UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

TFG del Grado en Ingeniería Informática

Development of a Chatbot for Tourist Recommendations

Presentado por Jasmin Wellnitz
en Universidad de Burgos — July 1, 2017
Tutor: Bruno Baruque Zanón

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería en Informática

D. Bruno Baruque Zanón, profesor del departamento de Ingenería Civil, Lenguajes y Sistemas Informáticos.

Expone:

Que la alumna D. Jasmin Wellnitz, con pasaporte L28PGM5NZ, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado Development of a Chatbot for Tourist Recommendations.

Y que dicho trabajo ha sido realizado por la alumna bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, July 1, 2017

Vº. Bº. del Tutor:

D. nombre tutor

Resumen

Este proyecto consiste en desarollar una aplicación de bot conversacional que es capaz de recomendar puntos de interés turisticos, personalizados a los gustos del usario. Se accede al bot conversacional usando la aplicación de mensajería instantánea Telegram.

El proyecto se focaliza en desarollar el lado servidor del interfaz de Telegram. Para ello se construye un servidor web Java que usa la plataforma de procesamiento de lenguaje natural API.AI para analizar la entrada del usario. Como prueba de su utilización la aplicación está desplegada en la plataforma-como-servicio Heroku.

Las recomendaciones dadas al usario están basadas en las informaciones el usario comparte con el bot conversacional y datos ya existentes de otros usarios parecidos. El algoritmo de recomendación aplicado combina los dos enfoques de la teoría de recomendación más importantes: el filtrado colaborativo y basado en el contenido.

Además, se emplea una base de datos geográfica usando PostGis, proporcionando las informaciones turisticas en las cuales las recomendaciones están basadas. En la demostración de la aplicación, se usa la información geográfica de Barcelona, España, empleando datos obtenidos de la plataforma libre de *OpenStreetMaps*, y por tal motivo, las recomendaciones proporcionadas están limitadas a esa ciudad.

Descriptores

bot conversacional, interfaz conversacional, Telegram, procesamiento de lenguaje natural, sistema de recomendación, base de datos geográfica, punto de interés

Abstract

This project deals with the development of a chatbot application that is able to recommend tourist points of interest customized to the user's preferences. The chatbot can be accessed using the instant messaging app Telegram.

The project's main focus lies on the development of a server-side architecture to the Telegram interface. In order to do so, a Java web application is set up that uses the natural language processing platform API.AI to parse the user input. The application is deployed to the platform-as-a-service Heroku.

The recommendations given to the user are based on the information the user shares with the chatbot and already existing data of similar users. The applied recommendation algorithm combines the two most important approaches of recommendation theory, content-based and collaborative filtering.

Furthermore, a geographic database is set up using PostGis, providing the needed tourist information the recommendations are based on. In the presented example of the application, geographic information of Barcelona, Spain, is used and therefore limiting the provided recommendations to this city.

Keywords

Chatbot, conversational interface, Telegram, natural language processing, recommender system, geographic database, points of interest

Contents

List of Figures

List of Tables

Introduction

A chatbot is a computer program that interacts with its users through a conversational interface. They have been a topic of interest in the field of computer science for decades, yet, due to the rise of smart devices in the last couple of years, chatbots have received a great deal of new attention. The possibilities provided by the conversational interface were rediscovered and combined with the state of the art instant messaging methods.

This project concentrates on this upcoming trend by developing a chatbot that can be accessed through an instant messaging platform, in this case the messenger Telegram. Using the messenger's interface, the user interacts with the chatbot and is provided with tourist recommendations. The developed application consists of three principal components:

1. The chatbot which forwards the user input to the natural language processing platform api.ai and interprets the response.
2. The recommendation system which computes the personalized recommendation based on the user's preferences.
3. The geographic information database which makes the needed information for recommendation available.

In the following paper, the developed application is presented by examining the main objectives, underlying theories and way of proceeding as well as describing encountered challenges in the course of the project.

Project Objectives

This project aims to develop a chatbot that is able to present customized tourist recommendations to its users via an instant messaging application. In the presented example of the application, the messenger Telegram is used to interact with the user. The following steps and objectives must be realized to develop the chatbot:

- A Java web server is set up which is connected to Telegram through its webhook API to enable user interaction. The application is deployed to a platform-as-a-service cloud server to enable the webhook integration.
- Design of a conversational interface: the conversation flow between user and chatbot should be designed in a natural and flexible way. After doing so, it is mapped to the natural language processing platform API.A which then parses the user input into formalized data. API.AI is accessed by the application through a REST-like HTTP endpoint. The parsed input is interpreted by the chatbot and triggers the desired behaviour, such as recommendation or storage of important user information.
- A recommender system must be implemented to provide personalized tourist recommendations. The recommender is based on data the user has shared with the chatbot and additional data of similar users. To overcome the problem of initially sparse user data, different recommendation methods are combined as well as retrieving existing user data from other sources and/or generating data.
- Retrieval of tourist data from the geographic information database OpenStreetMaps: the data is filtered so that only data of touristic importance is evaluated by the recommender and presented to the user. It is prepared in a way that similarity measures can be made between user interests and the point of interest inherent properties.

Theoretical Concepts

This section explains the most relevant theoretical concepts that were applied during the course of this project to provide a fundamental understanding of the application's main components, or more precisely the chatbot, the recommender as well as the geographic database.

3.1 Chatbots

A chatbot is a computer program that interacts with its user via a chat interface. About 60 years ago, the first steps in the development of chatbots were taken by the computer scientists Alan Turing and Joseph Weizenbaum, proposing the concept of computers communicating like humans do. One of the first natural language processing programs was ELIZA, developed by Joseph Weizenbaum in 1966. Although some users were tricked into thinking that ELIZA was an actual human conversation partner [1], this basic example was stretched to its limit quickly because of its simple rule-based structure. However, the fascination of computers being a conversation partner remained one of the big objectives of modern artificial intelligence.

The topic's big revival occurred with the introduction of mobile devices in the early 2000s. All of the sudden, developers were faced with the task of transforming their well-known desktop applications and websites into apps to make them suitable for a mobile market. Still, in the last couple of years it turned out that users actually do not like to use a variety of apps, but rather concentrate on only a few, mainly messaging apps. That is how in 2016, the idea of the conversational interface resurged when many global big-players like Google, Facebook, Microsoft, IBM or Amazon decided to take part in the development of conversational interfaces [2].

After explaining briefly the importance of chatbots over the years, the following part is going to define the different terms Natural Language Processing, Conversational Interface and Chatbots to outline the differences between

them. After that, the key concepts needed to model a conversation flow are introduced.

Natural Language Processing, Conversational Interface and Chatbots

Natural Language Processing is a component in the field of Artificial Intelligence in which natural language is analyzed and processed in a way that computers can use converted information easily in further algorithms [3]. A sub-discipline of Natural Language Processing is the so-called Natural Language Understanding (NLU).

Nowadays, many big players provide free to use natural language understanding platforms that make the development of conversational interfaces possible. One of these NLP-NLU platforms is API.AI which translates human language into a formal representation using machine learning techniques as well the later explained NLU concepts entities, intents and contexts [4].

Using a **conversational interface**, people are enabled to interact with virtual assistants, smart devices and social robot via their natural language [5]. Conversational interfaces are considered as the third wave of user experience, after the terminal and the graphical user interface. The aim is that by having a conversational interface, users do not have to adapt to the computer, but the computer has to adapt to the human way of communicating.

There are two basic types of conversational interfaces: voice assistants such as Apple's *Siri* or Amazon's *Alexa* that communicate using spoken language and **chatbots**, enabling communication via typing [6]. Basically, in chatbots pattern matching is used to interpret the user's inputs and templates are used to provide the system's output.

Concepts

The conversational interface used in this project is developed using the NLP-NLU platform API.AI that relies on certain key concepts that are explained in the following.

Using API.AI, an **agent** is modeled to parse the user's natural input into structured data. In an agent, the conversation flow with the user is specified using the key components entities, intents and context. After designing the agent, it can be integrated into many different platforms and thus providing an application's conversational interface [7]. The agent will rest located on the service provider servers and is integrated in our application by accessing it via HTTP endpoints.

The agent relies on **machine learning** algorithms to understand the user input and extract relevant data. Before being confronted with the actual

user, input examples are to be specified. Based on these examples, the agent's machine learning model decides which path of the conversation flow is chosen. As usual in machine learning, the agent learns to adapt better to the user as it is provided constantly with real-life conversations [8].

The main components used in modeling the conversation flow are entities, intents and context. **Entities** are domain objects an application takes actions on. They can be considered as parameters of an action to be taken. In API.AI, there are several already defined system entities, e.g. specifying parameters of time, units and geography. Additionally, the developer can define his own entities [9].

In an API.AI agent, user requests are mapped to **intents**. By matching the user input with previously specified examples, intents are extracted and used to trigger an action. In our tourist chatbot example, a typical intent would be "Give Recommendation" if the user asks for a tourist recommendation nearby [10].

When defining an intent, the developer has the option to set an output context. **Contexts** manage the conversation flow by distinguishing the state the conversation is in. Based on the state, the agent may take different decisions on the same user input [11].

3.2 Recommender Systems

With the advent of business in the Internet, so-called Recommender Systems were introduced and gained importance since the nineties [12]. A Recommender System aims to predict and quantify how a user reacts to a certain item. A variety of recommendation algorithms exists, all of them based on collected user preference data (e.g. item ratings). The most famous approaches are *collaborative* and *content-based filtering* that were also used in this project to recommend *Points of Interests* to the users based on their travel interests. In the following, the applied methods are introduced.

Content-based Filtering

Content-based recommendation systems calculate recommendations based on the properties and characteristics of an item. In general terms, an item is recommended to a user if he is interested in its properties. To define the similarity of items, an item profile is designed representing its important characteristics. In our tourist example, the item profile contains tourist categories based on OpenStreetMap tags. Then, user profiles are created containing the same categories as the item profile, indicating which characteristics a user prefers in an item. Usually, this user profile is filled by examining the user ratings and extracting the characteristic for the already rated items. The preference for a

user liking a certain item is then calculated by comparing its profile with the item profiles. To do so, different measures can be applied, one of the most famous being the cosine distance [13].

Collaborative Filtering

While the previously presented approach concentrates on the similarity between items for recommendation, collaborative filtering is based on the similarity between users. An item is recommended to a user when similar users have showed an interest in it before. Instead of profiles representing preferences, the only needed data for this approach is the matrix of user ratings. There are different measures to determine if two users are similar weighting user ratings differently, such as the *cosine distance* or *the jaccard distance*. The collaborative filtering mechanism is very successful as it often provides recommendations outside the expected scope of user interests. However, in order to work correctly, a large amount of user ratings is needed.

Hybrid Approaches

Due to the fact that collaborative filtering suffers from the so-called *cold start problem* (that is not performing well when there is only sparse user data), it is often combined with other approaches. One of the most common solutions is falling back to content-based algorithms when user ratings are not significant and hence building up the user rating database. In this project, a hybrid mechanism is applied, although some modifications of the traditional content-based approach were made. The detailed way of proceeding and made adaptions are explained in detail in chapter 4.5.

3.3 Geographic Information Processing

Point of Interest

In the context of geographic information, the expression Point of Interest [14] is often used to describe a map feature that has a certain significance. It is a broad term that reaches from functional services like post offices or car parks to tourist attractions. One of this project's main elements is to extract essential tourist information from the vast amount of geographic data, meaning finding relevant points of interests and outputting them in a comprehensible way for the user.

Geographic Database Systems

The geographic information used in this project is stored in a geographic or spatial database. While the most common kind of databases nowadays is relational, they are unable to handle geographic data because of its complexity

[15]. This is why for this task spatial database systems are used which provide geospatial data types in its data model and query language [16]. The objects in the database contain a spatial or geometric attribute which describes their location, shape, orientation and size.

In this project, geographic information from *OpenStreetMap* is used and stored in a spatial database using *PostGis*.

OpenStreetMap Data Model

To extract our needed points of interest from the raw geographic data, the underlying data structure of the OpenStreetMap information is examined, consisting of the following four principal elements [17]:

Nodes Points with a geographic position that are used to represent map features without a size, such as points of interest or mountain peaks.

Ways Ordered lists of nodes that are used both for representing linear features such as streets and rivers, and areas, like forests, parks, parking areas and lakes.

Relations Ordered lists of nodes, ways and relations, so-called members. Relations are used for representing the relationship of existing nodes and ways, e.g. long-distance motorways of areas with holes.

Tags Key-Value pairs storing metadata of the geographic objects they are attached to (namely node, way or relation). Tags can include type, name and a broad variety of map features.

Principally, the data structures *Nodes*, *Ways* and *Tags* are investigated in detail to construct the wanted *Points of Interests*. The exact proceeding of extracting the needed information from this data model is explained in depth in chapter 4.5.

Tools and Technologies

In this section, the tools and technologies used during the course of the projects are presented. The different components of the geographic database are described as well as the tools to design the conversational interface. Additionally, some alternatives to the eventually used tools are discussed in order to explain why the specific tools are chosen. In contrast, the tools used in the development process such as version control repository, integrated development environment or build-management tools are not explained in detail as they could be interchanged and are not essential for the application's characteristics.

4.1 Chatbot Platform

As already discussed in the previous chapter, there are several competing platforms that permit developers to build their own chatbot. Most of the major software big-players nowadays, such as Google, Facebook, Microsoft, are taking part in the development of these platforms and provide their own solutions, for example API.AI (Google), wit.ai (Facebook), luis.ai (Microsoft) or IBM Watson. Out of these contestants, API.AI and wit.ai seem to be the most widely known and easiest to use. Therefore, both options were investigated to see which one would be the most suitable for this project.

API.AI

API.AI [18] is a natural language processing platform that facilitates creating conversational user interfaces. It was launched in 2014 and acquired by Google in September 2016 [19]. In order to model conversations, entities and intents are used as key concepts. API.AI provides a rich management toolset as well as a simply one-click-integration mechanism to import the chatbot into a variety of mobile apps. Also, the modeled conversation flow in API.AI can be accessed by submitting queries through the REST-like HTTP endpoints.

API.AI offers a variety of prebuilt agents that can be used as a base for the own conversational interface. On top of it, the conversational interface can be enabled to support *small talk*, allowing it to reply to basic, unspecific user input without any further development.

Wit.ai

Like API.AI, Wit.ai is a natural language processing platform. It was acquired by Facebook in January 2015 and is free to use ever since then [20]. Its key concepts for language processing include entities, intents and, additionally, stories. Stories help to create basic user scenarios in which typical user and chatbot conversations are designed. Regarding integration, Wit.ai provides a web service API to integrate the designed conversational interface in messaging apps.

Choice of Platform: API.AI

After testing both platforms, both left a good impression and seem to be suitable to use. On the one hand, Wit.ai's story feature seems promising to design conversations in an easy way, but is still in beta status. However, API.AI convinces with its rich management toolset, an extensive documentation and the integration of simple conversational features such as *small talk*.

4.2 Geographic Database

In order to extract tourist points of interests from *OpenStreetMap*, a geographic database was set up as the application's backend. The following tools form either a part of the mentioned database or were used to set it up.

OpenStreetMap

OpenStreetMap [21] (or short OSM) is a collaborative project with the aim to collect and update free to use geographic data. Its main purpose is to be a central data source which can be e.g. used for rendering maps. The stored data contains infrastructural information such as roads or buildings as well as variety of additional informational tags. In this project, the OpenStreetMap data is used to extract necessary tourist information in order to create user recommendations. There is a big number of data dumps available that store the above mentioned data for either the whole planet or smaller regions or cities. These dumps can be downloaded in the file formats XML and PBF and imported into a PostgreSQL database.

PostgreSQL 9.5.5

PostgresSQL [22] is an object-relational database system. It is an open source software that is most extensively conform to the SQL standard ANSI-SQL 2008. In this project, PostgreSQL is used to store and manage the geographic database.

pgAdmin III 1.22.0

pgAdmin III [23] is an open source tool to facilitate the management of PostgreSQL databases by providing a graphic user interface. Among many features, it comes with an SQL editor tool to create and run queries and displays data entries.

Osmosis 0.44.1

Osmosis [24] is an open source command-line based application that is able to process data from OpenStreetMap. In this project, it was principally used to import data from OSM files into the PostgreSQL database.

At first, the similar tool *osm2psql* was investigated and used in this project. However, during research it turned out that osm2psql is not the right fit because its main objective is the import of .osm files for rendering purposes. Due to this reason, only data that are render relevant are imported into the PostgresSQL database. On the other hand, Osmosis imports all of the raw .osm data, namely Nodes, Ways, Relations and their corresponding tags [25]

4.3 Messenger - Telegram

Telegram [26] is an instant messaging app for smartphones, tablets or computers. There are available versions for iOS, Android, Windows Phone, as well as desktop applications for Windows, OSX and Linux. Telegram concentrates on speed and security of its messages.

In June 2015, Telegram introduced its Bot API, allowing third-party developers to integrate their own bots into the messenger. The bots are controlled sending HTTPS request. In this project, incoming updates from Telegram are received via an outgoing webhook [27].

4.4 Platform as a Service - Heroku

Heroku [28] is a platform-as-a-service that enables users to run their applications in the cloud. It supports several programming languages, among them Node, Ruby and Java. There are several pricing models offering a range of

different features. In this case, the free plan is used which comes with the inconvenience that the application sleeps after 30 minutes of inactivity, leading to a short delay every time the chatbot is accessed after not using it.

Likewise, Heroku Postgres is a database-as-a-service enabling the upload of the project's database into the cloud.

4.5 APIs and Libraries

Apache Mahout

Apache Mahout is an open-source project which provides a set of machine-learning algorithms for classification, clustering and recommendation [29]. Mahout plays a vital part in the project's development as Mahout's Java library forms the base for the application's recommender. The recommender's collaborative filtering approach is based on Mahout's simple user-based recommender [30] whereas some of Mahout's algorithms were adapted to create a content-based mechanism that fit this project's needs (more precisely in Relevant Aspects and Annex C – Design Specification).

Spark

Spark is a micro framework that is used to create Java web applications. [31]. Its most outstanding feature is the lightweight design which enables the setup of a web application with minimal overhead relying on Java 8 language features. A Spark application's key element is a set of routes which defines how the application is accessed by clients. A route consists of a HTTP verb, a path and a callback. In this project, a POST route is set up using Spark which receives the incoming messages from Telegram.

Foursquare

Foursquare [32] is a mobile service application which provides personalized recommendations of places for its users. In this project, Foursquare's RESTful API is used to access additional information OpenStreetMap does not provide, more precisely extracting photos for points of interests.

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.

5.1 Conversational Interface Design

Guidelines and Design Principles

As already explained in the 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 is a large number of sources in the online community, in particular blogs, articles and tutorials that center on how to provide the best possible user experience [33] [34].

Some of the concepts applied were:

- Usage of structured input, such as *mutually exclusive buttons*: 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 the possible user answers 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 large 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 Annex C - Design Specification). The different use cases of the conversation flow were structured as paths of the graph. Using API.AI's natural language parsing con-

cepts intents and contexts, those paths were then implemented in the natural language parsing platform and thus forming the conversation flow. 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.

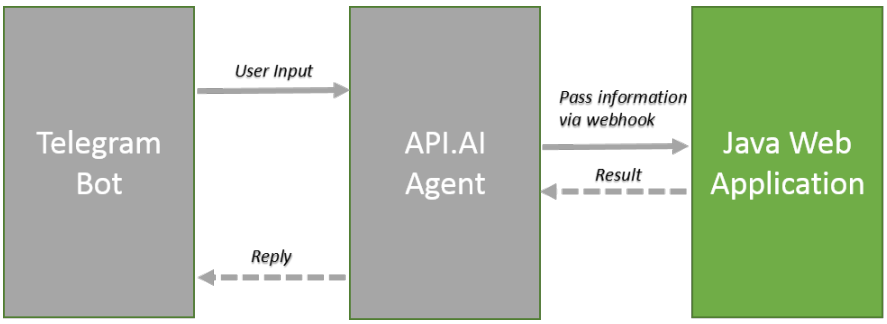

Figure 5.1: API.AI's one-click-integration

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, see **??** 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.

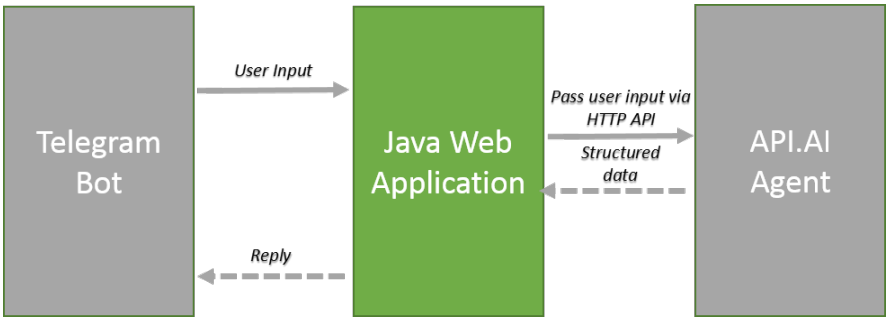

Figure 5.2: Final message flow design

5.2 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 previously explained, 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 [35]. 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 [36] 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. 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 self-defined 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 famous sights and typical POIs that are visited by tourists on a trip

Culture cultural institutions, such as museums, theatres, churches

Food gastronomical institutions such as restaurants and cafés

Shopping shopping malls and marketplaces

Nature city parks, beaches or famous natural sights

Nightlife 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 table 5.1. 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.

5.3 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 data structure as seen in the table 5.2. 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

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

Table 5.1: Mapping between OSM tags and tourist categories

User	POI	Rating
UserA	Sagrada Familia	5
UserB	Sagrada Familia	5
UserB	Casa Battló	4

Table 5.2: User Rating Matrix

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 ratings the user has previously given. 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 user profile as seen in table 5.3. 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.

User	Sightseeing	Culture	Food	Shopping	Nature	Nightlife
UserA	True	True	False	False	Unknown	False

Table 5.3: User Profile

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 and geographical proximity.

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 Annex C - 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 [37] and a TripAdvisor dataset [38]. The latter seemed promising: the TripAdvisor dataset gathered information of 7034 users that rated a total number of 32618 points of interest 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 (for technical details, see class RatingsGenerator.java).

After generating the data and implementing the recommender, an evaluation was made to see how well the recommender performs using partly real, partly generated user ratings (see Annex C - Tests). 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 more precise results.

Related Work

The developed application combines two interesting fields that are meeting the pulse of time, recommendation systems and conversational interfaces. Concerning recommendations, the following projects can be seen as related as they also center on the recommendation of tourist data:

Foursquare a location-based recommender service which adapts to the user's profile and returns venues given the user's proximity. As already discussed, some elements of the Foursquare API are integrated into the presented chatbot.

TripAdvisor a user-based tourist website. TripAdvisor is mainly a project in which user share their experiences and rate points of interest. Since 2014, TripAdvisor provides hotel recommendations based on the user's search history and feedback [39]. In this project, a TripAdvisor rating data set was used as a base for retrieving user ratings.

However, those examples present their results in a different way than the chatbot does. Of course, due to the recent hype of chatbots, there is a large variety of different chatbots for all possible kinds of platforms. In the following, some chatbots are presented that are similar to the one developed in this project:

Hipmunk: a Skype chatbot that gives travel advices and recommendations. Its focus is the recommendation of flights and hotels, which is a topic which is not covered in this project.

Mica, the Hipster Catbot: This might be the most similar project to the one developed. The chatbot can be accessed through a variety of messengers, such as Telegram, Skype or Facebook. It provides tourist recommendations based on the user's current location. However, this application

seems to be not as complex as the one developed in this project as the recommendations are not adjusted to the user's profile.

Conclusion

In this project, a chatbot for tourist recommendations was developed that can be accessed using the instant messaging app Telegram. The presented thesis contains the documentation of the application's course of development, points out the most noteworthy characteristics and serves to provide background knowledge of the applied theoretical concepts and technologies. A recurring theme throughout the documentation is the development of the application's three main components: the conversational interface, the recommender and the geographic database. These components were integrated in a webserver with a multilayered architecture that is deployed to the Platform-as-a-Service *Heroku*.

On a personal note, the student acquired new skills and knowledge, not only by exploring previously unknown fields of computer science such as recommender system theory. Moreover, the student has become familiar with managing an agile software development process and configuring an application's infrastructure, including the setup and deployment of a webserver and the integration of external services accessing REST-like HTTP APIs. All in all, developing a fully operative application from scratch can be considered as a personal milestone.

7.1 Future Work Lines

There are several future work lines that would lead to an overall improvement of the chatbot functionality, mostly helping to improve the usability and accessibility:

Conversational Interface

- Integrating the chatbot into other messengers, for instance *Facebook* or *Slack*. Due to the recent chatbot hype, there is a large number of

messengers a chatbot can be integrated into. The presented chatbot's architecture is designed in a way in which the main structure does not need to be modified for a change of messenger technology. Thus, an integration of different messengers can be done easily which leads to the chatbot targeting new user groups.

- Refinement of the conversational interface: In order to recommend points of interests to the user, the conversational interface has to filter the user's interests from his messages. In the presented version of the chatbot, this is done in a simple way, only checking for the appearance of certain key words used by the user that are similar to the defined tourist categories. The interest retrieval could be improved by proactively asking the user questions about his interests based on small talk between the user and the chatbot.

Recommendations

- The applied content-based recommendation is based on the similarity between the point of interest's item profile and the user's interests. To refine the recommendation, the categorization of interests should be refined, applying a finer tourism ontology. A hierarchy of tourism categories could be conceivable, so that the category food, for example, is subdivided into different cuisines, defining whether a user is interested in vegetarian food for instance. Refining the tourist categories will lead to a better adaptation to the user's interests.

- Retrieval of User Data: The more user data is available, the more significant are the collaborative filtering results. Therefore the research for user rating data sets could be readopted or more user ratings have to be generated.

Geographic Database

- Extending the chatbot's recommendation scope: For performance reasons and the lack of user ratings, the presented version is limited to give recommendation for Barcelona. To target a bigger audience, giving recommendations for a larger area would be necessary. A next step would be to extend the recommendation radius to all of Spain. Providing recommendations for the whole world would require a big amount of database storage space and probably a refactoring of the chatbot's recommendation and data access mechanism due to the massive increase of geographic data.

- In the presented version of the chatbot, the data is stored in an offline database, using a dump from 2017. To keep the recommended points of interest data up-to-date, the database should be updated regularly.

Bibliography

[1] Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9:36–45, 1966.

[2] Nicolas Bayerque. A short history of chatbots and artificial intelligence, 2016. [Internet; accessed 30/06/17].

[3] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. Technical report, NEC Labs America, 2008.

[4] API.AI. Introduction - understand the key concepts, 2017. [Internet; accessed 10/06/17].

[5] Michael McTear, Zoraida Callejas, and David Griol. *The Conversational Interface: Talking to Smart Devices*. Springer, 2016.

[6] John Brownlee. Conversational interfaces, explained, 2016. [Internet; accessed 10/06/17].

[7] API.AI. Agents, 2017. [Internet; accessed 10/06/17].

[8] API.AI. Machine learning, 2017. [Internet; accessed 10/06/17].

[9] API.AI. Entities, 2017. [Internet; accessed 10/06/17].

[10] API.AI. Intents, 2017. [Internet; accessed 10/06/17].

[11] API.AI. Contexts, 2017. [Internet; accessed 10/06/17].

[12] Charu C. Aggarwal. *Recommender Systems: The Textbook*. Springer, 2016.

[13] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2011.

[14] OpenStreetMap. Point of interest, 2017. [Internet; accessed 10/06/17].

[15] Philippe Rigaux, Michel Scholl, and Agnes Voisard. *Spatial databases - with applications to GIS*. Morgan Kaufmann Publishers, 2002.

[16] Ralf Hartmut Güting. An introduction to spatial database systems. *The VLDB Journal*, 3:357–399, 1994.

[17] OpenStreetMap. Elements, 2017. [Internet; accessed 10/06/17].

[18] API.AI. Welcome, 2017. [Internet; accessed 10/06/17].

[19] API.AI. Api.ai is joining google!, 2016. [Internet; accessed 30/06/17].

[20] Wit.ai. Wit.ai is joining facebook, 2015. [Internet; accessed 10/06/17].

[21] OpenStreetMap, 2017. [Internet; accessed 10/06/17].

[22] PostgreSQL. About, 2017. [Internet; accessed 10/06/17].

[23] pgAdmin. pgadmin, 2017. [Internet; accessed 10/06/17].

[24] OpenStreetMap. Osmosis, 2016. [Internet; accessed 10/06/17].

[25] OpenStreetMap. Osmosis, osm2postgresql & osm2pgsql – openstreetmap-daten, datenbanken und spielplätze, 2012. [Internet; accessed 10/06/17].

[26] Telegram. Telegram faq, 2017. [Internet; accessed 10/06/17].

[27] Telegram. Getting updates, 2017. [Internet; accessed 10/06/17].

[28] Heroku. The heroku platform, 2017. [Internet; accessed 12/06/17].

[29] Apache Mahout. What is apache mahout?, 2017. [Internet; accessed 30/06/17].

[30] Apache Mahout. Creating a user-based recommender in 5 minutes, 2017. [Internet; accessed 30/06/17].

[31] Spark. Spark - a micro framework for creating web applications in kotlin and java 8 with minimal effort, 2017. [Internet; accessed 30/06/17].

[32] Foursquare. Detailed docs - api endpoints, 2017. [Internet; accessed 12/06/17].

[33] Chatbots Magazine. 19 best ux practices for building chatbots, 2017. [Internet; accessed 20/06/17].

[34] Intercom. Principles of bot design, 2017. [Internet; accessed 20/06/17].

[35] OpenStreetMap. Map features, 2016. [Internet; accessed 20/06/17].

[36] OpenStreetMap taginfo. About taginfo, 2017. [Internet; accessed 20/06/17].

[37] TourPedia, 2017. [Internet; accessed 20/06/17].

[38] A. Roshchina, J. Cardiff, and P. Rosso. Twin: Personality-based intelligent recommender system. *Journal of Intelligent and Fuzzy Systems*, 28:2059–2071, 2015.

[39] Paul Sawers. Tripadvisor introduces 'just for you,' hotel recommendations based on search history and feedback, 2014. [Internet; accessed 30/06/17].