BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

CHATBOT CAPABLE OF INFORMATION SEARCH
CHATBOT PRO VYHLEDÁVÁNÍ INFORMACÍ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **MICHAL ĎURISTA**
AUTOR PRÁCE

SUPERVISOR **doc. Dr. Ing. JAN ČERNOCKÝ,**
VEDOUCÍ PRÁCE

BRNO 2019

Brno University of Technology
Faculty of Information Technology

Department of Computer Graphics and Multimedia (DCGM) Academic year 2018/2019

Bachelor's Thesis Specification

21921

Student:	**Ďurista Michal**
Programme:	Information Technology
Title:	**Chatbot Capable of Information Search**
Category:	Speech and Natural Language Processing

Assignment:

1. Get acquainted with the current state of chatbots and respective frameworks.
2. Get acquainted with the creation of chatbots in Microsoft botframework and with a suitable cloud service for their creation, for example MS Azure.
3. Suggest an algorithm capable of information search on a given, domain-restricted web page.
4. Implement a chatbot in the MS botframework, based on the developed algorithm and available services.
5. Assess its functionality by user testing, evaluate the tests, suggest and eventually implement improvements.
6. Create a 30s video presenting your work.

Recommended literature:

- Joe Mayo: Programming the Microsoft Bot Framework: A Multiplatform Approach to Building Chatbots, Microsoft Press, 2017.

Detailed formal requirements can be found at http://www.fit.vutbr.cz/info/szz/

Supervisor:	**Černocký Jan, doc. Dr. Ing.**
Head of Department:	Černocký Jan, doc. Dr. Ing.
Beginning of work:	November 1, 2018
Submission deadline:	May 15, 2019
Approval date:	November 1, 2018

Abstract

”Chatbot” is a very popular term in today’s artificial intelligence era. Chatbots can be seen in business solutions more a more nowadays. The main goal of this thesis is to create an algorithm that is capable of information retrieval and implement it into a chatbot. The information resides on a real customer’s web pages. The thesis also provides an overview of current chatbot situation along with the Microsoft technologies used for the development. The technological background of these technologies, mostly natural language processing techniques, is covered too. The thesis also describes the implementation of the algorithm and the chatbot itself as well as the real industrial environment testing process.

Abstrakt

Pojem ”chatbot” je v dnešnej dobe umelej inteligencie veľmi populárny výraz. Chatbotov vidno stále viac a viac v biznis riešeniach dnešných firiem. Hlavným cieľom práce je vytvoriť algoritmus, ktorý je schopný vyťahovať informácie a implementovať ho do chatbota. Tieto informácie možno nájsť na webových stránkach reálneho zákazníka. Práca rovnako poskytuje prehľad súčasnej situácie chatbotov ako aj Microsoft technológií pre ich vývoj. Technické detaily na ktorých tieto technológie pracujú, predovšetkým spracovanie prirodzeného jazyka, sú taktiež zahrnuté. Práca popisuje implementáciu algoritmu ako aj chatbota samotného spolu s procesom testovania v skutočnom priemyselnom prostredí.

Keywords

chatbot, natural language processing, Microsoft Bot Framework, BotBuilder, Azure Search, Text Analytics, Lucene search

Kľúčové slova

chatbot, spracovávanie prirodzeného jazyka, Microsoft Bot Framework, BotBuilder, Azure Search, Text Analytics, Lucene search

Reference

ĎURISTA, Michal. *Chatbot Capable of Information Search.* Brno, 2019. Bachelor’s thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Dr. Ing. Jan Černocký,

Rozšírený abstrakt

Chatboti sú stále častejšie skloňovaným pojmom v dnešnej dobe umelej inteligencie a strojového učenia. Prvý chatboti sa ale začali objavovať už v minulom storočí. Alan Turing sa začal zaoberať myšlienkou, či dokáže stroj napodobniť rozmýšľanie človeka. Vznikol vtedy takzvaný Turingov test, ktorého podstata je, že ak v komunikácii medzi strojom a človekom bude tento človek presvedčený o tom, že nekomunikuje so strojom, ale s ďalším človekom, možno stroj prehlásiť za inteligentný. Prvé pokusy pokoriť tento test sa objavili už v šesťdesiatych rokoch minulého storočia, keď vznikol systém nazvaný ELIZA, ktorý možno považovať za prvého chatbota. Tento chatbot predstieral, že je terapeut. Pýtal sa užívateľa rôzne otázky na ktoré potom odpovedal. Veľa užívateľov došlo k záveru, že chatbot ELIZA skutočne pochopil ich problémy, aj keď išlo o pomerne jednoduchý systém ktorý spracovával prirodzený jazyk na základe predefinovaných pravidiel.

V dnešnej dobe sa chatboti čoraz viac stávajú súčasťou biznis riešení firiem, predovšetkým vďaka ich využiteľnosti. Chatbotov možno vidieť predovšetkým ako čiastočnú náhradu pracovníka zákazníckej podpory. Fakt, že chatbot je počítačový program, nie človek, prináša isté výhody. Počítačový program dokáže pracovať nepretržite bez prestávok. Rovnako tak chatbot dokáže obslúžiť v podstate akýkoľvek počet zákazníkov naraz. Nevýhody sú ale očividné. Nakoľko je to stále len program, nie človek, častokrát nedokáže vyhovieť všetkým požiadavkám zákazníka. Akonáhle zákazník vybočí zo scenára, na ktorý je chatbot pripravený, zákazník nedostane odpovede na otázky, ktoré položil. Preto sa chatboti využívajú predovšetkým na často opakované požiadavky zákazníkov, aby sa reálny zamestnanec zákazníckej podpory mohol sústrediť na riešenie netriviálnych problémov.

Táto práca sa zaoberá implementáciou takéhoto typu chatbota pre firmu ComAp. Konkrétne, práca popisuje jednu časť tohto chatbota, ktorý sa volá InteliBot, zodpovednú za vyhľadávanie informácií. Tieto informácie sa týkajú predovšetkým vlastností produktov, ktoré firma ComAp predáva a ku ktorým poskytuje následný servis. Súčasťou informácií, ktoré vie InteliBot hľadať sú aj riešenia rôznych problémov, ktoré firma ponúka pre svojich zákazníkov. Práca popisuje algoritmus, ktorý je schopný tieto informácie nájsť na základe otázky užívateľa ako aj konkrétnu implementáciu tohto algoritmu do chatbota. Práca nerozoberá len implementáciu ale taktiež popisuje do hĺbky technológie, predovšetkým od firmy Microsoft, ktoré boli využité pre vytvorenie takéhoto chatbota. Predovšetkým sa jedna o Microsoft Bot Framework. Ide o veľmi prepracovaný nástroj na výrobu sofistikovaných chatbotov, ktorý poskytuje vývojárom možnosti na vytvorenie v podstate akéhokoľvek chatbota. Tento nástroj uľahčuje vývoj poskytnutím predefinovaných tried a metód, ktoré možno využiť pri vytváraní konverzačnej logiky ako aj pri vytváraní chatbota samotného. Práca takisto poskytuje pohľad na služby Microsoft Azure. Ide o cloudovú platformu od firmy Microsoft, ktorej služby boli pri vytváraní InteliBota, predovšetkým spomínaného algoritmu, využité.

Popri technológiách sú vysvetlené aj techniky spracovania prirodzeného jazyka, na základe ktorých tieto technológie pracujú. Ide predovšetkým o pochopenie prirodzeného jazyka pomocou vektorov slov. Záver práce sa venuje testovaniu InteliBota a zhodnoteniu výsledkov. S priebehom vývoju chatbota bola firma ComAp vždy oboznámená a teda vznikal pomerne veľký priestor pre implementáciu požiadaviek, ktoré vyplynuli z ich spätnej väzby. Týmto sa zabezpečila spokojnosť zákazníka ako aj pomerne vysoká úspešnosť InteliBota pri odpovedaní na užívateľské otázky.

Chatbot Capable of Information Search

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of Dr. Jan Černocký. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

. . . .
Michal Ďurista
May 12, 2019

Acknowledgements

I would like to express my thanks to my supervisor, Dr. Jan Černocký for his professional guidance and willingness. I would also like to thank to all members of Xolution development team, particularly Libor Bešenyi, the technical director of the company, for his continuous help and opinions during the work. I must not forget to thank my family and friends for the support during the study.

Contents

Chapter 1

Introduction

There are many definitions of a chatbot. The simplest one may be that a chatbot is a computer program that is trying to imitate a conversation with a human. It may just scan the input from a user and look for specific keywords or words patterns and provide an answer. It can also work in more sophisticated way by using various services, natural language processing systems, machine learning or artificial intelligence [3]. The input text can come from a written text or even as a spoken word, which is then translated into a text using various speech-to-text techniques. These voice-chatbots are also called interactive voice response (IVR) systems[1].

Another way to look at a chatbot might be that it is a conversational interface for software services just like a mobile or web application. A good example of this way of looking at a chatbot is a process of booking a flight. The user can do so by visiting a website, using mobile app, calling a human agent or by chatting with a chatbot that is able to book a flight for him or her, just like the human agent would do. All these ways are offering the same service - booking a flight [27].

In the context of this thesis, we can look at a chatbot as a replacement of a human employee that provides various information or a service requested by the user. There are many advantages for a company when it decides to use a chatbot. First of all, as already mentioned, a chatbot is a computer program, which means, it can run all the time without any exhaustion unlike a human employee. Another difference between a human and a bot is that a single chatbot is able to assist many users at a time. These aspects are one of the main reasons, why companies decide to involve chatbots in their business solutions more and more nowadays. However, there is no perfect chatbot that is able to answer all users questions or problems yet. That is why we cannot think of a chatbot as a complete replacement of a customer support employee. Better way of thinking is that it is a computer assistant that is able to resolve common user requirements, which tend to repeat most of the time, so that the human assistant can focus on more complicated quests.

1.1 Aims of thesis

The main goal of the thesis is to create a fully functional chatbot, also called InteliBot, for a company called ComAp[2]. More specifically, the thesis is dedicated to a specific InteliBot section called *I'm interested in...*. This section is responsible for searching various

[1] https://en.wikipedia.org/wiki/Interactive_voice_response
[2] https://www.comap-control.com/

information from the company website so that users do not have to find it manually. The thesis describes an algorithm used for the information retrieval along with the integration of the algorithm within the bot and the implementation of the chatbot section itself. InteliBot is designed and developed by a company called Xolution[3].

The thesis also aims to explain basic techniques lying behind text processing which is crucial in chatbots. Moreover, it provides an overview of the available tools for chatbot development as well as more in depth explanation of Microsoft technologies used to create chatbots, particularly InteliBot.

1.2 Structure of thesis

In chapter 2, a quick history of chatbots is provided together with some examples of already implemented and used chatbots. Chapter 3 explains basic natural language processing techniques that lie behind the technologies used for chatbot development, particularly Microsoft ones, that are described in chapter 4. Chapter 5 explains, how these technologies are used in the implementation of InteliBot, specifically the *I'm interested in...* section of it. In chapter 6, the testing process of the chatbot is described and chapter 7 summarizes the work.

[3] https://www.xolution.sk/

Chapter 2

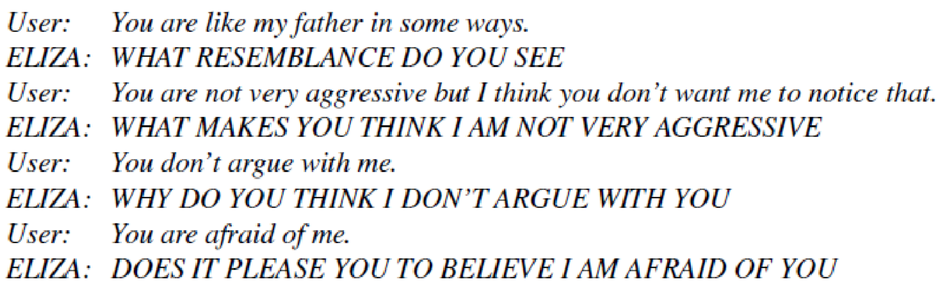
Overview of chatbots

Although the word chatbot is very popular and well-known nowadays, it was not always the case. Here is a brief history.

2.1 Chatbot evolution

Although it may look like a chatbot is pretty recent invention, the very first program that can be considered as a chatbot was a system called ELIZA by Joseph Weizenbaum in 1966. However, the history of chatbots can be traced back to 1950's when Alan Turing, the British computer scientist, started to wonder, whether machines can think. In 1950, he published an article called Computing Machinery and Intelligence followed by Turing test [31]. The whole idea behind is, that if a machine can imitate a human in such a way that it convinces the other person to believe that they are communicating with another human, not a machine, then the machine can be considered as intelligent [25].

ELIZA was created to pretend to be a therapist who asks questions and even provides answers. In its basis, it is a simple natural language processing system that can carry on a conversation. It uses pattern matching to recognize phrases like "You are X" and translates them into suitable outputs like "What makes you think I am X?". The success of ELIZA was pretty remarkable. Many people believed that it really understood their problems and even continued to do so after the operation of the program was explained to them [15, 33].



User: *You are like my father in some ways.*
ELIZA: *WHAT RESEMBLANCE DO YOU SEE*
User: *You are not very aggressive but I think you don't want me to notice that.*
ELIZA: *WHAT MAKES YOU THINK I AM NOT VERY AGGRESSIVE*
User: *You don't argue with me.*
ELIZA: *WHY DO YOU THINK I DON'T ARGUE WITH YOU*
User: *You are afraid of me.*
ELIZA: *DOES IT PLEASE YOU TO BELIEVE I AM AFRAID OF YOU*

Figure 2.1: ELIZA example [32].

Though it is considered to be the first chatbot ever, it was not until 1994 when the term 'ChatterBot' was coined by Michael Mauldin [16] to describe these conversational systems

[25]. Fast-forward to the year 2017, when Mobile World Congress[1] took place in Barcelona and chatbots were the main headliners. They were supposed to be the next big thing. However, even today they have not reached the level that web and mobile applications have in terms of popularity, usability or adaptation within the society. Anyway, they are getting popular and better every day [17].

2.2 Known chatbots

As mentioned in a previous section, the amount of chatbots is increasing almost on a daily basis, especially nowadays when artificial intelligence and machine learning are one of the major fields in today's technical world. Here are some honourable mentions.

A.L.I.C.E

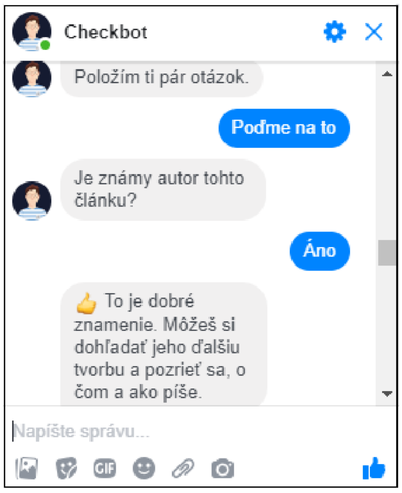
It is one of the most famous and one of the first chatbots to go online. The name itself stands for Artificial Linguistic Internet Computer Entity but is commonly referred to as Alicebot or Alice. It is a natural language processing chatbot developed in 1995 by Richard Wallace [13], later rewritten to Java in 1998. In 2001, Wallace published an AIML (artificial intelligence markup language) specification. It was designed to react to human input as naturally as possible. Alice was originally inspired by previously mentioned ELIZA and won many awards including three Leobner Prizes[2] (2000, 2001, 2004). It is the oldest Turing Test contest which started in 1991, however, the bot is still unable to pass the test completely. The chatbot even served as an inspiration for a movie called Her by Spike Jonze in which a human falls in love with a bot[3] [9].

Checkbot

Checkbot[4] is a very unique chatbot from Slovakia that came to light in 2019. Sharing disinformation from conspiracy websites or sources without any author is very actual and sad phenomenon that often leads to rise of extremism all around the world. This bot is able to determine whether a particular article is potentially dangerous or misleading or not. It runs on Facebook Messenger and the whole process of recognizing such potential dangers is based on asking a user certain questions and giving relevant and useful answers.

The very first question a user gets is to provide a link to the article he wants to check. This link is then ran through a list of known conspiracy websites[5] and the Checkbot tells if there was a match or not. However, it is impossible to check all unwanted webs. If the article was not found in the list, the chatbot at least provides a list of suspicious keywords that occurred in it. It also continues with asking further questions like whether there is an author, whether it spreads hatred against certain group of people and so on. The Checkbot then comes to a conclusion if the article is a potential threat and gives you a list of problematic facts that resulted from the questioning [24].

[1]https://www.mobileworldlive.com/mobile-world-congress-2017/
[2]http://aisb.org.uk/events/loebner-prize
[3]https://www.imdb.com/title/tt1798709/
[4]https://www.checkbot.sk/
[5]https://www.konspiratori.sk/zoznam-stranok.php


Checkbot

Položím ti pár otázok.

Poďme na to

Je známy autor tohto článku?

Áno

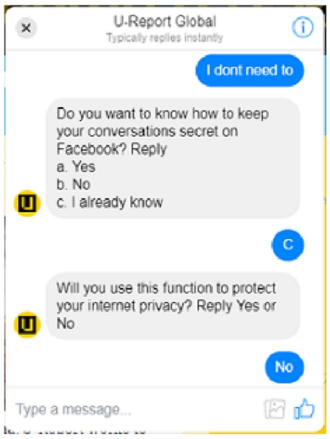
👍 To je dobré znamenie. Môžeš si dohľadať jeho ďalšiu tvorbu a pozrieť sa, o čom a ako píše.

Figure 2.2: Sample communication with Checkbot.

U-Report

Unlike previously mentioned chatbots, this chatbot[6] is not very talkative. When the user starts to chat with this bot, it only gathers basic information about the user like age, country and gender. At the end it offers to access more information about the bot if the user has other questions. This bot is used by international child advocacy nonprofit UNICEF. After gathering all the information about the user, U-Report sends out polls about urgent social issues on a regular basis. Users can then respond with their inputs and UNICEF subsequently uses this information for potential policy recommendations [28].

There is an example when this quite simple chatbot made a significant impact. Users in Liberia were asked whether teachers were coercing students into sex in exchange for better grades. The results were that about 86% of the asked children reported that this was an issue. This resulted in a collaboration between Liberia's Minister of Education and UNICEF to end this practices [29, 28].


U-Report Global

Typically replies instantly

I dont need to

Do you want to know how to keep your conversations secret on Facebook? Reply
a. Yes
b. No
c. I already know

C

Will you use this function to protect your internet privacy? Reply Yes or No

No

Figure 2.3: Sample communication with U-Report bot.

[6]https://ureport.in/

Chapter 3

Technological background

There are certain technologies and algorithms lying beneath the services that the chatbot in this thesis is using. Although complete understanding of these technologies is not necessary for the developer to build such chatbot, as these services mostly act as a black box, it is very beneficial to have at least some idea about them. This chapter is dedicated to the basics of such techniques.

3.1 Natural language processing

No computer program is able to fully understand a piece of text and convert it to a programmer friendly data structure that explains the real meaning of the text [8]. And computers are very good when it comes to working with such data structures. Database tables or some important medical records are a good example. When it comes to processing such data, computers are much faster than humans. However, processing unstructured data is much more difficult task for a computer. There are no standardised rules that can be applied on such data, unlike an ordinary computer program, which is essentially just a set of rules that it should operate by. Humans have developed the ability to understand natural language over the time that a human race exists. We can understand the real-world meaning of a text written in a book. We can feel the emotions. We can read between lines. A computer is unable to do such things [26].

Natural language processing, or NLP, is a subfield of artificial intelligence. Its main goal is to process and analyze natural language data. In other words, it is focused on the interaction between humans that use natural language and computers. History of NLP can be traced back to 1950s when already mentioned Touring Test was introduced.

Most of the NLP systems, or techniques, are based on machine learning due to recent significant advances in the machine learning field. This approach can be also called as statistical NLP. On the other hand, there is so called rule-based NLP. As the name suggests, such systems are designed as a set of rules which can be in a form of grammar or heuristic rules for stemming for example. There are many applications of NLP in real life. These include language translation applications, word processors and of course, chatbots.

There are two major areas where NLP is used:

3.1.1 Syntactic analysis

According to Oxford dictionary, the definition of a syntax is that it is the arrangement of words and phrases to create well-formed sentences in a language or that it is a set of rules for or an analysis of the syntax of a language[1]. In the context of NLP, the main goal of syntactic analysis is to evaluate the nature of how the language aligns with the grammatical rules. Algorithms basically derive meaning from words by using these rules [10]. Here are the most commonly used syntax techniques:

- lemmatization to return the base dictionary form of a word - lemma (word *better* is turned into *good*),
- morphological segmentation to divide words into units - morphemes (*unbreakable* into *un-break-able*),
- stemming for retrieving the root form of the word (*studying* is transferred to *study* by ommiting the *-ing* suffix),
- part-of-speech tagging to get the part of speech for words (word *play* can be a noun as well as a verb),
- word segmentation to divide a large text into distinct units (texts are divided into words, sentences, topics etc.),
- sentence breaking to recognize sentence boundaries,
- parsing to determine the parse tree of the sentence based on a context-free grammar[2].

These are just examples of the techniques used in syntactic analysis. Although they are very important part of natural language processing, their deep understanding is not necessary for a developer to build a chatbot when using one of the popular chatbot frameworks. The second part, which is semantic analysis is much more interesting and important for a chatbot developer.

3.1.2 Semantic analysis

The word semantics is defined as the branch of linguistics and logic concerned with meaning. Particularly in the lexical context it is concerned with the analysis of word meanings and relations between them[3]. This part is the difficult one in natural language processing. Algorithms are trying to understand the meaning and interpretation of words and sentences [10]. Here is a list of some techniques used in semantic analysis:

- machine translation to translate a text from one language to another,
- named entity recognition (NER) for classifying named entities from a text (from sentence "London is a nice city" the word *London* may be classified as a *Location* entity),
- Natural language understanding (NLU), see 3.2,
- Word sense disambiguation to determine, which particular words sense was meant for a sentence it occurred in.

Some of these techniques are essential in chatbot development and will be explained more in depth in further sections.

[1]`https://en.oxforddictionaries.com/definition/syntax`
[2]`https://en.wikipedia.org/wiki/Parse_tree`
[3]`https://en.oxforddictionaries.com/definition/semantics`

3.2 Natural language understanding

Although many people think that the term natural language processing is more or less the same as natural language understanding, there is a difference between these two. Natural language understanding, or NLU, is a subset of a wider field of NLP. As already mentioned, natural language processing is a common word for all the systems that let humans talk to computers in their natural language. It is able to break down the input text, analyze it, create a response in natural language and so on [18].

Meanwhile, NLU is a narrow, but very complex part that is responsible for processing unstructured input into structured data that computers can understand and act accordingly. Humans are very good when it comes to mispronunciations, colloquialisms, swapped words and other issues typical for human nature. However, it is a real challenge for a machine to handle these unpredictable inputs. In other words, NLU can be understood as the machine's ability to get the meaning of what people say in their natural languages [18].

In order to understand the natural input, computers have to transfer it to a structured ontology by using various techniques. They have to extract, identify and resolve entities. Also it is necessary to get the semantic meaning within the context in order to identify intents [18]. A sentence such as: „I want a flight to London on March 8“ would have a structure like:

want:flight {intent} / London {location} / March 8 {date}

And exactly entities and intents are essential in chatbot development in terms of NLU. To understand where these come from, vector semantics, particularly word vectors have to be explained.

3.3 Vector semantics

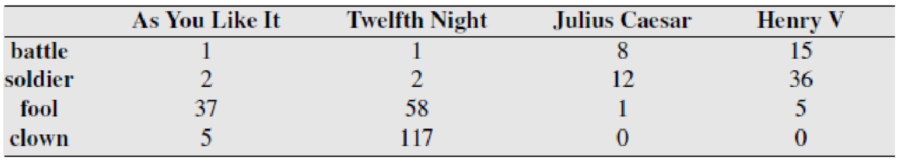
Vector models are a part of natural language processing for a long time. Whether it is named entity extraction, relation extraction, parsing or semantic role labeling, these models are widely used when it comes to such applications. Furthermore, they are the number one choice when computing semantic similarity, in other words the similarity between two words, documents and so on [15].

These models are usually represented as a co-occurrence matrix quantifying how often words co-occur. We can think of vectors in two contexts.

3.3.1 Vectors and documents

The first one is a document context represented by a term-document matrix where rows represent a word in the vocabulary and columns a document. In figure 3.1 there is an example of a selection of such matrix. It shows the occurrence of sample words in various plays by Shakespeare.

So, for example, the vector for *Twelfth Night* is [1,2,58,117] and for *Henry V* it is [15,36,5,0]. A collection of vectors, vector space, is defined by their dimension. For instance, the second dimension of these vectors represents how many times the word *soldier* occurred in the text. The documents from figure 3.2 can be represented as points in 4-dimensional space identified by their vectors. Since 4 dimensions are not ideal to understand and draw in 2-dimensional text, here is a visualization in two dimensions, for the word *battle* and the word *fool*:



	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	1	8	15
soldier	2	2	12	36
fool	37	58	1	5
clown	5	117	0	0

Figure 3.1: Example of term-document matrix for various Shakespeare plays [15].

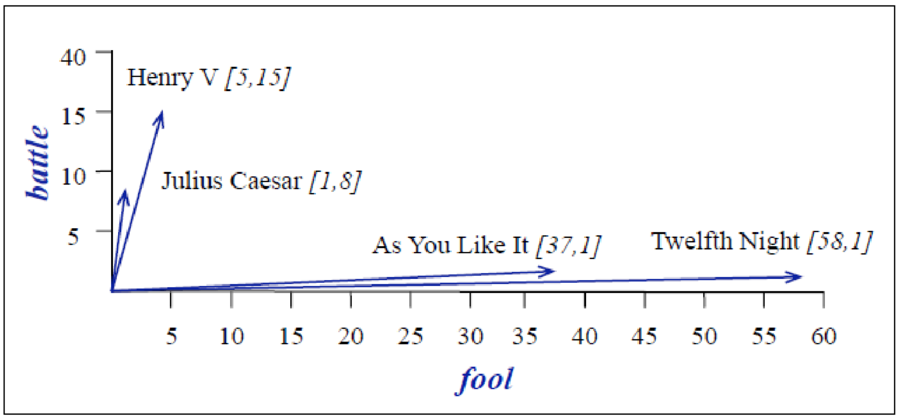


Figure 3.2: Document vectors visualization for the word *battle* and the word *fool* [15].

As seen in figure 3.2, the vectors where there are more fools and clowns look much more alike than the other two vectors. It is no coincidence, since the first two plays are comedies whereas the other two are not.

Although it is not important for a chatbot developer to understand how vectors work in documents, it gives a good foundation to comprehend the word vectors.

3.3.2 Word vectors

When dealing with simple tasks such as spam classification, traditional NLP approaches can do the job. These techniques however do not capture the syntactic and semantic relationships between words. For example, words *dog* and *cat* both refer to almost the same thing. They both represent an animal that is often mentioned in texts about pets but techniques such as one-hot encoding[4] are unable to recognise their similarity.

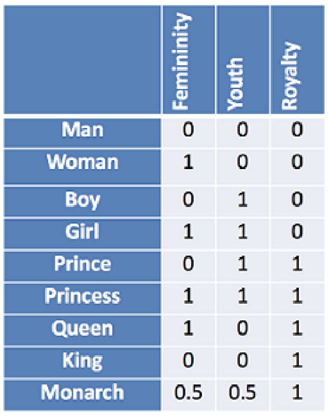
Vectors can represent words just like they represent documents as shown in the previous section. Word vectors are multidimensional vectors of real numbers where each point represents a dimension of the meaning of the word. In other words, semantically similar words will have similar vectors. The principle is similar to document vectors. For instance, words like *spoon* and *fork* will have more similar word vectors to the word *knife* compared to the word *faculty*.

Representing words as vectors has another advantage. Since a vector is in its basis a row of numbers, mathematical operations can be made upon it. Two vectors can be subtracted or added:

$$king - man + woman = queen \tag{3.1}$$

[4]https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f

This example demonstrates how subtracting maleness and adding femaleness to the word *king* results in the new vector representing word *queen*. As already said, each dimension captures a meaning and the real number, or in other words, the weight, represents how close is the association to it. Figure 3.3 should provide a better imagination.



	Femininity	Youth	Royalty
Man	0	0	0
Woman	1	0	0
Boy	0	1	0
Girl	1	1	0
Prince	0	1	1
Princess	1	1	1
Queen	1	0	1
King	0	0	1
Monarch	0.5	0.5	1

Figure 3.3: Example of weights for selected vocabulary of 9 words [2].

„You shall know a word by the company it keeps!“ is a famous quote by J. R. Firth in NLP from 1957. Simply put, words that occur in similar contexts usually represent similar meanings. The context can be understood as its surrounding, so called window. For example, two-window context means two preceding and two following words of the input. Anticipation or predicting the probability of a context of a word creates its word vector. In other words, by making such predictions on the chance that words are contextually proximate to a particular word, the weights that create the vector are learned. Word2vec model[5] is the right choice for this task. It is a single hidden layer neural network[6] for reconstructing the context. The output of such word2vec models is however not the vector itself since the output represent the probabilities that a random word from the corpus is contextually close to the input word. The vector can be explained as it is just a side effect, since it is the numerical representation of the word. The numbers are the weights from the hidden layer learned by the neurons. Therefore 200 neuron hidden layer will result in 200 dimensional word vector [5].

There is a lot more to explain regarding word vectors. Whether it comes to mathematical details, neural networks, neurons, weights to loss functions and so on. Nevertheless, explanation above should give a chatbot developer just the right idea of how machines extract the meanings of words and on what basis the NLU used in chatbot development is built.

3.4 Lucene search

Lucene search is also one of the technologies lying behind services used to create the chatbot described in this thesis. It is a Java library capable of full-text search. Its designed to easily add text search into applications or websites. The heart of Lucene search is its full-text index where the content can be added. Developer can then perform queries on it and get

[5]https://en.wikipedia.org/wiki/Word2vec
[6]https://en.wikipedia.org/wiki/Artificial_neural_network

the result in the way he or she specifies. It can be ranked by relevance to the query or other specifications such as date. The content may come from basically any source, whether it is a database, website or a file [1]. Many searches on the Internet are based on Lucene, including major companies like Twitter[7] or LinkedIn[8].

Lucene searches already mentioned index instead of the text directly. It is the reason why it is able to achieve relatively fast responses. For a better imagination, it can be explained as getting the pages with the specific keyword by looking at the index in a book instead of just going through every page of it. This approach is called an inverted index [1].

Indexes consist of documents. If an index is built from a database table of users, then a single user is one document of the index. Documents consist of fields. It is a simple name-value pair. Indexing means adding documents to so called IndexWriter, whereas searching means retrieving them via so called IndexSearcher. Searching can be carried out only on already built indexes. It is done by making a query and passing it to an IndexSearcher. So called hits are then returned. In this case, query has its own syntax[9] which allows to specify fields, their weights, boolean logic and other functionality [1].

3.5 Chatbot frameworks

Chatbot framework is a place where a developer can build a chatbot from a scratch. It typically provides a set of APIs, services, predefined classes and so on. This means that a chatbot developer must have advanced programming skills to use such tools. There are also so called chatbot platforms, which, unlike frameworks, provide easy-to-use tools for user without any programming knowledge. They usually offer drag-and-drop functionality that allows anyone to create a chatbot. These platforms include Aivo, Botsify, Chatfuel and many others. However, to create more sophisticated chatbots, one will need a framework to do so. Here are some of the most popular chatbot frameworks.

Amazon Lex

This chatbot framework from Amazon provides a service for creating conversational interfaces for any program using either text, or voice. It offers automatic speech recognition to convert speech to text and natural language understanding for identifying intents of the input. The famous voice assistant Amazon Alexa is powered by the same technologies, so developers have the tools they need to build sophisticated bots, just like Alexa [6].

For building a conversational chatbot using Amazon Lex, a developer does not need to have deep learning expertise. He or she just specifies the basic conversation flow in its console. It dynamically adjusts the responses. Chatbot can be built, tested and published via the console as well. It can be then added to web applications, mobile devices or chat platforms like Facebook Messenger. It offers integration with AWS Lambda, which offers to run the code without any managing servers or provisioning. Moreover, the integration with other AWS services like Amazon Cognito, Amazon CloudWatch and many other is also possible [7].

Amazon Lex is probably one of the best choices in terms of voice capabilities since it may be integrated with Alexa, particularly with Alexa Skills Kit (ASK). It is because developers

[7]https://twitter.com/
[8]https://www.linkedin.com/
[9]http://www.lucenetutorial.com/lucene-query-syntax.html

are provided with a choice to export their chatbot schema into ASK, which enables them to create a bot that can communicate using voice without any extra scaffolding. Another advantage this framework provides is that there is a big predefined list of subjects and entities prepared for the bots. All these facts contribute to an opinion, that Amazon Lex is a good pick when it comes to simplicity of the commands or for less experienced developers [4].

DialogFlow

Formerly known as Api.ai or Speaktoit, DialogFlow is a next framework that enables developers to create voice and text-based conversational interfaces such as chatbots. It is now owned by Google, however, there is some history behind. Speaktoit was founded in Russia back in 2010. In 2012 it got funding from Intel Capital [14], in 2014 it released Api.ai to developers all around the world to create Siri-like voice solutions [34]. It also stands behind Assistant, a famous personal assistant application, which was even in top 10 Android Apps of 2011 [30].

DialogFlow is able to analyze various input types, whether it is text or voice input. Responding to a user is similar, it can respond either with synthetic speech or through text. Just like Amazon Lex, it also provides natural language understanding system to understand the input from the user. It comes with prebuilt entities, but developers can create their custom entity types. Another feature is that it provides an integrated code editor, which enables developers to build serverless applications linked with their conversational interface. They can also provide custom WebHook which is hosted on premises or in the cloud. Furthermore, more than 20 languages and 14 different platforms are supported [11]. It also comes with an analytics tools so that the developer has an option to see how well is their agent doing and improve it eventually [12].

Microsoft Bot Framework

This framework is the place to go when searching for robust capabilities and serious customization. Complicated things tend to be more powerful. That is why this framework was chosen for creating the chatbot in this thesis and will be described more in depth in the next chapter.

Chapter 4

Microsoft technologies for chatbot development

Microsoft offers a set of cloud-based services, also known as Microsoft Azure or simply Azure. Many of these services are applicable to chatbot development, especially Language Understanding Intelligence Service (LUIS), Azure Bot Service or Azure Search. However, the first thing that needs to be covered is Microsoft Bot Framework.

4.1 Microsoft Bot Framework

There are three main components of Bot Framework: channels, the Bot Connector and the chatbot itself.

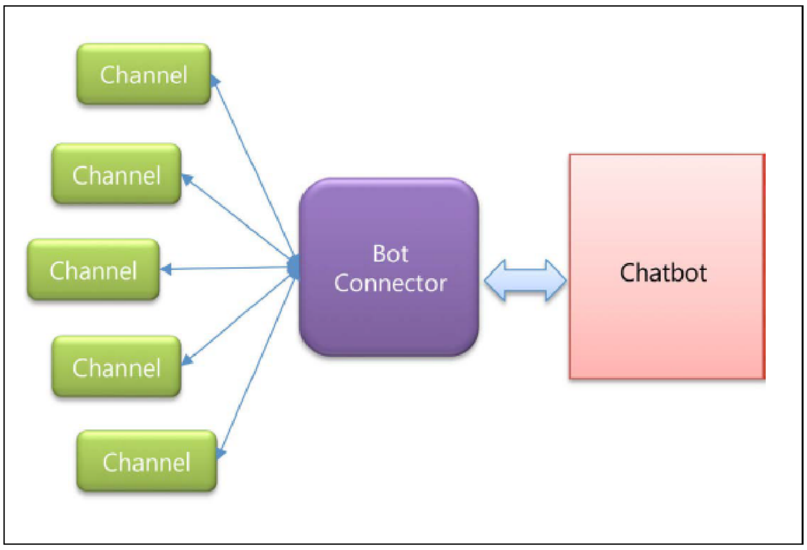


Figure 4.1: Communication flow between the components [19].

Channels are applications that communicate with the chatbot. These include standard apps like Skype, Facebook Messenger, Slack but even emails, SMS and many others. It is important to note that developers can build their custom channels like a web chat on their website for instance. Then there is Bot Connector. It is a cloud component that these means of communication send messages to as well as receive them from. The chatbot is the thing that developer cares about the most, since he or she needs to build the conversation logic in it [19].

Communication between the Bot Connector and other components is called routing. Despite that, there are other functionalities that connector offers as seen in figure 4.2. In addition to routing, it is able to store the state of the bot. State means custom information about the conversation, users or even the combination of these two, a user with conversation.

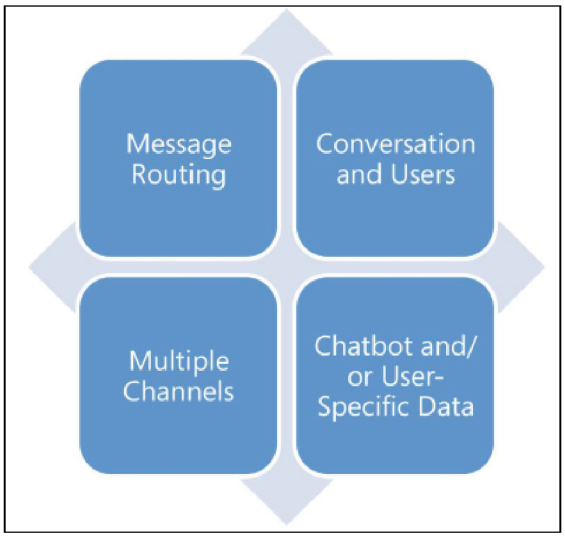


Figure 4.2: Services the connector offers [19].

The connector communicates with chatbot component via *activities*. An activity is a message that is able to carry the text, attachments like images, but it is also used for managing the conversation. Sometimes there is a need to send some event to or from the chatbot. A good example of this might be disabling the input text field so that the user cannot type any text. When certain conditions are met, the chatbot sends an activity to the connector with such event, which is then sent to the channel and it prevents the user from writing any text in this scenario. The Bot Connector is platform-agnostic because its interface is a REST API. That means that while the Bot Framework SDK (v3) is developed in C# programming language and supports only C# and node.js programming languages, the Bot Connector supports all programming languages since it offers a Connector REST API [19].

The chatbot component is the place where all the conversation logic happens. The chatbot can have any purpose the developer decides it to have. From the very start of the conversation, to the end of it, the chatbot is responsible for the flow. Of course, the conversation can go in the direction the bot is not designed for. Sometimes users simply try what the chatbot is capable of. That means that even when a user asks a question that the bot has no answer to, because it is from completely different topic, it is a good practice, that chatbots respond that they do not understand at least.

Chatbots, just like any other applications, can call several external services in order to manage the conversation flow. As will be shown in the further sections, there are Microsoft services used in the chatbot described in this thesis. Whether it is these services, the Bot Connector, channels or the chatbot component, it is obvious that the whole system is built on a distributed architecture unlike many different applications which reside on the same device [19].

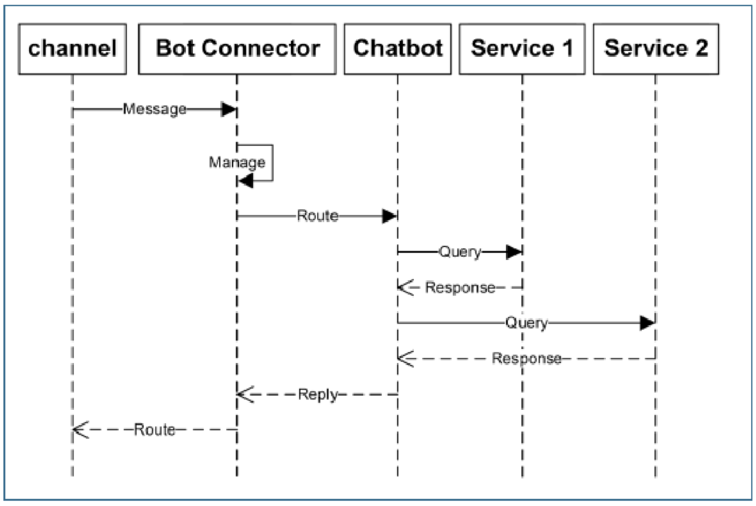


Figure 4.3: Components communicating with each another [19].

Figure 4.3 indicates not only the journey of the input text from the user on a particular channel to its response, but also that all separate components of the whole chatbot communicate across the Internet. Developers should be very careful when implementing external services into their chatbots. The Internet connection plays the major part here performance-wise. Poor Internet connection therefore results in the poor user experience, no matter how well-designed is the chatbot [19].

Now, that the components of the Bot Framework have been explained, let's have a closer look at the chatbot component itself. Bot Builder SDK (v3) offers developers a way to create their custom chatbots from the scratch.

4.1.1 Bot Builder

Chatbot is in our case a model view controller (MVC) web application[1]. Every chatbot has the *MessagesController* class that contains the *Post* method. This is the place, or the endpoint, that the Bot Connector communicates with. It receives activities that are processed and then passed to the *Dialogs* that manage the conversation flow. It is also primarily responsible for creating a typing activity so that a user sees that the chatbot is typing while it is really creating a response for the user.

All the methods to manage the conversation flow as well as its state are contained by a class called *Dialog*. It has the *Serializable* attribute decoration, since every dialog needs to be serialized. It is because the framework transfers the dialog along with its state across the Internet. Every dialog must at some point implement the *IDialog* interface. At some point because dialogs, just like any other classes can inherit from each other. *IDialog* has a *StartAsync* member with one *IDialogContext* parameter. It is the first method that is called when entering the dialog [19].

```
public async Task StartAsync(IDialogContext context)
{
    context.Wait(MessageReceivedAsync);
}
```

[1]https://en.wikipedia.org/wiki/Model-view-controller

The *IDialogContext* implements interfaces with members responsible for the conversation flow. Particularly it implements *IDialogStack* and *IBotContext*. Figure 4.4 shows the relationships between the interfaces.

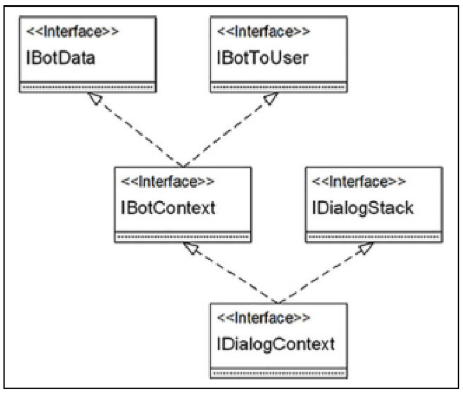


Figure 4.4: Hierarchy of the *IDialogContext* interface [19].

The *IDialogStack* is responsible for managing the stack of the dialogs since one dialog can call another one and thus creates a FIFO stack[2]. This calling is done by *Call* member. The opposite action, ending the dialog is executed by *Done*, popping it from the stack. Another very important member of this interface is *Wait* which is also used in the code snippet above. It specifies, which method will be next to call in the dialog. In this example, it will wait for the activity coming to the chatbot. Most likely it will be a text from the user, but as explained before, it can be also an event or any other type of activity.

As for the *IBotContext*, its most important part is the reference to the current activity. The last of the most important members of the *IDialogContext* interface is the *PostAsync* member, which is part of the *IBotToUser* and is responsible for posting an activity back to the user. Keep in mind, that this activity can be again not just a message, but an event as well [19].

4.2 Language Understanding Intelligence Service

As discussed in the previous chapters, communicating with machines so they understand what humans say is no easy task. There are techniques how to get intents and entities from user input that were also covered previously. Microsoft offers its users a service, that does all the work for them. It is called Language Understanding Intelligence Service, or LUIS. It is a part of Microsoft's Cognitive Services which also include functionality like speech-to-text service and so on.

Alongside with intents and entities there is the third major term, *utterance*. It can be understood as the text that the user types. Intents and entities are then extracted from these utterance texts received from Bot Connector. These extracted entities and intents are in a form of JSON text that LUIS returns. Here is an example of such JSON from LUIS documentation [22]:

```
{
  "query": "I want to call my HR rep.",
  "topScoringIntent": {
```

[2]https://en.wikipedia.org/wiki/FIFO_(computing_and_electronics)

```
    "intent": "HRContact",
    "score": 0.921233
  },
  "entities": [
    {
      "entity": "call",
      "type": "Contact Type",
      "startIndex": 10,
      "endIndex": 13,
      "score": 0.7615982
    }
  ]
}
```

LUIS API can be used in any application like mobile or web apps, not only in chatbot. However, Bot Framework integrates LUIS as part of its features, which means it is able to convert these JSON texts into predefined objects. The framework can then provide the data in form of parameters for a chatbot. This process is completely covered by the framework, so the developer does not need to worry about that. He or she just needs to specify the logic, typically the conversation logic, based on this information input [19].

Microsoft offers very intuitive web user interface to work with LUIS[3]. After signing in, users can create a new app where they can train their model. A model is a set of entities, intents as well as the utterances which are used to train the model. In this context, a model refers to the same thing as LUIS app, since it is just a single machine learning model. After creating the model, users can create intents. As described before, intent, as the name suggests, is the users intention or goal which they desire to accomplish. This intent is comprised of its name and the probable utterances that user may typically say. The more utterances the developer has, the better. However, there are sometimes no previous sample data. This is the case of the chatbot described in the thesis. Nevertheless, it is a good practice to create at least five utterances for an intent [19].

After creating intents, user may define their custom entities or use prebuilt ones such as numbers or dates. The next step following the creation of the entities is to label them in the utterances of the intents. Labeling is very user friendly, an user just needs to select one or more words and then pick from the list of entities as shown in figure 4.5. The same rule applies here as well, the more labeled entities, the better is the accuracy when recognizing users real input texts. In the JSON example provided before, the entity called *Contact Type* was recognized from the input text. Instead of the word *call*, there could be words like *message* or *email* which would result in a very similar JSON.

When the model is completed, which means it has sufficient number of utterances for all the intents the developer designed, it can be trained. Training is easy one-click action on the button *Train*. After the model is trained, it is ready to be deployed. However, there is also an option to test it first. A developer just needs to open the train tab and type any question or text, that a user might ask. This test inputs should be different from the utterances that the model is trained upon. If the results are relevant, there is no other obstacle to publish the app. After publishing, the LUIS app is ready to be used in an application providing the endpoint, model ID and subscription key. These can be found in the settings menu.

Models should be designed very carefully since it is very easy to train them in a wrong way so that they will provide unexpected results. Another important note to keep in mind

[3]https://www.luis.ai/

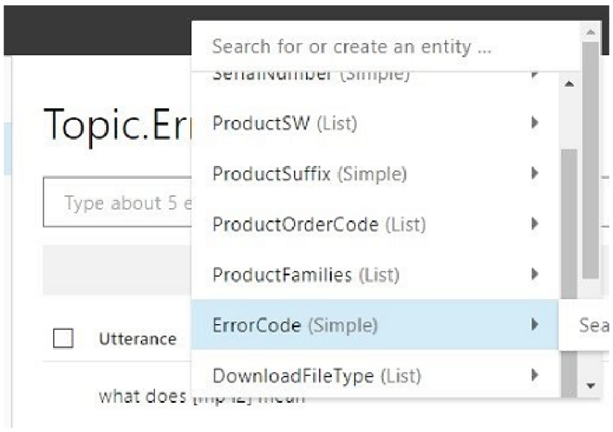


Figure 4.5: Labeling two words *mp l2* as a custom entity called *ErrorCode*. This entity is mainly used in Troubleshooting section of the InteliBot.

is that LUIS service is free only up to five transactions per second. If the application is expected to be used more frequently, better instance needs to be bought[4].

4.3 Azure Text Analytics

This is another service which is used for working with a text. It has four main features which include: entity recognition, sentiment analysis, language detection and key phrase extraction. The last of the mentioned features will be used in the algorithm described in section 5.1. The techniques used for key phrase extraction in this service are very similar to those in LUIS. The difference here is that the user cannot add their own training data or machine learning models. The models are pretrained by Microsoft so that they can be used straightaway [23].

4.4 Azure Search

It is also a cloud-based service offered by Azure. Azure Search is based on Lucene search explained in the previous chapter. It exposes and extends some functionality of the Lucene search for scenarios that are usable for Azure Search. There are four steps when executing a query:

- Query parsing
- Lexical analysis
- Document retrieval
- Scoring

Figure 4.6 shows the process of how the results are retrieved from the search request. A search request is a query with parameters, query terms, filter expression or ordering rules [20].

[4] https://azure.microsoft.com/en-us/pricing/details/cognitive-services/language-understanding-intelligent-services/

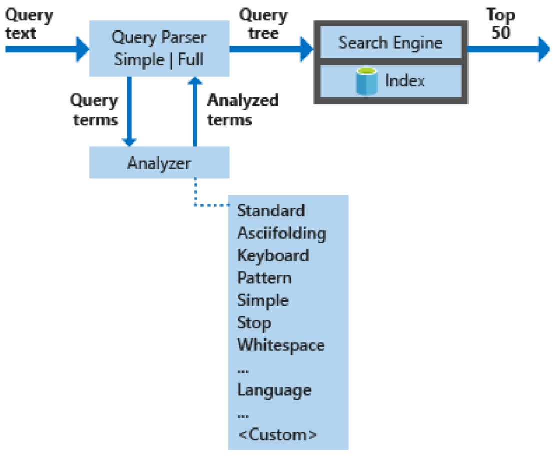


Figure 4.6: Azure search architecture from original documentation [20].

Parsing the query text is the first step when processing a full text search query in order to extract terms. These terms are then used when retrieving documents from the index. To capture as many matches from the index as possible, query terms are sometimes divided and reorganized to new ones. The best matches are then sent to the application [20].

When parsing the query, two languages are supported. The parameter *queryType* in the search request can be set to *simple* which is the default value, or *full*. The simple one supports similar functionality as users know from web search engines. It is intuitive and often there is no further need of any client-side processing. Full mode adds support for other operators such as fuzzy, wildcard or regex. There is another parameter *searchMode* responsible for Boolean queries. When set to *any*, it reflects *or* operator within the words. When set to *all*, explicit specification of Boolean operators is needed. In this case, a space means *and*. Query tree is the outcome of this process and can be understood as sub-queries. A sub-query can have different types such as phrase query in case of quoted terms, term query in case of standalone terms and so on[5] [20].

The structured query tree is processed by lexical analyzers. They take the input from the parser, process it and the tokenized terms are sent back to be integrated to the tree. Based on specific rules, the query terms are transformed to a given language. This process is called linguistic analysis and includes:

- Getting the root form of a word
- Getting rid of unnecessary words such as *the* or *and*
- Retrieving components from composite words
- Changing upper case words to lower case

This analysis is only applied to complete terms meaning term or phrase queries, thus prefix, wildcard, regex or fuzzy queries are not supported. They bypass this stage except lowercasing the words [20].

The next stage is document retrieval. As mentioned in section 3.4 about Lucene, retrieving means finding documents within the index. Retrieving is best explained by indexing

[5] `https://docs.microsoft.com/sk-sk/azure/search/query-lucene-syntax`

itself. Every searchable field has its inverted index. Index contains a list of terms from all documents. Terms map to the list of documents where they appear. In order to create terms of the index, the engine executes very similar lexical analysis as in the previous step. It is a good practice, although not necessary, to use the same analyzers for indexing as well as for the search. Terms within the index and query terms look then more alike [20].

The last step is to score the matches based on their statistical properties. Term frequency–inverse document frequency[6] or TF/IDF is responsible for this task. However the scoring formula has more to it than TF/IDF. Field length as well as other factors play a role here[7]. The scores can be then tuned by two ways: adding scoring profiles to the index[8] is the first option and term boosting is the second one. It provides an operator "^" which boosts the terms that need to have higher scores. However, this operator can be used only in full Lucene syntax by setting the *queryType* parameter to *full* [20].

It is also important that this service is free only for limited functionality. Chatbots however usually need to have better tier which is paid[9]. This is also the case regarding this chatbot.

[6]https://en.wikipedia.org/wiki/Tf-idf

[7]https://lucene.apache.org/core/4_0_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html

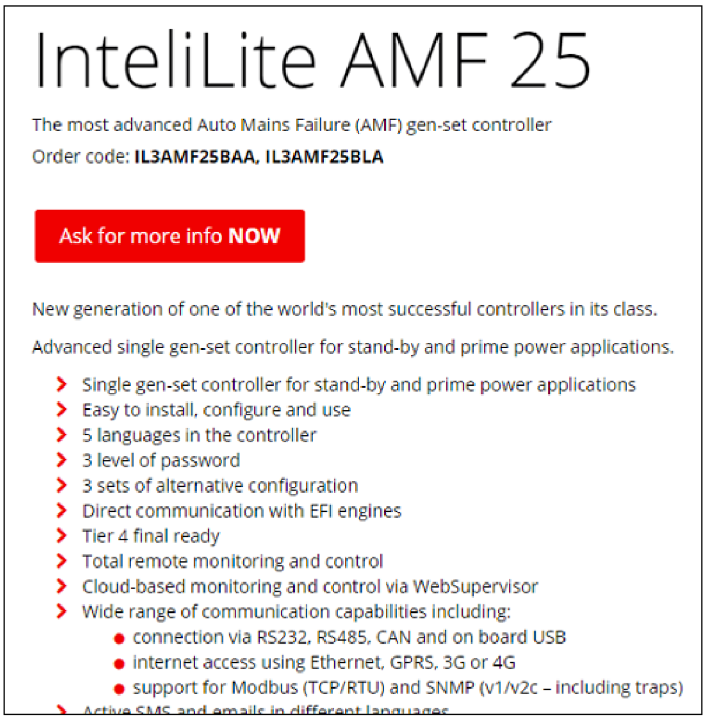
[8]https://docs.microsoft.com/sk-sk/azure/search/index-add-scoring-profiles

[9]https://azure.microsoft.com/en-us/pricing/details/search/

Chapter 5

Implementation of chatbot for information search

The main goal of the chatbot described in the thesis, also called InteliBot, is to provide support for customers and technicians of a company called ComAp[1]. The main business of the company is to sell controllers, mainly used in the electricity area. For better imagination, a good example of such controller is that when the main source of electric power in a hospital is out, the controller immediately resolves the problem by activating the backup generator to generate electricity. Hospitals can under no circumstances be without electricity at any time for obvious reasons. The company also provides further support for such controllers all over the world.



InteliLite AMF 25

The most advanced Auto Mains Failure (AMF) gen-set controller

Order code: **IL3AMF25BAA, IL3AMF25BLA**

New generation of one of the world's most successful controllers in its class.

Advanced single gen-set controller for stand-by and prime power applications.

- Single gen-set controller for stand-by and prime power applications
- Easy to install, configure and use
- 5 languages in the controller
- 3 level of password
- 3 sets of alternative configuration
- Direct communication with EFI engines
- Tier 4 final ready
- Total remote monitoring and control
- Cloud-based monitoring and control via WebSupervisor
- Wide range of communication capabilities including:
  - connection via RS232, RS485, CAN and on board USB
  - internet access using Ethernet, GPRS, 3G or 4G
  - support for Modbus (TCP/RTU) and SNMP (v1/v2c – including traps)
- Active SMS and emails in different languages

Figure 5.1: InteliLite AMF 25 specification.

[1]https://www.comap-control.com/

InteliBot has four main sections. However, only one of them is subject of the thesis. The section is called "I'm interested in...". This is mainly used by so called buyers of the organizations. In the case of previously mentioned hospital example, these people are responsible for purchasing the equipment that the hospital requires. This equipment may include things such as X-ray generators, operating tables and many other devices used by the hospital staff.

ComAps controllers[2] are one of such things, although it is not the first thing that comes to mind when talking about devices used in hospitals. The buyers typically have no technological knowledge of such controllers. This section is primarily used to provide the information about them as well as other products of the company[3]. This information can be found at the particular product website in a form of bullet points of the products' features as seen in figure 5.1. The user does not need to visit the website to find the feature of the product, he or she can simply ask the chatbot about it. An example of this might be to ask, whether the controller works in low temperatures. Users have many ways to ask about this particular feature. Another way to ask about the same feature would be to ask whether the controller is capable of working in the cold weather. However, sometimes there is a need to find the right products based on some feature. Another part of the section is to give the user a reference to solutions the company provides on different areas[4] if the user asks. This is done by offering the links to particular solutions in form of buttons the users can click on in the bot. InteliBot is designed to understand these questions and provide a suitable answer.

There are two major steps to achieve the task of finding the required information. The first one is to create an algorithm which delivers the most relevant results based on the user input. The next step is then implementing this algorithm into the chatbot's dialogs. The implementing phase goes with continuous testing and fixing found bugs as well as implementing suggested improvements. This will be described in a moment.

To get the idea about the whole InteliBot, other three sections will be briefly described. The first section is the password issue section, which handles password related problems such as resetting password. The next one is download files section. Users can download product-related files there without the need to visit the product website. Then there is a troubleshooting section where error codes or other problems are resolved. It is much faster to use the bot for these issues than looking into the datasheets or manuals of the controllers and find what particular error code means. These sections are mostly used by the technicians that are aware of the products and their behaviour or technical details.

5.1 Algorithm

The basic concept of the algorithm is simple. The heart is the Azure Search described in section 4.4. The goal is to get the best possible result from it based on the input that the user provides. In order to do so, certain steps need to be done. The most important step is to create a good search query from the input. The utterances that come to the bot are typically not suitable texts for such query. Passing the whole sentence that the user types will not lead to relevant results from Azure Search. More often than not, there will be no result at all. That is where Azure services LUIS and Text Analytics need to be used. This will be described in a moment. The basic concept of the algorithm is shown in figure 5.2.

[2] https://www.comap-control.com/products/controllers
[3] https://www.comap-control.com/products
[4] https://www.comap-control.com/solutions

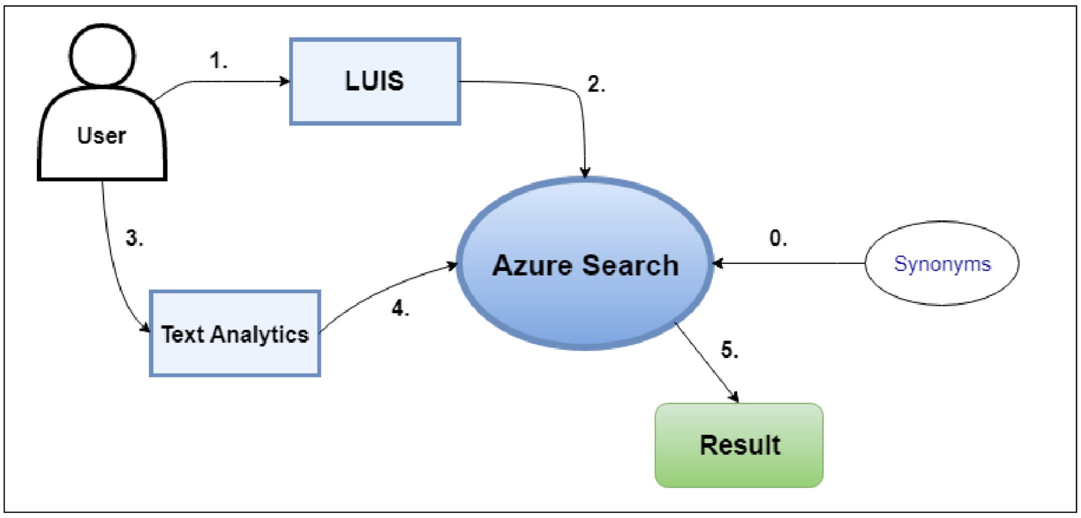
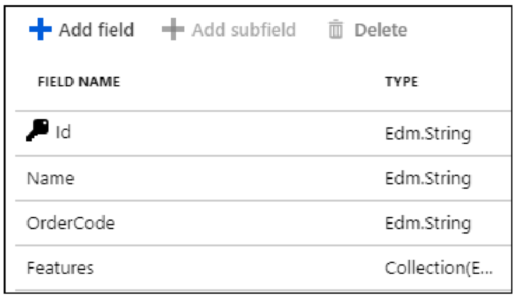


Figure 5.2: Simple algorithm illustration.

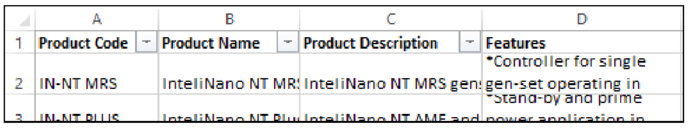
First of all, the Azure Search indexes for product features and solutions need to be filled with the data. As for the features of the products, the structure or the fields for a single document representing a single product are: *Id, Name, OrderCode, Features* as seen in figure 5.3. The first three are of type *string* and refer to the product itself. The *Name* field is the full name of a product. The *OrderCode* field is the unique identification of the product. For every product it can be found on its web page. The last string field is the *Id* which is a key field for the index and is generated from the *OrderCode* field. The last field *Features* is a list of the features for the product. The features reside at every products web page in form of bullet points as stated before.



FIELD NAME	TYPE
Id	Edm.String
Name	Edm.String
OrderCode	Edm.String
Features	Collection(E...

Figure 5.3: Product features document structure from Azure Portal.

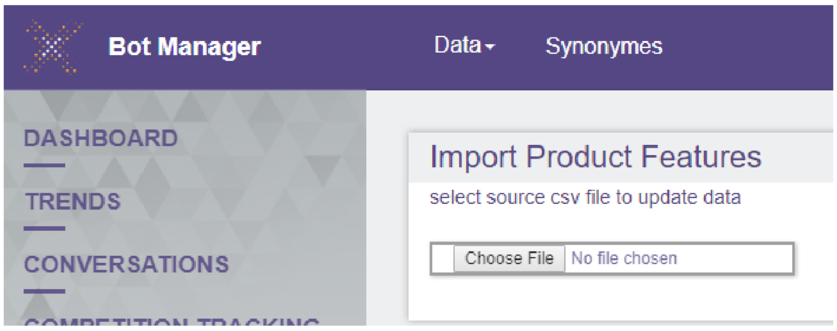
In order to get the structured data for the index, it was decided to manually create an Microsoft Excel file which contains very similar fields as needed for the index documents. This way it is easy to import the data from the Excel file to the index. Microsoft Excel offers exporting XLSX files, which is a default Excel file format, to comma-separated values (CSV). The import is done by a web tool called BotManager. There is a field to select a CSV file for uploading the product features as seen in figure 5.5. This tool was created by other members of the Xolution development team.



	A	B	C	D
1	Product Code	Product Name	Product Description	Features
2	IN-NT MRS	InteliNano NT MR:	InteliNano NT MRS gen:	*Controller for single gen-set operating in
3	IN-NT PLUS	InteliNano NT Plu	InteliNano NT AMF and	*Stand-by and prime power application in

Figure 5.4: Products features Excel file structure.

Uploading the solutions[5] is very similar to the previous process. The difference is that the documents have different structure. The fields are: *id, name, webPage, phrases*. The *name* field is the name of the solution, *id* is as in the previous case the key identifier for the index and is generated from the name. The *webPage* is a link to the particular solution web page and the *phrases* field contains phrases by which the solution can be found.



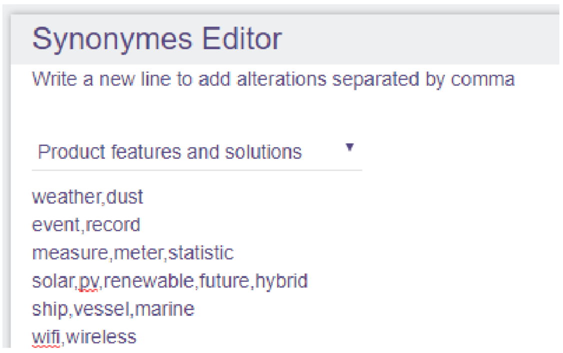
Import Product Features

select source csv file to update data

No file chosen

Figure 5.5: Product features file field in BotManager.

Another important step to get the Azure Search ready for information search is to upload synonyms into it. Azure Search offers an option to do so by adding synonym maps. It is done by API call, however, BotManager provides a user interface for this task. Words in one row, separated by commas or a special operator ”=>” are one set of synonyms. When comma is used, the query will expand the term with *OR* operator between the synonym phrases. The operator ”=>” is responsible for replacing a term sequence of a query that equals to left hand side, to whatever is on the right hand side of the operator [21].



Synonymes Editor

Write a new line to add alterations separated by comma

Product features and solutions

weather,dust
event,record
measure,meter,statistic
solar,pv,renewable,future,hybrid
ship,vessel,marine
wifi,wireless

Figure 5.6: Synonyms input field in BotManager.

The last visible row of the input from figure 5.6 means, that whenever there is term *wifi* in the query, it will expand to *wifi OR wireless*. The very same expansion would be the result if there was *wireless* in the query. An example of the ”=>” operator below shows

[5] https://www.comap-control.com/solutions

that query *Los Angeles* will be always rewritten to *LA*. It is however not applicable in the opposite way, so *LA* will not be replaced with *Los Angeles*.

Los Angeles => LA

After the Azure Search indexes are prepared, search queries can be sent into them. These queries need to be constructed from the input text. However, not every user input is meant for searching the information, thus retrieving documents from the indexes as will be shown in the implementation section. The basic concept of creating a query is again simple. For product features it is as follows: pass the utterance through the LUIS, get so called *keywords* from it and try different logic combinations as a query. If there is not a single feature found, repeat the steps and instead of LUIS use the Azure Text Analytics. Its key phrases extraction function is used in this case to extract keywords. Again, if no features were found, either there are really no such features that the user asked for in the particular product, or the algorithm was simply unable to find them. Typically it is the first case scenario, but the algorithm is not perfect either, and may sometimes fail.

As for the solutions part, the algorithm is very similar to product features. The input is sent to LUIS, keywords are extracted, various combinations of these keywords are sent to the Azure Search. However, if there is no success, the Text Analytics is not used. Instead, the algorithm tries to send the original input with *OR* operators between the words into the Azure Search. No success again means that there is really no answer for the user question or the algorithm failed again. Moreover, no success on solutions usually means no answer to the user question whatsoever, since the solution search typically follows the product features search as will be shown in the next section.

5.2 Implementation

The whole chatbot is implemented in C# programming language as it is supported by Bot Builder SDK (v3). Finding the right information from the Azure Search, whether it is product features or solutions is done by multiple methods. There is no need to describe all of them, however there are two methods, that handle the algorithm itself. One for product features and one for solutions.

5.2.1 Services

Each of these two methods resides in separate services. There is a service for features called *ProductFeaturesService* and a service for solutions called *SolutionService*. The services are components of the repository pattern which is part of Domain-Driven Design[6]. The architecture is shown in figure 5.7.

Product features

Method *SearchProductFeatures* is the heart of finding the features. It takes one mandatory parameter *textToSearch* and several arbitrary parameters, so that certain steps of the method can be skipped as the information is already passed to the method. One of these arbitrary parameters is *productCodes*, which is usually set to some product. The *textToSearch* is of a type *string* and represents the input, where the features should be found. The input

[6] https://en.wikipedia.org/wiki/Domain-driven_design

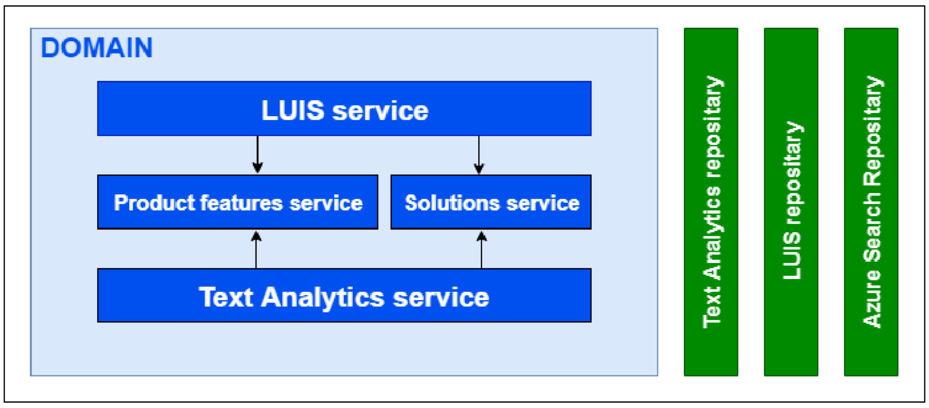


Figure 5.7: Architecture of the services.

goes first to the LUIS app called Web Sense. The app is very simple, it has only two intents. *Question* intent for possible user questions and *None* intent for anything other than that. It also has only three entities. *KeyWord* entity is the important one. It represents the keywords that are then passed to the Azure Search.

Figure 5.8 shows the basic idea behind the service.

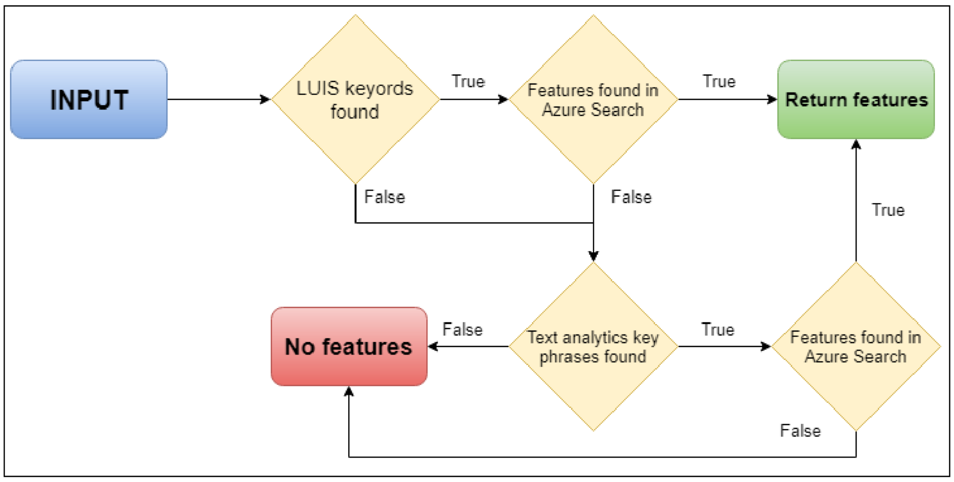

INPUT
LUIS keyords found
True
Features found in Azure Search
True
Return features
False
False
True
No features
False
Text analytics key phrases found
True
Features found in Azure Search
False

Figure 5.8: Product features service illustration.

If the LUIS app finds a single keyword entity containing only one word, there is nothing left than just to try finding the result based on this term. If any features are found, they are returned as a result of the method. This applies for the whole method, as soon as a feature, or more of them are found, they are returned and the method ends. If there are more words in a single entity found, they are combined to search queries. The principle here is to go from the most strict one, which is *AND* operator between the words. If there is no result, then *OR* is tried and if this fails, then fuzzy logic is applied. However there must be more than five characters in the entity for the algorithm to apply the fuzzy logic. Developers do not need to worry about implementing fuzzy logic. According to Azure Search query syntax[7], placing tilde character at the end of the term does the job. The official documentation for the query syntax contains an example that query *blue~* will result in *blue, blues* and *glue*.

[7]https://docs.microsoft.com/en-us/azure/search/query-lucene-syntax

A real life example:

- A user types "IL3AMF25BLA" which is an exact order code for product IL AMF 25 LT[8]. This way, a product context is set to the product.
- InteliBot asks the user, what would they like to know about the product.
- The user types "Does it work in low temperatures?".
- LUIS detects *low temperatures* as a key word entity.
- The service tries to send the "low temperatures" query to Azure Search as a single space represents *AND* operator. The query also contains a filter where the product code is specified.
- Azure Search returns no result.
- The service now tries query "low|temperatures" as "|" represents *OR* operator. The product code is again set to the same value.
- Azure Search returns one result and the conversation continues by printing the feature and asking further questions.

If more than one entity is found, very similar principle is applied. This time however, *AND* and *OR* operators are placed between the words from all the entities together. Fuzzy logic is not applied here. No result leads to the next step, which is to use Azure Text Analytics instead of LUIS app. The very same process is done here with the found key phrases. The last step of the method is to check the input as is, but only if it has one or two words. It is based on the assumption that the input contains the feature, not the whole sentence.

There is one other method that is important to mention. The method *GetProductsByFeatureAsync* with a single input parameter does the opposite job. It takes the features and finds all the products containing it. The input is also checked on keywords which are then sent to Azure Search with *AND* logic. If there are no keywords found, the whole input is checked on products.

Solutions

The principle of the method *SearchSolutions* is basically the same as it is in *SearchProductFeatures*. There are two main differences. The first one is that after no success from LUIS keywords, Azure Text Analytics is not used. Instead, it tries the original input with *OR* logic. The next difference is that a threshold is added to every search case. It is set to 10%. This value was set based on the experiments, where it was common that the Azure Search returned too many documents, where most of them were with a score less then 0,1. The basic concept of the service is shown in figure 5.9

As the information retrieval is ready, the next section will explain how it is used in the chatbot along with the conversation logic.

[8]https://www.comap-control.com/products/detail/intelilite-amf25

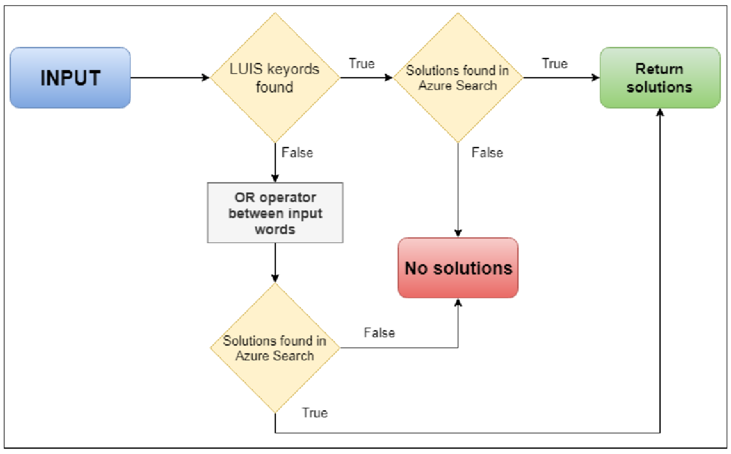
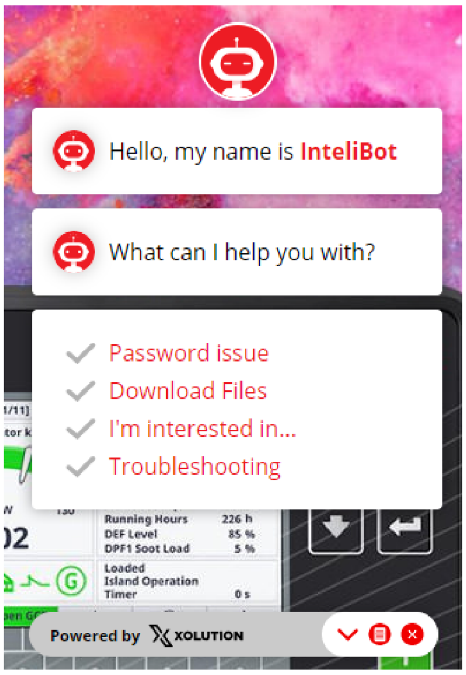

INPUT
LUIS keyords found
True
Solutions found in Azure Search
True
Return solutions
False
OR operator between input words
False
No solutions
Solutions found in Azure Search
False
True

Figure 5.9: Solutions service illustration.

5.2.2 Chatbot

In section 4.1 we discussed that *MessagesController* is the place that *BotConnector* communicates with. When the chatbot starts, the very first *activity* that comes to the bot is of type *conversationUpdate*. This is a signal that a conversation started. The controller handles the control of the conversation flow to the *RootDialog*. This dialog implements *IDialog* interface, so it has some members available as explained before.


Hello, my name is InteliBot

What can I help you with?

Password issue
Download Files
I'm interested in...
Troubleshooting

Figure 5.10: InteliBot intro messages.

Figure 5.10 shows, that the very first question that InteliBot asks is to choose a category. The choice is between four categories, or sections described at the beginning of the chapter. The selection is in form of buttons as also seen in the figure. The Bot Framework offers class *HeroCard* to work with buttons. A new object of this type with all the necessary properties needs to be created and passed as a parameter to *PostAsync*. This method, which is a part of *IDialogContext*, specifically *IBotToUser*, takes not only a plain string but also HeroCards and other types which it then sends as a message to a user. InteliBot has a wrapper called *MessageSender* for the function of sending messages, HeroCards and even events.

After the section pick, an activity is sent to the *RootDialog* where the text of the activity is checked on the section. At this point, typing anything is disabled for a user, so that he or she has to click on one of the options.

```
else if (activity.Text == ConversationSection.InterestedIn.Description())
{
    conversationSection = ConversationSection.InterestedIn;
    await InviteTheUser(context);
    await LogConversationStart(context, activity.Conversation.Id,
    conversationSection);
    context.Call(dialogFactory.Create<InterestedInDialog,
    QueryRecognitionResult>(null), AfterInterestedIn);
}
```

As shown in the code snippet above, if the section is "I'm interested in...", a dialog *InterestedInDialog* is created and called. The creation is designed through factory method design pattern[9]. *AfterInterestedIn* is a method that will be executed after the dialog ends. The dialog in this case can end with a *true* or *false* values passing them to *context.Done* member. The Boolean value specifies, whether any logical error occurred during the dialog flow. The example of an error might be that the user asks a question the InteliBot has no answer to multiple times. Dialog *FeedbackOkDialog* or *FeedbackErrDialog* are called based on this value, collecting the feedback from the user and specifying what will be next to happen.

The dialog flow of the *InterestedInDialog* dialog is shown in figure 5.11 although it does not cover some scenarios such as yes/no responses that are described later. The heart of the dialog is *Message* method. After the intro message from InteliBot which is simple „What information are you looking for?“, a user reply goes here. It is then checked, whether the message contains any product. If yes, *ProductIdentificationDialog* is called. Identifying the product from the text as well as *ProductIdentificationDialog* are very sophisticated systems and were mostly implemented by other members of Xolution development team. If a product was successfully identified from the input, the next step is to check the keywords for the features. If any, the *ProductFeaturesService* is called and features are being searched for, and the result is sent to the user. It is important to mention, that product context is saved here by setting a private field of the dialog. This is because a user may ask about another feature without specifying the product again as shown in figure 5.12. The keywords are saved too. This is for the opposite case, if the user asks about a feature and his or her next question is about the same feature but different product, without actually mentioning the feature.

Figure 5.12 shows the question „Do you want to know anything else about InteliLite AMF 25:“. It is very common for a user to answer such questions with yes or no instead of writing another question directly. This is also checked in the *Message* method by checking the LUIS intents explicitly. InteliBot uses different LUIS app here than it uses for keywords. This LUIS app is general for the whole chatbot and contains intents such as *Answer.Yes*, *Answer.No*, intents important for other sections as well as entities important for product recognition. However, this is not the only place where there are these yes/no controls. On

[9] https://en.wikipedia.org/wiki/Factory_method_pattern

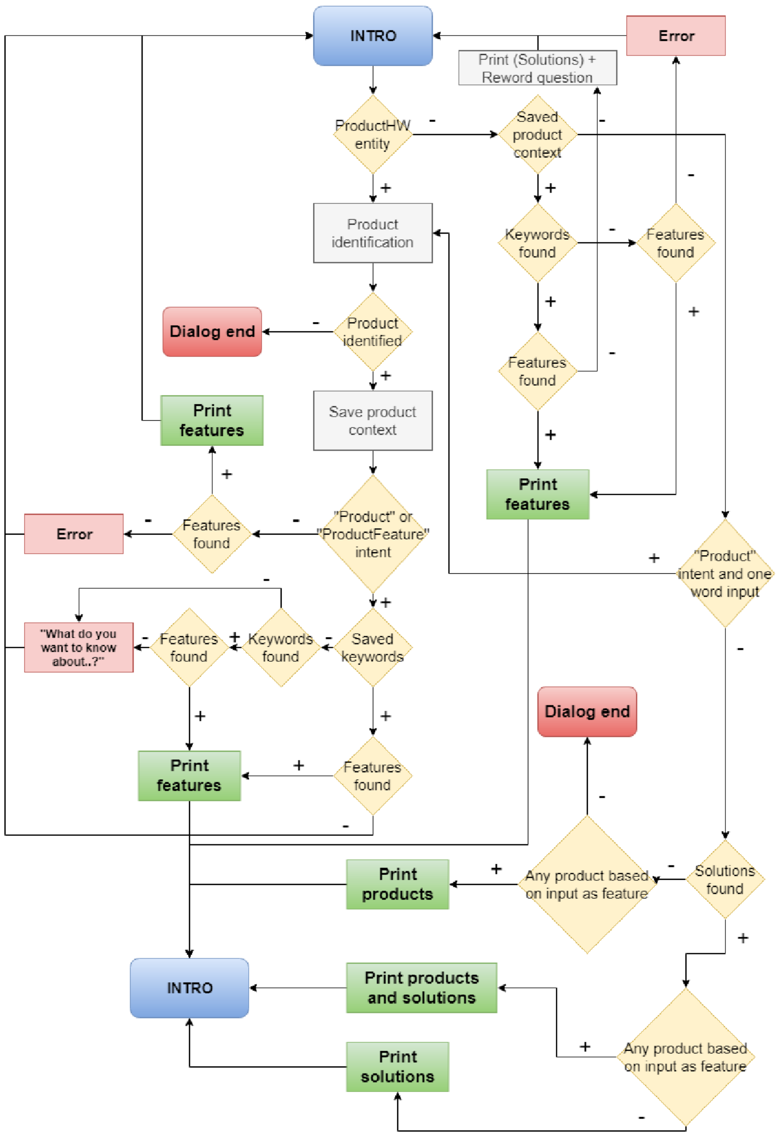
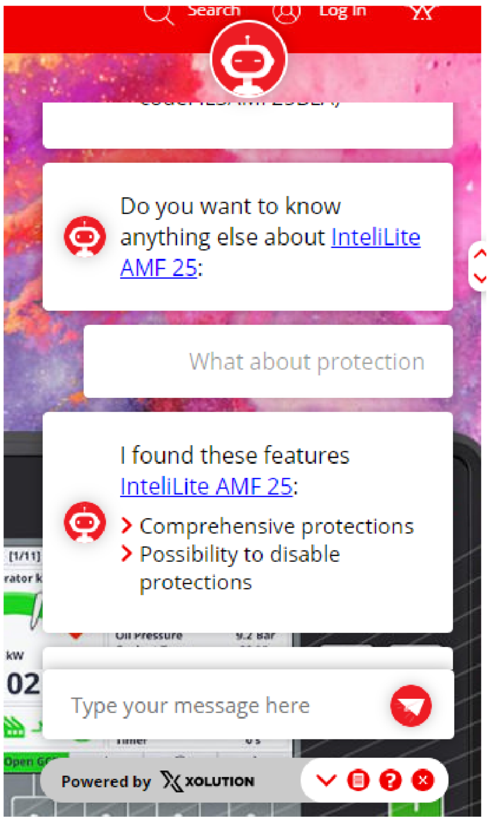


Figure 5.11: InterestedInDialog diagram.

various places in the dialog, instead of *Message* method, it is specified to call *MessageReceived*. This is where the integration of LUIS into Bot Framework comes in. Methods with a LUIS attribute specifying an intent are called when there is an intent match with the users input. This is possible because the dialog implements *LuisDialog* interface. It is shown in the code listing below. In this example, the method executes when the input intent is *Answer.No*. This is typical after questions such as ”Do you want to know anything else about {PRODUCT}:” when yes or no answers are expected.

```
[LuisIntent(LuisConstants.Intents.Answer.No)]
public async Task NoneOfAbove(IDialogContext context,
IAwaitable<IMessageActivity> activity, LuisResult luisResult)
{
    if (askedAgain)
    {
        askedAgain = false;
```


Do you want to know anything else about InteliLite AMF 25:

What about protection

I found these features InteliLite AMF 25:
- Comprehensive protections
- Possibility to disable protections

Figure 5.12: Asking about other information.

```
        currentProductContext = null;
        await AskIntroQuestion(context, true);
        return;
    }
    context.Done(false);
}
```

If no product was identified from the input, no product context is saved, the input is checked whether it contains any product features. This is the case when the user asks for products containing certain features. If there are no features in the input text, solutions are checked through *SolutionService*. Sometimes, there is not any information found whatsoever. InteliBot then asks the user to try to reword the question. After two proceeding unsuccessful answers, the dialog ends with an error and *FeedbackErrDialog* is called. On the other hand, in certain situations, the bot waits only for a minute for a user to answer. After one minute of inactivity, the dialog ends successfully, meaning there is *true* value in *context.Done*. This triggers *FeedbackOkDialog* from the *RootDialog*.

Figure 5.13 shows the overall architecture of InteliBot. The contribution of the author is also shown by green colour of the filling of the components. Orange components mean no contribution at all. Components without any colour marks are not part of the thesis, although there was, sometimes major, contribution to them.

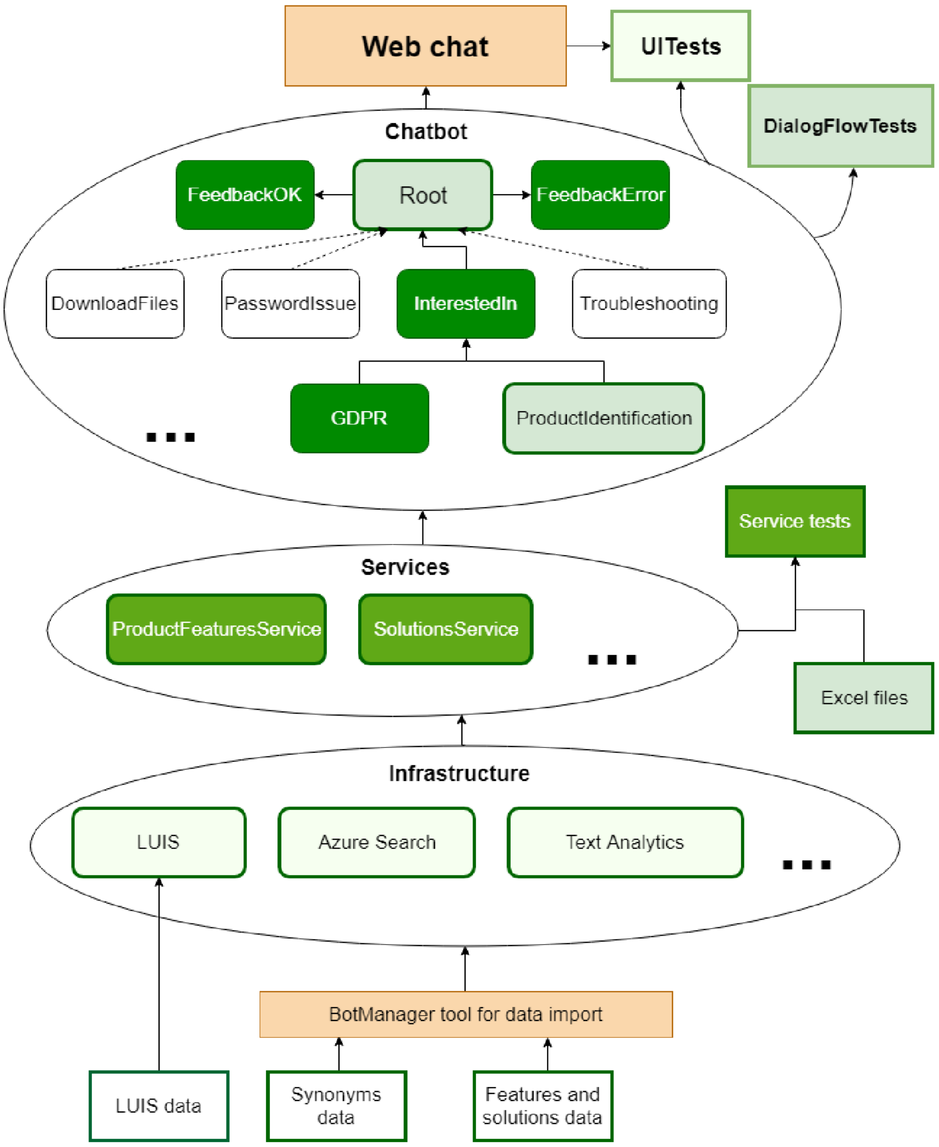


Figure 5.13: Illustration of InteliBot architecture with the contribution of the author (green: author's work, orange: others' work, white: author's work not covered by the thesis).

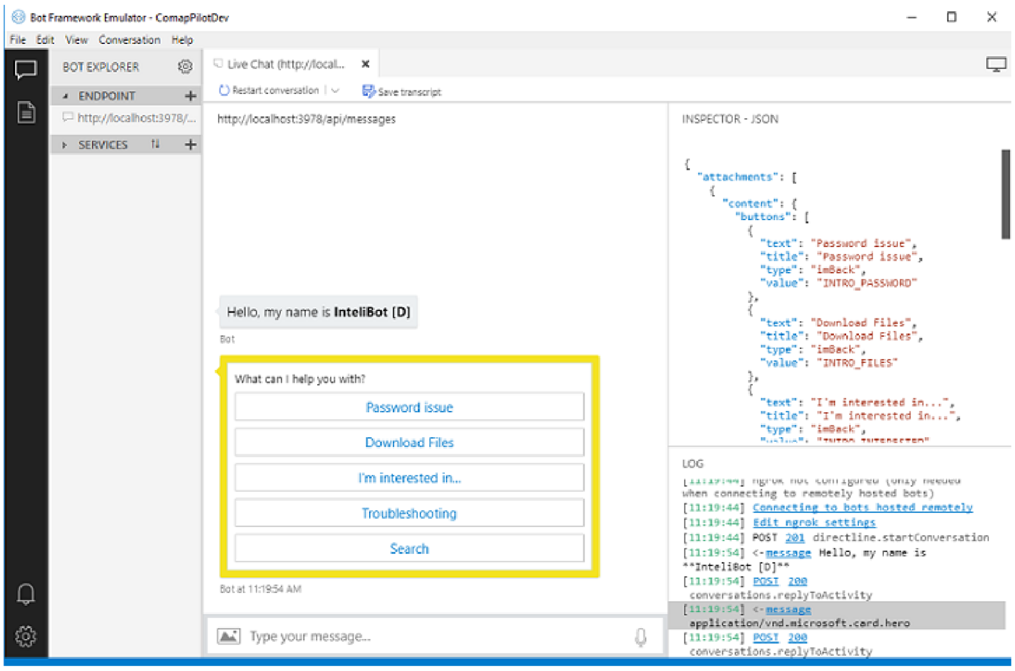
Chapter 6

Testing

The testing phase can be divided to two parts. The first part is the testing during the development by developers themselves. The second one is user testing, where users do not have any knowledge of the implementation.

6.1 Development phase

It is obvious that when developing an application, the developer wants to constantly test whether the implementation works in a way he or she desires. No matter how good the developer is, testing every change made is a good idea. This way bugs can be found at the very beginning. Chatbots are no exception here. However, deploying a chatbot to a communication channel after every change and then test it is not very efficient. Moreover, debugging is in this case even greater concern than the efficiency of the process.



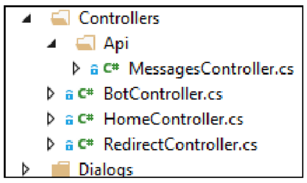
Hello, my name is **InteliBot [D]**

```
{
  "attachments": [
    {
      "content": {
        "buttons": [
          {
            "text": "Password issue",
            "title": "Password issue",
            "type": "imBack",
            "value": "INTRO_PASSWORD"
          },
          {
            "text": "Download Files",
            "title": "Download Files",
            "type": "imBack",
            "value": "INTRO_FILES"
          },
          {
            "text": "I'm interested in...",
            "title": "I'm interested in...",
            "type": "imBack",
```

```
[11:19:44] ngrok not configured (only needed
when connecting to remotely hosted bots)
[11:19:44] Connecting to bots hosted remotely
[11:19:44] Edit ngrok settings
[11:19:44] POST 201 directline.startConversation
[11:19:54] <-message Hello, my name is
**InteliBot [D]**
[11:19:54] POST 200
 conversations.replyToActivity
[11:19:54] <-message
 application/vnd.microsoft.card.hero
[11:19:54] POST 200
 conversations.replyToActivity
```

Figure 6.1: Bot Framework Emulator.

Microsoft Bot Framework luckily offers a tool for local testing the chatbot without the need of having a real communication channel. It is called Bot Framework Emulator and an example of it can be seen in figure 6.1. A developer needs to specify the endpoint to communicate with. When testing locally, the endpoint will be the *MessagesController* of the bot that is running on the localhost. This is of course a general rule for all the chatbots developed in Bot Builder (v3), no matter where they are currently running. In the example from figure 6.1, the endpoint is *http://localhost:3978/api/messages*. This means that the chatbot runs on port *3978* and the controller is specified as */api/messages*. As stated before, the chatbot is an MVC web application. Controllers do not have to have *Controller* suffix when specifying the URL, although it would run anyway. The *MessagesController* resides in *Api* folder of the chatbot project, thus the */api*. This is shown in figure 6.2.

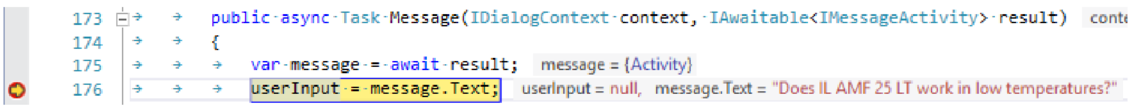


```
◢ Controllers
   ◢ Api
      ▷ C# MessagesController.cs
   ▷ C# BotController.cs
   ▷ C# HomeController.cs
   ▷ C# RedirectController.cs
▷ Dialogs
```

Figure 6.2: MessagesController location.

In addition to the endpoint, *Application Id* and *Application Password* need to be filled. These fields are found in the *Web.config* file of the chatbot project and are generated on the creation of the chatbot. When the configuration of the emulator is ready, the chatbot application is running on localhost, the testing can start. As seen in figure 6.1, the developer has other available features than just communicating with the chatbot. On the right side, the activities, that are in a JSON format, can be inspected. There is a log too, containing useful information such as HTTP return codes. The figure also shows two intro activities. They are almost the same activities as in figure 5.10 from the previous chapter. The only difference is the *Search* button which is there only for development purposes.

Debugging the application is very straightforward as well. Visual Studio integrated development environment (IDE) offers a concept of breakpoints that can be put anywhere in the code. The application pauses on the breakpoint hit so the developer can inspect the code. Figure 6.3 shows that the application is paused in the execution of *Message* method. Field *userInput* is still set to *null* since the code line has not run yet. However, the developer can see other fields values. In this case, field *message* already contains a value of type *Activity*. The next code line also shows the developer the already set value of *Text* property of *Activity* object.



```
        public async Task Message(IDialogContext context, IAwaitable<IMessageActivity> result)   conte
        {
            var message = await result;   message = {Activity}
            userInput = message.Text;   userInput = null,   message.Text = "Does IL AMF 25 LT work in low temperatures?"
```

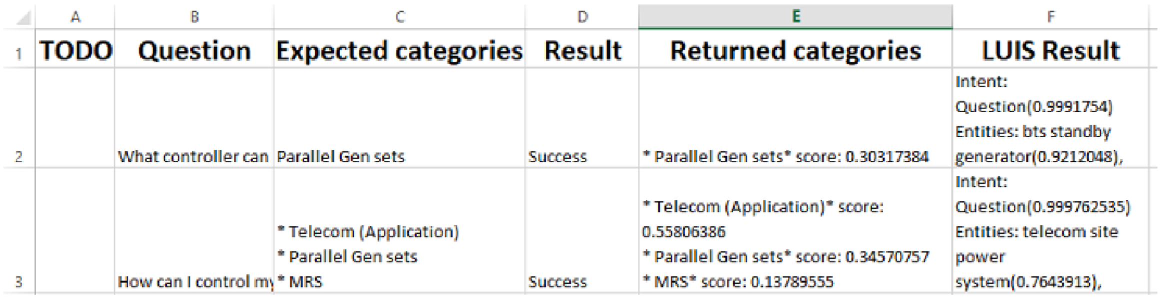
Figure 6.3: A breakpoint in *Message* method.

The combination of the emulator and Visual Studio IDE gives the chatbot developers a very powerful tool for chatbot testing as well as development. In case of InteliBot, the emulator is sometimes insufficient however. The reason is, that along with activities containing user and bot messages, there are also events and hidden data meant for the communication channel. In this instance, the channel is a custom made Web chat. It was developed by Xolution front-end development team. Sometimes, events such as disabling or enabling the user input need to be tested too. The easiest way of testing the chatbot through Web chat is to expose the localhost to the world. It is done by a tool called *ngrok*[1]. The new endpoint then needs to be set for the chatbot already deployed through Azure Portal. This way the chatbot no longer runs on the Azure where it was deployed, but wherever the developer specified, typically the localhost. Debugging through Visual Studio IDE is however the same.

6.2 User testing

Developers can create and test chatbots on their own. Nevertheless, the opinion from people that have nothing in common with the implementation is vital in any application development process. This makes sense, since users of chatbots should not have any instructions on how to use the chatbot ahead. The goal is to make the chatbot as intuitive and user-friendly as possible.

The whole InteliBot development process was managed by scrum methodology[2]. There is no need to go into deep details. The important part is, that the customer, ComAp, defines what they expect to be working in regular intervals called sprints. A single sprint usually takes two or three weeks. The requirements are then split to user stories. The example of such user story might be ”We want the bot to show the user the available solutions when a solution key word occurs in the input text and no product features were found”. Such user story is then divided to tasks like creating the corresponding service, implementing it into the InteliBot or testing. After every task is done, it is then tested. The testing of the task depends on its nature. When testing a service, a tester with no programming knowledge would not be very helpful. However, this issue was resolved by ”Excel tests”.



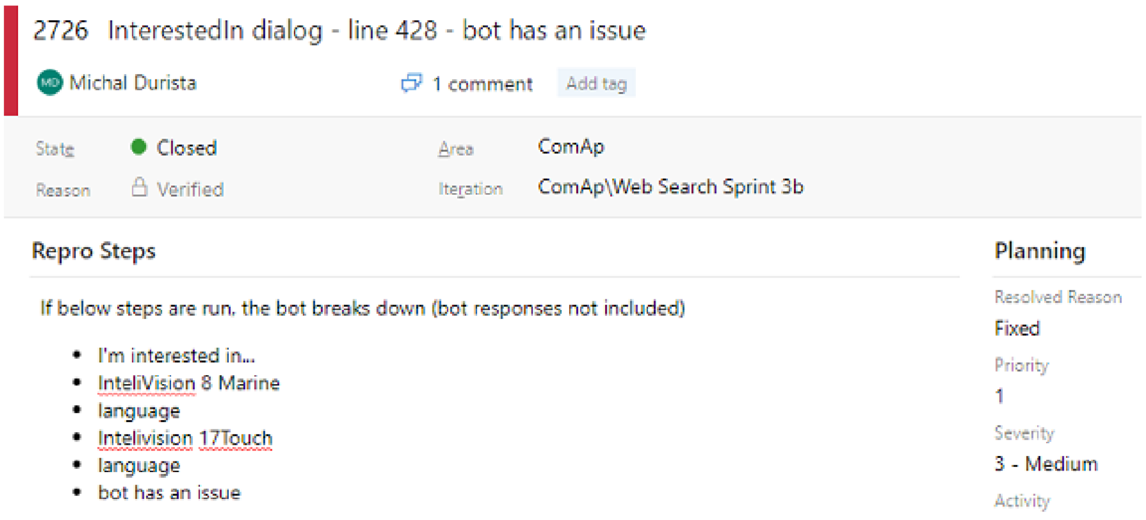
	A	B	C	D	E	F
1	TODO	Question	Expected categories	Result	Returned categories	LUIS Result
2		What controller can	Parallel Gen sets	Success	* Parallel Gen sets* score: 0.30317384	Intent: Question(0.9991754) Entities: bts standby generator(0.9212048),
3		How can I control my	* Telecom (Application) * Parallel Gen sets * MRS	Success	* Telecom (Application)* score: 0.55806386 * Parallel Gen sets* score: 0.34570757 * MRS* score: 0.13789555	Intent: Question(0.999762535) Entities: telecom site power system(0.7643913),

Figure 6.4: Solutions service Excel test example.

Figure 6.4 shows a part of ”Excel test” for the solution service. As can be seen, now everyone capable of working with Microsoft Excel can test the service. They just need to specify the question, what is expected to be returned, and run the test. The test even compares the result with expected answers and provides further information that might be helpful for a developer. There is more useful information not shown in the figure, such as what query went to Azure Search or at which point the algorithm returned the result. Provided the information, it is much easier to find the bug on test case failure. The tester then creates a bug and assigns it to a certain developer. The product features services, as well as other services not concerned by the thesis, were tested in a similar way.

[1]https://ngrok.com/
[2]https://en.wikipedia.org/wiki/Scrum_(software_development)

When all the conversation tasks are done, the conversation flow is tested manually at first. It is primarily done by Xolution testing team and even the project manager. If a bug was found, it is assigned to a developer along with the steps to reproduce it. This can be seen in figure 6.5.



2726 InterestedIn dialog - line 428 - bot has an issue

Michal Durista | 1 comment | Add tag

State	● Closed	Area	ComAp
Reason	Verified	Iteration	ComAp\Web Search Sprint 3b

Repro Steps

If below steps are run, the bot breaks down (bot responses not included)

- I'm interested in...
- InteliVision 8 Marine
- language
- Intelivision 17Touch
- language
- bot has an issue

Planning

Resolved Reason	Fixed
Priority	1
Severity	3 - Medium
Activity	

Figure 6.5: Example of a bug.

The whole user story has so called acceptance criteria, where major examples that must work are provided. If all the tasks and bugs are resolved, the criteria are met, the user story can be considered as done and it is closed. In certain intervals, the chatbot is deployed to the staging version, which goes to the customer so they can test it themselves. A presentation by the management team is followed eventually. The customer testing process involves primarily testing by the technical support team. It is then handled to various branch offices of the company for user acceptance test (UAT). Sometimes even selected distributors are included, mainly from Asian region, where chatting is more popular. The customer then sends the feedback after the process. Next user stories are then created based on it. This process repeats until the final version is released and even afterwords.

The manual testing is not the only way of testing the conversation flow. It is also done by end-to-end tests which are basically unit tests in .NET. They simulate real life communication with a bot. The messages are sent to the chatbot and the responses are compared with expected results. This way, when changing some part of the bot, in our case *InterestedInDialog* or related services, any unexpected behaviour resulted from the change can be detected straightaway. Code listing below shows a simple test with three user inputs. These are set as *Text* property of *toBot* activity on lines 18, 21 and 26. The bot responses are checked on various criteria such as number of returned activities (line 23) or text match (lines 24 and 28).

```
1 [TestMethod]
2 public async Task InterestedInDialogProductContextPositiveAnswer()
3 {
4     var instrumentationService = new Mock<IInstrumentationService>().Object;
5     using (new FiberTestBase.ResolveMoqAssembly(instrumentationService))
6     // register all necessary components to Conversation.Container
7     using (var container = Build(Options.MockConnectorFactory | Options.ScopedQueue))
8     {
        // now create a root dialog, instrumentation service is just a mock, we do not
            want to write to AI
        MakeRoot = () => new RootDialog(Conversation.Container.Resolve<IDialogFactory>()
            ,
        Conversation.Container.Resolve<IConversationService>(), instrumentationService,
            Conversation.Container.Resolve<IQueryRecognitionService>());

        SetAuthenticatedUserWithEmail();

        toBot.Text = ConversationSection.InterestedIn.Description();
        var toUserActivities = await GetResponses(container, MakeRoot, toBot);

        toBot.Text = "il mrs NT 16 lt";
        toUserActivities = await GetResponses(container, MakeRoot, toBot);

        toBot.Text = "usb";
        toUserActivities = await GetResponses(container, MakeRoot, toBot);
        Assert.AreEqual(2, toUserActivities.Length);
        Assert.IsTrue( toUserActivities[1].Text.StartsWith(Resource.
            InterestedInDialog_AnythingElse.Substring(0,10)));

        toBot.Text = "yes";
        toUserActivities = await GetResponses(container, MakeRoot, toBot);
        Assert.IsTrue(toUserActivities[0].Text.StartsWith(Resource.
            InterestedInDialog_WhatDoYouWant.Substring(0, 20)));
    }
}
```

Furthermore, there are also user interface (UI) tests which also simulate the communication. However, these tests do not send the messages directly to the bot. They are instead an automation of real manual testing. Selenium[3] tests are used for this scenario. The chatbot is opened in a real web browser and the tests work with the HTML components. This way, actions such as clicking, typing and so on can be simulated. This is shown in code listing below where after a user question, solutions should be offered. The test then clicks on one of the solutions and checks, whether it refers to the correct URL.

```
[TestMethod]
[TestCategory("Chrome")]
public void UIInterestedInMoreQuestions()
{
    ConfigureAndStartDriver();

    InterestedIn();

    AgreeWithGDPR();

    seleniumRepository.WaitForElementWhichContainText(Resource.
        InterestedInDialog_WhatInformationLooking);

    SendTextToBot("what are the outputs?");

    seleniumRepository.WaitForElementWhichContainText(Resource.
        InterestedInDialog_FoundForYouSolutions);

    seleniumRepository.ClickOnDivWithId(convertTextToID("PGI"));

```

[3]https://www.seleniumhq.org/

```
    CheckUrlOpenedInNewTab("https://www.comap-control.com/solutions/technology/power-
        generation-industrial");
}
```

This robust testing system should provide a guarantee that the InteliBot works according to customers expectations.

6.2.1 Outcomes of testing

There were twelve sprints together which means approximately twelve releases. The feedback was processed after every release, therefore InteliBot always corresponded with customers requirements. When artificial intelligence is involved, the results are almost never perfect. InteliBot is no exception, some goals were not fulfilled simply because, despite the enormous effort, systems like LUIS, Azure Search, and so on, are limited to some point as well. Even the algorithm could be maybe improved. However, InteliBot, particularly "I'm interested in..." section provides reliable results in most cases.

"Excel test" concerning solutions is a good example. After the initial testing, roughly half of the test cases were working correctly. Thresholds were adjusted, LUIS model was trained on more utterances, changes were made in the algorithm, more synonyms were added. The results were slightly better, around 70% of the cases were now correct. However, the team could not come up with more ideas for the failed scenarios. It was then decided that further improvement is not worth the effort and resources, and the results are sufficient.

Chapter 7

Conclusion

The goal of this thesis was to get acquainted with current chatbot situation, techniques and technologies behind chatbot development and to create and implement a chatbot called InteliBot capable of information search for a particular company. First, a brief history of chatbots along with real chatbots examples were presented. Also, theoretical background lying beneath Microsoft chatbot development technologies was covered. This includes techniques such as NLP, vector semantics or Lucene search. Next, Microsoft Bot Framework as well as Microsoft Azure services were explained in order to have everything needed for InteliBot development. Then the algorithm used for the information retrieval together with the implementation of "I'm interested in..." section of InteliBot was described. The testing process of the chatbot that accompanied the development was also covered.

Overall, the goals of the thesis as well as the requirements of the customer were accomplished. The "I'm interested in..." section works correctly in most cases although there is still a room for improvements. These improvements include adjusting the algorithm by changing the text processing phase or Azure Search query creation process. Providing more data for LUIS model and synonyms or implementing different conversation logic may also contribute to better results.

Despite the sophisticated testing process of the bot, after it is put into service, there will very probably be further demands for the change of IntelliBot. This expectation is based on the fact, that the real customers of ComAp company have not tried InteliBot yet. However, the nature of the changes will be most probably the same as described above.

Bibliography

[1] *Basic Concepts*. [Online; Accessed 12 April 2019].
Retrieved from: http://www.lucenetutorial.com/basic-concepts.html

[2] *Try to build a lower dimensional embedding*. [Online; Accessed 8 April 2019].
Retrieved from: https://shanelynnwebsite-mid9n9g1q9y8tt.netdna-ssl.com/wp-content/uploads/2018/01/3-dimensional-word-embeddings-example.png

[3] *Fenomén jménem chatbot 1. část*. October 2018. [Online; Accessed 8 March 2019].
Retrieved from: https://www.xolution.sk/knowledge-sharing/knowledge-base/329/fenomn-jmnem-chatbot-1-st

[4] *Popular Chatbot Frameworks*. 2018. [Online; Accessed 4 April 2019].
Retrieved from: https://discover.bot/bot-talk/beginners-guide-bots/popular-chatbot-frameworks/

[5] Ahire, J. B.: *Introduction to Word Vectors*. [Online; Accessed 8 April 2019].
Retrieved from: https://medium.com/@jayeshbahire/introduction-to-word-vectors-ea1d4e4b84bf

[6] Amazon: *Conversational interfaces for your applications powered by the same deep learning technologies as Alexa*. [Online; Accessed 4 April 2019].
Retrieved from: https://aws.amazon.com/lex/

[7] Amazon: *What Is Amazon Lex?* [Online; Accessed 4 April 2019].
Retrieved from: https://docs.aws.amazon.com/lex/latest/dg/what-is.html

[8] Collobert, R.; Weston, J.; Bottou, L.; et al.: *Natural Language Processing (Almost) from Scratch*. 2011.
Retrieved from:
http://www.jmlr.org/papers/volume12/collobert11a/collobert11a.pdf

[9] Debecker, A.: *A Closer Look at Chatbot ALICE*. 2017. [Online; Accessed 5 April 2019].
Retrieved from: https://blog.ubisend.com/discover-chatbots/chatbot-alice

[10] Garbade, M. J.: *A Simple Introduction to Natural Language Processing*. [Online; Accessed 8 April 2019].
Retrieved from: https://becominghuman.ai/a-simple-introduction-to-natural-language-processing-ea66a1747b32

[11] Google: *A DIALOGFLOW ENTERPRISE EDITION*. [Online; Accessed 10 April 2019].
Retrieved from: https://cloud.google.com/dialogflow-enterprise/

[12] Google: *DialogFlow Analytics documentation.* [Online; Accessed 10 April 2019]. Retrieved from: https://dialogflow.com/docs/training-analytics/analytics

[13] Henderson, H.: *Artificial Intelligence: Mirrors for the Mind.* Milestones in Discovery and Invention. Facts On File, Incorporated. 2007. ISBN 9781604130591. Retrieved from: https://books.google.cz/books?id=vKmIiICDIwgC

[14] Henni, A.: *Speaktoit secures funding from Intel Capital.* 2012. [Online; Accessed 4 April 2019]. Retrieved from: http://www.ewdn.com/2012/05/31/speaktoit-secures-funding-from-intel-capital/

[15] Jurafsky, D.; Martin, J. H.: *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* 2017. third Edition draft. Retrieved from: https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf

[16] L. Mauldin, M.: CHATTERBOTS, TINYMUDS, and the Turing Test: Entering the Loebner Prize Competition. 01 1994.

[17] Lee, J.: *CHATBOTS WERE THE NEXT BIG THING: WHAT HAPPENED?* 2018. [Online; Accessed 5 April 2019]. Retrieved from: https://blog.growthbot.org/chatbots-were-the-next-big-thing-what-happened

[18] lola.com: *NLP vs. NLU: What's the Difference?* [Online; Accessed 10 April 2019]. Retrieved from: https://medium.com/@lola.com/nlp-vs-nlu-whats-the-difference-d91c06780992

[19] Mayo, J.: *Programming the Microsoft Bot Framework: A Multiplatform Approach to Building Chatbots.* Microsoft Press. 2017. ISBN 9781509304981.

[20] Microsoft: *How full text search works in Azure Search.* 2018. [Online; Accessed 7 April 2019]. Retrieved from: https://docs.microsoft.com/en-us/azure/search/search-lucene-query-architecture

[21] Microsoft: *Synonyms in Azure Search.* 2018. [Online; Accessed 15 April 2019]. Retrieved from: https://docs.microsoft.com/en-us/azure/search/search-synonyms

[22] Microsoft: *What is Language Understanding (LUIS)?* 2019. [Online; Accessed 14 April 2019]. Retrieved from: https://docs.microsoft.com/sk-sk/azure/cognitive-services/luis/what-is-luis

[23] Microsoft: *What is Text Analytics API?* 2019. [Online; Accessed 4 April 2019]. Retrieved from: https://docs.microsoft.com/en-us/azure/cognitive-services/text-analytics/overview

[24] Štefan Porubský: *Checkbot dokáže odhaľovať blbosti a dezinformácie na slovenskom internete.* 2018. [Online; Accessed 2 April 2019].

Retrieved from: https://techbox.dennikn.sk/checkbot-dokaze-odhalovat-blbosti-a-dezinformacie-na-slovenskom-internete/

[25] Salecha, M.: *Story of ELIZA, the first chatbot developed in 1966*. October 2016. [Online; Accessed 9 March 2019].
Retrieved from: https://www.analyticsindiamag.com/story-eliza-first-chatbot-developed-1966/

[26] Seif, G.: *An easy introduction to Natural Language Processing*. [Online; Accessed 8 April 2019].
Retrieved from: https://towardsdatascience.com/an-easy-introduction-to-natural-language-processing-b1e2801291c1

[27] Shevat, A.: *Designing Bots: Creating Conversational Experiences*. O'Reilly Media. 2017. ISBN 1491974826.
Retrieved from: https://www.amazon.com/Designing-Bots-Creating-Conversational-Experiences-ebook/dp/B0723B91XD

[28] Shewan, D.: *10 of the Most Innovative Chatbots on the Web*. 2019. [Online; Accessed 2 April 2019].
Retrieved from: https://www.wordstream.com/blog/ws/2017/10/04/chatbots

[29] SOTOMAYOR, L.: *Success Story: U-Report Liberia exposes Sex 4 Grades in school*. 2016. [Online; Accessed 4 April 2019].
Retrieved from: https://ureport.in/story/194/

[30] Tedeschi, B.: *2011's Top 10 Apps for Android Phones*. 2011. [Online; Accessed 1 April 2019].
Retrieved from: https://www.nytimes.com/2011/12/29/technology/personaltech/in-2011-app-developers-turned-attention-to-android.html

[31] Turing, A. M.: I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*. vol. LIX, no. 236. 10 1950: pp. 433–460. ISSN 0026-4423. doi:10.1093/mind/LIX.236.433. http://oup.prod.sis.lan/mind/article-pdf/LIX/236/433/9866119/433.pdf.
Retrieved from: https://doi.org/10.1093/mind/LIX.236.433

[32] Weizenbaum, J.: ELIZA - A computer Program For the Study of Natural Language Communication Between Man and Machine. Communications of the ACM 9(1), 36-45. *Commun. ACM*. vol. 9. 01 1966: pp. 36–45. doi:10.1145/365153.365168.

[33] Weizenbaum, J.: *Computer Power and Human Reason: From Judgement to Calculation*. W.H. Freeman and Company. 1976. ISBN 0716704633.

[34] Wierema, S.: *Build your own Siri: Api.ai offers voice integration for all*. 2014. [Online; Accessed 4 April 2019].
Retrieved from: https://thenextweb.com/dd/2014/09/16/build-your-own-siri-api-ai-offers-voice-integration-for-all/

Appendix A

Contents of Attached CD

- `text/` – containing LaTeX sources and files used
- `bot_source/` – containing source files of the chatbot
- `xduris04_thesis.pdf` – thesis document
- `xduris04_video.mp4` – video