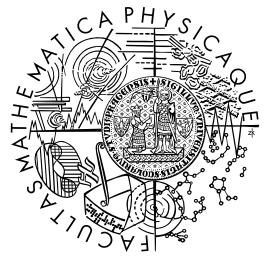


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Patrícia Schmidtová

A chatbot for the banking domain

Institute of Formal and Applied Linguistics

Supervisor of the bachelor thesis:	Mgr. et Mgr. Ondřej Dušek, Ph.D.
Study programme:	Computer Science
Study branch:	IOI

Prague 2019

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In Prague 19.07.2019 signature of the author

Title: A chatbot for the banking domain

Author: Patrícia Schmidtová

Institute: Institute of Formal and Applied Linguistics

Supervisor: Mgr. et Mgr. Ondřej Dušek, Ph.D., Institute of Formal and Applied Linguistics

Abstract: This thesis designs, implements and evaluates a task-based chatbot, which is expected to answer questions and give advice from the banking domain. We present an extendable natural language understanding (NLU) module based on GATE Framework which serves to create interpretations of the user's utterance. We implement a rule-based dialog manager component which is responsible for answering based on the NLU's interpretations and a stored context. We also implement a template-based natural language generation module. We then evaluate the chatbot with human testers, verifying it performs well in most cases and identifying areas for future improvement.

Abstract in Czech: Tato práce navrhuje, implementuje a vyhodnocuje úkolově zaměřeného chatbota, jehož úkolem je odpovídat na dotazy a poskytovat rady ohledně bankovnictví. Uvádíme rozšiřitelný modul porozumění přirozeného jazyka (NLU), založený na technologii GATE Framework, který slouží k vytvoření interpretací uživatelova dotazu. Pak implementujeme pravidlový dialogový manažer, který je odpovědný za odpovídání na základě interpretací od NLU a uloženého kontextu. Dále implementujeme model generování přirozeného jazyka, založený na šablonách. Chatbota hodnotíme za pomoci lidských testerů, ověřujeme, že ve většině případů je schopen plnit své úlohy, a identifikujeme možná budoucí vylepšení.

Keywords: dialog system, natural language understanding, NLP, chatbot, dialog manager

Keywords in Czech: dialogový systém, porozumění přirozeného jazyka, NLP, chatbot, dialogový manažer

I would like to thank my thesis supervisor, Mgr. et Mgr. Ondřej Dušek, Ph.D. and this thesis consultant Mgr. Vojtěch Hudeček for all their help and guidance, quick responses to my emails and being an endless source of motivation.

I would also like to thank my family, boyfriend and friends for their continuous support during my Bachelor's studies.

Last but not the least, I would like to thank my former colleagues from RobotReader and Datlowe for supporting me while working on this thesis. Especially Lucka, because if it wasn't for the conversation with her three years ago, this thesis would very likely be on a different topic.

Contents

1. Introduction

In the last decade, chatbots have become more popular and available than ever before. Thanks to the current technology, we can carry them in pockets, and they can wait for us at home. They have evolved into virtual assistants, such as Apple Siri, Amazon Alexa or Google Home, which are available 24/7 and never have to take a day off.

1.1 Dialog Systems in General

Three main components are in some way present in almost all dialog systems: natural language understanding (NLU), dialog manager (DM), and natural language generation (NLG). Natural language understanding is responsible for decoding the user's message and transforming it into a format that can be further processed by the DM. The dialog manager keeps track of the preceding context (dialog state) and uses it to find an answer to the NLU's representation of the user's message. Before we send the answer to the user, the natural language generation has to translate it into natural language. Spoken (voice-based) dialog systems also contain automated speech recognition (ASR) and text-to-speech synthesis (TTS). Automated speech recognition is responsible for transcribing the user's spoken utterance into written text, which is then passed to the NLU. Text-to-speech synthesis takes the answer from the NLG and generates its audio representation. This thesis deals with written chatbots, so we do not need ASR and TTS.

Dialog systems can be conversation-oriented and/or task-oriented. The former are used almost exclusively for amusement, while the latter usually perform a given task. The first chatbot, Eliza (Weizenbaum, 1966) was a conversation-oriented dialog system designed to behave like a Rogerian psychologist. The most recent advances in the field of conversational chatbots have centered around the Amazon Alexa Prize (Ram et al., 2018). In contrast, most of the virtual assistants, such as Siri, are task-oriented and help users by finding a nearby restaurant or by looking up the weather forecast. This thesis presents a fully task-oriented chatbot.

We can also divide chatbots based on their implementation. According to Jurafsky and Martin (2019), we can divide chatbots into rule-based and corpus-based. Rule-based chatbots usually have a dictionary and a set of rules that deterministically models their behavior. On the other hand, corpus-based chatbots use statistical methods in order to provide answers,

such as information retrieval and sequence to sequence neural networks. Such architectures are further described and compared in Gao et al. (2018).

1.2 Thesis Goals

This thesis aims to implement and evaluate a task-oriented chatbot in the banking domain. The development started as a project in the RobotReader company. The expectations for the chatbot were defined by an unnamed client of the company, who was in the process of starting a bank. Since the client had almost no data, it was more reasonable to make the chatbot rule-based. The client's vision was a friendly chatbot who would be able to show people that investments are not something to be scared of. Unfortunately, this chatbot was not deployed because the client's project was cancelled after a few months. We finished the implementation for the purposes of this thesis.

We set the following goals:

1. We will design and implement a rule-based natural language understanding module. The module should be easy to extend to another task, language, or domain.
2. We will design and implement a rule-based dialog manager, which will be able to respond using natural language using a built-in template NLG component.
3. We will evaluate this chatbot based on how well it performs the tasks it was designed for. Furthermore, we aim to learn from the mistakes discovered while testing and suggest further improvements.

1.3 Structure of This Thesis

In Chapter 2 we introduce the tasks to be solved by the chatbot. We further analyze each task and divide them into smaller tasks for the NLU module and the dialog manager.

In Chapter 3 we focus on the natural language understanding module and the representation of domain information. We introduce the used technologies, methods, and the architecture of the module.

In the first part of Chapter 4, we describe the dialog manager component of the chatbot. In the second part, we discuss the natural language generation component.

In Chapter 5 we introduce the experiments and provide their results. In the second part, we analyze the errors and suggest improvements for future work.

We provide a brief installation guide in Appendix B.

The complete JavaDoc and KDoc documentations of this thesis source code are included with the source code. Due to their length and layout, they are not suitable to be included in the printed version of the thesis.

2. Task Description

In this chapter, we will introduce the tasks to be solved by the chatbot. We will define them, suggest the motivation for them, and further divide the tasks into the goals of the NLU module and the Dialog Manager. We will also provide examples to deepen the understanding of the tasks. The tasks have been defined by the client of RobotReader and their solutions were implemented by the author of this thesis. All of the tasks are situated in the banking domain, and related to investments.

2.1 Frequently Asked Questions

The motivation for the first task is to create a more user-friendly alternative to lengthy and opaque pages full of frequently asked questions. The client provided a list of over a hundred questions about real estate fund investments, paired with answers. The chatbot is expected to analyze the user's message and either return the correct answer or apologize for not having the answer.

In this case, the output of the NLU module is a list of reasonable interpretations of the user's question. Since the frequently asked questions do not follow a specific pattern, the interpretations have to be as universal as possible. In order to keep the representation flexible, it cannot have a prepared rigid structure, so we need to put as much information about the words in the sentence into the representation itself. On the other hand, we do not want to clutter the representation with unnecessary words, so we also need to distinguish which words in the sentence contribute to its meaning. Therefore, the task of the NLU module is to identify as many important words as possible and then combine them into the final representation.

The Dialog Manager is expected to match the representation it gets from the NLU module with a question that is already in the database[1]. In order to do this, the Dialog Manager's task is to compare the input question with all of the questions in the database and compute a similarity measure we will use to find the best match. In case there are multiple matches, we will select the few best ones to make sure the user's question will be answered. In case none of the questions in the database is similar enough, the Dialog Manager admits its failure and apologizes.

[1]The questions in the database went through the same NLU processing in order to produce the final representation.

Now let us further demonstrate this task on an example:

(2.1) *Co všechno obsahuje cena uvedená u nemovitosti ?*
What all includes price displayed next to the property ?
'What does the price displayed next to the property include?'

The meaning of this sentence is contained in the following four words: the interrogative pronoun what, the verb to include, and the terms price and property. The remaining words do not appear in the output representation of this message because they are not essential. Similarly, if the user decides to be more eloquent and use auxiliary verbs or more padding words, they will not be included, which simplifies the Dialog Manager's matching task.

2.2 Questions About Investment Products

The goal of this task is to provide information about various investment products. In most cases, the user is interested in the properties of a given product or seeks a product that fulfills given criteria. In order to succeed in this task, the NLU module needs to identify all mentions of products, attributes, and restricting criteria. It is also necessary to classify the type of question asked. We divided the questions into the following groups based on their nature[2]:

- **What is it?**: The user wants to know the definition of a product or an attribute. It is the easiest type of questions to identify because there are not many variations of it. As for the extraction part, we only need to know the term to be defined.

  The Dialog Manager's task is also simple; it needs to query the database for the definition of a given term. In case the term is not found in the database, the chatbot should specify it understood the essence of the question but did not recognize the term.

- **What is the value of an attribute of a product?**: The user inquires the value of an attribute of a given product. The product and attribute in question must be extracted. Reliably identifying this type of queries is more difficult than the previous type. There are more ways to ask this question, but their syntax can be completely different, for example:

[2]The types listed here were designed and implemented by the author of this thesis. There are more types in the final product.

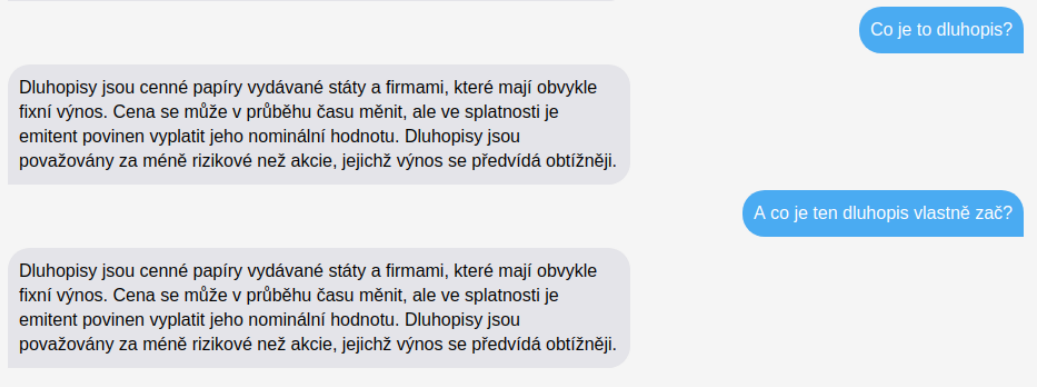


Figure 2.1: Two different variations of the "What is it" type question. First translates to "What are bonds?", the second roughly translates to "And what are those bonds anyway?". They both correctly receive the same answer.

(2.2) What is the *attribute* of *product*? – What is the liquidity of bonds?

(2.3) How *attributeAdverb* is *product*? – How liquid are bonds?

The Dialog Manager tries to find the answer in the database. There are three potential outcomes: it can find a prepared answer, a value to pass to a template, or it can find no answer. In the first two cases, the Manager returns the answer, and in the last case, it apologizes.

- **What is the value of an attribute?**: Similarly to the previous type of questions, the user expects to find out the value of an attribute of a specific product. However, the product is not explicitly mentioned - it is either implied or replaced by the pronoun *"it"*.

  We used basic coreference resolution (Clark et al., 2010, p. 611ff.) in order to answer messages of this type. The Dialog Manager remembers the products mentioned by the user, so it takes the last mentioned product and carries out its task as if it was a query of the previous type.

- **Compare two products**: In case the user cannot decide between two products, it might be helpful to provide a comparison between them. While there is no way of telling which product is better because different people have different preferences, it is possible to compare them based

on an attribute. This type of query will surely have an adjective in comparative and therefore, is easily distinguished.

The Dialog Manager gets the value of the attribute for both products and passes them to a template. The intended reply, however, is not the name of the product with a higher value. The should contain the values of the given attributes for both products which still implicitly answers the user's question. If the attribute is not supplied, the products are compared based on all attributes.

- **Tell me everything you know about ...**: The user might opt for a simpler question and asks about the product in general. The questions of this type might look like: What can you tell me about *product*?

  So far, the Dialog Manager's task is to list the known values of all of the attributes in separate messages. Future work might include priority rankings of attributes, so just the few essential ones would be selected.

2.3 Individualized Advice

The last task has been created as a proof of concept to show that this chatbot could be used to collect information from the user. The information could then be used to make a recommendation or a calculation. The demonstration task we chose was to build an *investment profile* of a person and subsequently give them a recommendation for the best investment product for them.

In this task, the Dialog Manager takes the initiative and asks the user questions in order to provide investment advice tailored to the user's needs and expectations. It is different from the previously mentioned tasks because the user does not have to write out whole sentences, but rather the minimal utterance that answers the question posed by the Dialog Manager. All of the extracted information can be either provided at once or message by message.

The first goal of the NLU module is to inform the Dialog Manager that the user is seeking personalized advice, such as the example below, and it should start asking questions. Then it has to relay every piece of information that could be valuable to the Dialog Manager.

(2.4) *Do čeho bys mi doporučil investovat?* – What would you recommend for investment?

The Dialog Manager has to identify what kind of information it needs and ask for it. It should keep asking questions until it has all the information. However, we anticipated that some users might ask a question before

answering, and in that case, answers that question first. Then it reminds the user to answer its previously asked question. Once it has all the information, it should create a recommendation based on a set of decision rules.

2.4 Conversation-specific Phrases

Apart from the domain-specific questions, the chatbot also recognizes greetings and polite phrases. Future work might include recognizing more out-of-scope questions and requests that are in no way essential to the chatbot's primary goal. That is because many testers were disappointed that the chatbot could not provide a weather forecast or tell a joke. This shows that small talk is important for some users' overall impression.

3. Natural Language Understanding

In this chapter, we will describe the natural language understanding component of the chatbot. First, we will introduce the frameworks, tools, and technologies used (Section 3.1). Second, we will discuss the methods and approaches used throughout the whole NLU Module (Section 3.2). Finally, we will take a look at its inner structure of and implementation (Section 3.3).

The primary goal of the Natural Language Understanding module is to decode the user's message into a format that can be further processed by the Dialog Manager, which is a separate module. In this thesis, we will refer to the meaning of the user's message as *intent*. Additionally, we set a secondary goal for practical purposes: to make the module easy to extend to another task within the domain, a distinct domain, or a different language. The NLU is designed to provide a variety of possible interpretations in order to maximize the chance that the Dialog Manager will understand the dialog act (McTear et al., 2016, p. 164ff.) successfully. The NLU module tries to find all the possibilities, identify all the fragments, and finally ranks and orders them by their likelihood. This enables us to keep the task-dependent dialog context separate from the NLU module.

We will show an example first and use it throughout the entire chapter to demonstrate various aspects of the NLU:

(3.1) *Co je rizikovější, akcie nebo dluhopisy ?*
What is riskier, stock or bonds ?
'Which one is riskier, stock or bonds?'

The best interpretation of this question is the *product comparison* type query presented in Section 2.2, where the comparison criterion is *risk* and the products to be compared are *stock* and *bonds*.

3.1 Used Frameworks, Tools & Technologies

The main building block of the NLU module is *GATE Framework 8.4* (Cunningham et al., 2013), which is an open source software for text analysis and natural language processing. It allows us to construct a complicated but

well-structured pipeline by neatly organizing resources. The analyzed text is stored in a structure called a *GATE Document* and will be referred to as *Document* in this thesis. The *Document* has its content and *Annotations* stored in *Annotation Sets*. The sole purpose of *AnnotationSets* is to group *Annotations*. For example, the "address" *AnnotationSet* stores *Annotations* that help us identify addresses, such as streets or towns. Every *Annotation* has a start and end offset, a type, and a *FeatureMap*. In this thesis, *Annotations* serve as an abstraction of concepts represented by words or phrases and allow us to write rules in a bottom up approach. Moreover, *GATE* comes with a variety of built-in tools, such as gazetteer and finite state automaton implementations. Those allow us to create and manipulate *Annotations* based on word lookups and rules.

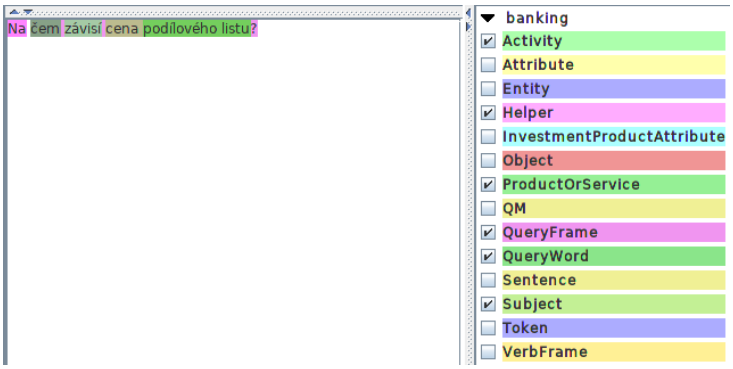


Figure 3.1: A preview of the GATE Developer. 'banking' is the name of the *Annotation Set*, the highlighted keywords below are *Annotation* types.

In addition to working with *GATE*, RobotReader provided a library called *lama-framework*[1] which serves as an extra layer and allows to access *GATE API* in the *Kotlin* language. Its primary function is to make the development and deployment of the NLU module smoother and more pleasant.

Since Czech is a fusional language with flexible word order, it would be a tedious task to extract information without any morphological and syntactic analysis. For this purpose, this thesis heavily relies on tools developed at the Institute of Formal and Applied Linguistics, namely *MorphoDiTa* for tokenization, part-of-speech tagging and lemmatization (Straková et al., 2014), *UDPipe* for dependency parsing (Straka and Straková, 2017), and *NameTag* for named entity tagging (Straková et al., 2014). Using them in the GATE environment has been possible thanks to the *czsem* package (Dědek, 2012).

[1] "LAMA" Stands for Linguistic Analysis and MAchine learning.

3.2 Methods and Approaches

In this section, we will cover some general principles, methods, and approaches that were used throughout the entire NLU Module implementation. The central idea is that the understanding of the user's utterance was built using a bottom-up approach; first, words or phrases that match specific rules get labeled with Annotations. In the next step of processing, those words and phrases are generalized - further represented only by their Annotation type and features. We form a solid foundation that can be expanded by merging Annotations into higher-level ones.[2]

In the first step, the utterance is tokenized, and the tokens are morphologically and syntactically analyzed. In the context of GATE, this means that Annotations of type Token[3] covered each token and the results of the morphological analysis (such as the part-of-speech tag or the lemma) were stored in the Token's FeatureMap.

The first of the methods we used is an in-domain entity gazetteer lookup. Solely looking up keywords or key phrases in the text would be ineffective, as Czech words usually appear in an inflected form. That is why the majority of gazetteers we use have been lemmatized first, and the entries are matched against Token lemmas. In this thesis, we use gazetteers to find banking-related terms from the domain ontology, and entities that will be further discussed in Section 3.3.

The majority of the understanding of the banking domain stems from the domain ontology provided by RobotReader. The ontology itself is not included in this thesis because it remains the property of the company. However, its role in the NLU Module is essential, and therefore, we will discuss it in this section. In Figure 3.2, we can see a preview of the ontology in the *Protégé* ontology editing tool (Musen, 2015).

The ontology divides instances into the following main 7 classes:

- The **verb** class mostly contains verbs related to banking or investment, such as *to invest* or *to buy*. In addition to those, the class also contains auxiliary verbs.
- **Attributes**, such as *risk*, represent the properties of products.
- The **products** are listed in the products or services class. Those include *bonds* and *stock* from Example 3.1.

[2]For example First name + Last name = Full name

[3]Token with a capital T will denote the Annotation

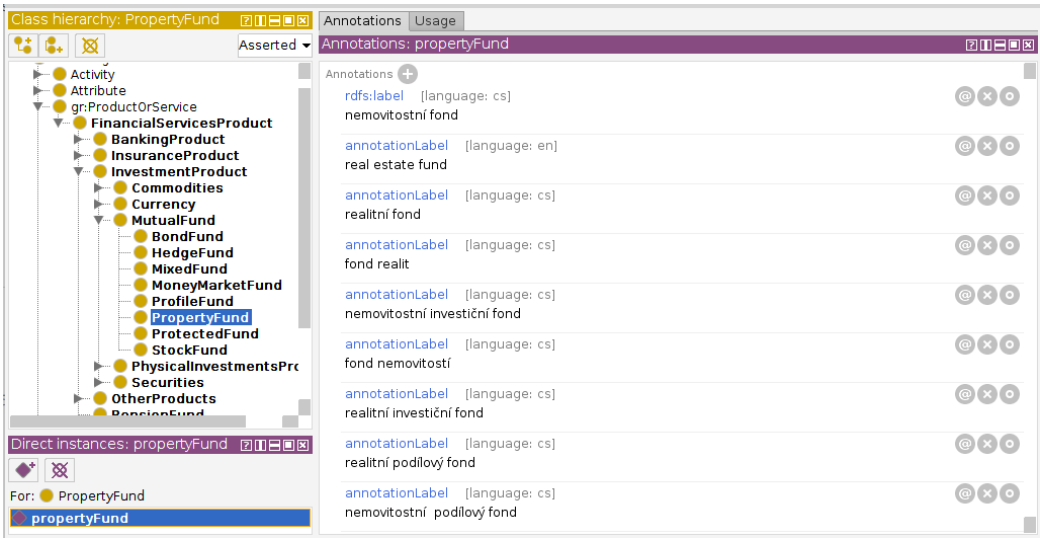


Figure 3.2: Preview of the RobotReader banking ontology

- **Objects** include other banking terms that could neither be classified as attributes nor as products, such as *price* or *contract*.
- **Modifiers** include quantifiers, derivations of words in the previously mentioned classes, and other adjectives that could further modify the meaning of a sentence.
- The instances in the **organizations** class are types of institutions which mediate the investments or state bureaus that regulate them, for example, *cadastre*.
- Finally, the **other helpful keywords** include query words and prepositions that are essential to the meaning of the sentence.

We also used regular expressions as they are a simple but powerful tool. They were used in their standard form, as well as in a GATE-enhanced form called *JAPE*. What makes *JAPE* unique is that it allows us to define sequences of Annotations. It is essential for building understanding from the bottom up. Typically when using *JAPE*, we have already previously created a collection of annotations, whether they are keywords, named entities, or possible values to be extracted. For example, a *JAPE* rule might consist of a sequence *Number* and *Period*, such as 5 years. The *JAPE* then suggests

that the two *Annotations* form a bigger *Annotation* which is again a *Period*, but this time has the duration of 5 years and not just one.

Keeping in mind that Czech has a flexible word order, we realize it would take an immense number of combinations in order to capture all possibilities. Therefore, we needed a stronger tool and decided to utilize syntactic dependencies with a tool called *GrExt*[4]. In Figure 3.3 we can see that changing the word order does not change the dependency tree of the given sentence. Let us recall that the dependencies have been created by UDPipe during the very first step of the processing. Similarly to the previous case, we also worked with the Annotations to provide some generalization. This helps us create broader rules because we do not need to specify the lemmas or tags of the words. Instead, we say that a given node in the syntactic tree should be a *Product*.

Figure 3.3: Three syntactically equivalent sentences with a different word order

However, the rules that were based on dependencies still proved to be quite rigid and required several rules to express a single type of query, for example due to word derivations. A new tool developed by RobotReader was explicitly designed to target this issue. This leads us to the last tool we used, the *Frames*. Each *Frame* models a query or a part of it in two ways: slots and rules. The slots represent essential parts of the query, and the rules check the integrity of data filling the slots and the relations among slots. The used relations can be dependencies or relative position in the sentence. Figure 3.4 gives an example on how slots are defined. The slots are defined in a JSON file, such as in Figure 3.4. Please note that there can be more types of slots, which is particularly helpful for tackling derivation. In the context of Example 3.1 this means that the attribute can either be the noun *risk* from the *Attribute* class, or the adjective *risky* from the *Modifier* class.

The rules are a set of boolean conditions and are a part of the source code. The following simplified rule corresponds to the slots in Figure 3.4:

[4]Developed by Datlowe, short for Graph Extractor.

```
{
  "name": "http://schema.rrdr.org/banking/multilang/frame_template/queryWhatFrame#",
  "$schema": "http://schema.rrdr.org/#",
  "description": "definition of a frame for query of type what",
  "type": "QueryWhat",
  "threshold": 1.0,
  "slotDefinitions": [
    {
      "types": ["QueryWord"],
      "name": "what"
    },
    {
      "types": ["Activity"],
      "name": "activity"
    },
    {
      "types": ["Entity"],
      "name": "entity"
    }
  ]
}
```


Figure 3.4: JSON frame definition for questions of type 'what is it?'.

(3.2) equals(instance(what), 'what') AND equals(instance(activity), 'be') AND exists(entity)

This rule essentially means that the query word needs to have the meaning 'what', the verb needs to mean 'to be' and the entity in question must be present in the query. This *Frame* has one rule, however, other *Frames* can have more. Each rule has a weight, and if met, the weight is added to a measure called *optionValue*. A higher *optionValue* means a better match. In order to create an *Annotation*, the *optionValue* has to exceed a given minimal threshold. For example, if the rule above does not apply to the user's message, then the output will not contain a 'what is it?' type question.

Even with *Frames*, we still build *Annotations* from the bottom up and that is why we first use *Frames* in order to group concepts into bigger annotations, such as grouping a verb with its subject and object. In Example 3.1 we first use *Frames* to group *'stock or bonds'* into an *Annotation* which represents the logical disjunction of products. The final *Frame* which marks the question as product comparison then takes the contents of this *Annotation* into a designed slot.

The tools mentioned in this section were either partially (*Frames*) or fully (the rest) defined in GATE resource files, since *GATE Developer* works by interpreting resource files and applying them to the Document.

3.3 Inner Structure of the NLU Module

As mentioned before, the primary goal of the NLU Module is to provide all possible interpretations of the user's utterance. The secondary goal is to maximize portability to another language, another task, or a different domain. In order to achieve both goals, the module consists of three smaller ones: a domain-independent module that mostly processes entities (Section 3.3.1), a chitchat layer that handles small talk with the user (Section 3.3.1, and a layer of domain information that extracts the final concepts to be further analyzed in the dialog manager (Section 3.3.3.

3.3.1 Common Entity Processing

In this section, we will cover the domain-independent part referred to as *common*. It consists of 4 smaller separate modules: numbers, dates, addresses, and names. In all of the modules, recall was favored over precision, because the extracted Annotations still had to fit into the rules of the banking module in order to be used. The used technologies were only gazetteers, regular expressions, and JAPE (see Section 3.2).

As this portion of the NLU module is domain-independent, it produces far more detailed output than the chatbot actually needs. However, this makes it ready to use as a base for other domains.

The goal of the *numbers* module is to find and interpret all numbers, including those that are written out as words, Roman numerals, and ordinal numbers. Interpretation means that in case the number in a digit format contains a period or a comma, the module needs to decide whether it is a digit separator or a decimal point, based on the number format. This module is also responsible for recognizing various units, such as currencies so that it can adequately mark numbers as amounts. As preparation for different tasks and domains, the module also annotates birth IDs, ID numbers of companies, credit card numbers, phone numbers, and dates in purely numeric format. All of those are not only annotated, but also extracted into a structural form for future use.

The *date* module expands on the information retrieved by the numbers module. It also annotates dates where the days and months are written out as words. However, the annotation of time periods and frequencies is this module's most significant contribution. That is because the investment advice depends on how long the user is willing to store their money in an investment product. The frequency of additional regular investments is also an essen-

tial factor. Apart from the previously mentioned Annotations, this module can also extract information about time intervals which could be particularly useful in case a client wanted to see information about the development on the stock market in a given time frame.

The *address* module recognizes and segments addresses, so there is no need for the user to fill out a form. This module combines the keywords from a gazetteer with named entities tagged by NameTag(Straková et al., 2014) in order to produce a reliable result. This module is redundant for the given tasks. Nonetheless, its intended purpose was for a task where the chatbot has the competence to gather information in order to generate and suggest a framework contract[5] for the user. Additionally, as it is one of the chatbot's task to inform about property investments, addresses seemed like a reasonable entity to extract.

The *names* module annotates personal names and titles. This information could be used in order to create a framework contract. The extracted data could also be stored in the dialog manager (as an extension of this work) so that the chatbot could address the user by name in order to create a more pleasant experience.

3.3.2 Chitchat Layer

The *Chitchat* layer serves to identify the user's utterances if they are not banking-related but rather greetings or polite phrases. This layer is simple so far because it has not been considered a priority. So far, the gazetteer lookup in this layer contains three types of concepts: greetings, requests, and signals. Greetings include basic welcome greetings, farewells and also friendly questions, such as *How are you?*. Requests contain keywords such as *Repeat it*, or requests to change the language of the conversation. Phrases such as *thank you* and *please*, *yes* and *no* are included in the Signal group.

Apart from the keyword lookup, this layer also includes dependency rules. Those serve to identify questions about the chatbot's abilities, such as *What can you do?*

Future work can expand on the keywords and also on the rules, in order to give the chatbot a friendlier personality and more everyday topics to discuss.

[5] A framework contract can be described as a rough draft for the final contract.

3.3.3 Banking Domain Layer

The understanding of the banking domain stems from the RobotReader banking ontology. The data from the ontology has been processed into gazetteers which are used to label all the words associated with a class in the ontology with an Annotation that represents a specific concept.

Feature Transfer After we have the entities from the domain-independent module, and all of the banking and task-related words and phrases are appropriately annotated, the *Annotations* are post-processed, which means that various important features, such as selected parts of the POS tag, are copied from the base Tokens to the *Annotations* that cover them. For example, in verbs, this means that the ontology-created annotation now carries information about the number, person, tense, negation, or a flag that the verb was found in the infinitive.

Syntactic preprocessing Before we apply dependency rules, we need to make sure that we can reasonably extract specific nodes in the dependency tree. The problem is that the Tokens that form one concept, for example, an amount of money, often appear far from each other in the dependency tree. To solve this problem, we create a new set of Tokens, where the concept that should not be separated into several concepts is glued together in one Token. Then we rerun syntactic analysis and use the resulting trees in the rules to come.

Syntactic rules Now that we have what we need, we can use the grouping rules mentioned in Section 3.2. All of the graph extraction rules have confidence scores, which denote how confident we are that the Annotations created by this rule are true positives. On the other hand, Annotations created by *Frames* use the *optionValue* (see Section 3.2 computed on-the-fly based on matched data; they are not given by the specific rule used but rather by the sum of importance of applicable rules. There is also an additional measure of the quality of the match called *coverage*. It represents the portion of the sentence that is covered by a given Annotation. Having a second measure proved to be essential because there were cases when a rule matched a portion of the sentence very well but ignored the rest, which changed the meaning of the sentence.

First, we use dependency rules, some to just group phrases and some to find and annotate whole queries. However, after the first few rules, it

became clear that while these rules are accurate, they are complicated even if they are supposed to model simple sentences, as illustrated in Figure 3.5. It would be manageable to write the dependency rules for the questions about investment products from Section 2.2, which all have a similar structure, but writing them for the frequently asked questions (Section 2.1) would be very impractical.

Figure 3.5: A type of query with the corresponding dependency rule.

In order to represent the frequently asked questions, we used a structure that resembles a set of vertices of a tectogrammatical tree (Sgall et al., 1986; Žabokrtský et al., 2008; Bejček et al., 2012). We chose this representation because it captures the semantics of the sentence well for our purposes while removing redundancy. Every node in the t-tree, which can be either a word or a phrase, is represented by the corresponding instance in the RobotReader banking ontology. Thanks to this approach, if two sentences differ in words which are synonyms, their output representations are identical. However, in most cases, auxiliary verbs do not make a difference in meaning, as illustrated in Example 3.3.3. Therefore, in order to recognize a question, it should not matter whether an auxiliary verb was used or not. To address this issue in the implementation, we replace it with the verb that was dependent on it. Since that verb was in the infinitive, we copy the information about the number, person, tense, and negation from the auxiliary verb. After that, we can safely delete the node representing the auxiliary verb.

(3.3) *Na čem závisí cena podílových listů?* – What influences the price of shares?

(3.4) *Na čem může záviset cena podílových listů?* – What may influence the price of shares?

The other types of queries are represented by *Annotations*. Their type is the name of the query type and they contain all extracted information in their *FeatureMap*. Each type of query discussed in Chapter 2 has one or more *Frames* corresponding to it and for that reason, there are 29 *Frames* at

the moment. *Annotations* created by *Frames* account for the vast majority of the output. Before they are passed over to the Dialog Manager, they are first scored and ordered based on the score. The score is computed as *optionValue* times coverage times a coefficient. The coefficient depends on the type of Annotation, and its role is to balance out the differences between the maximum possible values of *optionValues*. When the Annotations are in the right order, they are transformed into a JSON format per the company's conventions. The final JSON includes all types of entities that could potentially be the answer to the chatbot's question, the sorted list of JSON representations and the document content. All entities, products and attributes are referenced by their URI in the RobotReader ontology.

4. Dialog Manager

In this chapter, we will focus on the Dialog Manager component. First, we will explain how intents coming from the Natural Language Understanding Module are evaluated (Section 4.1). We will also discuss how answers are generated (Section 4.2). The Dialog Manager is rule-based, so in order to explain how it works, we will cover the critical decision points.

Before the Dialog Manager can start answering the user's questions, it needs to know the answers. So the very first step is loading the answers from the database into a designed class.

In order to characterize whether an answer was found, we use five basic *Result Types*:

- **Success**: An answer was successfully found, and the Dialog Manager will send it to the user
- **Failure**: The Dialog Manager has not found an answer, and does not know what the user's utterance means.
- **Verify**: The Dialog Manager successfully recognized the question (and the question makes sense) but did not find any answer in the database.
- **Next Slot**: This type is used when the Dialog Manager is asking the user for information and implies that it is ready to fill the next slot by asking a question. This is also considered a success.
- **Challenge**: The Dialog Manager asked a question, but the user said something that can not be used as an answer.

The first decision point is right at the beginning of the processing when the Dialog Manager receives its input from the NLU Module. It may receive the answer to a question it previously asked, a list of possible intent representations, or in case no intents were recognized, at least information about the keywords that appeared in the utterance.

4.1 Processing a List of Possible Intents

As mentioned at the end of Chapter 3, the possible intents[1] that are passed to the Dialog Manager are ranked and sorted by their score. The Dialog

[1] Referred to as *concepts* in the implementation

Manager can then proceed to go through the representations until it succeeds in finding an answer or until it runs out of intents.

The processing of the intents is based upon their type:

- **Frequently Asked Questions** are matched against all of the questions in the database. First, the content and lemmatized content are compared, and in case one of them matches, the questions have a match score of 1. If not, the sets of annotations and words are compared and produce a score. If the score is higher than a threshold, the answer is stored. When the question has been compared to all of the questions in the database, the matches are sorted, and only the best answers are sent to the user.

- **Questions About Investment Products** Recall that the NLU Module has already extracted the URIs[2] of the essential parts of the question, such as the products and attributes mentioned. The only thing the Dialog Manager has to do is to look into the database and either find a pre-written answer, or a value of the attribute in question that will be filled into the corresponding template (see Section 4.2.

- **Chitchat** The NLU Module already marks what kind of phrase it is, so the only thing left for the Dialog Manager to do is to pick a random answer from the possible set of answers and send it to the user.

- **Individualized Advice** We use frame-based dialog management for this task (Bobrow et al., 1977; Jurafsky and Martin, 2019, p. 427f). If the Dialog Manager receives a query of type *individualized* or *recommend*, it changes the state and starts up a *Frame*. Similarly to the *Frames* in the NLU Module, it has slots of a certain type. The user might provide some information in the same utterance, so all values that could be used to fill slots are moved to the *Frame*. If the *Frame* is not complete, it asks the user for additional information and changes the state in order to anticipate the answer. In case the user provided the answer, it will be used to fill the slot, and the Dialog Manager will keep asking until it fills all required spots. In case the Dialog Manager expects an answer but gets a question instead, it answers it. However, if no questions were recognized, the Dialog Manager will kindly remind the user to provide the answer.

[2]The database uses the same URIs as the ontology.

As the Frequently Asked Questions (see Section 2.1) look similar to the Questions About Investment Products (see Section 2.2), the NLU Module often provides representations of both types. In those cases, the Dialog Manager has to iterate through a couple of intents in order to reach the right one. In some cases, especially if the question is out of scope, the right representation of the intent might not even be present at all. In order to find the best possible option, the Dialog Manager returns the first successful answer. However, in case that none of the representations lead to a successful answer, the Dialog Manager remembers the first answer with the type *Verify* and returns it if no other option works out. Thanks to this approach, we can show that the Dialog Manager understood the type of the question by transcribing it to the user in the natural language generation module.

4.2 Answer Generation

In this section, we will discuss the ways how the Dialog Manager generates the replies it sends to the user. First, we will need to introduce the structure that the Dialog Manager uses to describe answers. Then we will go over the three main approaches we used to generate answers.

The *Dialog Answer* has a type, a textual template, and a map of values. The map is important to the answer generation because that is where the Dialog Manager stores the parameters for the templates. The text of the Dialog Answer is what is sent and shown to the user. In case the type of answer was *Verify*, the Dialog Manager will first transcribe its understanding of the question type, which is stored in a template, and then give the answer.

The easiest case is when the response is pre-written, saved in the database and there are no values to be filled-in. It makes sense for the frequently asked questions because each question is different and requires a different answer. However, writing such responses for every type of queries about products would be time-consuming and would not make sense. In some cases, such as the conversation-related utterances, there are several possible answers, and the Dialog Manager picks one randomly.

For most of the queries about investment products, the answers are generated using values from the database and templates. When the Dialog Manager finds the answer, it stores the values and the type of the query (and hence, the template to use) in the map. In the last stage, we iterate through the map with values, and the answers are sent to the user. We attempted to write the templates in a way that would not require the inflection of values that are filled in. The obvious negative is that the answers seem very rigid

and not human-like. However, using a statistical tool, such as MorphoDiTa (Straková et al., 2014) for the inflection of rarely used Czech words with a foreign origin often led to mistakes that were less likely to be forgiven by the user. Since the number of necessary inflections is limited, future work can include the prepared inflections in the database and then there will be no need to generate them.

The last approach is calculating the values first and then putting them into a template, such as the one in Example 4.1. It is used when giving individualized investment advice. First, the dialog manager has to decide which templates to use based on its calculations. In the previous case, the templates are picked beforehand no matter what the output values were. However, in this case, there is a set of template types, and the Dialog Manager needs to pick a subset of the templates to fill with values and concatenate.

(4.1) "*Jednorázovou investici (amountStarting) Kč s pravidelnou investicí (amountRegular) Kč si raději ukládej do prasátka* :)" –
It might be a better idea to store the (amountStarting) CZK with a regular investment of (amountRegular) CZK in a piggy bank :)

5. Evaluation

In this chapter, we will discuss the evaluation methods and results. We will also provide analysis of the errors that occurred during the evaluation. In order to evaluate the chatbot, we prepared four tasks for the users, each task testing a different aspect of the chatbot.

5.1 Evaluation Method and Results

The experiments were evaluated using a questionnaire with the following questions in addition to entries for tester anonymous ID and task ID):

- **Did the chatbot complete its task?** The possible answers were *yes*, *no*, and *I do not know/There was no task*
- **Did the chatbot understand your questions?**
- **Were the chatbot's answers relevant?**
- **Would you use it again to obtain financial advice?**

The last three questions were rated on a Likert scale (Hastie, 2012, p. 143f.) with the following options: agree, mostly agree, neither agree nor disagree, rather disagree, disagree. There was also an option for the testers to write additional feedback.

We gave the same basic instructions to all groups of testers (Table 5.1).

The testing was done through a web interface depicted in Figure 5.1.

Free chat task. The first group had no specified task , just to talk to the bot freely. They had to figure out what to ask and how to ask it by themselves. The main goal of this experiment was to find out whether the chatbot is intuitive to use. The secondary goal was to test how many of the testers' questions would fall into the frequently asked questions category. There were three testers in this group. Two of them rather agreed that the chatbot understood them, one was neutral. The chatbot's answers were relevant according to two testers (agree and rather agree) and one was again neutral.

Original	*Investbot je chatbot, jehož úkolem je poradit ti ohledně investičních produktů. Umí odpovídat na dotazy o nich, porovnávat je mezi sebou a případně na základě získaných informací doporučit vhodný produkt. Mluví česky a rozumí jenom větám s diakritikou. Předpokládá, že ve zprávě najde jenom jeden dotaz nebo vzkaz.*
Translation	Investbot is a chatbot whose task is to give you advice about investment products. It can answer questions about them, compare them, and recommend the best product based on the given information. It speaks Czech and only understands sentences with proper diacritics. It expects only one question within a message.

Table 5.1: Basic Instructions for Testers

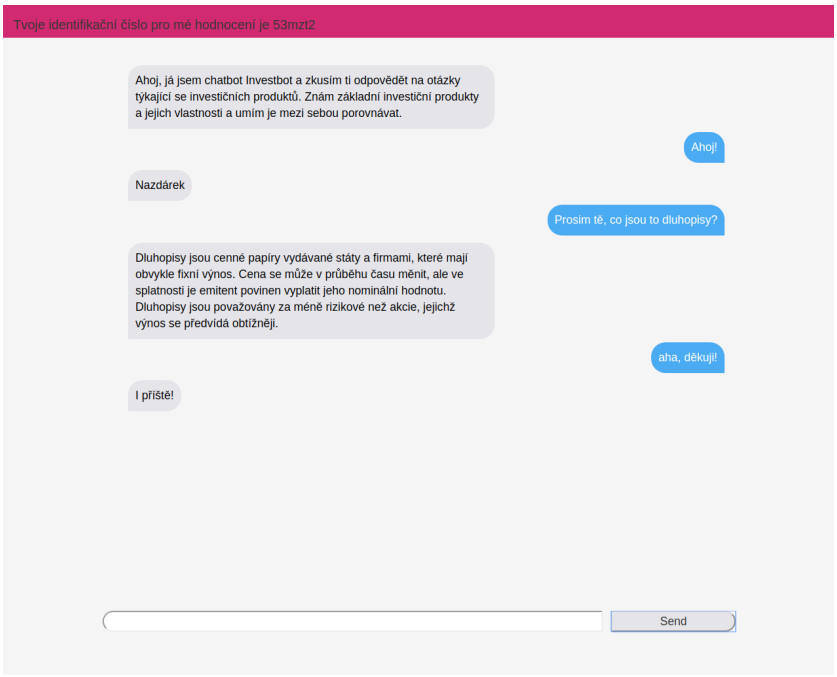


Figure 5.1: A screenshot of the testing interface.

Investment strategy task. The second group was supposed to ask the chatbot how to invest a certain amount of money, as shown in Table 5.2.

The goal of this experiment was to simulate the individualized advice

Original	Máš k dispozici částku nad 20 000 Kč (Konkrétní částku si, prosím, domysli), kterou si chceš nějakým způsobem nechat dlouhodobě zhodnotit. Nech si od Investbota poradit, který investiční produkt je pro tvoje potřeby nejvhodnější.
Translation	You have over 20 000 CZK (please think of a specific amount) worth of savings, and you want to increase that amount through investing for an extended period. Let Investbot recommend the best investment product for your needs.

Table 5.2: The Instructions for the Savings Task

task (see Section 2.3). The chatbot successfully noticed the testers wanted individualized advice and started asking questions. It was able to lead the testers through the questions without any significant problems. All of the users rather agreed the user understood their questions and its answers were relevant.

Product comparison task. The third group's task was to compare products. In order to do that, the testers needed to learn what products they could compare by asking the chatbot. The exact wording of the task in Czech can be found in Table 5.3.

There were several goals for this task. It tested how easy it would be for the testers to get the names of some products and their attributes. Then it examined how well the NLU Module would respond to various requests for comparison. Two of the three testers rather agreed that the chatbot understood their questions and gave relevant answers and one agreed to both.

Product explanation task. The last task assumes that the user already knows some terms from the field of investments, but does not know what they mean (Table 5.4).

This task was supposed to test how well the chatbot handles specific questions about investment products. All four testers rather agreed that the chatbot understood them and its answers were relevant.

Overall, all testers with a task description declared that the chatbot fulfilled its job. Furthermore, most testers continued after the task was com-

Original	Už delší dobu uvažuješ nad investováním svých úspor. V investičních produktech se moc nevyznáš a na Investbota se obracíš v naději, že ti pomůže získat alespoň základní povědomí o tvých možnostech. Vyber si pár produktů, informuj se o jejich vlastnostech a nech Investbota, ať je mezi sebou srovná na základě kritérii, na kterých ti záleží.
Translation	You have been considering investing your savings for a while. You do not know much about investment products and you are hoping that Investbot can help you gain some basic knowledge about investment products. Choose a couple of products, ask for information about their properties, and let Investbot compare them based on your criteria.

Table 5.3: The Instructions for the Product Comparison Task

Original	Kamarád se ti chlubil, že investoval do dluhopisů/akcií, ovšem tobě to nic neříká. Ptej se Investbota na vše, co tě o daném produktu zajímá.
Translation	One of your friends bragged that he invested into bonds/stock, however, you have no idea what that means. Ask Investbot anything you would like to know about the given product.

Table 5.4: Caption

pleted until they ran out of questions. The testers were students of colleges and universities in Prague. There were 13 testers randomly divided into four groups corresponding to the tasks.

The questionnaire results are in Appendix A in pie chart form, please see the thesis attachments for the .csv version and the dialog history.

5.2 Error Analysis

In this section, we will analyze the different types of errors that happened during testing. We will divide the problems into groups based on the cause. For each cause, we will give examples and make suggestions for future work.

5.2.1 Missing vocabulary / database entries

Some questions referred to attributes that were present neither in the ontology nor in the database, such as the recommended investment horizon. In one case a wrong calculation was made because the number dictionary did not contain the fraction *half*. Instead of interpreting the amount as half a million, the chatbot assumed that the user wanted to invest a million.

The most severe missing vocabulary problem was users asking about a term used in a definition that was unclear to them:

(5.1) **User**: *Jaké je riziko akcií?* – What is the risk of stock?

(5.2) **Chatbot**: *Investování do konkrétních akcií s sebou navíc ve srovnání s akciovým fondem, nese rizika defaultu konkrétního akciového titulu.* – Investing in specific stock has additional risks in comparison to the stock fund, because it carries the risk of the default of the specific stock.

(5.3) **User**: *Co je to default konkrétního akciového titulu?* – What is the default of the specific stock?

In some cases, the chatbot replied, "Was this the question?" and proceeded to rephrase the user's query, including the identification for the mentioned products, illustrated in Table 5.5.

Original	*Zněla otázka takto?, Otázku jsem pochopil jako "Jaký je (atribut) státní podpora (stateSubsidy)*[1] *u (produkt) Akcie (stock)."*
Translation	Was this the question? I understood the question as "What is the (attribute) state subsidy (stateSubsidy) of the (product) Stock (stock)."

Table 5.5: Response when the value in the database was not found.

This type of answer is very helpful to the system developer because it can tell us exactly what went wrong. In this case, both state subsidy and stock are in the database. However, the database has no entries about state subsidy of stock. In case either the product name or the attribute name was *null*, it means that the product or attribute (respectively) does not exist in the database.

All of the problems mentioned in this section can be solved by adding the given vocabulary into the ontology, by adding information into the database, or both.

5.2.2 Ontology ambiguity

During testing, we found out that hedge funds and mutual funds are both stored in the ontology under the instance mutual fund. Therefore, when found in the text by the NLU, they are labeled with the same URI (see Section 3.3.3). However, they are considered to be separate concepts in the database. In order to fix this problem, the two instances must be separated in the ontology.

We also ran into the opposite problem: A request for 'products' was easily handled; however, when asking the same question with 'investment products,' the chatbot failed to respond. This is because the two are considered to be different instances in ontology even though for the given tasks, there is no difference. In order to handle this problem, these two instances should be merged.

(5.4) *Jaké produkty znáš?* – Which products do you know?

(5.5) *Jaké investiční produkty znáš?* – Which investment products do you know?

5.2.3 Context

The users expected a more context-aware chatbot. Let us recall that this chatbot simply remembers a history of products explicitly mentioned by the user (see Section 4.1). Then it can tap into the history in order to retrieve the values of attributes of the previously mentioned product.

The first context-related problem occurred whenever the users followed the chatbot's questions until they received individualized advice. Then the chatbot offered them a product they have never heard about before, so naturally, they asked what it was. We imagine that adding the chatbot-suggested products into the context would not be much of an issue. However, the rule that would model a definition question with no term to define would have to be more strict in order to achieve a satisfactory precision.

(5.6) **Chatbot**: *Doporučuji ti investovat 2000 Kč do fondů peněžního trhu.* – I recommend you to invest 2000 CZK in monetary funds.

(5.7) **User**: *To je co?* – What is it?

The next context-related problem can be referred to as the negative context. Let us analyze it based on an example:

(5.8) *Řekni mi, prosím, něco ještě o nějakém dalším produktu.* – Please tell me something about some other product.

The user either expects to hear details about any specific product that has not been mentioned yet or he might want to hear a listing of products that have not been previously discussed. In both cases, the proposed solution for future work would be to get the set difference of products in the database and products mentioned. In case the user just wanted to hear something about one product, we can either select it by some given priority measure or randomly.

5.2.4 Implicit Queries

In some cases, the user's utterance did not contain any explicit questions or requests, but still carried a meaning:

(5.9) *No jde o to, že můj kamarád investoval do dluhopisů.* – Well, my friend invested in bonds.

In this specific case, the chatbot made a correct guess by responding with the definition of bonds. However, it might not always be so lucky, and therefore, it is essential to find a way to handle these types of messages. The chatbot should not say that it has no idea what the user is asking, because the user is not even asking. There are, of course, multiple possibilities of answering such messages. We suggest replying to the declarative sentences with no recognized meaning in a way that encourages the user to talk more, such as "I'm listening" until the user declares his intent in a more obvious way.However, finding the optimal strategy in this case would require more extensive testing.

Unsupported Types of Questions

Several users asked the chatbot "personal" questions, such as "Were you programmed in C++?" or "Who are your parents?". These questions would fall into the *ChitChat* category, so replying to them is not a priority.

A significant amount of people asked questions about why they should invest in some product. Even though this kind of questions is out-of-scope for the task of providing factual information, it makes sense to consider it in future work because it would make the chatbot more persuasive and lead to higher user satisfaction.

5.2.5 Other NLU errors

We have expected unsupported variants of supported questions to occur because it is never possible to think of all possibilities of asking the same question.

One example that occurred during the user tests is the following:

(5.10) *Mám tedy zvolit spíše dluhopisy či akcie ?*
Should I then choose rather bonds or stock ?
'Should I choose bonds or stock?'

In the context of the existing tasks, this could be interpreted as a request to compare bonds and stock. However, there were no rules that would describe the example above as a product comparison query. Adding them should solve the issue.

One of the trickier errors discovered while testing was when testers asked about the attributes of a previously mentioned product, but received a definition of the attribute instead. This initially seemed like a context problem. Nonetheless, upon examining the output JSON of the NLU Module, we discovered that the product-attribute interpretation was ranked lower than the attribute definition question that caught on. In order to fix this issue, we suggest either changing the weight coefficients used when ordering the interpretations or adding more importance to the coverage (see Section 3.3.3).

5.2.6 Grammar and Typos

A significant portion of the questions was not recognized correctly because the testers made typos or forgot to use diacritics. For these purposes, we suggest trying a statistical spellchecker, such as Korektor (Richter et al., 2012), and then running NLU with the corrected version as an input if running with the original fails.

6. Conclusion

Throughout this thesis, we designed, implemented and evaluated a task-based chatbot. We started by introducing dialog systems in general. We introduced the goals of this thesis.

We provided a detailed description of the tasks to be carried out by the chatbot: recognition of frequently asked questions, answering questions about investment products, providing individualized advice, and handling a small set of chitchat phrases. We analyzed the tasks and described the roles of the NLU module and the dialog manager in those tasks.

Then we introduced the natural language understanding component of the chatbot. First we discussed the frameworks, tools, and technologies we used. Then we presented the methods and approaches for rule-making. Finally, we described the structure of the NLU module and its output.

Then we discussed the dialog manager component. We explained how the dialog manager produces answers based on the NLU interpretations and the context it holds. We also explained how the answers are transformed into natural language.

Finally, we evaluated the chatbot using human testers. First, we described how the experiments were conducted and presented the results based on the testers' satisfaction. Finally, we carefully examined the evaluation results and provided thorough error analysis. We provided suggestions that could lead to improvements in future work.

6.1 Lessons Learned

Through the evaluation, we noticed that less than 5% of the tester messages were in the frequently asked questions category. This shows that the formulation of the task itself may be even more critical than its implementation. We suggest that in order to implement such a task, it should be based on a history of real questions of real customers. Even so, we believe most of them could successfully be represented using the product-attribute approach.

On the other hand, the questions about investment products were able to systematically cover the majority of the testers' questions. Furthermore, the existing database of products and attributes could gradually be expanded in order to support even more types of questions. We conclude this is a reasonable approach for a rule-based dialog manager.

The frame-based individualized advice task proved to be a good way to lead a user that has no prior knowledge of the banking domain or investments. We also consider this to be a good approach.

Overall, the chatbot succeeded in most of the types of questions it was supposed to understand. The majority of users concluded that the chatbot understood them correctly and provided relevant responses. Therefore, we are satisfied with the achieved results.

Bibliography

Eduard Bejček, Jarmila Panevová, Jan Popelka, Pavel Straňák, Magda Ševčíková, Jan Štěpánek, and Zdeněk Žabokrtský. Prague Dependency Treebank 2.5 – a revisited version of PDT 2.0. In *Proceedings of the 24th International Conference on Computational Linguistics (Coling 2012)*, pages 231–246, 2012. URL `http://www.aclweb.org/anthology/C12-1015`.

Daniel G Bobrow, Ronald M Kaplan, Martin Kay, Donald A Norman, Henry Thompson, and Terry Winograd. GUS, A Frame-Driven Dialog System. *Artificial Intelligence*, 8:155–173, 1977.

Alexander Clark, Chris Fox, and Shalom Lappin, editors. *The handbook of computational linguistics and natural language processing*. Blackwell handbooks in linguistics. Wiley-Blackwell, Chichester, West Sussex ; Malden, MA, 2010. ISBN 978-1-4051-5581-6 978-1-118-34718-8. OCLC: ocn500823419.

Hamish Cunningham, Valentin Tablan, Angus Roberts, and Kalina Bontcheva. Getting more out of biomedical documents with gate's full lifecycle open source text analytics. *PLOS Computational Biology*, 9 (2):1–16, 02 2013. doi: 10.1371/journal.pcbi.1002854. URL `https://doi.org/10.1371/journal.pcbi.1002854`.

Jan Dědek. *Semantic Annotations*. PhD thesis, Charles University in Prague, Czech Republic, 2012.

Jianfeng Gao, Michel Galley, and Lihong Li. Neural approaches to conversational AI. *CoRR*, abs/1809.08267, 2018. URL `http://arxiv.org/abs/1809.08267`.

Helen Hastie. Metrics and evaluation of spoken dialogue systems. Springer, New York, 2012.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2019. ISBN 0131873210.

Michael McTear, Zoraida Callejas, and David Griol. *The Conversational Interface: Talking to Smart Devices*. Springer Publishing Company, Incorporated, 1st edition, 2016. ISBN 3319329650, 9783319329659.

Mark A. Musen. The protégé project: a look back and a look forward. *AI Matters*, 1(4):4–12, 2015. doi: 10.1145/2757001.2757003. URL https://doi.org/10.1145/2757001.2757003.

Ashwin Ram, Rohit Prasad, Chandra Khatri, Anu Venkatesh, Raefer Gabriel, Qing Liu, Jeff Nunn, Behnam Hedayatnia, Ming Cheng, Ashish Nagar, Eric King, Kate Bland, Amanda Wartick, Yi Pan, Han Song, Sk Jayadevan, Gene Hwang, and Art Pettigrue. Conversational AI: the science behind the alexa prize. *CoRR*, abs/1801.03604, 2018. URL http://arxiv.org/abs/1801.03604.

Michal Richter, Pavel Straňák, and Alexandr Rosen. Korektor–a system for contextual spell-checking and diacritics completion. In Martin Kay and Christian Boitet, editors, *Proceedings of the 24th International Conference on Computational Linguistics (Coling 2012)*, pages 1–12, Mumbai, India, 2012. IIT Bombay, Coling 2012 Organizing Committee.

P. Sgall, E. Hajičová, and J. Panevová. *The meaning of the sentence in its semantic and pragmatic aspects*. D. Reidel, Dordrecht, 1986. ISBN 90-277-1838-5.

Milan Straka and Jana Straková. Tokenizing, pos tagging, lemmatizing and parsing ud 2.0 with udpipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver, Canada, August 2017. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/K/K17/K17-3009.pdf.

Jana Straková, Milan Straka, and Jan Hajič. Open-Source Tools for Morphology, Lemmatization, POS Tagging and Named Entity Recognition. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 13–18, Baltimore, Maryland, June 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P14/P14-5003.pdf.

Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):36–45, January 1966. ISSN 0001-0782. doi: 10.1145/365153.365168. URL http://doi.acm.org/10.1145/365153.365168.

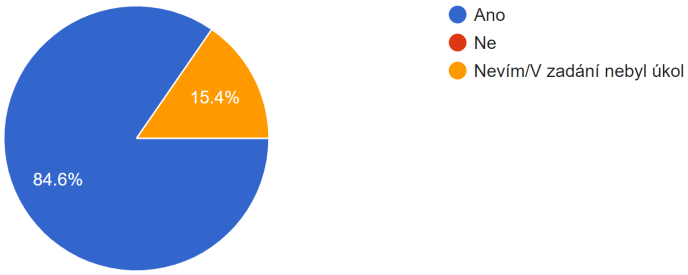
Z. Žabokrtský, J. Ptáček, and P. Pajas. TectoMT: highly modular MT system with tectogrammatics used as transfer layer. In *Proceedings of the*

Third Workshop on Statistical Machine Translation, pages 167–170. Association for Computational Linguistics, 2008. URL https://www.aclweb.org/anthology/W08-0325.

A. Questionnaire Results

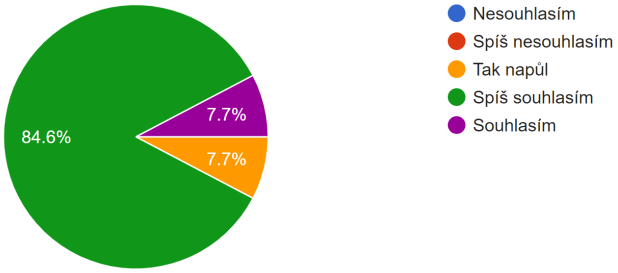
Splnil chatbot svůj úkol?

13 responses



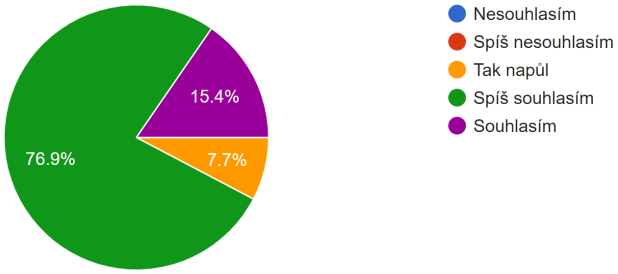
Rozuměl chatbot tvým dotazům?

13 responses



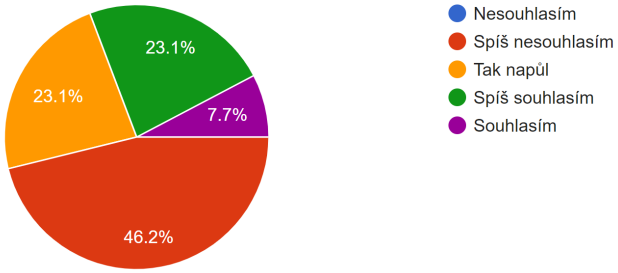
Dával relevantní odpovědi?

13 responses



Použil bys ho v budoucnosti pro získání finančních informací?

13 responses



B. Setup Guide

The source code of this thesis requires proprietary libraries to function, it is impossible to run or build the chatbot without it!

Other requirements:

- Java 8
- Kotlin 1.2.50
- GATE Developer 8.4
- Gradle 4.4.1 (in order to build the project)
- Although not necessary, we strongly recommend IntelliJ Idea.
- postgres 10

The open source libraries will download automatically during the build.

Instructions for building the project on Ubuntu 18.10:

1. Navigate into the root project folder (chatbot)
2. run "./gradlew build -x test –refresh-dependencies"

Once the project is built, there are two ways to test the chatbot:

1. **Using the BankingTest Class**: This approach tests the NLU module alone. Write any textual data to be analyzed into the testing document, change the gatePath variable to the actual location of GATE Developer, and run the test. Using this approach, the GATE Developer window will pop up and enable exploring the created *Annotations*. This approach does not require the database to be running.
2. **Using the Application in the Backend Module**: This approach runs the full chatbot as a client-server program based on Spring Websocket. First, you need to change the gatePath variable in *src/main/application.properties* to the actual location of GATE Developer. Make sure the postgres database is running and then run the Application. Once the Application is running, you can connect to it on localhost:8080.