Assignment 1: Knowledge Graph Schema Design

Mater Student Evangelia P. Panourgia, f3352402
Professor Dr. Panos Alexopoulos

Modeling KG Schema

Before delving into the modeling logic for the provided competency questions, it is important to note that this section outlines design decisions applicable to both Task 1 and Task 2. First, we will examine the design decisions related to the first competency question, which is:

"Which movies have been produced in the United States?"

Based on the provided competency question, we can infer that the following elements should be modeled: **movies** (already modeled), **have been produced**, and **United States**. Through the knowledge acquired in the master's course, combined with the support of ChatGPT, we will explore various potential alternatives. After a thorough analysis, we will conclude by selecting the most suitable option. Before analyzing the possible alternatives, it is important to note that in real-world applications, we would consult with the customer to gather more detailed information. This step helps us determine whether the data should be modeled as a simple attribute (a value), as a more detailed element (a class), or whether a hierarchical structure is needed, among other considerations.

- **1st Hierarchy (is-a relationship) for USA**. We propose modeling the geographical location (USA) in a class hierarchy where "GeographicalEntity" is a superclass, "Country" is a subclass, and "USA" is an instance of the "Country" subclass. According to theory, this design is logical because the USA is a specific instance of the class "Country," and "Country" is a subclass of the broader concept "GeographicalEntity" (all countries including USA are GeographicalEntities). The names of subclasses and superclasses are critical, as they define the hierarchy accurately. For example, if the superclass were named "Continent," it would be incorrect because a "Country" is not a type of "Continent." Proper naming ensures clarity and logical structure within the ontology.
  - **Prons**: Relationships make sense when representing "type-of" or "kind-of" relationships (e.g., USA is a type of Country). In addition, useful for inheritance if many countries share common attributes. Furthermore, regarding the scalability, if our customer wants an ontology to include more geographical entities, this hierarchical structure can work well, but for simpler use cases, it may create unnecessary indirection.
  - **Crons**: This approach works well for classifying countries but does not capture the specific relationship between a movie and where it was produced. The is-a relationship suggests taxonomy or categorization rather than production location. In addition, overcomplicates the ontology if your main focus is simply linking movies to their production locations. Furthermore, the query asks for a relationship between movies and countries (i.e., "produced in"). In a hierarchy model, you would need an additional property (such as "producedIn") to explicitly link the movie to a country. Without such a direct relationship, the system has to traverse the class hierarchy (GeographicalEntity (superclass) $\rightarrow$ Country (subclass) $\rightarrow$ USA (instance)), which adds unnecessary complexity.
  - **General Comment**:The hierarchical approach is best suited for cases where you need to build an ontology with complex geographical modeling and a variety of geographical entities (like cities, regions, or continents), not just countries. However, for a competence query focused on movie production countries, the hierarchy adds unnecessary complexity.
- **2nd Broader/Narrower Relationship (part-whole relation)**This approach models the relationship between geographical entities, where countries like the USA are part of larger entities such as continents (e.g., North America). The focus here is on geographical containment.
  - **Prons**: Useful for structuring geographical knowledge and showing hierarchical containment (e.g., USA is part of North America). In addition, if our customer wants an ontology that needs geographical hierarchy, this is appropriate.

- **Crons**: It does not directly answer the query. The competence query is about associating movies with their production locations, not about the part-whole relationship between regions and countries. Furthermore, introducing part-whole relations unnecessarily complicates the model when all you need is a direct link between a movie and the country where it was produced.
- **General Comment**: This method adds unnecessary complexity. It's useful for geographical hierarchy but not suitable for linking movies to production locations.

- **3rd Class "Country" and an Instance "USA"**: This approach models "Country" as a class and creates individual instances (like "USA"). Each instance (like "Avatar") of the class "Movie" (that is an already defined as Class from the demo example in the class) can then have a relationship (like "producedIn") to a specific instance of the "Country" class (e.g. Avatar (being Instance of the class Movie) "producedIn" (relation) USA (being Instance of the class Country)). For accuracy purposes, with the term "Movie", we mean "a form of entertainment that enacts a story by sound and a sequence of images giving the illusion of continuous movement.", with the term "Country", we mean "A nation with its own government, occupying a particular territory.", with the relation "producedIn", we want to show in which country each film produced in. Furthermore, we can define attributes in classes (e.g. for the class Country, we can create the attribute "countryCode", meaning "The code of the country" with data type string e.g. for USA -> US. At this point, we should mention, that due to time limitations, we implemented quite few attributes for specific circumstances, or we did not add comments for all classes / relations in Protege, but we mention the idea. It is obvious that the Instances pf the class Country are names of Counties e.g. USA, Greece etc.
  - **Prons**: It is a straightforward and flexible approach. You can easily link a movie to the country where it was produced through a "producedIn" relation. In addition, new instances (of the class "Country") can be added without changing the structure of the ontology. Furthermore, good for scalability if more countries need to be added. Last but not least, we have query efficiency, as the query becomes efficient as you only need to look for all instances (e.g. Avatar) of "Movie" where the "producedIn" relation points to the instance "USA." This is a direct query that can be executed quickly, without requiring any reasoning or traversal of hierarchies.
  - **Crons**: We need to create and manage instances for each country, but this is a minor overhead compared to the flexibility it offers.
  - **General Comment**: This is the most suitable and flexible approach. By modeling "Country" as a class and USA as an instance, you can directly link a Movie with Instace e.g. Avatar to its production country using a relation like "producedIn". This allows clean ontology design while maintaining the flexibility to add more countries and relationships in the future.

- **4th Attribute in the Movie Class (e.g., country)**. We could add a "country" attribute directly to the "Movie" class, where the value of this attribute is "USA" (or other countries).
  - **Prons**: It is simple and straightforward. Directly associates each movie with a country through a simple attribute. It is fast to implement for small-scale use.
  - **Crons**: This approach lacks flexibility and scalability. If more complex relationships are needed later (e.g., if you want to link movies to multiple countries or **add properties to countries**), the model becomes limiting. For example, we should ask for client clarifications here, e.g. Do you want to store attributes for the countries?, if yes, then we should create a class Country and define the attributes that define it.
  - **General Comment**: This is a simple approach for small-scale use, but it lacks flexibility. It's not recommended for an extensible or future-proof ontology.

So, we decided to use the **3rd** alternative based on the aforementioned analysis of alternatives. In the box below, we present the prompt in ChatGPT that was used to assist with evaluating the trade-offs of potential design alternatives.

Prompt for the Competence Query: "Which movies have been produced in the United States?"

You are a designer / modeler of an ontology graph. I have the following competence query:

Which movies have been produced in the United States?

Shall I use:

- A hierarchy (is-a) relationship for the entity related to the USA, or
- A Broader/Narrower relation (part-whole relationship), e.g., *is part of* (continent: North America, country: USA), or
- A simple class *Country* with an instance of it being *USA*, or
- An attribute in the *Movies* class with the name *country*?

Please explain the given alternative directions or suggest any other alternatives you consider. Also, clearly outline the pros and cons (i.e., trade-offs) of each approach, and conclude with the alternative you would choose.

If you want to have acess to the dialogue, please visit the following link:

- https://chatgpt.com/share/670a4e78-7a88-800e-8b9d-d6da7089f50b

"Which movies are comedies and which are romantic comedies?"

- **1st Hierarchical (is-a) Relationship** We can model this request, with a **spuperclass** that is **"Movie"**, **two subclasses** that is **"ComedyMovie"** and **"RomanticComedyMovie"** and for these subclasses to create the Instances Emma and Love Actually respectively. For accuracy purposes, as "RomanticCommedyMovie" we mean all movies / films combining both comedy and romantic together, whereas as "CommedyMovies" we mean all movies / films contains solely comedy. Again, we can define attributes for both super class (for the common attributes being inherited by the sub classes) and sub classes (for the distinct attributes of each category of films). For example, for the sub class "RomanticCommedyMovie", we can have an attribute romance_level having data type string and possible values high , low. Furthermore, we can create Instance of the super class if we don't know the type of a Movie. It is obvious that the Instances of the superclass and subclasses are names of films / movies.
  - **Prons**: Offers clear symantics as hierarchical relationships make it explicit that a RomanticComedyMovie is a type of Movie, and it can naturally inherit attributes from the superclass. As a result, subclasses create clear, well-defined distinctions between categories. Another important advantage is that if we want to query all RomanticComedyMovie instances becomes straightforward as we can simply ask for all instances of the subclass RomanticComedyMovie, and the hierarchy will handle the inheritance structure efficiently.
  - **Crons**: A strict hierarchy might not reflect the fluidity between genres. A movie might fit into multiple categories (e.g., a romantic comedy that also fits into the drama category), and a hierarchical structure might struggle to represent this flexibility. In addition, regarding maintenance, Each new type of movie may require the creation of a new subclass. Over time, this can lead to excessive subclass proliferation, which could make the ontology harder to manage.
  - **General Comment**: This approach is suitable if the aim is to query e.g. **all** romantic comedies effectively.
- **2nd Simple Attribute in the Movie Class** We can model the request with the class Movies having an attribute genre, with values like "Comedy," "Romantic Comedy," etc. and the instances of the class Movie like Love Actually and Emma would have a genre property with values such as "Romantic Comedy" and "Romance," respectively.
  - **Prons**: It offers simplicity and flexibility. The Movie class can have a property like genre (or type) that holds values such as "Comedy," "Romantic Comedy," etc., and new genres can easily be added as property values without modifying the class structure. In addition, it provides flexibility for multi-genre Movies if a movies belongs to more than one genre. Furthermore, it recuses maintainance, there's no

need to create a new subclass every time a new genre or sub-genre is introduced. Last but not least, large movie databases may benefit from this approach because attributes are more efficient to manage and query compared to deep subclass hierarchies.

- **Crons**: Using attributes rather than a subclass hierarchy may reduce semantic clarity as there's less explicitness about what a "Romantic Comedy" really is, as it's just a label on the instance, not a formal subclass that defines the movie's relationship to other types. Regarding complex queries, if we want to query all comedies require additional effort. It also, it doesn't enables attributes inheritance.
- **General Comment**: If we are prioritizing simplicity and flexibility in managing different movie types, and you do not expect highly complex queries, the simple attribute-based approach might be more practical. It allows for easier management of genres, especially when movies span multiple genres.

We decided to use the **1st** alternative based on the aforementioned analysis of alternatives (key reason query all movies of a specific type or know which movies belong to type Romantic Comedy and which to Romantic category). In the box below, we present the prompt in ChatGPT that was used to assist with evaluating the trade-offs of potential design alternatives.

Prompt for the Competence Query: "Which movies are comedies and which are romantic comedies?"

You are a designer / modeler of an ontology graph. I have the following competence query:

Which movies are comedies and which are romantic comedies?

Please, provide the prons and cons (i.e. trade-offs) of the following design decision:

- Shall I use a hierarchy (is-a) relationship for the entity related to the type of movies. For example, superclass Movie, sublasses : CommedyMovies and RomanticMovies with Instances "Love Actually" and "Emma" respectively? or
- Shall I use a simple attribute like type in the class Movie?

I want to query all romantic comedies not a specific romantic movie. So, which is the best alternative?

If you want to have acess to the dialogue, please visit the following link:

- https://chatgpt.com/share/670a5f33-c918-800e-bc85-8e0408ece3e6

"What rating did a reviewer give to a given movie and what was their review text?"

- **1st We can create a new class "Review" that has attributes "rating" and "reviewText" and we can add the relationships: a Person (class) has a "submittedFor" relationship with Review (class), that is "Person –submittedFor–> Review", and a Review (class) has "evaluated" relationship with "Movie" (class), that is "Review–Eavluated–>Movie".**
  - **Prons**: It offers flexibility as we can easily extend the Review class with more properties (e.g., date of review, likes, comments, etc.). in addition, it decouples the Person and Movie classes, allowing multiple reviews for the same movie by different reviewers or even reviews of multiple types (text, video). Furthermore, it can handle complex queries (e.g., find all reviews by a person, filter by rating, etc.) since the data is structured separately in the Review class.
  - **Crons**: It increases the complexity, as the ontology becomes more complex with an additional class (Review), requiring more maintenance and reasoning. Furthermore, querying becomes slightly more complex, as you'll need to traverse multiple relationships.
  - **Hint**: This approach is unavoidable if we want to represent attributes in relations using Protégé.
  - Note: For the class Person we will need an attribute to show if that Person is reviewer (it is analyzed in the next sections).
- **2nd Create a Direct Relationship Between Person and Movie Classes**. We can create a direct object relation between the Person (representing the reviewer) and Movie classes, such as **"submittedReviewFor"** (for best practices in its name, we used **For** as this indicates the target of the review — the object being reviewed in our case it implies the Movie) with the associated rating and review text as data properties on this relationship. More specifically, the new relation "submittedReviewFor" implies that a person (e.g. Evangelia

Panourgia being instance of the class Person), and assuming that she is reviewer, submitted a review in a movie (e.g. Avatar being an Instance of the class Movie).

- **Prons**: It offers simplicity as it is a straightforward model where you directly connect Person to Movie via a relationship, and add properties like rating (data type: number) and reviewText (data type: string) on this relationship. It is easy for qyuring purposes.
- **Crons**: Lacks of flexibility as it could be harder to extend this model later with additional properties, like review date, review platform, etc. Furthermore, it has limited extensibility as if we want to expand with more complex review data (e.g., multiple reviews per movie, review metadata, or co-reviews), this model becomes restrictive.
- **General Comment**: This approach is good if we want a simple design, if we finally select it, in Protégé, we will need to do a "trick" because it doesn't support attributes in relations (see: **1st** approach for this competence question).

- **Food for thought** Introducing a Reviewer subclass for the class Person: In this approach, we can define a new subclass Reviewer that inherits from the Person class. Only individuals classified as Reviewer would be explicitly modeled as those capable of providing reviews, while all other people remain instances of the general Person class. **Prons:** semantic clarity as it offers clear role distinction and easy filtering as It becomes easier to query specifically for people who are reviewers, as you have a distinct class representing them. **Cons**: it provides unnecessary complexity as if our ontology is small or doesn't anticipate significant differences between Person and Reviewer, introducing a Reviewer subclass may overcomplicate the model. Simply, an additional **attribute (e.g., reviwerStatus and for reviwers accepting the value is_reviwer)** on the Person class might suffice in such cases without needing a distinct subclass.
- To maintain flexibility and simplicity in the ontology, we will choose the **2nd** option. Additionally, for the purposes of this assignment, we will use this design pattern to demonstrate in the following sections how we will implement attributes and relationships across different software platforms. In Protégé, for example, we will handle this by introducing an additional class. Regarding the potential inheritance (meaning a potential subclass Reviewer of the super class Person), we decided not to use it, instead of that, we decided to add an **attribute** that is (**reviewerStatus**) so as to define if a Person is reviewer. This is because of the fact that we assumed that the attributes of a Person with a Reviewer doesn't different a lot.
- **Note**: For this query, we assume that we want to see a specific reviewer e.g. Evangelia Panourgia (representing a reviewer) the rating and review text that she submitted for a specific movie e.g. Avatar (it is discussed below in the section related to implementation of queries).

Prompt for the Competence Query:What rating did a reviewer give to a given movie and what was their review text?

You are a designer / modeler of an ontology graph. I have the following competence query:

> *What rating did a reviewer give to a given movie and what was their review text?*
>
> I have already a class Person and a superclass class Movie having as subclasses CommedyMovie and RomanticCommedyMovie.
>
> Please, provide the prons and cons (i.e. trade-offs) of the possible ways of design.

Subsequent Prompt for the aforementioned competence question.

Shall I add hierarchy (is a) in Person class with sublclass Reviewer?
List please prons and cons.

If you want to have access to the dialogue, please visit the following link:

- https://chatgpt.com/share/670a9b41-8e88-800e-b5d3-621826aa88da

Before proceeding with the implementation of our design decisions, we will first provide a short **summary** of them:

Summary: Modeling Decisions for the Ontology.

- **"Which movies have been produced in the United States?"**
  - **Modeling Decision**: Add **Class "Country"** and an **Instance "USA"** and **relationship "producedIn"** representing that a Movie e.g. Avatar (being Instance of the class Movie) "poducedIn" a Country (like USA). Representation $Avatar--producedIn-->USA$.
- **"Which movies are comedies and which are romantic comedies?"**
  - **Modeling Decision**: Hierarchical (is-a) Relationship: a spuperclass that is Movie, **two subclasses** that is **ComedyMovie** and **RomanticComedyMovie** and for these subclasses to create the Instances **Emma** and **Love Actually** respectively.
- **"What rating did a reviewer give to a given movie and what was their review text?"**
  - **Modeling Decision**: Create a Direct Relationship Between Person and Movie Classes: **"submittedReviewFor"** with relation attributes **rating** and **review text**..
  Representation: $EvangeliaPanourgia--SubmittedReviewFor-->Avatar$,
  where EvangeliaPanourgia is Instance of the class Person and Avatar is Instance of the super class Movie. Furthermore, we decided and additional attribute for defining if an Instance of Person is reviewer or not, for this purpose we defined the attribute **reviwerStatus**, if reviwerStatus equal to $is\_reviewer$ (potential value) then the instance is reviewer. **Hint**: For Protégé we are obliged to create an additional class.

Prompt Strategy

For the scope of this assignment, we utilized ChatGPT-4 to assist us with both the modeling process and SPARQL queries (as discussed in Section 2). Although competence questions should not be considered in isolation (based on Professor's guidance that is "Consider the competency questions all together, not individually."), for guiding ChatGPT, we addressed each competence query individually. In other words, we employed a "divide and conquer" strategy ("διαιρεί και βασίλευε") to systematically approach each query. Using our judgment, we took into account the overall picture to ensure that each design decision did not adversely affect the other competence queries. It is important to be mentioned we tried to follow the **Six strategies for getting better results** offered by the official website of *OpenAIPlatform* [1]. To be more precise, we included details in our queries so as to get more relevant answers (e.g. on our owns we expressed possible design alternatives), we asked the model to adopt a persona (e.g. **You are a designer / modeler of an ontology graph**), we used **delimiters** in order to clearly indicate distinct parts of the input (e.g. we used many times the symbol : - (dash) or **""** (double quotes)), also, we provided examples (e.g. in the section for SPARQL we provide specific examples regarding the dummy data and the returned data), and regarding the latter one that is the returned data of the query, we applied the best practice of specifying the desire output (e.g. We explicitly specified the column names and the expected data we were anticipating to receive). Last but not least, some times we needed to re-ask ChatGPT for clarifications or corrections to specific parts of its answers.

Task 1 (Neo4j)

"Which movies have been produced in the United States?"

- We created Instances USA and Avatar for the classes Country and Movie (for the super class for simplicity) respectively.
- We created the relationships Avatar isInstanceOf Movie, USA isInstanceOf Country and between the Instances Avatar ProducedIn USA.

```
// Create the Country class and USA instance
CREATE (:Country {name: 'Country'});
```

```
CREATE (usa:CountryInstance {name: 'USA'})-[:isInstanceOf]->(:Country {
    name: 'Country'});

// Create the Movie class and Avatar instance
CREATE (:Movie {name: 'Movie'});
CREATE (avatar:MovieInstance {name: 'Avatar'})-[:isInstanceOf]->(:Movie
    {name: 'Movie'});

// Create the relationship indicating that Avatar was produced in the
    USA
MATCH (usa:CountryInstance {name: 'USA'}), (avatar:MovieInstance {name:
    'Avatar'})
CREATE (avatar)-[:producedIn]->(usa);
```

Listing 1: Construction of the graph Movies Produced IN USA.

– The query returns the movies produsedIn USA is:

```
MATCH (movie:MovieInstance)-[:producedIn]->(country:CountryInstance {
    name: 'USA'})
RETURN movie.name AS Movie
```

Listing 2: Retrieve the Movies Produced IN USA.

– The output of our query returns:

```
+------------------+
|      Movie       |
+------------------+
|  "Avatar"        |
+------------------+
```

Listing 3: Query Output

"Which movies are comedies and which are romantic comedies?"

– We constructed the Instances **Love Actually** and **Emma** for the sub classes **RomanticComedyMovie** and **ComedyMovie**, respectively. The super class of the aforementioned ones is Movie. Furthermore, for demonstration purposes, we added some attributes distinguishing the types of movies in sub classes. Here's the Cypher query to model that structure:

```
// Create the Movie superclass node
CREATE (:Movie {category: 'Movie'})

// Create the subclass nodes with distinguishing attributes

// Romantic Comedy subclass
CREATE (:RomanticComedy {category: 'Romantic Comedy', romance_level:
    'high', humor_style: 'witty', relationship_focus: 'multiple
    relationships'})

// Comedy subclass
CREATE (:Comedy {category: 'Comedy', humor_style: 'dry humor',
    setting: 'period'})

// Create movie instance nodes

// Love Actually as an instance of Romantic Comedy
CREATE (:MovieInstance {title: 'Love Actually', release_year: 2003,
    duration: 135})
```

```
16
17 // Emma as an instance of Comedy
18 CREATE (:MovieInstance {title: 'Emma', release_year: 1996, duration:
       121})
19
20 //change executable cell
21
22 // Establish relationships
23
24 // Romantic Comedy as a subclass of Movie
25 MATCH (m:Movie {category: 'Movie'})
26 MATCH (rc:RomanticComedy {category: 'Romantic Comedy'})
27 CREATE (rc)-[:SUBCLASS_OF]->(m)
28
29 //change executeble cell
30 // Comedy as a subclass of Movie
31 MATCH (m:Movie {category: 'Movie'})  // Rematch to ensure it's part
       of WITH
32 MATCH (c:Comedy {category: 'Comedy'})
33 CREATE (c)-[:SUBCLASS_OF]->(m)
34
35 //change executable cell
36 // Love Actually as an instance of Romantic Comedy
37 MATCH (la:MovieInstance {title: 'Love Actually'})
38 MATCH (rc:RomanticComedy {category: 'Romantic Comedy'})  // Match
       again to ensure relationship
39 CREATE (la)-[:INSTANCE_OF]->(rc)
40
41 //change executable cell
42
43 // Emma as an instance of Comedy
44 MATCH (e:MovieInstance {title: 'Emma'})
45 MATCH (c:Comedy {category: 'Comedy'})  // Match again to ensure
       relationship
46 CREATE (e)-[:INSTANCE_OF]->(c)
```

Listing 4: Creating Nodes (based on hierarchy) and Relationships among Movies and their categories.

– For retrieving the Movies being Romantic Comedies and Comedies, showing their type, we wrote the following Cypher query:

```
1 // Retrieve the movies and their categories (Comedy or Romantic
      Comedy)
2 MATCH (mi:MovieInstance)-[:INSTANCE_OF]->(subclass)
3 RETURN mi.title AS Movie,
4        CASE
5           WHEN subclass:Comedy THEN 'Comedy'
6           WHEN subclass:RomanticComedy THEN 'Romantic Comedy'
7        END AS Category
```

Listing 5: Cypher Query

– The output of the query is:

```
+-----------------+---------------------------+
|     Movie       |     Category              |
+-----------------+---------------------------+
|  "Love_Ctually" |     "Romantic Commedy"    |
+-----------------+---------------------------+
|  "Emma"         |     "Commedy"             |
+-----------------+---------------------------+
```

Listing 6: Query Output

– As an additional step, we can visualize this hierarchy:

```
// Retrieve the movies and their categories for visualization,
   showing the names of the subcategories
MATCH (mi:MovieInstance)-[:INSTANCE_OF]->(subclass)
RETURN mi,
       CASE
          WHEN subclass:Comedy THEN 'Comedy'
          WHEN subclass:RomanticComedy THEN 'Romantic Comedy'
       END AS category
```

Listing 7: Visual our Graph of Hierarchy

– For appearance purposes, we customize our plot, via adapting the node size etc.

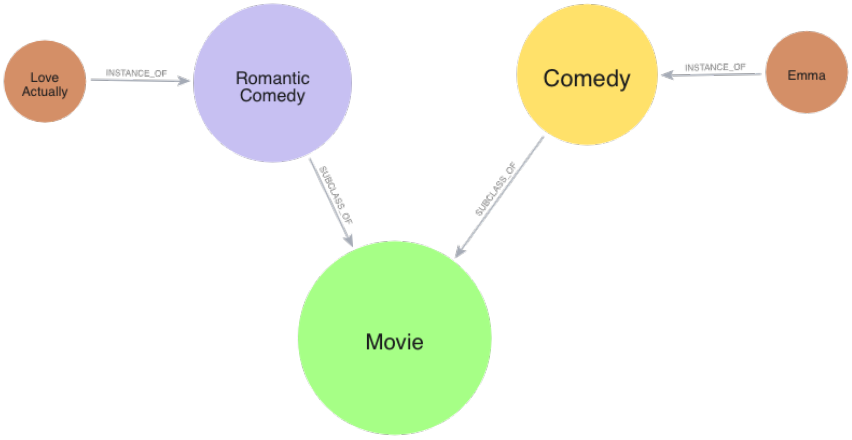

Figure 1: Visualization of Movies and their Categories (Comedy, Romantic Comedy)

"What rating did a reviewer give to a given movie and what was their review text?"

– We assume, we want to fetch / retrieve **a specific Reviewer** e.g. Evangelia Panourgia (checking firstly if she is a reviewer using the associated Person's attribute) and we want to see her rating and text for a **specific Movie** e.g. Avatar.

– We created the Intance EvngeliaPanourgia for the class Person with reviewer status (that is the attribute), where reviewerStatus: "is_reviewer".

– We created an Instance Avatar of the class Movie.

– We created the **relationship submittedReviewFor with attributes rating and reviewText**

– $EvangeliaPanourgia --submittedFor-> Avatar$

```
  // Create the Person node with attributes
```

```
CREATE (p:Person {name: "EvangeliaPanourgia", reviewerStatus: "
    is_reviewer"})

// Create the Movie node
CREATE (m:Movie {title: "Avatar"})

// Create the Person class node (if it doesn't already exist)
MERGE (classPerson:Class {name: "Person"})

// Create the Movie class node (if it doesn't already exist)
MERGE (classMovie:Class {name: "Movie"})

// Create the relationship EvangeliaPanourgia isInstanceOf Person
CREATE (p)-[:isInstanceOf]->(classPerson)

// Create the relationship Avatar isInstanceOf Movie
CREATE (m)-[:isInstanceOf]->(classMovie)

// Create the review relationship EvangeliaPanourgia submitted a
    review for Avatar
// with rating and review text attributes
CREATE (p)-[:submittedReviewFor {rating: 5, review_text: "Fine film
    !"}]->(m)
```

Listing 8: Construction of example for attributes in relation.

– To retrieve the review text and rating that Evangeleia Panourgia submitted for the movie Avatar:

```
  MATCH (p:Person)-[r:submittedReviewFor]->(m:Movie)

  WHERE p.name = "EvangeliaPanourgia" AND m.title = "Avatar"

   RETURN p.name AS reviewer, m.title AS movie, r.rating AS rating, r
      .review_text AS review_text
```

Listing 9: retrieve data for the example for attributes in relation.

– The output of the query is:

```
+------------------------+-----------+--------------+------------+
|     reviwer            |     rating|  text_review |  movie     |
+------------------------+---------------------------+------------+
|  "EvangeliaPanourgia" |       5   |  "Fine film!"|   "Avatar" |
+------------------------+---------------------------+------------+
```

Listing 10: Query Output

Task 2 (Protégé)

"Which movies have been produced in the United States?"

– We created a new **class** with name : **Country** (tab: Entities->Classes->Add subclass (being subclass of the root). For localization purposes, we added from Annotations tab **rdfs:label** for example Country (for English language, en) and Χώρα (for Greek language, el). Furthermore, we add also from Annotations tab **rdfs:comment** so as to express what we mean with the term country in this ontology.

– We created a new **object property (relation)**, we define domain (Movie) and Range (Country), and we give an rdfs:comment.

– We created an **Instance USA** of the class Country (Individual by Class->Country->Add individual). Furthermore, for query purposes we added an Instance in Movie e.g. Avatar. Then for the instance Avatar we created an object property assertion that is PRODUCED_IN USA.

Having created a simple use case with the necessary data elements, we attempted to retrieve data relevant to our competency query. To assist with this, we also utilized ChatGPT.

Prompt for Retrieving the Competence Query:"Which movies have been produced in the United States?"

I use Protege. I have the following elements in my ontology:

- a class Movie, and an insance of this class Avatar
- a (super)class Country and and for this class an instance USA
- a relationship PRODUCED_IN with Domain: Movie and Range: Country. and a relatioship between Avatar PRODUCED_IN USA (between the Instances)

Please, write SPARQL query in order to answer the following competence query:

"Which movies have been produced in the United States?"

In other words, in the specific ontology I am waiting to return the instance Avatar, as this movie produced in USA.

In SPARQL we have the pre-defined code:

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?subject ?object
WHERE ?subject rdfs:subClassOf ?object

Please guide me step-by-step, as I am beginner in protege and as a result in SPARQL query.

If you want to have access to the dialogue, please visit the following link:

– https://chatgpt.com/share/670b9a4d-41e0-800e-b320-070cac288bb6

The following SPARQL query retrieves all movies produced in the USA:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX : <http://neo4j.com/voc/movies#>  # Replace this with your actual
    ontology's namespace

SELECT ?movie
WHERE {
    ?movie rdf:type :Movie .            # Get all instances of Movie
    ?movie :PRODUCED_IN :USA .          # Restrict to those produced in
       USA
}
```

Listing 11: SPARQL Query to Find Movies Produced in the USA

The query returns the following result:

```
+-----------------+
|      movie      |
+-----------------+
|   :Avatar       |
+-----------------+
```

Listing 12: Query Output

"Which movies are comedies and which are romantic comedies?"

– We created a superclass named Movie and two separate subclasses called ComedyMovie and RomanticComedyMovie. Instances of these subclasses will be: Love Actually, an instance of RomanticComedyMovie and Emma, an instance of ComedyMovie.

Prompt for Retrieving the Competence Query:"Which movies are comedies and which are romantic comedies?"

I use Protege. I have the following elements in my ontology:
• a class Movie, and an insance of this class Avatar
• in adition, for the (super)class Movie, we have the subclasses that is ComedyMovie and Romantic-ComedyMovie. Instances of these subclasses will be: Love Actually, an instance of RomanticComedyMovie and Emma, an instance of ComedyMovie.

I have the competence query:
"Which movies are comedies and which are romantic comedies?"

So, for *RomanticComedyMovie*,I want to return the query all movies belong to this category, in our case the instance *Love_Actually* and for *ComedyMovie* the instance Emma.

In SPARQL we have the pre-defined code:
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?subject ?object
WHERE ?subject rdfs:subClassOf ?object

Please guide me step-by-step, as I am beginner in protege and as a result in SPARQL query, and provide me the query aswering to the provided competence query.

The following SPARQL query retrieves movies along with their respective categories, displaying the category to which each movie belongs. If we want to view the movies belonging to the one category e.g. Romantic-Commedies, we can remove the UNION operator with its block of code for the type of category we don't want in out output result.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX mo: <http://neo4j.com/voc/movies#>


SELECT ?movie ?category
WHERE {
  {
    # Query for instances of ComedyMovie
```

```
    ?movie rdf:type mo:ComedyMovie .
    BIND("ComedyMovie" AS ?category)
  }
  UNION
  {
    # Query for instances of RomanticComedyMovie
    ?movie rdf:type mo:RomanticComedyMovie .
    BIND("RomanticComedyMovie" AS ?category)
  }
}
}
```

Listing 13: SPARQL Query to Find Movies Produced in the USA

The query returns the following result:

```
+-----------------+---------------------------+
|     Movie       |     Category              |
+-----------------+---------------------------+
|  :Emma          |     "CommedyMovie"        |
+-----------------+---------------------------+
|  :Love_Ctually  |     "RomanticCommedyMovie"|
+-----------------+---------------------------+
```

Listing 14: Query Output

If you want to have access to the dialogue, please visit the following link:

- https://chatgpt.com/share/670ba862-51b0-800e-9030-f09737e27957

What rating did a reviewer give to a given movie and what was their review text?

- For this modeling we decided that alternative related to the creation of a **Direct Relationship** Between **Class: Person** and **Class Movie** Classes: the direct object property (relation) between the Person and Movie classes named as **"submittedReviewFor"** with attributes **rating** and **text_review**. At this point, it is crucial to be pointed out **Protégé don't support** the operation of **attributes in relations**. To overcame this difficulty, we followed the strategy: we created a new **class MovieReview** (the name is descriptive via specializing the Review that is for the Movies) and for this class we added two **attributes**: **rating** (with values e.g. integer number ranging from 0 to 5; this **must** be defined as comment) and **text review** (with values being data type of string). Furthermore, we needed a new relation **evaluated** connecting the class Review with the class Movie ($MovieReview --Evaluated-> Movie$). In addition, with purpose to identify if an instance of the class Person is reviewer, in the class Person we added an **attribute reviewerStatus** with values:

- **is_reviewer** (where is_reviewer: The person is currently a reviewer) and
- **not_a_reviewer** (where not_a_reviewer: The person is not a reviewer.)

and we added a relation **submittedFor** connecting the class Person with the new Class Review ($Person --submittedFor-> MovieReview$).

Pragmatically implementation:

- For the attribute **reviewerStatus** of class Person : Entities->DataProperties (Define Annotations->comment so as to show what it represents and the defined values, Domains:Person and Range:xsd:string (str))

- For the new class **MovieReview** (Entities->Classes) and for its attributes (Entities->DataProperties add rating and text_review with data types int and string respectively. For the former, we added the range of numbers (0..5) explaining each number what means).
- For the two relations we have: **SUBMITTED_FOR** (Domain: Person -> Range: MovieReview) and **EVALUATED** (Doamin: MovieReview -> Rnage: Movie).
- For the Instances, we created an Instance for the class Person named **EvangeliaPanourgia**, and for this instance we added value is the attribute **reviwerStatus**, the value is **is_reviewer** (Data Property assertions), in addition, we created an Instance for the class MovieReview named **AvatarReview** (related to the review of the Instance of the Superclass Movie that is Avatar) and we add for this instance in the attributes rating and text review the values 5 and "The best movie! I recommend it to all!" respectively. Furthermore, we created an object property assertion for the Instance EvangeliaPanourgia (SUBMITTED_FOR AvatarReview) and for the Instance AvatarReview we created object property assertion (EVALUATED Avatar).

Prompt for Retrieving the Competence Query:"What rating did a reviewer give to a given movie and what was their review text?"

I use Protege. I have the following elements in my ontology:

•a class Person with attribute *reviewerStatus*, for this class we have an instance *EvangeliaPanourgia* with value is_reviewer for the attribute *reviewerStatus*.
•a class *MovieReview* with attributes text_review (data type string) and rating (data type decimal) for this class we have an Instance *AvatarReview* with values "the best movies! I recommend it to all!" and 5 respectively. In addition, for the Instance *AvatarReview* We have defined Object property assertions : EVALUATED Avatar.
•the relation connects Person and MovieReview names SUBMITTED_FOR.
•a class Movie with Instance Avatar.
•the relation connects class MovieReview with class Movie is EVALUATED.

I have the competence query: "What rating did a reviewer give to a given movie and what was their review text?"

In our case, we assume that EvanageliaPanourgia represent a reviewer and the given movie is Avatar.

I am waiting to return EvangeliaPanourgia gave in avatar the following values : 5 (rating) , "The best movie! I recommend it to all!"

In SPARQL we have the pre-defined code for prefixes:
PREFIX rdf : < http : // www . w3 . org /1999/02/22 - rdf - syntax - ns# > 2
PREFIX owl : < http : // www . w3 . org /2002/07/ owl# > 3
PREFIX rdfs : < http : // www . w3 . org /2000/01/ rdf - schema# > 4
PREFIX xsd : < http : // www . w3 . org /2001/ XMLSchema# > 5
PREFIX mo : < http : // neo4j . com / voc / movies# >

Please guide me step-by-step, as I am beginner in protege and as a result in SPARQL query, and provide me the query aswering to the provided competence query.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX mo: <http://neo4j.com/voc/movies#>

SELECT ?reviewer ?rating ?text_review ?movie
WHERE {
  # Find the person with reviewer status 'is_reviewer'
  ?reviewer mo:reviewerStatus "is_reviewer" ^^xsd:string.

```

```
  # Get the review submitted by this person
  ?reviewer mo:SUBMITTED_FOR ?review .

  # Get the rating and text review from the movie review
 ?review mo:rating ?rating .
  ?review mo:text_review ?text_review .

  # Ensure that the review is associated with a movie
  ?review mo:EVALUATED ?movie .

  # Filter for EvangeliaPanourgia as the reviewer
  FILTER (?reviewer = mo:EvangeliaPanourgia)

  # Filter for Avatar as the movie
  FILTER (?movie = mo:Avatar)
}
}
```

Listing 15: SPARQL Query to Find Rating and Text gace EvangeliaPanourgia to Avatar movie

The query returns the following result:

```
+------------------------+-----------+-------------+-------------+
|     reviwer            |     rating|  text_review|  movie      |
+------------------------+-------------------------+-------------+
|  :EvangeliaPanourgia   |    (*)[1] |  (*)[2]     |   Avatar    |
+------------------------+-------------------------+-------------+
```

Listing 16: Query Output

(regarding (*****) of listing query output) The following bullet points presents the whole values which are returned for the columns rating and text_review.

(Note: Due to appearance purposes we selected to symbol them with star (*))

- [1] : "5"|$http://www.w3.org/2001/XMLSchema\#decimal>$
- [2] : "The best movie! I recommend it to all!"|$http://www.w3.org/2001/XMLSchema\#string>$

Note: after the values 5 , "the best movie ..." there is a string related to data type of the attribute value.

Future Work

Due to time constraints, we did not add attributes for the subclasses of Movie in Protégé, nor did we include detailed descriptions for each element. Additionally, we did not create a large set of dummy data. Our primary goal was to explore various design alternatives and present them through small-scale examples. For improved demonstration purposes, we would address these aspects in future iterations.

References

[1] OpenAI. *Six Strategies for Getting Better Results from GPT*. Available at: `https://platform.openai.com/docs/guides/prompt-engineering/six-strategies-for-getting-better-results`. Accessed: October 14, 2024.