*SACJ* 35(1) July 2023
Viewpoint

# Ontology-driven development of dialogue systems

Anna Litvin[a], Oleksandr Palagin[a], Vladislav Kaverinsky[b], Kyrylo Malakhov[a]

[a] Microprocessor Technology Lab, Glushkov Institute of Cybernetics of the National Academy of Sciences of Ukraine, Kyiv, Ukraine
[b] Frantsevic Institute for Problems in Material Science of the National Academy of Sciences of Ukraine, Kyiv, Ukraine

**ABSTRACT**
A new technique and its software implementation are presented to create a deeply semantically structured ontology using plain natural language text as input, without regular structure or any previous tagging and markup. The new approach is primarily aimed at highly inflectional languages, and is implemented for Ukrainian. The automatically created ontologies (in OWL) could be easily converted to other graph databases formats, such as Neo4j, and were successfully evaluated as valid ontologies using Protégé, RDFlib and Neo4j environments. An integrated approach is proposed for the development of natural language dialogue systems driven by the ontology-related graph database using the Cypher language for the formal queries. The original phrases 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's phrase goes through a series of checks. Based on their results, a set of basic templates for the formal requests are 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 engineering, Ontology learning, Knowledge management, Knowledge base, Relation semantics, Neo4j, Cypher, Dialogue systems

**Categories:** • **Information systems** ∼ **Information retrieval, Document representation, Ontologies**

**Email:**
Anna Litvin – litvin_any@ukr.net,
Oleksandr Palagin – palagin_a@ukr.net,
Vladislav Kaverinsky – insamhlaithe@gmail.com,
Kyrylo Malakhov – malakhovks@nas.gov.ua (CORRESPONDING)

**Article history:**
Received: 20 April 2023
Available online: 31 July 2023

## 1 INTRODUCTION

An essential element of every dialogue and reference system is its knowledge base (KB). The fundamental challenges in the design and development of information systems of this type are

Litvin, A., Palagin, O., Kaverinsky, V., Malakhov, K. (2023). Ontology-driven development of dialogue systems [Viewpoint]. *South African Computer Journal* 35(1), 37–62. https://doi.org/10.18489/sacj.v35i1.1233

*SACJ* is a publication of the South African Institute of Computer Scientists and Information Technologists.
ISSN 1015-7999 (print) ISSN 2313-7835 (online).

the structure of the KB, its fulfilment, development kits (devkits), maintenance, and interaction techniques.

Often, a deep scientific study and analysis is needed to comprehend the advantages and drawbacks of different approaches and find the optimal solution to some specific tasks set. The current research focuses on the questions of completely automatic creation of the KB (based on the natural language text with its structure and appropriate devkits) and method of subsequent interaction with the KB using a natural language interface.

Manual creation of a KB is a complicated, time-consuming, and complex process, which inspires finding techniques to automate the process. After automated creation, it might only take a minor manual correction and fine-tuning to make the KB more accurate. Furthermore, creating dialogue systems that can be *trained* by using natural language texts without any formal structure is highly desirable. Process automation will significantly assist in handling a large volume of data stored as plain text or collected via the Internet. This system could help users find answers to their questions as relevant contexts are extracted from texts. Also, users can get answers as inferences made from semantic data retrieved from the analysed text. The current state of computing and software techniques makes it possible to solve such a problem.

Ontologies are a well-known and proven method of organising semantically structured knowledge bases, which often comes with a particular type of graph database. Furthermore, given the specificity of the considered problem that involves dealing with natural language texts, which represent semantic structures by their nature, the use of ontologies appears to be an even more appropriate approach. Because one of the popular languages for ontology engineering is OWL (Antoniou & Plexousakis, 2016), which is a standard, it will appear as the foundation for ontology creation in this study. Moreover, conversion of OWL ontologies to other graph formats, for instance – the Neo4j graph database management system (GDMS) is possible.

Completely automatic ontology creation could be considered a specific type of machine learning which consists of semantic structuring of large amounts of textual information in a way that is acceptable for subsequent machine manipulation. This manipulation could be, for example, the automatic data collection from the Internet for a specific topic (knowledge domain). This direction in informatics is usually called *ontology learning* and has been actively developing in recent years. Automatic KB creation using natural language text is a specific type of machine learning.

Automatic ontology creation has been discussed in many studies (An & Park, 2018; Balakrishna & Srikanth, 2008; Elnagar et al., 2020). However, the development technique and structure of the ontology are highly dependent on the assigned purpose and knowledge domain. The original (native) natural language, its structure, and lexical and grammatical peculiarities also play a significant role in the approach to be developed. The implementations of most existing natural language processing (NLP), natural language understanding, natural language generation and natural language inference methods are devoted to the English language. This is because English has become the most widely accepted international language and the most useful in many branches of modern life. It should be noted that ontology learning approaches

https://doi.org/10.18489/sacj.v35i1.1233

are not sufficiently developed for Slavic inflectional languages, e.g. Ukrainian. Our previous work (Litvin et al., 2021) is devoted to this question. However, the approach considered is only acceptable for previously tagged texts or text sets with a predetermined regular structure (such as official documents, letters, etc.). The technique presented in this study makes it possible to process texts without previous tagging and markup or regular structure.

The current study is devoted to the development of a reference system with a natural language (NL) interface, of which the main feature is the automatic building of the ontology-related graph through the semantic analysis of natural language text. A further part of the system provides natural language interaction with the graph database. The key concept is to convert user phrases into formal queries to the computer ontology. Since the system uses Neo4J as its core graph database management system, the Cypher language is used for the queries. The system also includes a module for the natural language generation of responses based on the results of a formal request. The current study primarily focuses on highly inflectional languages, which include East Slavic languages such as Ukrainian.

The core of the system is the computer ontology (in OWL) which is represented as a graph database dedicated to specific domain knowledge. This ontology must have a predefined structure to make it easier and more predictable to integrate 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 outputs (responses) given by the system and its domain knowledge depend on the texts used as data for its learning.

It is important to note that the system proposed in this study intends to work primarily with the grammatically and orthographically correct text of scientific style and information technology domain knowledge.

## 2 STATE OF THE ART IN ONTOLOGY-DRIVEN DIALOGUE SYSTEMS AND ONTOLOGY LEARNING APPROACHES

The most important aspects covered in the study are ontology methods for the automatic generation of ontologies from natural language text, and their structural features; generation of formal queries, and natural language responses, mainly forming a natural-language interface of a graph database. Some of the relevant information on this topic can be found in our previous studies (Litvin et al., 2020, 2021; Palagin et al., 2014; Palagin et al., 2020, 2018).

Natural language dialogue systems, so-called chatbots, have a long history and a variety of approaches. In the following, we will review some examples of dialogue systems that have been developed in recent years, especially those that use computer ontologies in one way or another.

A typical example of a natural language dialogue system for English is described in Quamar, Lei et al. (2020) and Quamar, Özcan et al. (2020). The analysis of the user's input phrase assumes that English sentences have a fairly regular structure, which can be expressed quite well by a set of templates. Each of the templates has a constant part and some variables. The

https://doi.org/10.18489/sacj.v35i1.1233

constant part indicates its basic semantic type (`intention`), and the variable parts indicate the places from which the corresponding concepts are to be extracted (placeholders). These placeholders are specified according to certain `intentions` of the concepts expected to be at the positions. We will consider an example from the aforementioned studies. Here is a template:

```
Show me {@M} by {@D} for {@V}.
```

The curly brackets here are the placeholders. The markers in the placeholders here indicate the following:

`@M` corresponds to the main requested concept;

`@D` is the selection category for concepts such as `@M`;

`@V` is the filter parameter.

To illustrate the idea, we consider a typical phrase, e.g.: `Show me admits by major diagnostic category for 2017`. It fully complies with the above template. The core entity that the user asks for is `admits` (in this case it corresponds to `number of hospitalisations`). The category of selection and sorting is `major diagnostic category` and the filter parameter is `2017`. The constant part of the template with certain positions and types of placeholders defines the corresponding `intention`. For each `intention` there is a corresponding set of database (DB) queries and instructions for displaying the results on the user interface. Databases for the main information storage are relational. The system also contains an ontology used to structure the categorisation of types and measurements of data stored in the main database. The `intentions` and concepts derived from the user's source phrase are compared with the ontology to determine those closest to the requested dimensions and categories from those available in the databases. One of the features of the system is that the ontology is automatically created based on a relational data model.

The approach presented here also focuses on the problem of pronoun substitution, which means the following: if a template variable appears to be a pronoun or just empty, the program uses the relevant data from the last of the previous queries. If there is no information in the previous queries, the default values are substituted instead of the pronoun. These defaults are formed on the basis of the most common queries collected during system usage. Currently, the system does not include automatic learning, although the authors have declared the possibility of its development in the future.

Some dialogue systems use an ontology as the main knowledge base. For formal queries, SPARQL is often used. Since the user interface assumes the use of natural language, one of the key tasks is the conversion of a natural language query into a formal query. One example of automated conversion of natural language queries into SPARQL frameworks is PAROT (Ochieng, 2020), which uses an approach that generates the most probable RDF triple based on the user's question. The triple is then validated by a special module that includes a dependency

https://doi.org/10.18489/sacj.v35i1.1233

analyser to process user queries on RDF triplets. The RDF triplets obtained in this way are then transformed into ontological triplets using a special thesaurus. The generated ontological triplets are used to build a SPARQL query, which is used to obtain answers from the ontology. Testing of the PAROT framework by the author (Ochieng, 2020) showed that it has an accuracy of about 81–82% for simple questions, about 43–56% for complex ones, and for a specific thematic dataset (geography) the accuracy is increased to 88%.

Another example of an implementation of natural language conversion with SPARQL techniques is FREyA (Damljanovic et al., 2012), available on GitHub (V. Kumar, 2022). FREyA provides an interactive natural language interface for ontology queries. It uses parsing combined with ontology-based search to interpret questions and, if necessary, prompt the user. User choices are used to train the system, which improves the accuracy of its operation. This system is currently implemented for English only. In the GitHub repository (V. Kumar, 2022) some examples are given to illustrate how natural language questions could be converted to SPARQL using FREyA. Note that the FREyA configuration can be tuned to a specific ontology structure.

Another system is the LODQA (Linked Open Data Question Answering) system (Shaik et al., 2016) that accepts a natural language input query and returns SPARQL queries along with the corresponding responses. The system is composed of several modules. The first module processes the natural language input query. 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 ontology graph is the focus of the query, i.e., what the user will get in 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 dataset. As soon as the first module has generated a pseudographic template from the given natural language query, the next module is activated, which is responsible for finding URIs and node values in the pseudographic template. URIs and values must be present in the target dataset. For normalisation, each node of the pseudographic template is associated with the URI of the dataset. The concept in natural language could be normalised (reduced to its original grammatical form) in more than one way due to possible ambiguities. Therefore, more than one template could be obtained from a pseudographic template. The third module for the generated pseudographic template performs a search in the target dataset for the relevant parts, taking into account possible changes that may occur in the dataset. To account for 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 differences. SPARQL queries are then sent to the target endpoint, where responses are obtained and then returned to the user. These query arguments can be of a primitive type, such as `S, N, or NP`, or complex, such as `S/NP, or NP/N`. A slash means the argument should be displayed on the right, and a backslash means the argument should be displayed on the left. The system uses the following notation for parts of speech, e.g.: `NN` –

https://doi.org/10.18489/sacj.v35i1.1233

noun, DT – definition (adjective), VB – verb. To facilitate the identification of RDF triplets, the words in the sentence are lemmatised and assigned the appropriate grammatical properties. The LODQA system currently only works with the English language. Detailed features of its operation are not given in Shaik et al. (2016), but are limited to a general description and analysis of working examples.

Although the development of dialogue systems, as well as the machine processing and *understanding* of natural language text, is mostly done for the English language, it is not limited to it. For example, Altinok (2018) presents a dialogue system for the German language, which uses an ontology to act as a dialogue manager (OntoDM) that maintains the state of the conversation. The ontology is also used as a knowledge base. These roles are combined. Domain knowledge is used to track objects of interest, i.e., ontology nodes (classes), which are products and services represented in the ontological knowledge base. In this way, the conversation history memory capability was introduced. A significant part of the article by Altinok (2018) is devoted to the peculiarities of linguistic problems of German language processing. At the time of the publication of byt Altinok (2018), the research work was still in progress and the criteria for the quality evaluation of the system had not yet been obtained. Another example is the article by Jung and Kim (2020), which describes the development of a dialogue system for the Korean language, which is fundamentally different from European languages.

One of the most promising graph database management systems is Neo4j (Goel, 2015; Helou et al., 2019), which provides relatively high performance as well as scalability and is suitable for working with large amounts of data. It is also currently one of the most popular graph databases. The formal query language used in Neo4j is Cypher. It has a wide range of capabilities, is quite flexible, and is open to adding functionality through plug-ins, such as the implementation of typical algorithms on graphs. However, unlike SPARQL, there are currently not many developments to convert natural language queries to formal queries in Cypher, with the exception of Sun (2018) and Srinivas (2023). The system proposed by Srinivas (2023) is rather primitive as queries must have a predefined structure. This approach is close to the one presented by Quamar, Özcan et al. (2020): a set of sentence templates in natural language, where some fragments are replaced by a special notation, as places from which the concepts are to be extracted for substitution in a query template. Each such template sentence corresponds to a specific query pattern on Cypher. The main advantage of the described approach is its simplicity, while the main disadvantage is that a real dialogue system requires a large number of such sentence templates containing all possible query options. Moreover, this approach is justified for languages with a regular sentence structure, such as English, where fewer phrase patterns are needed. Inflectional languages, e.g. Ukrainian, have a complex sentence structure with relatively free word order. This fact significantly increases the number of required templates.

Another important concept to be considered here is called ontology learning (OL), which aims to automatically or semi-automatically discover ontological knowledge from various forms of data and can overcome the bottleneck of ontology acquisition in ontology development (Zhou, 2007). The term *o*ntology learning had its origins in 2000 when Maedche

and Staab (2001) introduced it as a newly emerging field of research aiming at nothing less than the automatic generation of ontologies (Watrobski, 2020). Ontology development still faces some challenges, such as knowledge acquisition and the lack of sufficiently validated and generalised development methodologies. Zhou (2007) introduced the concept of rapid ontology development (ROD). It consists of three phases: design, learning, and validation. The design phase involves the identification and detailed analysis of domains, requirements, and relevant resources with the help of users and/or domain experts. A variety of techniques can be used for domain analysis, such as interviews, questionnaires, and informal text analysis. In the learning phase, appropriate learning techniques are selected, implemented, and then applied to discover ontologies from domain sources. The learning results are evaluated and refined in the validation phase, where the discovered ontologies are checked for redundancy, conflicts, and/or missing information. The active involvement of users and domain experts is highly desired in this phase. ROD is an iterative process that is repeated until the result is acceptable to users and/or knowledge engineers. The framework for ontology learning consists of information extraction, ontology discovery, and ontology organisation. Ontology discovery here means that supervised and unsupervised learning algorithms have been applied to discover ontological concepts and relations from the extracted information.

The ROD approach is neither new nor the only existing technique of OL. The process of ontology learning is performed in an automatic or semi-automatic way using several sources, with an insistent need for human involvement (Navarro-Almanza et al., 2020). Fully automated approaches are available, but they are still difficult tasks because in most cases human involvement is required. OL systems can be categorised according to the type of data they learn from Ma and Molnár (2020). These data types are unstructured, semi-structured, and structured (Faizi et al., 2020; Konys, 2017; Watrobski et al., 2014). More specifically, ontology learning methods can be classified according to the technique used, specifying the following solutions: methods and utilities for ontology learning from semi-structured schemas, knowledge bases, text, dictionaries, relational schemas, the Web, social data, and across languages, based on term extraction and concept formation, based on relation discovery (Ibrahim et al., 2023; S. Kumar & Kumar, 2022; Sivasubramanian & Jacob, 2020). These approaches can also be distinguished based on statistics, rule-based approaches, hybrid techniques, linguistic techniques, as well as logical techniques and resources (Ibrahim et al., 2023; Konys, 2015; Konys, 2017; Konys & Drazek, 2020; S. Kumar & Kumar, 2022). These solutions use different techniques depending on the different goals to be achieved. Depending on the results, these approaches differ depending on the techniques used. Some of them start to create an ontology from scratch, while others import and use existing ontologies. In addition, ontology learning systems differ in the degree of automation from semi-automated, collaborative, or fully automated (Konys, 2015; Konys, 2018; Konys & Drazek, 2020). Watrobski (2020) reports that there are at least 22 different approaches to OL today.

The OL methods that are closest to the scope of our study and are of interest to many researchers are those that use natural language text as an input source. Among the techniques used for this purpose are various combinations of NLP approaches, machine learning

(including deep learning), knowledge extraction using through patterns, semantic similarity metrics, clustering methods, statistical approaches, word patterns, etc. In the current research, a method is considered, which is primarily aimed at highly inflectional type languages, in the core of which there is a rule-based statistical analysis, which provides information about the semantic relations between entities mentioned in the analysed text. The information obtained in this way will be used to create an OWL ontology.

Thus, the main purpose of this research was the development and study of methods for automatic generation of ontologies from a natural language text, focusing primarily on highly inflectional types of languages, e.g., Ukrainian, and its subsequent use as a knowledge base in dialogue and reference systems using Neo4j and the Cypher query language.

Each of the main parts forming the system is considered in the next section: automatic generation of the ontology, natural language interface of the graph database and synthesis of responses in natural language using the results of the formal query.

## 3 PROPOSED APPROACH TO ONTOLOGY GENERATION FROM PLAIN NATURAL LANGUAGE TEXT

A complete description of the technique of automatic ontology creation based on a natural language text is given in (Litvin et al., 2020, 2021). Let us consider the ontology structure itself in terms of OWL.

In the proposed basic approach, there is a rule-based syntax-semantic analysis method. It is known that in highly inflectional languages, the main role of words bounding in a sentence belongs to the combination of certain flections (changeable endings of words). A developed language system of highly varying flection combinations for different parts of speech allows the expression of considerable semantic information. Thus, mere analysis of word forms for compatibility checking has the potential to bring not all, but a significant part of semantic information.

In the considered approach we have proposed about 90 semantic types, each of them actually could have several (up to more than a hundred) sub-types depending on such additional characteristics such as gender, tense, number, or a certain preposition for each of the words from the considered pair. However, these additional sub-types are not used in ontology creation, but they may be useful in the future for even deeper and more precise information structuring. Moreover, they come directly from the so-called `correlators`, which are the parts of the analyser that give the programs the options of how one or another semantic type could be expressed. There are several systems of semantic types. The set considered here could be derived from the astigmatic basis set by the method proposed in (Litvin et al., 2020, 2021).

The creation of an ontology from a text includes two key operations, which are performed by separate program modules. The first, probably the most important and the most resource-consuming part consists of the syntax-semantic analysis of the input text. The corresponding

https://doi.org/10.18489/sacj.v35i1.1233

program module contains a manually created set of so-called `correlators` and `determinates`.

The `determinates` represent combinations of possible flections of words and prepositions between them (if any) corresponding to each option of semantic sub-types. They also indicate whether the combination is inverse and which of the words in the pair is to be considered the main one. The labels of the semantic sub-types are given as contention symbols, e.g. `K1001, K4801, K6201`, etc. An example of a line from the `determinates` file for the Ukrainian language is shown in Listing 1.

```
1 ім під ям  L   I   K6201K8644K8646
```

Listing 1: An example of a line from the `determinates` file for the Ukrainian language

Here we can see flections for the first and the second of the possibly related words *ім* and *ям*. The preposition *під* is assumed to be between them. The symbol `L` marks that the main word is the left one, as the given flections, and the symbol `I` says that the link is inverse. The possible semantic sub-types for the combination are designated as `K6201, K8644, K8646`.

The `correlators` represent the correspondence of each of the semantic sub-types to the possible options of parts of speech combinations, including their order in the pair. In addition, the verbose names of the semantic types (`macro types`) are given. For each of these `macro types` there may be several (up to more than a hundred) suitable sub-types. An example of a line from the file of `correlators` for the Ukrainian language is shown in Listing 2.

```
1 K3506 отделимостьдействия_ S4S1;S4S6;S4S13;S4S5;S4S3;S4S10;S4S11;S4S12;S4S18;S4S22;S4S25
```

Listing 2: An example from the file of `correlators` for the Ukrainian language

Here we have K3506, which is a semantic sub-type label. Then there is *отделимость_действия* (Engl.: separability of an action), which is a verbose name of the corresponding semantic *macro type*. Then there is a sequence of possible parts of speech pairs given as contentious symbols. For example, `S1` is a noun, and `S4` is a verb. The pairs are separated by semicolons.

Also, the program has a dictionary including word stems, lemmas, and flections. The dictionary gives the correspondence between stems, lemmas, and sets of flections. The dictionaries are stored in a special compact format and could be created automatically using open-language data.

The main purpose of the program is to use these data and the input text to find and typify the words in it, to determine stems and flections, to recognise the links between the words, and to find out their semantic types. The other result of such an analysis is obtaining the related word groups in the sentences. Formally, a group is a fully connected graph. Practically such groups could correspond to a whole simple sentence, a part of a complex sentence, or a participial sub-phrase. The outputs of this module are two XML files *allterms.xml* and *parce.xml*. They are used for the subsequent creation of the OWL ontology.

The *allterms.xml* file is just a list of terms – nouns and groups of names found in the analysed text with some of their properties. It consists of two main parts: `<terms> and <phrases>`. The first one contains terms. An example of how a term is displayed is shown in Listing 3.

https://doi.org/10.18489/sacj.v35i1.1233

```
<term>
    <ttype>Noun_noun</ttype>
    <tname>тіло людини</tname>
    <wcount>2</wcount>
    <osn>тіл</osn>
    <osn>люд</osn>
    <sentpos>1/1</sentpos>
    <sentpos>1/2</sentpos>
    <reldown>2</reldown>
    <reldown>4</reldown>
</term>
```

Listing 3: An example of how a term is displayed in *allterms.xml* file

The <ttype> tag specifies the sequence of parts of speech that form the term. The <tname> tag is the text of the term as given. The <wcount> tag indicates the number of words in the term. Tags <osn> are given for each of the words and represent stems. Tags <sentpos> indicate the position of the words in the text (the sentence number, from 0 / the word position in the sentence, from 1). Tags <reldown> and <relup> are optional. They show the relations of the considered term to other terms in the file. Tag <reldown> points to the term of the narrowing context - any of its words could be found in this term, but the considered term contains more. The <relup> tag points to the term of expanding context – contains all words of this term and some others. Tags <reldown> and <relup> help to build a hierarchy of terms in the created ontology.

The <sentences> tag part contains only the texts of all sentences from the considered text in the <sent> tag.

The file *parce.xml* represents the syntax-semantic scheme of each sentence of the text. The sentence structures are given in container tags <sentence>. This container contains the following tags: several <item> tags representing the words and their properties; <sentnumber> – the number of the sentence in the text (from 1); <sent> – the text of the sentence. An example of the <item> tag is shown in Listing 4.

```
<item>
    <word>Книга</word>
    <osnova>кни</osnova>
    <lemma>книга</lemma>
    <kflex>а</kflex>
    <rel_type>К0</rel_type>
    <flex>га</flex>
    <number>1</number>
    <pos>1</pos>
    <group_n>1</group_n>
    <speech>S1</speech>
    <relate>0</relate>
</item>
```

Listing 4: An example of the <item> tag in *parce.xml* file

The <word> tag contains the text of the word as it appears in the current text. The <osnova> tag represents the root of the word. The <lemma> tag gives the lemma, the dictionary form of the word. Tags <kflex> and <flex> are flections. <kflex> – is just the ending, but <flex> is the part of the word that could be changed. The <number> tag is the

number of the word in the phrase. The `<group_n>` tag indicates that the word belongs to a group associated with the phrase. The `<speech>` tag contains the mark of the corresponding part of speech. The `<relate>` tag indicates the number of a word from which there is a semantic relation to the considered word. If the word has no incoming relations, as in the example above, its value is set to zero. And the `<rel_type>` tag is the type (sub-type) of the semantic relation. The `Ko` value means the absence of the relation or its unknown type.

The *allterms.xml* and *parce.xml* files are used to create an OWL ontology. Before describing the technique of its creation, let us consider the appropriate structure of the ontology that we are going to generate.

The ontology consists of classes and properties. The main classes are `Action`, `Adjective`, `Adverb`, `Name`, `Number`, `Preposition`, `Term`, `Negation`, and `UndefinedEntities`. So, we can see those ontology entities are sorted by their parts of speech. Subclasses of the class `Term` are the name groups and nouns with a hierarchical structure. Descendants of the `Name` class are given names. The most important properties are as follows: `SentenceGroups`, `Groups`, and `WordsLink`. The descendant properties of `WordsLink` are the semantic types. In certain ontologies, not all 90 of them may be presented, but only those that appear in the considered text. The descendants of these properties are certain links between entities (represented by classes). The *domain* of such a property is the main word in the linked pair, and the *range* is a dependent one. The property groups' descendants represent the linked groups of sentences. They can have types specified in their range field. It can be *subordinate*, *participial* or just nothing for other cases. The property `SentenceGroups` descendants represent the sentences, and descendants are subsequently linked to word groups.

With the two files mentioned and the file with `determinates` containing the verbose names of the semantic types, it is not difficult to create a short-described ontology. All OWL entities are first created as OOP representation objects. The root classes and properties are created first and are mandatory. Then a hierarchical structure of terms (nouns and name groups) is created using the *allterms.xml* file. Then, using *parce.xml*, the classes of other types of words and properties corresponding to the relations between words are created. At the same time, properties of type `WordsLink` are created, representing the semantic types. Since only one sub-type is specified in the *parce.xml* file, `determinates` are used to determine the semantic `macro type`. Since the words belong to the associated groups and these to the sentences, this information is used to create corresponding `Groups` and `SentenceGroups` sub-properties. The `Groups` and `SentenceGroups` descendant properties are supplemented with a label that contains the text of the corresponding group or sentence. These contexts seem to be useful for more informative ontology responses. The typing of the related word groups is done by the presence of certain words in the group: subordinating union, participle, and gerund.

The Neo4J GDMS could be used to work with the ontology of the described type. For this purpose, an OWL file should be loaded into it using the *Neosemantics* plugin. In this case, classes and properties become graph nodes of the corresponding type, which are `Class` and `Relation`. Relationships between nodes can be of the following types `SCO` - a subclass of;

https://doi.org/10.18489/sacj.v35i1.1233

SPO - a sub-property of; DOMAIN; RANGE. The Cypher language is used for queries.

## 4 ANALYSIS OF THE USER INPUT PHRASE TO GENERATE CYPHER QUERIES TO THE NEO4J ONTOLOGY REPOSITORY

In highly inflectional languages, word order is less important than the fact that a word exists in a certain form. In this case, it may be sufficient to perform a series of tests on the sentence in question to verify several criteria. Based on the test results it may be possible to determine some semantic information and the entities (words, name groups) used to represent it. In the simplest test version of the system that exists now, there are the following main checks:

1. Question word – 6 lists + absence of such word. The result is the number of sufficient lists from 1 to 6 or 0 if there is no question word in the sentence. It can be not a single word but a combination of words, which is used as a marker of a certain type of question.

2. One word from given lists (most of them are specific verbs) – 6 lists + absence of words from all lists. The result is the number of the sufficient list from 1 to 6, from 0 – if there are no such words in the sentence. The words that clarify the general semantics of the sentence, such as location, aim, way of doing, etc., are checked here.

3. A noun in the nominative case, excluding words from the check (2), if there are any. The result can be 1 - there is such a word (+ the word itself) or 0 – there is no such word. Several entities can be selected from the phrase.

4. A verb, except those from the lists in check (2), if any. The result can be 1 – there is such a word (+ the word itself) or 0 – there is no such word. Several entities can be selected.

Even this short test has a large number of variants (196) for the possible results, making it possible to have several templates or different types of templates.

However, even this is not enough. Therefore, an additional check should be done to find some additional relations in the analysed sentence. The procedure is as follows: adjectives related to the word from sentence (3), which must be close to it and match it in number and gender; nouns in indirect cases (they form the basis of the additional relations) and adjectives related to them; and last but not least, checking the presence or absence of negation predicates. According to the results of these additional checks, the modifier templates can be selected to be added to the base template.

The templates are stored as XML files with a special structure. An example of one of the simplest templates is shown in Listing 5.

https://doi.org/10.18489/sacj.v35i1.1233

```
<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>
```

Listing 5: An example of XML template for Cypher query formation to obtain contexts which include the given term

While the given template is one of the simplest, it is clearer to explain its general structure. The chapters of the XML template `<match>`, `<where>`, and `<return>` correspond to certain sections of a Cypher query (Jung & Kim, 2020). The template contains variables. They are described in the `<variables>` block. Each variable is defined by its name and its binding in the corresponding XML containers. The binding can have the values `input` or `output`. The input variables are to be replaced with the values of the input parameters during query generation. The output ones define the parameters to be obtained as a result of the query execution. The `<id>` container is required for the template identity and is used to search for it. The `<verbose_name>` tag is not used by the program but helps a human to easily identify it during system development and maintenance. The `<type>` tag indicates the type of template - base or additional. An example of a base template is given in Listing 5.

Let us look at the structure of additional templates. An example of one of them is shown in Listing 6.

https://doi.org/10.18489/sacj.v35i1.1233

```
<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>
```

Listing 6: An example of an additional XML template aimed at the addition of some conditions related to the subject adjectives' presence

The template in Listing 6 also has `<match>`, `<where>`, and `<return>` blocks. Their content is not independent but should be added to the appropriate parts of a query formed by the base template. In this case, some of the sections may be empty. The main feature of an additional template is the presence of blocks `<block_union>` and `<next_item_union>`. The `<block_union>` tag indicates how the `<where>` block must be united with the query formed by the base template. The `<next_item_union>` tag specifies the union type for the repeated elements of the `<where>` block in the case where the corresponding variable is presented as a list (array). For example, the `INPUT_VALUE_ADJ` variable could correspond to several adjectives associated 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 do not participate in the transfer of values to the forming query, nor the return of results. They are only used to mark the template parts that should not be duplicated during the part repetition. They are implemented with an order number, for example, `ADJ_PLUS_1, ADJ_PLUS_2, ADJ_PLUS_3`, ..., etc.

Let us take a closer look at the structure of formal queries and how they are created. The presented structure of the ontology makes it possible to search for contexts or individual terms. It has allowed not only the presence of some entities in the considered context but also their relations according to a certain semantic category. In the presented scheme there is a basic

https://doi.org/10.18489/sacj.v35i1.1233

query template, aimed at obtaining information of a certain type in a given form, and an additional modifier template, which optionally adds the description of additional conditions. The above template is aimed at obtaining a context containing a specific term (word). However, the term must not only be presented in the context but also form a link with others, which could guarantee that the term is *organically* implemented in the context.

From here on, most of the query template examples here are given in a simplified format - without XML tags. Cypher queries are divided into three main parts: `MATCH`, `WHERE`, and `RETURN`. The `MATCH` block specifies a pattern for linking the nodes in the oriented graph. The `WHERE` part specifies the conditions that characterise the entities (nodes and relationships) from the `MATCH` case. The `RETURN` block shows what is to be returned as a result and with what names (aliases). In the example in Listing 7, there is a class identified by the variable `inp`. In the `WHERE` block, a condition is added that says that the `name` field of the `inp` node must be equal to a specific value (here `INPUT_VALUE` is the text of the input value). From the `MATCH` block, it is clear that `inp` is a node because of the parentheses, and it must be of type `Class`. It must be linked to another node *n* of type *Relationship*, which corresponds to an OWL ontology property. The link type is undefined in this case (the square brackets are empty), and the direction of the link is also undefined. So, the node could be linked as either `DOMAIN` or `RANGE`. In this case, there is no need to specify the direction of the link, because it is known that such links always come from a property to a class. We also know that this property must be linked to some class `x`. It is also 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` is a sentence. As a result, the query returns the value of `rel_sent.label` with the alias `context`, which is the sentence context.

```
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;
```

Listing 7: Cypher query to request the properties of an entity

Listing 7 also shows how to query the properties of an `INPUT_VALUE` entity contained in the ontology. To specify what the `INPUT_VALUE` is or could be an additional statement is added to the `MATCH` block: `(n)-[:SPO]->(prop_type_1)`. This specifies that the property `n` must be a child of `prop_type_1`. This is where the link direction is specified. In the `WHERE` block, sufficient values of the `label` field of `prop_type_1` are given. To make the query

https://doi.org/10.18489/sacj.v35i1.1233

template more universal, since it is not known whether `INPUT_VALUE` is a noun or a verb, some options are given for the possible value of `prop_type_1.label`, combined with logical `or`. If the ontology has a semantic category hierarchy, the construction could be simplified as shown in Listing 8.

```
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;
```

Listing 8: Cypher query to retrieve the properties of an entity in case the ontology has a semantic category hierarchy

The result is the query field `label` of the `x` node. This will be the properties of an `inp` object. The contexts are also requested to recognise the conditions in which the property of the entity is mentioned. Similarly, actions of an object could be requested. For this purpose, it is only necessary to set another value for `prop_type_1.label` in the `WHERE` block, namely: `prop_type_1.label = object-action`.

If there are several possible options of relation in the query (`prop_type_1.label`), the result can contain its certain value, which then helps in answer synthesis. The next example (Listing 9) illustrates a query of object localisation without its type concretisation (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;

```

Listing 9: Request object localisation using Cypher query

The main peculiarity here is the statement `prop_type_1.label as predicate` in the `RETURN` block. This makes it return the specific semantic type of the obtained result.

In some cases, instead of predicates, lists of some entities (verbs, nouns, adjectives) can be included in a query. The peculiarity here is that the conditions are given for the node of

https://doi.org/10.18489/sacj.v35i1.1233

the ontological graph linked to x. Thus, the requested object must not only be linked to some term x by the specific relationship, but this term must also be from a specific list. If the terms (or actions) are additionally classified in the ontology, the condition for the term will only be a descendant of a certain category.

Special mention should be made of modifier templates – fragments that can be added to the main query templates. Let us consider an example where the input parameter is not a single word, but a group of nouns. Therefore, there are related nouns and adjectives. To link to the input adjective concept, the relevant statements must be added as shown in Listing 10.

To the MATCH block:

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

and in the WHERE block:

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

Listing 10: Parts of the Cypher query to be added for a link to the subject adjectives presence condition introduction

For the additional adjectives, add the same blocks but with further variables `inp_adj_2`, `inp_adj_3`, etc. It is also possible to add a condition of a noun in the indirect case by the statements shown in Listing 11.

To the MATCH block:

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

and in the WHERE block:

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

Listing 11: Parts of the Cypher query to be added for the condition of additional conditions expressed with nouns in indirect cases

In Listing 11, there is a condition for the presence of a noun in the same group where the main concept is included. However, conditions for the presence of adjectives related to this noun could also be added as shown in Listing 12.

https://doi.org/10.18489/sacj.v35i1.1233

To the 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 the WHERE block:

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

Listing 12: Parts of the Cypher query to be added for the condition of adjectives related to nouns in indirect cases

In some cases, you may need to add a negation predicate to a query. To do this, you add a construction to the query as shown in Listing 13.

To the MATCH block:

```
    (neg:Class)-[]-(neg_rel:Relationship),
    (neg_rel)-[:SPO]->(rel_group)
```

and in the 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")
```

Listing 13: Adding negation predicate condition parts to the Cypher query

## 5 SYNTHESIS OF NATURAL LANGUAGE RESPONSES BASED ON THE RESULTS OF FORMAL QUERY EXECUTION

The user interface of a dialogue system that simply displays the results of a formal query, even if it is beautifully designed, may not look very friendly, and sometimes may not even be completely understandable to a human. Therefore, the next important problem is the synthesis of natural language responses. Some principles of the approach of response generation based on information from the results of formal queries and the analysis of the source phrase using template instructions are described in our previous research (Litvin et al., 2020, 2021; Palagin et al., 2011; Palagin et al., 2020; V. Y. Velychko et al., 2014; V. Velychko et al., 2022).

In general, during the system development, the decision on how the response should appear in the user interface has to be balanced between providing ready-made contexts and text

synthesis. For example, to provide some tables, graphical objects, or other media to illustrate the response, the best option is to use ready-made contexts with links to the relevant files. In the current study, we omit methods for visualising and creating 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 Quamar, Lei et al. (2020), can be well implemented.

Contextual responses may be the best option when you need to provide detailed information. The synthesised responses provide better clarity for more specific questions where a formal answer is just a list of entities from the ontology. In this section we provide some examples of synthesised responses with instruction templates for some typical cases. These templates also provide user contexts (sentences) that explain and confirm the statement. 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. Attempts have been made to add response templates in the form of XML descriptions, but this has resulted in increased complexity and reduced performance of the software.

Listing 14 shows the response template of a question about entity properties.

```
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
```

Listing 14: The response template for the entity characteristic

The PyMorphy2 library methods (Litvin et al., 2020, 2021; Palagin et al., 2022) are used for determining the morphological characteristics of the word (part of speech, gender, case, etc.) and for word form matching. In the simple example in Listing 14, the part of speech of `INPUT_VALUE` must be checked. It can be a noun or a verb. If it is a noun, the `result` value must be gender-matched with `INPUT_VALUE`.

A more complicated example is shown in Listing 15. Here the subject of the query is object localisation. The particular localisation predicate is not specified in the input parameters of the query but appears in its results. As mentioned above, a certain semantic predicate could be used in a response synthesis.

https://doi.org/10.18489/sacj.v35i1.1233

```
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
```

Listing 15: The entity localisation response generation template with tuning for different localisation types

From the example in Listing 15 we can see that a certain type of semantic predicate (in this case, localisation) determines the appropriate preposition and case for the value of the `result` variable for the Ukrainian language.

## 6 CONCLUSIONS AND FURTHER RESEARCH

A technique for the automatic generation of an OWL ontology from natural language text has been proposed. It is assumed that the language of the considered text is of the highly inflectional type. A feature of the method is that it does not require any previous tagging of the text, or a regular structure. The essence of the technique consists of a rule-based syntax-

https://doi.org/10.18489/sacj.v35i1.1233

semantic analysis method and the fact that a large amount of semantic information in highly inflectional languages could be obtained by analysing the combinations of different parts of speech flections and prepositions. The ontology creation includes two stages: syntax-semantic analysis with intermediate creation of XML files, and ontology generation based on the corresponding OWL file. The proposed method was implemented as software and parameterised for the Ukrainian language. It was tested on real texts, which showed its efficiency. The created ontologies appear to be valid and can be processed by Protégé, RDFlib, and Neo4j (using the Neosemantics plugin). The ontologies appear to be deeply semantically structured and at the same time rather simple and regularly organised. Thus, the developed software system is a promising tool that can significantly and effectively automate the creation of graph databases using only plain text.

An approach and the corresponding software toolkit are developed for the creation of natural language dialogue systems based on the automatically built ontology for inflectional languages, in particular Ukrainian. An analysis technique is developed within the framework of the approach of an initial user phrase aimed at the formation of formal queries in the language Cypher. The essence of the method is a series of checks for the occurrence of certain words and/or word forms in the initial phrase. Depending on the set of check results, the main query template (or a group of such templates) is selected. Components from modifier templates are added to the main template (to its corresponding sections) as a result of additional checks, which make appropriate clarifications and extensions of the query. Query variables are supplemented with concepts obtained during the corresponding checks. Several queries (packages) can be created based on one initial phrase. An approach to the synthesis of natural language responses using query results and the values of source variables is also proposed. The unique feature of the approach is the use of specific values of semantic predicates obtained as a result of the query to the ontology, which allows the program to formulate the response more accurately and correctly by using appropriate prepositions and word forms. These response templates also provide instructions for matching word forms of concept 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.

In a future study, our team plan to implement the ontology-related system as a part of the knowledge-oriented digital library of the smart-system for remote support of rehabilitation activities and services (Chaikovsky et al., 2023; Malakhov, 2022, 2023; Palagin et al., 2022). Further research will aim to develop original instruments and tools with the purpose of optimising user queries, and optimising usability for ontology-related systems.

https://doi.org/10.18489/sacj.v35i1.1233

## CREDIT AUTHORSHIP CONTRIBUTION STATEMENT

Anna Litvin: Conceptualisation, Methodology, Writing – original draft preparation.

Vladislav Kaverinsky: Methodology, Software, Validation, Resources, Writing – review & editing.

Oleksandr Palagin: Project administration, Supervision, Conceptualisation, Methodology.

Kyrylo Malakhov: Software, Validation, Resources, Term, Writing – review & editing.

## ACKNOWLEDGEMENTS

Corresponding author Kyrylo Malakhov, on behalf of himself and co-authors Anna Litvin, and Vladislav Kaverinsky, thanks Oleksandr Palagin, Academician of the National Academy of Sciences of Ukraine, Doctor of Technical Sciences, Professor, Honored Inventor of Ukraine, Deputy Director for Research of the Glushkov institute of Cybernetics of the National Academy of Sciences of Ukraine, Head of the Microprocessor Technology Lab, who served as the scientific supervisor for this research.

The research team of Glushkov institute of Cybernetics would like to extend our special thanks and appreciation to Katherine Malan, Editor-in-Chief of the South African Computer Journal, for her unwavering support and dedication to promoting Ukrainian science during wartime within scholars publishing.

## FUNDING

This study would not have been possible without the financial support of the National Research Foundation of Ukraine. Our work was funded by Grant contract:

- Development of the cloud-based platform for patient-centered telerehabilitation of oncology patients with mathematical-related modeling, application ID: 2021.01/0136.

## DECLARATION OF COMPETING INTEREST

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## References

Altinok, D. (2018). An ontology-based dialogue management system for banking and finance dialogue systems. *arXiv*, *v1*(arXiv:1804.04838). https://doi.org/10.48550/arXiv.1804.04838

An, J., & Park, Y. B. (2018). Methodology for automatic ontology generation using database schema information. *Mobile Information Systems*, *2018*, 1–13. https://doi.org/10.1155/2018/1359174

Antoniou, G., & Plexousakis, D. (2016). Semantic web. In *Encyclopedia of database systems* (pp. 1–5). New York: Springer. https://doi.org/10.1007/978-1-4899-7993-3_1320-2

Balakrishna, M., & Srikanth, M. (2008). Automatic ontology creation from text for National Intelligence Priorities Framework (NIPF). *Ontology for the Intelligence Community*, 8–12. https://ceur-ws.org/Vol-440/Proceedings.pdf

Chaikovsky, I., Dykhanovskyi, V., Malakhov, K. S., & Bocharov, M. (2023). *Military medicine: Methods of control, improvement in individual combat readiness and telerehabilitation of military personnel*. Iowa State University Digital Press. https://doi.org/10.31274/isudp.2023.128

Damljanovic, D., Agatonovic, M., & Cunningham, H. (2012). FREyA: An Interactive Way of Querying Linked Data Using Natural Language. In R. García-Castro, D. Fensel & G. Antoniou (Eds.), *The Semantic Web: ESWC 2011 workshops* (pp. 125–138). Springer. https://doi.org/10.1007/978-3-642-25953-1_11

Elnagar, S., Yoon, V., & Thomas, M. (2020). An automatic ontology generation framework with an organizational perspective. *Knowledge Flow, Transfer, Sharing, and Exchange – Hawaii International Conference on System Sciences*. https://doi.org/10.24251/HICSS.2020.597

Faizi, S., Salabun, W., Ullah, S., Rashid, T., & Wieckowski, J. (2020). A new method to support decision-making in an uncertain environment based on normalized interval-valued triangular fuzzy numbers and COMET technique. *Symmetry*, *12*(4), 516. https://doi.org/10.3390/sym12040516

Goel, A. (2015). *Neo4j cookbook*. Packt Publishing.

Helou, S. E., Kobayashi, S., Yamamoto, G., Kume, N., Kondoh, E., Hiragi, S., Okamoto, K., Tamura, H., & Kuroda, T. (2019). Graph databases for openEHR clinical repositories. *International Journal of Computational Science and Engineering*, *20*(3), 281–298. https://doi.org/10.1504/IJCSE.2019.103955

Ibrahim, S., Fathalla, S., Lehmann, J., & Jabeen, H. (2023). Towards the multilingual semantic web: Multilingual ontology matching and assessment. *IEEE Access*, *11*, 8581–8599. https://doi.org/10.1109/ACCESS.2023.3238871

Jung, H., & Kim, W. (2020). Automated conversion from natural language query to SPARQL query. *Journal of Intelligent Information Systems*, *55*(3), 501–520. https://doi.org/10.1007/s10844-019-00589-2

Konys, A. (2015). A tool supporting mining based approach selection to automatic ontology construction. *9th Multi Conference on Computer Science and Information Systems*, 3–10. https://www.iadisportal.org/isa-2015-proceedings

Konys, A. (2017). Ontology-based approaches to big data analytics. In S. Kobayashi, A. Piegat, J. Pejaś, I. El Fray & J. Kacprzyk (Eds.), *Hard and soft computing for artificial intelligence, multimedia and security* (pp. 355–365). Springer International Publishing. https://doi.org/10.1007/978-3-319-48429-7_32

Konys, A. (2018). Knowledge systematization for ontology learning methods. *Procedia Computer Science*, *126*, 2194–2207. https://doi.org/10.1016/j.procs.2018.07.229

Konys, A., & Drazek, Z. (2020). Ontology learning approaches to provide domain-specific knowledge base. *Procedia Computer Science*, *176*, 3324–3334. https://doi.org/10.1016/j.procs.2020.09.065

Kumar, S., & Kumar, P. S. (2022). OLGA: An Ontology and LSTM-based approach for generating Arithmetic Word Problems (AWPs) of transfer type. *arXiv*, *v1*(arXiv:2211.12164). https://doi.org/10.48550/arXiv.2211.12164

Kumar, V. (2022). FREyA – a natural language interface for querying ontologies [Github ]. https://github.com/nmvijay/freya

Litvin, A. A., Velychko, V. Y., & Kaverynskyi, V. V. (2020). Method of information obtaining from ontology on the basis of a natural language phrase analysis. *Problems in Programming*, *2-3*, 323–330. https://doi.org/10.15407/pp2020.02-03.322

Litvin, A. A., Velychko, V. Y., & Kaverynskyi, V. V. (2021). Tree-based semantic analysis method for natural language phrase to formal query conversion. *Radio Electronics, Computer Science, Control*, *57*(2), 105–113. https://doi.org/10.15588/1607-3274-2021-2-11

Ma, C., & Molnár, B. (2020). Use of ontology learning in information system integration: A literature survey. In *Intelligent information and database systems* (pp. 342–353). Singapore: Springer. https://doi.org/10.1007/978-981-15-3380-8_30

Maedche, A., & Staab, S. (2001). Ontology learning for the Semantic Web. *IEEE Intelligent Systems*, *16*(2), 72–79. https://doi.org/10.1109/5254.920602

Malakhov, K. S. (2022). Update from Ukraine: Rehabilitation and Research [Letter to the Editor]. *International Journal of Telerehabilitation*, *14*(2), 1–2. https://doi.org/10.5195/ijt.2022.6535

Malakhov, K. S. (2023). Update from Ukraine: Development of the Cloud-based Platform for Patient-centered Telerehabilitation of Oncology Patients with Mathematical-related Modeling [Letter to the Editor]. *International Journal of Telerehabilitation*, *15*(1), 1–3. https://doi.org/10.5195/ijt.2023.6562

Navarro-Almanza, R., Juárez-Ramírez, R., Licea, G., & Castro, J. R. (2020). Automated ontology extraction from unstructured texts using deep learning. In *Intuitionistic and Type-2 fuzzy logic enhancements in neural and optimization algorithms: Theory and applications* (pp. 727–755). Springer International Publishing. https://doi.org/10.1007/978-3-030-35445-9_50

Ochieng, P. (2020). PAROT: Translating natural language to SPARQL. *Expert Systems with Applications: X*, *5*, 100024. https://doi.org/10.1016/j.eswax.2020.100024

Palagin, O. v., Malakhov, K. S., Velychko, V. Y., & Semykopna, T. V. (2022). Hybrid e-rehabilitation services: SMART-system for remote support of rehabilitation activities and services. *International Journal of Telerehabilitation, Special Issue: Research Status Report – Ukraine*. https://doi.org/10.5195/ijt.2022.6480

Palagin, O. v., Petrenko, M., & Malakhov, K. S. (2011). Technique for designing a domain ontology. *Computer means, networks and systems*, *9*(10). http://dspace.nbuv.gov.ua/xmlui/handle/123456789/46447

Palagin, O. v., Petrenko, M., Velychko, V., & Malakhov, K. S. (2014). Development of formal models, algorithms, procedures, engineering and functioning of the software system "instrumental complex for ontological engineering purpose". *CEUR Workshop Proceedings*, *1843*, 221–232. http://ceur-ws.org/Vol-1843/221-232.pdf

Palagin, O. v., Velychko, V. Y., Malakhov, K. S., & Shchurov, O. S. (2020). Distributional semantic modeling: A revised technique to train term/word vector space models applying the ontology-related approach. *CEUR Workshop Proceedings*, *2866*, 342–353. http://ceur-ws.org/Vol-2866/ceur_342-352palagin34.pdf

Palagin, O. v., Velychko, V., Malakhov, K. S., & Shchurov, O. (2018). Research and development workstation environment: The new class of current research information systems. *CEUR Workshop Proceedings*, *2139*, 255–269. http://ceur-ws.org/Vol-2139/255-269.pdf

Quamar, A., Lei, C., Miller, D., Ozcan, F., Kreulen, J., Moore, R. J., & Efthymiou, V. (2020). An ontology-based conversation system for knowledge bases. *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 361–376. https://doi.org/10.1145/3318464.3386139

Quamar, A., Özcan, F., Miller, D., Moore, R. J., Niehus, R., & Kreulen, J. (2020). Conversational BI: An ontology-driven conversation system for business intelligence applications. *Proceedings of the VLDB Endowment*, *13*(12), 3369–3381. https://doi.org/10.14778/3415478.3415557

Shaik, S., Kanakam, P., Hussain, S. M., & Narayana, D. S. (2016). Transforming natural language query to SPARQL for semantic information retrieval. *International Journal of Engineering Trends and Technology*, *41*(7), 347–350. https://doi.org/10.14445/22315381/IJETT-V41P263

Sivasubramanian, S., & Jacob, S. G. (2020). An automated ontology learning for benchmarking classifier models through gain-based relative-non-redundant feature selection: A case-study with erythemato-squamous disease. *International Journal of Business Intelligence and Data Mining*, *16*(3), 261–278. https://doi.org/10.1504/IJBIDM.2020.106132

Srinivas. (2023). Convert English sentences to cypher queries [GITHUB]. https://github.com/gsssrao/english2cypher

Sun, C. (2018). *A natural language interface for querying graph databases* (Doctoral thesis). Massachusetts Institute of Technology. https://dspace.mit.edu/handle/1721.1/119708

Velychko, V. Y., Malahov, K., Semenkov, V., & Stryzhak, O. (2014). Integrated tools for engineering ontologies. *International Journal Information Models and Analyses*, *3*(4), 336–361. http://www.foibg.com/ijima/vol03/ijima03-04-p03.pdf

Velychko, V., Voinova, S., Granyak, V., Ivanova, L., Kudriashova, A., Kunup, T., Malakhov, K. S., Pikh, I., Punchenko, N., Senkivskyy, V., Sergeeva, O., Sokolova, O., Fedosov, S., Khoshaba, O., Tsyra, O., Chaplinskyy, Y., Gurskiy, O., Zavertailo, K., & Kotlyk, D. (2022). *New information technologies, simulation and automation* (S. Kotlyk, Ed.). Iowa State University Digital Press. https://doi.org/10.31274/isudp.2022.121

Watrobski, J. (2020). Ontology learning methods from text - an extensive knowledge-based approach. *Procedia Computer Science*, *176*, 3356–3368. https://doi.org/10.1016/j.procs.2020.09.061

Watrobski, J., Jankowski, J., & Piotrowski, Z. (2014). The selection of multicriteria method based on unstructured decision problem description. In D. Hwang, J. J. Jung & N.-T. Nguyen (Eds.), *Computational collective intelligence. technologies and applications* (pp. 454–465). Springer International Publishing. https://doi.org/10.1007/978-3-319-11289-3_46

Zhou, L. (2007). Ontology learning: State of the art and open issues. *Information Technology and Management*, *8*(3), 241–252. https://doi.org/10.1007/s10799-007-0019-5