejemplo, nos conectamos al SPARQL enpoint de DBpedia en http://dbpedia.org/snorql/ y realizamos esta consulta:

```
SELECT * WHERE { :Stanley_Kubrick dbpedia2:name ?name . }
```

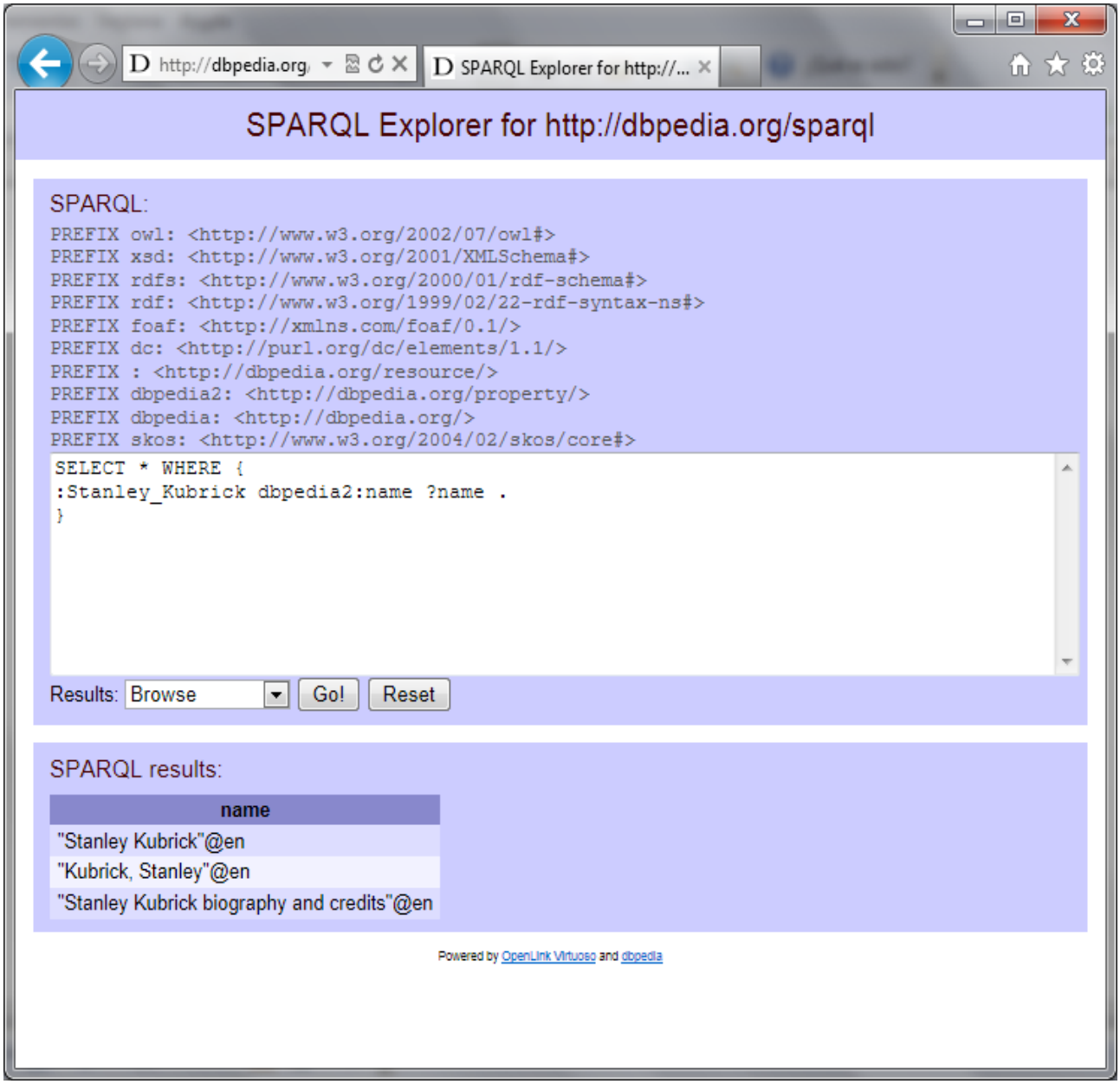


Figura 24: Consulta SPARQL sobre DBpedia.

Obtendremos un "resultSet" con tres resultados, tal y como podemos observar en la imagen anterior.

Si queremos consultar todas las ternas que contiene el documento RDF y que describen el recurso http://dbpedia.org/resource/Stanley_Kubrick podemos emplear esta consulta:

```
SELECT * WHERE { :Stanley_Kubrick ?propiedad ?valor . }
```

El patrón de grafo SPARQL es similar al patrón de terna que acabamos de ver, pero permite especificar reglas de selección mucho mas complejas. Vamos a ver el funcionamiento con esta consulta.

```
PREFIX dbpedia1: <http://dbpedia.org/ontology/>
SELECT ?titulo WHERE {
        ?titulo a dbpedia1:Film  .
        ?titulo dbpedia1:director :Stanley_Kubrick  .
} ORDER BY DESC(?titulo) LIMIT 5 OFFSET 1
```

Figura 25:Consulta SPARQL sobre documento RDF.

1. Crea un "resultSet" vacío.
2. Recupera el siguiente recurso del documento RDF. Si no quedan mas recursos, entonces devuelve el "resultSet".
3. Procesa la primera terna. (?titulo a dbpedia1:Film .)

   - Si el recurso actual no no es del tipo "Film" va al punto 5.
4. Procesa la segunda terna . (?titulo dbpedia1:director :Stanley_Kubrick .)

   - Si el recurso actual no tiene una propiedad que se llame dbpedia1:director va al punto 5.

   - Si el recurso actual tiene una propiedad que se llame dbpedia1:director y ésta vale :Stanley_kubrick

   entonces el valor del sujeto de la terna, donde hemos situado la variable (?titulo), se añade al "resulSet".
5. Volver al punto 2.

Es evidente que la consulta basada en modelo de grafo busca en todos los recursos de tipo "Film" e intenta buscar en ellos si tienen definida la propiedad dbpedia1:director. Es importante observar que el contenido de la variable obtenido como resultado de ejecutar la primera terna pasa como valor a la variable de la segunda terna y esto ocurre para cada recurso tipo "Film" encontrado en el grafo.

En la siguiente figura, podemos ver el resultado de la ejecución de la consulta, el cual esta ordenado de forma descendente y limitado a los cinco primeros resultados comenzando desde el primero de todos ellos tal y como hemos indicado (ORDER BY DESC(?titulo) LIMIT 5 OFFSET 1).

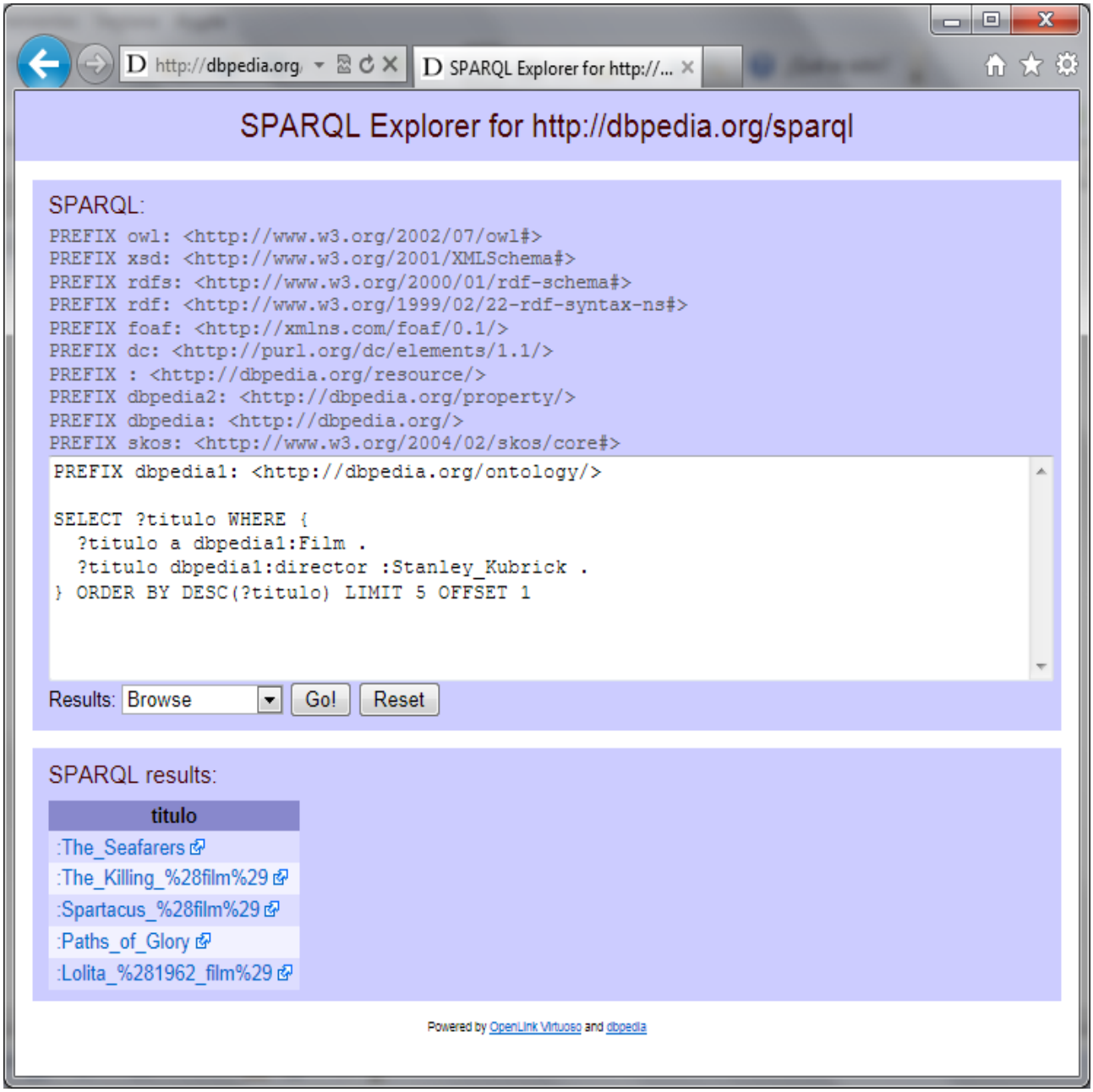


Figura 26: Resultado de consulta SPARQL sobre documento RDF.

Para ampliar mas información sobre como realizar consultas debemos utilizar la bibliografía recomendada[19].

19 Learning SPARQL *Querying and Updating with SPARQL 1.1* by Bob DuCharme O'REILLY

3. Prototipos.

Los prototipos desarrollados realizan las siguientes funciones. Uno de ellos, conectándose a la DBpedia, rellena con instancias la ontología creada y el otro permite navegar la ontología creada junto con las instancias añadidas de una forma visual a través de un navegador Web.

Los prototipos se han desarrollado utilizando eclipse como IDE de desarrollo y Apache Tomcat como servidor de aplicaciones. También utilizan Apache Derby en modo embebido dentro de la aplicación con la finalidad de servir de almacén de datos para la ontología que hemos creado y para las instancias de la misma rellenas con los datos que se extraen del sitio web de DBpedia. Tanto la extracción de documentos como su consulta se realizan desde la aplicación en un navegador Web. El lenguaje de programación de los prototipos es Java y se ha usado Jena que es un Framework de desarrollo en Java de aplicaciones de tecnología Web semántica.

Antes de continuar con la descripción de los prototipos, haremos una breve introducción de DBpedia y de Jena.

3.1 DBpedia.

DBpedia[20] es un proyecto que tiene como finalidad extraer información estructurada de Wikipedia y hacer que esta información esté disponible en la Web, haciendo mas fácil su uso gracias a las tecnologías de Web semántica. DBpedia nos permite realizar consultas complejas sobre los datos extraídos de Wikipedia.

El proyecto[21] se basa en extraer información semántica a partir de la información estructurada que contiene un documento Web y hacer esta actividad de forma automática, sin ninguna intervención manual. Una vez extraída la información, esta se transforma en un documento RDF, de manera que hay una correspondencia entre la página original de Wikipedia y la página de DBpedia (la que está en formato legible por personas). El proceso de extracción se repite para cada página de Wikipedia y el resultado, todos los documentos RDF que se construyen, se añaden a un fichero, el fichero RDF de DBpedia. En este fichero, todos los documentos RDF comparten el mismo conjunto de ontologías y por lo tanto podemos hacer consultas sobre el fichero y encontrar la información que queremos con relativa facilidad.

Las consultas sobre el contenido de DBpedia se pueden realizar de diferentes formas. Sin embargo la que mas nos interesa es la que el proyecto DBpedia en si tiene como finalidad, facilitar el acceso a la información a través del SPARQL endpoint. Este es el método que se usa en el prototipo para extraer la información que necesitamos para crear y poblar instancias de la ontología que hemos creado. A pesar de que no vamos a acceder a las páginas Web para extraer contenido de ellas ni tampoco a los ficheros individuales que contienen la información en formato RDF, haremos a continuación un pequeño repaso de ellos para entender mejor el proceso de transformación que se realiza sobre la información al pasar desde Wikipedia hasta DBpedia.

Tal y como hemos indicado hay una correspondencia entre las páginas de Wikipedia y el contenido de DBpedia. Por ejemplo, la pagína:

http://en.wikipedia.org/wiki/Stanley_Kubrick

Tiene correspondencia con esta otra:

http://dbpedia.org/resource/Stanley_Kubrick

20 http://dbpedia.org/About
21 http://www.swib09.de/vortraege/20091124_jentzsch.pdf

Sin embargo cuando accedemos desde un navegador Web el sitio de DBpedia nos redirige a la que es comprendida por personas:

http://dbpedia.org/page/Stanley_Kubrick

Y el fichero que contiene el documento RDF con el contenido de los datos de esta página es:

http://dbpedia.org/data/Stanley_Kubrick.rdf

La información que se extrae de una página Web de Wikipedia proviene de un área de la misma que está convenientemente estructurada por medio de una plantilla que se denomina Infobox. De este área no se extrae la imagen. Si desde el navegador Web accedemos una página de Wikipedia que contenga un Infobox, por ejemplo:

http://en.wikipedia.org/wiki/Stanley_Kubrick:

La parte de la plantilla Infobox aparece a la derecha de este texto.

Podremos observar que se compone, sencillamente, de parejas propiedad + valor, sobre un sujeto, el recurso que estamos accediendo desde el navegador. (http://en.wikipedia.org/wiki/Stanley_Kubrick). Los mismos elementos que componen un documento RDF. Para ver el formato fuente de la plantilla Infobox, podemos seleccionar la pestaña "Edit" en la misma página.

Figura27 Imagen obtenida de Wikipedia. Representa contenido de plantilla Infobox.

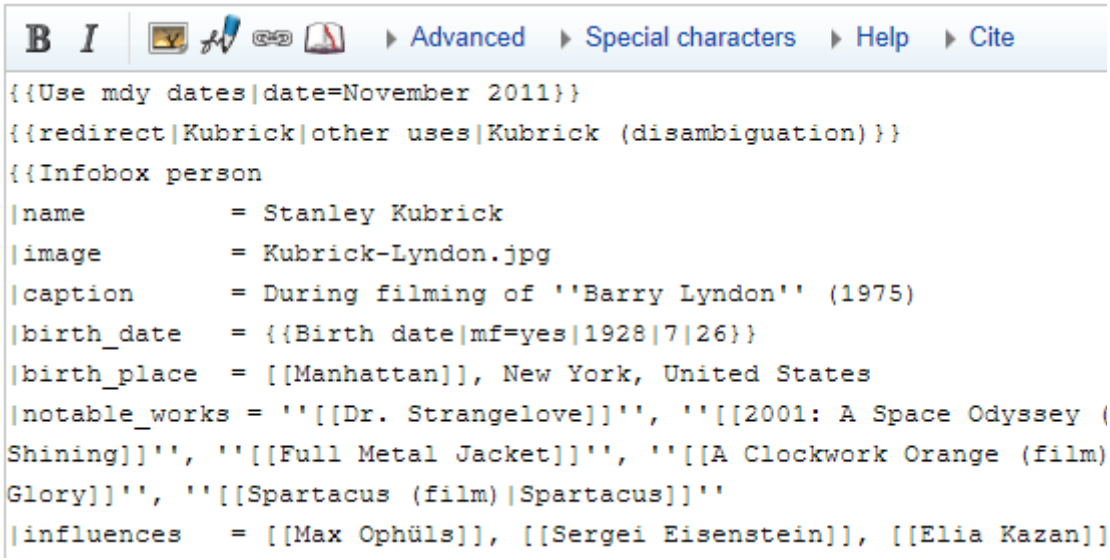


Figura 28 Contenido plantilla Infobox Wikipedia. Página de Stanley Kubrick.

La ontología de DBpedia se basa en el contenido de las diferentes plantillas Infobox que contiene Wikipedia y en algunas de las etiquetas de los datos que se encuentran en cada una de ellas. Hay una plantilla de tipo Infobox para cada categoría que se contempla en Wikipedia, como Personas, Ciudades, Países, Música etc..., y sobre la información contenida en las plantillas se han construido las diferentes clases.

3.2 Jena.

Es un proyecto open source desarrollado en el año 2000 en HP Labs[22] y desde entonces ha sido ampliamente utilizado como Framework de desarrollo de aplicaciones con técnicas de Web semántica. Desde noviembre de 2010, el proyecto está adoptado por la Apache Software Foundation[23].

Jena incluye un API que nos permite realizar estas tareas:

- Leer e interpretar documentos RDF de fuentes externas, ficheros o URLs.
- Generar y grabar documentos RDF. Añadir y quitar ternas de un documento RDF.
- Usar un sistema de almacenamiento eficiente en disco para las ternas RDF.
- Navegar y hacer búsquedas sobre un documento RDF.
- Consultar un fichero RDF por medio de SPARQL.
- Inferencia usando ontologías RDFS y OWL.

22 http://www.hpl.hp.com/
23 http://www.apache.org/

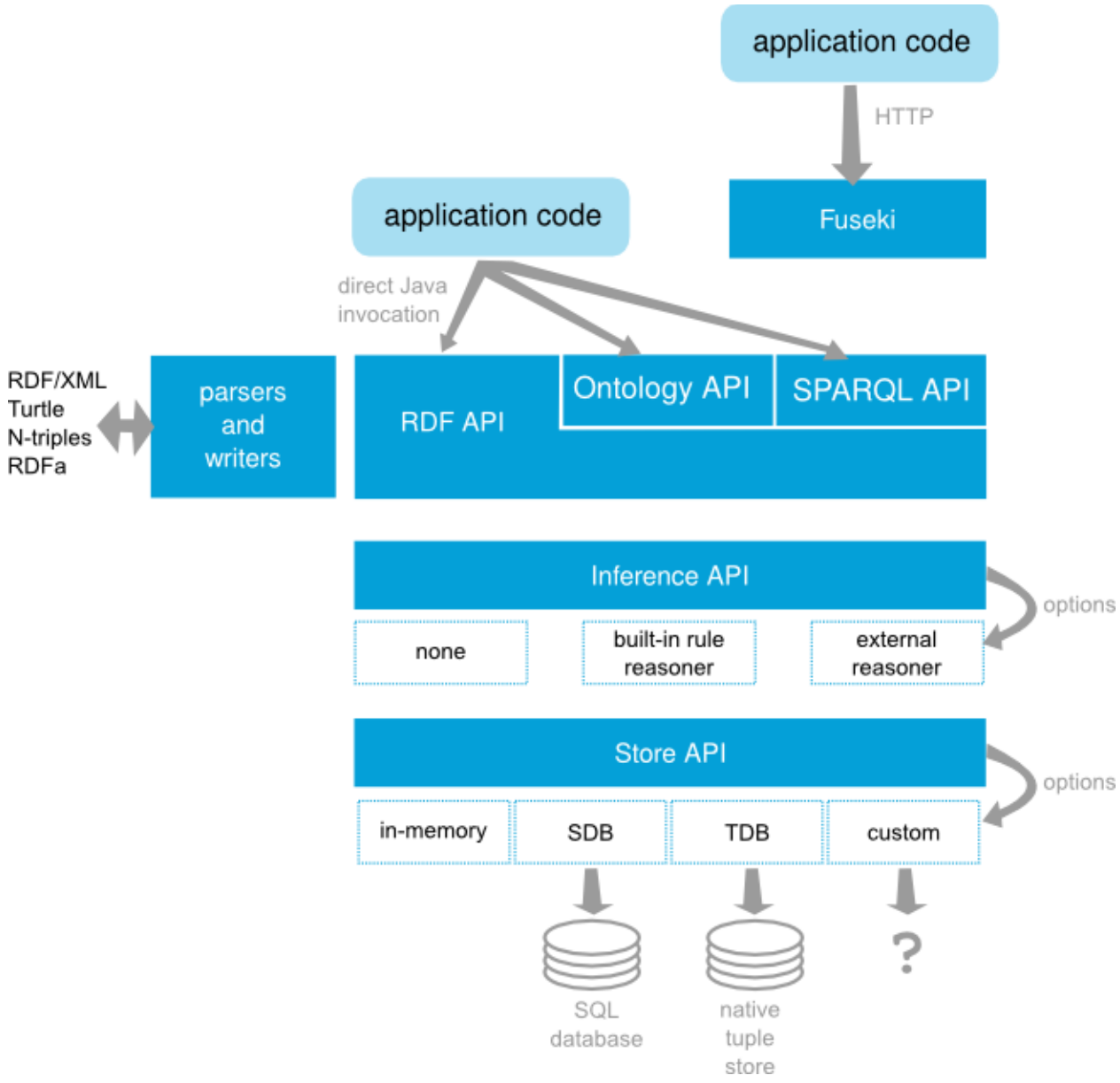


Figura 29Mapa de arquitectura de Jena.[24]

La arquitectura de Jena se compone de estos elementos que pueden ser utilizados directamente desde una aplicación Java.

- **RDF API** que permite gestionar las ternas RDF y sus componentes :
  - Resource. Representa un recurso RDF por medio de su URI o como anónimo.
  - Literal. Representa valores de datos como números, string, fecha, etc.
  - Statement. Representa una terna RDF.
  - Model. Representa un grafo completo.

24 http://incubator.apache.org/jena/about_jena/architecture.html

- **SPARQL API** que da soporte al lenguaje de consultas para RDF.
- **Ontology API**, que soporta los dos lenguajes de ontologías para RDF, el RDFS y el OWL.
- **Inference API**, que permite, a partir de las reglas semánticas definidas con RDF, RDFS y OWL, inferir información que no está explícitamente declarada en el documento RDF. Por ejemplo, si C es subclase de B y B es subclase de A, entonces C es subclase de A. El API de inferencia de Jena, permite que estas ternas estén almacenadas en el documento igual que si se hubieran añadido de forma explícita. Jena suministra varios motores de reglas de inferencia internos para RDFS y OWL (built-in rulesets) y también permite usar otros que sean externos a Jena y desarrollados por terceros (external reasoner).
- **Store API**, que permite almacenar un documento RDF en memoria, en una base de datos SQL, o en TBD que es un sistema de almacenamiento de alto rendimiento, propio, no SQL y especifico para ternas RDF.

Ademas, con Fuseki[25], que es un servidor SPARQL, usando el protocolo SPARQL sobre HTTP, podemos realizar consultas y actualizaciones sobre documentos RDF usando SPARQL[26].

3.3 Descripción de la aplicación.

La aplicación tiene como requisito extraer información de Wikipedia y/o DBpedia y crear instancias de la ontología que hemos diseñado y desde un navegador Web navegar por las relaciones de la ontología.

El usuario de la aplicación no necesita realizar ninguna acción para identificarse como usuario de la misma. Directamente accederá al menú principal de la aplicación.

La idea es que una aplicación de estas características debería funcionar conceptualmente como un rastreador, buscando datos para crear las instancias y sólo necesitaría una URL de partida para comenzar a realizar su trabajo. En este ejemplo, ni siquiera necesitamos este dato ya que usamos el DBpedia SPARQL endpoint y con una simple consulta SPARQL predefinida en la aplicación se puede desencadenar todo el proceso. Sin embargo, para limitar el tiempo que se tarda en realizar la extracción de los datos y el volumen de los mismos, la aplicación dispone de una lista interna de directores de Cine a partir de la cual se realizan el resto de consultas. Es decir, se omite la consulta inicial desencadenante de todo el proceso. Además para cada director, el número de películas sobre las que se obtiene información está también limitado a cinco títulos.

Las consultas son sencillas. Podemos consultar información sobre Películas, Directores, Productores y Protagonistas.

Todas las consultas están diseñadas de la misma forma. Elegimos el tipo de consulta, y sobre la lista de seleccionamos el elemento individual sobre el que queremos obtener información. La información se muestra en forma de ficha en la que se indica a quien o que corresponde, y además sus detalles.

3.4 Análisis de funcionalidades.

Las funcionalidades de la aplicación son:

- Extracción de datos de DBpedia a través de su SPARQL endpoint.
- Creación de instancias a partir de los datos extraídos y de la ontología creada en este proyecto.

25 http://openjena.org/wiki/Fuseki
26 http://www.w3.org/TR/sparql11-update/

- Almacenar las instancias creadas de forma persistente, y la ontología, para poder realizar una consulta de las mismas por medio de una aplicación desde una navegador Web.
- Crear un modelo de inferencia con un reasoner RDF Schema, la ontología y las instancias creadas. Sobre el modelo de inferencia realizar la siguiente consultas:
  - ✔ Consultar relación de películas.
  - ✔ Consultar la ficha de una película.
  - ✔ Consultar relación de directores.
  - ✔ Consultar la ficha de un director.
  - ✔ Consultar relación de productores.
  - ✔ Consultar la ficha de un productor.
  - ✔ Consultar relación de protagonistas.
  - ✔ Consultar la ficha de un protagonista.

3.5 Interfaz de la aplicación.

Como requisito, la aplicación se ha de utilizar desde un navegador Web, así, se ha construido una aplicación Web que permite, de una forma sencilla, realizar las dos funciones que necesitamos. La aplicación se compone de cuatro pantallas.

Una pantalla inicial desde la que podemos realizar el proceso de descarga y lanzar las consultas.

Figura 30 Pantalla principal aplicación.

Una pantalla que muestra las Obras sobre las que se ha extraído información después de pulsar el botón "Descarga". La información que se muestra en esta pantalla se obtiene a partir de las instancias de la clase Obra que hemos obtenido gracias a la inferencia.

Figura 31 Pantalla de informe sobre las Obras de las que se ha extraído información de DBpedia.

La pantalla de consulta. Permite consultar Películas, Directores, Productores y Protagonistas, dependiendo de la opción que seleccionemos en la pantalla principal actuando sobre estos controles.

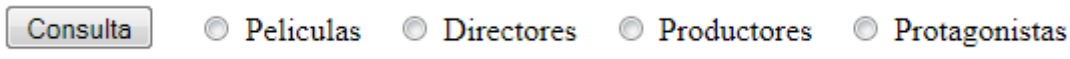
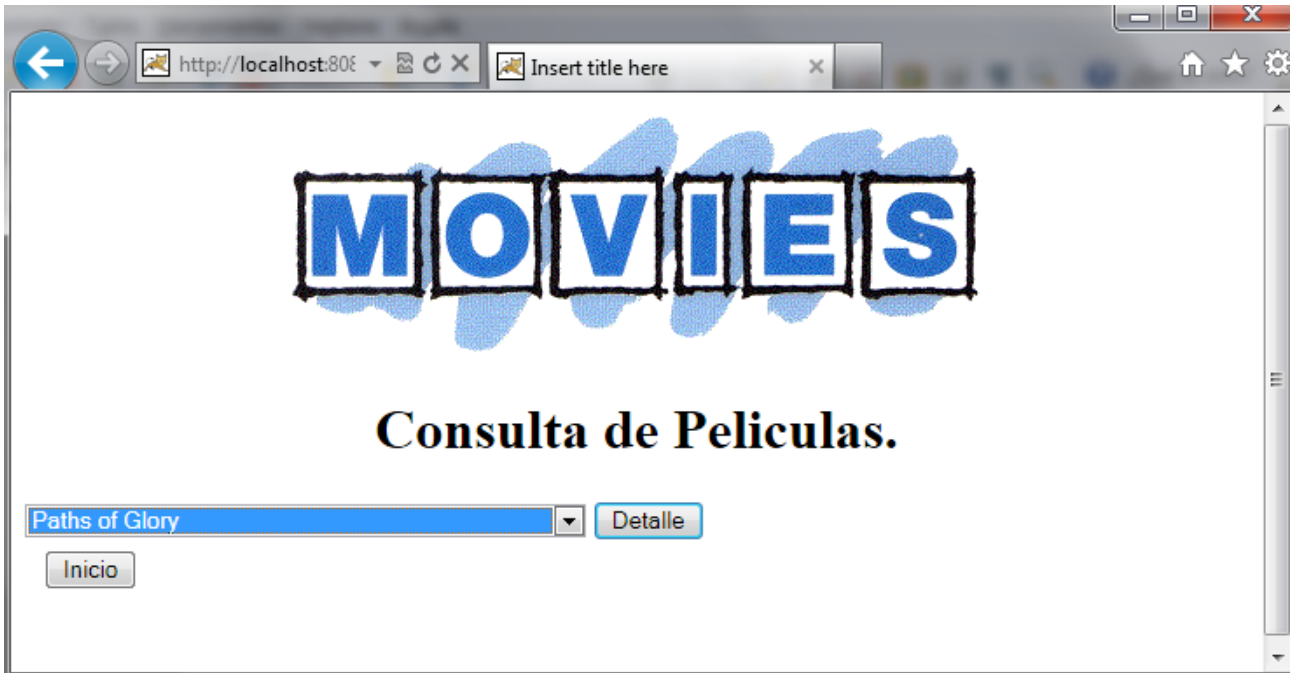


Figura 32 Controles aplicación de consulta.

El listado de películas se puede obtener gracias a la inferencia. Todas las instancias de la clase Película se han tipado por medio del reasoner RDF Schema.



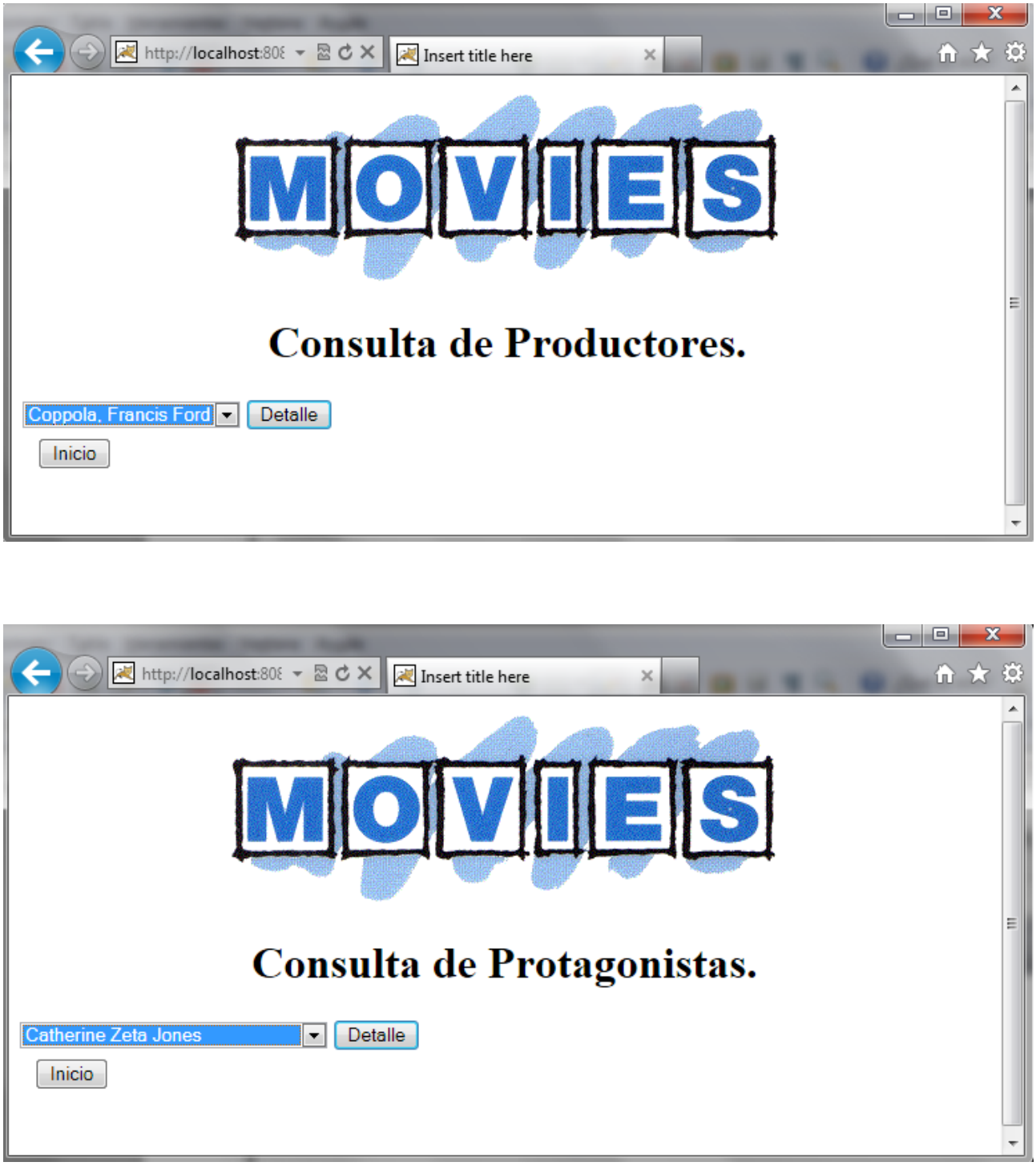
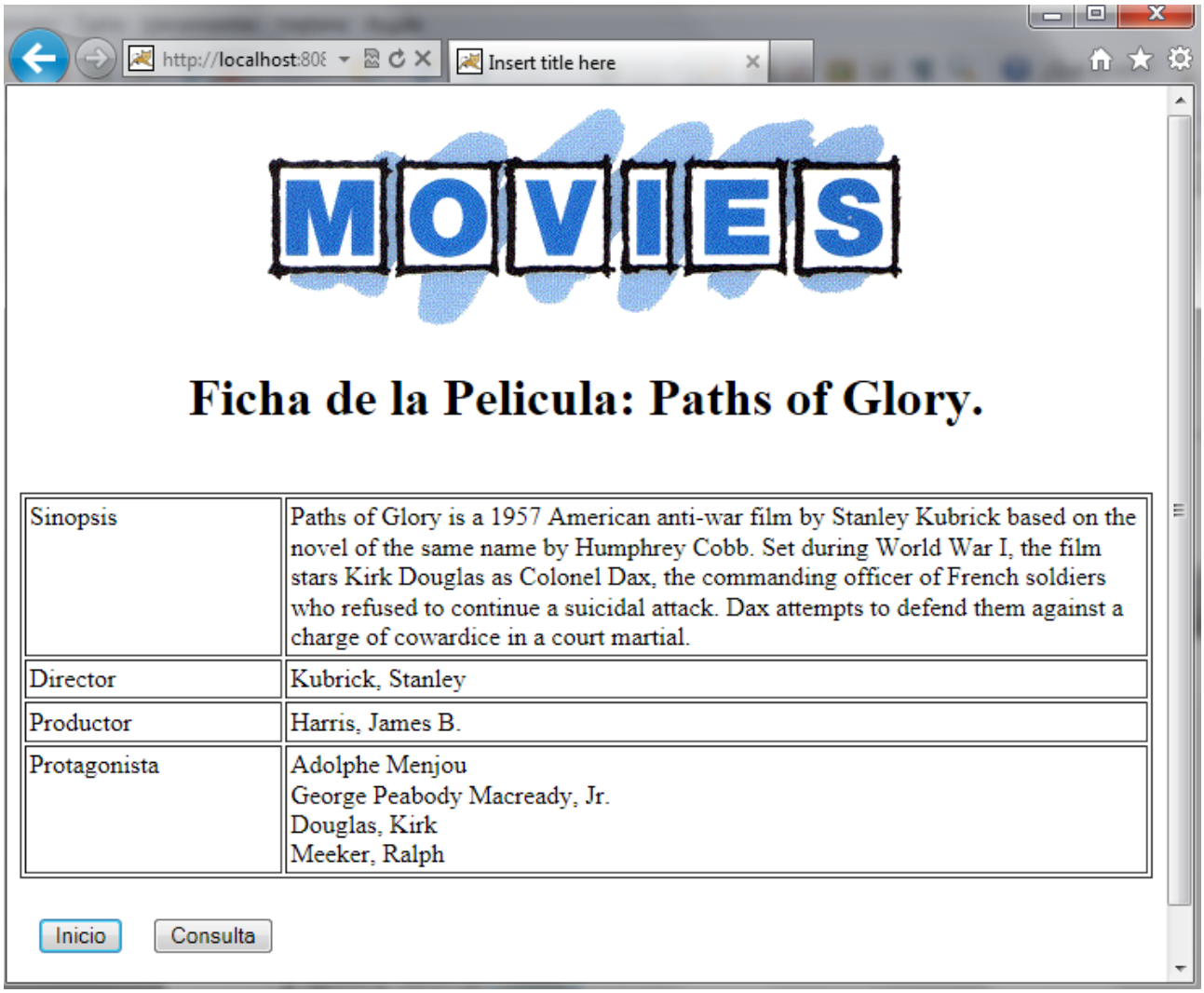
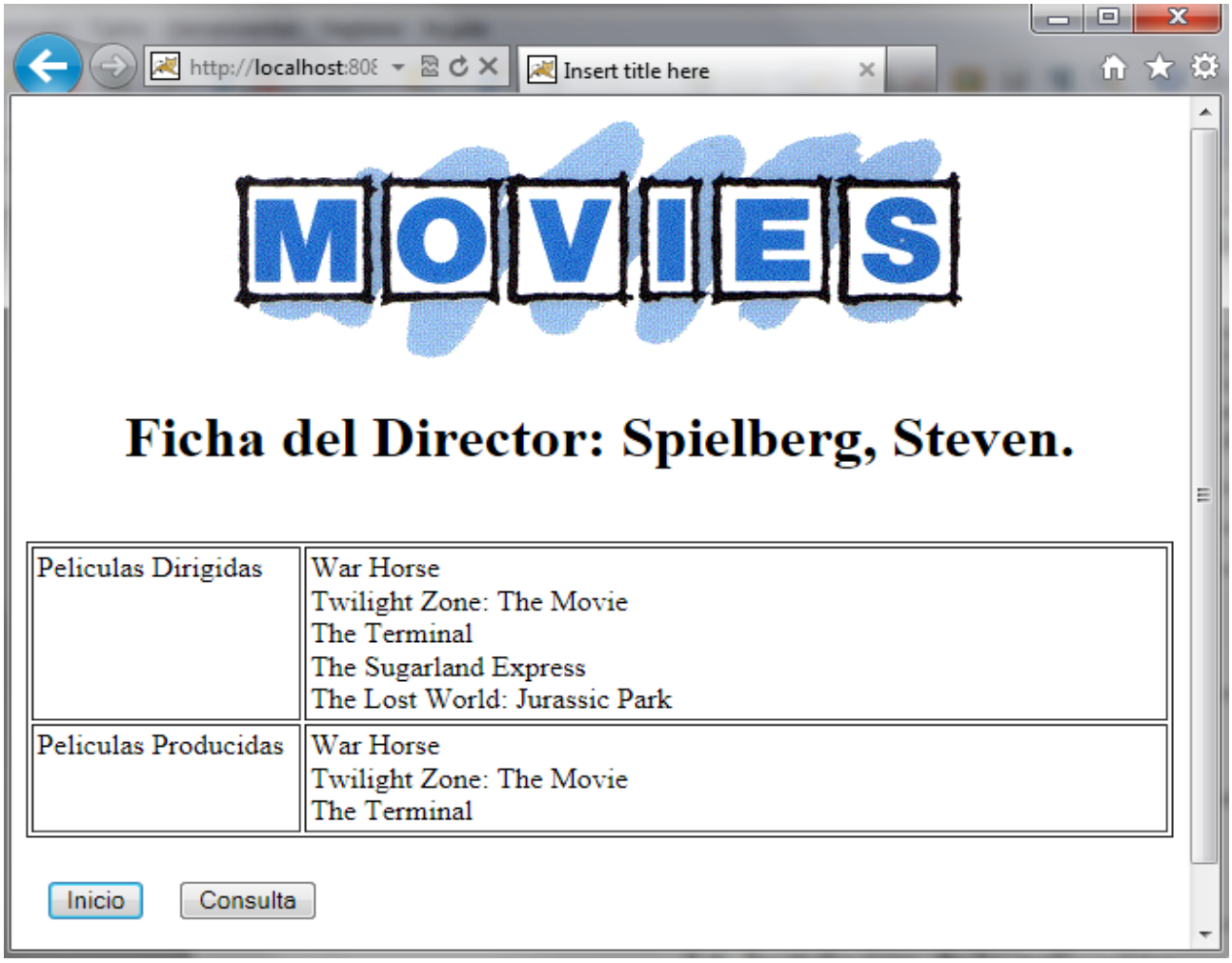
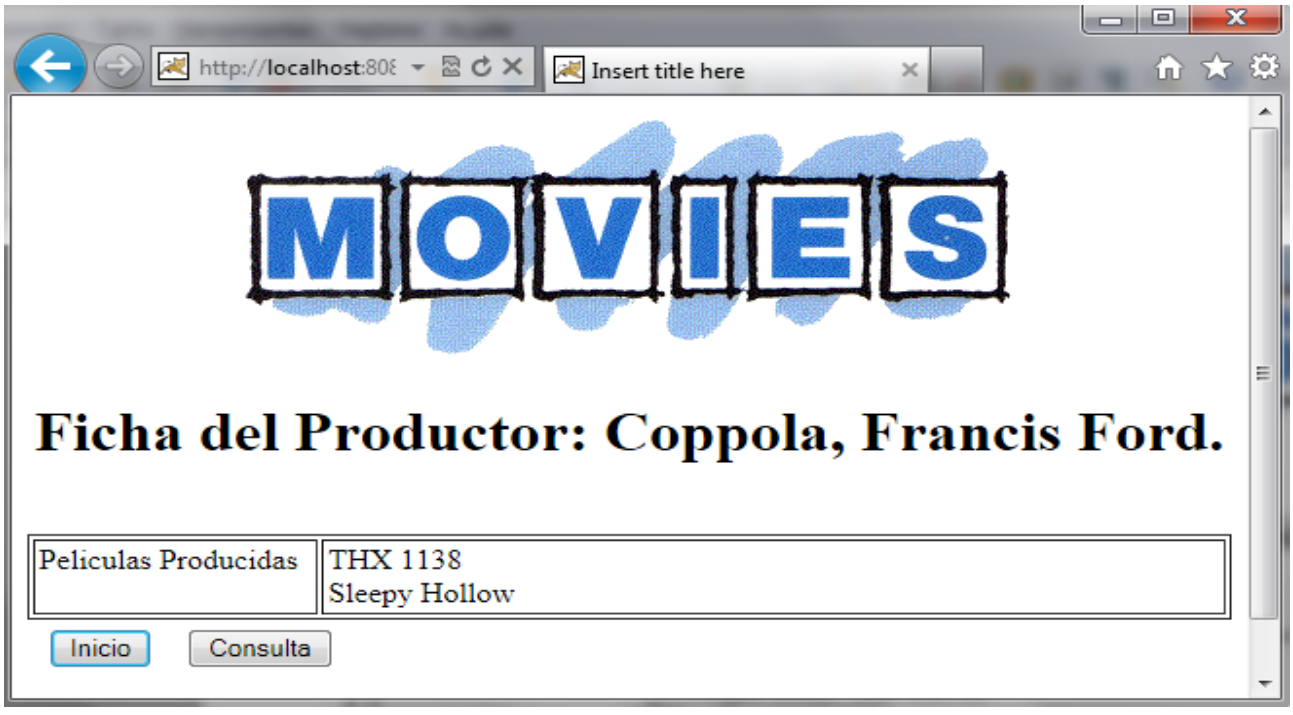


Figura 33 Pantallas de consulta de Películas, Directores, Productores y Protagonistas.

Finalmente, la pantalla de detalle. Se activa en las pantallas de consulta, pulsando el botón “Detalle”.




http://localhost:80
Insert title here
MOVIES
Ficha del Director: Spielberg, Steven.
Peliculas Dirigidas
War Horse
Twilight Zone: The Movie
The Terminal
The Sugarland Express
The Lost World: Jurassic Park
Peliculas Producidas
War Horse
Twilight Zone: The Movie
The Terminal
Inicio
Consulta


http://localhost:80
Insert title here
MOVIES
Ficha del Productor: Coppola, Francis Ford.
Peliculas Producidas
THX 1138
Sleepy Hollow
Inicio
Consulta

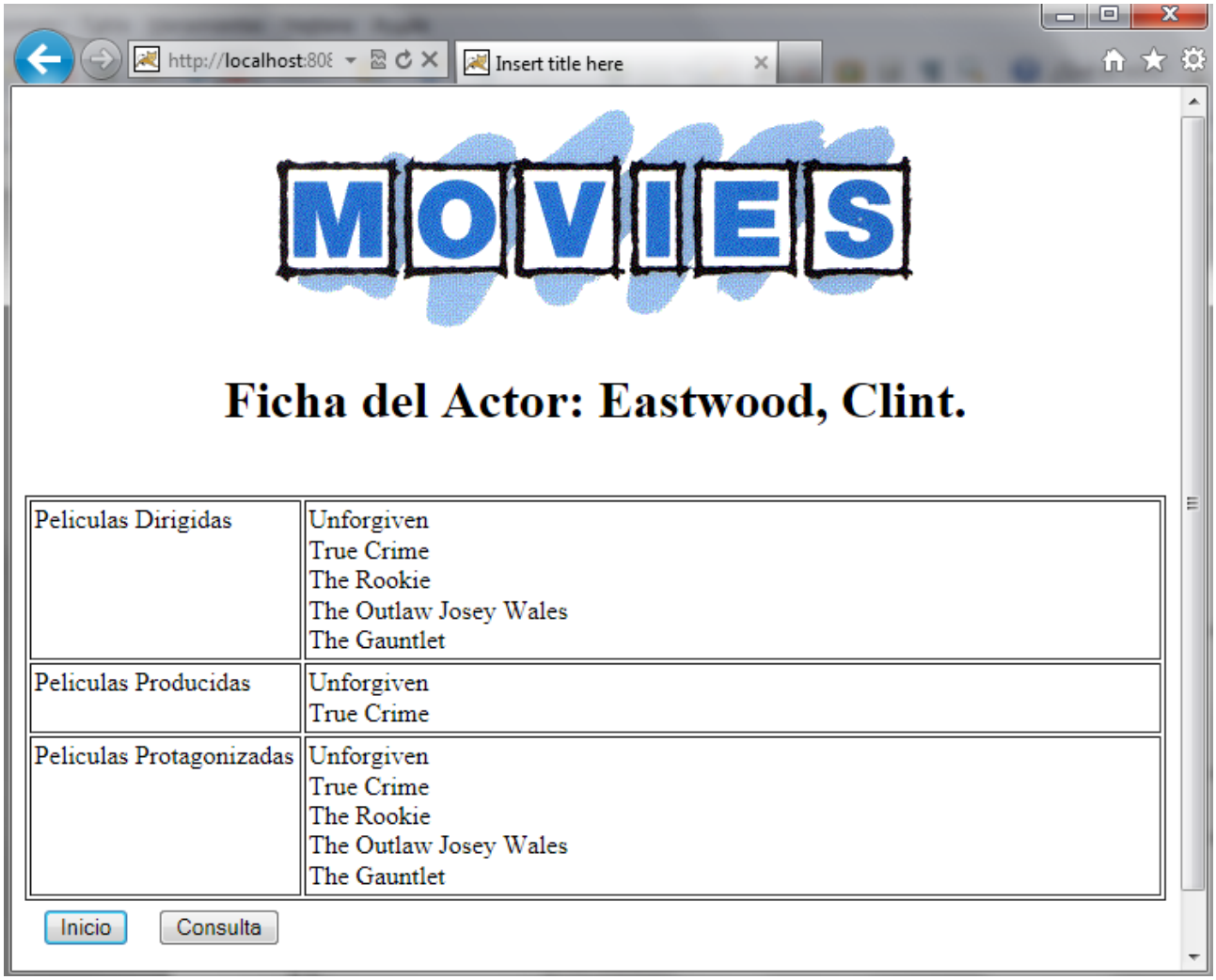


Figura 34 Pantallas con las fichas de detalle. Película, Director, Productor, Protagonista.

3.6 Diagrama de Jerarquía de clases.

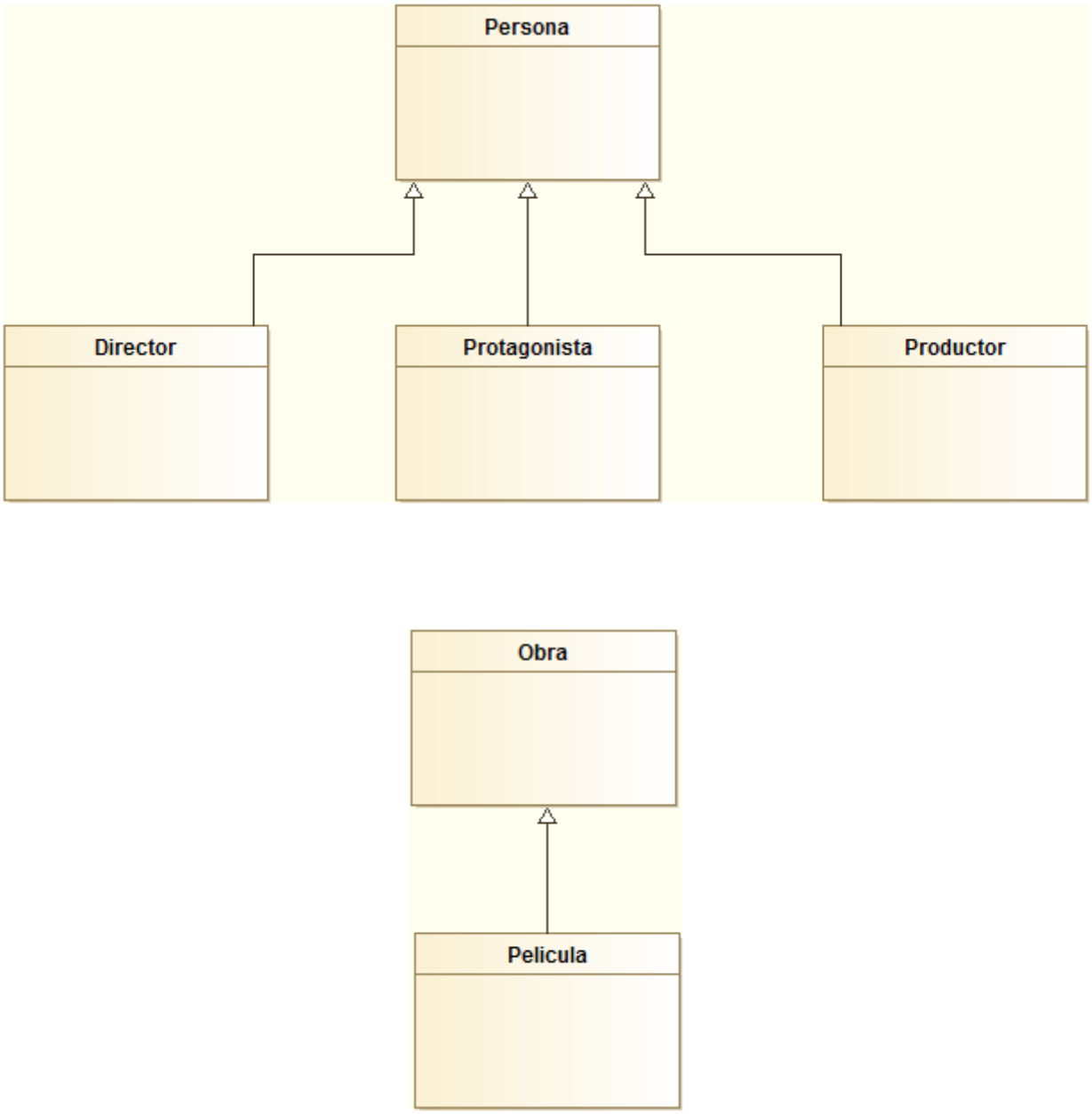


Figura 35 Diagrama de jerarquía de clases.

3.7 Diagrama de clases.

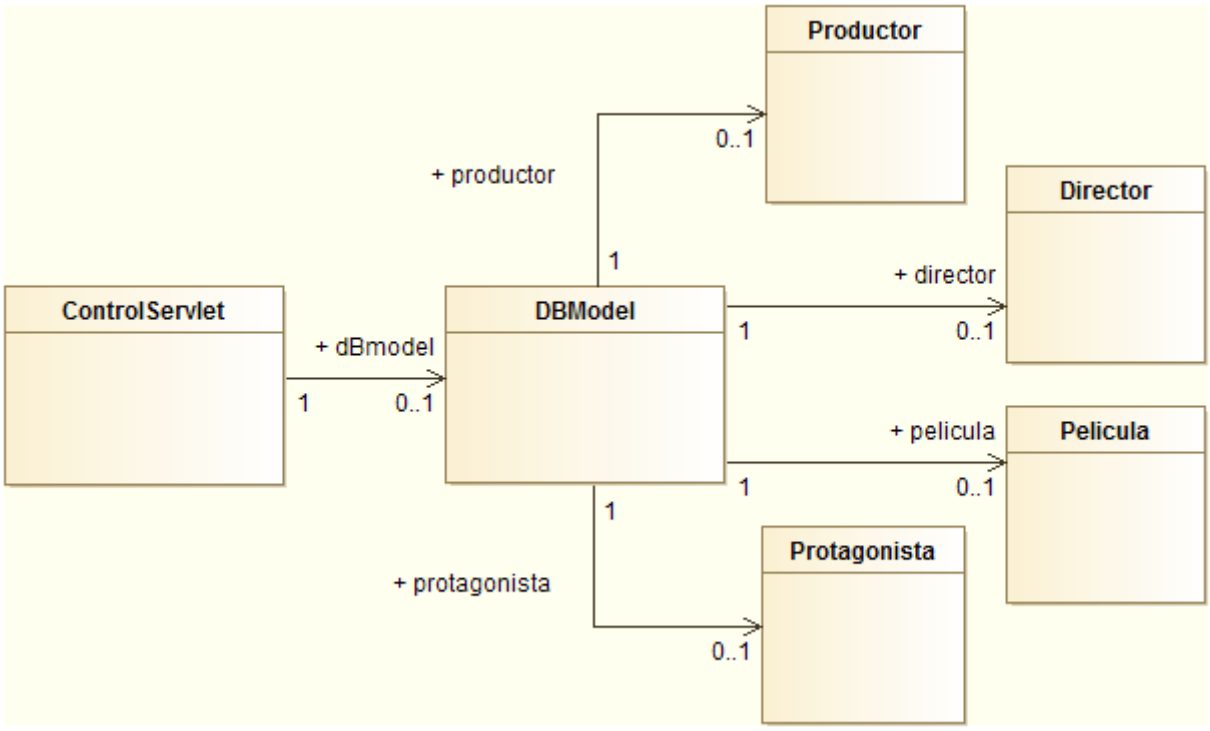


Figura 36 Diagrama de clases.

3.8 Diagramas de secuencia.

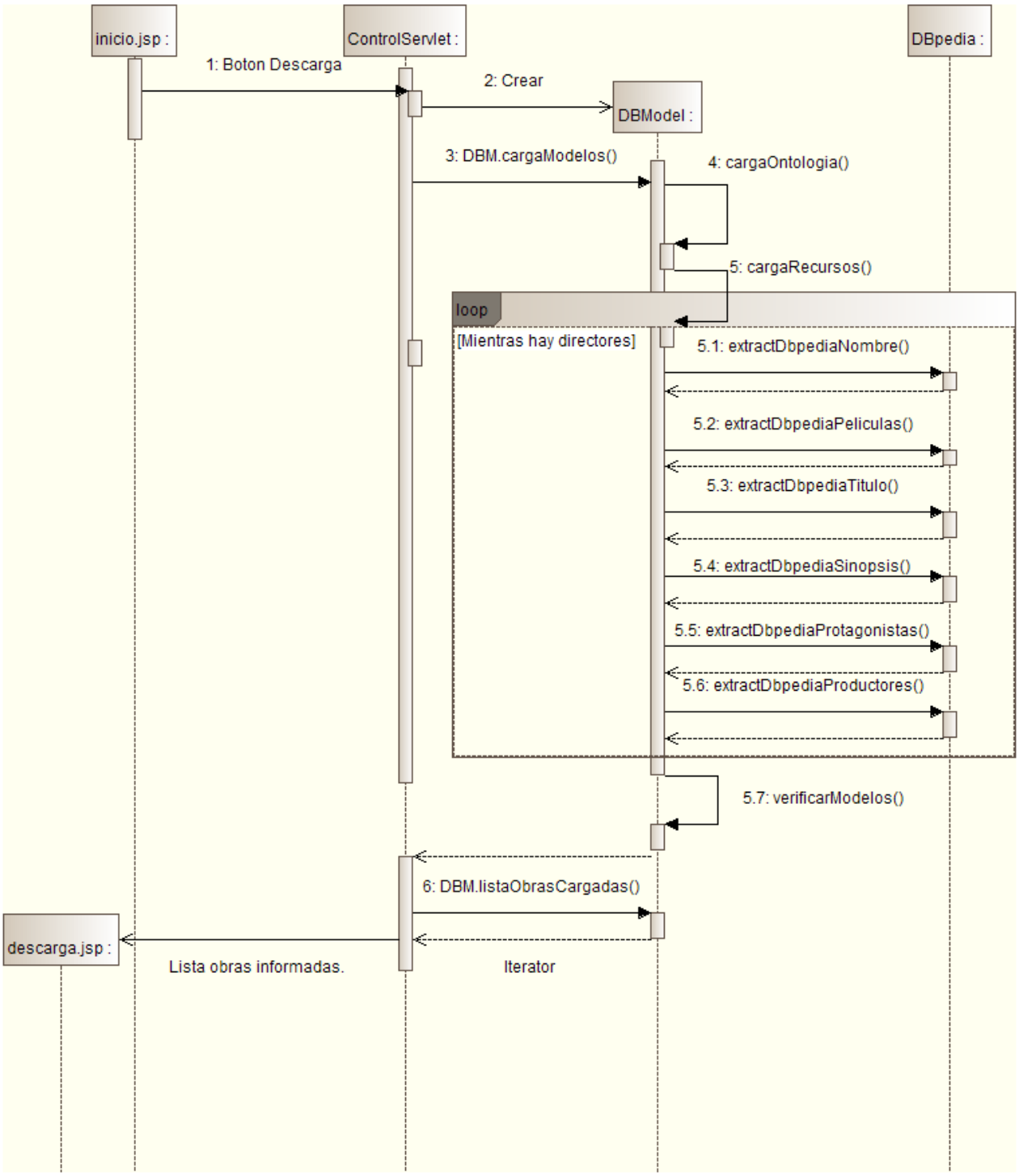


Figura 37 Diagrama de secuencia correspondiente al prototipo de extracción de datos y creación de instancias.

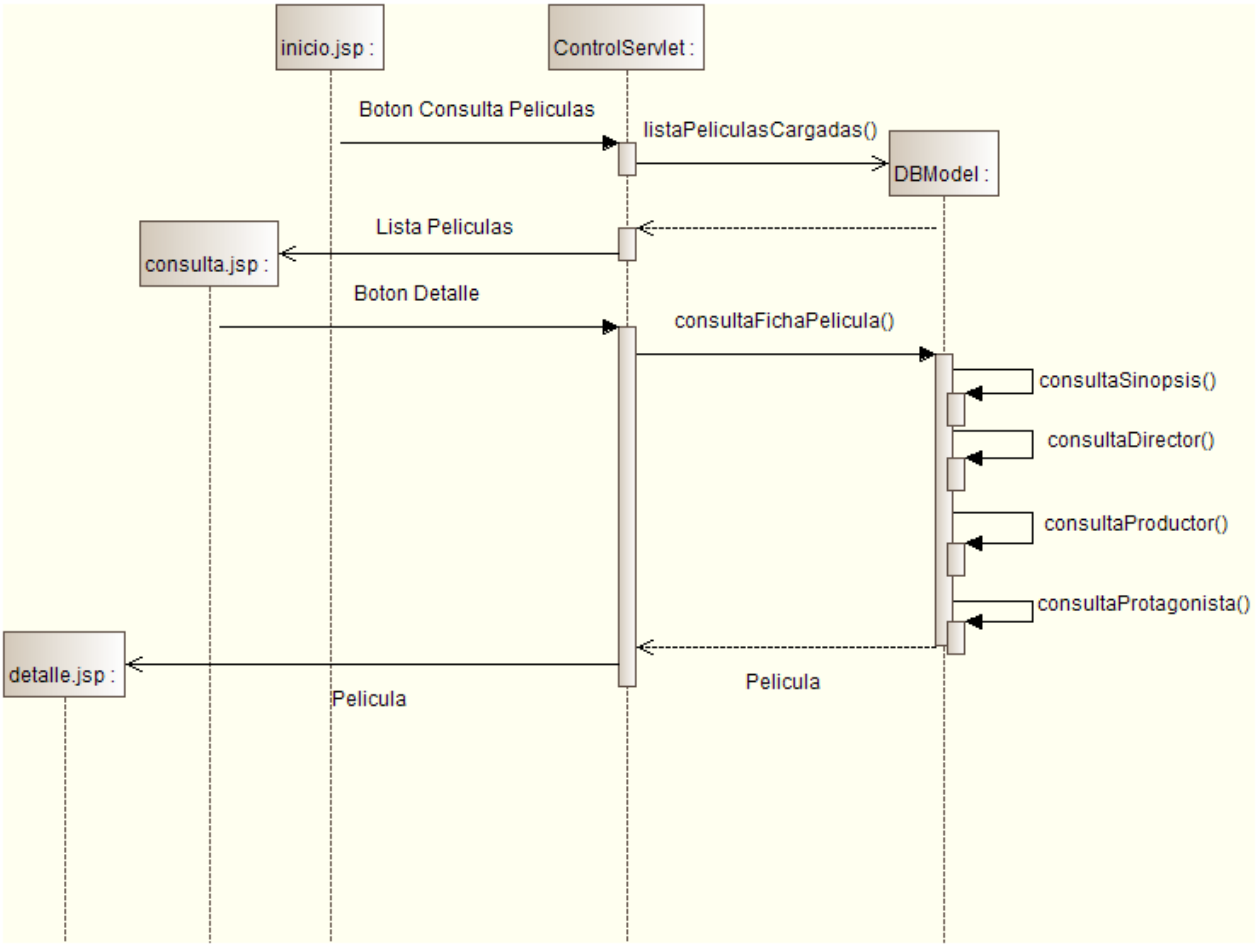


Figura 38 Diagrama secuencia consulta ficha película.

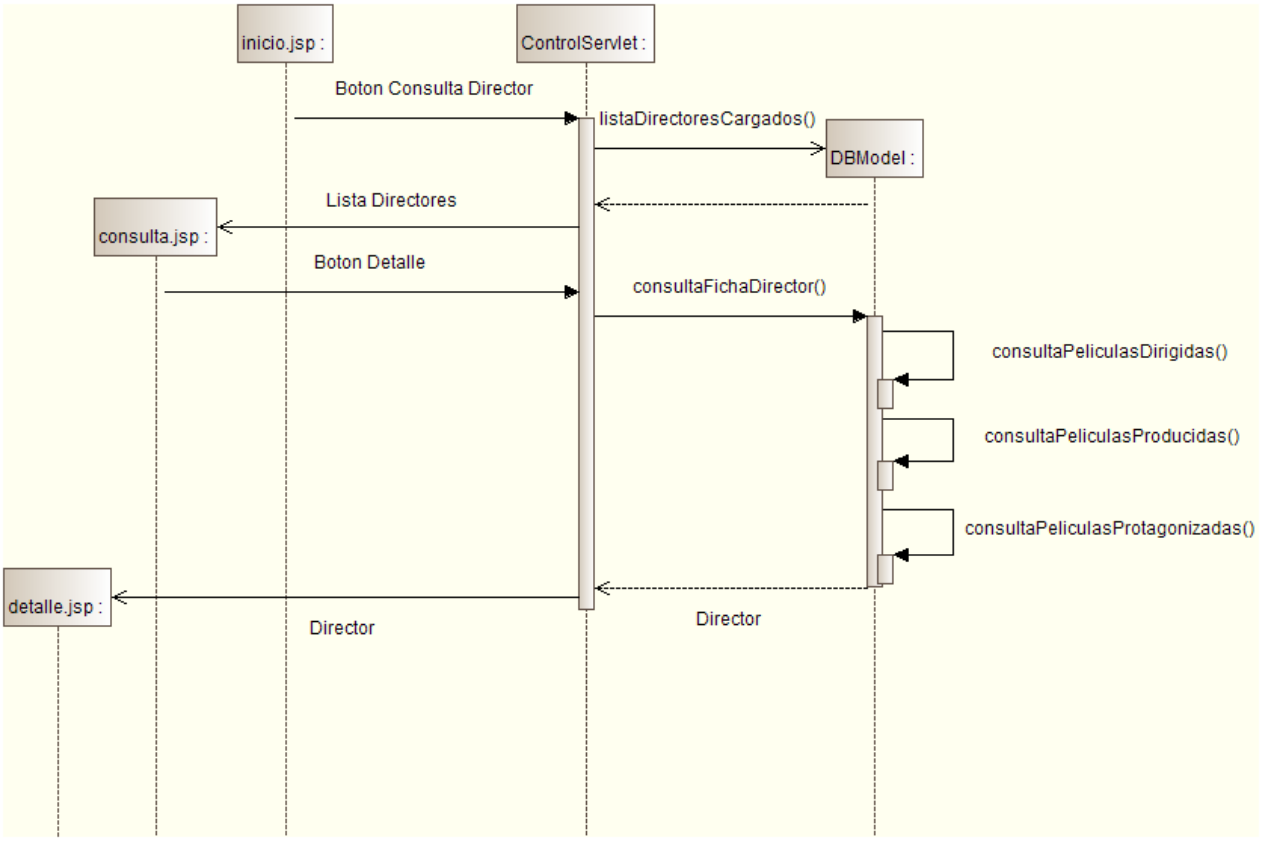


Figura 39 Diagrama secuencia consulta ficha director.

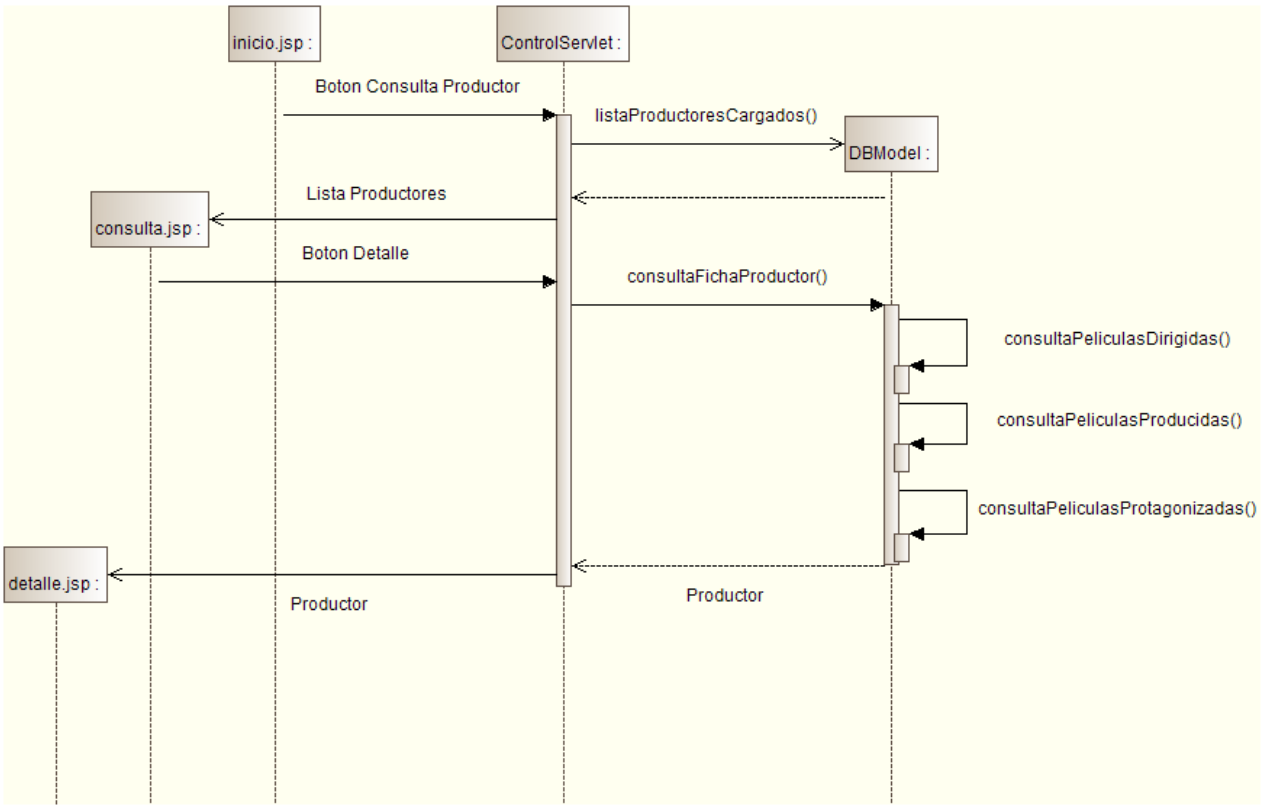


Figura 40 Diagrama de secuencia consulta ficha productor.

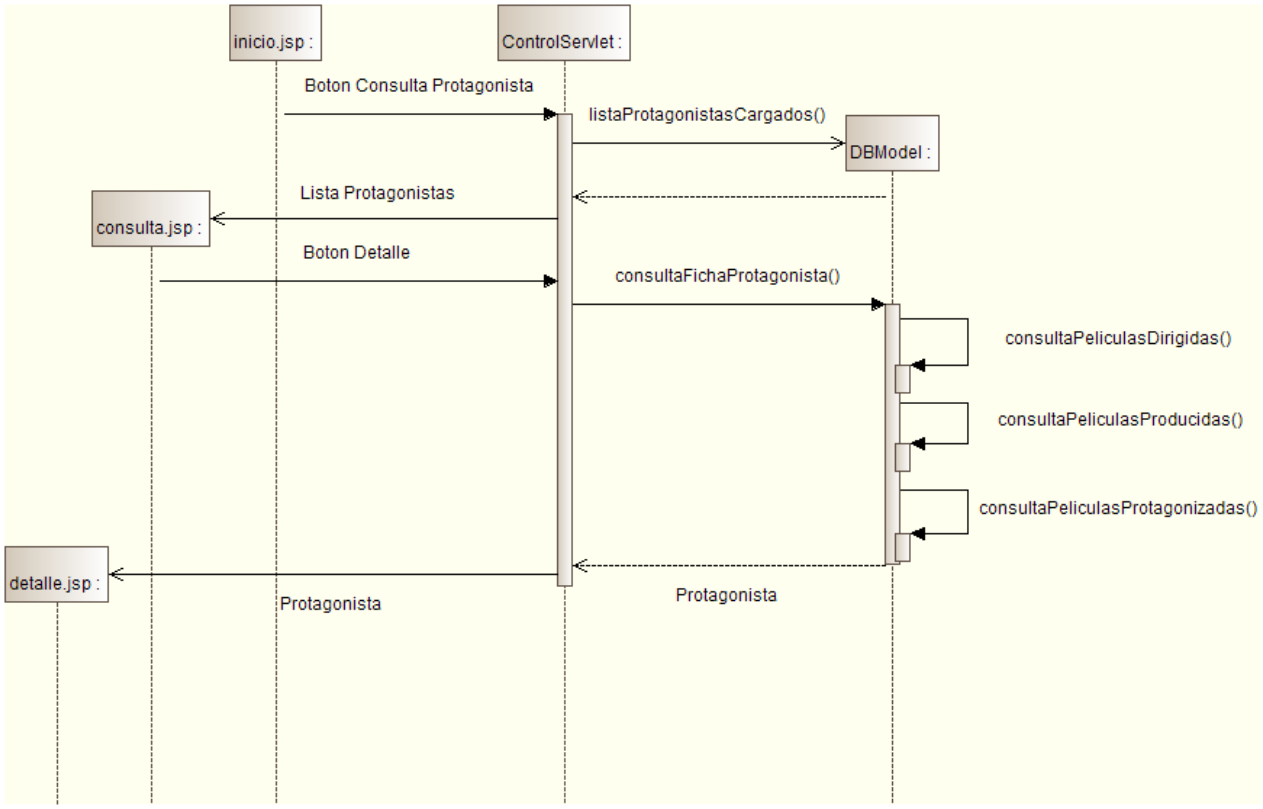


Figura 41 Diagrama de secuencia consulta ficha protagonista.

3.9 Arquitectura de la aplicación.

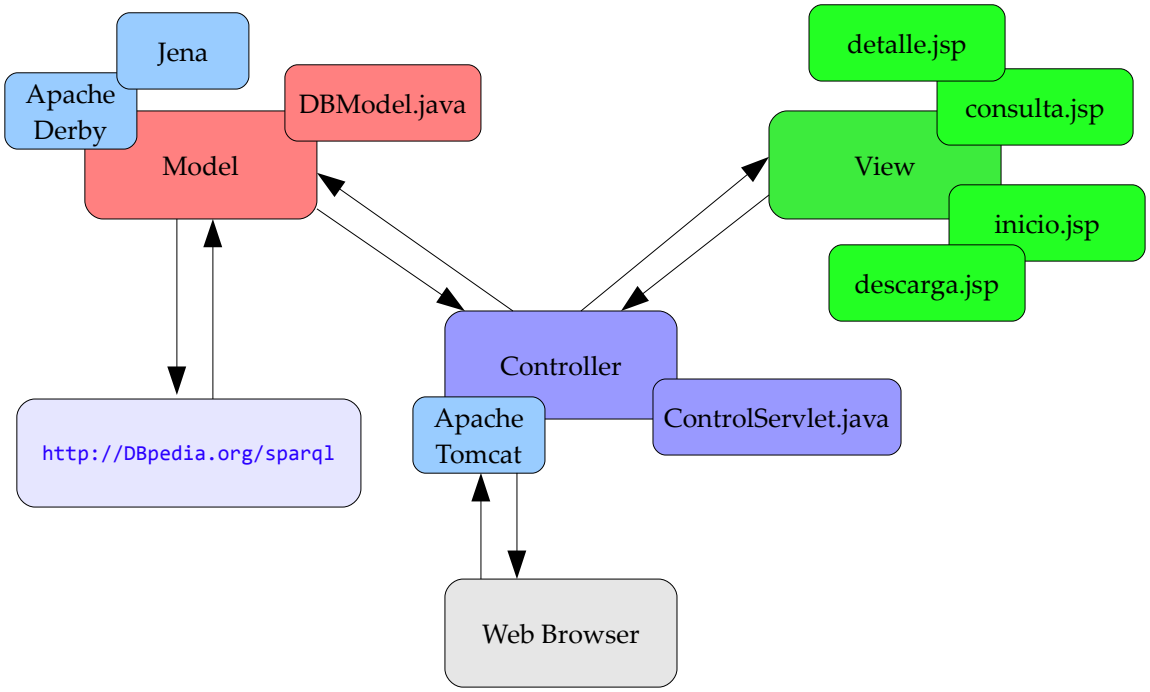


Figura 42 Arquitectura de los prototipos.

La arquitectura de los prototipos se basa en el modelo MVC. Este modelo separa los datos de la aplicación, la interfaz de usuario y la lógica de negocio en tres componentes distintos.

Model – Esta es la representación específica de la información con la que trabaja la aplicación. No hemos definido modelo de datos de usuario. Usamos el modelo de Jena que es transparente para el usuario y el desarrollador. Las tablas en el gestor de bases de datos se crean de forma automática:

```
ModelMaker maker = null
maker = ModelFactory.createModelRDBMaker(conn);
….
m = maker.createModel(MODEL_NAME);
…..
RDirector = m.createResource(RESOURCE_URI + director);
```

El método *createModelRDBMaker* de la clase `ModelFactory` nos devuelve un objeto de la clase `ModelMaker` , este es el que maneja los modelos RDF y los convierte en persistentes en la base de datos sin que tengamos que realizar ninguna operación SQL contra ella. Cuando definimos RDirector en el ejemplo (en el prototipo se hace así) se realiza una operación de entrada salida contra la base de datos para grabar el recurso en ella. Esta es la gran ventaja de utilizar la persistencia de Jena en bases de datos.

En el prototipo, la clase **DBModel.java** es la que desempeña esta función.

View – Este componente se encarga de presentar el modelo en un formato adecuado para interactuar con él. Es la interfaz de usuario. En el prototipo la función la desempeñan cuatro JSP.

inicio.jsp – Presenta la pantalla principal.

descarga.jsp – Presenta el informe resultado de la extracción de datos de DBpedia.

consulta.jsp – Presenta el listado de Películas, Directores, Productores y Protagonistas, desde esta pantalla se puede consultar la ficha de detalle.

detalle.jsp – Presenta la ficha de una Película, un Director, un Productor o un Protagonista.

Controller – Este componente responde a eventos, las acciones del usuario en las cuatro pantallas anteriores, haciendo peticiones al modelo y manejando el flujo de la interacción con el usuario.

En el prototipo, la clase **ControlServlet.java** es la que desempeña esta función.

4. Conclusiones.

Las técnicas de Semántica aplicadas a la Web permiten que la información esté mejor definida. En una Web con mas significado encontraremos soluciones a los problemas mas habituales de búsqueda de información, a medida que vayamos utilizando una infraestructura común, mediante la cual podamos compartir, procesar y transferir información de forma rápida y sencilla.

Es evidente que la Web ha transformado nuestra forma de trabajar y de comunicarnos, la economía a nivel global permitiéndonos realizar todo tipo de transacciones a cualquier hora del día y de la semana. Nunca hemos tenido acceso a tantos recursos y a tanta información ni tampoco tantas facilidades para generarla. Una vez mas nos encontramos con un ejemplo muy común en nuestros días, el éxito se ha convertido en su

principal problema. El exceso de información y su heterogeneidad, tanto en contenido como en sus características, y el fuerte crecimiento diario, están generando un grave problema interoperabilidad., Se necesitan soluciones que permitan a los usuarios de la Web delegar funciones y tareas en el software.

Como hemos visto en este trabajo, de una forma sencilla, con las técnicas de semántica podemos resolver estos problemas. Con el software que hemos desarrollado, la ontología y los prototipos, podemos procesar el contenido de la Web, razonar con él y realizar deducciones lógicas para obtener resultados precisos a partir de la información y los datos que tenemos.

En el futuro, sería deseable que la información no sólo fuera tratada como datos de entrada y salida de las aplicaciones, sino en términos de su significado. Para ello, hoy en día, los metadatos ya juegan un papel muy importante, están abriendo el camino, no para que la Web sea aun mas inteligente, sino para que los propios datos que están almacenados en ella sean mas inteligentes.

Para obtener estos "datos inteligentes", en este proyecto estamos utilizando RDF, RDFS y SPARQL. Con estas tecnologías es posible que se pueda convertir la Web en una infraestructura global en la que compartir información y reutilizar datos y documentos entre los usuarios se realice de forma mas eficiente. Es decir, facilitarán el desarrollo de muchas de las posibles aplicaciones que tiene la Web.

Uno de los principales problemas es la ausencia de contenido semántico y esto es debido a la ausencia de métodos de anotación semiautomáticos.

Como conclusión, generar este tipo de contenido es bastante difícil, y parece tarea imposible abordar un proyecto de estas características a gran escala. No parece que con este tipo de sistemas podamos transformar de golpe el contenido de la Web actual.

Si que parece mas posible, que esta tecnología, estratifique aun mas la Web actual. Así, es posible que se creen distintas capas con diferentes grados de "inteligencia de datos".

Desde mi punto de vista, y a medida que evolucione la tecnología de semántica, lo que veremos en un futuro próximo es el fruto de muchas iniciativas individuales y en sectores importantes como el financiero y el industrial, a menor escala, impulsadas por un fuerte impacto en ahorro de costes sobre todo en mantenimiento de aplicaciones.

5. Instalación de la aplicación.

Se adjunta un fichero para desplegar en el servidor de aplicaciones, sfernandezve_PEC3.war. La base de datos[27] no es necesario definirla, ya que se crea desde la propia aplicación en un directorio que se llama C:\DevApp\triplestore\. Tampoco es necesario arrancar ninguna instancia del gestor de bases de datos ya que éste está embebido dentro de la aplicación. La estructura de tablas de Jena se crea, de forma automática, al utilizar la primera vez la aplicación dentro del esquema APP de Derby, ya que no usamos usuario al crear la base de datos.

El fichero que contiene la ontología, cine.rdfs, se encuentra en el directorio WEB-INF/classes/ y se lee por medio del class loader. Este fichero se almacena de forma persistente por medio de Jena en Derby.

Para iniciar la aplicación, una vez desplegada en Tomcat, y suponiendo que usa el puerto 8080, debemos poner en el navegador:

http:\\localhost:8080\sfernandezve_PEC3\inicio.jsp

Acto seguido nos aparecerá una pantalla en la que se nos permite realizar descarga de información o consulta. La primera vez que se use la aplicación hay que realizar una descarga. La disponibilidad de datos,

27 http://db.apache.org/derby/integrate/plugin_help/derby_app.html#About+Schema+Names

dependerá de la disponibilidad del sitio Web de DBpedia y la velocidad del enlace del usuario.

La descarga de información obtiene cinco películas de cada uno de los siete directores definidos, (35 películas en total) usando el servicio SPARQL EndPoint de DBpedia. Hay que tener un poco de paciencia, ya que tarda un poco en descargar los datos.

Una vez descargados los datos podemos realizar consultas o nuevas descargas. Una nueva descarga reemplazará el contenido de la base de datos, pero como no variamos la lista de directores, la información que descarguemos será prácticamente la misma.

6. Tecnologías utilizadas en el desarrollo e instalación del software.

A continuación, y brevemente, se describe el proceso de instalación de las distintas tecnologías necesarias para poner en marcha el proyecto, y lo mas importante, cómo debemos proceder, paso a paso, para que todo funcione correctamente.

6.1 Instalación Eclipse.

Descargamos el software de:

http://www.eclipse.org

Una vez descargado obtendremos un fichero con este nombre:

eclipse-jee-indigo-SR1-win32.zip

Descomprimimos el fichero en una carpeta. No es necesario ninguna acción adicional.

6.2 Instalación de Jena.

Descargamos el software de:

http://jena.sourceforge.net

Una vez descargado obtendremos un fichero con este nombre:

jena-2.6.4.zip

Descomprimimos el fichero en una carpeta y seguimos las instrucciones de instalación que se facilitan en el fichero readme.html.

Definimos una variable de entorno **JENAROOT** cuyo valor sea la ruta del directorio en el que hemos descomprimido el producto y una vez realizada esta operación procedemos a definir en la variable CLASSPATH las librerías JAR incluidas en el directorio %JENAROOT%\lib. Finalizado este proceso, adaptamos eclipse para utilizar Jena. Abrimos eclipse, y desde la pestaña “Window” elegimos la opción “preferences”. Desde aquí desplegamos la opción “Java”, “Build Path”, “User Libraries”, pulsamos sobre el botón “New”, y en la pantalla en la que nos pide el nombre de la librería de usuario indicamos “jena”, pulsamos el botón “Ok”. Ahora, seleccionamos la librería de usuario que acabamos de crear (“jena”) y pulsamos el botón “Add JARs...”, navegamos hasta el directorio %JENAROOT%\lib y seleccionamos todos

los JAR contenidos en él. Para que la librería sea útil en nuestros proyectos eclipse, debemos incluirla como librería de usuario. Para ello, pulsamos con el botón derecho del ratón sobre el proyecto y elegimos la opción "properties" y en el siguiente menú seleccionamos la opción "Java Build Path", después seleccionamos la pestaña "Libraries" , pulsamos sobre el botón "Add Library...", seleccionamos la opción "User Library", botón "Next", y marcamos la casilla en la que aparece "jena", que es la librería de usuario que acabamos de crear en el paso anterior. A partir de este momento, ya podemos usar Jena en nuestro proyecto.

Para tener disponible Jena en nuestra aplicación web desplegada en Tomcat, copiaremos los ficheros "jar" contenidos en la carpeta "lib" de instalación del producto en la carpeta " WEB-INF/lib" de nuestro proyecto eclipse.

6.3 Instalación JDK.

Descargamos el software de:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Elegimos Java SE 6 Update 29, descargamos el fichero:

jdk-6u29-windows-i586.exe

Ejecutamos el programa y una vez finalizada la instalación ponemos en la variable PATH:

C:\Program Files\Java\jdk1.6.0_29\bin

6.4 Instalación Apache Tomcat

Descargamos el software de:

http://tomcat.apache.org

Una vez descargado obtenemos un fichero con este nombre:

apache-tomcat-6.0.33-windows-x86.zip

Descomprimimos el fichero en una carpeta y seguimos las instrucciones de instalación que se facilitan en el fichero RUNNING.txt.

Definimos la variable JAVA_HOME y le asignamos el pathname de la instalación del JDK.

JAVA_HOME=C:\Program Files\Java\jdk1.6.0_29

Arrancamos TOMCAT:

$CATALINA_HOME\bin\startup.bat

Comprobamos el funcionamiento desde el navegador.

http://localhost:8080/

Nos aparece la pantalla con las aplicaciones Web por defecto de TOMCAT.

Paramos TOMCAT:

$CATALINA_HOME\bin\startup.bat

Procedemos ahora a configurar Eclipse para que funcione con TOMCAT.

Entramos en Window → Preferences → Server → Runtime Environments, pulsamos el botón Add, elegimos

de la lista “Apache Tomcat v6.0”, marcamos la casilla “Create a new local server” y en la pantalla siguiente el path de instalación de Tomcat. Ahora pulsamos el botón “Installed JREs...”, pulsamos el botón “Add”, seleccionamos “Standard VM”, pulsamos el botón “Next”, en la pantalla que nos aparece indicamos el path de instalación del JDK que acabamos de instalar. Una vez finalizado, elegimos este JRE en la pantalla de configuración de Tomcat y pulsamos el botón “Finish”.

Para poder probar las aplicaciones las desplegaremos usando un fichero con extensión WAR generado a partir del proyecto eclipse. Debemos pulsar sobre e proyecto con el botón derecho del ratón y elegir “Export” , en la pantalla siguiente seleccionar “Web” y “WAR file”.

La función de despliegue se realiza desde la pantalla de aplicaciones Web por defecto de TOMCAT en http://localhost:8080/ . Seleccionamos “Tomcat Manager” en la sección “Administration” de la pantalla. Nos aparece un menú pidiendo un usuario y una contraseña. Estos, de deben configurar en el fichero:

$CATALINA_HOME\conf\tomcat-users

Incluyendo en él estas sentencias:

<role rolename="manager-gui"/>

<user username="**admin**" password="**admin**" roles="manager-gui"/>

En las que indicamos que el usuario administrador es “admin” y su sontraseña “admin”. Una vez identificados nos aparecerá la pantalla del Gestor de Aplicaciones Web de Tomcat. Indicamos en “Seleccione archivo WAR a cargar” la ruta completa del fichero WAR, por ejemplo C:\DevApp\sfernandezve_PEC3.war y pulsamos el botón desplegar.

6.5 Instalación Apache Derby.

Descargamos el software de:

http://db.apache.org/derby/

Una vez descargado obtenemos un fichero con este nombre:

db-derby-10.8.2.2-bin.zip

derby_core_plugin_10.8.2.zip

derby_ui_doc_plugin_1.1.3.zip

Procedemos de la siguiente forma.

Descomprimimos el fichero db-derby-10.8.2.2-bin.zip en una carpeta. Configuramos la variable de entorno DERBY_INSTALL con el path del directorio que hemos creado. Añadimos a la variable de entorno CLASS_PATH %DERBY_INSTALL%/lib/derby.jar y %DERBY_INSTALL%/lib/derbytools.jar

Verificamos la instalación de Derby con este comando “java org.apache.derby.tools.sysinfo”.

Descomprimimos los otros dos ficheros en la carpeta en la que tenemos instalado Eclipse. Paramos Eclipse, si lo tenemos arrancado, y lo volvemos a arrancar. Los ficheros quedan en el directorio de plugins de Eclipse.

Apache Derby, lo vamos a utilizar en modo embebido, es decir, se ejecutará en la misma JVM que nuestra aplicación, no permitiendo el acceso concurrente de varios usuarios al mismo tiempo. De esta forma se arranca y se para con la propia aplicación y no requerirá de mantenimiento.

Ahora configuramos eclipse para utilizar Derby. Desde la opción Window → Show View → Other..., desplegamos la opción “Data Management”, seleccionamos “Data Source Explorer” y pulsamos el botón

"OK". Repetimos el procedimiento y seleccionamos la opción "SQL Results". Desde la pestaña "Data Source Explorer" posicionamos el ratón sobre "Database Connections", pulsamos el botón derecho y en el menú seleccionamos la opción "New". Nos aparece la pantalla "Connection Profile". Seleccionamos Derby, pulsamos el botón "Next", en la siguiente pantalla seleccionamos del desplegable Drivers el que se llama "Derby Embedded JDBC Driver 10.1 Default" y en la pestaña "General" ponemos como "Database location" el path donde se ubicará la base de datos y su nombre, elegimos como nombre de la base de datos "triplestore", dejamos en blanco los parámetros "User name" y "Password" y activamos la casilla "Create database" para que se cree la base de datos en caso de que no exista. Pulsamos el botón "Test connection", nos aparecerá una pantalla con el mensaje "Ping succeeded". Comprobamos a continuación que la carpeta "triplestore" se ha creado en el directorio que hemos indicado. Pulsamos el botón "Finish" y la conexión queda creada.

Añadimos Apache Derby a nuestro proyecto. Sólo copiamos al directorio WEB-INF/lib el fichero derby.jar.

A partir de este momento ya podemos trabajar con la base de datos.

6.6 Instalación protégé OWL editor 4.1

Descargamos el software de:

http://protege.stanford.edu/

Una vez descargado obtenemos un fichero con este nombre:

protege-4.1.239.zip

Procedemos de la siguiente forma.

Descomprimimos el fichero protege-4.1.239.zip en una carpeta. Ejecutamos run.bat de la carpeta anterior y ya tenemos en funcionamiento el producto. Es open source.

Es muy útil para aprender cómo se crean las ontologías. Podemos seguir el tutorial que se encuentra en:

http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/

También lo podemos utilizar para cargar en el los ficheros que se adjuntan, cine.rdfs, cine.rdf y cine_resultado_modelo_inferencia.txt. Podemos ver las clases, las propiedades y las instancias. Además es una forma fácil de comprender el tipado de instancias y como funciona un reasoner.

6.7 Instalación de Modelio.

Descargamos el software de:

http://www.modelio.org/

Una vez descargado obtenemos un fichero con este nombre:

modelio-2011-12-06-win32.win32.x86

Lo descomprimimos en una carpeta.

No necesita instalación. Ejecutamos el programa modelio.exe desde la carpeta en la que hemos creado.

Con este programa podemos crear todo tipo de diagramas UML. También es open source y está bastante bien.

6.8 Instalación de Nvu.

Descargamos el software de:

http://net2.com/nvu/

Una vez descargado obtenemos un fichero con este nombre:

nvu-1.0-win32-full.zip

Procedemos de la siguiente forma.

Descomprimimos el fichero nvu-1.0-win32-full.zip en una carpeta. Ejecutamos nvu.exe de la carpeta anterior y ya tenemos en funcionamiento el producto.

Con este software podemos diseñar, de una forma muy sencilla, páginas HTML. El código generado lo podemos "trasladar" a un JSP. Es open source.

7. Bibliografía.

- Liyang Yu. A Developer's Guide to the Semantic Web. Springer.

- Vipul Kashyap., et al. (2004): Information Modeling on the Web: The Role of Metadata, Semantics, and Ontologies in The Practical Handbook of Internet Computing. Munindar. Edited by P. Singh. Chapman Hall and CRC Press, Baton Rouge. 2004.

- Antoniou G., van Harmelen F. (2004). A semantic web primer. The MIT Press.

- Naci Dai., et al. eclipse Web Tools Platform, Developing Java Web Applications. Pearson Education. 2007.

- Matthew Horridge. A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools Edition 1.3. The University Of Manchester. 2011.

- Dean Allemang; James Hendler (2007). Semantic Web for the Working Ontologist: Morgan Kaufman Publishers.

- Learning SPARQL Querying and Updating with SPARQL 1.1 by Bob DuCharme O'REILLY

8. Enlaces de interés.

http://www.w3.org/TR/2008/REC-xml-20081126/#sec-xml-and-sgml

Resource Description Framework (RDF) model and syntax specification, a W3CRecommendation, 22 February 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

http://www.w3.org/TR/2006/REC-xml-names-20060816/

http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/#higherorder

http://www.w3.org/TR/rdf-syntax-grammar/

http://www.w3.org/TR/rdf-schema/

http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html

http://www.w3.org/TR/2009/REC-xmlbase-20090128/

http://www.w3.org/TR/rdf-sparql-query/

http://www.joseki.org

http://virtuoso.openlinksw.com

http://dbpedia.org/snorql/

http://dbpedia.org/About

http://www.swib09.de/vortraege/20091124_jentzsch.pdf

http://www.hpl.hp.com/

http://www.apache.org/

http://incubator.apache.org/jena/about_jena/architecture.html

http://openjena.org/wiki/Fuseki

http://www.w3.org/TR/sparql11-update/

http://db.apache.org/derby/integrate/plugin_help/derby_app.html#About+Schema+Names

http://www.eclipse.org

http://jena.sourceforge.net

http://www.oracle.com/technetwork/java/javase/downloads/index.html

http://tomcat.apache.org

http://db.apache.org/derby/

http://protege.stanford.edu/

http://www.modelio.org/

http://net2.com/nvu/