UDC: 004.93 DOI: https://doi.org/10.15407/jai2021.02.088

SOFTWARE ARCHITECTURE OF THE QUESTION-ANSWERING SUBSYSTEM WITH ELEMENTS OF SELF-LEARNING

A. Hlybovets[1], A. Tsaruk[2]
[1, 2] National University of Kyiv-Mohyla Academy
Skovorody str, 2, Kyiv, 04070

[1] https:// orcid.org / 0000-0003-4282-481X

Abstract. Within the framework of this paper, the analysis of software systems of question-answering type and their basic architectures has been carried out.

With the development of machine learning technologies, creation of natural language processing (NLP) engines, as well as the rising popularity of virtual personal assistant programs that use the capabilities of speech synthesis (text-to-speech), there is a growing need in developing question-answering systems which can provide personalized answers to users' questions. All modern cloud providers proposed frameworks for organization of question answering systems but still we have a problem with personalized dialogs. Personalization is very important, it can put forward additional demands to a question-answering system's capabilities to take this information into account while processing users' questions.

Traditionally, a question-answering system (QAS) is developed in the form of an application that contains a knowledge base and a user interface, which provides a user with answers to questions, and a means of interaction with an expert. In this article we analyze modern approaches to architecture development and try to build system from the building blocks that already exist on the market. Main criteria for the NLP modules were: support of the Ukrainian language, natural language understanding, functions of automatic definition of entities (attributes), ability to construct a dialogue flow, quality and completeness of documentation, API capabilities and integration with external systems, possibilities of external knowledge bases integration

After provided analyses article propose the detailed architecture of the question-answering subsystem with elements of self-learning in the Ukrainian language. In the work you can find detailed description of main semantic components of the system (architecture components).

Keywords: question-answering systems, chatbots, Google Dialogflow, self learning, software architecture.

АРХІТЕКТУРА ПРОГРАМНОГО ЗАСТОСУНКУ ПИТАЛЬНО-ВІДПОВІДАЛЬНОЇ ПІДСИСТЕМИ З ЕЛЕМЕНТАМИ САМОНАВЧАННЯ

А.М. Глибовець[1], А.П. Царук[2]
[1, 2] Національний університет «Києво-Могилянська академія»
вул. Сковороди, 2, м. Київ, 04070

[1] https:// orcid.org / 0000-0003-4282-481X

Анотація. У рамках даної роботи проведено аналіз програмних систем запитально-відповідного типу та їх базових архітектур.

З розвитком технологій машинного навчання, створенням механізмів обробки природної мови (NLP), а також зростанням популярності віртуальних персональних помічників, які використовують можливості синтезу мовлення (перетворення тексту в мовлення), зростає потреба в розробці питально-відповідальних систем, які можуть вести персоналізований діалог з користувачем. Усі сучасні хмарні провайдери запропонували фреймворки для побудови питально-відповідальних систем, але ми все ще маємо проблему з персоналізованими діалогами на основі баз знань. Персоналізація дуже важлива, вона може висувати додаткові вимоги до архітектури системи при веденні діалогу з користувачем.

Традиційно систему запитань-відповідей (QAS) розробляють у вигляді програми, що містить базу знань та інтерфейс користувача, що надає користувачеві відповіді на запитання, і засіб взаємодії з експертом. У цій статті ми аналізуємо сучасні підходи до розробки архітектури та намагаємося побудувати систему з блоків, які вже існують на

ринку. Основними критеріями при обранні модулів NLP були: підтримка української мови, розуміння природної мови, функції автоматичного визначення сутностей (атрибутів), здатність побудови діалогового потоку, якість і повнота документації, можливості API та інтеграція із зовнішніми системами, можливості інтеграції зовнішніх баз знань.

Після проведеного аналізу в статті пропонується детальна архітектура питально-відповідальної підсистеми з елементами самонавчання українською мовою. У роботі ви знайдете детальний опис основних семантичних компонентів системи (компонентів архітектури).

Ключові слова: питально відповідальні системи, чат-бот, Google Dialogflow, самонавчання, архітектура програмного застосунку.

Introduction

In nowadays world, there is no area of life that could be possible without the daily use of information technologies and the Internet. Knowledge and information are stored in digital formats, print media are being almost completely replaced by the capabilities of information systems. Due to the increase in data, it is becoming increasingly difficult to find relevant answers to our questions in the ocean of information. This is changing both the needs of users for information retrieval capabilities, and the ways to meet those needs. In particular, this applies to such a class of information systems as question-answering systems. The first systems of this kind were developed as natural language shells for expert systems applied in specific areas. Modern automated question-answering systems are designed to work with much broader subject areas, and are being now actively used in commercial, social, educational and other fields [1].

With the development of machine learning technologies, creation of natural language processing (NLP) engines, as well as the rising popularity of virtual personal assistant programs that use the capabilities of speech synthesis (text-to-speech), there is a growing need in developing question-answering systems which can provide personalized answers to users' questions. This is because an answer to a certain question often depends on additional contextual data or on additional information about, for example, the author of the question etc. Such information can be personalized, which puts forward additional demands to a question-answering system's capabilities to take this information into account while processing users' questions.

Approaches to the development of information systems are also undergoing significant changes. With the development of cloud technologies, the need for self-administration of infrastructures has been eliminated, and such solutions as PaaS (platform as a service) and SaaS (software as a service) have started to appear. Complex monolithic architectures of information systems are beginning to be replaced by such approaches as microservices development, the use of serverless technologies and ready-made services used as components in the software development [2].

That is why the research of the development of architecture of a question-answering system with elements of self-learning is important and relevant.

Development of the architecture of a question-answering subsystem with elements of self-learning

The basic architecture of highly specialized question-answering systems is shown in Figure 1.

Traditionally, a question-answering system (QAS) is developed in the form of an application that contains a knowledge base and a user interface, which provides a user with answers to questions, and a means of interaction with an expert. In order to decompose the system into separate components, let us define the following main external components: a user interface, an engine for processing and understanding natural language, knowledge base and algorithms for finding answers. The main part of a question-answering system, which will carry out the integration of components, will be called an integration subsystem.

Developing a graphical user interface can be a separate task. However, we decided to use a ready-made solution. The advantages of such approach are a large number of available ready-made solutions with effective integration tools

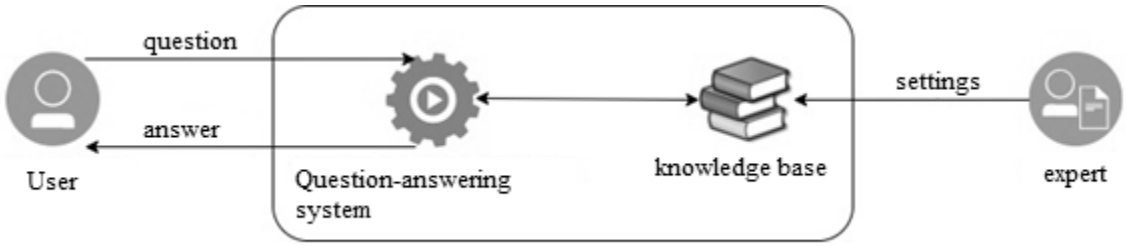

Fig. 1. The basic architecture of highly specialized question-answering systems

which make it possible to cut down the development time by adding new graphical interfaces with minimal changes.

Since the system is designed for building a dialogue, and the number of users is not limited, we have chosen an ideal tool for messaging – a chatbot. Most of messaging systems also provide advanced ways of integration with external systems. This approach allows to easily realize new integrations into the system by adding a controller and conducting a message transfer to a specialized processor. The general architecture of this approach to integration is shown in Fig.2.

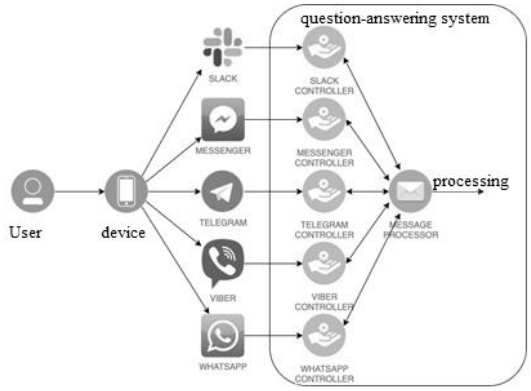

Figure 2. Architecture of integration of a question-answering system with messaging systems

It is clear that a natural language understanding engine is one of the key components of the question-answering system. It performs the task of processing a user's question, namely identification of an intention (goals, objectives) behind a user's question and definition of certain contextual entities (attributes) of the question.

The tasks of processing and understanding natural language (Natural Language Processing - NLP, Natural Language Understanding - NLU) are complex tasks in machine learning [3]. Large cloud solution providers, such as Microsoft, IBM, Google, Amazon, have been constantly working to develop and improve their own products of this kind [4, 5, 6, 7]. We analyzed products they offer and selected candidates for the engine position. The evaluation criteria were:

1. Support of the Ukrainian language.
2. Natural language understanding.
3. Functions of automatic definition of entities (attributes).
4. Ability to construct a dialogue flow.
5. Quality and completeness of documentation.
6. API capabilities and integration with external systems.
7. Possibilities of external knowledge bases integration.

There were examined the following software products: Watson Assistant from IBM, QnA Maker and Bot Framework from Microsoft, DialogFlow from Google and Amazon Lex from Amazon. The table 2.1 gives the results of the comparative analysis of these products. Depending on a criterion, the assessment was conducted either using a five-point rating scale (1-5) or a two-point one (yes/no).

We considered Watson Assistant and DialogFlow to be the best candidates. However, only DialogFlow provides opportunities for understanding the Ukrainian language, and that determined the final choice.

The scheme of integration with the engines of processing and understanding natural language is shown in Figure 3. As can be seen from the figure, a question-answering system

receives questions from a user via Telegram, sends it to the DialogFlow engine, which searches for answers and sends them back to the user.

Table 2.1. The results of the comparative analysis

Criterion	Watson Assistant	QnA Maker	DialogFlow	Amazon Lex
Support of the Ukrainian language	no	no	yes	no
Natural language understanding	yes	yes	yes	yes
Definition of entities	yes	no	yes	yes
Constructing a dialogue flow	yes	no	yes	yes
Documentation	5	2	4	3
API capabilities	4	3	4	4
Internal knowledge base	no	no	no*	no

* available in a beta version in the English language

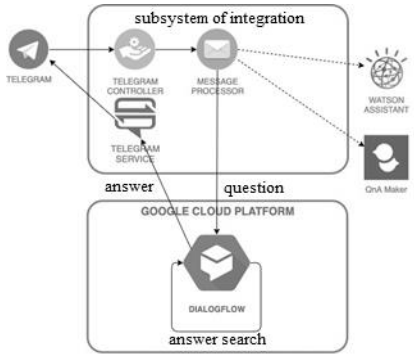

Fig. 3. The scheme of integration with the engines of processing and understanding natural language

Knowledge base and algorithms for finding answers

This component of the system is designed to find and determine answers to questions from users. One of the objectives of the paper was to introduce elements of self-learning, therefore, in order to implement this component of the system, applying just a third-party solution would not be enough.

Developing solutions for the system to perform its task to construct a dialogue flow with a user, we have identified the following components of the dialogue:

Context-independent questions, an answer to which can be formulated by default every time such a question is asked. An example of such a question could be «Коли відбувається вступ в НаУКМА?» ("When are the NaUKMA entrance exams?") Answers to such questions can be obtained from the knowledge base or another place where the information is stored by a key or another identifier.

- *Context-dependent questions,* an answer to which may depend on the information available in the system and the current context. For example, if an applicant is communicating with the system that provides information on admission to Bachelor's programs, the system can clarify the question received «Які є спеціальності?» ("What are the study programs?"), by asking a user «Який факультет тебе цікавить?» ("Which faculty are you interested in?") and, after the clarification, it provides a list of study programs for a particular faculty, such as the Faculty of Informatics. In this case, a usual question-answering system will reproduce such a dialogue every time a user asks the same question. When implementing the self-learning function, the system can identify a user and "know" what he is interested in, so when a user asks «Нагадай, які є спеціальності?» ("Remind me, what are the study programs?"), after a while, the system will immediately give this user a list of study programs of the Faculty of Informatics. However, if a user asks «Які є спеціальності на факультеті гуманітарних наук?» ("What are the study programs at the Faculty of Humanities?"), the system will take into account the specific meaning of the essence of "faculty" present in the user's question.
- *Generic elements of a dialogue* that do not require a search on the knowledge base, and are used to logically build a dialogue. These can be elements of a "small talk", such as greetings and farewells, short phrases that complete a thread like "yes", "no", "of course" and so on. Such answers can be attributes of a dialogue and stored together with the dialogue itself.

Creation of dialogue scripts is supported by the DialogFlow system, it was given the task

to organize main possible directions of how a conversation with a user can develop. There one can also store generic elements of the dialogue, as well as consider the system as a possible place for storing context-independent data and working with them. It is also possible to use the graphical interface of the system, API calls, SDK libraries to configure dialogues.

The DialogFlow system is also meant to work with context-dependent user questions. This makes it possible to build typical dialogues, where the system "leads" a user according to a certain scenario. For example, a system that allows a user to place an order in an online store, by communicating with the chatbot, can provide to a user a description of a particular product, and upon receiving the question «А яка ціна?» ("And what is the price?") the system will inform the price of this particular product. However, support for these contexts can have some limitations. For instance, in DialogFlow the context lifespan automatically expires in 15 minutes, or after five conversational turns [3]. These limitations are nonessential for the development of standardized user dialogue scenarios, but considering self-learning tasks of a question-answering system, this means that a system based on them will at best have "short" memory, or no such characteristics at all. Another limitation of the DialogFlow is the inability to derive rules based on certain data.

Therefore, the module of intellectual processing of questions was taken out in a separate subsystem. This, in its turn, led to the necessity to solve the task of DialogFlow integration with a specific external system for processing messages from users. Such integration is provisioned in DialogFlow and is called "Fulfillment". The integration can be done in two possible ways:

1. Sending a request to an external system (webhook)

2. Using Google Cloud Functions

Since DialogFlow is a Google product that is realized as one of Google Cloud Platform's services, a serverless technology from the same platform, Google Cloud Functions, was chosen as the basis for the implementation of this module. Cloud Functions supports coding in such programming languages as JavaScript, Python, Go, Java [3]. The choice was driven by the following factors:

- built-in ability for integration of DialogFlow with Cloud Functions;
- the trigger model based on the occurrence of the event that fully meets the needs of the task (processing an event "a message from a user");
- the use of a single cloud platform which increases productivity;
- simplicity and rapid deployment of the solution.

Therefore, with this approach, the integration subsystem will be able to direct questions from a user to DialogFlow. Next, the NLP engine will be able to determine the essence of a question, as well as additional contextual entities present in the question. If the question cannot be processed in DialogFlow, this information will be sent to the part of the software application that is run in Cloud Functions. Hereinafter we will call this subsystem Fulfillment Cloud Functions. This subsystem will be able to receive or analyze both information obtained from DialogFlow and additional information, such as information about a user or data of previous conversations, including the one from external services and databases, and to make a response based on this information. Thus, this subsystem can complement and expand the current context of the dialogue, and generate an answer to a user, or generate certain events for DialogFlow, which will allow to control the dialogue in DialogFlow. Additionally, with the help of the Fulfillment Cloud Functions subsystem it will be possible, for instance, to obtain data from external knowledge bases, or dynamic information from external sources. After processing a request from DialogFlow, the subsystem will send a response to this request back to DialogFlow, which in turn will send a response to a user to the integration subsystem.

If necessary, a module for managing DialogFlow via API can be developed and added to an integration subsystem; let's call it DialogFlow API Admin.

A part of the architecture of the question-answering system related to the Google Cloud Platform is shown in Figure 4.

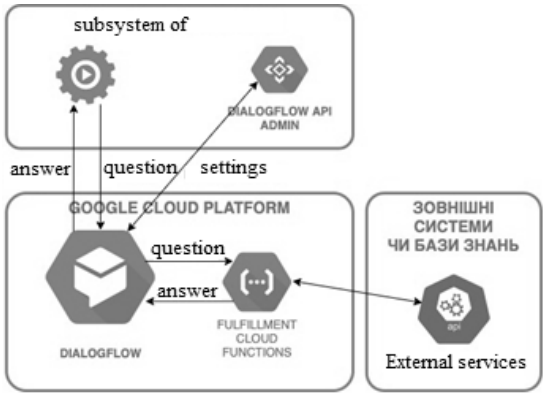

Figure 4. Organizing work and interaction of Google Cloud Platform components

Upon selection of individual components, the general view of the QAS architecture will be as shown in Figure 5.

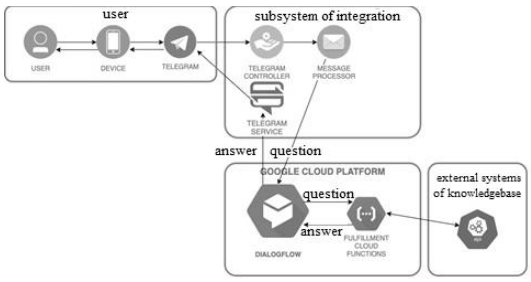

Figure 5. The general view of the architecture of a QAS upon selection of individual components

So far, we have only mentioned the details of the integration subsystem design, considering it rather as a certain whole object. Now we are going to explain its architecture and main functions in more detail.

Let's start with the functionality. It is worth noting the following:

1. *Integration with the Telegram messaging system*. Receiving questions from users from Telegram and sending answers to users to Telegram.
2. *Processing of received messages with questions.* A message is not only a text entered by a user, but also some additional data, like a user or a conversation ID, some information about a user etc. Such data can be processed by the system with the aim to save service information necessary for the operation of the system. Besides that, messages may contain data that cannot be processed by the question-answering system (i.e., images), and should be ignored, or, on the other hand, contain control commands designed to control the settings of the system itself. Therefore, we have also included in the processing of messages processing of questions and sending them to the NLU engine, processing and execution of commands for the subsystem, and processing of additional message data.
3. *Integration with the engine of processing and understanding natural language.* It includes sending questions from users and getting answers from the engine. It must be noted that in addition to the answer itself, the integration subsystem must also receive from DialogFlow additional information about the context of the dialogue, identification of a user's intention, the entity data that was singled out by the engine from the original question. In order that a QAS could perform its self-learning function, the integration subsystem must conduct the processing and be responsible for storing this data for later use, in particular in the Fulfillment Cloud Functions subsystem.

As it can be seen from the functionality description, such integration subsystem will be able to store certain data, namely, information about a user and DialogFlow data, which can be later used in the Fulfillment Cloud Functions subsystem. The data has a "key-value" format, and the values can be either of simple types (i.e., numeric, string, logical), or consist of sets of values of simple types. Therefore, to effectively store such data, it is recommended to choose a "key-value" or document-oriented database.

After analyzing the database ratings [8], we chose Amazon DynamoDB.

In order to store data in the integration subsystem, it is desirable to use a multilevel architecture model [4]. To simplify the architecture scheme, we introduced User Service for services responsible for storing user data, and Context Service for services responsible for storing contextual data received from DialogFlow.

For the integration with the engines of processing and understanding natural language, we have chosen to develop our own services – DialogFlow Service, IBM Watson Service and QnA Maker Service, respectively. Transparent organization of work with any service should ensure an additional provider – NLU Service Provider, which will act as a facade and redirect requests to a certain service, depending on the current settings. DialogFlow Service will receive users' questions from the Text Processor through the provider and send them for processing to DialogFlow. After receiving the result from DialogFlow, the service will send context data to the Context Service for further storage, and send the answer to the user to the Telegram Service so it can be sent to Telegram. The detailed architecture of the integration subsystem is shown in Figure 6.

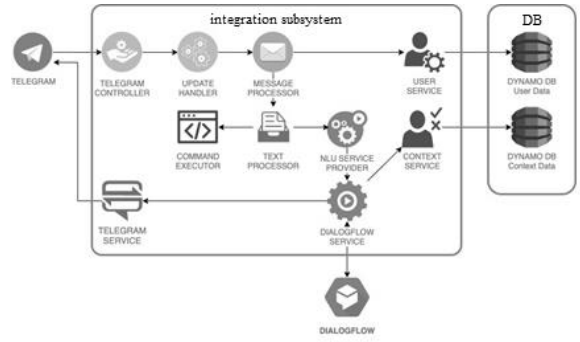

Figure 6. The architecture of the integration subsystem

And finally, on the basis of our research, it is possible to present a detailed architecture of the question-answering subsystem with elements of self-learning. This architecture is shown in Figure 7.

In this case, the integration subsystem, the Fulfillment Cloud Functions subsystem, and the Context API subsystem are software applications that must be realized directly in the process of development of this architecture.

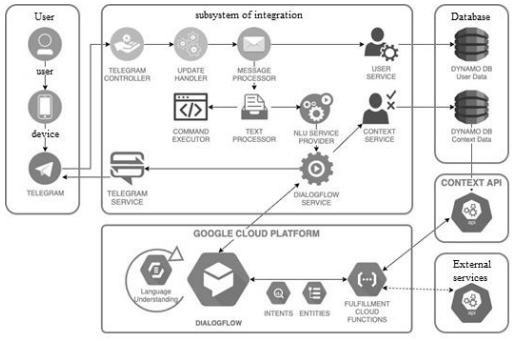

Figure 7. The detailed architecture of the question-answering subsystem with elements of self-learning

The developed architecture provides integration with other external systems through the Fulfillment Cloud Functions subsystem. At the same time, such integration is not critical for the operation of the key features of the system, and allows to further expand its capabilities.

Conclusions

This work has analyzed basic approaches to the development of software system architecture that ensures the effective operation of the question-answering system with elements of self-learning in the Ukrainian language. Based on the research, there have been selected software products (Google DiaglowFlow, Google Cloud Platform, Telegram, Amazon DynamoDB) that can be used as components in a modern question-answering system and ensure its optimal performance, taking into account the proposed methods of integration.

References

1. Ali Mohammed Nabil Allam, Mohamed Hassan Haggag. The Question Answering Systems: A Survey. International Journal of Research and Reviews in Information Sciences. 2012. 2. No3. pp.10-21.
2. Wilhelm Hasselbring. Component-based software engineering. International Journal of Software Engineering and Knowledge Engineering. May 2002.
3. Google DialogFlow. Google Cloud Platform. [Electronic resource]. Available at: https://cloud.google.com/dialogflow/docs/
4. Amazon Web Services. [Electronic resource]. Available at: https://docs.aws.amazon.com/
5. Fourault Sebastien. "The Ultimate Guide To Designing A Chatbot Tech Stack" 2017. [Electronic resource]. Available at:

https://chatbotsmagazine.com/the-ultimate-guide-to-designing-a-chatbot-tech- stack-333eceb431da

6. Mark Richards. Software Architecture Patterns. O'Reilly Media, Inc. 2015.
7. The Complete Guide to Chatbots in 2018. Sprout Social. [Electronic resource]. Available at: https://sproutsocial.com/insights/topics/chatbots
8. DB-Engines Ranking. [Electronic resource]. Available at: https://db-engines.com/en/ranking

Received 17.10.2021
Accepted 26.11.2021