**CSCI 586 Project Report**

**Spring 2014**

**Group #5**

**Books and Movies based on books**

**Group Members:**

Anjul Kumar

Zhi Qiao

1. **Problem Statement**

The aim of the project is to create a federated ontology, based on the domain books and movies based on books, that can serve as an informational guide for users. The different aspects of the domain are to be considered and pre-existing ontology sources for each aspect is to be used to build the meta-ontology.

1. **Scope of the Project**

The source for the book aspect of the domain is DBpedia and movies aspect is Linkedmdb.

The data collected for the books aspect are:

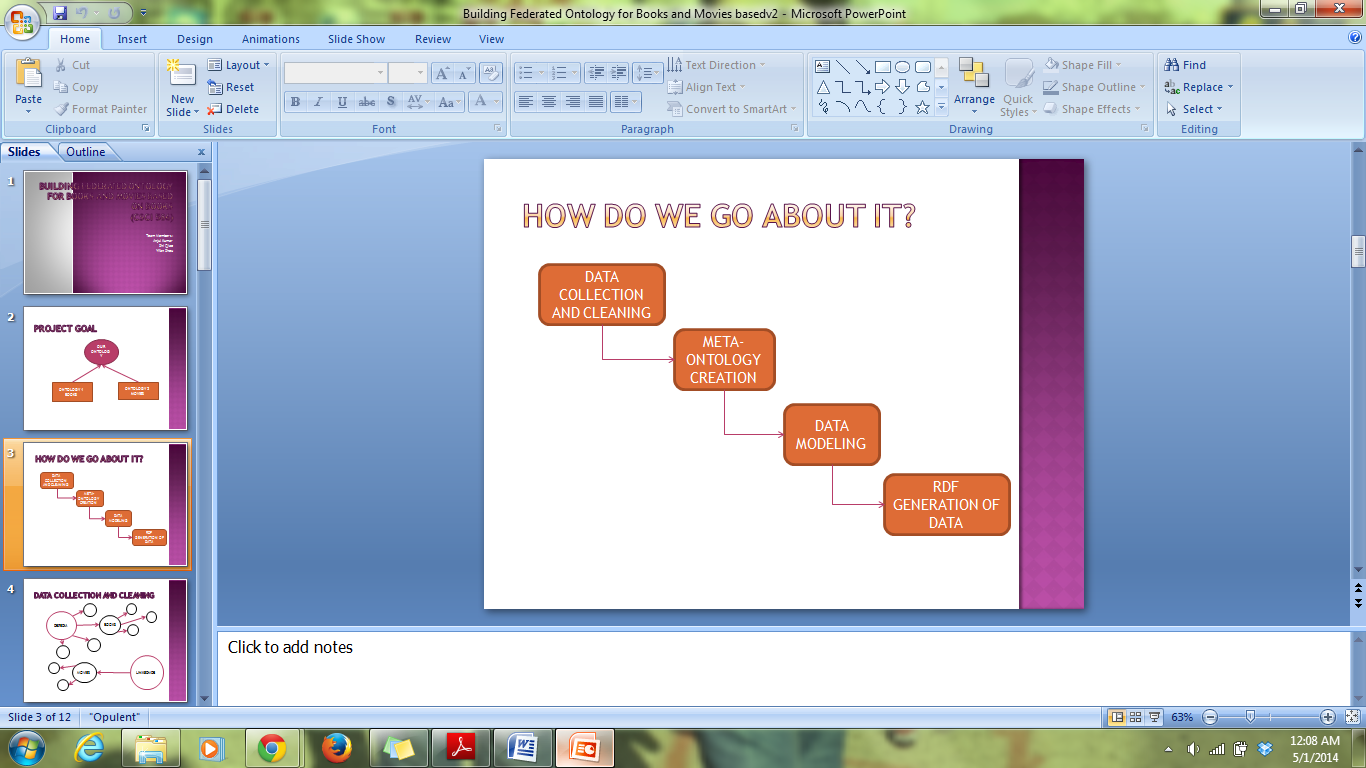
* Books - and its data properties - title and number of pages.
* Writer - and its data property - name.

The data collected for the movies aspect are:

* Movies - and its data properties - title, runtime and release date
* Actor - and its data properties - name.
* Director - and its data properties - name.
* Producer - and its data properties - name.
* Genre - and its data properties - description.

1. **Steps Involved**

The figure below shows the steps followed to create the federated ontology for the domain books and movies based of books.

****

**Figure1: Steps Involved**

1. **Data Collection and Cleaning:**

The source chosen to retrieve data for books was DBpedia and the source for movies data was Linkedmdb. The data from the sources were chosen to fit the design of our ontology. Relevant data were retrieved from each source through SPARQL queries. Retrieved data was exported to Excel file format.

1. **Meta-Ontology Creation:**

In this step, we designed a meta-ontology using Protégé incorporating both aspects of the domain books and movies based on books. The classes created were Books, Movies, Genre and Person where Person class has the sub-classes Actor, Director, Producer and Writer(Author). The relationship and the inverse relations between the classes is defined. Also, the data properties of each class is defined.

1. **Data Modeling:**

The data was modeled to be incorporated into created ontology using Karma, a tool developed by Information Integration group at USC-ISI. The Semantic classes were mapped to the corresponding columns in the data and their relations were established using the data and object properties.

1. **RDF Generation of Data:**

The RDF generated was then exported from Karma in Turtle format(ttl format). This ttl file was then imported into Protégé and was further used to query for information.

1. **Design and Methodology**
2. **Data Collection:**

The data was from DBpedia and Linkedmdb for books and movies respectively was collected using SPARQL queries. The screenshots for output haven't been displayed due to less clarity of the many columns outputted.

1. **SPARQL query to DBpedia:**

SELECT ?film\_title ?BOOK\_name ?nameAuthor

WHERE

{?subject <http://purl.org/dc/terms/subject> <http://dbpedia.org/resource/Category:American\_novels\_adapted\_into\_films>.

OPTIONAL

{ ?movies rdf:type <http://dbpedia.org/ontology/Film>;

rdfs:label ?film\_title;

dbpedia-owl:basedOn ?subject.

?subject rdfs:label ?BOOK\_name;

dbpedia-owl:author ?author.

?author foaf:name ?nameAuthor.

FILTER LANGMATCHES( LANG(?film\_title), 'en')

}

FILTER LANGMATCHES( LANG(?BOOK\_name), 'en')

}

1. **SPARQL Query to Linkedmdb**

SELECT ?movie\_title ?genreN ?rd ?dirN ?prodN ?actorN

WHERE

{

?movie rdf:type <http://data.linkedmdb.org/resource/movie/film>;

rdfs:label ?movie\_title;

movie:director ?dir;

movie:producer ?prod;

movie:actor ?actor;

movie:initial\_release\_date ?rd.

?dir movie:director\_name ?dirN.

?prod movie:producer\_name ?prodN.

?actor movie:actor\_name ?actorN.

OPTIONAL

{

?movie1 rdf:type <http://data.linkedmdb.org/resource/movie/film>;

rdfs:label ?movie\_title;

movie:genre ?genre.

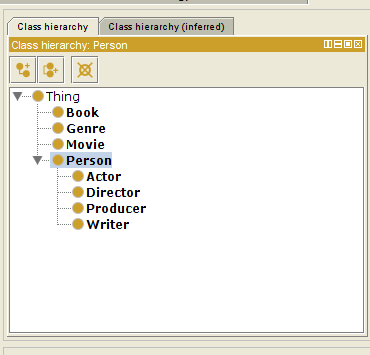
?genre movie:film\_genre\_name ?genreN

}

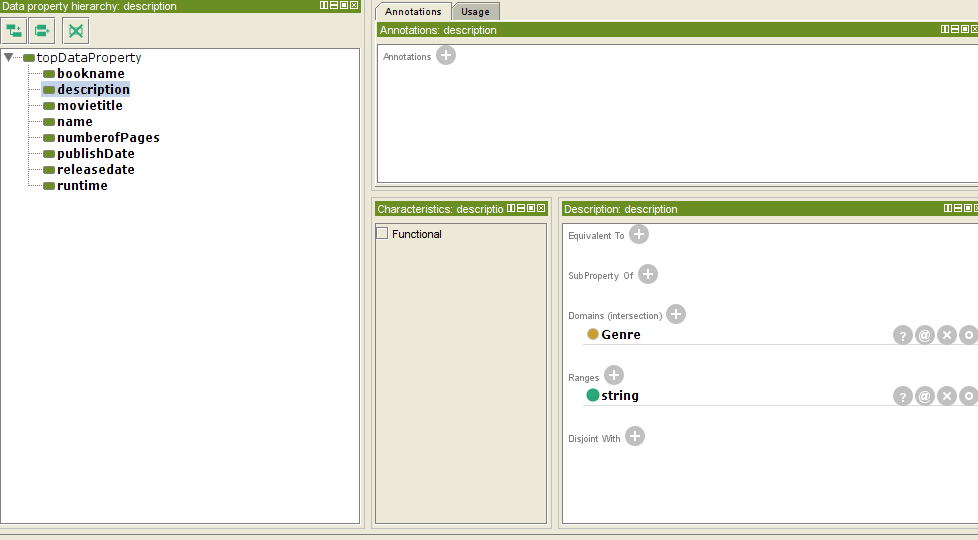
}

1. **Ontology:**

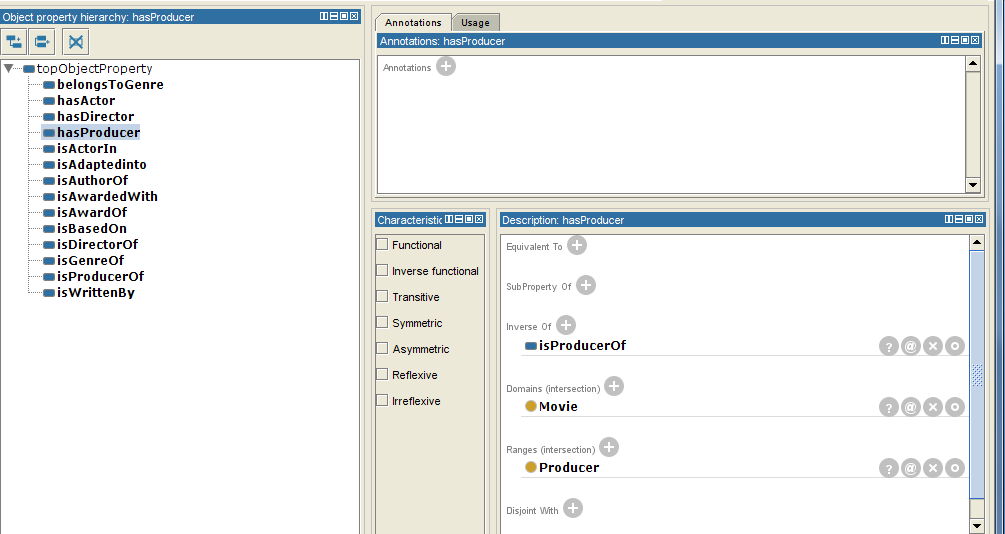
The design for the meta-ontology is as shown in the figure below. Figure 2 shows the classes and sub-classes created. Figure 3 shows the data properties of all the classes. Figure 4 shows the object properties(relations) and the inverse relations between the classes. Figure 5 shows the OntoGraph of the meta-ontology.



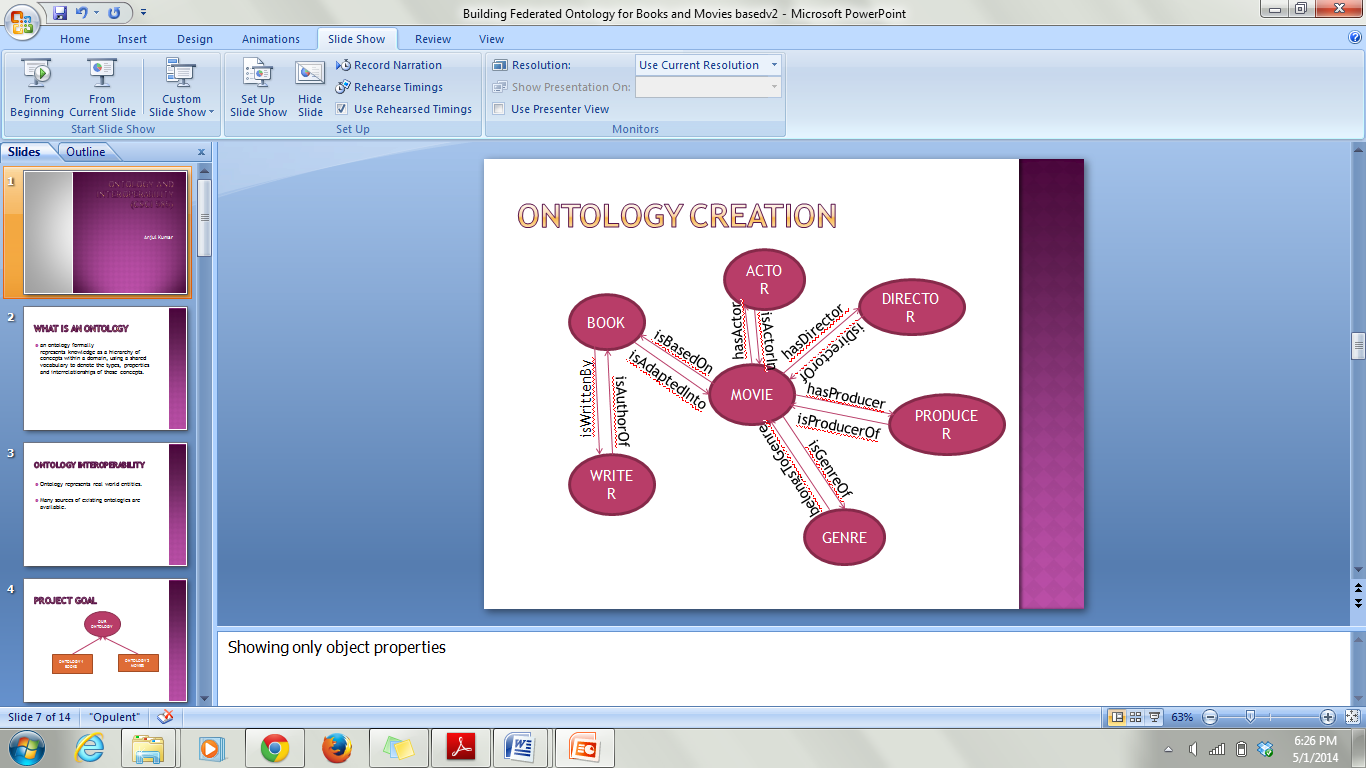
**Figure 2: Classes in the ontology**

****

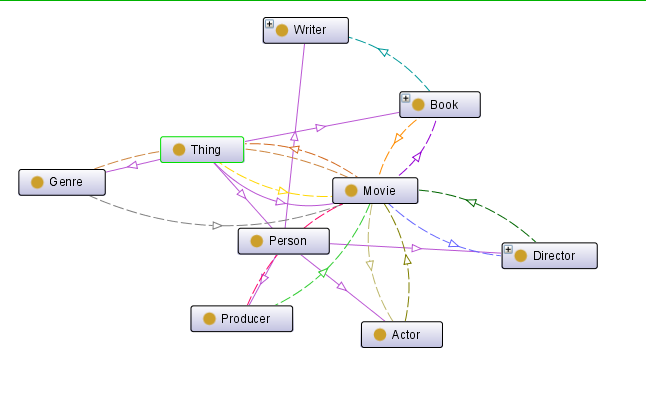
**Figure 3: Data properties of every class as described in the scope of the project**

****

**Figure 4: Object Properties of the classes as seen in Protégé**

****

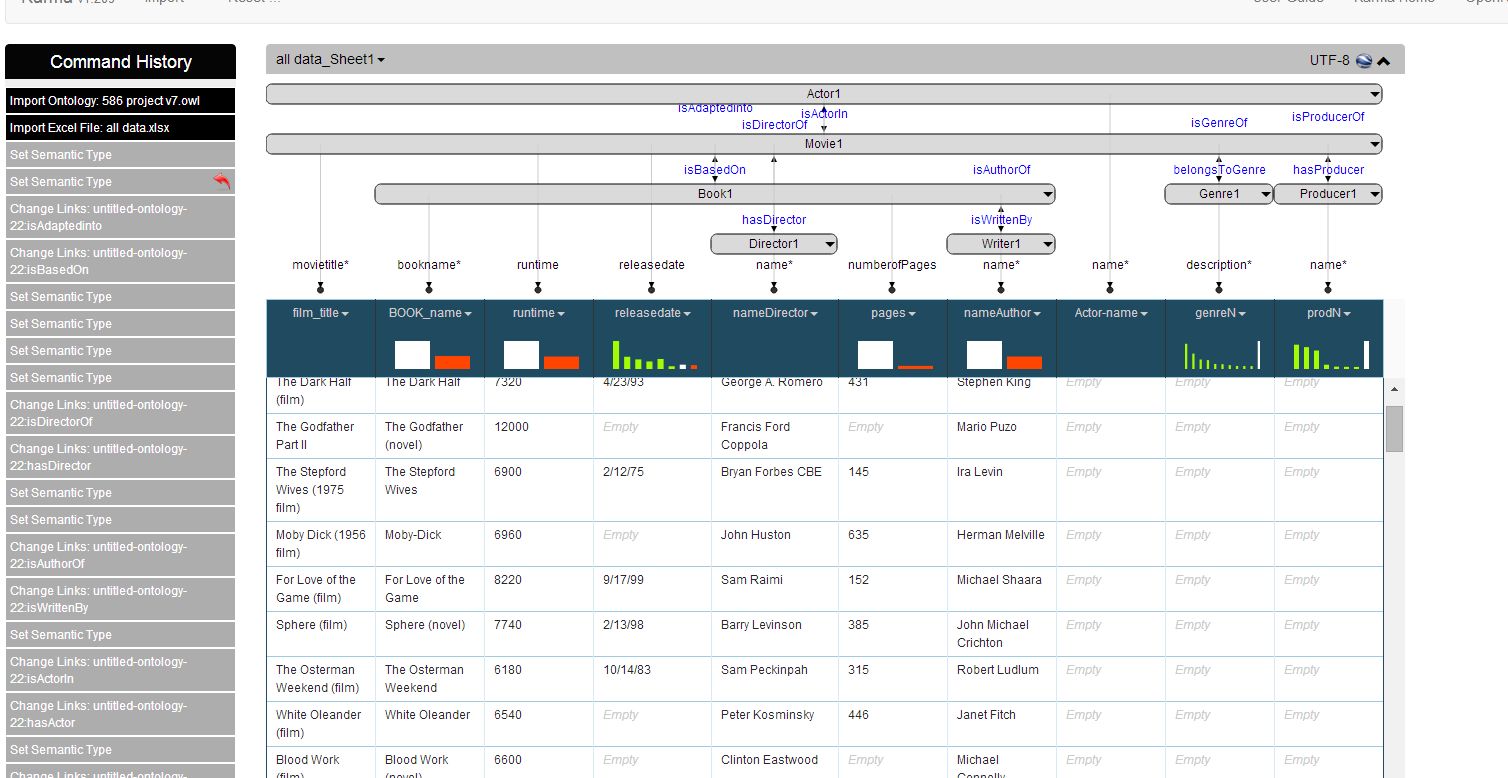
**Figure 4 continued: Object Properties diagrammatic representation**

****

**Figure 5: OntoGraph as seen in Protégé**

1. **Data Modeling**

The screenshot below shows how the data was modeled using the Karma Tool. The long rectangular boxes represent the instances of the classes and arrows represent the data and object property mapping. The blue labeled property along double sided arrows between classes are the object property and the black labeled property along single sided arrows towards the table are data property.



1. **Tools**
2. **Protégé:**

Protégé is an open source ontology editor and knowledge-base framework which supports modeling ontologies via a web client or a desktop client. Protégé provides the ability to develop ontologies in a variety of formats including OWL, RDF(S), and XML Schema. Protégé is a Java-based plug-and-play environment that makes it a flexible base for rapid prototyping and application development.

1. **Karma:**

Karma is an information integration tool that enables users to quickly and easily integrate data from a variety of data sources including databases, spreadsheets, and delimited text files, XML, JSON, KML and Web APIs. The information can be integrated by modeling it according to ontology of one's choice using a graphical user interface that automates much of the process. Karma recognizes the ontology classes and then uses the ontology to propose a model that maps the data appropriately to the ontology. The generated can then be manipulated to make certain corrections or changes. Once the model is complete the data can be integrated as RDF or store it in a database.

1. **Sample SPARQL Queries**
2. **Movies Based on Books**

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX towl:<http://www.semanticweb.org/fcqiao/ontologies/2014/3/untitled-ontology-22#>

SELECT $movie\_title $book2

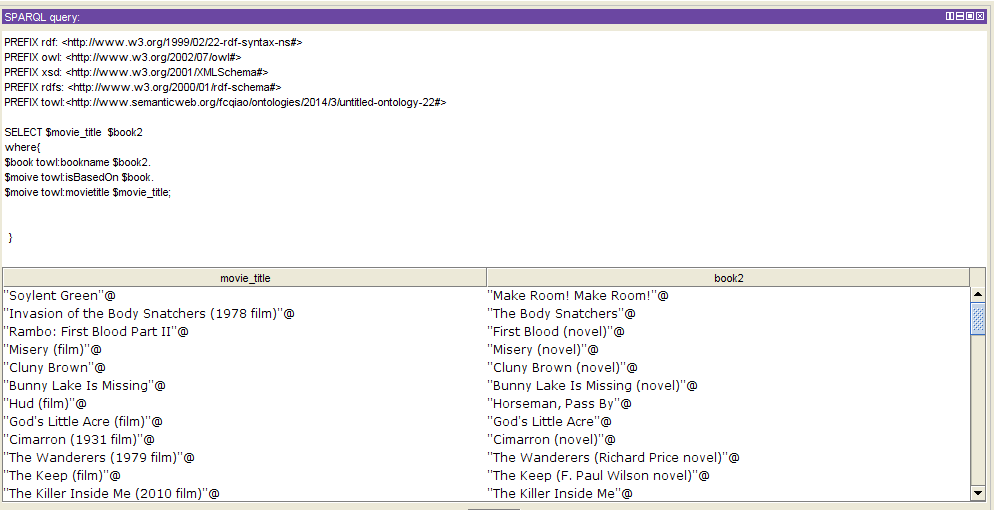
where{

$book towl:bookname $book2.

$moive towl:isBasedOn $book.

$moive towl:movietitle $movie\_title;

}



1. **List of Books and the number of pages ordered by number of pages**

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX towl:<http://www.semanticweb.org/fcqiao/ontologies/2014/3/untitled-ontology-22#>

SELECT $book\_name $bookpage

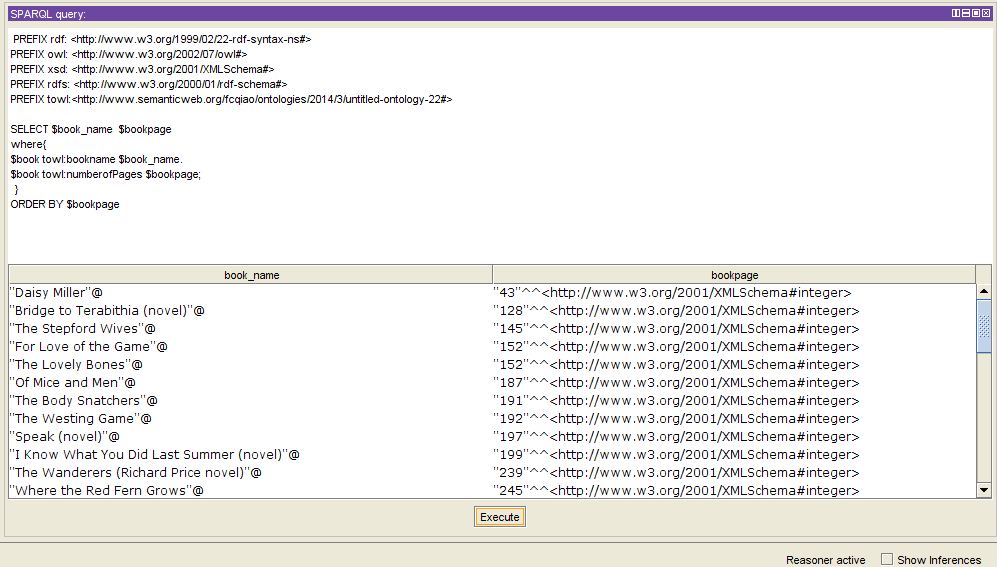
where{

$book towl:bookname $book\_name.

$book towl:numberofPages $bookpage;

}

ORDER BY $bookpage



1. **Movie and the actors that starred in each movie**

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX towl:<http://www.semanticweb.org/fcqiao/ontologies/2014/3/untitled-ontology-22#>

SELECT $movie\_name $actor\_name

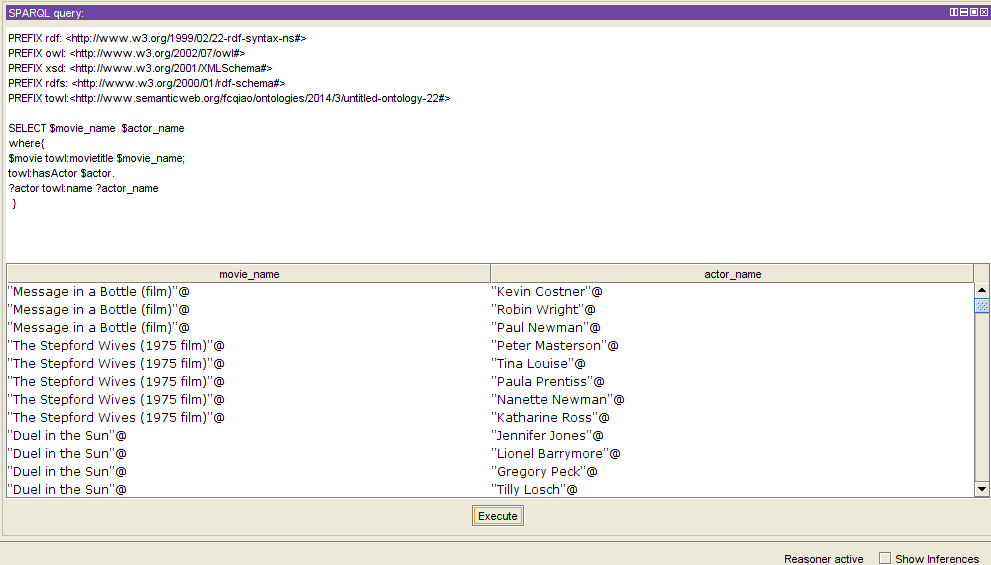
where{

$movie towl:movietitle $movie\_name;

towl:hasActor $actor.

?actor towl:name ?actor\_name

}



1. **Movies in which an actor whose has 'John' in his name**

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX towl:<http://www.semanticweb.org/fcqiao/ontologies/2014/3/untitled-ontology-22#>

SELECT ?actor\_name ?movie\_name

WHERE

{

?actor towl:name ?actor\_name;

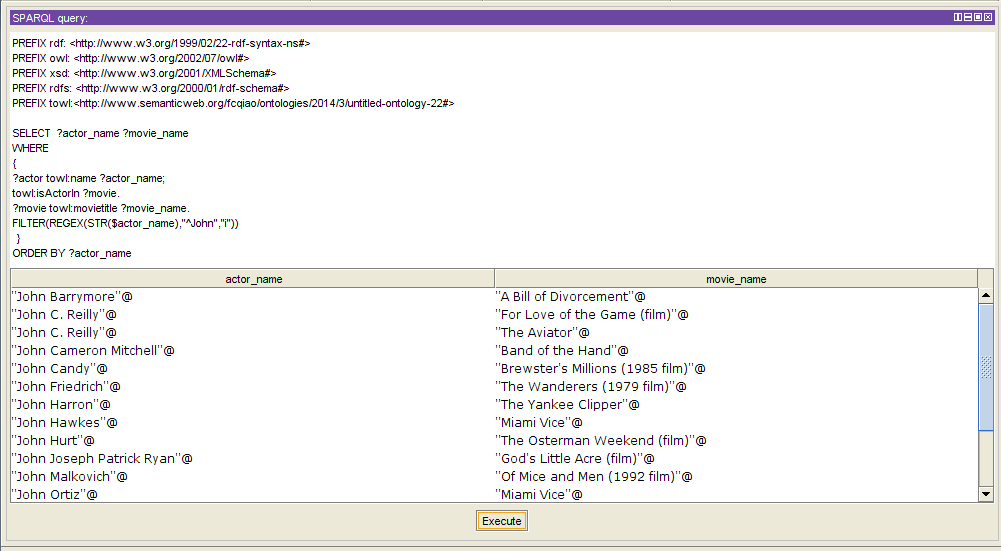
towl:isActorIn ?movie.

?movie towl:movietitle ?movie\_name.

FILTER(REGEX(STR($actor\_name),"^John","i"))

}

ORDER BY ?actor\_name



1. **Movies released in 1997 ordered by runtime(in seconds)**

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX towl:<http://www.semanticweb.org/fcqiao/ontologies/2014/3/untitled-ontology-22#>

SELECT $movie\_name $book\_name $date $run\_time

WHERE

{

$book towl:bookname $book\_name.

$movie towl:isBasedOn $book;

towl:movietitle $movie\_name;

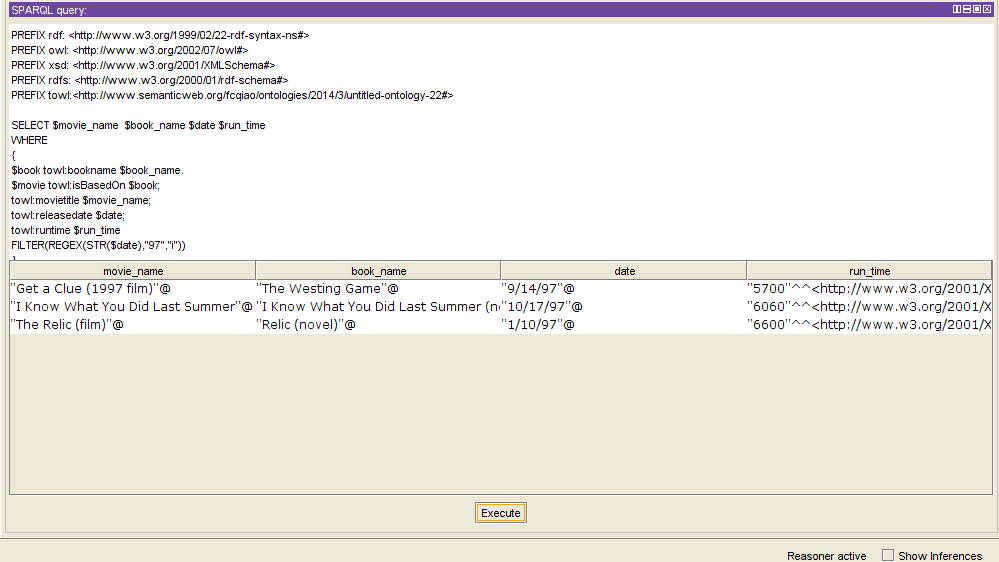
towl:releasedate $date;

towl:runtime $run\_time

FILTER(REGEX(STR($date),"97","i"))

}

ORDER BY($run\_time)



1. **Person who is a director and a producer and the movies he/she has directed and produced**

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX towl:<http://www.semanticweb.org/fcqiao/ontologies/2014/3/untitled-ontology-22#>

SELECT DISTINCT $director\_and\_producer $movie\_directed\_and\_produced

WHERE

{

$person towl:isDirectorOf $movie1;

towl:isProducerOf $movie2;

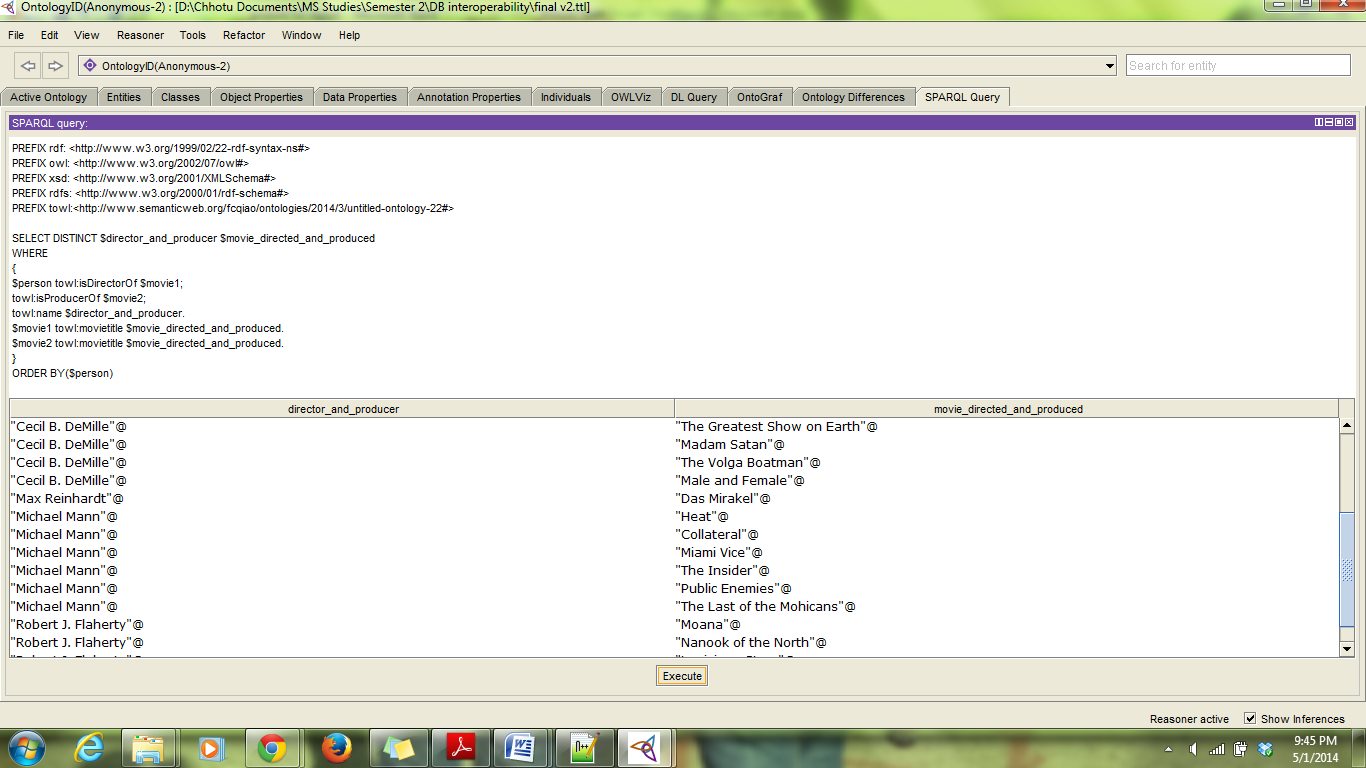
towl:name $director\_and\_producer.

$movie1 towl:movietitle $movie\_directed\_and\_produced.

$movie2 towl:movietitle $movie\_directed\_and\_produced.

}

ORDER BY($person)



1. **Movies based on books, the corresponding book and the book's author, ordered by number of pages in book**

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX towl:<http://www.semanticweb.org/fcqiao/ontologies/2014/3/untitled-ontology-22#>

SELECT $movie\_title $book\_name $author\_name $number\_of\_pages

where{

$book towl:bookname $book\_name;

towl:isWrittenBy $author;

towl:numberofPages $number\_of\_pages.

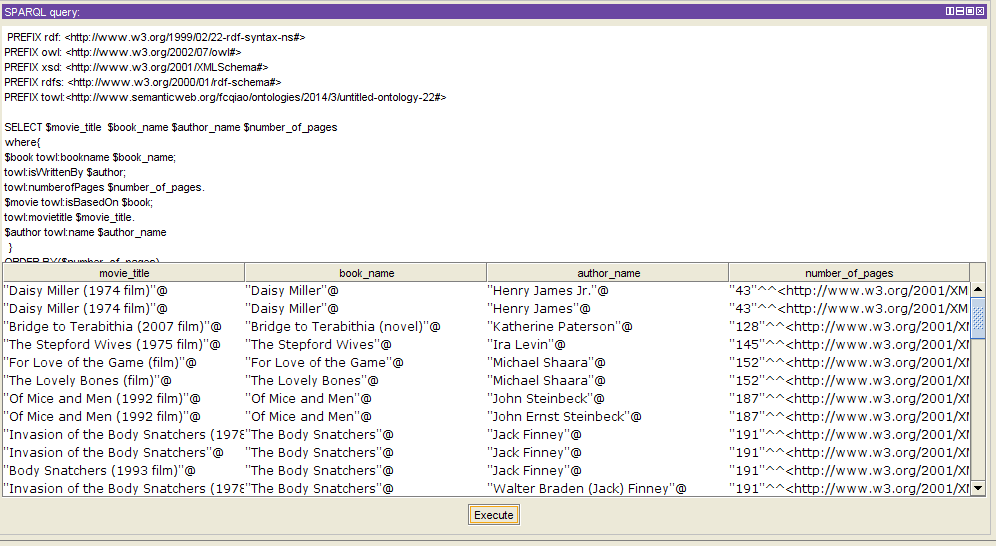
$movie towl:isBasedOn $book;

towl:movietitle $movie\_title.

$author towl:name $author\_name

}

ORDER BY($number\_of\_pages)



1. **Movies based on Stephen King's books**

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX towl:<http://www.semanticweb.org/fcqiao/ontologies/2014/3/untitled-ontology-22#>

SELECT $movie\_title $book\_name

where{

$book towl:bookname $book\_name;

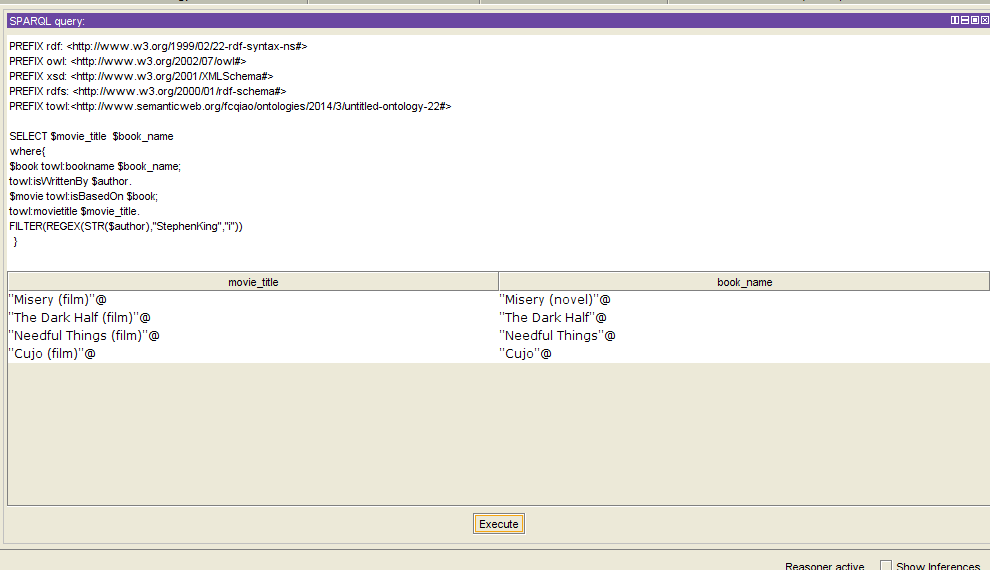
towl:isWrittenBy $author.

$movie towl:isBasedOn $book;

towl:movietitle $movie\_title.

FILTER(REGEX(STR($author),"StephenKing","i"))

}



1. **Indie Movies released in 2005**

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX towl:<http://www.semanticweb.org/fcqiao/ontologies/2014/3/untitled-ontology-22#>

SELECT $movie\_name $genre\_type $date

WHERE

{

$movie towl:belongsToGenre $genre;

towl:movietitle $movie\_name;

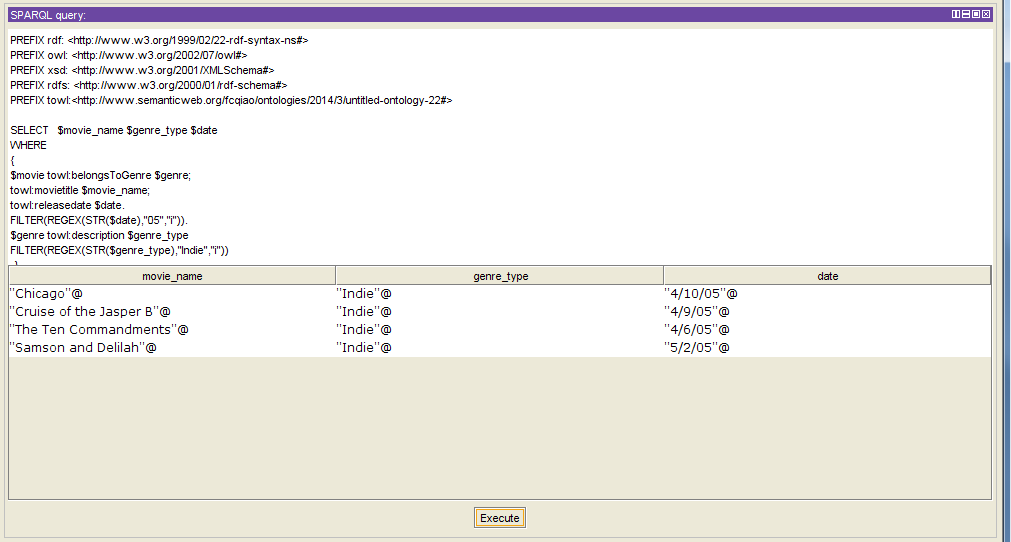
towl:releasedate $date.

FILTER(REGEX(STR($date),"05","i")).

$genre towl:description $genre\_type

FILTER(REGEX(STR($genre\_type),"Indie","i"))

}



1. **References**

[1] http://protege.stanford.edu/

[2] http://www.w3.org/TR/rdfsparqlquery/

[3] http://www.isi.edu/integration/karma/

[4] http://dbpedia.org/page/Category:American\_novels\_adapted\_into\_films

[5] http://wiki.linkedmdb.org/Main/Interlinking