

# Enriching Wolff Bioscopen Website Using Linked Data from DBpedia

Semantic Web Project

MA Wicaksana (s1507850)  
Iwan Timmer (s0201170)  
KKT Chandrasekar (s1736620)

February 1, 2016

## Abstract

We developed an improved version of Wolff Bioscopen website which uses Linked Data from DBpedia to provide non-trivial dynamic information which cannot be found in search engine easily. To achieve this, we developed a movie ontology and enriched original Wolff movie data with additional data from DBpedia by using the self-developed ontology as the reference. We also report challenges and we faced during developing the application, such as dealing with data ambiguity, finding appropriate Linked Data providers to interlink with our application, making decision whether to create self-developed ontology or reuse the existing one, and the limitation from the tools we used.

## 1 Introduction

Wolff Bioscopen is one of the major cinema chain in the Netherlands. Its official website[18] is mostly used for providing information about film listings and for ticket reservations, as can be seen in Figure 1. The screenshot shows us brief information about the film: the title, short description, list of actors playing in it, duration time, the director, genre, and so on.

However, the information provided feels quite limited. There is also no information about other people who have important role in a movie, such as the producers, the writers, the cinematographers, the composers, and so on. It would also interesting to have other additional information, such as how much budget was allocated for this movie, where was the shooting location, etc. Additionally, it also feels static and separated. People listed there is just a collection of names. There is no internal links providing concise information regarding their personal background and previous works (only some well-known actors are linked towards their personal website). The film page in Figure 1 shows Bruce Dern as one of the actor in *'The Hateful Eight'*, but it cannot dynamically shows all movies he starred previously. It also cannot shows what kind of professional relationship they have. For example, what movies did Bruce Dern and Samuel L. Jackson played together in the past? who are the ones whom Samuel L. Jackson like to work together with?

In short, Wolff website has high potential that has not fully reached yet. By providing such kinds of dynamic information, Wolff website would become more informative and more interesting, which in turn, could attract more visitors and potentially increase their revenue through ticket sales.

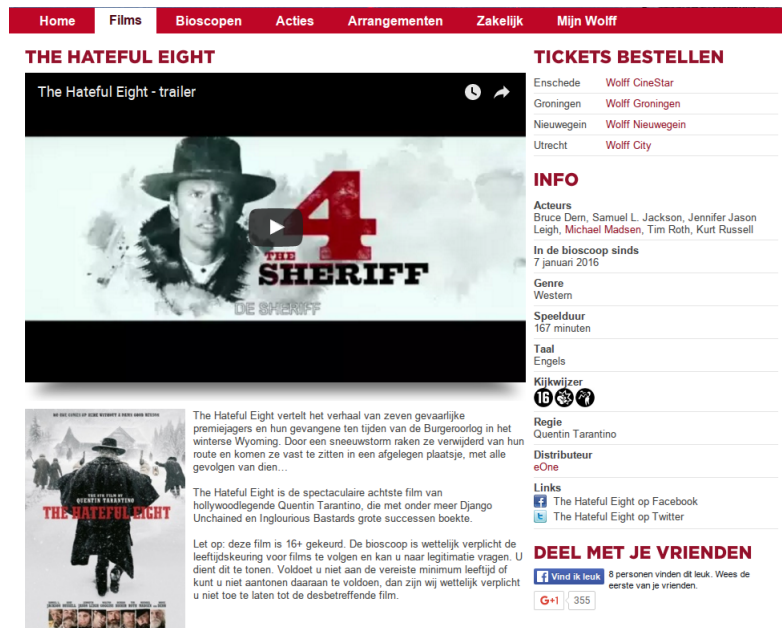


Figure 1: Example of one of film page in Wolff website

However, providing those additional data could be cumbersome. We should constantly track all updates of works and people in film industries. Doing it manually is practically impossible, and it is not the main task of movie theater website to provide that. One promising solution is to use Linked Data. As described in [2], Linked Data refers to data published on the Web in such a way that it is machine-readable, its meaning is explicitly defined, it is linked to other external datasets, and can in turn be linked from external datasets. Using Linked Data, we can connect existing data in Wolff website with movie-related datasets provided by Linked Data providers. In this way, Wolff may reuse information provided by others and provide semantically-rich contents towards its visitors.

The goal of our project is to enrich movie data from Wolff website, and connect it with Linked Data provider. We choose DBpedia[3] as the provider due to its status as the largest open datasets provider today. We also developed a web application as a proof-of-concept of the improved-version of Wolff website using Semantic Web technology.

The report is structured as the following. Section 1 provides the background and motivation of this project. Section 2 gives description over the system we built and the workflow. Section 3 explains our self-developed ontology in detail. Section 4 explains our strategy on retrieving the original datasets and on enriching it. Section 5 gives non-trivial SPARQL queries we use against our RDF datasets to demonstrate the power of Linked Data. Section 6 discusses our results and challenges we faced throughout the project. And finally, section 7 concludes the report with remarks and notes for future work.

## 2 System Description

The system we built is illustrated in Figure 2. To structure our datasets, we need to define the ontology first as the common vocabularies. We develop Wolff ontology by using the existing data

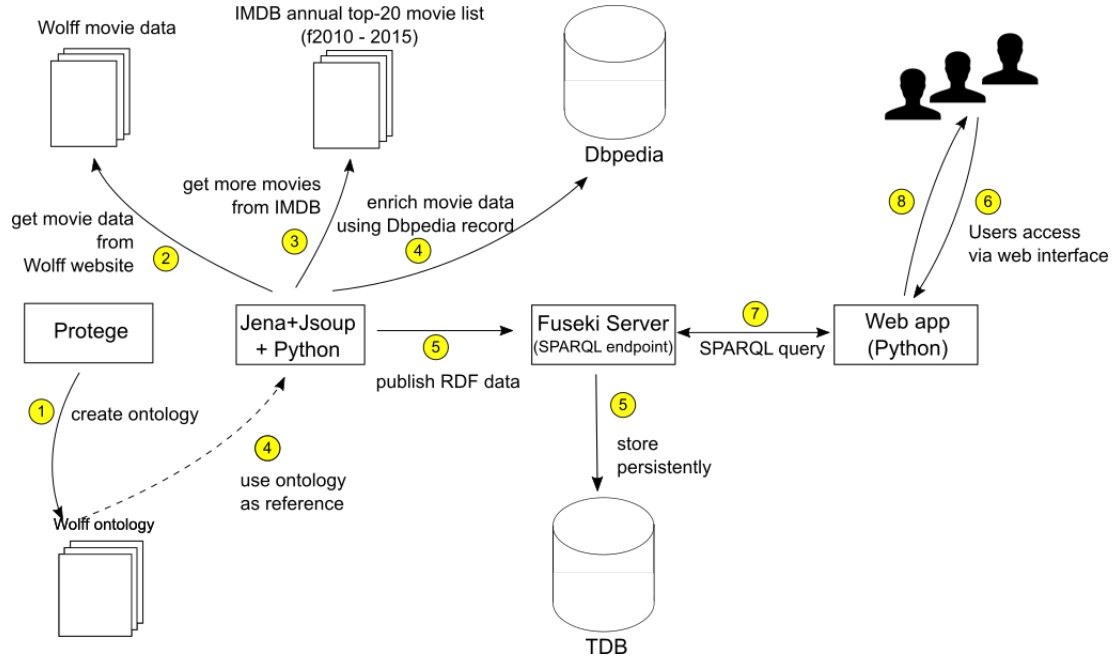


Figure 2: System description

in Wolff website to identify properties and classes needed. Since data provided from the website is limited, we add additional data (or, additional vocabularies) related to movie production and to person. We use Protege[13] to create our ontology. More detailed explanation about our ontology can be found in section 3.

The next step is to retrieve movie data from Wolff website and transforms it into RDF datasets. We built a Java application as a web crawler by using Jsoup[9], a Java HTML parser, to scrape all information related to movies currently on-screen in Wolff cinemas. Since Wolff website has only few movies at a time (they do not provide movie history), we decided to add more movies in order to make our datasets larger. We use the top-20 most popular movies from each year between 2010 - 2015 listed in IMDB[8] (retrieved using Python script) and built another java application to get related information from DBpedia. All data collected before were transformed into RDF data using Jena[1] by using the self-developed ontology as the reference. This RDF datasets creation functionality is also included in Java applications we mentioned before. Further explanation about datasets enrichment can be found in section 4.

Thirdly, The newly-created RDF data are transferred to Fuseki Server[6]. Fuseki server is an application to serve RDF data over HTTP (acting as a SPARQL endpoint), so that it can be consumed by other application remotely. The RDF data is stored persistently in TDB[17].

Lastly, we developed web application to demonstrate our concept of improved Wolff website using Semantic Web technology. We use Flask[4], a lightweight Python-based web framework to develop the prototype. The web app accesses data from the Fuseki server through SPARQL query. The website display all movies in our datasets including its related information. For each person web page, we provide the query results described in section 3.

All files are included as the deliverables and are explained in Appendix A.

### 3 Ontology

Linked Data is all about structured data. To structure our data, we need a reference, or vocabularies, to share common understanding. In Semantic Web world, this reference is called *ontology*. Our knowledge domain is Wolff movies. Therefore, we make use of the existing data provided in their website into our ontology, such as movie title, actors, duration, genre, release date, and so on. Since Wolff data is limited, we add our ontology with additional data by taking inspiration from DBpedia ontology for film.

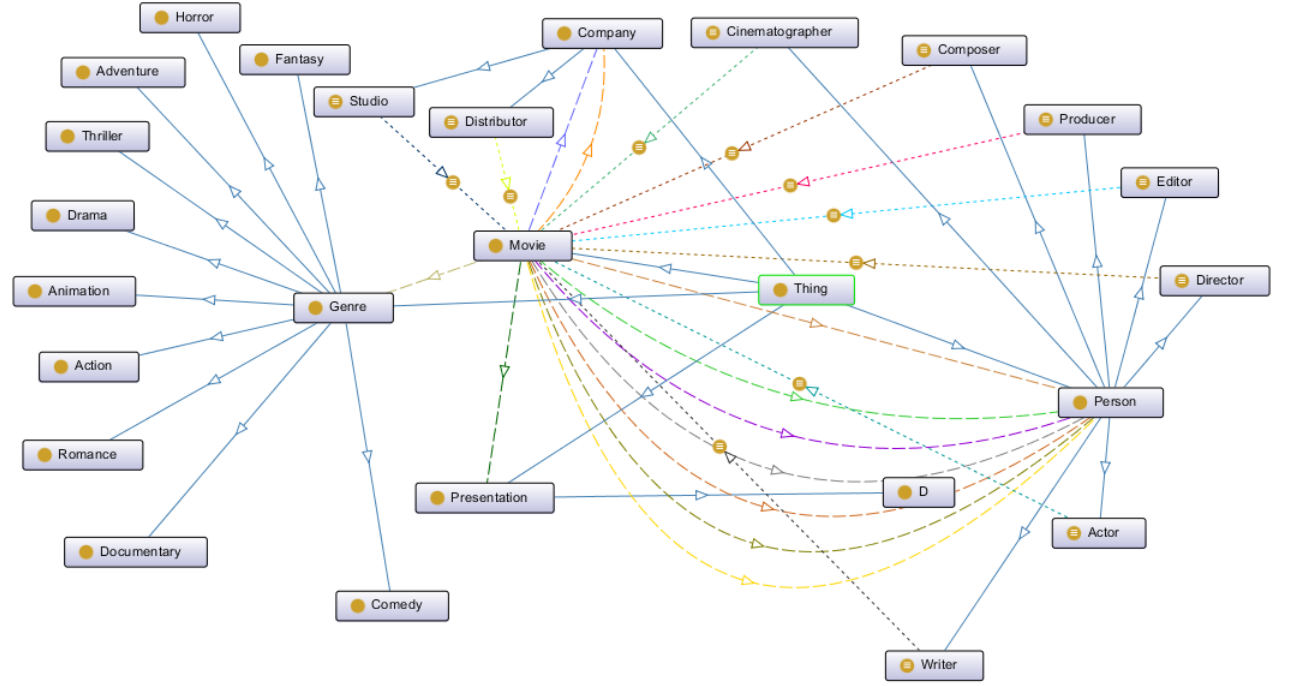


Figure 3: Ontology

Our ontology uses OWL[11] because of its nature as the most expressive ontology language available, compared to RDFS. We especially need one of its equality expression, *sameAs*, to describe that our RDF individual are the same with its equivalent resource in DBpedia datasets. It is important to simplify our SPARQL queries in section 5.

The ontology describes classes and properties related our domain, i.e., Wolff movies. There are five main classes which directly derived from OWL's top-level class, *Thing*: *Company*, *Genre*, *Movie*, *Person*, and *Presentation*. Furthermore, there are 20 sub-classes in total under the main classes. Figure 3 provide illustration of all classes and their relationships.

*Movie* represents a movie. *Person* describes all persons involved in a movie production. It includes *Actor*, *Director*, *Editor*, *Producer*, *Composer*, and *Cinematographer*. *Genre* describes the genre of a movie, ranging from *Drama*, *Animation* to *Fantasy*. *Company* describes companies related to movie production. We define two of them: *Studio* which produced the movie, and *Distributor* which distributed movies to cinemas. Finally, we define *Presentation* and its single subclasses *3D* to describe a 3D movie.

Object Property	Domain	Target
hasActor	Movie	Person
hasCinematographer	Movie	Person
hasDirector	Movie	Person
hasEditor	Movie	Person
hasDirector	Movie	Person
hasMusicComposer	Movie	Person
hasProducer	Movie	Person
hasWriter	Movie	Person
hasGenre	Movie	Genre
hasPresentation	Movie	Presentation
isDistributedBy	Movie	Company
isProducedBy	Movie	Company
isGenreOf	Genre	Movie
Produced	Company	Movie
Distributes	Company	Movie
becomesActorIn	Person	Movie
becomesCinematographerIn	Person	Movie
becomesDirectorIn	Person	Movie
becomesEditorIn	Person	Movie
becomesMusicComposerIn	Person	Movie
becomesProducerIn	Person	Movie
becomesWriterIn	Person	Movie

Table 1: Object properties defined

The definition of subclasses of Person in description logic is described in Listing 1. For example, in line 1, an Actor is defined as a Person who becomes an actor in a certain Movie. The rest of the subclasses have similar explanation.

```

1 Actor  $\equiv$  Person  $\sqcap \exists$ BecomesActorIn.Movie
2 Cinematographer  $\equiv$  Person  $\sqcap \exists$ BecomesCinematographerIn.Movie
3 Composer  $\equiv$  Person  $\sqcap \exists$ BecomesMusicComposerIn.Movie
4 Director  $\equiv$  Person  $\sqcap \exists$ BecomesDirectorIn.Movie
5 Producer  $\equiv$  Person  $\sqcap \exists$ BecomesProducerIn.Movie
6 Writer  $\equiv$  Person  $\sqcap \exists$ BecomesWriterIn.Movie

```

Listing 1: Description logic for some sub-classes

We focus the design of our ontology on Movie. Therefore, most relationships between classes are between Movie with the others. Object properties (Table 1) is used to describe relationships between entities in the ontology. All object properties with prefixes 'has' and 'is' have domain Movie and targeted to its respective classes. For example, hasActor connects a Movie to Persons who become the actors. The description of the rest of the object properties are self-explanatory.

Some object properties in Table 1 are not used directly in our RDF data. Those indirect properties are defined to provide bi-directional properties between classes. Basically, these properties (Listing 2) are the inverse of its counterparts. So, becomesActorIn represents the opposite of hasActor, i.e., to connect a Person to a Movie he/she plays in. The same explanation applies for the rest object properties listed below.

Another type of property is data property (table 2). Data property is attributes used to describe characteristics of a class. As with object property, most of data properties belong to

Data Property	Domain	Target
abstract	Movie	String
birthDate	Person	dateTime
budget	Movie	double
country	Movie	String
duration	Movie	double
language	Movie	String
name	Person	String
nationality	Person	String
releaseDate	Movie	dateTime
title	Movie	String

Table 2: Data properties defined

**Movie.** A Movie has a title (`title`), short description (`abstract`), amount of budget in US Dollar (`budget`), origin country (`country`), duration time in minute (`duration`), the language (`language`), and its release date (`releaseDate`).

```

1 becomesActorIn ≡ hasActor⁻
2 becomesCinematographerIn ≡ hasCinematographer⁻
3 becomesDirectorIn ≡ hasDirector⁻
4 becomesEditorIn ≡ hasEditor⁻
5 becomesMusicComposerIn ≡ hasMusicComposer⁻
6 becomesProducerIn ≡ hasProducer⁻
7 becomesWriterIn ≡ hasWriter⁻

```

Listing 2: object properties with inverse relationships

## 4 Enriching Original Datasets

The original data provided by Wolff website consists only movie title, list of actors, directors, release date, and duration. Since we define many more elements in our ontology (see section 3), we need to complete these original data.

The first option was to use linked movie database (LinkedMDB[10]). LinkedMDB is a linked open data provider specialized in movies. Unfortunately, the datasets are not well-updated. LinkedMDB datasets does not contain recently-released movies, which are the common ones on-screen in cinemas. Thus, it is not suitable to be used to enrich or to be linked by a cinema website such as Wolff's. Another problem we encountered is that LinkedMDB SPARQL endpoint is not reliable. It seems working well for one-time query. However, when we run a program which connected to it frequently, we often got server connection problems. It could be because they have limited network resources, hence limiting the number of connections from clients.

The next option is to use DBpedia[3]. DBpedia is a project to convert Wikipedia content into structured knowledge so that Semantic Web techniques can be employed against it. It means that each Wikipedia page has its own DBpedia resource. Since it is based on Wikipedia datasets, DBpedia have the largest open datasets available today. Another advantage is because Wikipedia content is actively maintained by its large volunteers, hence DBpedia datasets is relatively up-to-date.

As described in section 2, we developed Java application to scrape data from Wolff website and enrich each movie data using DBpedia datasets, to fill in the missing information as described in our ontology. The application is also automatically generate RDF datasets, by using

Jena framework. When a movie/company/person is found, it creates a resource. RDF triples are generated for each information related to the resource. For example, when the application scrape a movie 'The Hateful Eight' from Wolff website, it creates a resource<sup>1</sup>. Every information found in Wolff website, and from DBpedia as well, related to this movie (e.g., its title, duration, actors, etc.) is translated into RDF triples, with the movie resource as the subject, the captured information as the object, and the appropriate properties from our ontology as the predicate.

Wolff present a variety of movies, from Dutch films, German, Hollywood, and even Turk one. Problem arises when it comes to find the correct resources in DBpedia. For international movies (and for people internationally well-known), most of them have their own resources in DBpedia. For local movies (and people), they have their local DBpedia resources. For example, Sneekweek is a Dutch film which has wiki page only in <http://nl.dbpedia.org>, and therefore has resource only in Dutch DBpedia. The problem is, each local DBpedia has their own ontologies. To make matter worse, some country does not even have their local DBpedia, such as Turkey. Meanwhile, at the time of RDF datasets development, Wolff had a Turk movie on-screen. Accommodating this would make our program very complicated. Therefore, we decided to use only movies which have resources in DBpedia to be included in our RDF datasets.

Since we only got few data from Wolff website, we added our datasets using top-20 movies for each year between 2010-2015 from IMDB[8]. We only grab the movie list from IMDB, since IMDB does not allow the usage of scraper from their website without their written approval. We retrieved all data needed for each movie in this movie list from DBpedia using the same procedures as mentioned before. The resulting RDF datasets comprise of around 120 movies, hundreds persons, and couple of companies.

## 5 SPARQL Queries

SPARQL[16] is the query language to explore RDF datasets available, typically the ones provided by Linked Data providers. Here, we provide two examples of non-trivial SPARQL queries against our RDF datasets. The following queries are to find relationship information between the subject with other person. Both of them use federated query (performs queries from our SPARQL endpoint and DBpedia endpoint). This technique demonstrates one of the powerful features of Semantic Web: to interlink data. Using SPARQL federated query[15], we demonstrate that we are able to link our data with its related datasets in DBpedia so that we do not need to maintain large datasets in our local system. To relate our data with DBpedia datasets, we make use of `owl:sameAs` that we employed for all individuals in our RDF datasets which have its equivalent nodes in DBpedia datasets. Using this method, we are able to eliminate ambiguity possibilities during querying DBpedia datasets. All queries are included in file `views.py` (see Appendix A).

### 5.1 Query 1: People whom the subject frequently works together with

The first question is: *"with whom does a subject frequently works together with, and in what movies?"*. For example, we know that Leonardo diCaprio has worked together with director Martin Scorsese in many movies in the past. But to find the similar knowledge over arbitrary person in movie industry is not an easy question. We cannot immediately find the answer through quick search in Google, especially if the subject is not really well-known. Using Linked Data, we may find

<sup>1</sup>resource name: `<http://www.wolff.nl/2016/wolff.owl/movie#The_Hateful_Eight>`



this kind of information quickly. Our SPARQL query in Listing 3 demonstrates that. Here, we use *Quentin Tarantino* (`wolff-p:Quentin_Tarantino`) is the example of the subject.

```

1 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
2 PREFIX owl: <http://www.w3.org/2002/07/owl#>
3 PREFIX dbo: <http://dbpedia.org/ontology/>
4
5 SELECT DISTINCT
6   ?name
7   (GROUP_CONCAT(DISTINCT ?movietitle; SEPARATOR = "; ") AS ?movielist)
8   (COUNT(DISTINCT ?movietitle) AS ?movietotal)
9 WHERE {
10  wolff-p:Quentin_Tarantino owl:sameAs ?dbplink .
11 SERVICE <http://dbpedia.org/sparql> {
12  ?movie a dbo:Film .
13  ?movie ?p ?dbplink .
14  ?movie rdfs:label ?movietitle .
15  ?movie ?p2 ?actor .
16  ?actor a dbo:Person .
17  ?actor rdfs:label ?name .
18  FILTER (langMatches(lang(?movietitle),"en"))
19  FILTER (langMatches(lang(?name),"en"))
20  FILTER (?actor != ?dbplink)
21 }
22 }
23 GROUP BY ?name
24 ORDER BY DESC(?movietotal)
25 LIMIT 15

```

Listing 3: People with whom the subject frequently works with

The query starts by querying our local SPARQL endpoint (line 10) about the object of triples which have Quentin Tarantino as its subject and `sameAs` as its predicate. The result is basically the URI of DBpedia resource for Quentin Tarantino. This URI is retrieved during the enrichment process.

The next step is to query DBpedia endpoint (line 11). The query (line 12-17) basically asks all resources defined as `Film` where the movie has any relationship (predicate) with Quentin Tarantino, and at the same time the film also has any relationship with resource defined as a person. We ask for the title (label in DBpedia) of the movie, and the name of the person. Of course, such query will return many results. We then group the results by the person's name (line 23), sort the result based on the amount of movies where Quentin Tarantino worked with him/her in the past (line 24), and limit the output to only the top-15 (line 25). The expected output is the name of the person, list of movies where they worked together, and the number of those movies (line 6-8). The result of this query is shown in section 6.

## 5.2 Query 2: Oscar winners whom the subject have worked together in the past

The second question is: *"Which Oscar winners whom the subject worked together in the past, and in what movies?"*. This is more or less similar to the previous query, and is difficult to find the answer by quick googling for arbitrary person. Here, we define Oscar winners as actors who ever won either best actor, best actress, best supporting actor, or best supporting actress of Academy Awards. Listing 4 shows the SPARQL query. Again, we use Quentin Tarantino as the example subject.

```

1 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
2 PREFIX owl: <http://www.w3.org/2002/07/owl#>

```



```

3 PREFIX dbo: <http://dbpedia.org/ontology/>
4 PREFIX dct: <http://purl.org/dc/terms/>
5
6 SELECT DISTINCT ?strippedname ?strippedtitle WHERE {
7   wolff-p:Quentin_Tarantino owl:sameAs ?p_link .
8 SERVICE <http://dbpedia.org/sparql> {
9   ?movie a dbo:Film .
10  ?movie ?pre ?p_link .
11  ?movie dbo:starring ?actor .
12  { ?actor dct:subject <http://dbpedia.org/resource/Category:Best_Actor_Academy_Award_winners>
13    .}
14  UNION {
15    ?actor dct:subject <http://dbpedia.org/resource/Category:Best_Actress_Academy_Award_winners> .
16  }
17  UNION {
18    ?actor dct:subject <http://dbpedia.org/resource/Category:
19      Best_Supporting_Actor_Academy_Award_winners> }
20  UNION {
21    ?actor dct:subject <http://dbpedia.org/resource/Category:
22      Best_Supporting_Actress_Academy_Award_winners> }
23  ?actor rdfs:label ?name .
24  ?movie rdfs:label ?title .
25  FILTER(?p_link != ?actor)
26  FILTER (langMatches(lang(?name),"en"))
27  FILTER (langMatches(lang(?title),"en"))
28  BIND(str(?name) AS ?strippedname)
29  BIND(str(?title) AS ?strippedtitle)
30 }
31 }

```

Listing 4: Oscar Winners with whom the subject frequently works with

The explanation is pretty similar to the first query. Here, we also use Quentin Tarantino as the example of the subject. Directive UNION (line 13, 15, and 17) is used to combine result of queries in line 14, 16, and 18. The expected results are the Oscar winner names and the respective movies where the subject and the actor worked together. The output of this query is presented in section 6.

## 6 Discussion

### 6.1 Web application and the query results

We have developed web application as the interface to access our RDF datasets and to perform SPARQL queries mentioned in section 5. It is used only as a proof-of-concept of implementing Semantic Web technology in a website, hence it does not provide full functionalities as in the original Wolff website. If a person or movie link returns an error page, it means that the respective data is not in our datasets (the reason there is a link even though the data is not in the local RDF datasets is because we directly put the result of DBpedia query into the page). Figure 4 shows the front page of our web application. It lists all movies in our RDF datasets. It also shows short description for each movie, which retrieved from DBpedia.

If we click button 'view details' in any movie, we will be brought to the movie page (Figure 5 as an example). There, all information<sup>2</sup> retrieved about the movie during scraping and enrichment process is displayed, together with the link for each person or company. If we click on any person link in Figure 5, we will be brought to the person's page. There, user will be

<sup>2</sup>except for the image, which retrieved through ordinary scraping to Wikipedia by the web application

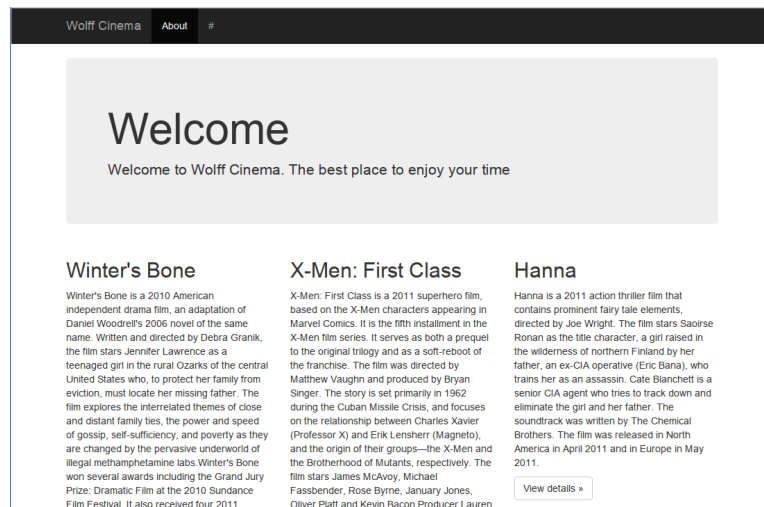


Figure 4: Screenshot of the web application

shown the name of the person, the person's birthdate, the person's nationality (if available), list of movies he/she involved in the past, result of query 1 and query 2, and his/her image from Wikipedia (Figure 6).

The result of query 1 (Listing 3) and query 2 (Listing 4) for the subject Quentin tarantino are illustrated in Figure 7 and Figure 8, respectively. For the first query result, it shows that Quentin Tarantino have worked together mostly with Lawrence Bender (a movie producer) in 12 movies. At the second place is Sally Menke, an editor. As for the actor, Mr. Tarantino's favorite is Michael Medsen (5 movies together). The result of query 2 is self-explanatory.

## 6.2 Dealing with ambiguity


The main challenge on working with Linked Data is about dealing with ambiguity. For example, if we search for 'The Revenant'—the latest movie of Leonardo DiCaprio in 2015—in Wikipedia, we will get multiple results: ranging from comics, novels, to movies. In SPARQL query, it is relatively easy to find 'The Revenant' that is the movie, by using triple '?s a dbo:Film'. However, it will also returns multiple results, since apparently there are movies with the same title in the past. To overcome this, one trick is to parse the value of `rdfs:label`, as DBpedia usually puts suffix '<year> film' to differentiate movies with the same title. However, this solution is movie-specific, and it might be not applicable for other domain. It becomes worse when the movie uses title from common words, such as 'Joy'. The same problem arises for people as well, since many people share the same names.

## 6.3 Finding the Linked Data provider

As we mentioned in brief in section 4, we decided not to use LinkedMDB eventhough it specializes in providing movie-related Linked Data. Apparently, LinkedMDB does not provide complete and up-to-date datasets. Movies recently released are not there yet. Even *Argo*<sup>3</sup>, an Oscar-winning film in 2013, is not there.

<sup>3</sup>[http://www.imdb.com/title/tt1024648/?ref\\_=nv\\_sr\\_1](http://www.imdb.com/title/tt1024648/?ref_=nv_sr_1)

Wolff Cinema
About
#



### In Time

In Time (previously titled Now and I'm mortal) is a 2011 American dystopian sci-fi thriller film written, directed, and produced by Andrew Niccol and starring Amanda Seyfried and Justin Timberlake which takes place in a society where people stop aging at 25 and each have a clock on their arms that counts down how long they have to live. The film was released on October 28, 2011.

Director	<a href="#">Andrew Niccol</a>
Producer	<a href="#">Andrew Niccol</a> , <a href="#">Eric Newman</a> (producer), <a href="#">Marc Abraham</a> ,
Actor	<a href="#">Justin Timberlake</a> , <a href="#">Alex Pettyfer</a> , <a href="#">Amanda Seyfried</a> , <a href="#">Cillian Murphy</a> ,
Composer	<a href="#">Craig Armstrong</a> (composer)
Cinematographer	<a href="#">Roger Deakins</a>
Editor	
Writer	<a href="#">Andrew Niccol</a>
Duration	109.0 minutes
Budget	\$ 4.0E7
Genre	<a href="#">Thriller</a> , <a href="#">Fantasy</a> ,
Country	<a href="#">United States</a> ,
Distributor	<a href="#">20th Century Fox</a>
Studio	<a href="#">Strike Entertainment</a>

Figure 5: Screenshot of the web application

Wolff Cinema
About
#



### Quentin Tarantino

Birthdate: 1963-03-27

Figure 6: Example of the top part of a person page

People whom Quentin Tarantino frequently collaborates with

Name	#	Movies
Lawrence Bender	12	From Dusk till Dawn (film series), Kill Bill, Pulp Fiction, From Dusk Till Dawn 2: Texas Blood Money, Killing Zoe, Four Rooms, Inglourious Basterds, Jackie Brown (film), Kill Bill Volume 1, Kill Bill Volume 2, Reservoir Dogs, Kill Bill: The Whole Bloody Affair,
Sally Menke	11	Kill Bill, Grindhouse (film), Death Proof, Pulp Fiction, Four Rooms, Inglourious Basterds, Jackie Brown (film), Kill Bill Volume 1, Kill Bill Volume 2, Reservoir Dogs, Kill Bill: The Whole Bloody Affair,
Robert Rodriguez	9	From Dusk till Dawn (film series), Grindhouse (film), Desperado (film), Death Proof, Four Rooms, From Dusk till Dawn, From Dusk Till Dawn 2: Texas Blood Money, Dead On: The Life and Cinema of George A. Romero, Kill Bill Volume 2,
Robert Richardson (cinematographer)	8	Natural Born Killers, Kill Bill, Django Unchained, Inglourious Basterds, Kill Bill Volume 1, Kill Bill Volume 2, Kill Bill: The Whole Bloody Affair, The Hateful Eight,
Scott Spiegel	5	From Dusk till Dawn (film series), My Name Is Modesty, From Dusk Till Dawn 2: Texas Blood Money, Hostel: Part II, Hostel (2005 film),
Michael Madsen	5	Kill Bill, Hell Ride, Kill Bill Volume 2, Reservoir Dogs, Kill Bill: The Whole Bloody Affair,
Guillermo Navarro	4	Desperado (film), Four Rooms, From Dusk till Dawn, Jackie Brown (film),
David Carradine	4	Kill Bill, Hell Ride, Kill Bill Volume 2, Kill Bill: The Whole Bloody Affair,
Uma Thurman	4	Kill Bill, Kill Bill Volume 1, Kill Bill Volume 2, Kill Bill: The Whole Bloody Affair,
Juliette Lewis	4	Natural Born Killers, From Dusk till Dawn (film series), Daltrey Calhoun, From Dusk till Dawn,
Harvey Keitel	3	From Dusk till Dawn (film series), From Dusk till Dawn, Reservoir Dogs,
Elizabeth Avellán	3	From Dusk till Dawn (film series), Grindhouse (film), Death Proof,
Boaz Yakin	3	From Dusk till Dawn (film series), Hostel: Part II, From Dusk Till Dawn 2: Texas Blood Money,
Eli Roth	3	Grindhouse (film), Hostel: Part II, Hostel (2005 film),
Rza	3	Kill Bill, Kill Bill Volume 1, Kill Bill Volume 2,

Figure 7: The first query result for subject Quentin Tarantino  
Oscar winners with whom Quentin Tarantino ever worked together

Name	Movie title
Gene Hackman	Crimson Tide (film)
Nicolas Cage	The Rock (film)
Denzel Washington	Crimson Tide (film)
Sean Penn	The Cutting Edge: The Magic of Movie Editing
Jamie Foxx	Django Unchained
Jodie Foster	The Cutting Edge: The Magic of Movie Editing
Christoph Waltz	Django Unchained
Sean Connery	The Rock (film)
Christopher Walken	True Romance
Tommy Lee Jones	Natural Born Killers
George Clooney	From Dusk till Dawn
George Clooney	From Dusk till Dawn (film series)
Marisa Tomei	Four Rooms

Figure 8: The second query result for subject Quentin Tarantino

Therefore, we resort to DBpedia since it directly derives its datasets from Wikipedia, which is the largest encyclopedia on-line and has large number of volunteers to maintain the articles. However, DBpedia also poses the same problem of data completeness. Consider movie Spectre, the latest James Bond film. It can be assumed as a popular movie. However, DBpedia does not provide complete list of its actors<sup>4</sup>, eventhough its wikipedia infobox<sup>5</sup> have a relatively complete data about the movie. This is a common problem with community-based project.

<sup>4</sup>[http://dbpedia.org/page/Spectre\\_\(2015\\_film\)](http://dbpedia.org/page/Spectre_(2015_film))

<sup>5</sup>[https://en.wikipedia.org/wiki/Spectre\\_\(2015\\_film\)](https://en.wikipedia.org/wiki/Spectre_(2015_film))

## 6.4 Self-developed ontology vs reusing existing Ontology

In this assignment, we developed our own movie ontology. However, there is already exist an established and well-known movie ontology from `schema.org`[14]. This schema has been used widely, for example by Google Knowledge Graph Search API[7]. In part of our ontology, we also define a class to describe a person, including the properties. There is also already many ontologies available to describe people. One of them is FOAF[5], which is used to describe people and their relations to other people and objects, and is commonly used.

Developing our own ontology brings benefit where we can accurately defines our knowledge domain in details. However, when we publish our datasets publicly, it comes with a cost of complexities for others to link into our data. For a well-known domain, such as movie, it would be better to use the existing ontology specifically developed for that area. In this way, people who would like to interlink with our datasets may directly use ours because they are already familiar with the ontology.

## 6.5 Making use of reasoning feature

Another powerful feature of Semantic Web is its reasoning capability. A semantic reasoner is a software which has capability to infer logical consequences from a set of asserted facts or axioms<sup>6</sup>. In Listing 2, we show some object properties which are the inverse of its counterparts. In fact, during generating the RDF data, we only use property with prefix 'becomes', in the hope that we can demonstrate the use of Jena reasoning feature by querying the 'has' properties (to show the inverse relationship from person to movie). Unfortunately, we failed to do so. After discussing this problem in Jena mailing list, it might be because the state of Jena reasoner which is not mature yet. One of the suggestion is to use another OWL DL reasoner Pellet[12]. However, due to time restriction, we are not able to use this tool and decide to leave reasoning feature from our project.

## 7 Conclusions

Movie domain is a semantically-rich area and it has a high potential to leverage the power of Semantic Web, especially the Linked Data, to develop a dynamic website that is very informative. In this project, we use Wolff website as a case study of a static web page, and develop an improved version of it by using Linked Data provided by DBpedia. In this way, we are able to show some dynamic improvement such as providing capabilities to show information about movie person which cannot be easily searched in Google, such as professional works between people in the pass.

Albeit its promising potentials, implementing Semantic Web provides challenges: (1) Dealing with data ambiguity. We need to identify the best way to filter out resources which happen to have similar properties with our intended one (for example, having the same title or name). (2) Finding appropriate Linked Data providers to interlink with our application. DBpedia is the largest Linked Data available today. But we need to be careful because not all of its resource have complete information. (3) making decision whether to create self-developed ontology or reuse the existing one. We believe that self-developed ontology is beneficial when our knowledge domain is very specific and its ontology has not been defined before. If not, then reusing popular ontology is the better option, since people may directly use our datasets. Finally, (4) limitation from the tools we used. In order to develop a Linked Data application, we need to know the

---

<sup>6</sup>[https://en.wikipedia.org/wiki/Semantic\\_reasoner](https://en.wikipedia.org/wiki/Semantic_reasoner)

capabilities of the tools we are going to use. For example, if we heavily need the reasoner, then Jena might be not an appropriate tool since its reasoner is not mature yet.

## A Deliverables

Figure 9 illustrates the directory tree of our deliverables. Some important files/folders:

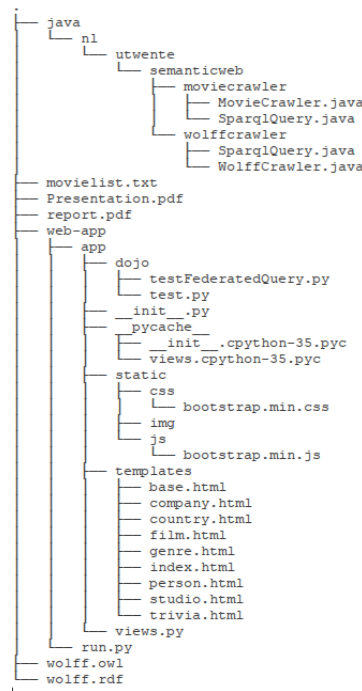


Figure 9: Directory tree of the deliverables

**report.pdf** This report

**wolff.owl** Wolff movie ontology

**wolff.rdf** RDF datasets. Should be uploaded to Fuseki Server and to be put in dataset 'wolff' (otherwise, file `views.py` in `web-app` directory should be modified)

**java/nl/utwente/semanticweb/wolffcrawler/WolffCrawler.java** Java program to scrape movie data from Wolff website, enrich it using DBpedia data, and create the RDF datasets

**java/nl/utwente/moviecrawler/moviecrawler/MovieCrawler.java** Java program to get additional movie datasets from DBpedia, based on the list of movies (`movielist.txt`) taken from IMDB

**movielist.txt** List of movies for additional datasets; top-20 movies for each year between 2010-2015

**web-app/** Flask web application. To run it, simply do this: `'python run.py'`. Requires packages Flask, SPARQLWrapper, and bs4 installed.

**web-app/app/run.py** The main Python script for the web application. It also contains all SPARQL queries used, including queries in section 5 in generalized form (the subject depends on the input variable).

**presentation.ppt** The presentation slide

## References

- [1] *Apache Jena*. URL: <https://jena.apache.org/>.
- [2] Christian Bizer, Tom Heath, and Tim Berners-Lee. “Linked data-the story so far”. In: *Semantic Services, Interoperability and Web Applications: Emerging Concepts* (2009), pp. 205–227.
- [3] *DBpedia project*. URL: <http://wiki.dbpedia.org/>.
- [4] *Flask*. URL: <http://flask.pocoo.org/>.
- [5] *FOAF Vocabulary Specification 0.99*. URL: <http://xmlns.com/foaf/spec/>.
- [6] *Fuseki*. URL: [https://jena.apache.org/documentation/serving\\_data/](https://jena.apache.org/documentation/serving_data/).
- [7] *Google Knowledge Graph API*. URL: <https://developers.google.com/knowledge-graph/?hl=en>.
- [8] *IMDB: Most popular Titles*. URL: <http://www.imdb.com/search/title?sort=moviemeter>.
- [9] *Jsoup: Java HTML Parser*. URL: <http://www.jsoup.org>.
- [10] *Linked Movie Database*. URL: <http://www.linkedmdb.org/>.
- [11] *OWL 2 Web Ontology Language Document Overview (Second Edition)*. URL: <https://www.w3.org/TR/owl2-overview/>.
- [12] Bijan Parsia and Evren Sirin. “Pellet: An owl dl reasoner”. In: *Third International Semantic Web Conference-Poster*. Vol. 18. Citeseer. 2004.
- [13] *Protégé*. URL: <http://protege.stanford.edu/>.
- [14] *Schema.org: Movie*. URL: <http://schema.org/Movie>.
- [15] *SPARQL Federated Query*. URL: <https://www.w3.org/TR/sparql11-federated-query/>.
- [16] *SPARQL Query Language for RDF*. URL: <https://www.w3.org/TR/rdf-sparql-query/>.
- [17] *TDB*. URL: <https://jena.apache.org/documentation/tdb/>.
- [18] *Wolff Bioscopen*. URL: <http://www.wolff.nl/>.