Issue 17, 2025

https://doi.org/10.23939/sisn2025.17.170

UDC 004.89

DEVELOPMENT OF A UNIFIED OUTPUT FORMAT FOR TEXT PARSERS IN THE ONTOLOGY CONSTRUCTION SYSTEM FROM TEXT DOCUMENTS

Andrii Chornyi[1], Dmytro Dosyn[2]

[1, 2] Lviv Polytechnic National University,
Information Systems and Networks Department, Lviv, Ukraine,
[1] E-mail: andrii.o.chornyi@lpnu.ua, ORCID: 0009-0007-4005-4088
[2] E-mail: dmytro.h.dosyn@lpnu.ua, ORCID: 0000-0003-4040-4467

The challenge of effectively constructing ontologies from text documents remains unresolved, posing a critical gap in modern knowledge extraction methodologies. One of the primary obstacles is the lack of a standardized output format across different NLP tools, particularly text parsers, which serve as the foundational step in multi-stage knowledge extraction processes. While several widely used text parsers exist, each excels in specific functions, making it beneficial to leverage multiple parsers for more comprehensive ontology construction. However, this approach introduces the issue of reconciling their disparate output formats.

To address this challenge, we propose using a graph database to store parser outputs in a subject-predicate-object triple format, enabling seamless integration and further processing through rule-based transformations using SPARQL queries. A key advantage of this approach is the ability to execute new transformation rules dynamically, allowing for greater flexibility and efficiency in ontology generation.

As part of our research, we developed an intelligent agent in Java capable of constructing semantic graphs from natural language text using a rule-based approach. The agent was employed to evaluate the relationship between the execution time of syntax-semantic transformation rules and variables such as text corpus size and dataset sample dimensions. This evaluation was made possible through the implementation of first-level reflection for the studied transformation rule.

The results demonstrate that our approach – standardizing parser outputs via a graph database – roves effective in terms of both computational complexity and processing speed. By streamlining the ontology construction process, our method paves the way for advanced automated learning of intelligent agents based on textual information, unlocking new possibilities for modern science in the realm of knowledge extraction and representation.

Keywords - natural language processing, ontology, automatic ontology construction, automated learning, syntax-semantic patterns.

Problem statement

In today's information society, most of the knowledge is represented in the form of unstructured texts. The effective extraction of structured information from such sources is critically important for various fields, including science, business, and technology. Natural Language Processing (NLP) methods play a key role in this process, providing tools for analyzing and interpreting textual data. In other words, they open the door to opportunities for automating the process of text comprehension (Shaptala, 2023). This paves the way for the automatic learning of intelligent agents across different subject domains, ranging from general knowledge to highly specialized fields.

One of the promising directions is the use of ontologies for knowledge formalization, which enables structuring information and facilitating its further utilization. Ontologies provide a shared understanding of a subject domain and promote the integration of data from various sources. Therefore, the development of a tool for the automatic generation of ontologies based on unstructured text is a relevant research topic (Hlybovets & Bobko, 2012). In addition to enabling machine interpretation, it also makes automated logical inference possible.

The process of automatically constructing ontologies from unstructured texts faces several challenges. Among them is the lack of unified standards (Lytvyn & Cherna, 2014), particularly the diversity of formats and structures in the source data obtained from various NLP tools, which complicates further analysis. The use of graph databases, such as Apache Jena Fuseki, can facilitate the efficient storage and processing of unified data (in RDF format), which simplifies the construction of syntactic-semantic patterns using SPARQL queries (Haiko, 2023).

Analysis of Recent Studies and Publications

- *Overview of existing Text Parsers.*

The process of automatically constructing an ontology from natural texts is a multi-stage pipeline of methods based on the Methodology of Ontology Learning (Asim, Wasim, Khan, Mahmood, & Abbasi, 2018). In general, these methods can be divided into two groups: natural text processing methods and ontology construction methods [3]. The first group is applied at the initial stage of the process. The author of (Asim, Wasim, Khan, Mahmood, & Abbasi, 2018) logically concludes that achieving higher accuracy in ontology learning requires effective preprocessing of data using high-quality linguistic methods.

The linguistic methods that make up this first group are integrated into text parsers – software components or tools that analyze textual data, breaking it down into structured elements based on predefined rules or formats. Parsers transform unstructured or semi-structured textual data into a more processable format, such as tree structures, arrays, or other data representations.

In modern Natural Language Processing (NLP), there is a wide range of text parsers classified based on different methods and approaches, with a detailed analysis provided in the article (Shvorob, 2015). Generally, parsers can be divided into two main categories: statistical and rule-based. Statistical parsers leverage machine learning techniques to analyze text, while rule-based parsers rely on predefined grammatical rules. Some modern parsers combine both approaches to achieve higher accuracy and efficiency.

Based on data from open Internet sources (Zezula, 2020) (Kumari, 2024), a list of relevant parsers is provided. The relevance of these libraries was verified by checking the date of the last update in their respective GitHub repositories. The criterion for relevance is that the latest commit must have been made no earlier than six months ago, meaning no older than July 2024.

Table 1

List of relevant text parsers

Library	Language	License Type	Commercial use
NLTK	Python	Apache 2.0*	Restricted*
spaCy	Python	MIT	Allowed
scikit-learn	Python	BSD	Allowed
gensim	Python	LGPL	Allowed
TextBlob	Python	MIT	Allowed
CoreNLP	Java	GPL	Restricted**
OpenNLP	Java	Apache 2.0	Allowed
PyTorch	Python	BSD	Allowed
Stanza	Python	Apache 2.0**	Restricted**
GATE	Java	LGPL-3	Allowed

**: NLTK is licensed for non-commercial use, but commercial licenses are available for some corpora.*

***: Stanford provides paid licenses for commercial use of CoreNLP and Stanza.*

The scientific interest among the libraries listed in Table 1 is drawn to Stanford CoreNLP, NLTK, Apache OpenNLP, spaCy, and Gate. Specifically, the study (Schmitt, Kubler, Robert, Papadakis, & LeTraon, 2019) is dedicated to examining these parsers, comparing them as software for Named Entity Recognition (NER). The authors emphasize the importance of NER as a key stage in Natural Language Processing (NLP) systems, which allows for the identification and classification of entities (such as persons, places, etc.) in the text. The primary goal of the study (Schmitt, Kubler, Robert, Papadakis, & LeTraon, 2019) is to objectively assess the performance of these five popular NER systems. The authors felt the need for a replicated study due to significant discrepancies in the results found in other sources. Researchers used two different corpora (CoNLL2003 and GMB) to evaluate the performance of the software.

According to the results of the study (Schmitt, Kubler, Robert, Papadakis, & LeTraon, 2019), Stanford CoreNLP demonstrated the most stable performance, surpassing other programs by 15-30%. However, there are significant discrepancies between the results of this and other studies, which can reach up to 66%. The summarized results of the study, based on the criteria of performance, cost, documentation, and the ability to recognize major entity types (person, organization, location), are presented in Table 2.

Table 2

The summarized results of the experimental study (Schmitt, Kubler, Robert, Papadakis, & LeTraon, 2019)

Parameter	Stanford CoreNLP	NLTK	OpenNLP	SpaCy	Gate
Performance	Best	Moderate	Moderate	High	Moderate
License	GNU GPL	Apache 2.0	Apache 2.0	MIT	GPL v3
Documentation	Comprehensive	Wide	Limited	Comprehensive	Limited
Entity Support	PER, ORG, LOC	PER, ORG, LOC	PER, ORG, LOC	PER, ORG, LOC + GPE	PER, ORG, LOC

In the work (Dosyn, Daradkeh, Kovalevych, Luchkevych, & Kis, 2022), a similar list of parsers is discussed as the most relevant, supplemented by the Link Grammar Parser library, which was initially built in ANSI C as the formal grammar system Link Grammar. During its development, it became part of the Java RelEx (RelEx Dependency Relationship Extractor) distribution. The authors of this work provide additional characteristics of the parsers, which play an important role in the development and research of automatic ontology construction from text, specifically algorithms, classifiers, and the presence of the OWL API.

Table 3

Additional parser parameters presented in the work (Dosyn, Daradkeh, Kovalevych, Luchkevych, & Kis, 2022).

Parameter	Stanford CoreNLP	NLTK	OpenNLP	SpaCy	Gate
Algorithms	CRF	Max. entropy	Max. entropy	Neural (2.0)	JAPE
Classifiers	CoNLL, MUC6-7, ACE	StanfordNER		OntoNotes	
OWL API	+	–	+	-	+

The authors of the article (Basaraba, Bets, & Bets, 2024), which discusses the issues of recognizing and decoding phraseological units, also consider these five libraries as NLP tools for solving the task they set, briefly presenting their general functional characteristics.

A detailed assessment of the functional capabilities of each of the parsers listed above was conducted through an analysis of information from open sources, including the official resources of each library (Stanford CoreNLP website, n.d.) (NTLK website, n.d.) (Apache Open NLP Website, n.d.) (spaCy website, n.d.) (GATE website, n.d.). For convenience of evaluation, the results of this analysis are presented in the comparative table below.

Table 4

Comparative table of the functional capabilities of text parsers.

NLP Component	Stanford CoreNLP	NLTK	OpenNLP	SpaCy	GATE
Tokenization	Yes	Yes	Yes	Yes	Yes
Sentence splitting	Yes	Yes	Yes	Yes	Yes
POS tagging (Part-of-Speech)	Yes	Yes	Yes	Yes	Yes
Lemmatization	Yes	Yes	–	Yes	Yes
Stemming	–	Yes	–	–	Yes
Named entity recognition (NER)	Yes	Yes	Yes	Yes	Yes
Syntactic dependency parsing	Yes	Yes	–	Yes	Yes
Semantic role labeling	Yes	–	–	–	–
Chunking (Phrase grouping)	–	Yes	Yes	–	Yes
Word vectorization (Word Embeddings)	–	–	–	Yes	–
Coreference resolution	Yes	–	–	–	Yes
Sentiment analysis	Yes	–	–	Yes	-
Ability to train custom models	Yes	Limited	Yes	Yes	Yes

Based on the data presented in Table 4, it can be concluded that none of the parsers provides a complete spectrum of text analysis. On the other hand, returning to the previously mentioned quote from the author (Asim, Wasim, Khan, Mahmood, & Abbasi, 2018), for higher accuracy in ontology learning, effective preprocessing of data is necessary. Greater efficiency in data preprocessing can be achieved through a more comprehensive text analysis. Therefore, an important task for the ontology construction system from text is to ensure the capability of processing output data from different parsers.

- *Overview of existing approaches to ontology construction from text.*

The main idea of this research was formed based on the CROCUS project (Cognition of Relations Over Concepts Using Semantics), described in the works (Dosyn, Daradkeh, Kovalevych, Luchkevych, & Kis, 2022) (Dosyn & Lytvyn, 2021). This project is dedicated to the automated learning of ontologies from text. Its main goal is to develop a system capable of automatically extracting semantic relationships in texts and forming hierarchical knowledge structures for specific users. The authors emphasize the assessment of the relevance of information specifically for the individual user. The proposed CROCUS project is built in Java using the Link Grammar Parser and WordNet API. The research (Dosyn, Daradkeh, Kovalevych, Luchkevych, & Kis, 2022) outlines two main approaches to detecting semantic relationships in texts, namely:

analysis of sentence component trees to identify explicit relationships;

use of a naive Bayes classifier to detect implicit semantic relationships, which requires a deeper analysis of sentence parts.

In the work (Mousavi, Kerr, Iseli, & Zaniolo, 2014), the OntoHarvester system is proposed, which presents a new (at the time) approach to automatically creating domain ontologies from a small corpus of texts using deep NLP analysis. The OntoHarvester system begins with a small set of concepts (seeds) and iteratively expands the ontology by adding new terms that have strong semantic connections with the existing concepts. According to the authors, this approach allows for the creation of comprehensive ontologies from small text corpora, while remaining resilient to noise and focused on a specific domain. The OntoHarvester work is based on the following steps:

Construction of TextGraphs – using the SemScape system to create graphs that represent grammatical relationships between terms in the text.

Extraction of ontological relationships – using graph templates (GD-rules) to identify relationships such as part_of and type_of.

Extraction of new concepts – adding new terms to the ontology if they have strong connections with existing concepts.

Detection of new types of relationships – identifying new types of relationships between existing concepts to further expand the ontology.

According to the research findings, the authors (Mousavi, Kerr, Iseli, & Zaniolo, 2014) claim that their proposed approach achieves higher accuracy and coverage compared to other methods and demonstrates resilience to noise in the text while being able to create complex ontologies from small text corpora. This result is achieved using GD-rules (Graph Domain Rules or GD Rules).

According to the authors (Mousavi, Kerr, Iseli, & Zaniolo, 2014), GD-rules are more powerful compared to traditional approaches that use tree-like templates or regular expressions. They allow for accounting for complex grammatical structures in the text, significantly improving the accuracy of extracting ontological relationships. Moreover, unlike statistical methods, GD-rules do not require large volumes of data for training, making them more practical for working with small text corpora.

A similar inductive approach based on linguistic patterns is discussed in the article (Doroshenko, 2018) for extracting factual information. The authors present some basic correspondences between linguistic templates and their ontological interpretation in canonical form.

Another relevant practical idea and its research, related to the field of ontology construction from text, are presented in the article (Zlatareva & Amin, 2021). The authors propose the creation of a question-answering (QA) system, built on the principle of converting queries formulated in natural language into SPARQL queries, which are used in semantic web applications to retrieve data from graph databases. A vast number of such resources are interconnected through the Linked Open Data Cloud (Linked Open Data Cloud, без дати) and provide users with direct access to thousands of RDF/RDFS datasets via SPARQL endpoints. The main problem addressed by this work lies in the difficulty for regular users to use the SPARQL query language.

The proposed methodology involves converting an unstructured natural language question into structured constructs that can be processed by a machine, like previous works, using a rule-based approach.

The use of machine learning (ML) systems, such as BERT (Devlin, Chang, Lee, & Toutanova, 2019), is another approach widely applied in ontology building systems from unstructured text, often in combination with rule-based approaches.

The importance of ML components in solving the task of ontology building from text is also confirmed by the authors of (Basaraba, Bets, & Bets, 2024), who in their study attempt to address the issue of recognizing and decoding phraseological units. This problem is more related to general NLP, but it can serve as a foundation for recognizing indirect statements made in a specific context, applicable to a wide range of NLP applications.

Formulation of the Article's Objective

The aim of this work is to unify the output data format of text parsers within an ontology construction system from textual documents and explore approaches for the automatic extraction of semantic relationships from unstructured texts using syntactic-semantic patterns (Vovnianka, Dosyn, & Kovalevych, 2014) (Mousavi, Kerr, Iseli, & Zaniolo, 2014). Additionally, it seeks to address the problem of the lack of unified standards by integrating syntactic parsing results into graph databases (RDF) and leveraging SPARQL queries to identify and apply syntactic-semantic patterns.

Main Results

- *Selection of technologies for the ontology construction system.*

Based on the data presented in Table 4 and the core idea of continuing the CROCUS project (Dosyn, Daradkeh, Kovalevych, Luchkevych, & Kis, 2022) (Dosyn & Lytvyn, 2021), the development and research of an ontology construction system from text have been initiated using the Stanford CoreNLP text parser (Stanford CoreNLP website, n.d.). This choice is driven by factors such as relevance, functional completeness, and the technological stack, particularly the use of Java. Another crucial factor in selecting this parser is the comprehensiveness of its documentation, as noted in Table 2.

Additionally, the findings of the comparative study (Nanavati & Ghodasara, 2015) were considered, which analyzed two popular natural language processing (NLP) tools – Stanford NLP and Apache OpenNLP – focusing on their efficiency in part-of-speech (POS) tagging. According to results of the study (CoreNLP vs Apache OpenNLP, n.d.), if accuracy and speed are the priority, Stanford NLP is the better choice. Publicly available sources indicate that this preference is widely supported by users. As stated in (CoreNLP vs Apache OpenNLP, n.d.), at the time of writing this article, the popularity of Stanford NLP significantly exceeded that of Apache OpenNLP, with a ratio of 9.2 to 6.8.

The analytical review (Manning, et al., 2014) was also taken into account, in which the authors highlight the strengths of Stanford NLP, namely: ease of use, a universal interface, high-quality analysis, flexibility, support for multiple languages, and open-source availability.

According to the classification proposed by the authors (Yunchyk, Kunanets, Pasichnyk, & Fedoniuk, 2021), the formal task of developing a text-based ontology construction system is to create *an artificial, virtual, reactive intelligent agent with a learning function* based on unstructured texts.

From a software architecture perspective, such a system is a modular intelligent agent based on the principles and analysis proposed in (Chornyi, 2024), which explores the principles of building intelligent agents, emphasizing their adequacy and functionality.

In order to implement multi-agent capability (Chornyi, 2024), two communication interfaces were created using widely known modern protocols: REST API and SPARQL. These interfaces can be used for both agent-to-agent and human-to-agent communication. This approach enabled the development of a multi-agent system with either hierarchical (command-subordinate) or peer-to-peer relationships between agents.

The REST API module is implemented using the Java framework Spring Boot, which is widely adopted and has comprehensive documentation.

The SPARQL endpoint is implemented using the Apache Jena Fuseki module, which serves as a SPARQL server. In this case, it functions as an in-memory server for storing and processing the agent's (ontology-building system's) operational graph data. Apache Jena Fuseki enables the creation of a standalone server, which can later be used for permanent data storage and for handling large volumes of data. This provides an alternative to Neo4J, as proposed in (Chornyi, 2024).

All these modules are integrated into a multi-tier modular architecture, allowing seamless expansion of the agent's functionality by adding new modules and layers.

The task of integrating multiple NLP modules into a single system faces the problem of the lack of unified standards (Lytvyn & Cherna, 2014), the solution to which, within the context of building an ontology from text, is the unification of the output data format of NLP systems.

- *Solution to the problem of unifying the output data of text parsers.*

As mentioned earlier, based on the data in Table 4, it is evident that for the most accurate semantic analysis of text, it makes sense to use multiple parsers. Additionally, comparing the results of the same analysis from different parsers will provide a higher reliability score for the outcome.

The idea of integrating different parsers into the software code of a single intelligent agent has led to the problem of processing their output data, as the data representation formats in different parsers vary. Since the CROCUS system's technological stack is based on the Java programming language, below is an overview table of the data structures of the CoreNLP, OpenNLP, and GATE text parsers, as potential candidates for modular integration into the developing intelligent agent system.

Table 5

Data Structures (Classes) of Text Parsers in Java.

Parser Output Data	CoreNLP	OpenNLP	GATE
Text Document	CoreDocument	SentenceSample	Document
1	2	3	4
Text Annotations: boundaries, type, and additional features	Annotation		AnnotationSet
Tokens with annotations: parts of speech, lemmas, named entities, etc.	CoreLabel	POSSample	Annotation
Attributes and characteristics of annotations in key-value format	CoreMap	FeatureMap	FeatureMap
Dependency graph between words in a sentence	SemanticGraph	–	–
Tree structure representing sentence syntax parsing	Tree(Constituency Parse)	Parse	-
Represents extracted triples from text: subject, predicate, object	RelationTriple (OpenIE)	–	–
Representation of relations between words referring to the same entity, i.e., for coreference resolution	CorefChain	–	–

Continuation of Table 5

1	2	3	4
Marks the boundaries of a text substring corresponding to a certain annotation	Span	Span	-
Marks the affiliation to a phrase (named entity or noun phrase)	–	ChunkSample	–
Represents an event with its characteristics	–	Event	–
Represents a sequence of elements (e.g., tokens and their tags) for storage and further processing	–	Sequence	–
Represents a collection of text documents that may contain annotations and are used for text processing and analysis	–	–	Corpus
Represents an individual element of text (e.g., word or character) that is part of a document and may have associated annotations	–	–	Node
Contains the textual content of the document	–	–	DocumentContent

Working with the representations of output data from text parsers presented in Table 5 is somewhat complicated by the lack of unification – all this data is objects of Java classes, not united by an abstract structure. Specifically, each NLP tool (CoreNLP, OpenNLP, GATE) uses its own data structure, which does not always align with others. For example, CoreNLP has Annotation, GATE has AnnotationSet, and OpenNLP does not have a clear equivalent. This complicates the integration of results from different parsers and their interaction with other systems.

Moreover, these structures are of different types and often represent tree-like or list-based data constructs, which makes them rigidly tied to a specific order and hierarchy of data. For example, a syntactic tree (Tree or Parse) stores the structure of a sentence, which can be difficult to use for deeper analysis, especially when there is a need to analyze other data or relationships simultaneously.

Since the data in this representation are Java objects, the use of rule-based approaches for ontology construction from text is complicated by excessive dependence on the code of the intelligent agent program, thereby nearly making automatic learning impossible due to the need for recompilation.

Serialization into XML or JSON format could partially solve the problem of data storage and transmission between different modules or agents (actors), but it does not provide sufficient flexibility. This approach merely "freezes" the structure in its current form, without allowing for dynamic changes or transformations. Furthermore, for further processing of such data, deserialization is required, which necessitates that each actor knows the structure of all the data, or serialization must be performed with conversion into a single format.

The serialization operation has the following mathematical representation:

$$S(o) = f(o) \rightarrow D,$$

where o – object; $f(o)$ – transformation function into the data format D (e.g., JSON), which is typically linear in complexity $O(n)$, where n – size of the object, or the number of elements (objects or data structures) that need to be serialized or deserialized.

Deserialization represents the inverse operation:

$$o' = f^{-1}(D),$$

where o' – reconstructed object; $f^{-1}(D)$ – inverse deserialization operation.

The complexity of the serialization-deserialization approach is as follows:

Serialization: $O(n)$, where n is the number of elements that need to be processed.

Deserialization: $O(n)$, since all object fields must be read and reconstructed.

Searching for relationships between entities: $O(n^2)$ or even $O(n^3)$, as the entire volume of deserialized data needs to be analyzed.

Thus, the overall complexity of this approach is as follows:

Optimistic scenario: $O(n) + O(n) + O(n^2) = \boldsymbol{O(n^2)}$.

Pessimistic scenario: $O(n) + O(n) + O(n^3) = \boldsymbol{O(n^3)}$.

Thus, in addition to the aforementioned limitations, this approach has several drawbacks related to the complexity of algorithms, which are further exacerbated by data storage challenges due to the need for full deserialization during searches and complete file rewrites even for partial modifications. Converting text parser data into a graph using a graph database (e.g., Apache Jena Fuseki or Neo4j) provides a way to overcome these drawbacks.

This paper proposes representing elements such as words, entities, and annotations as graph nodes, while relationships between them are represented as edges. This approach enables a more natural representation of non-linear connections (e.g., dependencies between words in different parts of a sentence or coreference between entities). The graph-based representation of text parser data thus serves as a unified data format that is comprehensible to all actors, as it is fundamentally based on a simple subject-predicate-object triple structure.

The main operations when working with a graph are:

adding a node

$$\mathrm{V}_i = \{A_1, A_2, \dots, A_k\},$$

where $A_1, A_2, \dots, A_k$ – attributes of the node (e.g., token, POS tag), which can also be represented as nodes.

adding an edge

$$E_{ij} = \left(V_i, V_j, R\right),$$

where V_i, V_j – nodes; R – type of relationship (e.g., syntactic dependency).

searching for a connection

$$T\left(V_i, V_j\right) = \min_{P} \sum_{(V_k, V_m) \in P} \omega_{km},$$

where P – path between V_i, V_j, and ω_{km} – weight of the edge.

The complexity of the graph-based approach is as follows:

Adding a node or edge: $O(1)$

Searching for neighboring nodes: $O(1)$ (on average, for a well-structured graph).

Searching for relationships between entities: $O(\log n)$ or $O(d)$,
where n – number of nodes; d – degree of the node:

$$d = \frac{2E}{V},$$

V – total number of nodes (vertices) in the graph;

E – total number of edges (connections) in the graph.

Thus, a graph database (DB) provides faster access to information, reducing the complexity of searching from $O(n^2)$ (during deserialization) to $O(\log n)$ or even $O(1)$, depending on the structure of the graph. Therefore, the overall complexity of this approach will be:

$$MAX\big(O(\log n), O(1)\big).$$

On the other hand, serialization-deserialization is suitable for data storage and transmission but is inefficient for analysis and quick access.

Furthermore, a graph DB allows for easy formulation, storage, and application of rules in a rule-based approach to ontology construction from text, where a rule can be represented in the following form:

$$q_{sel}(G_{syn}) \rightarrow q_{ins}(G_{sem}),$$

where q_{sel}, q_{ins} – queries (select, insert respectively) to the graph database; G_{syn} – syntactic relationship graph, and G_{sem} – semantic relationship graph.

A set of rules $Q = \{Q_1, Q_2, \dots, Q_k\}$, where $Q_i = (q_{sel\,i}, q_{ins\,i})$ can easily be stored, transmitted, and applied, as the rules themselves are textual data in well-known formats, such as SPARQL or Cypher. Moreover, one of the most important aspects is that the graph DBMS, like other DBMSs, serves as a query interpreter. Therefore, adding new rules does not require any compilation and can be done "on the fly" during the functioning of the intelligent agent. This feature opens up wide opportunities for the automatic learning of the ontology construction system from text.

The graph structure also facilitates the training of neural networks, which can be used as an ML module in the intelligent agent. This is because vector representations can be easily formed (e.g., through GNN – Graph Neural Networks) and algorithms can be applied to find connections, hidden patterns, clustering, or inductive learning.

Thus, the use of a graph database enables the unification of data represented in different structures within a single parser, as well as data from different parsers (CoreNLP, OpenNLP, GATE), even those implemented in a different technological stack, such as NLTK and SpaCy.

- *The implementation of a rule-based approach for ontology construction from text using a syntactic graph.*

After the output data from the text parsers was converted to the RDF graph format using a graph database, the rule-based approach for ontology construction from text represents a typical ETL (Extract, Transform, Load) process, as shown in **Error! Reference source not found.**.

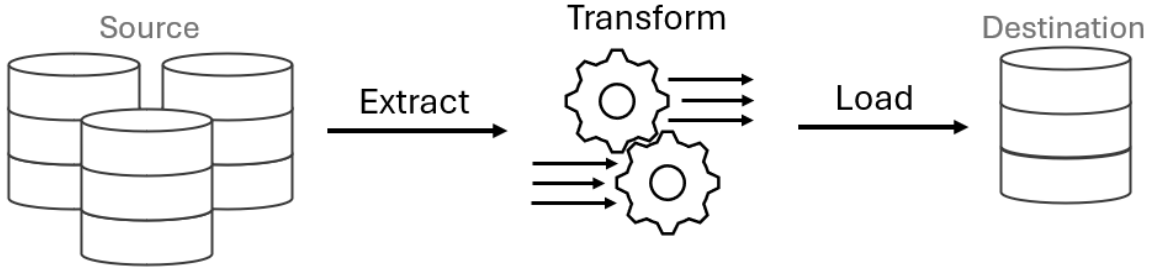

Fig. 1. Typical ETL process diagram.

The formalization of syntactic-semantic rules in the form of ETL processes opens new horizons in the research and development of intelligent agents capable of learning from unstructured text documents.

In particular, the division of ETL stages at the application level of the intelligent agent enables the evaluation of each rule by parallelizing the data recording processed at each of these stages, as shown in Fig. 2. This, in turn, opens opportunities for analyzing the effectiveness of each rule without increasing the complexity of the main algorithm, keeping it at the level of $MAX(O(\log n), O(1))$.

From an epistemological point of view, this approach represents an implementation of the first-level reflection of the intelligent agent, which in this case is expressed in the agent's ability to evaluate its own activities, specifically the process of constructing an ontology from text.

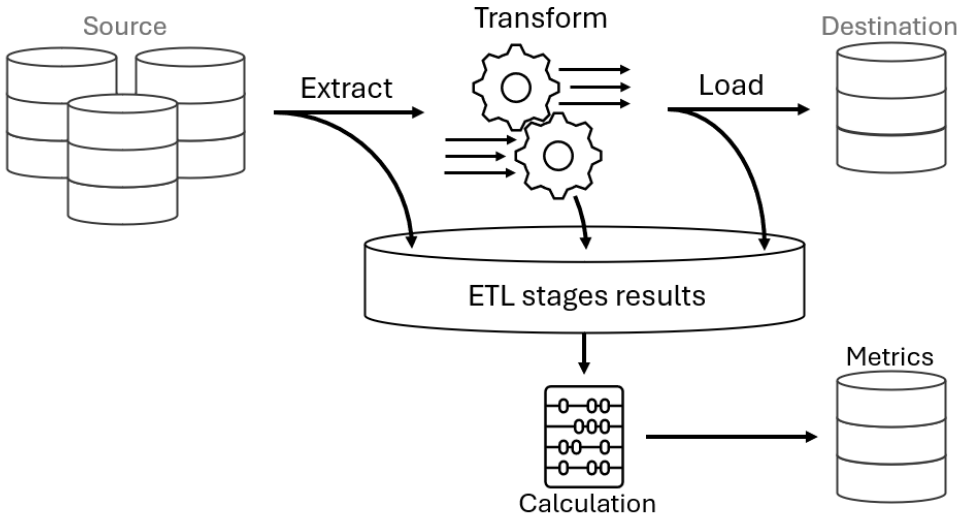

Fig. 2. ETL diagram of the rule-based approach with first-level reflection.

In order to analyze this approach, a study of the syntactic-semantic rule was conducted with the following logic:

if there is a "compound" relationship between nodes in the syntactic graph, and the subject and object of this relationship are connected to the literal "PERSON" via the relationship NamedEntityTagAnnotation, **then** create a node of type "person" in the semantic graph. The name of this node will be the subject of the "compound" relationship, and the surname will be the object of the "compound" relationship. The result for the sentence "Alice Johnson prepared a detailed report for the upcoming meeting." is shown in Fig. 3.

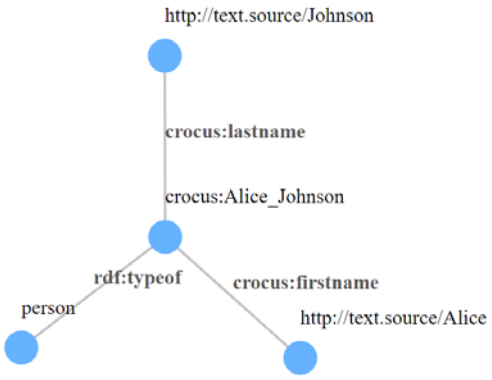

Fig. 3. The result of applying the person extraction rule to a sentence.

This rule is implemented at the graph database level using the SPARQL query provided below. The relative simplicity of this rule allowed for its execution without dividing the ETL processes at the application level. This implementation enabled the analysis to be conducted without the technological delays that would have been introduced by Java code execution.

In the provided SPARQL code for the "person" entity extraction rule, the ETL processes (Extract, Transform, Load) and the reflection blocks are highlighted with comments.

```
PREFIX crocus: http://crocus.science/
DROP GRAPH <http://localhost:3330/opgraph/statrule>;
INSERT
{
//----- LOAD -----
 GRAPH <http://localhost:3330/opgraph/semgraph>
 {
       ?fulln ?firstName ?fn .
       ?fulln ?lastName ?ln .
       ?fulln ?rdfType ?rdfPerson .
 }
//----- Reflection -----
 GRAPH <http://localhost:3330/opgraph/statout>
 {
       ?fulln ?firstName ?fn .
       ?fulln ?lastName ?ln .
       ?fulln ?rdfType ?rdfPerson .
 }
 GRAPH <http://localhost:3330/opgraph/statin>
 {
       ?subject ?compound       ?object .
       ?subject ?namedEntityTag ?parserTypePerson .
       ?object  ?namedEntityTag ?parserTypePerson .
 }
 GRAPH <http://localhost:3330/opgraph/statrule>
 {
       ?ruleId ?ruleStart ?startTime .
 }
}
WHERE
{
 GRAPH <http://localhost:3330/opgraph/syngraph>
 {
       BIND(NOW() AS ?startTime)
//----- EXTRACT -----
       BIND(<http://nlp.stanford.edu#compound> AS ?compound)
       BIND(<http://nlp.stanford.edu#NamedEntityTagAnnotation> AS
?namedEntityTag)
       BIND("PERSON" AS ?parserTypePerson)

       ?subject ?compound       ?o .  BIND(IRI(?o) AS ?object)
       ?subject ?namedEntityTag ?parserTypePerson .
       ?object  ?namedEntityTag ?parserTypePerson .

//----- TRANSFORM -----
       BIND(IRI(crocus:PersonFirstLastNameRule) AS ?ruleId)
       BIND(IRI(crocus:startTime) AS ?ruleStart)
       BIND(<rdf:typeof> AS ?rdfType)
       BIND("person" AS ?rdfPerson)
       BIND(IRI(CONCAT("crocus:", REPLACE(STR(?object), ".*[/#]", ""),
"_",  REPLACE(STR(?subject), ".*[/#]", ""))) AS ?fulln)
       BIND(<crocus:firstname> AS ?firstName)
       BIND(<crocus:lastname> AS ?lastName)
       BIND(?object  AS ?fn)
```

```
        BIND(?subject AS ?ln)
  }
};
//----- Reflection -----
INSERT
{
  GRAPH <http://localhost:3330/opgraph/statrule>
  {
        ?ruleId ?ruleEnd ?endTime .
        ?ruleId ?ruleDuration ?duration .
  }
}
WHERE
{
  GRAPH <http://localhost:3330/opgraph/statrule>
  {
        ?rule ?ruleStart ?startTime .
  }
  BIND(IRI(crocus:PersonFirstLastNameRule) AS ?ruleId)
  FILTER(?rule = ?ruleId)
  FILTER(?ruleStart = IRI(crocus:startTime))
  BIND(IRI(crocus:endTime) AS ?ruleEnd)
  BIND(IRI(crocus:duration) AS ?ruleDuration)
  BIND(NOW() AS ?endTime)
  BIND((?endTime - ?startTime) AS ?duration)
};
```

In scope of this SPARQL script, writing of input and output data of the rule into the graphs *statin* (Extract stage) and *statout* (Load stage) is being performed, as shown in Fig. 2, along with the execution duration of the rule into the *statrule* graph (Transform stage).

The study was started with a text corpus of 20 sentences, adding one sentence at a time from 1 to 20. However, this volume of text did not lead to significant changes in the execution time of the rule, sufficient for analysis. Therefore, a nonlinear increment was applied, doubling the number of sentences in the text corpus at each subsequent step up to 640 sentences. To better understand the actual volume of the text corpus, it should be noted that 80 sentences correspond to approximately one page in A4 format with 1 cm margins on all sides, using 12 pt Times New Roman font.

The text data was generated using an LLM system with the following requirements: each sentence must mention one person in the format of a combination of <FirstName LastName>, each of which (combination) should be unique.

As a result of the experiment, the performance characteristics of the person entity extraction rule, presented above in the SPARQL script, were obtained (Table 6).

Table 6

The dependence of the execution time of the rule on the quantitative characteristics of the working graphs.

N_{sent}	T_{inc}	T_{full}	V_{syn}	E_{syn}	$V_{EXTRACT}$	$E_{EXTRACT}$	V_{LOAD}	E_{LOAD}
1	0.026	0.022	83	251	3	3	4	3
2	0.003	0.027	147	463	5	6	7	6
3	0.002	0.021	218	692	7	9	10	9
4	0.003	0.020	282	915	9	12	13	12
5	0.003	0.029	342	1116	11	15	16	15
6	0.003	0.028	391	1280	13	18	19	18

Continuation of Table 5

1	2	3	4	5	6	7	8	9
7	0.005	0.038	466	1518	15	21	22	21
8	0.004	0.031	521	1707	17	24	25	24
9	0.004	0.041	575	1894	19	27	28	27
10	0.004	0.028	633	2090	21	30	31	30
11	0.005	0.029	690	2282	23	33	34	33
12	0.003	0.042	743	2454	25	36	37	36
13	0.004	0.038	814	2684	27	39	40	39
14	0.004	0.031	867	2849	29	42	43	42
15	0.003	0.039	915	3007	31	45	46	45
16	0.002	0.042	978	3206	33	48	49	48
17	0.003	0.047	1049	3436	35	51	52	51
18	0.002	0.048	1105	3627	37	54	54	55
19	0.003	0.037	1169	3834	39	57	58	57
20	0.003	0.043	1233	4046	41	60	61	60
40	0.005	0.049	2306	7456	77	116	117	120
80	0.010	0.059	4327	13237	96	176	177	243
160	0.013	0.078	7689	21790	100	255	256	468
320	0.064	0.100	13556	35359	102	384	385	849
640	0.088	0.133	31732	76519	147	567	568	1263
	$r(T_{full})$		0.93	0.95	0.93	0.98	0.98	0.96
	$r(T_{inc})$		0.93	0.93	0.73	0.91	0.91	0.94

N_{sent} – the number of sentences in the text corpus;

T_{inc} – the execution time of the rule (script) in the incremental sentence adding mode;

T_{full} – the execution time of the rule (script) in the full text corpus adding mode;

V_{syn} – the total number of nodes in the syntactic graph;

E_{syn} – the total number of edges (relations) in the syntactic graph;

$V_{EXTRACT}$ – the number of nodes extracted by the rule from the syntactic graph during the Extract stage;

$E_{EXTRACT}$ – the number of edges (relations) extracted from the syntactic graph during the Extract stage;

V_{LOAD} – the number of nodes stored by the rule in the semantic graph during the Load stage;

E_{LOAD} – the number of edges (relations) stored by the rule in the semantic graph during the Load stage;

$r(T_{full})$ – the Pearson linear correlation vector between T_{full} and the parameters V_{syn}, E_{syn}, $V_{EXTRACT}$, $E_{EXTRACT}$, V_{LOAD}, and E_{LOAD};

$r(T_{inc})$ – the Pearson linear correlation vector between T_{inc} and the parameters V_{syn}, E_{syn}, $V_{EXTRACT}$, $E_{EXTRACT}$, V_{LOAD} та E_{LOAD}.

The correlation between the execution time and the quantitative parameters of the working graphs of the syntactic-semantic transformation was calculated separately, based on the data from Table 6, and is not part of the rule. This indicator was calculated additionally to better understand which specific characteristics of the working graphs and queries have a greater impact on the performance of the syntactic-semantic rules in the proposed approach.

The correlation values indicate that the execution time of the rule is directly dependent on the size of the working graphs and the samples. However, when the entire text corpus is loaded at once and the rule is executed, the execution time is more strongly dependent on the number of edges (connections) in the sample

$E_{EXTRACT}$ at the Extract stage (pattern matching in the syntactic graph). In contrast, during incremental analysis, which involved gradually increasing the size of the text corpus by adding new sentences to the existing ones and executing the rule at each stage, a stronger dependency of the execution time was observed on the number of edges (relations) in the sample E_{LOAD} at the Load stage (inserting into semantic graph).

This pattern indicates the presence of a caching function in the graph database server used in the study (Apache Jena Fuseki), as, with the incremental increase in the text corpus volume, the syntactic analysis data from the previous step were not deleted. Accordingly, the data of the query at the Extract stage of the investigated rule at each step (except for the first one) was partially cached.

Analyzing this information on the charts shown in Fig. 4, a sharp increase (by 5 times) in the execution time of the transformation (rule) was observed after the text corpus was increased from 160 sentences to 320. This clearly indicates a cache memory overflow in the graph database server.

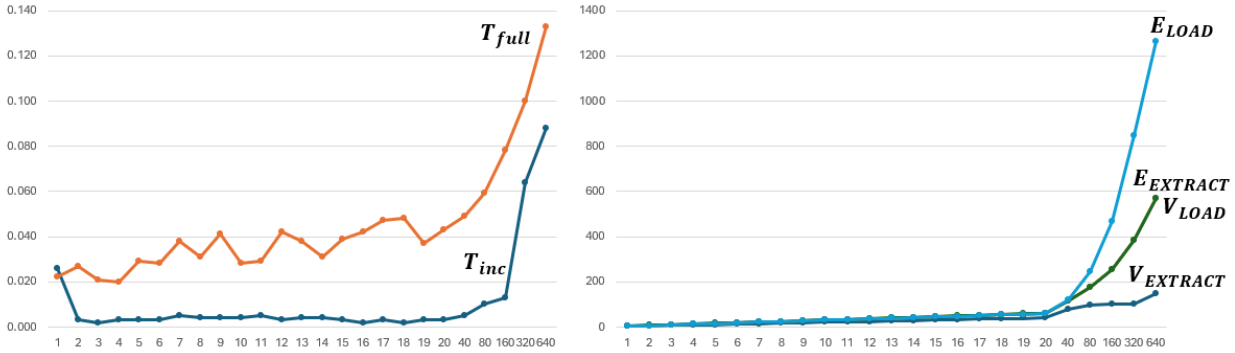

Fig. 4. The charts of the results of the person extraction rule execution (Table 6).

To further investigate the impact of data caching in the graph database server on the execution time of syntactic-semantic rules, an additional study was conducted. The essence of this study was to analyze the change in execution time during repeated rule executions. The results of this study are presented in Table 7. The research algorithm involved repeating the rule execution ten times, measuring the time after its initial execution on the freshly loaded syntactic graph for different text corpus volumes.

Table 7

The impact of caching on the repeated execution of the syntactic-semantic rule.

N_{sent}	T_{full}	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}
1	0.022	0.005	0.003	0.003	0.003	0.005	0.003	0.001	0.003	0.002	0.001
2	0.027	0.003	0.004	0.002	0.002	0.002	0.003	0.002	0.002	0.002	0.002
3	0.021	0.005	0.003	0.002	0.003	0.002	0.004	0.003	0.005	0.002	0.003
4	0.020	0.004	0.003	0.004	0.003	0.002	0.002	0.003	0.002	0.002	0.002
5	0.029	0.005	0.004	0.003	0.004	0.004	0.001	0.003	0.004	0.003	0.004
6	0.028	0.009	0.006	0.004	0.005	0.004	0.007	0.003	0.005	0.005	0.003
7	0.038	0.008	0.005	0.006	0.005	0.004	0.003	0.005	0.003	0.003	0.005
8	0.031	0.006	0.005	0.005	0.005	0.005	0.004	0.005	0.003	0.003	0.003
9	0.041	0.006	0.006	0.005	0.005	0.003	0.004	0.003	0.006	0.005	0.005
10	0.028	0.008	0.006	0.004	0.003	0.005	0.004	0.004	0.005	0.005	0.003
11	0.029	0.006	0.004	0.005	0.004	0.004	0.004	0.004	0.004	0.004	0.004
12	0.042	0.011	0.007	0.007	0.005	0.005	0.005	0.005	0.005	0.003	0.003
13	0.038	0.010	0.005	0.006	0.005	0.004	0.005	0.004	0.005	0.004	0.003
14	0.031	0.006	0.006	0.005	0.004	0.003	0.003	0.005	0.004	0.004	0.004
15	0.039	0.009	0.006	0.005	0.004	0.003	0.004	0.004	0.004	0.004	0.003

Continuation of Table 5

1	2	3	4	5	6	7	8	9	10	11	12
16	0.042	0.009	0.013	0.008	0.003	0.006	0.007	0.006	0.002	0.004	0.003
17	0.047	0.011	0.005	0.005	0.004	0.006	0.005	0.004	0.003	0.005	0.003
18	0.048	0.008	0.007	0.007	0.006	0.005	0.003	0.004	0.003	0.003	0.002
19	0.037	0.008	0.006	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.003
20	0.043	0.009	0.007	0.007	0.004	0.005	0.009	0.005	0.006	0.004	0.004
40	0.049	0.016	0.011	0.012	0.011	0.009	0.005	0.006	0.007	0.005	0.006
80	0.059	0.019	0.020	0.012	0.010	0.011	0.013	0.011	0.009	0.008	0.009
160	0.078	0.034	0.030	0.023	0.023	0.020	0.019	0.016	0.016	0.016	0.015
320	0.100	0.041	0.033	0.030	0.031	0.025	0.027	0.022	0.024	0.025	0.022
640	0.133	0.054	0.043	0.043	0.034	0.035	0.033	0.032	0.031	0.030	0.031

N_{sent} – the number of sentences in the text corpus;

T_{full}– the execution time of the rule after the addition of the text corpus and its initial execution;

$T_1 \ldots T_{10}$ – the execution time of the rule during each subsequent repetition attempt.

As shown in Table 7, each subsequent execution of the rule is faster until a certain minimum is reached, which further confirms the presence of a query caching function that helps optimize rule-oriented methods for ontology construction from text. For clarity, a chart depicting the change in execution time depending on the number of repetitions is provided. The chart includes data on the nonlinear increase in the volume of the text corpus (20, 40, 80, 160, 320, and 640 sentences), as this is where the time deviation is the most noticeable.

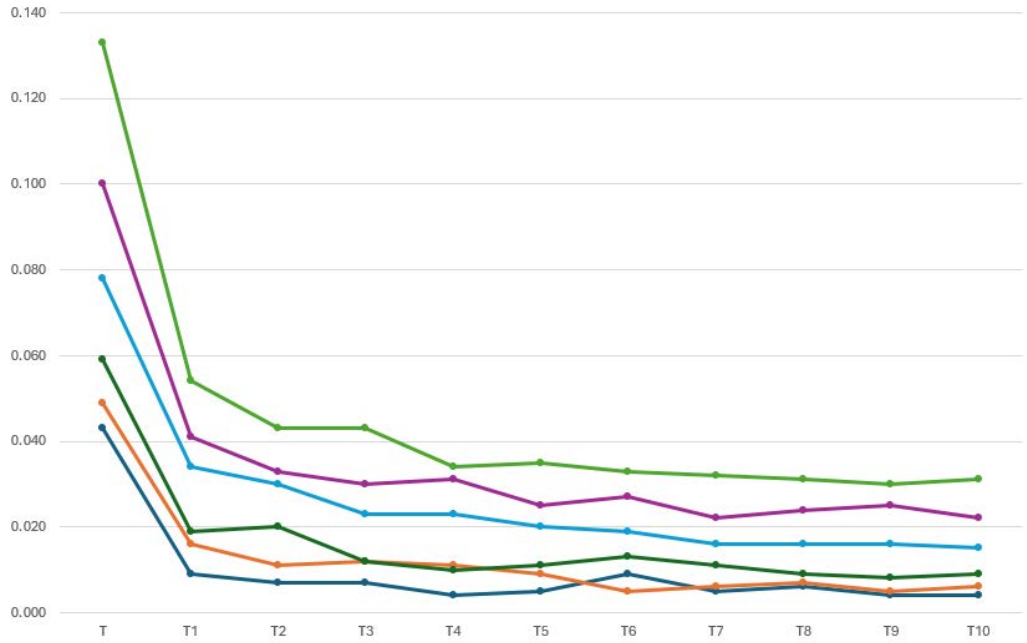

Fig. 5. The dependence of the rule execution time on the number of repetitions.

It is evident that data caching occurs at the Extract stage (Fig. 1. Typical ETL process diagram.), and each subsequent execution of the rule is several times faster. Therefore, for optimizing the performance of rule-based approaches for ontology construction from text, it makes sense to group rules based on the similarity of input data (samples at the Extract stage). This approach significantly speeds up the execution of syntactic-semantic transformations, as grouping the rules means that each subsequent rule has a partially or fully cached set of input data.

Conclusions

The construction of an ontology from textual documents is a crucial task in the process of automated learning for intelligent agents. Solving this problem opens new possibilities for building systems with a causal type of reasoning, which, unlike probabilistic models – an analogy of collective thinking – lays the foundation for implementing logical inference processes and searching for the optimal strategy of an intelligent agent.

This article proposes an approach that significantly advances the solution to this problem. The proposed approach aims to enhance the analysis of textual information by unifying data from different text parsers to extract a greater number of syntactic and semantic characteristics. A graph database is suggested as an operational repository for storing the output data of text parsers, bringing them into the simple and widely recognized RDF format, with an atomic structure in the form of the triple "subject-predicate-object." Such unification enables further interpretation and processing of text parser data by various systems.

Additionally, the article presents the concept of constructing a semantic graph, which serves as the basis for ontology representation. The proposed concept follows a rule-based approach to constructing semantic structures from syntactic graphs using ETL processes and SPARQL queries. This formalization of rules opens new horizons for automated learning of an intelligent agent, as rules in the form of SPARQL scripts can be easily added and instantly interpreted "on the fly."

Furthermore, the implementation of first-level reflection for the studied rule allowed for an assessment of the execution time dependence on the volume of the text corpus and the size of data samples at different ETL process stages. The results of the experimental study clearly demonstrate the impact of data sample caching on rule execution time. This paves the way for optimizing the rule-based process of ontology construction from text by properly formalizing and clustering rules based on the similarity of input data samples obtained from text parsers.

REFERENCES

1. Apache Open NLP Website. (n.d.). (Apache) Retrieved from https://opennlp.apache.org/
2. Asim, M. N., Wasim, M., Khan, M. U., Mahmood, W., & Abbasi, H. M. (2018). A survey of ontology learning techniques and applications. Database: The Journal of Biological Databases and Curation, 2018(bay101). doi:10.5120/2610-3642
3. Basaraba, I., Bets, I., & Bets, Y. (2024). Current trends in the recognition and decoding of phraseological units. Current Issues of the Humanities, 74(1), 211-216. doi:10.24919/2308-4863/74-1-29
4. Chornyi, A. (2024). Development of an adequate intellectual agent for a wide subject area as a model for further scientific research. Abstract. Retrieved from https://www.academia.edu/127201897
5. CoreNLP vs Apache OpenNLP. (n.d.). (Awsome Java) Retrieved from https://java.libhunt.com/compare-corenlp-vs-apache-opennlp
6. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019, June). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL HLT 2019), pp. 4171-4186. doi:10.48550/arXiv.1810.04805
7. Doroshenko, A. (2018). Development of information technology for intellectual analysis of factographic information. Bionics of Intelligence, 1 (90), 116-121. doi:10.11591/eei.v11i5.3075
8. Dosyn, D., & Lytvyn, V. (2021). Models and methods for determining the usefulness of ontological knowledge: Monograph. Lviv: "Novyy svit – 2000".
9. Dosyn, D., Daradkeh, Y., Kovalevych, V., Luchkevych, M., & Kis, Y. (2022). Domain Ontology Learning using Link Grammar Parser and WordNet. MoMLeT+DS 2022: 4-th International Workshop on Modern Machine Learning Technologies and Data Science. Leiden-Lviv, The Netherlands-Ukraine. Retrieved from https://ceur-ws.org/Vol-3312/paper2.pdf
10. GATE website. (n.d.). Retrieved from https://gate.ac.uk/
11. Haiko, C. (2023). Ontology-driven means for processing and presentation of large arrays of unstructured texts. Innovative Technologies and Scientific Solutions for Industries, 2(24), 27-38. doi:10.30837/ITSSI.2023.24.027
12. Hlybovets, M., & Bobko, O. (2012). The methods of automatic ontology generation. NaUKMA Research Papers. Computer Science, 138, 61-67. Retrieved from https://ekmair.ukma.edu.ua/handle/123456789/1917

13. Kumari, P. (2024, October 26). 7 Top NLP Libraries For NLP Development. Retrieved from https://www.labellerr.com/blog/top-7-nlp-libraries-for-nlp-development
14. Linked Open Data Cloud. (n.d.). Retrieved from https://www.lod-cloud.net/
15. Lytvyn, V., & Cherna, T. (2014). The problem of automated development of a basic ontology. Journal of Lviv Polytechnic National University "Information Systems and Networks", 805, 306–315.
16. Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., & McClosky, D. (2014). The Stanford CoreNLP Natural Language Processing Toolkit. Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, (pp. 55-60). Baltimore, Maryland, USA. doi:10.3115/v1/P14-5010
17. Mousavi, H., Kerr, D., Iseli, M., & Zaniolo, C. (2014). Harvesting Domain Specific Ontologies from Text. International Conference on Semantic Computing. Newport Beach, CA, USA. doi:10.1109/ICSC.2014.12
18. Nanavati, J., & Ghodasara, Y. (2015, November). A Comparative Study of Stanford NLP and Apache. International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, 5(5), 57-60. Retrieved from https://www.ijsce.org/wp-content/uploads/papers/v5i5/E2744115515.pdf
19. NTLK website. (n.d.). (NLTK Project) Retrieved from https://www.nltk.org/
20. Schmitt, X., Kubler, S., Robert, J., Papadakis, M., & LeTraon, Y. (2019). A Replicable Comparison Study of NER Software: StanfordNLP, NLTK, OpenNLP, SpaCy, Gate. Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS). Granada, Spain. doi:10.1109/SNAMS.2019.8931850
21. Shaptala, R. (2023). Dictionary embeddings for document classification in low-resource natural language processing. – Qualification scientific work as manuscript. Kyiv. Отримано з https://ela.kpi.ua/items/14de271d-5971-4cdc-92e6-8e645336332d
22. Shvorob, I. (2015). Comparative analysis of methods for syntactic parsing of texts. Journal of Lviv Polytechnic National University "Information Systems and Networks", 814, 197-202. Retrieved from http://nbuv.gov.ua/UJRN/VNULPICM_2015_814_22
23. spaCy website. (n.d.). Retrieved from https://spacy.io/
24. Stanford CoreNLP website. (n.d.). Retrieved from https://stanfordnlp.github.io/CoreNLP/
25. Vovnianka, R., Dosyn, D., & Kovalevych, V. (2014). The method of knowledge extraction from text documents. Journal of Lviv Polytechnic National University "Information Systems and Networks", 783, 302–312.
26. Yunchyk, V., Kunanets, N., Pasichnyk, V., & Fedoniuk, A. (2021, 10). Analysis of artificial intellectual agents for e-learning systems. Journal of Lviv Polytechnic National University "Information Systems and Networks", 10, 41-57. doi:10.23939/sisn2021.10.041
27. Zezula, T. (2020, August 29). 15 Natural Language Processing Libraries Worth a Try. Retrieved from https://www.tomaszezula.com/natural-language-processing-libraries
28. Zlatareva, N., & Amin, D. (2021). Processing Natural Language Queries in Semantic Web Applications. The 7th World Congress on Electrical Engineering and Computer Systems and Science (EECSS'21). doi:10.11159/cist21.108

РОЗРОБЛЕННЯ ЄДИНОГО ФОРМАТУ ВИХІДНИХ ДАНИХ ДЛЯ ТЕКСТОВИХ ПАРСЕРІВ В СИСТЕМІ ПОБУДОВИ ОНТОЛОГІЇ З ТЕКСТОВИХ ДОКУМЕНТІВ

Андрій Чорний[1], Дмитро Досин[2]

[1, 2] Національний університет "Львівська політехніка",
кафедра інформаційних систем та мереж, Львів, Україна
[1] E-mail: andrii.o.chornyi@lpnu.ua, ORCID: 0009-0007-4005-4088
[2] E-mail: dmytro.h.dosyn@lpnu.ua, ORCID: 0000-0003-4040-4467

Проблема відсутності ефективних засобів побудови онтологій з текстових документів все ще залишається невирішеною. Її розв'язання стикається з низкою викликів, зокрема, відсутністю єдиного формату вихідних даних різних NLP інструментів, зокрема текстових парсерів, які є

першою ланкою в багатоетапному процесі видобування знань. На сьогоднішній день існує декілька популярних текстових парсерів, кожен з яких має свої особливості та переваги у реалізації окремих функцій. З метою ефективнішого вирішення проблеми побудови онтології з тексту доцільним є використання декількох текстових парсерів, що породжує проблему узгодження форматів вихідних даних цих NLP інструментів.

Для вирішення задачі уніфікації формату вихідних даних текстових парсерів, запропоновано використання графової бази даних для їх збереження у форматі триплета суб'єкт-предикат-об'єкт з метою подальшого опрацювання з використанням правило-орієнтованих трансформацій на основі SPARQL запитів. Суттєвою перевагою такого підходу є можливість виконання кожного нового правила "на льоту".

В рамках дослідження розроблено інтелектуального агента на мові Java, здатного будувати семантичні графи з природомовного тексту на основі правило-орієнтованого підходу. За допомогою розробленого інтелектуального агента проведено оцінку залежності часу виконання правила синтаксично-семантичної трансформації від об'єму текстового корпусу та розмірів вибірок даних. Дане оцінювання стало можливим за рахунок імплементованої рефлексії першого рівня для досліджуваного правила трансформації.

За результатами дослідження, запропонований підхід уніфікації вихідних даних текстових парсерів з використанням графової бази даних показав свою ефективність з точки зору складності операції та швидкодії. Розроблений підхід побудови онтології з тексту відкриває перед сучасною наукою нові горизонти для автоматизованого навчання інтелектуального агента на основі текстової інформації.

Ключові слова – опрацювання природної мови, онтологія, автоматична побудова онтології, автоматизоване навчання, синтаксично-семантичні шаблони