Facebook Integrated Chatbot for Bulgarian Language Aiding Learning Content Delivery

George Pashev, Silvia Gaftandzhieva

University of Plovdiv "Paisii Hilendarski", 24 "Tsar Assen" Str., Plovdiv, Bulgaria

***Abstract* – The paper presents a chatbot oriented linguistic approach and a software prototype which addresses the need of delivering learning content based on queries in Bulgarian language. Two distinct database query generation approaches are presented, discussed and implemented. Pros and cons for each of them are discussed.**

***Keywords* – Facebook, Chatbot, Learning content.**

1. Introduction

Facebook has been widely researched as a tool for e-learning. Some statistical outcome has been presented in [1]. Towner and Muñoz [2] discusses some aspects of the Usage of Facebook as a Classroom Connection as a formal or informal tool for tutoring. As a widely used social network across Europe, it has become a viable opportunity for trainers/teachers to reach more of their students in a more informal and friendly way.

The topic of using the social network Facebook for semantic analysis of content, as well as for finding different users with a specific profile is becoming more and more actual.

Al-Kouz, de Luca and Albayrak [3] present a model and system for finding experts in various subject areas using Facebook. Various attempts can be found in the literature to evaluate expertise through statistical models such as classification to k closest and evaluation of their expertise. Neil [4] reviews the use of social networking sites in education. Andriani [5] describes some aspects of using social networks as a tool for finding future university of prospective students. Balakrishnan and Gan [6] explore the relationship between students' learning styles and the way they use social networking sites. All these works prove that the direction related to the use of social networks in e-learning is promising as well as the finding of appropriate content or an expert to do a specific targeted work.

Kazanidis et al. [7] present a quantitative analysis comparing Moodle LMS and Facebook as LMS. Users of Facebook and Moodle face similar user experience, but Facebook outcompetes Moodle as a platform with a much more significant social presence.

In the light of the recent Coronavirus Pandemic, some universities meet various problems in e-learning. Students need to use various LMS platforms and video conferencing software in the context of different courses and different requirements of different professors. The last imposes unnecessary overhead to their everyday routine. Moreover, the lack of physical contact with professors makes things even worse. Students experience more and more difficulties in reaching proper educational content that would meet both personal learning requirements and professors' requirements.

To address the issues presented in the above paragraph, more automation and integration with popular social networks is needed. An essential means are chatbots because they tend to be user friendly and provide more informal interaction. Moreover, they provide a very simple GUI, integrated well with social media and various LMSs.

This paper presents a model and system that facilitates the e-learning process, as well as finding useful learning data or expert educators from the learners through the use of a virtual assistant, which is a virtual Facebook user. It identifies some requirements needed to achieve such integration with

DOI: 10.18421/TEM103-01
https://doi.org/10.18421/TEM103-01

Corresponding author: George Pashev,
University of Plovdiv "Paisii Hilendarski", 24 "Tsar Assen" Str., Plovdiv, Bulgaria
Email: georgepashev@uni-plovdiv.bg

Received: 18 April 2021.
Revised: 02 July 2021.
Accepted: 09 July 2021.
Published: 27 August 2021.

chatbots and a software prototype that to a certain extent meets them. Convenient usage of semantic oriented databases such as Elastic [8] is an essential part of the solution. However, sometimes chatbots need to be even more robust and try to generate queries to university databases in such cases in which semantic search fails to find enough relevant objects in the semantic database. An attempt to solve this problem is also presented. Also, this approach would enable the chatbot to answer questions that are not merely related to course contents, which are available as Learning Objects in the semantic database.

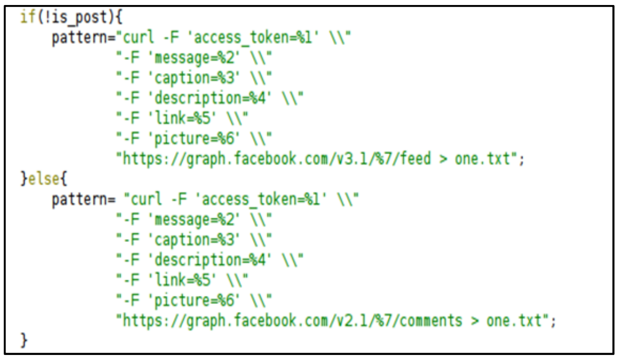
2. Realization

In the first step of the realization, the requirements for the model have been defined. The proposed model has to meet the following requirements:

- automate the process of posting on Facebook;
- maintain a connection to the knowledge base of learning objects and experts;
- knowledge base of learning objects and experts to support semantic indexing and semantic search;
- perform a semantic search on Facebook users' queries in the knowledge base and to return the closest result of the user's query;
- generate SQL queries in RDB from user input in a natural language if a semantic search in semantic database “fails” to fetch enough relevant objects.

The following part of this section describes the realization of each requirement defined above.

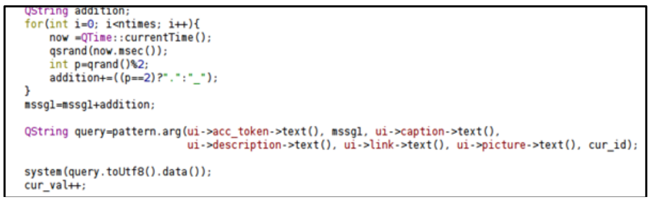
To automate Facebook posting, a C/C ++ based component is used, which through the REST API of the Facebook Graph API [9] posts content in a list of target objects (e.g., Facebook groups, pages, the wall of the virtual user). Different templates are used in cases where a post or comment on a post is created (see Figure 1).



```
if(!is_post){
    pattern="curl -F 'access_token=%1' \\"
            "-F 'message=%2' \\"
            "-F 'caption=%3' \\"
            "-F 'description=%4' \\"
            "-F 'link=%5' \\"
            "-F 'picture=%6' \\"
            "https://graph.facebook.com/v3.1/%7/feed > one.txt";
}else{
    pattern= "curl -F 'access_token=%1' \\"
            "-F 'message=%2' \\"
            "-F 'caption=%3' \\"
            "-F 'description=%4' \\"
            "-F 'link=%5' \\"
            "-F 'picture=%6' \\"
            "https://graph.facebook.com/v2.1/%7/comments > one.txt";
}
```

Figure 1. Templates of REST queries to GRAPH API

This component also uses a strategy to "deceive" Facebook that the user is human by making random delays in posts and comments. After the delay, it makes the request itself (see Figure 2).



```
QString addition;
for(int i=0; i<ntimes; i++){
    now =QTime::currentTime();
    qsrand(now.msec());
    int p=qrand()%2;
    addition+=((p==2)?".":"_");
}
mssgl=mssgl+addition;

QString query=pattern.arg(ui->acc_token->text(), mssgl, ui->caption->text(),
                          ui->description->text(), ui->link->text(), ui->picture->text(), cur_id);

system(query.toUtf8().data());
cur_val++;
```

Figure 2. Creation of request from template and data

The component for supporting semantic indexing and search uses the Elastic (or ElasticSearch) engine, which is a well-known Java and REST-based NoSQL engine for storing and semantic indexing of JSON objects.

Trial JSON structures of learning objects and experts (teachers) have been created. Figure 3 presents an exemplary tree structure of an expert.

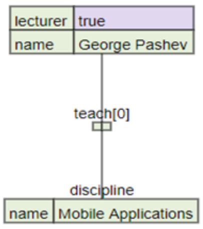


Figure 3. Tree structure of a JSON object for an expert (teacher)

To be able to search for an object by semantic similarity, the natural language query has to become a template JSON search object. The following algorithm has been implemented for this purpose (see Figure 4):

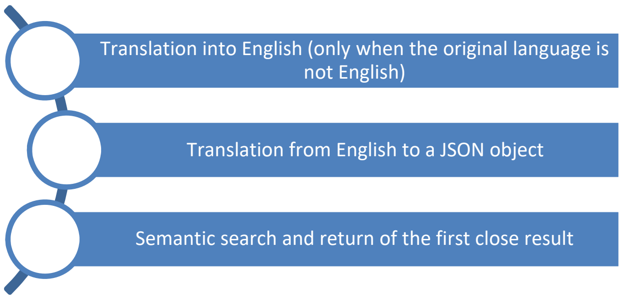

Translation into English (only when the original language is not English)
Translation from English to a JSON object
Semantic search and return of the first close result

Figure 4. Sequence of steps in semantic search

We will look at each of these steps with a specific search example. Let us look at the sample query: “Which lecturer teaches on mobile apps?”. This query goes through the following steps:

- Step 1. Translation into English - in this case, the language is English and we will skip this step: “Which lecturer teaches on mobile apps?”;
- Step 2. Translation from English to a JSON object – the result after translating to a template JSON object is {lecturer: true, teach: “mobile apps”};
- Step 3. Semantic search and return of the first close result - after performing the semantic search, it finds the nearest object, described in Figure 3.

For step 1: Machine translation into English, the Java library is used: java-google-translate-text-to-speech [10].

For step 2: Translation to a JSON object, an algorithm is implemented. For each simple sentence (the separator between "simple sentences" can also be ",", ";") the algorithm does the following:

- removes question marks and prepositions, leaving only nouns and verbs - e.g., in the example remains "lecturer teaches mobile apps";
- turns nouns into a basic form: "lecturer teach mobile app";
- in the new list of nouns and verbs it does the following:
    - if there is a noun that is not followed by another noun, it turns it into a predicate attribute, e.g., lecturer: true;
    - if there is a verb followed by a group of nouns, the name of the verb serves as the name of the new attribute, and the nouns serve as the text value of the attribute; e.g.: teach: "mobile app";
    - if there are only nouns in a simple sentence, the first one is used for the attribute name and the others for the value.

For the steps requiring the definition of the parts of speech and the basic forms, the WordsApi knowledge base accessible through the REST API is used [11].

To perform a semantic search on Facebook users' queries in the knowledge base and to return the closest result of the user's query, it is necessary to obtain a natural language response from the structure of the JSON object. The implemented algorithm does the following:

- it first looks for an attribute named name and uses its value as the beginning of a sentence. Then, for each other attribute:
    - if it is a noun with the value predicate: it generates part of a sentence: e.g. from lecturer: true will generate *is a lecturer*;
    - if it is a verb, it converts the verb to the third person and from the value generates a string for nouns then, e.g. from teach: [discipline: {name: "Mobile Applications"}] it will generate teaches Mobile Applications;
    - if it is a noun with a value other than Boolean, then the possessive is generated by the following example: "name": "Georgi Pashev" becomes whose/which name is Georgi Pashev.

In all three variants, if the value is a complex object, it will be subjected to a string recursive transformation of the object according to the same rules.

Figure 5 presents the overall structural diagram of the virtual assistant. When a query is placed (text comment or post on the wall or personal message) by a user, the Search Query Generator (JSON object) from text saves the query in a collection of queues in Elastic.

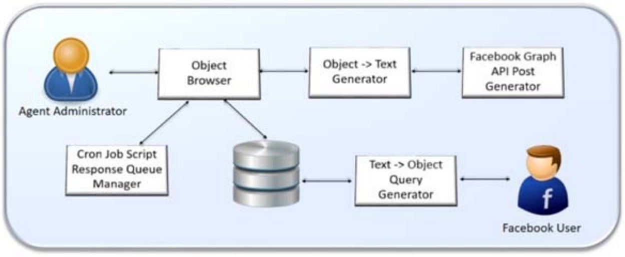

Agent Administrator
Object Browser
Object -> Text Generator
Facebook Graph API Post Generator
Cron Job Script Response Queue Manager
Text -> Object Query Generator
Facebook User

Figure 5. Structural diagram of the virtual assistant

Then, a special script (called by the cron service) takes the next request in the queue and finds the semantically closest object corresponding to the proceeded request. The found object passes through a Generator of text from an object and subsequently, a comment (response) is posted using Graph ID of the text of the request through the Generator of posts, which uses the Facebook Graph API. If one wishes, the agent administrator can publish (for example, on the agent wall) a selected JSON object according to the same scheme as the cron script.

Sometimes objects in the result from execution of the queries in Elastic are not similar enough to the query object. Semantic oriented databases in practice are widely targeting very specific applications and their structure is quite limited in addressing the needs of those applications. This is why quite often user input queries might require answers, that are not present in those databases as query objects or data is structured inconveniently.

In such a case, we incorporate a strategy to try to translate the natural language query to SQL and try to perform a search in various SQL databases, which tend to be richer in content.

An algorithm for translation from natural language (currently Bulgarian) to SQL is currently implemented in C# and makes use of a proprietary morphological analyzer, which will be discussed in future papers.

The algorithm executes in the following phases:

- splits user input into sentences;
- for each sentence it does normalization, tagging and generation of a List of SelectQuery Item.

Normalization is the process in which "stopwords" are removed from the sentence – for example, prepositions and other words that are considered to have no significance in the formation of the meaning. Punctuation is removed and capital letters are replaced with non-capital letters. Each word is then replaced with its base form.

Using our proprietary morphological analyzer, each word is tagged. The tag contains information about which part of speech is the word in the Bulgarian Language.

Merely linguistic tagging, however, is not enough to perform translation to SQL. The algorithm needs to find which word for example refers to the table name or column name in relation or view in the Relational Database. Available tags are

TABLE_NAME, COLUMN_NAME, VAL, NONE. The tag VAL refers to words that are recognized as search value and will be translated to value in the WHERE clause in the query.

After Tokenisation, the algorithm needs to find some relations between tokenized words. The following strategy is used:

1) All Table name tokens are found.

2) For each found column name, a query in SQL Structure of the Database is performed to which table in the sentence it is attributed to. If there is only one table, which is present in the sentence, then the column is associated with this table. If there are two or more table tokens in the sentence which might be associated with this column, then the one closest to the column in the text is selected and is associated with it.

3) If any table in the sentence is found, which can be associated with this column name, the column token is transformed to a token of type NONE and is excluded from future phases in the algorithm. (See Figure 6.)

```
//select queries structures generation after tokenization
List<SelectQuery> selects = new List<SelectQuery>();
foreach (var item in tokens) //loop through tokenized sentences
 {
  Dictionary<string, string> mapUniqueTables = new
  Dictionary<string, string>();
 SelectQuery selectQuery = new SelectQuery
 {
  uniqueTableNames =new List<string>(),
  columns=new List<KeyValPair>(),
haveValues=false
};
int iTokenItems = -1;
foreach (TokenItem token in item) //loop through words in sentence
 {
   iTokenItems++;
if (token.tokenID == TokenEnum.TOKEN_COLUMN_NAME)
     {if(!selectQuery.uniqueTableNames.
Contains(token.tableNameForCol))
selectQuery.uniqueTableNames.
Add(token.tableNameForCol);
KeyValPair kvp = new KeyValPair
      {key = token.word,tableName = token.tableNameForCol,
val = new List<string>() };
for (int i = iTokenItems + 1; i < item.Count; i++)
        {
         TokenItem lookaheadToken = item[i];
if (lookaheadToken.tokenID == TokenEnum.TOKEN_
COLUMN_NAME || lookaheadToken.tokenID ==
TokenEnum.TOKEN_TABLE_NAME)break;
if (lookaheadToken.tokenID == TokenEnum.TOKEN_VAL)
            {//perhaps the value after the column is meant
//to be for this column, use it
             kvp.val.Add(lookaheadToken.word);
             selectQuery.haveValues = true;
            }
}
selectQuery.columns.Add(kvp);
  }elseif (token.tokenID == TokenEnum.TOKEN_TABLE_NAME)
selectQuery.uniqueTableNames.Add(token.word);
}
selects.Add(selectQuery);
 }
```

Figure 6. Generation of SelectQuery Objects

Currently, the algorithm for the generation of Select Query from generated SelectQuery object (see Figure 7) has some limitations. If there are more than one table tokens, they are assumed to form SQL inner join. To make performance faster, all available inner join structures are previously generated and available in a dictionary, with keys – sets of tables, which are in the join and are added to the query as initially generated texts. Value tokens, which are physically after a column token in the text, are assumed to be values for this column. If the column is a numeric type, only numeric values are used for the value. If a column is of text data type, the operator SQL like is used in the formation of the text in the WHERE clause and if there are more than one words that are tokenized as VAL tokens, they participate in AND expression, containing more than one SQL like operator for each word.

This algorithm is quite heuristic and tries not to deal with specific language syntax rules and sentence structure. The last makes parsing not quite successful in some cases but keeps the algorithm universal enough to work with other languages different than Bulgarian. For example, if we want the algorithm to be able to parse English language sentences, all we need to do is to integrate the implementation with a morphological analyzer for English Language and provide a dictionary for synonyms of table and column names for the English Language. The current implementation also faces some limitations regarding more complex relations between columns and values participating in where clause. It does not discover when to use for example OR or NOT relations in query building. This issue will be addressed in future papers.

```
string selectQueriesTextOutput = "";
foreach (SelectQuery selQ1 in selects)
     {
      SelectQuery selQ = selQ1;
      selectQueriesTextOutput += "SELECT ";
int i = -1;
      List<String> aa = new List<String>();
foreach (var kvp in selQ.columns)
      {
if(!aa.Contains(kvp.tableName + "." + kvp.key))
        {
         aa.Add(kvp.tableName + "." + kvp.key);
        }
      }
```

Figure 7. Code fragment of generation of SQL Select

3. Conclusion

It can be argued that the paper demonstrates some level of achievement of its goals by using very simple algorithms that can work with simple sentences and simple structure tree objects that are suitable for a Facebook virtual assistant, and they are a good compromise between the quality of the generated texts and the ability to recognize natural language queries and the speed with which the virtual assistant has to work with groups of trainees who are in the order of dozens.

More precise testing should be done and, if necessary, more complex linguistic algorithms and paradigms of object representation and metadata related to them, such as generation templates or text parsing rules, should be introduced.

The advantage of the present approach is that texts can be generated without such metadata available using heuristic algorithms. The combination of semantic databases as Elastic and SQL Query generation and search if the semantic search fails with less relevant search results ensures better search results, not necessarily limited only to course specific contents.

Future papers will include integration with more social networks and platforms, overcoming some cons of the SQL SELECT Query generation algorithm like the discovery of relations between discovered tags by incorporating more strategies like template discovery and translation based on regular expressions and formal grammars.

Acknowledgements

The paper is supported within the National Scientific Program "Young scientists and Post-doctoral students" in accordance with Appendix No. 11 of Council of Ministers Decision No. 577 of 17 August 2018.

References

[1]. Sana, K., & Sheikh, T. B. (2015). A Study on the Role of Facebook in E-Learning. *International Journal of Education and Management Engineering*, *5*(5), 1-11.

[2]. Towner, T. L., & Muñoz, C. L. (2011). Facebook and education: a classroom connection?. In *Educating educators with social media*. Emerald Group Publishing Limited.

[3]. Al-Kouz, A., Luca, E. W. D., & Albayrak, S. (2011). Latent semantic social graph model for expert discovery in facebook. In *11th International Conference on Innovative Internet Community Systems (I2CS 2011)*. Gesellschaft für Informatik eV.

[4]. Selwyn, N. (2009). Faceworking: exploring students' education-related use of Facebook. *Learning, media and technology*, *34*(2), 157-174.

[5]. Kusumawati, A. (2014). Social Networking Sites for University Search and Selection. *Journal of Education and Practice*, *5*(25), 130-142.

[6]. Balakrishnan, V., & Gan, C. L. (2016). Students' learning styles and their effects on the use of social media technology for learning. *Telematics and Informatics*, *33*(3), 808-821.

[7]. Kazanidis, I., Pellas, N., Fotaris, P., & Tsinakos, A. (2018). Facebook and Moodle integration into instructional media design courses: A comparative analysis of students' learning experiences using the Community of Inquiry (CoI) model. *International Journal of Human–Computer Interaction*, *34*(10), 932-942.

[8]. Elastic.(2021). Retrieved from: https://www.elastic.co/ [accessed: 10 March 2021].

[9]. Venkataramani, V., Amsden, Z., Bronson, N., Cabrera III, G., Chakka, P., Dimov, P., ... & Puzar, L. (2012, May). Tao: how facebook serves the social graph. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (pp. 791-792).
DOI: https://doi.org/10.1145/2213836.2213957

[10]. De Carolis, B., Redavid, D., & Bruno, A. (2015). A Sentiment Polarity Analyser based on a Lexical-Probabilistic Approach, Proceedings of 1st AI*IA Workshop on Intelligent Techniques At Libraries and Archives co-located with XIV *Conference of the Italian Association for Artificial Intelligence*, IT@LIA@AI*IA 2015.

[11]. WordsApi: An API for English Language.(2021). Retrieved from: https://www.wordsapi.com/ ,
[accessed: 10 March 2021].