Computer Systems and Communication Technology
W. Zheng (Ed.)

doi:10.3233/ATDE240007

The Use of Python, Owlready, Sparql in Processing the Words Ontological Model of Public Political Discourse

Ayaulym SAIRANBEKOVA[a,1], Gulmira BEKMANOVA [b], Assel OMARBEKOVA [c], Assel MUKANOVA [d] and Altanbek ZULKHAZHAV [e]
[a,b,c,e] *L. N. Gumilyov Eurasian National University*
[d] *Astana International University*
ORCiD ID: Ayaulym SAIRANBEKOVA https://orcid.org/0000-0002-0814-1532,
Gulmira BEKMANOVA https://orcid.org/0000-0001-8554-7627,
Assel OMARBEKOVA https://orcid.org/0000-0002-9272-8829,
Assel MUKANOVA https://orcid.org/0000-0002-8964-3891,
Altanbek ZULKHAZHAV https://orcid.org/0000-0002-4491-3253

Abstract. The article describes a technology processing ontological model of words in public political discourse. The research task is developing an information question-answering system of political discourse in Kazakh language. The Python programming language, Sparql data-query language, and Owlready module are used to develop the system.

Keywords. Artificial intelligence; knowledge base; discourse; ontology; formalization, python, owlready, sparql, OWL.

1. Introduction

This work is carried out within the framework of the project BR11765535 “Development of Scientific and Linguistic Foundations and IT Resources to Expand the Functions and Improve the Culture of the Kazakh Language”.

The authors of the article are conducting work related to the processing of sentiment in socio-political discourse and public speeches [1], knowledge acquisition based on ontologies in natural language processing [2], and ontology process using python and sparql [3].

Nowadays, many people express their civic and political positions through the social networks.

The processors have already been made that work in this direction based on artificial intelligence methods. In Kazakhstan, scientists from L.N. Gumilyov Eurasian National University [4, 5], Al-Farabi Kazakh National University [6], and the International University of Information Technologies (International Information Technologies University) [7] researched the sentiment analyzer of official and unofficial information sources based on the texts sentiments analysis.

[1] Corresponding Author: Ayaulym Sairanbekova, sairanbekova98@gmail.com

An academic ontology is a specification of concepts, their attributes, and relationships in a certain subject area [1]. Ontologies allow to perform logical inference to get new information by reasoning and link it together with different pieces of knowledge from the ontology [8]. The Protégé editor is mainly used to create, maintain, and evaluate ontologies. However, it is not enough to develop the interface of its applications [9]. The data-query language SPARQL is mainly used for ontology programming interfaces; we used the OWLAPI module in our work.

The module was used for making a tool processing biomedical ontologies [10], a data system connecting different bulletin boards using an ontology coordination approach [11], explaining contingencies in production planning [12], a semantic web infrastructure analyzing dataflow [13,14].

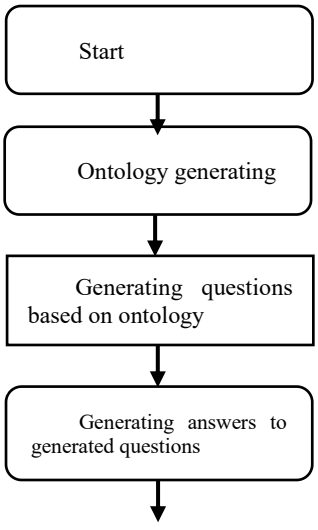
2. Ontology Data Processing Algorithm

The question-and-answering system on public political discourse is available at the link https://kazlangres.enu.kz/#/answer/question/3.

This system was developed based on the ontology of political discourse (https://webprotege.stanford.edu/#projects/16410306-5223-4a1b-85bf-61fb3bd3a5e1/edit/Classes?selection=Class(owl:Thing)).

System operation algorithm:

1. database of questions is formed on ontology base,
2. all questions are stored in the database,
3. all answers to generated questions are stored in the database,
4. user asks a question,
5. the question text is being checked for the presence of an entity,
6. if the question matches a keyword in the ontology, then all answers associated with this entity are returned.
7. if the question does not match the keyword in the ontology, the question is checked with the generated questions,
8. if the question matches the generated questions, then all similar questions and their answers are returned.



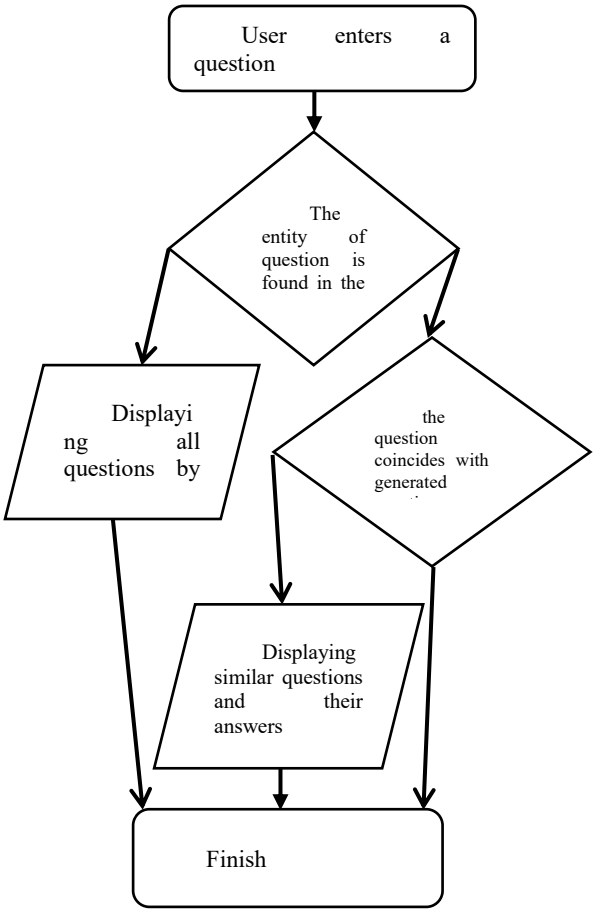


Figure 1. System operation algorithm.

The procedures are described below.

The procedure for generating questions based on ontology:

```
questions = [
# Дегеніміз не деген сұрақтар формасы
{
   'question': random.choice(["{} дегеніміз не?#0", "{} деген не?#0"]),
   'query': '''
      SELECT ?olabel ?label
         WHERE { ?subject rdfs:subClassOf ?object .
      ?subject rdfs:label ?label .
      ?object rdfs:label ?olabel
      FILTER(
         (REGEX(STR(?label), """ + question.lower() + """) || REGEX(STR(?label),
""" + question + """) || REGEX(STR(?label), ''' + question.capitalize() + '''))
         )
```

```
            }
            order by   ?label
        "",
        'answer': "{} дегеніміз - {}#0",
    },

    {
        'question': random.choice([source +" бойынша {} анықтамасы#2", "{} анықтамасы#2", "{} анықтамасын айт#2"]),
        'query': '''
            SELECT ?object ?label
                WHERE { ?subject kazont:definition ?object .
            ?subject rdfs:label ?label .
            FILTER(
                (REGEX(STR(?label), "''' + question.lower() + '") || REGEX(STR(?label), "''' + question + '") || REGEX(STR(?label) ,"' + question.capitalize() + '"))
                )

            }
        '',
        'answer': random.choice(["{} анықтамасы: {}#2", "{} дегеніміз: {}#0"]),
    },
    {
        'question': random.choice(["{} үшін мысал келтір#2", source + " бойынша {} үшін қандай мысалдар келтірілген#2", "{} мысалдары#2"]),
        'query': '''
        SELECT ?subject ?label
        WHERE
        {
            {
                ?subject rdf:type ?object .
                ?object rdfs:label ?label .
                FILTER(
                (REGEX(STR(?label) ,"''' + question + '") || REGEX(STR(?label), "' + question.capitalize() + '"))
                    && (LANG(?label) = "" || LANG(?label) = "kz" )
                )
            }
            union
            {
                ?object kazont:example ?subject .
                ?object rdfs:label ?label .
                FILTER(
                (REGEX(STR(?label), "''' + question + '") || REGEX(STR(?label), "' + question.capitalize() + '"))
                && (LANG(?label) = "" || LANG(?label) = "kz" )
                )

            }
```

```
        }
        '",
        'answer': random.choice(["{} үшін мысалдар: {}#0", "{} мысалдары: {}#2"]),

    },

    {
        'question': random.choice(["{} түрлері қандай?#2", "{} " + source + " бойынша қандай түрлерге бөлінеді#0"]),
        'query': '"
            SELECT ?slabel ?label
                WHERE { ?subject rdfs:subClassOf ?object .
            ?object rdfs:label ?label .
            ?subject rdfs:label ?slabel .
            FILTER((REGEX(STR(?label), "" + question + '") || REGEX(STR(?label), '" + question.capitalize() + '")) && (LANG(?slabel) = "" || LANG(?slabel) = "kz" ) )
            }
        '",
        'answer': random.choice(["{} түрлері: {}#2", "{} келесідей бөлінеді: {}#0"]),
    }

  ]
```

Question update procedure:

```
def update_question(self, question):
    try:
    # Database Connection
        if self.cursor.closed:
            self._delete_instance()
            self = self.get_instance()
        cursor = self.cursor
        # Checking if a question exists in the database
        cursor.execute("SELECT id FROM public.question WHERE question = %s", (question['question'],))
        question_id = cursor.fetchone()

        if not question_id:
            # If there is no question, insert it into the 'question' table
            print(cursor.execute("INSERT INTO public.question (question) VALUES (%s) RETURNING id", (question['question'],)))
            question_id = cursor.fetchone()['id']

        # Inserting question-related answers into the 'answer' table for answer_data in question['answers']:
            cursor.execute("INSERT INTO public.answer (answer, question_id) VALUES (%s, %s)", (answer_data['answer'], question_id))

        # Save change
        self.connection.commit()
        # Closing cursor
```

```
        cursor.close()
        return True  # Inserted successfully
    except Exception as e:
        print(f" Error inserting into database: {str(e)}")
        self.connection.rollback()  # Transaction rollback in case of error
return False  # An error occurred while inserting
```

`pg_trgm` for postgresql is used for searching questions from the database. `pg_trgm` is an extension to PostgreSQL providing support for operators and functions for performing operations on the lines using trigrams. Trigrams are groups of three sequential character in a line. This extension provides powerful tools for performing various operations with the text, such as detecting similar lines, performing similar trigram counting, and much more.

Some of the key functions and operators provided by `pg_trgm` are:

1. **similarity(text1 text, text2 text)**: This function compares two texts and returns similarity value between them. It uses trigrams to calculate similarity, and the result is represented as a floating-point number ranging from 0 to 1, where 0 indicates no similarity at all and 1 indicates a complete match.
2. **“%” и “<%”**: These operators can be used to perform comparisons of text lines using trigrams. "%" returns "true" if two lines have similarity more than 0.3, while `<%` returns `true' if the similarity is more than 0.6.
3. **word_similarity(text1 text, text2 text)**: This function works similar to `similarity()`, but it only considers words, not characters.
4. **show_trgm(text text)**: This function returns an array of trigrams for the specified text.

Using “pg_trgm” can be useful for implementing full-text search, as well as for searching for similar texts and auto-completion of text queries. It carries efficient searching even in the cases where texts can contain typos or spelling variations.

```
def get_question_similarity(self, question):
    try:
        # Database Connection
        if self.cursor.closed:
            self._delete_instance()
            self = self.get_instance()
        cursor = self.cursor

        # Request to get a question by its ID
        cursor.execute('''
        SELECT id, question FROM
            (SELECT id, question, similarity(question, %s) AS sim
        FROM
            question
        WHERE
            similarity(question, %s) > 0.3) AS quest
        ORDER BY
            sim DESC
        LIMIT 10''', (question,question))
        question_datas = cursor.fetchall()
        result = []
```

```
        for question_data in question_datas:
            question_id = question_data['id']
            # Runnnig a query to get answers related to the question
cursor.execute("SELECT * FROM public.answer WHERE question_id = %s",
(question_id,))
            answers_data = cursor.fetchall()

            # Dictionary formation based on the received data          question_dict
= {
                'question': question_data['question'],
                'answers': [{'answer': answer_data['answer']} for answer_data in
answers_data]
            }
            result.append(question_dict)

          # Closing cursor
          cursor.close()
          return result
      except Exception as e:
          print(f" Error while retrieving data from database:
{str(e)}")
          return None
```

If a question is not found in the question database, then the search procedure for a similar question is performed (a function for calculating the Levenshtein distance between two lines):

```
levenshteinDistance(a, b) {
  if (a.length === 0) return b.length;
  if (b.length === 0) return a.length;

  const matrix = [];

  // Fill in the matrix
  for (let i = 0; i <= b.length; i++) {
    matrix[i] = [i];
  }

  for (let j = 0; j <= a.length; j++) {
    matrix[0][j] = j;
  }

  for (let i = 1; i <= b.length; i++) {
    for (let j = 1; j <= a.length; j++) {
      const cost = a.charAt(j - 1) === b.charAt(i - 1) ? 0 : 1;
      matrix[i][j] = Math.min(
        matrix[i - 1][j] + 1,
        matrix[i][j - 1] + 1,
        matrix[i - 1][j - 1] + cost
      );
    }
```

```
    }

    return matrix[b.length][a.length];
  },
this.questions.sort((a,b)=>
        this.levenshteinDistance(a.question, this.question) -
this.levenshteinDistance(b.question, this.question)
        );
```

3. Conclusion

The practical significance of the study is focus on solving the applied problem of developing a question-answering system based on the ontology of political discourse. The operation algorithm of the question-answering system, the knowledge base formation of questions and answers based on the knowledge base are described, and the Python program codes for system use are described in detail. The obtain results can certainly be applied developing applications in various subject areas basing on ontologies.

Summing up, we can draw conclusions about the relevance and development trends of this direction. To date, a lot of work is being done around the world in the direction of artificial intelligence and intelligent systems based on knowledge. The research carried out in this work gives new ideas and opens up new possibilities of intelligent systems based on bases of the knowledge for further work and study.

References

[1] G. Bekmanova, B. Yergesh, A. Ukenova, A. Omarbekova, A. Mukanova, Y. Ongarbayev. Sentiment Processing of Socio-political Discourse and Public Speeches. Lecture Notes in Computer Science. Том 14108, pp. 191 – 205, ICCSA 2023, doi:10.1007/978-3-031-37117-2_15

[2] G.Yelibayeva, A. Sharipbay, G. Bekmanova, A. Omarbekova. Ontology-based extraction of Kazakh language word combinations in natural language processing. ACM International Conference Proceeding Series. pp. 58 - 595, DATA 2021. doi:10.1145/3460620.3460631.

[3] A. Omarbekova, A. Sharipbay, A. Barlybaev. Generation of Test Questions from RDF Files Using PYTHON and SPARQL. Journal of Physics: Conference Series. Том 806, Выпуск 121, CCEAI 2017. doi: 10.1088/1742-6596/806/1/012009.

[4] B. Yergesh, L. Kenzhina. Analysis of the users' emotional state in social networks. ACM International Conference Proceeding Series. Article number 3492654. 7th International Conference on Engineering and MIS, ICEMIS 2021. 2021. Код 175544. ISBN 978-145039044-6. doi: 10.1145/3492547.3492654.

[5] A. Boranbayev, G. Shuitenov, S. Boranbayev. The Method of Analysis of Data from Social Networks Using Rapidminer. Advances in Intelligent Systems and Computing. Volume 1229 AISC, Pp. 667 – 673. 2020 Science and Information Conference, 2020. Kod 241959. ISSN 21945357. ISBN 978-303052245-2. doi: 10.1007/978-3-030-52246-9_49.

[6] D. Sultan, A. Suliman, A. Toktarova, B. Omarov, S. Mamikov, G. Beissenova. Cyberbullying detection and prevention: Data mining in social media. Proceedings of the Confluence 2021: 11th International Conference on Cloud Computing, Data Science and Engineering. Pp.338 – 342. Article number 9377077 11th International Conference on Cloud Computing, Data Science and Engineering, Confluence. 2021. Код 167955. ISBN 978-073813160-3. doi:10.1109/Confluence51648.2021.9377077.

[7] B. Gulnara, Z. Ilyas, Z. Gulnara. The development of a web application for the automatic analysis of the tonality of texts based on machine learning methods. International Conference on Control, Automation and Systems. Volume 2018. Article number 8571950. 18th International Conference on Control, Automation and Systems, ICCAS 2018. Kod 143670. ISSN 15987833. ISBN 978-899321515-1.

[8] N. Guarino, D. Oberle, S. Staab, Handbook on ontologies, Springer, 2009, Ch. What Is an Ontology?, pp. 1–17.
[9] A. Rector, M. Horridge, L. Iannone, N. Drummond, Use Cases for Building OWL Ontologies as Modules: Localizing, Ontology and Programming Interfaces & Extensions, in: 4th Int Workshop on Semantic Web enabled software engineering (SWESE-08), 2008.
[10] J.-B. Lamy, Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. Artificial Intelligence in Medicine. Vol. 80, pp. 11 – 28. 2017. doi: 10.1016/j.artmed.2017.07.002.
[11] R. Jain, N. Duhan. OntoJob Query Processor: An Ontology Driven Query Processing Method in Jobology Information System. Journal of Computer Science. Vol. 16, Выпуск 5, pp. 702 – 714. 2020. doi: 10.3844/JCSSP.2020.702.714.
[12] P. Schuller. A new OWLAPI interface for HEX-programs applied to explaining contingencies in production planning. CEUR Workshop Proceedings. Vol. 2659, pp. 25 – 31. 2020. NeHuAI 2020.
[13] E. Jajaga, L. Ahmedi. C-SWRL: A Unique Semantic Web Framework for Reasoning over Stream Data. International Journal of Semantic Computing. Vol. 11, Выпуск 3, pp. 391 – 409. 2017. doi: 10.1142/S1793351X17400165.
[14] E. Jajaga, L. Ahmedi. C-SWRL: A Unique Semantic Web Framework for Reasoning over Stream Data. International Journal of Semantic Computing. Vol. 11, Выпуск 3, pp. 391 – 409. 2017. DOI 10.1142/S1793351X17400165] [E. Jajaga, L. Ahmedi. C-SWRL: SWRL for Reasoning over Stream Data. ICSC 2017. Pp. 395 – 400. doi: 10.1109/ICSC.2017.64