Dialogue Systems 2

Course Project: Movie search app

Lawin Khalid

1. Introduction

This report will present and discuss the implementation procedure of the course project, including pointing out some issues that was encountered during building the application. The purpose of this project was to build a movie search app, capable of finding various information about movies and report this information to the user. The aim was to implement a natural dialogue flow, and to explore the capabilities of the platform that was used for building the application.

The platform used was Talkamatic Dialogue Manager (TDM). To get the information for the queries, the API Open Movie Database was used.

2. OMDb API

There are several suitable APIs available for the task in question, including Open Movie Database[1] (OMDb), The Movie Database[2] (TMDb), and My API Films[3]. OMDb is simpler and does not support as many queries as the other options. However, I found OMDb to be sufficient for this project. Furthermore, since the focus of this project is on dialogue management, rather than the type of queries we can handle, I opted to go with the simpler OMDb API.

The API supports two different kind of queries, here called *information query* and *search query*. Example outputs are shown in figure 1 and figure 2, respectively. The only mandatory parameter is movie title. Optional parameters are year, type (movie, series or episode), plot length (short or full), and output format (JSON or XML).

Information queries give us information about a specific movie release. If no year is specified, the information for the topmost hit is returned. Plot length is defaulted to *short*, output format is defaulted to *JSON*, while the default value for type is empty, which means that all types are included in the returned result. I opted to keep the JSON output format, and leave the type parameter empty. Parameters as year and plot length are used in this project. Information queries are requested by the following format:

www.omdbapi.com/?i=tt3896198&apikey={API_key}&t={info_title}&y={year}&plot={plot_length}

Search queries list all relevant releases for the queried title. The top 10 hits are returned, sorted by popularity. The format for search queries is:

www.omdbapi.com/?i=tt3896198&apikey={API_key}&s={search_title}&y={year}

---

[1] www.omdbapi.com

[2] www.themoviedb.org/documentation/api

[3] www.myapifilms.com/index.do

```
{ "Title": "The Illusionist",
  "Year": "2006",
  "Rated": "PG-13",
  "Released": "01 Sep 2006",
  "Runtime": "110 min",
  "Genre": "Drama, Mystery, Romance",
  "Director": "Neil Burger",
  "Writer": "Neil Burger (screenplay), Steven Millhauser (short story \"Eisenheim the
Illusionist\")",
  "Actors": "Edward Norton, Paul Giamatti, Jessica Biel, Rufus Sewell",
  "Plot": "In turn-of-the-century Vienna, a magician uses his abilities to secure the love of
a woman far above his social standing.",
  "Language": "English",
  "Country": "USA, Czech Republic",
  "Awards": "Nominated for 1 Oscar. Another 10 wins & 10 nominations.",
  "Poster": "https://m.media-
amazon.com/images/M/MV5BY2VkMzZlZDAtNTkzNS00MDIzLWFmOTctMWQwZjQ1OWJiYzQ1XkEyXkFqcGdeQXVyNTIzOT
k5ODM@._V1_SX300.jpg",
  "Ratings": [ { "Source": "Internet Movie Database", "Value": "7.6/10‟ },
    { "Source": "Rotten Tomatoes", "Value": "73%" },
    { "Source": "Metacritic", "Value": "68/100‟ } ],
  "Metascore": "68",
  "imdbRating": "7.6",
  "imdbVotes": "329,866",
  "imdbID": "tt0443543",
  "Type": "movie",
  "DVD": "09 Jan 2007",
  "BoxOffice": "$39,680,302",
  "Production": "Yari Film Group",
  "Website": "http://www.theillusionist.com/",
  "Response": "True‟ }
```

Figure 1. Example output in JSON format of an information query from OMDb API.

```
{"Search":[
{"Title":"The
Illusionist","Year":"2006","imdbID":"tt0443543","Type":"movie","Poster":"https://m.media-
amazon.com/images/M/MV5BY2VkMzZlZDAtNTkzNS00MDIzLWFmOTctMWQwZjQ1OWJiYzQ1XkEyXkFqcGdeQXVyNTIzOT
k5ODM@._V1_SX300.jpg"},
{"Title":"The
Illusionist","Year":"2010","imdbID":"tt0775489","Type":"movie","Poster":"https://m.media-
amazon.com/images/M/MV5BMTg0MjI4MTcxMF5BMl5BanBnXkFtZTcwNTQ1ODUwNA@@._V1_SX300.jpg"},
{"Title":"A Turn of the Century
Illusionist","Year":"1899","imdbID":"tt0000246","Type":"movie","Poster":"https://m.media-
amazon.com/images/M/MV5BMDQ4Mjk5MmYtNDAxYi00N2VlLWEyOTItNzg0YWI5MTMwYmRjXkEyXkFqcGdeQXVyNTI2NT
Y2MDI@._V1_SX300.jpg"},
{"Title":"The
Illusionist","Year":"1983","imdbID":"tt0087456","Type":"movie","Poster":"https://images-
na.ssl-images-
amazon.com/images/M/MV5BMTUxMDM2MjM2M15BMl5BanBnXkFtZTcwMTgyMjUzMQ@@._V1_SX300.jpg"},
{"Title":"The Disappearing
Illusionist","Year":"2015","imdbID":"tt2720764","Type":"movie","Poster":"https://images-
na.ssl-images-
amazon.com/images/M/MV5BMjMyNTI1NjQ1MV5BMl5BanBnXkFtZTgwODYyMzUxNzE@._V1_SX300.jpg"},
{"Title":"The Illusionist","Year":"2009","imdbID":"tt1553917","Type":"movie","Poster":"N/A"},
{"Title":"The War
Illusionist","Year":"2001","imdbID":"tt0278813","Type":"movie","Poster":"N/A"},
{"Title":"The Illusionist","Year":"1901","imdbID":"tt0451420","Type":"movie","Poster":"N/A"},
{"Title":"The Illusionist","Year":"2015","imdbID":"tt4334706","Type":"movie","Poster":"N/A"},
{"Title":"The Making of 'The
Illusionist'","Year":"2007","imdbID":"tt5292902","Type":"movie","Poster":"N/A"}],
"totalResults":"13","Response":"True"}
```

Figure 2. Example output in JSON format of a search query from OMDb API.

3. Implementation

Implementation was first made with explicit grammar entries, with RASA disabled. The reason for this was to reduce the building time when testing the app. However, RASA was enabled and tried towards the end of building the application. The discussion in this report assumes that RASA is disabled, unless otherwise specified.

3.1 Queries

Implemented queries for this project are shown in table 1. All queries take the movie title as parameter, since this is required for all queries with OMDb API. The year parameter is only required if there are more than one release with the same title, else the system will automatically fill the parameter with the only available release. This feature, called incremental search, is further discussed later in the report. Available sources for the rating query are IMDb, Rotten Tomatoes, Metacritic, and an average score of all three. Plot length can be short or full. No actions were suitable for this application.

Query name	**Parameters**	**Returns**
release_year	title	The release year of the movie
director	title, (year)	The name of the director of the movie
genre	title, (year)	The genre(s) of the movie
starring	title, (year)	Names of the 4 main actors of the movie
rating	title, (year), source	The rating of the movie, from the specified source
plot	title, (year), plot length	The short or full plot of the movie
is_starring	title, actor	Yes or no, whether the specified actor is starring in the movie

Table 1. Implemented queries in the movie search application.

Entity recognizer was not used in the implementation. Therefore, every individual had to be explicitly defined in the ontology and the grammar. Individuals were implemented for years between 1950 and 2018, and a few movie titles and actors, enough to demonstrate the capabilities of the application, but not at all enough for a complete useful application. Also, sources and plot lengths were defined.

3.2 Basic dialogue features

The TDM platform offer many features for dialogue management. The term *basic* is used here, not to denote that these features are basic in nature, but that the implementation of them is straightforward and partially built in the system, which leads to easy and user-friendly development.

Supported basic dialogue features for this application are the following:

- Incremental statements
- One-shot statements
- Over-answering
- Other-answering
- Answer revision
- Topic recognition
- Topic switch
- Context recall

Implementation of the basic dialogue features followed the previous labs of the course, and need no in-depth explanation. I started with incremental statements by adding grammar entries without slots for the user questions, as well as adding grammar entries for system questions. When this worked, I continued to add entries with slots for user questions to handle one-shot statements. The slots for each query correspond to the parameters in table 1. These were also the parameters in the queries of service interface as well as the predicates to findout in the plans of the domain.

Over-answering was handled by defining user answer in the grammar. Other-answering, answer revision, topic recognition, topic switch and context recall are all build into the TDM system and needed no further implementation.

3.3 Advanced dialogue features

Two more advanced dialogue management features were implemented in the application, presented in the following subsections.

- Incremental search
- Continuous queries about the same movie

3.3.1 Incremental search

One question that arose during the implementation was how to deal with the years of the movies. Since there are sometimes more than one release with the same title, we want the system to ask the user about which release he or she wants information about. Consider the following two dialogues:

```
U> Who directed the illusionist?
S> What year?
U> 2006
S> The director is James Cameron.

U> Who directed Pulp Fiction?
S> What year?
U> 1994
S> The director is Quentin Tarantino.
```

The first dialogue is reasonable, since there is more than one movie with the title *The Illusionist*. But if we create a slot to be filled for the year parameter, the system will ask for the year also when there is only one movie with that title, as in the second dialogue. This is redundant and not user friendly.

Another option is to default the year parameter as an empty string, and use the bind function of TDM. The API will then return the top hit if the user does not specify the year, while the user can explicitly specify the year if he or she wants information about a specific release. This option also has some problems, though. The user must in advance know if there are more than one movie with that title, and should also know the year for the requested release. If not, the user must first make a release_year query to find out what releases there are. If no year is specified, and the top hit is not the release that the user intended, the answer by the system would be misleading.

The best solution for this issue was to implement incremental search. For the implementation, the predicates title_to_search and year_to_search were made sub features of the new parent predicate movie_to_search. Moreover, in the plans of the domain, the predicate to findout was now set to movie_to_search, and the following parameter was added:

```
<parameters predicate="movie_to_search" question_type="wh_question" source="device"
device="MoviesDevice" incremental="true">
  <ask_feature predicate="title_to_search"/>
  <ask_feature predicate="year_to_search"/>
</parameters>
```

Also, a new function was added in the device, which takes title and year (defaulted to an empty string) as arguments, makes an API search query, and returns to the other functions the list of available releases. The dialogue flow of the system is now as follows:

- Ask for the title.
- Make an API search query for the title.
- If there is only one matching release, fill the slot with that release.
- If there are more than one release, ask for the year.
- Make an API movie information query with title and year, and report the information.

The application is now able to handle the following dialogues correctly. It only asks for the release year in the first example, here exemplified together with a suitable topic switch and context recall.

```
U> I want to know the director of a movie
S> What title?
U> The Illusionist?
S> What year?
U> What years are there?
S> I found 6 movies with that title, released 2006, 2010, 1983, 2009, 1901 and 2015.
S> Returning to the director. What year?
U> 2010
S> The director is Sylvain Chomet.

U> I want to know the director of a movie
S> What title?
U> Pulp Fiction?
S> The director is Quentin Tarantino.
```

There are two system queries that do not take year as an optional parameter, namely release_year and is_starring, as specified in table 1. Hence, incremental search was not implemented for these queries, but instead other solutions were applied. Regarding release_year, that takes title as the only parameter, the system will return all available release years for that title. Is_starring takes title and actor as parameters and will return whichever release the actor is starring in, as demonstrated below. These solutions were simply applied in the device.

```
U> Is Leonardo DiCaprio starring in Titanic?
S> I found 2 movies with that title. Leonardo DiCaprio is starring in the one from 1997.
```

3.3.2 Continuous queries about the same movie

The nature of the application suggests that the user might want to get different types of information about the same movie. That is, we want the system to be able to move between plans with the filled parameters kept intact. Consider the following dialogue flow:

```
U> Who directed Pulp Fiction?
S> The director is Quentin Tarantino.
S> Returning to main menu. What would you like to do?
U> What are the actors of Pulp Fiction?
S> The main actors are Tim Roth, Amanda Plummer, Laura Lovelace, John Travolta.
S> Returning to main menu. What would you like to do?
U> What's the IMDb rating of Pulp Fiction?
S> The IMDb rating is 8.9/10
S> Returning to main menu. What would you like to do?
```

We can assume that most users would find this rather repetitive and not optimal. What we want is that the system doesn't go back to the main menu after each completed query, and instead let the user ask follow-up questions without the need to specify the movie title again. This feature is not supported in TDM, but there is a workaround to handle the issue. First, we choose to not forget the predicates in the top action in the domain. This will keep the filled parameters intact between queries. We then want to silence the prompt for the top action, to avoid the system to repeat "Returning to main menu. What would you like to do?" after reach completed query. This is accomplished by adding the bolded entries below in the domain.

```
<parameters question_type="goal" verbalize="false"/>

<goal type="perform" action="top" reraise_on_resume="false">
  <plan>
    <findout type="goal"/>
  </plan>
</goal>
```

We can now make continuous queries about the same movie, without losing the filled parameters, and without hearing the repetitive prompt after each completed query. However, we should also give the user the possibility to go back to the top and forget all parameters. To accomplish this, an action was implemented, with the only purpose of forgetting all predicates, and prompting the user that we now have returned to the main menu. This action is triggered by user utterances as *thanks*, *ok*, *start over*, etc. Since this issue is not natively supported in TDM, there is no documentation covering this. The only option I found to make the system prompt was to implement a complete action in all files, including the device, to be able to create a report in the grammar. In addition, a greeting was also added to the grammar, to simulate the initial "What would you like to do?". This must be activated by the argument `--should-greet` when running the app. The app can now handle continuous queries about the same movie as exemplified below.

```
U> Who directed Pulp Fiction?
S> The director is Quentin Tarantino.
U> And who is starring?
S> The main actors are Tim Roth, Amanda Plummer, Laura Lovelace, John Travolta.
U> What's the IMDb rating?
```

```
S> The IMDb rating is 8.9/10
U> What about Rotten Tomatoes?
S> The Rotten Tomatoes rating is 94%
U> Thanks
S> Returning to main menu. What would you like to do?
```

4. Discussion

As presented in the previous section, the movie search app is able to simulate natural dialogues quite well. We were able to produce a more human-like dialogue flow, and solved some common issues in dialogue management. This was partly thanks to the effortless built-in capabilities of the TDM platform, and partly the result of implementing some more advanced features. However, there are still some issues with the service.

Topic recognition is not supported for more than five options. More options raise a Generate Error. This seems to be a limitation in TDM. While I can understand that a system question of the type "Do you want to know the director, know the actors . . ." would be unclear with too many options, it would clearly be preferable to at least have a different type of system question, rather than raising an error. For the movie search app, since all seven queries take title as parameter, the service would crash when the user throws out a title name as reply to the initial question "What would you like to do?". However, topic recognition worked fine for other predicates that are not shared with more than five queries, such as year, rating source and plot length. Only the first dialogue below is supported by the system.

```
U> 1997
S> Do you want to know the director, know the genre, know the actors, know the plot or know
the rating?

U> The Godfather
```

- Results in Generate Error

With the workaround for the top action, we can switch freely between different queries keeping already filled slots. However, we cannot change the filled slots and remain in the current plan. Native support for removing the top action, would be beneficial. The last line in the example dialogue below is a question that is silenced in my app, hence put in parenthesis.

```
U> Who directed The Godfather?
S> The director is Francis Ford Coppola.
U> What about Jurassic Park?
S> (What would you like to do?)
```

The implementation of incremental search resulted in some issues with context recall when providing the title in the initial question, as demonstrated below. It might be the case that the parent predicate movie_to_search must be incrementally filled with title_to_search and year_to_search through the device parameter, but when we provide the title in the initial question, we use a grammar entry with a slot for title_to_search instead, which seem to confuse the system. I tried to alter the predicate slots in the grammar, but without success. Without proper documentation for incremental search, this issue was left broken for this project, although I believe there is already a solution for this.

```
U> Who directed Titanic?
S> What year?
U> What years are there?
S> I found 2 movies with that title, released 1997 and 1953.
S> So, What year?
U> 1997
S> (What would you like to do?)
```

RASA generally improves the interpretation, especially regarding utterances with noise, and variations of grammar entries. Without RASA enabled, we must match the grammar entries exactly, which is not user friendly. However, we do get some strange behavior and misinterpretations of topic recognition, which seems to be generally biased towards knowing if someone is starring in a movie. With RASA enabled, topic recognition for years, as well as for "IMDb", fails.

The interaction tests in the appended code work fine with RASA disabled, but a few dialogues fail after enabling RASA. Nevertheless, RASA is enabled in the submitted code to enhance the general user experience and to highlight the problematic behavior of the system with RASA enabled.

5. Future work

Trivial improvements for building a complete movie search application would be to add more queries, such as runtime, awards, language, etc. A more extended API should also be considered. Moreover, adding more movie titles and actors, or preferably, implementing entity recognition would be a necessity. We would also want to solve the issues mentioned in the previous section. Some of these should be solvable with proper documentation for TDM.

Furthermore, support for more advanced dialogues would highly enhance the natural feeling of the dialogue, as well as extending the practical capabilities of the application.

```
U> Who directed Titanic?
S> What release?
U> The one starring Leonardo DiCaprio.
S> The director is James Cameron.
```

This dialogue should be solvable with today's version of TDM. Similar to the query is_starring, we could in the device make a function that go through all releases of the title and look for a match of the actor.

```
U> Who directed Jurassic Park?
S> The director is Steven Spielberg.
U> What's his latest movie?
S> Ready Player One
```

For this dialogue, we must let the service remember the system answer of a question and save it in a slot. After having worked with incremental search, I learned that new individuals of dynamic sorts can be added during runtime, so my guess is that TDM would be able to handle this, either today or in the near future.