A Textual Information Extraction Application based on XML Data Models and a Multidimensional Natural Language Processing Pipeline Approach

1st Tobias Dorrn
Fraunhofer IOSB
Karlsruhe, Germany
tobias.dorrn@iosb.fraunhofer.de

2nd Natalie Dambier
Fraunhofer IOSB
Karlsruhe, Germany
natalie.dambier@iosb.fraunhofer.de

3rd Almuth Müller
Fraunhofer IOSB
Karlsruhe, Germany
almuth.mueller@iosb.fraunhofer.de

4th Achim Kuwertz
Fraunhofer IOSB
Karlsruhe, Germany
achim.kuwertz@iosb.fraunhofer.de

***Abstract*— Modern data and information systems usually contain considerable amounts of data and documents and thus provide a large amount of information. The automatic extraction of domain-specific information is all the more important in order to improve work with such systems. If information is available as free text information, machine processing can prove to be a difficult technical hurdle. State-of-the-art approaches use modern Natural Language Processing (NLP) methods to solve such tasks. In this paper, we want to introduce a data-driven approach, applying an XML data model to an application-specific scenario, using different NLP methods, which are combined into a multidimensional pipeline. It is important to understand how certain NLP methods can be used and what their limitations are. Individual modern NLP methods are often not sufficient and resilient enough to solve complex information extraction tasks. Therefore, it has to be examined how such problems can be alleviated or circumvented by a combination of different NLP methods. As a distinction to categorical grammar models, all cases considered here should be available as free text. The approach presented in this paper is still a work in progress, yet first evaluation results will be given.**

Keywords—NLP, XML, pipeline, application, information extraction

I. Introduction

Modern data and information systems usually contain considerable amounts of data and documents and thus provide a large amount of information. In addition to the storage of data and information, further machine processing is also an essential reason for using such systems. For the further processing of information, the underlying data format of the stored data is of decisive importance. Often, information conforming to certain data formats has to be entered by users in a structured way using special input forms or specialized applications. However, it becomes more difficult if the input or storage data does not have a structured data format. The automatic extraction of specific information from e.g. text-based documents without a clearly defined data format often represents a very challenging task with the intention of making working with such systems easier. In this article, a scenario is considered in which freely formatted information can be checked for specific content using AI methods and a given data model and can be understood by machines to enable further processing. The aim of the approach described here is that such information can also be processed and stored in an unstructured (e.g. as free text) form, since the appropriate tools for information processing are not always available or the information is already available in an unstructured form.

This paper addresses the following application-specific problem: as input a text is considered, or as a first draft, only one sentence of a text. In addition, an existing data model in the form of a controlled vocabulary and a (partial) set of rules that describes how sentences can be formed from this vocabulary is assumed to be given. In contrast to grammar-based language models [1], [8], the texts as well as the underlying data model considered here are not strictly formalized, which means that texts, formulated without knowledge of a data model or set of rules, can be processed with higher success. The question to be answered is how relevant information with respect to the data model can be extracted from such a text and subsequently displayed in a structured manner, with the aim of being able to more easily collect and process relevant textual information in relation to a specific data model. The structure of the considered data model will be described in more detail in section II.A.

As a first step to solving this problem, existing methods from the field of Natural Language Processing (NLP) were examined with respect to their applicability as a solution. These were NLP

„*2022 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*“

„DOI: 10.1109/INISTA55318.2022.9894171.“
978-1-6654-9810-4/22/$31.00 ©2022 IEEE

methods such as Dependency Parsing (DP) but also higher-level methods such as Reading Comprehension (RC) and Open Information Extraction (IE). As a restriction resulting from the considered application scenario, specific constraints had to be taken into account, such as the offline usability of the considered methods as well as the use of domain-unspecific - open source - models, since domain-specific language models for the predefined data model were not available. A first look at the NLP methods mentioned shows that although individual NLP methods can partially solve the problem, they cannot adequately solve the overall problem, i.e., the extraction of all relevant information. More detailed results of this examination will be given in section II.A. Another problem is often the robustness of such NLP methods and the dependence of their results on the input data. In order to be able to adequately solve the defined problem under the stated constraints, the use of a processing chain in the form of a multidimensional pipeline was examined as a next step. A similar approach is e.g. taken in [4] and [9]. The proposed approach for such a processing chain is described as an overview in section II.C and in detail in section III. Various combinations of NLP methods were tested as necessary preliminary considerations and then implemented in the form of a pipeline. For the pipeline itself, the results of individual NLP methods are passed on to subsequent modules in the form of a pipeline protocol. The designed pipeline protocol will also be presented in section III. Python was used as programming language for implementing the pipeline and a class derived from the data type dictionary was used as the interface format for the pipeline protocol. An essential part of this paper is to answer the question of what a pipeline could look like in order to be able to solve the task considered here.

First results for a proof-of-concept evaluation of the proposed pipeline approach are given in section IV. Section V then concludes this paper. Since the proposed approach is still a work in progress, the remaining implementation aspects as well as future work are also discussed in section V.

II. Preliminary Investigation

In this section, the considered problem is stated in more detail, describing e.g., the structure of the predefined data model with is given prerequisites in the considered application scenario. Also, the results of a first evaluation of single, readily available NLP methods to solve this problem are given.

A. Task Description

In order to enable the extraction of information from free text, common NLP methods were first examined. In the considered application scenario, the data model is given as a predefined artifact created by domain experts. For the proposed approach, the given data model was first preprocessed, translating its contents into an easier-to-handle representation based on XML. This representation was chosen for the data model in order to be able to better compare the results of an information extraction from a free text. An example of the structure of the XML representation of a possible data model is given in Fig. 1. For the sake of simplicity and visualization, only a toy data model is presented in this paper. Yet, this toy data model exhibits the same structure as the predefined data model given in the considered application scenario. Here we use a well-formed XML file divided into two data classes. The first class contains all the root elements (not shown in Fig. 1). Each root element is the representative of a category element (in Fig. 1, two category elements are given: “dog” and “cat”). In the second data class of the XML structure, which is inside the root element, all possible searchable properties for a root element are listed. The aim of this XML structure is first to check a given free text for corresponding category elements and then to extract any properties based on the given data model.

As the final goal of the considered information extraction task, relevant content of a given input text shall be identified, mapped to the XML data tree, and be displayed as annotated text. Finally, the identified and mapped text can be further processed in a structured manner.

```
<item value="Dog">
    <item value="kind"></item>
    <item value="characteristics">
        <item value="Color">
            <item value="white"></item>
            <item value="black"></item>
            <item value="brown"></item>
            <item value="grey"></item>
            <item value="red"></item>
        </item>
        <item value="Size">
            <item value="tiny"></item>
            <item value="small"></item>
            <item value="big"></item>
            <item value="little"></item>
            <item value="midsize"></item>
            <item value="medium"></item>
            <item value="huge"></item>
            <item value="giant"></item>
        </item>
    </item>
    <item value="activities">
        <item value="playing"></item>
        <item value="sleeping"></item>
        <item value="eating"></item>
    </item>
</item>
```

```
<item value="Cat">
    <item value="kind"></item>
    <item value="characteristics">
        <item value="Color">
            <item value="white"></item>
            <item value="black"></item>
            <item value="brown"></item>
            <item value="grey"></item>
            <item value="red"></item>
            <item value="solid"></item>
            <item value="bi-color"></item>
            <item value="tabby"></item>
        </item>
        <item value="Age">
            <item value="kitten"></item>
            <item value="junior"></item>
            <item value="adult"></item>
            <item value="mature"></item>
            <item value="senior"></item>
        </item>
    </item>
    <item value="activities">
        <item value="playing"></item>
        <item value="sleeping"></item>
        <item value="eating"></item>
    </item>
</item>
```

Fig. 1. Example toy data model, represented as an XML file, showing the general structure of data models considered in this paper. Here, an XML model with two category elements (*“dog” and “cat”*) is depicted. All desired category element properties are listed within each category element.

In Table 1, an example result is shown for processing a given example sentence. Here, the different relevant parts of the sentence are identified according to the data model and annotated with the respective categories as given in the data model.

TABLE I. Extraction and Processing Example for a Given Sentence with Respect to the Data Model Shown in Fig. 1.

Sentence: There is a tiny black dog, playing in front of a tree.						
Annotated result						
	Object size	***Object color***	***Object***		***Object activity***	
There is a	tiny	black	dog	,	playing	in front of a tree.

According to the structure of the considered data model, two main tasks can be identified for the extraction of information:

1. Determination of the category element based on the given data model (see Fig. 1).

2. Determination of the associated properties of the determined sentence object depending on the associated data model.

B. *Considered NLP Methods*

As stated before, as a first approach to tackle the described problem, different NLP methods were evaluated. All NLP methods mentioned in the following refer to the NLP Framework Allen NLP [2].

Dependency Parsing (DP) is an NLP method to find dependencies between individual sentence elements. DP always returns all elements of a sentence as its result, which makes it suitable as a robust starting method for a chained approach, since no information about a sentence is lost. Furthermore, category elements found within the dependency graph can also be prioritized by determining the distance to the root graph element. (see Fig. 2, here "is" is the root graph element). Using the example in Fig. 2, two possible extraction elements can be identified by comparing each graph node to the main elements of the data model (see Fig. 1). The comparison of the category element priority shows that "dog" has a higher priority than "cat", since "dog" is closer to the root graph element, making it a valid candidate for being the main message category of the considered sentence. If you do not want to extract information for all main elements (see Fig. 1) at once, the step of prioritization proposed here is also suitable as a possible entry procedure for the approach presented in section III.

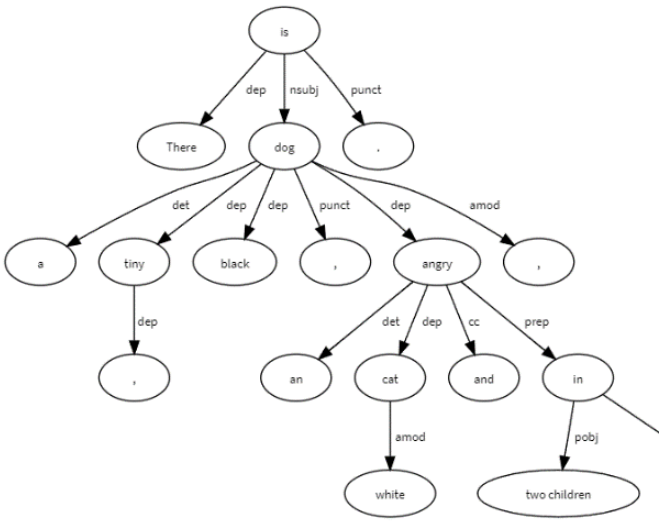

Fig. 2. Example of a shortened Dependency Parsing output (*sentence: "There is a tiny, black dog, an angry white cat and two children in front of a tree."*). Dependencies to a main element can be below (*direct dependency*), above or as a neighboring (*not shown here*) element.

Apart from determining the message category as described in the last section, DP can also be used to extract properties from an existing data object. However, the disadvantage is that the relevant information has to be extracted from the result with great effort. Furthermore, depending on the domain and level of training of the model used, the results can have different qualities in relation to their position in the dependency graph. It should be noted that the dependencies (from the perspective of the dependency graph) on an object can occur below as direct dependencies, above as superordinate objects (including their child objects) and also next to an object as neighboring elements (including their child objects), which makes an evaluation more complex as with other methods (see Fig. 2.). Another disadvantage is that DP should only be used on single and simple (without using conjunctions or multiple listing of main topic elements of the same type) sentences, otherwise the results will deviate too much from the search pattern suggested here (taking into account the superordinate, neighboring, and child nodes). As already mentioned, DP always returns the entire content of a sentence, which would be disadvantageous with regard to the search pattern proposed here in the case of long sentences whose contents do not only refer to one searched object. In such a case, it would make sense to first carry out a semantic sentence separation applying other NLP methods. Further information on the DP model used here can be found in [7] and [10].

Reading Comprehension (RC) is a higher-level method for answering questions about a text. Apart from the input text, a question must also be formulated as input. Based on the question, the input sentence is separated on a semantic level, the result is ideally the answer to the question. The approaches presented here in the paper use neural-based RC models (e.g. BiDAF) as described in [5] and [11]. In order to be able to use RC on a larger scale, additional information about a given sentence often has to be available in order to be able to formulate more comprehensive questions about a given text. Therefore, the sole use of RC based on the task described here is not recommended and should instead be supplemented with other methods. As the following example shows (see Table 2), the question asked here (Question: "What color is the dog?") can only be asked if it is already known that the given sentence is also about a dog (i.e., dog being the message category of the given sentence).

TABLE II. EXAMPLE OF EXTRACTING INFORMATION FROM A SAMPLE SENTENCE USING READING COMPREHENSION (SENTENCE: "THERE IS A TINY, BLACK DOG, AN ANGRY WHITE CAT AND TWO CHILDREN IN FRONT OF A TREE.", QUESTION: " WHAT COLOR HAS THE DOG?", ANSWER: "BLACK").

Sentence: "There is a tiny, black dog, an angry white cat and two children in front of a tree."		
Question: "What color has the dog?",		
	What color has the dog?	
There is a tiny,	black	dog, an angry white cat and two children in front of a tree.

The advantage of RC is that it allows direct searching (see Table 2) for specific information in a given text and the use of texts with more than one sentence. The disadvantage, however, is robustness, especially when the approach used for RC has been specifically trained or fine-tuned for the considered application domain or use case. The results of RC can vary greatly, depending on the employed approach (or, more specific, the used model), and the formulation of the question. Furthermore, only a single question can be answered at a time, so that the search for multiple pieces of information requires a chain of multiple such procedures.

Similar to DP, open information extraction (IE) is a procedure that identifies specific dependencies between the elements of a sentence. In contrast to DP, however, IE

specifically refers to verbs that appear in the sentence. This can also be a disadvantage if relevant verbs are not contained in the data model or not given in the sentence to be examined. The following example (Fig. 3) shows two different sentences being checked for their verb relationships using IE. It is evident that the arguments found here can contain both the searched main elements (see turquoise elements in Fig. 3) and their properties. The results of this procedure are therefore effective, but not always sufficient, since the results sometimes have to be further divided. Essentially, IE alone is usually not sufficient for the task described here, but it is suitable as a method in combination with other methods. More detailed information on the IE model used here can be found in [6].

Fig. 3. Example of an extraction result based on two sentences using IE (*sentence: "There is a tiny black dog in playing front of a tree" and "A small white dog is playing across the street"*). IE generally returns three different result categories, verbs (*marked in purple*), arguments related to the verb (marked in turquoise) and argument modifiers, which contain additional information about the arguments (*marked in green*). It is evident that not all verb relations found also contain results suitable for the considered application in the form of arguments.

C. Combination of NLP Methods

On the basis of the previous considerations, the possibility of manually linking together the considered methods was examined in a further step. For such a concatenation, both the semantic (i.e., results that return sentence parts or fragments) and information-gaining results (i.e., results that contain additional information about the sentence and cannot be represented as sentence parts or fragments) of the considered methods were taken into account. As a result of our preliminary considerations for the sentences considered in the paper, the relevant information could always be derived, either with the aid of the examined methods or with a manual, sentence-specific combination of them. Fig. 4 below shows an example of a possible combination of two NLP methods.

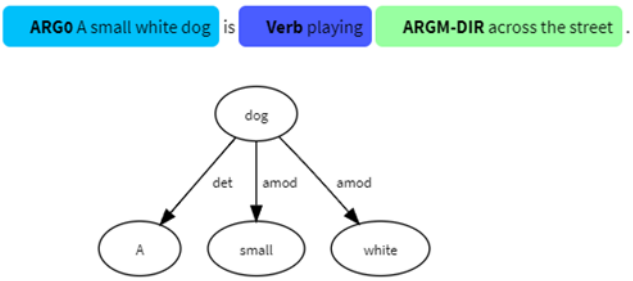

Fig. 4. Analysis example for the possible combination of different NLP methods in the considered application scenario. In this example (*sentence: "A small white dog playing across the street"*), two steps (*step 1 uses IE, step 2 uses DP*) are necessary to determine all the information needed. Step 1 finds "playing" as a property and the argument (*sentence fragment*) to examine further in the second step using IE. In the second step, the main element "dog" (*see Fig. 1*) and two other properties (*"small" and "white"*) for the main element can be found using DP.

D. Conclusion

In order to extract relevant information from free text in the application scenario and under the constraints considered in this paper, a combination of existing NLP methods is usually required, since individual methods do not always return complete and suitable results.

III. Solution Approach/ Concept

Due to the promising results of the preliminary investigation, as a next step the application of a processing chain was examined in more detail. The processing chain was designed in the form of a pipeline, with the aim of providing a robust and sufficient solution to the considered problem, i.e. the automated identification and mapping of free text elements in a sentence to a predefined data model. Details regarding the proposed processing chain and its ability to provide explanations for these single processing steps on-the-fly will be presented in the following.

A. Pipeline Architecture

To implement a suitable pipeline architecture, Python was used in conjunction with a dynamic pipeline pattern approach using the Python fastcore library[1]. AllenNLP was chosen as the NLP framework and the underlying data model is based on XML (similar in structure to the data model depicted in Fig. 1). The NLP methods used here are again Dependency Parsing (DP), Reading Comprehension (RC) and Open Information Extraction (IE).

The pipeline itself consists of individual NLP methods that can be executed in series (see Fig. 5). The approach is designed to support fully automated pipelines, which also take into account errors within individual processing steps (making this a dynamic pipeline approach). The NLP methods communicate with each other via a specially designed pipeline protocol that contains all the results of the previous pipeline steps, which means that each pipeline process can access all the results of its predecessors. The details of the designed pipeline protocol are described in Fig. 6. Proceeding with this protocol approach allows more complex processing structures and the passing of a variety of information between the individual processing steps. Apart from the results of the NLP procedure, other process-relevant information about the processes taking place within the pipeline can also be obtained and displayed (multidimensional pipeline). In addition, by persisting and displaying the results of each pipeline step in the protocol, the traceability of the overall process and how results are generated is improved, grating the proposed pipeline approach at least a partly inherent explainability, also with regard to the AI methods examined here. Before a pipeline is set up, the corresponding specifications are first transferred to the pipeline protocol based

[1] https://github.com/fastai/fastcore

on the data model (Fig. 5). This specifies a knowledge-based pipeline behavior using the data model. For complex data models with more than one category element (see Fig. 1), the inputs must be pre-analyzed (Fig. 5) to pick the correct main element (see Fig. 1) in the data model and in the sentence. Separate pipeline steps and boundary conditions are then stored for each main element, which are entered in the protocol. When running through the pipeline, these specifications are checked and executed. In the event of an error, the approach is designed to execute alternative paths and steps (dynamic pipeline approach). These alternative paths should then also be specified in the pipeline protocol by the pipeline processes and should also be stored in the data model in advance.

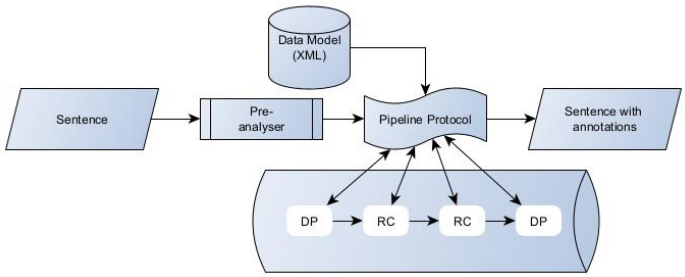

Fig. 5. Construction of the pipeline structure. The input parameter is free text, which is to be checked for the content of the data model as shown in Fig. 1. The pipeline uses the methods presented in section II to produce an annotated text output according to the given data model. The pipeline protocol serves as a language between the pipeline processes. As a preprocessing step, the main element according to Fig. 1 is always examined first.

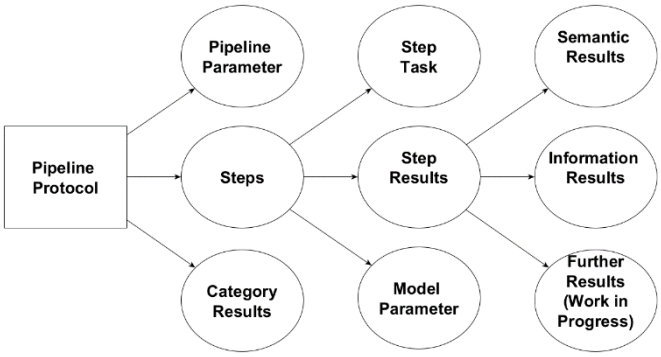

Fig. 6. Simplified structure of the pipeline protocol. The specifications of the data model (*pipeline parameter, step task and model parameter"*), the identification results regarding the main element from the preliminary analysis (*"category results"*) and the results of the individual processing steps *("Step Results"*) are stored here. The step results can consist of several individual results and can be passed on to each subsequent step (*multidimensional pipeline. "Further Result" means that combined results from different steps are also possible*). Furthermore, the input sentence, the annotated output sentence, and other information are stored in the protocol (*not shown here*).

IV. Proof of Concept

A. Status of Implementation

The proposed pipeline architecture is currently still a work in progress. At the moment, the following aspects of the pipeline have been implemented:

- The pipeline is still implemented in a static manner, which means that neither alternative paths nor error analyzes are being carried out within the pipeline. In addition, the pipeline must still be specified manually.
- The methods Dependency Parsing and Reading Comprehension are implemented in the pipeline.
- Only the semantic step results (see Fig. 6) can currently be used for the pipeline protocol.
- Pre-analyzer and annotated output (see Fig. 5) is implemented.

B. Evaluation

Table 3 shows the results of the processing chain using the currently implemented pipeline shown in Fig. 5 and the data model shown in Fig. 1. The task was to use the given data model to determine the relevant main element ("dog") and to extract all associated properties ("tiny", "black" and "playing") that are listed in the data model (Fig. 1). Based on the given task, two possible pipeline approaches could be determined (see Fig. 7).

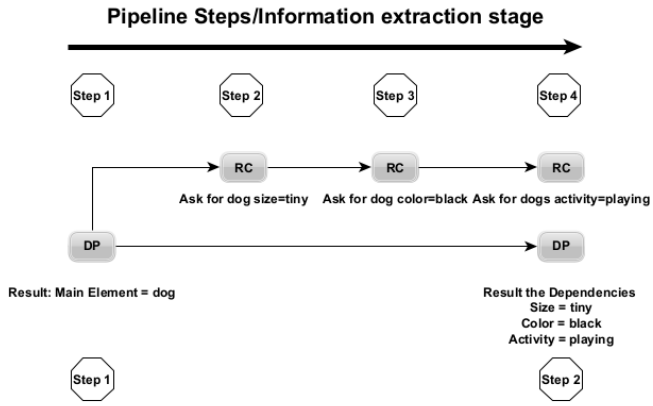

Fig. 7. Result chain of two possible pipeline approaches. The task was to first determine the main element (*"dog"*) from a given sentence (*sentence: "There is a small black dog playing in front of a tree"*) according to a give data model (*see Fig. 1*). In the next step or steps, all properties corresponding to the determined main element should be listed according to the given data model (*see Fig. 1*). For both approaches presented here, the main element is first determined using DP (*see also Table 3*), then the properties can be extracted either via a pipeline with triple use of RC or single use of DP (*cf. Table 3*).

TABLE III. Annotated results for two different pipeline approaches according to Fig. 7

Sentence: There is a tiny black dog, playing in front of a tree.						
Annotated result						
	Object size	***Object color***	***Object***		***Object activity***	
There is a	tiny	black	dog	,	playing	in front of a tree.
Pipeline 1-4 (DP-RC-RC-RC)	Step 2 (RC)	Step 3 (RC)	Step 1 (DP)		Step 4 (RC)	
Pipeline 1-2 (DP-DP)	Step 2 (DP)	Step 2 (DP)	Step 1 (DP)		Step 2 (DP)	

V. Conclusion and Further Work

A. Conclusion

For the information extraction from free text using a data model and pipeline approach, the representation of a domain-specific solution based on existing unadapted (language) models (zero-shot learning [3], [12]) was presented. Based on preliminary considerations, a suitable solution approach in the form of NLP pipeline processing was proposed. A main element of the proposed pipeline architecture is the use of a pipeline protocol, storing the intermediate results of the processing steps and, at the same time, acting as a means of improved traceability and inherent partly explainability. The results of a first proof-of-concept implementation were evaluated using simplified sentences. The presented approach is still a work-in-progress.

B. Planned Further Work

As for future work, the full implementation of the designed pipeline approach as well as a respective evaluation of its capabilities is planned. This includes the extent of the pipeline with full dynamic behavior.

Furthermore, it is planned to examine how NLP methods adapted to the specific domain described in a data model by training (fine-tunic) can improve the overall performance of the approach. Also, the application of the pipeline to other data models and other domains is planned.

References

[1] Kanakam, P., Hussain, S. M., Narayana, D. S., and Gupta, S. Semantic Representation of Natural Language Query Using Combinatory Categorical Grammar and Lambda Calculus. In: *proceedings of International Conference on Signal Processing, Control and Data Analytics*. 2015.

[2] AllenNLP: A Deep Semantic Natural Language Processing Platform; Matt Gardner and Joel Grus and Mark Neumann and Oyvind Tafjord and Pradeep Dasigi and Nelson F. Liu and Matthew Peters and Michael Schmitz and Luke S. Zettlemoyer, 2017, arXiv:1803.07640

[3] XIAN, Yongqin; SCHIELE, Bernt; AKATA, Zeynep. Zero-shot learning-the good, the bad and the ugly. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017. S. 4582-4591.

[4] HAHN, Udo, et al. An annotation type system for a data-driven NLP pipeline. In: *Proceedings of the Linguistic Annotation Workshop*. 2007. S. 33-40.ZHANG, Xin, et al. Machine reading comprehension: a literature review. *arXiv preprint arXiv:1907.01686*, 2019.

[5] Zhang, X., Yang, A., Li, S., & Wang, Y. Machine reading comprehension: a literature review. *arXiv preprint arXiv:1907.01686*, 2019.

[6] Stanovsky, G., Michael, J., Zettlemoyer, L., and Dagan, I.Supervised open information extraction. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. 2018. S. 885-895.

[7] DOZAT, Timothy; MANNING, Christopher D. Deep biaffine attention for neural dependency parsing. *arXiv preprint arXiv:1611.01734*, 2016.

[8] MÖRBITZ, Richard; VOGLER, Heiko. Weighted parsing for grammar-based language models. In: *Proceedings of the 14th International Conference on Finite-State Methods and Natural Language Processing*. 2019. S. 46-55.

[9] PARIKH, Jignesh R., et al. A data-driven architecture using natural language processing to improve phenotyping efficiency and accelerate genetic diagnoses of rare disorders. *Human Genetics and Genomics Advances*, 2021, 2. Jg., Nr. 3, S. 100035.

[10] ZHANG, Wenwen. Neural Dependency Parsing of Low-resource Languages: A Case Study on Marathi. 2022.

[11] Seo, M., Kembhavi, A., Farhadi, A., & Hajishirzi, H. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*, 2016.

[12] CHEN, Shiming, et al. Transzero: Attribute-guided transformer for zero-shot learning. In: *AAAI*. 2022. S. 3.