A New Approach to Automatic Ontology Generation from the Natural Language Texts with Complex Inflection Structures in the Dialogue Systems Development

Anna A. Litvin [1], Vitalii Yu. Velychko [1], Vladislav V. Kaverinsky [2]

[1] The V.M. Glushkov Institute of Cybernetics of the NAS of Ukraine, Akademika Glushkov avenue, 40, Kyiv, 03187, Ukraine

[2] The I.M. Frantsevic Institute for Problems in Material Science of the NAS of Ukraine, Krzhyzhanovskiy street, Kyiv, 03142, Ukraine

Abstract

An integrated approach is created for the development of natural language dialogue systems driven by an ontological graph database. Ontology here has a defined regular structure that contains typed semantic relationships between concepts, as well as related contexts, which may also have a multilevel structure and additional typing. The ontology is created automatically due to the semantic analysis of a natural language using a specially developed original software, which is set up to work with inflected languages, in particular Ukrainian. The ontology description is serialized in OWL format. To work as part of the dialogue system, the ontology is transferred to the graph database Neo4j. The Cypher language is used for formal queries. The original phrases of the user are subject to a special method of semantic analysis, which determines the type of formal query to the database. The essence of the analysis is that the text of the user phrase goes through a series of checks. Based on their results, a set of basic templates for formal requests is determined, as well as additional constructions that are attached to the basic template. Some of the checks may also return the notion of substitution to certain specified positions of the formal query. Formal queries can return both contexts and lists of ontology concepts. In addition to concepts, queries can also return information about specific semantic predicates that connect them, which simplifies the synthesis of natural language responses. The synthesis of answers is based on special templates, the choice of which is directly related to the corresponding template of the formal query.

Keywords

ontology, Neo4j, Cypher query language, text analysis, automatic ontology generation, semantic analysis, natural language text synthesis, NLP, NLU

1. Introduction

Creating a dialogue system that can be "trained" using natural language texts without regular structure or prior markup is an important problem with a solution is highly desirable. Automation of the process will greatly help to work with a significant amount of information stored as text or collected over the World Wide Web. Such a system could help users to find answers to their questions in the form of the appropriate contexts extracted from the texts or even as conclusions drawn from semantic data obtained from the analyzed text. The current study is devoted to the development of such kind of a dialogue system. The main feature of the proposed method is the automatic building of the ontological graph through the semantic analysis of a natural language text. Another part of the

1 13th International Scientific and Practical Conference from Programming UkrPROG'2022, October 11-12, 2022, Kyiv, Ukraine
EMAIL: litvin_any@ukr.net (A.1); insamhlaithe@gmail.com (A.3)
ORCID: 0000-0002-5648-9074 (A.1); 0000-0002-7155-9202 (A.2); 0000-0002-6940-579X (A.3)

system is the natural language user interface for the graph database, which provides the conversion of user phrases into formal queries to the ontology. The system also includes a module for the synthesis of natural language responses based on the results of a formal request. It should be noted that the current study is primarily aimed at inflectional languages, which include East Slavic languages, in particular, Ukrainian (for which the examples of implementation of the developed method are given here).

Automatic creation of a database using natural language text in this case can be considered as a particular kind of machine learning. The core of the system is an ontology which is represented as a graph database dedicated to a specific topic. This ontology must have a predefined structure to make easier and more predictable its integration with programs. Nevertheless, the specific content of the ontology is not predetermined and depends on the information from the text submitted as the input data. Thus, the certain results (answers) given by the system and their subject area depend only on the texts used as material for its "learning".

The important notice is that the system proposed in this paper is designed to work primarily with the grammatically and orthographically correct text of scientific and technical style.

2. Analysis of modern achievements in the field of natural language processing methods for working with ontological knowledge bases

As it was mentioned in the introduction, the design and development of natural language dialogue systems is a complex task, which includes building a database and modules for interaction with it, a

semantic analyzer of natural language text, procedures for the answers forming, and providing content in the context of dialogue. The ontology creation is not the main subject considered in the present work. The main topic here is creation of formal queries and the formation of natural language responses, which mainly form a natural language interface of a graph database. More information on the ontology structure and methods of its automatic creation could be found in [1], the problem of staying inside the dialogue context is considered in [2]. Natural language dialogue systems, so-called chatbots, have a long history and a number of approaches. Below we are to consider some interesting examples of dialogue systems developed in recent years, in particular those that in one way or another use an ontology in their structure.

A good example of a natural language dialogue system is described in [3, 4]. Like most others, it deals with the English language and its structural features. The means of the analysis of the user's source phrase assumes that sentences in English have quite a regular structure that can be expressed through a rather restricted set of templates. The constant part of such a template corresponds to its semantic type ("intention"), and the variable parts show the places in the phrase, which concepts are to be extracted from. These placeholders are specified according to the certain expected "intentions" of the extracted concepts. For example, there is a template: "Show me {@M} by {@D} for {@V}.". The curly brackets here mark the places where the concepts are expected. The markers in the placeholders here show the following: @M corresponds to the main requested concept, @D is the selection category for concepts such as @M, @V is the filter parameter. For example, there is a phrase "Show me admits by major diagnostic category for 2017", which fully satisfies the above template. The main concept that the user asks to show is admitting (in this case it is "number of hospitalizations"), the category of selection and sorting is a major diagnostic category (basic diagnostic categories), the filtering parameter is "2017" in the pattern of which year concept could be guessed. The structure and the constant part of the query determine its "intention". For each "intention" there exist a certain package of queries to databases and instructions on how to visualize and present their results in the user interface. Databases containing basic information in this case are mostly relational. However, the system also contains an ontology, which serves to structure the categorization of types and measurements of data stored in the main database. The "intentions" and concepts derived from the source phrase of the user are compared with the ontology to determine the closest to the requested dimensions and categories from those available in the databases. That is, ontology in this case plays

quite a secondary role. The ontology is created automatically based on a relational data model. The authors note the ability of the system to stay in the context of dialogue. Their approach is mostly focused on pronouns substitution. If the variables of the analysis template appear as pronouns or merely empty, then the program uses the relevant data from the last of the previous queries. In the case when there is no information in previous queries, the default values are substituted. These default values are formed based on the most common requests gathered during the system usage. Currently, the system does not contain automated learning, although the authors have declared the possibility of its development in the future.

The main features of the system from [3, 4] can be briefly described as follows: works only with English and adapted to its features; analysis of output phrases is based on patterns; not capable of automatic learning; the main data is stored in a relational database, the ontology exists, but plays a

supporting role, and is created automatically based on a relational database; has a set of specified "intentions" and related schemes of information presentation (in the form of tables, diagrams and graphs); does not generate natural language responses; implements methods for staying in the context of dialogue.

Dialogue systems which use ontology as the main knowledge base are usually merely natural language interfaces of a graph database. As a language for formal queries SPARQL is often used. The main task appearing during their development is the conversion of a user's natural language request into a formal one. Below are presented some examples of such converters that have been being developed in recent years.

One of the examples of automated conversion of natural language queries into SPARQL frameworks is the PAROT [5]. It uses an approach that generates the most probable RDF triple based on the user's request. The triplet is then checked by a special module containing a dependency analyzer to process user requests to RDF triplets. Then the RDF triplets obtained in this way are to be transformed into ontological triplets using a special thesaurus. The generated ontological triplets are used to build a SPARQL query, which is used for answers obtained from the ontology. Testing of the PAROT framework by the authors [5] showed that for simple questions it shows an accuracy of about 81 – 82 %, for complex ones – about 43 – 56%, and for a specific thematic data set (geography) accuracy raised to 88%.

Another example of natural language conversion into SPARQL techniques implementation is FREyA [6]. It is available on the GitHub [7]. FREyA offers an interactive native language interface for ontology queries. It uses parsing combined with ontology-based search to interpret questions and, if necessary, engages the user. User selection is used to train the system, which improves the accuracy of its operation. This system is currently implemented for English only. In [7] some examples are given which illustrate how questions in natural language could be converted to SPARQL using FREyA. It should be noticed that the FREyA configuration can be tuned for a certain ontology structure.

Also, it seems to be worth reminding the LODQA (Linked Open Data Question Answering) system presented in [8]. It accepts a query in natural language as the input and returns SPARQL queries along with the corresponding responses as a result. The system consists of several modules. The first module processes the request in natural language. It is responsible for parsing and creating a graphical representation of the query, called a pseudographic template. The pseudographic template contains nodes and links. The nodes usually correspond to the basic name groups and the links to the dependencies between them. In addition, the pseudographic template indicates which node of the ontological graph is the focus of the query, i.e., what the user is going to get as a response to the query. A pseudographic template is a search graph template of a target graph of RDF subgraphs that match it. However, it is called a pseudographic template because it is not yet based on the target data set. No sooner the first module has generated a pseudographic template from the given natural language query, than the next module is activated, which is responsible for finding URIs and nodes values in the pseudographic template. URIs and values must be present in the target data set. To normalize, each node of the pseudographic template is associated with the URI of the dataset. The concept in natural language could be normalized (reduced to the initial grammatical form) in more than one way because of possible ambiguity. Therefore, more than one template could be obtained

from one pseudographic template. The third module for the created pseudographic template performs a search in the target data set for the relevant parts, taking into account possible changes that may occur in the data set. To account for the structural differences between the bound pseudographic template and the actual structure of the target data set, this module attempts to generate SPARQL queries for all possible structural variations. SPARQL queries are then sent to the target endpoint, where responses are to be obtained and then sent to the user. These query arguments can be a primitive type, such as S, N or NP, or complex, such as S / NP, or NP / N. A slash means that the argument should be displayed on the right, and a backslash means that the argument should be displayed on the left. The system uses the following notation of parts of speech, for example, NN – noun, DT – definition (adjective), VB – verb. To facilitate the identification of RDF triplets, the words in the sentence are lemmatized and assigned with the appropriate grammatical characteristics. The considered LODQA system is focused on working only with English. Detailed features of its functioning in [8] are not given, limited to a general description and analysis of examples of work.

Although the development of dialogue systems, as well as machine processing and "understanding" of natural language text, are mostly carried out for the English language, they are not limited to it. For example, in [9] is presented a dialogue system for the German language. This article seems to be interesting because it also involves ontology. In this case, the ontology acts as a dialogue manager (OntoDM), which maintains the state of the conversation. Ontology is also used here as a knowledge base. These roles are combined. Subject area knowledge is used to track objects of interest, i.e., ontology nodes (classes) that are products and services represented in the ontological knowledge base. In this way, there was introduced the ability of the conversation's history memory. Also, much of the article [9] is devoted to the peculiarities of linguistic problems of German language processing. By the time of publishing [9], the research work was still proceeding and the quality assessing criteria for the system was not yet obtained. The work [10] is an example of developing a dialogue system for the Korean language, which is fundamentally different from the European type.

One of the most promising graph database management systems (DBMS) is Neo4j [11], which provides fairly high performance and scalability, and is suitable for working with large amounts of data. It is also currently one of the most popular graph DBMSs. The language of formal queries adopted in Neo4j is Cypher. It has a wide range of capabilities, is quite flexible and open for extra functionality through plug-ins, for instance, for the implementation of typical algorithms on graphs. However, at present, unlike SPARQL, there are not many developments to convert natural language queries into formal queries on Cypher. Among the few examples could be considered the works [12, 13]. The system proposed in [13] is quite primitive. Requests must have a predefined structure. This approach is close to that presented in [3]: a set of sentence templates in natural language, where some fragments are replaced by special notation, as places from which the concepts are to be extracted for substitution into a query template. Each such template sentence corresponds to a specific query pattern on Cypher. The described approach has its advantages and disadvantages. The main advantage is its simplicity. And the main disadvantage is that a real dialogue system requires a large number of such sentences-templates, which include all possible options for asking. Moreover, this approach is justified for languages with a regular sentence structure, such as English, where fewer phrase patterns are needed. Inflective languages, such as Ukrainian, have a complex sentence structure with quite a free word order. This fact significantly increases the number of required templates number.

Thus, the main purpose of this research was to develop a natural language dialogue system based on ontology, which is created automatically through semantic analysis of natural language text, taking into account the peculiarities of inflective languages, in particular, Ukrainian, and uses Neo4j and Cypher query language to work with its knowledge base.

Below we will consider each of the three main parts of the system: automatic creation of the ontology using natural language text, natural language interface of the graph database and synthesis of answers in natural language using the results of the formal query.

3. Analysis of the user's input phrase for the formal queries to the ontology creation

A detailed description of the ontology automatic creation technique based on a natural language text is given in [1]. Let us consider the ontology structure itself in the terms of OWL.

The ontology has the following root classes:

 Action – actions expressed by verbs;

- Adjective – adjectives and participles;
- Adverb – adverbs and gerunds;
- Name – proper names;
- Number – numbers (uncertain also) and digit symbols;
- Preposition – prepositions;
- Term – nouns and nouns groups. Has a hierarchical structure from more common (from one word) to more certain terms;
- Negation – negative particles;
- UndefinedEntities – all the entities from the text that the class doesn't suitable for any of the ones listed above;
- PhraseType – types of the linked word groups. It has two child classes: MainNarration (the main part of the sentence) and SubordinatePhrase. These classes do not have descendants but are used as "Domain" values for the ontology properties responsible for the groups' characterization. SubordinatePhraseType – is used for subordinate phrase classification. For the moment it has the following subclasses: movement_in, actor, Participial, AdverbialPhrase, movement_out, goal, place, consequence, object, subject, cause, condition, and instrument. The listed subclasses do not have descendants but are used as "Range" for the properties that have SubordinatePhrase as their "Domain" value.
- SentTypes – sentences typing. It has the following subclasses: Narration, Interrogative та Imperative. These classes do not have descendants but are used as "Range" for the properties responsible for the sentences' characterization.
- The properties of the ontology are devised in the following three root groups:
- WordsLink – used for single entities linking; Groups – linked word groups; SentenceGroups – sentences.

The "WordsLink" property has descendants that match semantic types. In a more primitive version of the ontology, the descendants are merely the semantic types themselves but only those that have been found in parsed text. For example, "action addressing", "object entry", "quality change", "tool", "quantity", "adjacent localization", "destination", "separation", "object-action", "transfer", "compatibility", etc. In a more complicated version of the ontology, the descendants of "WordsLink" have an additional structure with a hierarchy. The scheme of the "WordsLink" descendants structure is given as a tree in Figure 1.

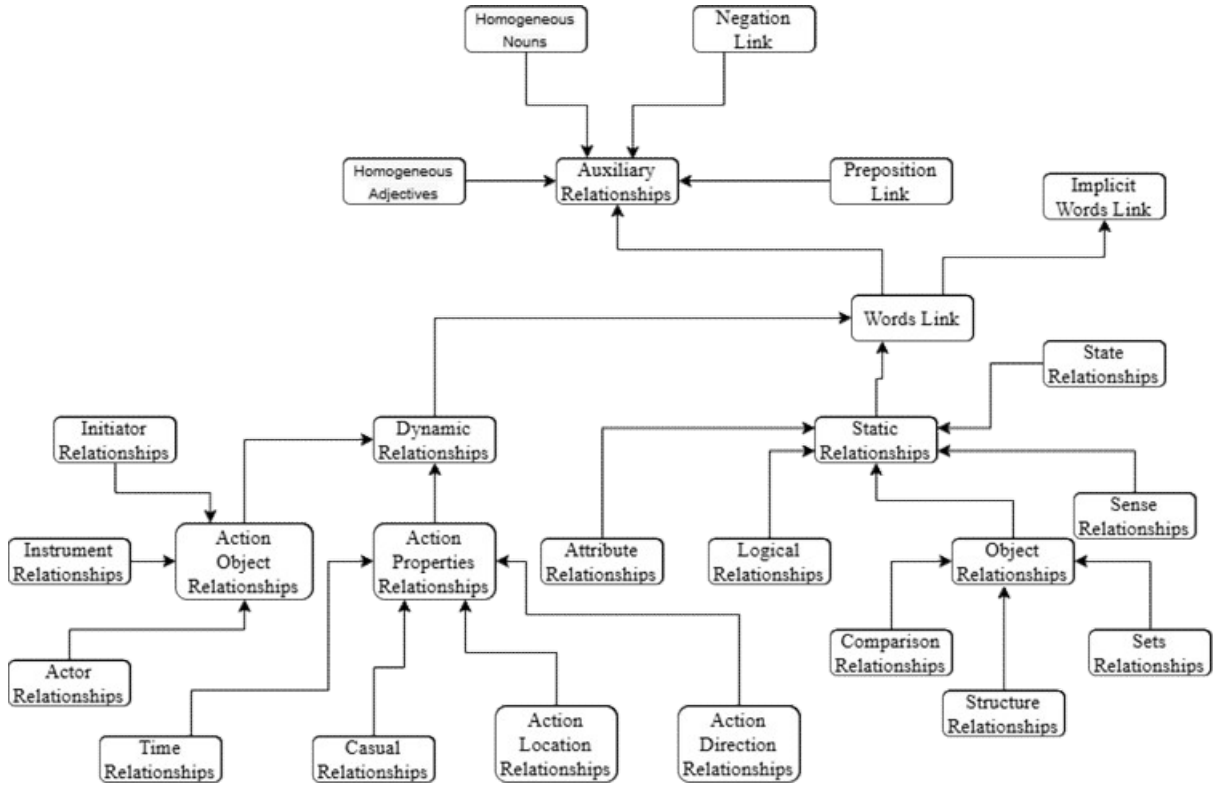

Figure 1: Scheme of “WordsLink” properties higher-level hierarchy in the ontology

The given here structure covers only the higher level of the semantic relationships hierarchy typing that remains the same for all built ontologies. The presence of lower-level entities depends on their presence in the considered text. They also could be hierarchy structured. For the moment the developed system operates with about 80 possible final semantic categories and this is not the limit.

The final descendants of the "WordsLink" property correspond to the specific types of connections between certain concepts. Each of them occurs only once in the ontology, even if it could be found several times in the considered text. "Domain" of the such property refers to the main concept of the linked pair, and "Range" to the dependent one. In addition, these properties are also heirs of the groups where the pair linked in this way is observed.

The “Groups” property characterizes groups of linked words. The descendant properties of Groups correspond to certain groups. Sub-properties of the groups are the above-mentioned properties, which show the connections between the concepts. The sub-properties of “SentenceGroups” correspond to sentences. As a label parameter, they contain the full text of the sentence (context). Their descendants are properties of the “Groups” type that correspond to the groups in the given sentence.

Neo4J DBMS could be used to work with the ontology of the described type. For this purpose, an OWL file is to be loaded to it using the "Neosemantics" plug-in. In this case, classes and properties become the graph nodes of the corresponding type which are "Class" and "Relationship". Relationships between the nodes can have the following types: SCO – a subclass of; SPO – subproperty of; DOMAIN; RANGE. The Cypher language is used for the queries.

Building an ontological graph based on natural language text is perhaps the most important part of a system that is responsible for collecting and structuring information. The construction of a semantically structured database requires semantic analysis of the considered text. Thus, an important part of the study was the development of its methodology, adapted for the East Slavic languages, which are of inflectional type. The important peculiarity of these languages is that words connection appears mainly mostly through a combination of certain flections (variable word endings). Behind these variations of word forms, which belong to the relevant parts of speech and the combination of concepts with prepositions, there lies a huge amount of semantic information. Moreover, there are other factors, such as word order. The order of words in inflectional languages is not very strict and

might be considered quite a secondary factor. Nevertheless, the words even in such types of languages do not go completely randomly. Moreover, in some cases, it may even become even a defining feature.

4. Analysis of the user's input phrase for the formal queries to the ontology creation

As proposed in our previous researches [14, 15, 16] tree-based method of the query template determined through the analysis of words sequence is quite demanding for the effort and time needed for such a tree development. At the same time in inflective languages, the word order is a less significant factor. A more valuable one is just presence of the certain words in specific forms. Thus, it seems that often enough would merely testify the considered phrase through several criteria. Grounding on the test results it might be possible not only to determine the most appropriate formal query template or the group of such templates but also to select the input entities for them. In the simplest test version of the system, which exists now, there are the 4 following main checks:

1 – question word – 6 lists + absence of such word. The result is the number of the sufficient list from 1 to 6 or 0 if there is no question word in the sentence.

2 – the presence of a word from given lists (most of them are specific verbs) – 6 lists + absence of words from all of the lists. The result is the number of the sufficient list from 1 to 6, of 0 – if there are no such words in the sentence.

3 – the presence of a noun in the nominative case except words from the check (2) if any. The result may be 1 – such a word exist (+ the word itself) or 0 – there is no such word. Several entities could be selected.

4 – the presence of a verb, except ones from lists in the check (2) if any. The result may be 1 – such a word exist (+ the word itself) or 0 – there is no such word. Several entities could be selected.

Even this brief test has quite enough options for its results that make it possible to have a number of templates or various types of templates.

Then an additional test is to be performed. Its procedure is as follows: adjectives linked to the word from clause (3) that must be close to it and fit it with number and gender; nouns in indirect cases (they form the base of the additional circumstances) and adjectives linked to them; and the last but not the lease is the check of presence or absence of negation predicates. An additional test is needed for the modifier templates adding.

The template is stored as XML files of a special structure. Here is an example of one of such (the simplest ones) templates:

```
<template>
    <verbose_name>Common information</verbose_name>
    <id>1</id>
    <type>base</type>
    <variables>
        <variable>
            <name>INPUT_VALUE_1</name>
            <destination>input</destination>
        </variable>
        <variable>
            <name>CONTEXT</name>
            <destination>output</destination>
        </variable>
    </variables>
    <match>
        (inp:Class)-[]-(n:Relationship),
        (n:Relationship)-[]-(x:Class),
```

```
            (n)-[:SPO]->(rel_group),
            (rel_group)-[:SPO]->(rel_sent),
            (rel_sent)-[:SPO]-(sent_super)
        </match>        <where>
        inp.label = "INPUT_VALUE" and
        sent_super.name = "SentenceGroups"
        </where>
        <return>
            DISTINCT rel_sent.label as CONTEXT;
        </return>
    </template>
```

In the given example it is possible to explain the common structure of the query template. The XML-template chapters <match>, <where>, and <return> correspond to certain sections of a Cypher query [11]. Some parts of the chapter's content are the template variables. The variables themselves are described in the chapter <variables>. Each of the variables is defined by its name and destination in the appropriate XML containers <name> and <destination>. The destination can have values "input" or "output". The input variables are to be substituted with the input parameter values and the output ones define the parameters that should be obtained as a result of the query execution. The container <id> is needed for the finding and identity of the template. Moreover, here is a tag <verbose_name> that helps to identify a template not only by machine but also by a human during the system development. Further, most of the query template examples here shall be given in a simplified mater – without XML tags. Tag <type> shows the type of a template – base or additional. Above is given an example of a base one. Let us consider the structure of the additional templates. Here is an example of one of them:

```
    <template>
        <verbose_name>Adjective linked to subject</verbose_name>
        <id>1</id>
        <type>additional</type>
        <variables>
            <variable>
                <name>INPUT_VALUE_ADJ</name>
                <destination>input</destination>
            </variable>
            <variable>
                <name>ADJ_PLUS</name>
                <destination>intermediate</destination>
            </variable>
            <variable>
                <name>INP_ADJ</name>
                <destination>intermediate</destination>
            </variable>
        </variables>
        <block_union>and</block_union>
        <next_item_union>or</next_item_union>
        <match>
            (inp:Class)-[]-(ADJ_PLUS:Relationship),
            (ADJ_PLUS:Relationship)-[]-(INP_ADJ:Class),
            (ADJ_PLUS)-[:SPO]->(rel_group)
        </match>
        <where>
```

```
            INP_ADJ.label = "INPUT_VALUE_ADJ"
        </where>
        <return></return>
    </template>
```

The template also has blocks <match>, <where> and <return>. However, their content of them is not independent but is to be added to the appropriate parts of a query formed through a base template. Some of the chapters in this case could be merely empty. The main feature of an additional template is the presence of the chapters <block_union> and <next_item_union>. Tag <block_union> shows how the block <where> must be united to the query formed by a base template. Tag <next_item_union> determines the union type for the repeated elements of the block <where> in a case when the appropriate variable is presented as a list (array). For instance, for the given above template, the variable INPUT_VALUE_ADJ could correspond to a number of adjectives linked with the object. The values of <block_union> and <next_item_union> could be "and" or "or". Also, the variables of the additional templates can have the third type of <destination> - "intermediate". Such variables neither take part in transferring values into the forming query nor in the results returning. They just needed to mark the template parts that are not to be duplicated during the part repeating. Instead, they are implemented with an order number, for example, ADJ_PLUS_1, ADJ_PLUS_2, ADJ_PLUS_3, …, etc.

Let us consider in more detail the structure of the formal queries and the manner of their formation. The presented structure of the ontology makes it possible to search for contexts or individual terms. Not only has it allowed just the presence of some entities in the context considering, but also their relationships according to a certain semantic category. In the presented scheme there is a base query template, aimed to obtain information of a certain type in a given form, and additional modifiers template that optionally adds the description of extra circumstances. Let us consider some types of queries. The already given above template is aimed at a context obtaining which includes a

specific term (word). However, the term must not only be presented in the context but form a link with others. This could guarantee that the term is “organically” implemented into the context.

Cypher queries are devised into three main parts: MATCH, WHERE, and RETURN. The MATCH block gives a linking pattern of the nodes in the oriented graph. In the WHERE part, the conditions are given that characterize the entities (nodes and relationships) from the MATCH case. The RETURN block shows what is to be returned as a result and with what name (alias). In the presented example there is a class marked by the variable "inp". In the WHERE block, a condition for it is added, which says that the "label" field of the node "inp" must be equal to a specific value (here and below INPUT_VALUE is the text of the input value). From the MATCH block, it is clear that “inp” is a node because of parentheses and it must have the type “Class”. It must be linked with another node "n" of type "Relationship", which corresponds to an ontology property from OWL. The link type is undefined in this case (square brackets are empty), and the direction of the link is also not specified. So, the node could be linked either as a "DOMAIN" or "RANGE". There is no need to specify the link direction in this case because it is known that such links always come from a property to a class. Also, it is given that this property must be linked with some class "x". Further given that the property linking these classes must have a relation to the sentence "rel_sent". The condition "sent_super.name = "SentenceGroups" guarantees that the "rel_sent" shall be a sentence. As a result the query is to be returned "rel_sent.label", which contents the sentence context with the alias "CONTEXT”.

Let us come to a more complicate example. Here we are to request the characteristics (properties) of an INPUT_VALUE entity included in the ontology. . Ми хочемо запросити відомі в онтології характеристики (визначення) об'єкта INPUT_VALUE. In other words, what the INPUT_VALUE is or could be. The query is as follows:

```
MATCH (inp:Class)-[]-(n:Relationship),
   (n:Relationship)-[]-(x:Class),
   (n)-[:SPO]->(prop_type_1),
   (n)-[:SPO]->(rel_group),
   (rel_group)-[:SPO]->(rel_sent),
```

```
    (rel_sent)-[:SPO]-(sent_super)
WHERE
    inp.name = "INPUT_VALUE" and
    (prop_type_1.label = "object property" or
prop_type_1.label = "action property" or
prop_type_1.label = "action separately" or
prop_type_1.label = "action level")      and
    sent_super.name = "SentenceGroups"
RETURN DISTINCT x.label as result,  rel_sent.label as context;
```

Compared to the previous example an extra statement is added to the MATCH block: (n)-[:SPO]>(prop_type_1). This gives information that the property "n" must be a child of "prop_type_1". Here the link direction is specified. In the WHERE block is given sufficient values of the "label" field of "prop_type_1". To make the query template more universal, as it is not known whether INPUT_VALUE is a noun or verb, some options are given for the possible "prop_type_1.label" value united with logical "OR". If the ontology has a semantic categories hierarchy, the construction could be simplified as follows:

```
MATCH (inp:Class)-[]-(n:Relationship),
    (n:Relationship)-[]-(x:Class),
    (n)-[:SPO]->(prop_type_1),
    (n)-[:SPO]->(rel_group),
    (rel_group)-[:SPO]->(rel_sent),
    (rel_sent)-[:SPO]-(sent_super),
    (prop_type_1)-[:SPO]->(prop_type_category)
WHERE
            inp.name   =   "INPUT_VALUE"   and
prop_type_category.label  =  "entities  properties"
and
    sent_super.name = "SentenceGroups"
RETURN DISTINCT x.label as result,  rel_sent.label as context;
```

As a result, the query "label" field of the "x" node is to be returned. That will be the characteristics of an "inp" object. Also, the contexts are requested to recognize the circumstances where the entity's property is mentioned.

In a close manner actions of an object could be requested. For this purpose, it is just needed to set another value for “prop_type_1.label” in WHERE block, namely: prop_type_1.label = "objectaction".

If there are several possible options of relationship in the query (prop_type_1.label) the result may include its certain value, which then helps in the answer synthesis. The next example illustrates a query of an object localization without its type concretization (“Where is INPUT_VALUE?”).

```
MATCH (inp:Class)-[]-(n:Relationship),
    (n:Relationship)-[]-(x:Class),
    (n)-[:SPO]->(prop_type_1),
    (n)-[:SPO]->(rel_group),
    (rel_group)-[:SPO]->(rel_sent),
    (rel_sent)-[:SPO]-(sent_super)
    (prop_type_1)-[:SPO]->(prop_type_category)
WHERE
     inp.label = " INPUT_VALUE " and
prop_type_category.label = "localization" and
sent_super.name = "SentenceGroups"
RETURN DISTINCT x.label as result, rel_sent.label as context,
          prop_type_1.label as predicate;
```

The main peculiarity here is the statement "prop_type_1.label as predicate" in the RETURN block. That makes it return the certain semantic type of the obtained result.

In some cases, instead of predicates lists of some entities (verbs, nouns, adjectives) could be included in a query. The peculiarity here is that conditions are given for the node of ontograph linked with “x”. Thus, the requested object not only must be linked with some term “x” through the specific relationship, but this term must be from a certain list. If the terms (or actions) are additionally classified in the ontology, the condition for the term will be merely being a descendant of a specific category.

A special mention should be made of modifier templates – fragments that could be added to the main query templates. Let us consider an example where the input parameter is not a single word, but a noun group. So, there are linked nouns and adjectives. To link to the input adjective concept there must be added the appropriate statements to the MATCH block:

```
(inp:Class)-[]-(adj_plus:Relationship),
(adj_plus:Relationship)-[]-(inp_adj_1:Class),
(adj_plus)-[:SPO]->(rel_group) and
```

in WHERE block:

```
and
inp_adj_1.label = "INPUT_VALUE_ADJ"
```

For the extra adjectives, the same blocks are to be added but with variables inp_adj_2, inp_adj_3 etc.

It is also possible to add a condition of a noun in indirect case presence through the following statements:

in MATCH block:

```
(inp_noun_1:Class)-[]-(noun_plus:Relationship),
(noun_plus)-[:SPO]->(rel_group) and
```

inWHERE block:

```
and
inp_noun_1.label = "INPUT_VALUE_NOUN"
```

Here in the example, there is a condition of the presence of one noun in the same group where the main concept is included. Nevertheless, conditions of adjectives presence linked with this noun also could be added:

in MATCH block:

```
(inp_noun_1:Class)-[]-(adj_plus_add:Relationship),
(adj_plus_add:Relationship)-[]-(inp_adj_add:Class),
(adj_plus_add)-[:SPO]->(rel_group) and
```

in WHERE block:

```
and
inp_adj_add.label = "INPUT_VALUE_ADJ_ADD"
```

Also in some cases, a negation predicate should be added to a query. For this purpose the following construction must be added to it: in MATCH block:

```
(neg:Class)-[]-(neg_rel:Relationship),
(neg_rel)-[:SPO]->(rel_group) and in
WHERE block:
and
        (neg.label = "no" or
neg.label = "not" or  neg.label =
"forbidden" or  neg.label =
"impossible" or  neg.label =
"cant" or  neg.label = "unable")
```

5. Synthesis of natural language answers based on the results of formal queries execution

The user interface of a dialogue system, which displays merely the results of a formal query, even being pretty designed, may not look so friendly, and sometimes could be even not quite understandable for a person. Therefore, the next important problem is the synthesis of natural language answers. Some principles of the approach of answers formation, based on information taken from the results of formal queries, and the analysis of the source phrase using templates-instructions are described in our previous research [17]. In general, during the system development, deciding on how the answer ought to appear in the user's interface is to be balanced between providing readymade contexts and text synthesis. For example, to provide some tables, graphical objects, or other media illustrating the answer, the best option is to use ready-made contexts containing links to the relevant files. In the current study, we omit representation and creation methods of graphical and tabular materials (charts, graphs, diagrams) based on the results of queries in the user interface, although this approach is quite desirable in certain types of systems and, as demonstrated by [4], may well be implemented. Contextual responses may be the best option if you need to provide detailed information. The synthesized answers provide greater ease of perception for more specific questions, in which a formal response is just a list of entities from the ontology. Here are provided with some examples of answers synthesizing instruction templates for some typical cases. These templates also give user contexts (sentences) that illustrate and confirm the statement. The templates below are presented in human-readable form (a kind of meta-language). In a software implementation, they are software entities (classes with methods) in the Python language that are attached to the system in a specific module file. An attempt was also made to add response templates in the form of XML descriptions, which, however, led to greater complexity and lower performance of the software.

Let us consider an example of a question about entity characteristics (properties). Here is the answer template:

Repeat for each result:
if INPUT_VALUE noun:
INPUT_VALUE + може бути + result (fit the gender)
+ context
is INPUT_VALUE verb:
INPUT_VALUE + можна + result
+ context

For the word’s morphological characteristics determination (part of speech, gender, case, etc.) and for word form fitting PyMorphy2 library methods are used [17]. In the simple example above part of the speech of INPUT_VALUE must be checked. It could be a noun or verb. If it is a noun, the "result" value must be fitted in gender with INPUT_VALUE.

Let us consider a more complicated example. Here the subject of the query is object localization. The certain localization predicate is not specified in the input query parameters but appears in its results. As it was mentioned above, a certain semantic predicate could be used in an answer synthesis.
Repeat for each result:
INPUT_VALUE + знаходиться +
if predicate = "localization in set":
+ серед + result (plural, genitive case)
+ context
if predicate = "localization near":
+ біля + result (genitive case)
+ context
if predicate = "objective localization":
+ на + result (locative case)
+ context

if predicate = " objective entering":
+ у + result (locative case)
+ context
if predicate = "localization between objects":
+ між + result (plural, instrumental case)
+ context
if predicate = "localization behind object":
+ за + result (instrumental case)
+ context
if predicate = "localization in front of object":
+ перед + result (instrumental case)
+ context
if predicate = "localization under object":
+ під + result (instrumental case)
+ context
if predicate = "localization above object":
+ над + result (instrumental case)
+ context
if predicate = "localization in object":
+ всередині + result (genitive case)
+ context

From the given example we can see that a certain type of semantic predicate (localization in this case) determines the appropriate preposition and case for the value of the "result" variable for the Ukrainian language.

6. Conclusions and further prospective

An approach and the corresponding software toolkit are developed for the construction of natural language dialogue systems based on automatically through a natural language text semantic analysis built ontology. Within the framework of the approach is developed an analysis technique of an initial user's phrase adapted for inflective languages, in particular Ukrainian, aimed to the formation on its basis of formal queries in the Cypher language. The essence of the method is a series of checks for the presence in the initial phrase of certain words and/or word forms. Depending on the set of test results, is selected the main query template (or group of such templates). Components from modifier templates are added to the main template (to its corresponding sections) as a result of additional checks, which make the appropriate clarifications and extensions to the query. Query variables are supplemented with concepts obtained when performing the appropriate checks. Several queries (packages) can be created based on one initial phrase. Also proposed here is an approach to the synthesis of natural language responses using query results and the values of source variables. The peculiarity of the approach is the usage of including specific values of semantic predicates obtained as a result of the query to the ontology, which allows the program more accurately and correctly formulate the answer by using the appropriate prepositions and word forms. Also, these answer templates provide instructions for fitting word forms of concepts-results with the original concepts.

Based on the proposed approach, an experimental dialogue system was developed, which proved to be workable. It can become a prototype for the development of new more powerful dialogue styles able to be "learned" using natural language texts provided in the form of documents, or as search results obtained from the Internet. A further perspective of the system development is to allow it to create more detailed classified ontologies and expand the number of checks and variants of their results. Accordingly, a large number of basic and additional formal query templates and corresponding response synthesis templates can be created.

7. References

[1] A. Litvin, V. Velychko, and V. Kaverinsky, A new approach to automatic ontology creation from the untagged text on the natural language of inflective type, Proceedings of the International conference on software engineering "Soft Engine 2022", NAU, Kyiv Ukraine, 2022, pp. 37 – 45.

[2] A. Litvin, V. Velychko, and V. Kaverinsky, Development of natural language dialogue software systems, Information Theories and Applications 28 (2021), pp. 233–270. doi: 10.54521/ijita28-03-p03

[3] A. Quamar, F. Ozcan, D. Miller, R. J. Moore, R. Niehus, J. Kreulen, Conversational BI: An Ontology-Driven Conversation System for Business Intelligence Applications, Proceedings of the VLDB Endowment 13, pp. 3369–3381. doi: 10.14778/3415478.3415557

[4] A. Quamar, C. Lei, D. Miller, F. Özcan1, J. Kreulen, R. J. Moore1, V. Efthymiou, OntologyBased Conversation System for Knowledge Bases, (2020), pp. 361–375. doi: 10.1145/3318464.3386139

[5] P. Ochieng, PAROT: Translating natural language to SPARQL, Expert Systems with Applications 5 (2020), pp. 1–16. doi: 10.1016/j.eswax.2020.100024

[6] D. Damljanovic, M. Agatonovic, H. Cunningham, FREyA: an interactive way of querying linked data using natural language, The Semantic Web: ESWC 2011 Workshops, 2011, pp. 125–138.

[7] GitHub: FREyA documentation. URL: https://github.com/nmvijay/freya

[8] S. Shaik, P. Kanakam, S.M. Hussain, and D. Suryanarayana, Transforming natural language query to SPARQL for semantic information retrieval, International Journal of Engineering Trends and Technology 7 (2016), pp. 347–350. doi: 10.14445/22315381/IJETT-V41P263

[9] Duygu Altinok, Ontology-Based Dialogue Management System for Banking and Finance Dialogue Systems, in: 1st Financial Narrative Processing Workshop., Japan, Miyazaki, 2018. doi: 10.48550/arXiv.1804.04838.

[10] H. Jung and W. Kim, Automated conversion from natural language query to SPARQL query, Journal of Intelligent Information Systems 55 (2020), pp. 501–520.

[11] A. Goel, Neo4J Cookbook, Birmingham: Pact Publishing Ltd, 2015.

[12] C. Sun, A Natural Language Interface for Querying Graph Databases, Master's thesis, USA: Massachusetts Institute of Technology, Cambridge, USA, 2018.

[13] GitHub: Convert English sentences to Cypher queries documentation. URL: https://github.com/gsssrao/english2cypher

[14] A. Litvin, V. Velychko, and V. Kaverinsky, Tree-based semantic analysis method for natural language phrase to formal query conversion, Radio Electronics, Computer Science, Control 3-4 (2021), pp. 65 – 72. doi: 10.15588/1607-3274-2021-2-11

[15] A. Litvin, V. Velychko, and V. Kaverinsky. Method of information obtaining from ontology on the basis of a natural language phrase analysis, in: CEUR Workshop Proceedings, CEUR-WS, Kyiv, Ukraine, 2020: pp. 323–330. URL: https://ceur-ws.org/Vol-2866/ceur_322_330_litvin_velichko.pdf.

[16] O.V. Palagin, K.S. Malakhov, V.Yu. Velychko, T.V. Semykopna, Hybrid e-rehabilitation services: SMART-system for remote support of rehabilitation activities and services, Int J Telerehab. (2022). doi:10.5195/ijt.2022.6480

[17] A. Litvin, V. Velychko, and V. Kaverinsky, Synthesis of chat-bot responses in the inflecting natural language based on the results of queries to ontology and analysis of the chat previous phrase, Information Theories and Applications 27 (2020), pp. 152–199. URL: http://www.foibg.com/ijita/vol27/ijita27-02-p03.pdf