Semantic Web: Project documentation

Marc Kirchmann and Sebastian Leitz

Cooperative State University Baden-Wuerttemberg
Master Computer Science
minf010@cas.dhbw.de, minf11@cas.dhbw.de

Abstract. We will create a web interface that enables users to list and find data from a set of films and associated information. This data will be stored within a server that the interface communicates with.

One part of the project will be to find suitable data from public sources and store this in a format that can be understood by software such as Jena. We will configure this software in an appropriate way so that it is able to read the stored data and provide this to other systems.

In a second step we will create and design a small web application, that uses an API of the software from step one to fetch data from the server. It will be possible to conduct simple search operations in that application which in turn communicates with the server API in order to find matching data.

1 Introduction

This document describes the authors' efforts to fulfill the tasks given in the assignment on the "Semantic Web" lecture. The assignment requests to create any kind of ontology using RDF and make this available for reading and querying to users via a website.

2 Structure and Usage of the Movie Ontology

Our semantic search webpage should enable a user to search for movies or other associated artifacts, e.g. a particular genre or a film's most famous actors. Thus an ontology is required which puts all of those things in relation. We decided to adopt an existing ontology, called "MO - the Movie Ontology" [1]. It makes use of OWL to map movie entities to classes and defines class hierarchies as well as predicates that show their relationship. To describe its elements, it uses other well-known ontologies, e.g. from "DBpedia" [2]. Although the used ontology provides a lot of entities, only a subset of them is used for the webpage. The following sections introduce the entities that can be searched and their associated predicates. For the sake of readability, the prefix of a complete URI is omitted.

2.1 Movie

A movie is represented as an OWL class. It is the central element of the ontology. Listing 1.1 shows how it is defined in an RDF file that uses RDF/XML notation.

Listing 1.1. OWL Movie Class in RDF/XML notation

```
<owl:Class rdf:about="&www;Movie"/>.
```

Most of the other parts of the ontology have a direct or indirect connection to it, i.e. a predicate describing their relationship. An example is given in listing 1.2. As a movie usually belongs to one or more genres, this is represented by the predicate belongsToGenre. The range and domain entries define the OWL classes that are used as subject (domain $\rightarrow$ a movie) or object (range $\rightarrow$ a genre) in a statement that uses belongsToGenre as its predicate.

Listing 1.2. Exemplary Movie predicate in RDF/XML notation

```
<owl:ObjectProperty rdf:about="&movieontology;belongsToGenre">
        <rdfs:range rdf:resource="&movieontology;Genre"/>
        <rdfs:domain rdf:resource="&www;Movie"/>
    </owl:ObjectProperty>
```

2.2 Actor

Another important part of the ontology is the OWL class for actors, shown in listing 1.3. It is a base class of Person, which itself is part of DBpedia's ontology. That means every Actor implicitly must be a Person and inherits the properties of a Person, too.

Listing 1.3. OWL Actor Class in RDF/XML notation

```
<owl:Class rdf:about="&ontology;Actor">
        <rdfs:subClassOf rdf:resource="&ontology;Person"/>
</owl:Class>
```

An actor or actress plays in a movie. This is what the predicate in listing 1.4 is about. It defines the relationship hasActor between a Movie and an Actor and its reverse definition:

- Movie hasActor Actor.
- Actor isActorIn Movie.

Listing 1.4. Exemplary Actor predicate in RDF/XML notation

```
<owl:ObjectProperty rdf:about="&movieontology;hasActor">
        <rdfs:range rdf:resource="&ontology;Actor"/>
        <owl:inverseOf rdf:resource="&movieontology;isActorIn"/>
        <rdfs:domain rdf:resource="&www;Movie"/>
</owl:ObjectProperty>
```

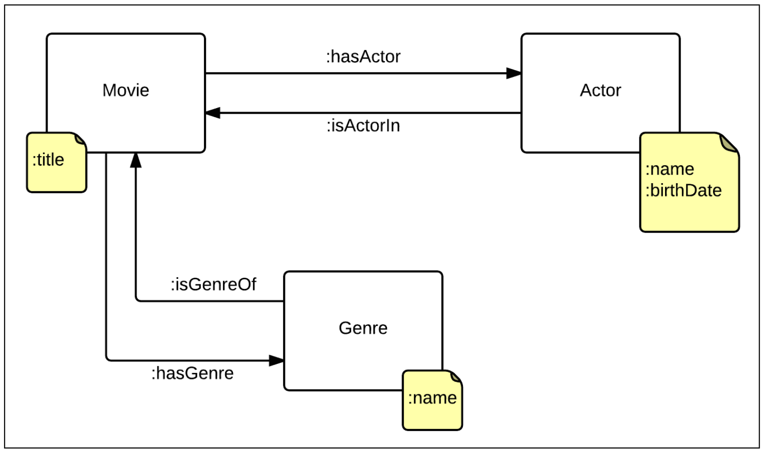

Fig. 1. Movie Ontology Overview

3 Data Storage Setup

4 Semantic Movie Search Website

Part of the given task was to make all the data described in the two previous sections available for user interaction via a website.

We started by defining the relevant functionality of the website in order to build an HTML prototype:

1. Search field for user input
2. Button to send the query to the server
3. A subsection to display a list of results
4. If required, a view giving details on a single result

The websites involves a certain amount of dynamic information, so we decided to use a small server-sided PHP script to take care of any logical decisions. We also integrated a library for SPARQL queries in order to simplify our script. The library acts as an abstraction layer and takes care of many small items such as connecting to the server, checking for errors as well as evaluating and transforming the response from the server. **citation needed**

4.1 The Search Interface

We constructed a simple markup which includes all items from above listing with HTML 5 and some basic CSS styles. HTML 5 takes care of a placeholder text within the search field to guide the user and also makes sure no empty queries are sent (required form field).

Each time the user makes a request to the website, a PHP script is called to action by the webserver. The script evaluates the current URI and the determines whether a query has been submitted by the user. This is decided based on the presence of the *query* variable.

If the query variable is not present, the script outputs the static HTML content of the search interface with an appropriate HTTP Content-type header.

Otherwise, the value of the query variable is extracted from the URI and used to fetch data from the server. In a first step, the user input is transformed into a SPARQL query, which is then evaluated by the Fuseki server.

4.2 Query Construction

4.3 Response Evaluation

The query constructed by the above step is sent to the Fuseki server using the SPARQL abstraction layer as well.

The server returns a SPARQL result within an XML document. The abstraction layer parses this XML structure and inserts an XSL file reference. Our PHP script then sends the resulting new XML structure to the user's browser with an HTTP Content-type of *application/xml*.

5 Conclusion

[1]

References

1. Bouza, A.: Mo the movie ontology (2010) [Online; 26. Jan. 2010].
2. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. Semantic Web Journal (2014)