Method of Developing an Ontological System with Automatic Formation of a Knowledge Base and User Queries*

Oleksandr Palagin [1,†], Mykola Petrenko [1,*,†], Anna Litvin[1,†] and Mykola Boyko [1,†]

1 Glushkov Institute of Cybernetics of National Academy of Sciences of Ukraine, 40 Glushkov ave., Kyiv, 03187, Ukraine

Abstract

The article briefly discusses the models and tools for creating an ontological system that includes such components as automatic construction of an ontological knowledge base, analysis of short natural language messages in Ukrainian, and formation of queries in SPARQL and Cypher based on them. The server used is Apache Jena Fuseki, and the data warehouse is the Neo4J graph database. The approach is based on the fact that a user's natural language query is subjected to a series of sequential checks. Their results determine a set of semantic types expressed in the phrase (natural language query) and the corresponding concepts that define them. The result of these checks is a set of four values – the codes of the check results, as well as the subjects and predicates, if present. This information is enough to select a set of basic templates for formal queries. Even based on the results of such basic checks, it is possible to create enough basic templates to generate the final query. The proposed approach has a basic query template aimed at obtaining information of a certain type in a given form, as well as additional modifier templates that optionally construct query strings in the corresponding blocks of the main query by introducing additional conditions. Finally, we describe the process of automatic generation of SPARQL queries to a contextual ontology using the example of a knowledge base of medical articles from peer-reviewed open access journals.

Keywords

Semantic Web technology, ontology knowledge base, OWL ontology, SPARQL and Cypher languages, Neo4J graph database.

1. Introduction

The development of applications based on Semantic Web, Big Data, Natural Language Processing technologies in combination with neural network technologies has de facto become one of the most relevant areas of scientific research and practical development. In particular, this also applies to the construction of ontological systems (OnS) and relevant knowledge bases that are of interest to users.

14th International Scientific and Practical Conference from Programming UkrPROG'2024, May 14-15, 2024, Kyiv, Ukraine

* Corresponding author.

† These authors contributed equally.

palagin_a@ukr.net (O. Palagin); petrng@ukr.net (M. Petrenko); litvin_any@ukr.net (A. Litvin); swsfrmac@gmail.com (M. Boyko)

0000-0003-3223-1391 (O. Palagin); 0000-0001-6440-0706 (M. Petrenko); 0000-0002-5648-9074 (A. Litvin); 0000-0003-1723-5765 (M. Boyko)

© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR-WS.org/Vol-3806/S_2_Palagin.pdf

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

Information processing and knowledge representation based on ontologies emerged as a result of the search for a standard protocol for organizing knowledge in various fields of knowledge. This paradigm aims to offer a unified scheme and basic principles for the systematic representation, categorization, and interconnection of knowledge, regardless of the field of knowledge. The emergence of ontological strategies has made it possible to effectively build knowledge-based systems and, most importantly, laid the foundation for transdisciplinary interaction and ontological engineering in the field of modern artificial intelligence [1, 2, 3].

The distinctive feature of human intelligence is the ability to assimilate information from one source and adapt it in different areas, which is the basis of creativity and innovation. In order for universal machine intelligence to be practically effective, it must go beyond simple text comprehension. Its true advantage lies in its ability to use its store of knowledge to solve new problems. The ability of an artificial intelligence system to apply knowledge in diverse and new scenarios may well become the defining criterion for assessing its intellectual depth [4].

We have developed the above-mentioned OnS described in [5, 6, 7]. It greatly speeds up the receipt of scientific information by the user, but its weakness was the manual or automated creation of a database of scientific publications and SPARQL queries.

Formal queries to the knowledge base are essential for working with ontology. In our work, we consider queries in SPARQL and the increasingly promising query language Cypher, used in the Neo4J graph database. It is important to note that, presently, the creation of Cypher queries based on natural language phrases is underexplored in other research, especially for the Ukrainian language. Therefore, this research direction is relatively novel and relevant.

Communication with the knowledge base involves the use of formal query languages. Consequently, when creating dialogue and reference systems with a natural language interface, there is a need for the automated generation of packages of formal queries based on natural language user queries. These queries aim to retrieve relevant information from the knowledge base expressed in natural language.

2. Approach to creating formal queries based on analysis of natural language user messages

To address this challenge, we proposed an approach based on subjecting the user's natural language query to a series of checks. The results of these checks determine the set of semantic types expressed in the phrase and the corresponding concepts that specify them. The schema of a reasonably straightforward yet effective version of this method is provided in the Figure 1.

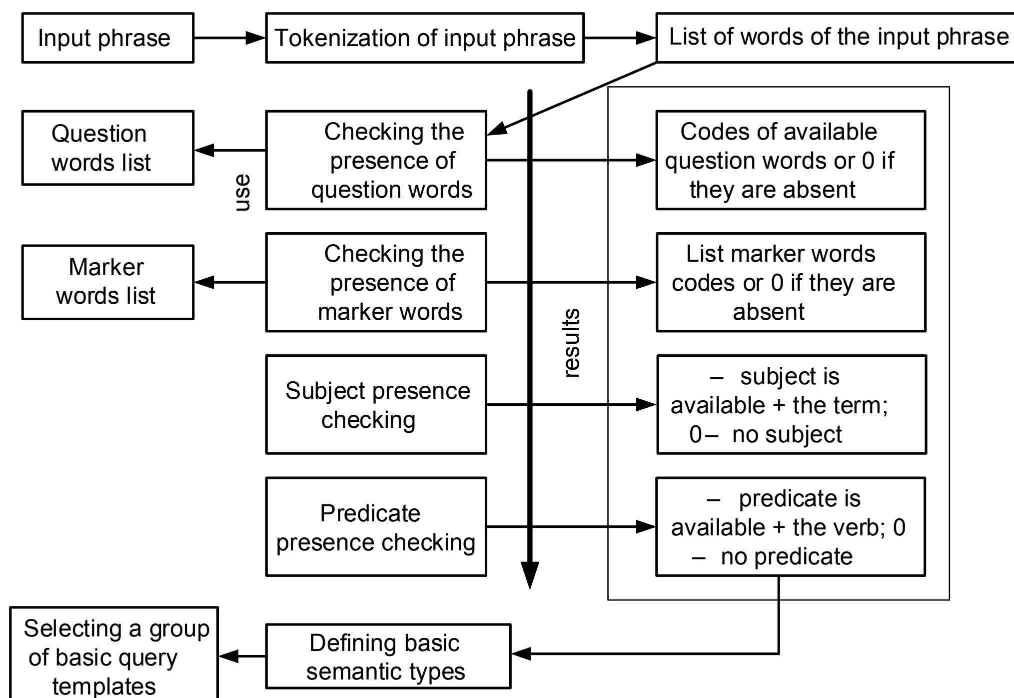

Figure 1: A scheme for parsing the user's phrase to select a basic set of formal query templates

It is worth noting that for inflected languages such as Ukrainian, word order is less crucial, and the presence of specific words and their word forms is more important. Figure 1 illustrates the schema for determining a set of basic semantic types and corresponding primary query templates. For this purpose, the processed expression, tokenized to the level of sentence lists and corresponding words, undergoes four successive checks:

Checking for the presence of question words (a critical point for determining the type of requested information).

Checking for marker words, primarily verbs such as "located," "works," "stands," "sends," etc. These words are categorized into groups of synonyms with similar semantic colouring.

Checking for the presence or absence of the subject and the subject itself, if present.

The next important aspect after the subject is *checking for the presence of a meaningful predicate,* if it does not fall into the categories of words from the second check.

The result of this analysis is a set (or list of sets) of four values — result codes from the checks — as well as subjects and predicates if present. This information is sufficient for selecting a set of basic templates for formal queries. Even based on the results of such basic checks, more than a dozen basic templates can be created.

Further, additional checks follow to determine more significant semantic nuances. However, for these checks, there are no separate basic templates; otherwise, the number of templates would significantly increase, and the templates themselves would have considerable code duplication. Instead, based on the results of these additional checks, modifications (changes and additions) are made to the basic templates according to the corresponding instructions. This makes the method more flexible and simplifies both the process of automatic analysis and the construction of the corresponding software system.

We will briefly describe the Neo4J graph database management system (DBMS) and its Cypher query language. In addition to DBMSs that work in conjunction with OWL/SPARQL mechanisms, such as Jena Fuseki, which is currently the de facto standard, there are alternative approaches to graph databases that can also be used to store and manipulate ontologies. The Neo4j DBMS [8] provides fairly high performance and scalability, and is also suitable for working with large amounts of data. The formal query language used in Neo4j is Cypher. It is quite powerful, flexible, and open to extending functionality through plug-ins, for example, to implement typical algorithms on graphs. However, at the moment, unlike SPARQL, there are not many developments for converting natural language queries into formal Cypher queries. Let's take a closer look at the queries described in this language.

3. XML templates for queries in the Cypher language

Query templates are stored in the form of an XML file with a specific structure. Here, we provide examples of templates for the Cypher query language, utilised in the Neo4j graph database. Its application is more preferable due to the substantial size of the ontology created through entirely automatic syntactic-semantic analysis of the text. This preference arises from the high performance of the Neo4j database management system in handling extensive datasets.

The ontology was formed according to the method described in [8]. Based on the syntactic and semantic relations between the concepts in the sentences identified during the text analysis, an ontological graph structure is built. Sentence contexts and their parts are also stored in the created OWL ontology. These sentences are associated with sets of semantic relations, specified by the corresponding entities. The typification of semantic categories is enclosed in an established hierarchical structure, which is described in [9]. In this paper, an example of an ontology is used to describe the approach to building queries in Cypher in a particular subject area.

As a working example of an OWL context ontology created on the basis of a set of documents with a predefined structure, we used a medical rehabilitation ontology based on files of scientific articles.

Let us consider an example of one of the simplest of such 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>.
```

The sections of the XML template, namely <match>, <where>, and <return>, correspond to specific sections of the formal query in the Cypher language [8]. Certain fragments of the content (text) in these sections serve as variable templates. Variables are described in the <variables> section, where each variable is defined by its name – <name> and its destination – <destination>. The destination can have values of either 'input' – indicating values substituted into the template, or 'output' – signifying variables not replaced during query formation by specific output values. Instead, they serve as references to the names and quantities of parameters whose values are obtained upon query execution. The <id> tag for the template identifier serves to match it with the result of the user phrase analysis, as well as the corresponding template for response formation. The <verbose_name> tag is included solely for human recognition of query templates during system development and maintenance. In the subsequent examples, query templates will be presented in a simplified form without XML tags. The <type> tag indicates the template type; currently, there are two types: 'base' – representing the primary template, and 'additional' – signifying an additional modifier template.

We should also dwell on the structure of additional templates. Here is an example of one of them:

```
<template>
      <verbose_name>An adjective related to the 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>
```

This type of template also includes the <match>, <where>, and <return> blocks. The content of these sections is added to the corresponding blocks of the base template. Each of these blocks can be empty or absent. Unique features of additional templates include the <block_union> and <next_item_union> tags. The <block_union> tag contains the type of union for the entire formed <where> block with the base template. The <next_item_union> tag indicates the type of union for repeated elements of the <where> block if the corresponding input variable is represented as a list (array). For example, in the above template, the variable INPUT_VALUE_ADJ may correspond to a series of adjectives related to the subject. The parameter values for <block_union> and <next_item_union> can be "and" or "or." Additionally, intermediate variables constitute a third type (<destination>) specific to additional templates. These variables do not participate in data transmission to the query or in their direct retrieval. Their main feature is that when repeating the block during query formation, these variables are not fully duplicated but are added with the next sequential number, such as ADJ_PLUS_1, ADJ_PLUS_2, ADJ_PLUS_3, and so forth.

4. The process of automatic generating queries based on templates

Let's delve deeper into the structure of formal queries and the method of their formation. The ontology's structure allows for targeted search of both contexts and individual concepts, considering the presence of these concepts in the context and their relatedness based on a specific semantic type criterion. In the proposed scheme, there is a basic query template aimed at obtaining information of a specific type in the specified form, along with additional modifier templates that optionally construct query strings in corresponding blocks of the main query, introducing additional conditions.

Let's examine examples of some query templates.

Let's start with perhaps the simplest one mentioned earlier. This template is designed to obtain the context (sentence) in which the queried single concept (word) is not just present

but forms a connection with other words. This ensures that the concept "organically" fits into the context.

In Cypher, queries are divided into three main blocks: MATCH, WHERE, and RETURN. The MATCH block specifies the pattern of relationships between nodes in a directed graph. The WHERE block imposes conditions on the properties (characteristics) of the nodes and/or relationships specified in the MATCH block. The RETURN block indicates what should be output as a result and under what name (alias). In this case, there is a specific class marked with the variable 'inp.' In the WHERE block, we imposed a condition that the label property of the inp node should be equal to the queried concept (hereafter, in query templates, INPUT_VALUE represents the text of the input concept). In the MATCH block, it is specified that inp is a node (enclosed in parentheses) of type Class. It is connected to another node 'n' that has a type Relationship (property in OWL). The type of the relationship is not defined (square brackets are empty), and the direction of the relationship is not indicated. This means it can be bound both to the DOMAIN and the RANGE. Specifying directions is unnecessary since it is known that such relationships are constructed from the property to the class. Additionally, we specified that this property should also be associated with a certain class 'x'. Then, we indicated that the property uniting these classes should relate to some sentence 'rel_sent.' The condition 'sent_super.name = "SentenceGroups"' guarantees that 'rel_sent' is indeed a sentence. As a result, we request to output 'rel_sent.label,' which contains the sentence context under the alias CONTEXT.

Let's consider a somewhat more complex example. We want to inquire about the known characteristics (definitions) of the object INPUT_VALUE in the ontology. In other words, what can INPUT_VALUE be (or is)? The query will take the following form:

```
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 separability" or
    prop_type_1.label = "impact level")
     and
    sent_super.name = "SentenceGroups"
RETURN DISTINCT x.label as result, rel_sent.label as context;
```

Compared to the previous example, in the MATCH block, one line has been added: (n)-[:SPO]->(prop_type_1). This provides information that the property 'n' must be a child in relation to 'prop_type_1' (relationship type). Here, we explicitly specify the direction of the relationship. In the WHERE block, we specify 'prop_type_1' through possible values of the label parameter. To make the template more universal, as we a priori do not know whether INPUT_VALUE is a noun or a verb, several options for the value of prop_type_1.label are

provided through logical OR. With the introduction of additional hierarchy of semantic relations into the ontology, this construction can be simplified:

```
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 = "property types"
      and
     sent_super.name = "SentenceGroups"
RETURN DISTINCT x.label as result, rel_sent.label as context;
```

As a result, we output the label for the nodes 'x'. This represents the characteristics (properties) of the object 'inp.' Additionally, we inquire about the sentence text to understand the context in which this property is mentioned.

Similarly, one can inquire about the actions of the object. For this, it is only necessary to specify a different value for prop_type_1.label in the WHERE block, specifically: prop_type_1.label = "object-action".

In the case of using a condition with multiple possible types of relationships (prop_type_1.label), the obtained type value can also be included in the result, aiding in response synthesis. Let's provide an example where the location of an object is queried without specifying the type of localization ("Where is INPUT_VALUE located?").

```
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 types" and
     sent_super.name = "SentenceGroups"
RETURN DISTINCT x.label as result, rel_sent.label as context,
               prop_type_1.label as predicate;
```

The main distinction here is the presence of the prop_type_1.label as predicate expression in the RETURN block. This allows for returning the specific semantic type of the obtained result. It should be considered when generating response text.

In some cases, in addition to predicates in queries, lists of concept variants (characteristic verbs, nouns, adjectives) can be used. The main difference is that conditions are imposed on

the ontology vertex associated with x.label. In other words, the queried object must be linked to a certain concept 'x' with a relationship of a specific type, for example, "object-action," and this 'action' should be described by the text parameter label as one of those listed in the set. In the future, there are plans to introduce an a priori (independent of the analyzed text) classification of actions, features, and concepts into the ontology, which will simplify the query and eliminate the need for such lists – the concept 'x' simply needs to be a child in relation to, for example, verbs of a certain category.

Let's discuss template modifiers separately – fragments that are added to the main query templates. For instance, the input parameter is not a single word but a name group, i.e., connected nouns and adjectives. To associate adjectives with the input concept, the following lines should be added to the respective blocks:

In the MATCH section:

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

In the WHERE section:

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

Similar blocks can be added for additional adjectives with variables like inp_adj_2, inp_adj_3, and so forth. Conditions for the presence of a noun in the genitive case can be added to the query using the following blocks:

In the MATCH section:

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

In the WHERE section:

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

Here, a condition is added only for the inclusion of this additional noun in the same group as the main queried concept. Conditions for the presence of associated adjectives with this noun can also be imposed:

In the MATCH section:

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

In the WHERE section:

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

In specific cases, it may be necessary to attach a predicate of negation to the query. To achieve this, conditions for inclusion in the group of the negation particle or another negation predicate are added to the query:

In the MATCH section:

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

In the WHERE section:

```
and
```

```
(neg.label = "no" or
neg.label = "not" or
neg.label = "forbidden" or
neg.label = "prohibited" or
neg.label = "unnecessary" or
neg.label = "impossible" or
neg.label = "needlessly")
```

Template modifiers – fragments that are added to the main query templates – are discussed in more detail in [9, 10, 11]. They also discuss the process of automatic generation of SPARQL queries to a contextual ontology using the example of a knowledge base of medical articles from peer-reviewed open access journals.

5. Automatic generation of SPARQL queries to contextual ontology using the example of a knowledge base of medical articles from peer-reviewed open access journals

The system receives a textual message as input. Initially, the text is cleaned from disallowed characters, which, in this case, include Latin (English) alphabet letters, whitespace, period, hyphen, paragraph symbols, and line breaks. All other characters are considered disallowed and are removed for further processing. Tokenization of the text into individual words is performed using NLTK library tools, along with the identification of their parts of speech. The words are lemmatized – brought to their base form, and stop-words are removed. Stop-words include articles, conjunctions, prepositions, pronouns, question words, auxiliary and modal verbs, and particles. The list of stop-words has been extended to include common conversational phrases such as "give," "show," and "present," which, while prevalent in interactions with the help system, do not provide informative content in this context. Additionally, the text is purged of words not included in the list of concepts presented in the knowledge base. Consequently, the processed text represents a list of meaningful words brought to their base form.

Subsequently, the system determines the specific semantic category or set of categories expressed in the given message. Each of these categories corresponds to a separate SPARQL query. Currently, 26 such query categories have been implemented: "synonyms," "symptoms," "indications," "patient education," "description," "reimbursement," "test," "rule out," "treatment summary," "relations," "prognosis," "diagnosis need for treatment," "contraindications precautions," "causes," "pathogenesis," "g-code," "any_code," "any_icd," "icd 9," "icd 10," "icd 11," "hcpcs," "cpt," "articles," "references," and "contexts." However, there is room for expanding their number. The determination of a specific semantic category is based on the presence of certain marker words in the obtained list. The mapping of these categories to marker words is implemented in the form of a dictionary in the file marker_words.json. In this dictionary, each of the specified categories is associated with a list or set of lists of marker words. For example:

```
"icd 10": [
      ["icd", "10", "code"],
      ["icd-10", "code"],
      ["icd", "10"],
```

```
            ["icd-10"]
        ],
        "risk factors": [
            ["higher", "risk", "factors"],
            ["risk", "factor"],
            ["risk"]
        ]
```

The list of words undergoes verification across all available categories. During this process, it is noted which specific marker words from the list identify each particular category. If the lists of markers intersect, the longest one is selected, containing all words found in the analyzed list. Words that belong to a syntactic-semantic group (sentence or sentence part) and were not identified as markers for any of the categories are marked as query parameters. The semantic category determines the query type. If no existing special semantic category is assigned to a syntactic-semantic group, it is categorized as "contexts," and the words themselves become parameters for the corresponding query.

The output returns list of identified special semantic categories and their corresponding words – query parameters.

A dedicated module within the software system is responsible for the generation of SPARQL queries. Each identified special semantic category corresponds to its own SPARQL query template. The templates are stored in the form of a JSON dictionary in a file. Here is an example of such a template:

```
  "description": {
  "verbose": "description for the case of",
  "parts": [{
  "type": "constant",
  "n": 1,
  "body": ["PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-
syntax-ns#>",
            "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-
schema#>",
            "PREFIX owl: <http://www.w3.org/2002/07/owl#>",
            "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>",
            "PREFIX name:
<http://www.semanticweb.org/ContextOntology#>",
            "SELECT DISTINCT ?topic ?context_text ?article_name
?scope",
            "WHERE {"
      ]},
  {
      "type": "listed input",
      "n": 2,
      "body": ["?word_>_order_< rdfs:label >_input_<@en.",
            "?word_>_order_< name:relate_to_context ?context."
           ]},
  {
```

```
"type": "constant",
"n": 3,
"body": [
        "?context rdf:type ?context_class.",
        "?context name:relate_to_article ?article.",
        "?article rdfs:label ?article_name.",
        "?title name:relate_to_article ?article.",
        "?title rdf:type name:cl_title.",
        "?title rdfs:label ?scope.",
        "?out_context name:relate_to_article ?article.",
        "?out_context rdf:type ?out_context_class.",
        "?out_context rdfs:comment ?context_text.",
        "?out_context_class rdfs:label ?topic.",
        "FILTER  ((?out_context_class  =  name:cl_description)
&&",
             "(?context_class = name:cl_title ||",
             "?context_class = name:cl_description ||",
             "?context_class = name:cl_synonyms ))",
             ".} ORDER BY ?scope ?topic"]}],
"outputs": {
"scope": {"sortage": "primary", "verbose": "Scope: "},
"topic": {"sortage": {"by each": "scope"}, "verbose": "Topic:
"},
"context_text": {"sortage": {"by each": "topic"}, "verbose":
""},
"article_name":  {"sortage":  {"group":  "scope"},"verbose":
"Related articles:"}}}
```

In this example, the keys in the dictionary represent the names of the corresponding special semantic categories, here, "description" (description of the specified phenomenon). The values are dictionaries with the following keys:

"verbose" – a phrase fragment preceding the response.

"parts" – a list of sections forming the SPARQL query (the main section).

"outputs" – instructions for structuring the response data obtained during the query execution.

Let's examine the "parts" section in more detail. The value for the key "parts" is a list, each element of which corresponds to a query section. The template for a query section is a dictionary with the following keys:

"type" – the type of the section. The following types are anticipated: "constant" – the section is inserted into the query without modifications; "listed input" – for each of the provided parameters (words), the section is inserted into the query. In this case, the part >input< is replaced with the value of the respective input parameter of the query, and the part >order< is replaced with an incremental integer value (1, 2, 3, ... n), converted to a string type to avoid repetition of the same query variable; "single input" – instead of the part >input<, one parameter is inserted only once, and for the next parameter, if any, a completely new query is generated.

"n" – the order of the section template in query formation to prevent mixing of sections.

"body" – a list of query lines to be formed. The lines may contain placeholders for input parameters >input< and incremental values >order<.

The functioning of the module essentially involves selecting the appropriate templates from the dictionary and concatenating the query sections in the specified order, replacing the corresponding placeholders with input parameters when necessary.

The "outputs" section, like "verbose," is not directly used by the program module but is passed along with the query and is necessary for further response formation.

The execution of SPARQL queries, as well as the storage of the knowledge base, is performed by the Jena Fuseki database management system. Interaction with it is facilitated by the process_queries.ru module using the SPARQL Wrapper library tools. Since the ontology is distributed – divided into parts, each query from the received package is executed separately for each part. Executing SPARQL queries is a relatively slow process. To expedite the process, queries to all ontology parts are executed concurrently in multiple threads. Substantive responses may be obtained from one or several ontology parts. The tables obtained from several ontology parts are merged.

Let us briefly describe the implementation of creating an OWL ontology [10, 12, 13, 14, 15]. To implement the creation of a knowledge base in the form of an OWL ontology in RDF/XML format, special scripts in Python were developed. The process consists of two stages.

1. Automated creation of JSON representation of input article files.

2. Formation of the OWL ontology. At this stage, an OWL ontology is formed using the resulting set of JSON structures. The hierarchical structure of the JSON vocabulary keys forms the basis of the future OWL class system, while the corresponding contextual values become named entities in their respective classes. Each article file name is converted to a named entity in the "Articles" class. The OWL property "Associate with Article" establishes relationships between contexts and the corresponding articles in which they appear. The named entities defined in the contexts are also converted to named entities in the Word class and linked to the corresponding contexts using the "Link to Context" OWL property. This structure allows you to select specific contexts in the ontology using SPARQL queries.

As mentioned above, user queries to a large ontology are too slow to be executed. To speed up the process of obtaining an answer, we consider the possibility of using hardware based on programmable logic integrated circuits [16, 17, 18].

In conclusion, we will present the areas of development of knowledge-oriented systems and their applications that are relevant and promising today from the point of view of the general consideration of scientific knowledge and its effective practical application in the creation of innovative technologies.

Firstly, the analysis of ontological contexts in any subject area makes it possible to build a time trajectory of the process of forming secondary knowledge on the basis of primary knowledge, and thus develop an effective technology for building new knowledge and new innovative technologies based on it.

Secondly, based on the paradigm of transdisciplinary development of science, it is promising to use the functionality described in this paper to form promising clusters of convergence of scientific disciplines and relevant technologies [19].

The immediate tasks may include the formation of an effective toolkit for scientific researchers, including for orientation in the field of their own publications in the subject area and comparison with existing ones in the information space [6, 7].

Undoubtedly, the construction of intelligent reference systems in subject areas (including the aforementioned medical and rehabilitation) is a direct continuation of the research performed by the authors. One of the actual applications of such systems is the creation of comfortable conditions for managing knowledge bases by a wide range of non-professional (in terms of information technology) users.

Finally, we cannot but mention the task of forming a linguistic and ontological picture of the world within the framework of the general evolutionary program and the formation of the planetary consciousness of the modern generation [19].

In addition, it is important to note that [20] deals with the problem of publishing the achievements of Ukrainian scientists under martial law of the Russian armed aggression in ranked journals.

6. Conclusion

The paper considers a method of constructing formal queries to an ontology generated automatically on the basis of a natural language text by a Ukrainian owl. An ontological graph structure is built on the basis of syntactic and semantic relations between concepts in sentences identified during text analysis. Sentence contexts and their parts are also stored in the OWL ontology. These sentences are associated with sets of semantic relations specified by the corresponding entities. The typification of semantic categories is enclosed in a hierarchical structure. This example of an ontology is used to describe the approach to building queries in Cypher. The peculiarity of the approach is that a formal query is automatically built from blocks of templates (main and auxiliary) that are customizable according to the defined semantic categories present in the analyzed text and the entities that specify them.

7. Further research

Our research is far from complete. It is necessary to optimize the procedures for creating and executing SPARQL user queries, for which it is necessary to determine the need to split a large ontology into several parts. At the same time, it is necessary to ensure their parallel execution. In addition, it is necessary to consider the usefulness of developing hardware to speed up the execution of processes and procedures in the system.

8. Acknowledgments

The research was supported by a grant from the National Research Foundation of Ukraine under the project 2021.01/0136 (2022-2024, project in progress) "Development of a cloud-based platform for patient-centered tele-rehabilitation of cancer patients based on mathematical modeling" [21, 22, 23, 24, 25], at the Glushkov Institute of Cybernetics of the National Academy of Sciences of Ukraine, Kyiv, Ukraine.

References

[1] Gomez-Perez A., Fernandez-Lopez M., Corcho O. Ontological Engineering. Advanced Information and Knowledge Processing. Springer-Verlag, London, 1 edition, 2004. ISBN 978-1-85233-551-9.
DOI: 10.1007/b97353.

[2] Studer R. Staab S., editor. Handbook on Ontologies. Springer Berlin Heidelberg, Berlin, Heidelberg, 2 editions, 2009. ISBN 978-3-540-70999-2. DOI: 10.1007/978-3-540-92673-3.

[3] OntoChatGPT information system: Ontology-driven structured prompts for ChatGPT meta-learning / Palagin O., Kaverinskiy V., Litvin A., Malakhov K. International Journal of Computing, 22(2):170–183, July 2023. ISSN 2312-5381, 1727-6209. DOI:10.47839/ijc.22.2.3086.

[4] Ford M. Rule of the Robots: How Artificial Intelligence Will Transform Everything. Basic Books, New York, first edition, 2021. ISBN 978-1-5416-7473-8.

[5] Malakhov, K., Petrenko, M., Cohn, E. (2023). Developing an ontology-based system for semantic processing of scientific digital libraries. South African Computer Journal, 2023 35(1), 19–36.
https://doi.org/10.18489/sacj.v35i1.1219.

[6] Oleksandr Palagin, Mykola Petrenko, Mykola Boyko. Proceedings of the 13th International Scientific and Practical Programming Conference UkrPROG 2022. Kyiv, Ukraine, October 11–12, 2022. URL: https://ceur-ws.org/Vol-3501/s26.pdf.

[7] M.G. Petrenko, O.V. Palagin, M.O. Boyko, S.M. Matveyshyn. Knowledge-Oriented Tool Complex for Developing Databases of Scientific Publications and Taking into account Semantic Web Technology. Control Systems and Computers, 2022, Issue 3 (299), pp. 11–28. DOI: https://doi.org/10.15407/csc.2022.03.011.

[8] Goel A. Neo4J Cookbook. Birmingham: Pact Publishing Ltd. May 28, 2015., 1st Edition. 226 P. ISBN-13: 9781783287253. DOI: https://www.packtpub.com/en-sk/product/neo4j-cookbook-9781783287253?type=print

[9] 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.

[10] A. Litvin, V. Velychko, and V. Kaverinsky. A New Approach to Automatic Ontology Generation from the Natural Language Texts with Complex Inflection Structures in the Dialogue Systems Development, CEUR Workshop Proceedings, 2023, Vol. 3501. pp. 172–185.
https://ceur-ws.org/Vol-3501/s16.pdf.

[11] Kaverinsky, V., Malakhov, K. Natural Language-Driven Dialogue Systems for Support in Physical Medicine and Rehabilitation, South African Computer Journal, 2023, Vol. 35, No. 2, pp. 119 – 126. DOI: https://doi.org/10.18489/sacj.v35i2.17444.

[12] 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.

[13] O.V. Palagin, M.G. Petrenko, S. Yu. Svitla, V.YU. Velychko. About one approach to analyzing and understanding natural language objects. Computer tools, networks and systems. 2008, №7. pp.128–137.
[14] O. Palagin, V. Kaverinsky, A. Litvin, and K. Malakhov. Ontology-driven development of dialogue systems, South African Computer Journal. – Vol. 35. No. 1. – 2023. – P. 37 – 62. DOI: http://dx.doi.org/10.18489/sacj.v35i1.1233.
[15] Petrenko, N.G. Computer ontologies and ontology-driven architecture of information systems. Book "Information Models of Knowledge", ITHEA, Kiev, Ukraine – Sofia, Bulgaria, 2010, pp. 86–92.
[16] Kurgaev, A. F., & Petrenko, M. G. (1995). Processor structure design. Cybernetics and Systems Analysis, 31(4), 618–625. DOI: https://doi.org/10.1007/BF02366417.
[17] Petrenko, M., & Sofiyuk, A. (2003). On one approach to the transfer of an information structures interpreter to PLD-implementation. Upravlyayushchie Sistemy i Mashiny, 188(6), pp. 48–57. https://www.scopus.com/inward/record.uri?eid=2-s2.0-0442276898&partn erID=40&md5=44974b40409363e5fe4378e240149c52
[18] Petrenko, M., & Kurgaev, A. (2003). Distinguishing features of design of a modern circuitry type processor. Upravlyayushchie Sistemy i Mashiny, 187(5), 16–19. https://www.scopus.com/inward/record.uri?eid=2-s2.0-0347622333&partnerID=40&md5=7283307afdf891445ec9062c7b2ff80a
[19] Alexander V. Palagin, Mykola N. Petrenko. Methodological Foundations for Development, Formation and IT-support of Transdisciplinary Research // Journal of Automation and Information Sciences, Volume 50, 2018, Issue 10, PP. 1–17, DOI: https://doi.org/10.1615/JAutomatInfScien.v50. i10.10.
[20] Inefuku, H., Malakhov, K., Cohn, E. R., & Collister, L. B. (2023). Service Diversification, Connections, and Flexibility in Library Publishing: Rapid Publication of Research from Ukraine in Wartime. *Case Studies in Library Publishing*, 1(1). https://cslp.pubpub.org/pub/084se42n/release/1
[21] Malakhov, K. S. (2023a). Insight into the Digital Health System of Ukraine (eHealth): Trends, Definitions, Standards, and Legislative Revisions. International Journal of Telerehabilitation, 15(2), 1–21. DOI: https://doi.org/10.5195/ijt.2023.6599
[22] Malakhov, K. S. (2023b). Letter to the Editor – Update from Ukraine: Development of the Cloud-based Platform for Patient-centered Telerehabilitation of Oncology Patients with Mathematical-related Modeling. International Journal of Telerehabilitation, 15(1), 1–3. DOI: https://doi.org/10.5195/ijt.2023.6562
[23] Palagin, O. V., Malakhov, K. S., Velychko, V. Yu., & 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, 1–16. https://doi.org/10.5195/ijt.2022.6480
[24] Malakhov, K. (2022). Letter to the Editor – Update from Ukraine: Rehabilitation and Research. International Journal of Telerehabilitation, 14(2), 1–2. https://doi.org/10.5195/ijt.2022.6535
[25] O. Palagin, V. Kaverinsky, M. Petrenko, and K. Malakhov, Digital Health Systems: Ontology-based Universal Dialog Service for Hybrid E-rehabilitation Activities Support, The 12th IEEE International Conference on Intelligent Data Acquisition and Advanced

Computing Systems: Technology and Applications. – Dortmund, Germany, 2023. – P. 84 – 89. DOI: http://dx.doi.org/10.1109/IDAACS58523.2023.10348639