Streamultis

Steven Emmink, Elise Vink, Yürri Arnold, and Renske Diependaal

Vrije Universiteit Amsterdam, De Boelelaan 1105, 1081 HV, Amsterdam, The Netherlands

1 Description of the Application and Users

In this section, we will describe the goal of the application and the expected users. The name of the application is: ”Streamultis” or in short: ”SM”. The application will be a support application that helps people in making a choice out of certain stream services based on filters. In The Goal of the Application segment, we will elaborate on the specific aim of the application. We will also explain what tasks the application will perform and what it exactly does. In The users of the application segment, we will speculate on what kind of users we expect will use the application. Moreover, the specific need the application satisfies will be explained in detail.

1.1 The Goal of the Application

The main goal of the application Streamultis is to give more information about streaming services to help the user decide which streaming service best fits their needs. Another option could be to ask the application how many movies are suitable for children within a streaming service., the application is able to show how many movies are suitable for underage people, or which database and thus streaming service has the most horror movies. It is also possible to compare the services and ask for an overview showing the number of series and/or movies per service. Additionally, it will also be possible to search for a series and/or movie and ask which streaming service offers this series/movie. It is also possible to search for a movie with a certain actor or age restriction. In short, the application allows the user to compare streaming services based on several properties, for example type of series/movies, price per month, the number of movies/series with or without a certain restrictions.

1.2 The Users of the Application

The expected users of our application are people who require more assistance with choosing a particular streaming service. As streaming services have expanded rather quickly over the last decade, offering a continuously growing library of movies and series, it could be quite difficult for people to decide which streaming service they should choose. Moreover, streaming services nowadays also promote their brand using exclusive titles, making it more difficult to choose as people might want to watch a certain series that is only available on one specific streaming service. It could be quite confusing and tedious work to search whether a streaming service offers that specific series the user would like to watch. Additionally, there might be (more) other titles the user would like on a different streaming service. Streamultis will be an application that offers consultancy for those who are agonizing on which streaming platforms to pay a subscription for. As such, the goal of our application is to support the user to choose the best suitable streaming service. They could access the application via the Web, using a tablet, smartphone or computer.

2 Design and Walkthrough

In this section, we will describe the Design of the Application and the Interaction with the Application. In the Design of the Application segment, we will explain in detail how the application will be designed and how the user will perceive the application. In the Interaction with the Application segment, we will explain how the user would be able to interact with the application.

2.1 Design of the Application

In the application, the user would be able to select filters in order to see results bound to these restrictions. To make the results easier to interpret, we categorise all provided filters on the left side and all the results on the right side of the page shown to the user. On the left side, the user will be able to select filters concerning their interests. The options of the filters will not be shown directly, to conserve overview of the possible filters. Instead, a drop-down menu will be implemented for some filters. In the drop-down menu you can select one or multiple options, depending on the filter. On the right side, the results will be displayed dynamically, restricted by the selected filters on the left side. The filters on the left side will differ in format. Some will be a search bar, to search for a movie name for example. Others will allow the user to select multiple options, like the genre filter. Users are opted to select movies/series that are either in for example the genre horror or in the genre drama. Some filters will only allow one option, such as the age restriction filter. The data presented on the right side will be shown in the form of tables, this will give a clear, structured overview of all the results. Each streaming service will have their own table with the first 5 results that can be watched on that streaming service, bound to the restrictions of the filters. Clicking on the 'GO'-button on the filters-section will result in the tables being filled with the first 5 results. The name of the service will be displayed at the top of the table, all movie- and TV show titles will be shown right under each other. Then all (requested) details of these shows will be shown next to each other, right under their dedicated table header. These design choices were made to create an understandable, clear and easy-to-use web application. As all filters and results will be structured in a neat way using tables and sections, the learnability of the application will be greatly improved. Moreover, its efficiency will be satisfactory, as users will be able to quickly search what they would like to see. As the application will use very simple, understandable interaction with the user, its memorability will also be great. Additionally, errors such as a wrongly chosen filter will easily be undone by clicking it again to remove the filter. All these decisions will, hopefully, lead to satisfactory usage of the application.

2.2 Interaction with the Application

The application will work as follows: there will be a drop-down menu on the left side of the web application and within this menu the user is able to select for each filter: genres, age restrictions or year produced, if they wish to do so. Otherwise they can search for a movie- or show title. The presented result will show on which streaming service this is available and what rating these movies or shows have, neatly presented in tables. The user is also able to see which streaming service offers the most movies or shows via a different method, for example: we will make an option that shows this amount or through an empty search. Each interaction or filter the user makes or adds will change the results dynamically, showing for the filter which streaming service offers what. The user is able to change or delete the filters that have been made. Should the tables exceed the given surface, then we will show the top five results and the user is able to scroll through these tables to look at the remaining results.

3 Domain and Conceptualization

The domain we have chosen is streaming services. We use the information about movies and TV shows such as actors/actresses and age restrictions to determine which streaming service would be best for the user of our application. The ontology is created by firstly, creating the most obvious classes that are needed for the project and then the classes needed for filtering the movies or TV shows. The properties are created by looking at which properties are needed to use all the classes properly.

3.1 Domain and Scope of the Ontology

The goal of our web application is to help users choose a streaming service that best fits their needs. This means that the domain is media, more specifically a domain in movies/series and streaming services. We focus on the titles, their genres, the artists, and other properties such as age restriction. We use these properties to filter the movies and series a certain streaming service has using the unions and intersections, as the selected filters should all apply to the result. The result would be a list of moves/series that conform to the selected filters of the user.

3.2 Methodology of the Ontology

For the ontology, we used the top-down methodology. So firstly, we started with creating the classes and properties of our domain. Because the applications shows which movies/TV-shows are on different streaming services, we created the class StreamingService, Movie and TVshow. We also want the user to be able to filter movies to, for example, see how many comedy shows there are on each streaming service. Classes such as Genre, Actor/Actress and AgeRestriction are necessary for filtering the movies and TV-shows. Then, we made the properties for which we looked at the classes. We created the properties according to those classes. So, for example, for the class StreamingService we created the property hasStreaming Service and for the class AgeRestricion we made the property hasAgeRestricion. The instances we made our self were for classes which did not have many instances. An example of this is the class StreamingService which has only four instances (Netflix, Hulu, PrimeVideo and Disney+). It was not doable to make all the instances for the bigger classes such as Actor/Actress, Movie and TVshow so we used the databases to create these classes.

3.3 Conceptualization of the Domain

We have made the classes based off of everything we need to filter movies and TV shows within multiple streaming services. The subclasses all are part of the main class Genre or Person. We used the same categorization in genre as used on the IMDb website. This makes it easier for us to integrate the IMDb database because all the genres are similar. The classes and properties are shown below:

Classes:

- StreamingService
- Movie
- TVSeries
- AgeRestriction
- Person
  - Actor
  - Actress
- Genre
  - Action
  - Adult
  - Adventure
  - Animation
  - Biography
  - Comedy
  - Crime
  - Documentary
  - Drama
  - Family

- Fantasy
- Film Noir
- Game Show
- History
- Horror
- Musical
- Music
- Mystery
- News
- Reality-TV
- Romance
- Sci-Fi
- Short
- Sport
- Talk-Show
- Thriller
- War
- Western

Properties:

– rdf:type
– rdfs:subClassOf
– stream:Director
– stream:hasActor
– stream:hasAgeRestriction
– stream:hasDirector
– stream:hasDirectors
– stream:hasGenre
– stream:hasGenres
– stream:hasIMDbRating
– stream:hasIMDBRating
– stream:hasNameID
– stream:hasPrimaryProffesion (to differantiate actors, directors, writers, etc. from the database)
– stream:hasRatingIMDB
– stream:hasStreamingService
– stream:hasTitleID
– stream:hasTitleType (movie or serie)
– stream:hasTomatoesRating
– stream:producedYear
– stream:RatingIMDB

Data properties:

– birthname
– title

4 Ontology and Inferencing

In this section we discuss and show our made ontology. In the Reusable Ontologies section, we show what ontologies we found and how we want to use them. In Ontology Conceptualization in OWL we discuss how we want to model our ontology, in Integration of External datasets we show how we integrated the different ontologies and lastly in Meaningful Inferences we show what we have inferred.

4.1 Reusable Ontologies

We found an ontology which is already related with data from IMDb: movieontology.owl. Since we will also use data from the IMDb database, this ontology will be easier to use in combination with the IMDb data than another movie ontology. The IMDb database is discussed further in the next section. It would be ideal if there was an ontology on streaming services which we could use. Sadly, we were not able to find a suitable and useful ontology on the internet. Therefore, we will use DBO vocabulary as the second ontology. This vocabulary contains a lot of general terms but also some movie terms which would be useful for this project.

4.2 Ontology Conceptualization in OWL

For importing the movie-ontology and the DBO ontology, we directly imported it with the .owl file. Some classes and properties have been removed from the movieontology. This is because many of these are not needed for this specific project. For example, classes like Territory and Awards have been removed. These properties did not seem like relevant factors for the user in choosing a streaming service. The properties that are related to these classes have also been removed. The movie-ontology we re-used did also have useful classes such as Actor and Actress. From the DBO ontology we could re-use dbo:People. We aligned Actor and Actress to be subclasses of People. Our own added classes are StreamingService, AgeRestriction and Genre. We added the class StreamingService to easily give an overview of which movies and series are available in a certain streaming service. AgeRestriction can be used by the user to decide if a streaming service has enough movies suitable for their children for example. The movie-ontology did have a class Genre but all the subclasses have been changed by us to the ones IMDb uses to categorise movies. We did this to align the two ontologies. Next, we saved the ontology in Turtle format and uploaded it to a GraphDB repository.

4.3 Integration of External Datasets

Our datasets were originally in .csv or .tsv format. We used the OntoRefine tool offered in GraphDB to transform these files to RDF. We did this by constructing several triples and creating classes for the instances that did not yet have a class in our original ontology. We made sure that the triples we created were mostly aligned with the triples from our ontology. An example of the kind of triples we created is stream:Title stream:hasTitleID stream:titleID. We had to create a new class for titleID's, since the IMDb Datasets had this unique identifier to ensure that data about movies and series can be combined with their other databases. The IMDb database consists of several datasets, containing information about names, movies, series, actors, etc. These databases were too big for the OntoRefine tool to read at once, often resulting in errors or estimated running times of 1300 mins. For this reason we decided to try and split the databases and uploaded them this way. We also uploaded a four Kaggle databases in our repository, see the subsection External Sources below. These were less big and were uploaded in one go.
Then we had to align a few instances and predicates with the ontology and IMDb Database, such as AgeRestrictions. We could edit the values in the column and any alignments we missed, we were able to fix with a Sparql query in graphDB. After uploading both our databases and ontology in graphDB, we started to mix them to create a comprehensive database, containing information about series and movies offered on the streaming services mentioned earlier. For instance, we did not need all the movies and series and their information from the IMDb database, just the movies and series from which we knew were also offered on one of the streaming services.

4.4 Meaningful Inferences

Using the titles from movies and series in the Kaggle Databases, we were able to match the IMDb databases with the information from Kaggle. This means that we were able to see actors and directors from movies on Netflix, for example.

5 Data Reuse

In this section we describe which sources we have used and how we intend to use them.

5.1 External Sources

We would like to use the IMDb database (https://www.imdb.com/interfaces/) to create a list of movies and series and their properties, such as directed by John Smith or age restrictions. Furthermore, we need to know which streaming services offer which shows and movies. Initially, we wanted to use two databases we found on kaggle.com:

– TV shows on Netflix, Prime Video, Hulu and Disney+ - https://www.kaggle.com/ruchi798/tv-shows-on-netflix-prime-video-hulu-and-disney
– Movies on Netflix, Prime Video, Hulu and Disney+ - https://www.kaggle.com/ruchi798/movies-on-netflix-prime-video-hulu-and-disney

These two databases contain a list of tv-shows and movies, respectively. The databases also show for each movie/series whether they are provided on Netflix, Prime Video, Hulu and/or Disney+. The databases also give some additional information such as the year in which the movie was produced. Both databases are not in RDF. We also added another two other databases we found on kaggle.com to add extra information about movies and TV-shows on Netflix and Disney+. In these databases, additional information about the actors playing in movies/TV-shows can be found. These databases are:

– Disney+ movies and TV shows - https://www.kaggle.com/unanimad/disney-plus-shows
– Netflix movies and TV shows - https://www.kaggle.com/shivamb/netflix-shows

Together, these four datasets allow us to combine information about a movie or tv-show with the information about whether or not a movie is view-able on Netflix/Prime Video/Hulu/Disney+. This allows users the opportunity to filter all series on Netflix by an age restriction, or to see how many films Disney+ has that are directed by Steven Spielberg. This information can be helpful in deciding if a streaming service can satisfy your streaming needs or in comparing streaming services. We will also use the following SPARQL endpoint for a list of movies:

– http://dbpedia.org/ontology/

6 Queries

In this section we will discuss which queries are essential to retrieve data and test whether our database works correctly. This is discussed in the following subsections: Complex SPARQL Queries, where we show what queries we think are essential for testing, and SPARQL Queries and Inferencing where we look at and discuss the different inferenced data.

6.1 Complex SPARQL Queries

A possible query could be to search all movies and series that have genre Horror OR Drama OR Comedy AND have age restriction PG AND have Harry Styles as actor. We do not want to show too many results at once on our web application, since that would mean a very long page, so we filter the results to only show the first 30 results.

```
SPARQL Query:
SELECT ?moviesSeries ?actor ?genre ?ageRestriction ?streamingService WHERE {
{?moviesSeries has_Genre Horror  . }
UNION
{?moviesSeries has_Genre Drama  .}
UNION
{?moviesSeries has_Genre Comedy  .}
?moviesSeries has_ageRestriction PG .
?moviesSeries has_Actor HarryStyles  .
?moviesSeries has_Actor ?actor  .
?moviesSeries has_Genre ?genre  .
?moviesSeries has_ageRestriction ?ageRestriction  .
?moviesSeries streamingService ?streamingService  .
};
Filter 30
```

The user could also want to look up all movies with producer Steven Spielberg that is on streaming service Netflix.

```
SPARQL Query:
Select ?moviesSeries ?actor ?genre ?ageRestriction WHERE {
?moviesSeries rdf:type Movies  .
?moviesSeries has_Producer StevenSpielberg  .
?moviesSeries streamingService Netflix  .
?moviesSeries has_Actor ?actor  .
?moviesSeries has_Genre ?genre  .
?moviesSeries has_ageRestriction ?ageRestriction  .
};
Filter 30
```

If the user wanted to know which streaming service has the most animated movies with no age restriction, the user could use this query.

```
SPARQL Query:

Select ?moviesSeries ?actor ?genre ? ageRestriction ?streamingServices WHERE {
?moviesSeries rdf:type TVseries .
?moviesSeries has_Genre Animation .
?moviesSeries has_ageRestriction ALL .
?moviesSeries has_Actor ?actor  .
?moviesSeries has_Genre ?genre  .
?moviesSeries has_ageRestriction ?ageRestriction
?moviesSeries streamingService ?streamingService
};
Filter 30
```

We also had to run a few complex Sparql queries to make sure our data was in a format in which we could easily use it. For instance, in the IMDb datasets, all actors of a specific movie were given in a single string. So, we ran a query to split this string and insert a triple for every actor.

```
DELETE {
    ?s stream:hasActor ?o .
}
INSERT {
    ?s stream:hasActor ?obefore2 .
    ?s stream:hasActor ?oafter2 .
}

where {
    ?s stream:hasActor ?o .
    ?s rdf:type stream:Title .
    filter contains(str(?o), "%2C")
    bind( strafter(str(?o), "http://www.semanticweb.org/eliseProjectOntology3") as ?o2)
    bind( IRI(strbefore( str(?o2), "%2C" )) as ?obefore )
    BIND (IRI(replace(str(?obefore), "http://example/base/", str(stream:)))  AS ?obefore2)
    bind( IRI(strafter( str(?o2), "%2C")) as ?oafter)
    BIND (IRI(replace(str(?oafter), "http://example/base/", str(stream:)))  AS ?oafter2)
}
```

We also created a Sparql query to change the prefix of a f

6.2 SPARQL Queries and Inferences

We did not manage to download the final dataset and therefore could not see any inferences.