Relevant Aspects of Project Development

This chapter of the documentation examines the most relevant aspects of the project development. Like in the previous chapters, we will examine the main components of the developed software and outline which self-made solutions were designed as well as which difficulties were encountered during the course of the project and how they were managed.

Conversational Interface Design  
**Guidelines and Design Principles**

As already explained in the chapter Theoretical Concepts, the conversational interface can be seen as the new discovery in human machine interaction. The aim is to adapt the interface to the user’s way of thinking and therefore allowing the user to interact with the interface in a natural way. In order for this interaction to be successful, the conversational interface has to be designed intuitively so the user does not have to study tutorials or manuals beforehand.

Due to this fact, several design principles were studied to ensure a natural interaction behaviour. Because of the topic’s sudden fame, there are a big variety of sources in the online community, in particular blogs, articles and tutorials that center on how to provide the best possible user experience [1][2]

Some of the concepts applied were:

* Usage of structured input, such as *mutually exclusive button*s: In suitable situations, the messenger’s keyboard is updated in a way so that the user has the option to choose between different predefined answers. This limits the conversation flow opposed to the infinite number of possible answers that can be expected when the user answers freely. However, the right balance between using structured input and letting the answer independently has to be maintained so that the conversation does not seem forced and unnatural. In this project, the concept of mutually exclusive buttons is applied for instance when the user is asked to rate a point of interest that was previously recommended to him, letting him choose between a rating of 1 to 5 stars.
* The user should always be able to interrupt the chatbot and reset the current state of the conversation. If the user does not want to answer a chatbot question, the chatbot should not be stuck in the conversation or insist on the user to answer by no means.
* A conversational interface is text-based. Although being intuitive, this also comes with the risk to overwhelm the user with too much information. Thus, the chatbot’s text messages need to be designed concise and short. An exception is made when the user asks the chatbot to explain all of his features, being followed by a detailed overview of the chatbot.
* The chatbot should always show an immediate reaction to the user input. If the request handling takes longer on the server side of the chatbot, the user is informed that the action might take a while, e.g. by textual representation or revealing that the chatbot is still processing. In the presented software, this happens when the chatbot is asked to give a recommendation, which leads him to modify Telegram to show a “Processing…” response.

**Conversation Flow Design**

Developing a conversational interface, one is faced with an infinite number of user answers. Of course, not all of these variants can be covered by the interface, even though machine learning methods are applied by the natural language parsing platform to deduce the meaning of the user input. During the design of the conversational interface, one important step was to specify the conversation flow between user and chatbot. In order to do so, one had to be aware of the chatbot’s features and capabilities. These features were then specified in detail in the project’s use cases. Based on these use cases, the first approach was to design textual conversation scripts between the chatbot and the user. However, due to the number of different conversations, it was hard to get an overview of the overall conversation flow as the scenarios were not put into a coherent context.

Visualizing the conversation flow using a directed graph turned out be really useful to contextualize the different interactions ***(see Design Specification (Set Reference))***. The different use cases of the conversation flow were structured as paths of the graph. Using API.AI’s natural language parsing concepts intents and contexts, those paths were then implemented in the natural language parsing platform and thus forming the conversation flow how it can be accessed using the Telegram messenger. User input that is not found in the conversation flow diagram is not part of the project’s requirements and therefore covered by fallback answers.

**Messenger Integration**

A very useful feature provided by the natural language parsing platform API.AI is the so-called One-Click Integration. Using this form of integration, a large number of messengers can be hooked to API.AI so the user input is transferred directly to API.AI. After the user input is parsed by API.AI, the parsed output can be either transmitted to an own web service for further processing or directly to the messenger and thus being presented to the user. Because of the simplicity of the setup, this feature seemed very promising and a mockup was implemented as a first step. Unfortunately, it turned out that this approach does not allow to access a lot of needed functionalities provided by Telegram, for example handling the location attachment or accessing Telegram user data, such as user name and id.

Because of this fact, an architecture was designed and set up in which Telegram is connected directly to the web service via webhook and the user messages are forwarded to API.AI’s http API by the web service. An appealing feature of the developed architecture is that the server was designed in a way that the messenger layer and natural language parsing layer can be interchanged easily with different technologies. The architecture design is explained more precisely in the ***Design Specification.***

*OpenStreetMap* Information Processing

In this section, it is examined how the tourist information needed for the chatbot’s recommendation process was retrieved from the OpenStreetMap database. As already stated in (ref Theoretical Concepts), the basic elements *nodes*, *ways* and *tags* of OpenStreetMap’s data model were taken into consideration. The most important role in this context is played by the tags, consisting of key-value pairs that give conclusions about the attached objects, in this case the nodes and ways. Because of this fact, the tags are examined for keys that would indicate if the nodes or ways are of touristic interest.

There is a large number of different tags that can be found in OpenStreetMap[ref]. Due to the fact that OpenStreetMap is an open source community project, the elements are tagged by users. However, there is no standard on tagging elements so that tags are invented and used as needed. As a consequence, there is no reliable way of knowing if an element belongs to a certain tourist category. In order to ensure if a tag is worth to be taken into consideration, the OpenStreetMap taginfo is used to get an overview of how frequent a tag is used.

Based on the discovered tags, a PostGIS query was designed, searching for nodes and ways containing the found key-value attributes (see Design Specification for technical details). Every time a recommendation process is triggered, the query is sent to the database retrieving all nodes and ways in a certain recommendation radius. The resulting elements are converted into our points of interests. In doing so, a mapping between the OpenStreetMap tags into the tourist categories takes place.

In the application’s data model, a point of interest contains an item profile giving conclusion of the membership to a number of tourist categories. These categories are used later in the recommendation process to match the point of interest with the user interests. The tourist categories are defined as followed:

* Sightseeing contains famous sights and typical POIs that are visited by tourists on a trip
* Culture contains cultural institutions, such as museums, theatres, churches
* Food contains gastronomical places such as restaurant and cafes
* Shopping contains shopping malls or marketplaces
* Nature contains city parks, beaches or famous natural sights
* Nightlife, contains nightlife institutions, such as bars or discos

Of course, this is a rather rough categorization of points of interest, and could be subdivided into categories with finer granularity in a future version of the application to obtain more precise results.

The mapping between OpenStreetMap tags and the above mentioned tourist categories can be found in the following table. The table’s cells contain the value of the respective tag key of the columns. So is, for instance, an OpenStreetMap element that is described by the key-value pair *tourism->attraction* categorized as a points of interest that belongs to the category *sightseeing*. Additionally, it is possible to assign multiple categories to a point of interest.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Amenity** | **Tourism** | **Historic** | **Shop** | **Beach** | **Leisure** | **Cuisine** |
| **Sightseeing** | - | Attraction, Viewpoint | All Values | - | - | - | - |
| **Culture** | Place of worship, Theatre | *Museum, Artwork* | - | - | - | - | - |
| **Food** | Café, Restaurant | - | - | - | - | - | *All Values* |
| **Shopping** | Marketplace | - | - | Marketplace, Mall | - | - | - |
| **Nature** | - | Picnic Site | - | - | *All Values* | Park, Nature Reserve | - |
| **Nightlife** | Pub, Nightclub, Casino | - | - | - | - | - | - |

Recommender System Development

In this section, it is examined how the underlying recommender was designed as well as the encountered difficulties that were brought with the development.

**Recommendation Method**

As already mentioned briefly in the Theoretical Concepts, two different recommendation mechanisms were applied and combined in this project: collaborative filtering and a content-based inspired approach.

When the user asks for a recommendation, he is asked to introduce his current location. Based on this location and the user’s identifier, a collaborative filtering algorithm is applied first. Given the ratings the user has made before and the ratings similar users have made, the user is recommended a point of interest. These information can be found in a ratings matrix that is stored in the database, having the following data structure:

|  |  |  |
| --- | --- | --- |
| **User** | **POI** | **Ratings** |
| *UserA* | *Sagrada Familia* | *5* |
| *UserB* | *Sagrada Familia* | *4* |
| *UserB* | *Casa Battló* | *5* |

*This table contains example ratings of two users. Due to the fact that both have rated the Sagrada Familia, UserA may be recommended the point of interest that UserB has visited and also liked.*

The collaborative filtering mechanism is very efficient as it computes suitable recommendations without needing further information about the points of interest. However, in order to compute significant result, there has to be sufficient data available. If the collaborative recommender fails to compute a recommendation, (e.g. when there is no information about the user available yet), a content-based inspired approach is used as a fallback mechanism.

In a traditional content-based filtering approach, points of interest would be recommended to the user based on already made ratings of the user. Opposed to the collaborative filtering approach, the recommendation is based on points of interests similar to the ones the user has already rated. However, these circumstances do not differ much from the collaborative filtering mechanism, meaning that there would still be a problem in situations where the user logs in for the first time and does not have ratings yet. Because of this fact, a self-designed content-based *inspired* approach was implemented. For this approach, user interests are used as well as the point of interest’s item profile. In order for this mechanism to work, the user has to answer questions about his interests when he starts the conversation for the first time. These interests are mapped to the tourist categories shown in the previous section, resulting in the following user profile:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **User** | **Sightseeing** | **Culture** | **Food** | **Shopping** | **Nature** | **Nightlife** |
| *UserA* | *True* | *True* | *False* | *False* | *Unknown* | *False* |

*User Profile - The preferences and dislikes towards the tourist categories are saved. If the opinion on a category is unknown, it is ignored in the recommendation process*

As described before, a point of interest is given an equivalent item profile based on its OpenStreetMap tags. Using the user’s current location, the PostGIS database is searched for POIs situated in the specified recommendation radius. Based on the similarity of tourist profiles, those POIs are returned by the recommender that are most similar to the given user profile. Therefore, we do not have a common recommendation but more of a search of POIs based on the profile similarity.

Based on the fact that there is no need for user ratings in this approach, the fallback mechanism hardly fails and always returns points of interests when there are any in the specified recommendation radius. However, it is not as precise as the collaborative filtering mechanism and has the probability to return POIs the user might not be interested in as the tourist categories are rather general (see Future Work Lines). If there is no information about the user available, he is provided with points of interests purely based on the distance to him.

The technical details of how the recommendation mechanism is implemented using the Mahout framework is explained in more detail in the *Design Specification*.

**Overcoming the Cold Start Problem**

In recommender systems, the cold start describes the well-known problem of a recommender needing a certain amount of user data to perform correctly or give significant results.

One of the biggest difficulties encountered in the course of the project was the lack of sufficient user data needed for the collaborative algorithms. The points of interests that are recommended are based on geographical data from OpenStreetMap. However, OpenStreetMap does not provide information of ratings for its elements. A reason for this is that the open source project aims to give exclusively objective data, not user opinions that is rather subjective and prone for changes. After researching for ratings, it became quickly clear that there were no available ratings data sets that were based on OpenStreetMap data to be found. Only a few data sets were found that contained information about tourist points of interest as well as user ratings, such as the Tourpedia datasets[v] and a TripAdvisor dataset[ta]. The latter seemed promising: the TripAdvisor dataset gathered information about 7034users that rated a total number of 32618 points of interests all over the world.

After analyzing the data set using the Python data analysis library Pandas (see Python script), it became quickly clear that the given data is not sufficient for the recommender to perform properly as the ratings are spread for points of interests all over the world. Since our chatbot is limited to recommend POIs close to the user’s current location, a recommendation area had to be chosen with a close-knit rating coverage. To keep the applied database small, it was decided to limit the chatbot’s recommendation scope to a single city. Barcelona was chosen because of its touristic importance in Spain and comparably frequent appearance in the TripAdvisor dataset (98 ratings). Out of those ratings, only those were extracted that were rated by more than one user to be relevant for recommendation purposes, leaving only 30 ratings.

A second problem was the difficulty to match the TripAdvisor data to the data points from OpenStreetMap. Unfortunately, the name description of points of interest differ between TripAdivsor and OpenStreetMap (the latter being an open source project consisting of data introduced by users, therefore not always being reliable). A mapping of the points of interests mentioned in the TripAdvisor dataset had to be done manually. After removing non-existing points of interest, a matrix had been created having 16 different users and a total number of 28 ratings. These numbers were still not sufficient for a recommendation, so the decision was made to generate more ratings. To do so, the TripAdvisor website was examined to look for more important points of interests in Barcelona, the corresponding OpenStreetMap id was filtered out of the database and some artificial users with different tourist interests were generated, rating all of the POIs based on their membership to the tourist category (see RatingsGenerator)

The evaluation of the Recommender using the partly real, partly generated user ratings can be found in the section. Using the gathered user information, the performance of the recommender can be described as acceptable – however, a more in depth evaluation and generation of user ratings would lead to a more precise recommender.

Bibliography

1: <https://chatbotsmagazine.com/19-best-practices-for-building-chatbots-3c46274501b2>

2: <https://blog.intercom.com/principles-bot-design/>

X: http://wiki.openstreetmap.org/wiki/Map\_Features

10: <https://taginfo.openstreetmap.org/about>

V: <http://tour-pedia.org/about/datasets.html>

Ta:

1. A. Roshchina, J. Cardiff and P. Rosso. (2015). TWIN: Personality-based Intelligent Recommender System, Journal of Intelligent & Fuzzy Systems, IOS Press, vol. 28, no. 5, pp. 2059–2071, DOI: 10.3233/IFS-141484.