Visualization of LLM Annotated Documents

Teodor Valchev, Nikolay Paev
Artificial Intelligence and Language Technology
Institute of Information and Communication Technologies
Bulgarian Academy of Sciences
Bulgaria
teodorvulchev@gmail.com
nikolay.paev@iict.bas.bg

Abstract

Manual annotations play a crucial role in the Natural Language Processing domain. The paper presents an automatic annotation and visualization system for documents in the field of Social Studies and Humanities. The current annotation is on two levels, Named Entities and Events. The system combines automatically generated annotations from language models with a powerful text editor that is extended to accommodate manual annotation. The goal is to support the extraction of information from historical documents by scientists in the field of social studies and humanities. At the time of writing, the system is still in development.

1 Introduction

In this paper, we present the User interface (UI) to semantically annotated documents related to a knowledge graph representing the related knowledge of our CLaDA-BG[1] project. The aim of the developed system is to support the annotation of documents with the goal of expanding the Bulgarian-centric Knowledge graph and supporting researchers in the area of Social Sciences and Humanities (SS&H) in doing their investigations. The current architecture of the CLaDA-BG system is presented in Figure. 1. The main components of the architecture comprise (1) a Knowledge Graph and (2) Document Database that contain a large set of documents annotated with knowledge from the knowledge graph. The Knowledge Graph provides a contextualization of different datasets related to Bulgarian language, culture, and history. We call it *BGKG* (BulGarian-centric Knowledge Graph) because it represents main facts about people, settlements, locations, events, documents, organizations, etc. connected to Bulgaria. The Document Database contains a huge number of documents including archive documents, newspaper articles, letters, papers, description of artifacts, etc. Documents are annotated with concepts or instances from the knowledge graph. The annotation of documents supports search via queries expressed as textual elements, concepts, and facts defined in the terms of BGKG. The queried documents are post processed in different ways. The two main ones are: (1) ranking with respect to the query terms, and (2) extraction of new knowledge from extracted documents. This architecture assumes various types of users including at least the following ones: researchers, BGKG curators and Documents annotators. Researchers access the systems in order to find the necessary documents supporting their research. They produce new research represented as documents similar to the ones within the Document database. The BGKG curators manage the knowledge within it by checking its correctness, mapping different representations of the same knowledge, and adding new information. Annotators perform annotation of the documents manually or semi-manually, usually as a post editing after automatic annotation.

In our view this architecture is a way to provide access to NLP technologies to end users (researchers, teachers, etc.) who are not familiar (and not willing to become familiar) with these technologies. Thus, they will prefer to work as they are used to in their research. Our observations are that researchers in the area of SS&H usually are working with WYSIWYG[2] editors such as MS Word, Google Docs. Therefore, we consider as the main component of the UI a structural editor in which

[1]CLaDA-BG is a Bulgarian national research infrastructure for resources and technologies for linguistic, cultural and historical heritage, integrated within CLARIN EU and DARIAH EU.

[2]WYSIWYG stands for *What You See Is What You Get*

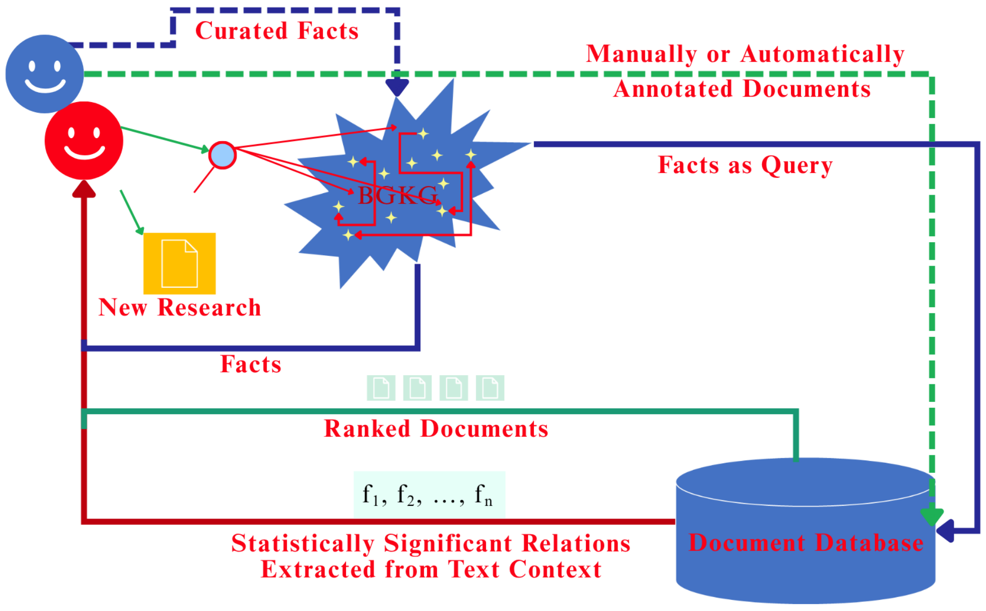


Figure 1: The Architecture of the CLaDA-BG Project.

the user can create new documents describing their new research, taking notes, etc.

Thus, the same editor is used to examine selected documents in the database, taking notes, annotations, and corrections. The editor has to support creation of well formatted documents representing scientific papers. Additionally, it has to be easily extendable to represent complex annotations across the content of the documents. As representation of the documents, we consider XML version of HTML (XHTML) — requiring the HTML document to be well-formed XML document. Having such well-formed XHTML documents makes it easy to add a minimal number of non-standard elements and attributes.

In this paper, we present the structure of the annotation of the documents, the automatic annotation, the architecture of the editor and its functionalities - visualization, (partial) re-annotation, linking the document to the knowledge graph and to annotated documents.

The structure of the paper is as follows: in the next Section 2 some works are discussed in relation to our work. In Section 3 the overall system architecture is presented with the document representation, database servers, the local LLM (Large Language Models) server, and the UI. Section 4 presents the workflow of the system, and Section 5 concludes the paper.

2 Related Works

Manual annotation of documents is a crucial step in all natural language processing tasks. The paper is concerned with the UI to support the work of the main types of users of our system. We consider as related works mainly systems for manual annotation of documents.

For many years, we used the CLaRK System (Simov et al., 2001) for corpus annotation, lexicon development, and more. The system main interface is an XML editor. In addition several tools for processing XML documents are internally implemented, such as Regular grammars over XML documents, constrains for validations of different annotation and/or insertions of valid XML fragments. The tools of CLaRK System allow us to solve most of the processing that we wanted to implement. But the system has some shortcomings. First, it is not connected to any external databases. Thus, users need to take care of document management by themselves. Second, the tools require knowledge of XML related technologies such as XPath[3] which is very powerful for processing XML document, but they are a burden for many of the potential users. In addition, the editor does not support any formatting instructions, making the system difficult for unfamiliar users. Thus, our work here draws on our experience with the CLaRK system.

[3]XML Path Language (XPath): https://www.w3.org/TR/xpath/

We have similar experience with the following systems: the GATE Teamware — (Bontcheva et al., 2013), the INCEpTION platform — (Klie et al., 2018), SpaCy: Industrial-Strength Natural Language Processing[4]. All of them have functionality for creation of rules for automatic text processing including regular expression rules, programming languages — Java, Python, for processing the predefined document data models. They allow for calling external processing tools including machine learning models, large language models (LLMs), etc. Behind these functionalities, these tools provide document visualization of the annotations, some of which we incorporate in our work. Such as coloring schema styles, tooltips, etc.

In our case, the main deviations from these tools are that we need a WYSIWYG editor[5] integrated with the rest of the architecture of our system. This is important because researchers value the structural presentation of documents, not just their content. This applies not only to their own documents, but also to the documents they use in their research.

Neves and Ševa provide a comprehensive review of manual annotation tools — (Neves and Ševa, 2019). They defined a set of evaluation criteria for what makes an annotation tool useful. Their results show that none of the tools they reviewed met their criteria fully (Functional, Data, and others). As a selection criteria for tools to be extensively reviewed in their study, they used: (1) availability, (2) to be web accessible (downloadable or online), (3) to be easy to install, (4) working for their field of studies, and (5) to allow definition of annotation schema. Of the 78 tools they considered, 63 were not selected for a detailed evaluation because they did not meet at least one of the five requirements.

3 System architecture

In this section, we present the document representation and the main components of our system - the backend server, databases, automatic annotation server, and UI.

3.1 User interface

The developed version of our software, as is currently, meets: web, easy to install, working in their field of studies. In the future, we plan to allow schema configuration and allow open availability.

[4]`https://spacy.io/`

[5]Such type of editors are most frequently used by researchers in SS&H area. Thus, they reflect their experience.

3.2 Document representation

As was mentioned above, we need to define an extended version XHTML. The main idea is to use XHTML to support the format of the original papers that are annotated and uploaded to the system or the paper created by the user in their own research activities. In order to perform experiments with the extended version of the basis XHTML format, we select an existing freely available web based HTML editor, which is not focused on annotation: the TinyMCE rich text editor[6] (GPL licensed version[7]).

We have experimented with several schemes for representation of annotation data. The result of these experiments shows that using more than one element which allows inclusion of several annotation elements the editing of the annotations and their interaction with the standard XHTML elements complicate the editing process. Thus, we decided to minimize the number of new elements. Experiments were performed using the TinyMCE Annotations API, but the span approach made the represention of overlapping annotation not user friendly in the resulting XHTML. Spacy annotation tool was reviewed and as a result only one new type of elements $< tok > token < /tok >$ is added with a number of new custom attributes. The extension of XHTML with this type of elements is call *cladaHTML*. More detailed explanation is available in Subsection 3.1.

The performed experiments using multiple tags showed that issues may arise, caused by mixing of the representations of the structured annotated document and the stylization.

Documents are represented in the database as tables of tokens, sentences, annotations, and known facts.

Usage of just one element seems too small addition, but representation over tokens allows for complex structures of annotations within Universal Dependencies CoNLL-U format[8]. Many other projects are using variants of CoNLL format.

```
<b><i>Text</b></i>
```

Listing 1: Crossing HTML elements which is erroneous in general HTML, and not well formed in XHTML.

[6]`https://www.tiny.cloud/`

[7]`https://github.com/tinymce/tinymce`

[8]`https://universaldependencies.org/format.html`

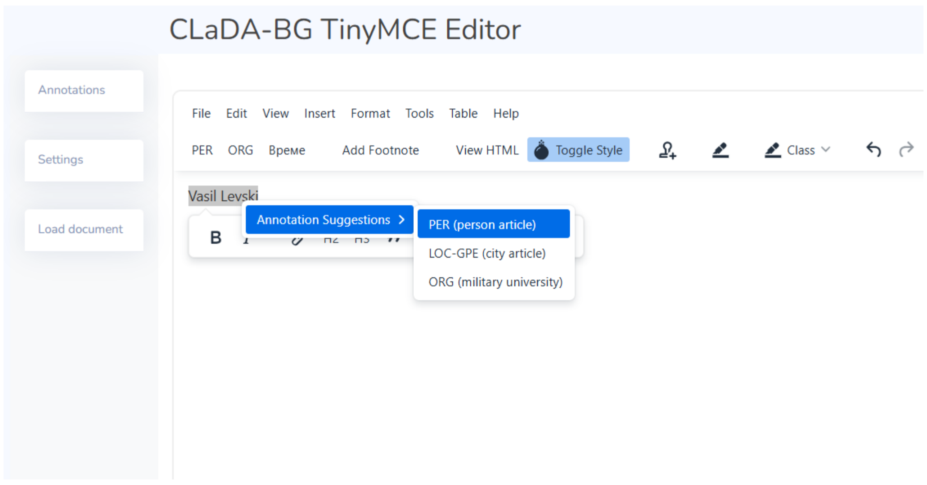


Figure 2: Annotation recommendation

```
<sentence id="1">
  <anot class= "class-1">
    <tok>The </tok>
    <anot class= "class-2">
      <tok>sun</tok>
      <tok>is </tok></anot>
    <tok>shining </tok>
  </anot>
</sentence>
```

Listing 2: Annotated example sentence with multiple elements

3.3 Back-end

In the back-end, there are a builder and a destructor for the cladaHTML, which build the documents from the tables in the relational database or convert them to database SQL queries to update the information in the tables. The relational database model is used to represent the documents with annotations, tokenization, etc. The scheme of the database is specified by the CLaDA-BG team and allows for searching of facts in the documents and in related documents, mentions, and more. The relational database model is also used as an intermediate representation of the documents annotated by LLMs.

3.4 Knowledge and Documents database

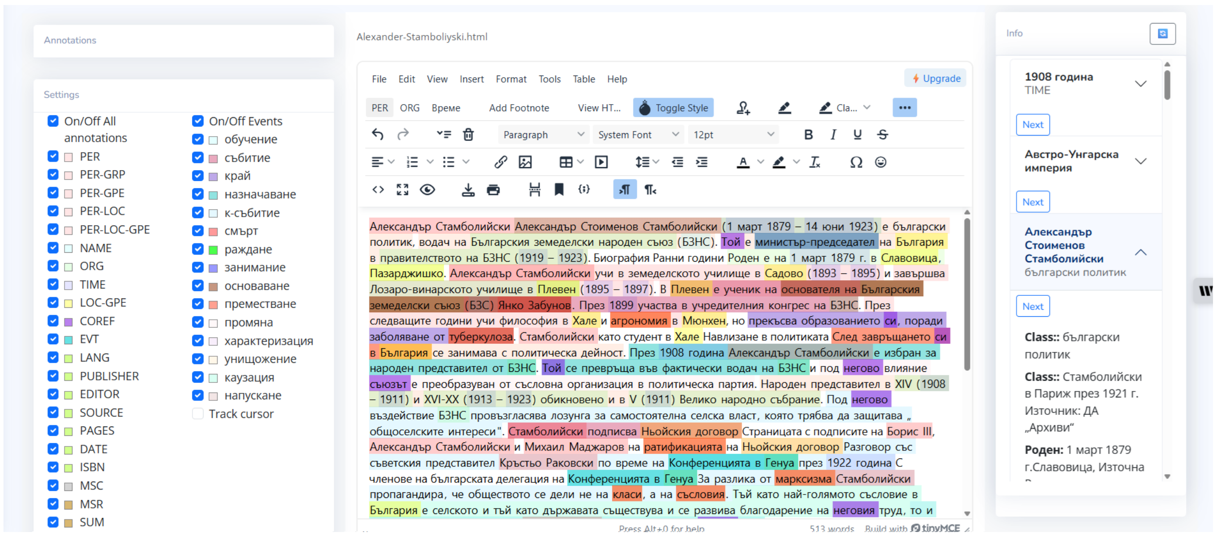
The main database for storing documents has the following tables with appropriate relations: Documents, Events, NEs (Named entities), Roles, Sentences, Tokens, and URLs. They allow for search queries like: All the documents where some Event/NEs is mentioned, searching for documents with close sentences, etc.

Using URLs, we can identify different occurrences (different names, pronouns, etc.) of the same object in the same or between multiple documents. The records in the database are structured in a way that allows for easy building of a fully functional knowledge graph.

The UI is web-based and is built on top of TinyMCE Text Editor, extended with JavaScript code. A screenshot of the UI is provided in Figure 3. A custom footnote and endnote changes tracking assistant is implemented. Coloring is achieved using Cascading Style Sheets (CSS) technology, but due to limitations in most browsers, only one rule per class from the same type can be visualized at the same time. To bypass this restriction, dynamic CSS coloring rules (single- and multiclass) are generated in the browser as the document is loaded in the editor. Rules are generated only for available combinations of classes, so we save ourselves from generating all possible combinations of classes and the linked exponential growth of all subsets.

The TinyMCE text editor internally is representing the document in HTML format (setting is available for XHTML), and allows the definition of custom tags. Only one custom tag $< tok >< /tok >$ with custom attributes is added, dividing the tokens. We call this language extension cladaHTML as mentioned earlier. In that way, we are preserving the behavior of all features of the editor and simultaneously adding new functionality. The reason for using only a single new tag is that in XHTML almost all elements must have a parent element, and tag misnesting[9] is not allowed. Misnesting occurs when XHTML tags are not properly nested, meaning that the order in which tags are closed does not

[9]https://w3c.github.io/html-reference/syntax.html



✓ On/Off All annotations	✓ On/Off Events
✓ PER	✓ обучение
✓ PER-GRP	✓ събитие
✓ PER-GPE	✓ край
✓ PER-LOC	✓ назначаване
✓ PER-LOC-GPE	✓ к-събитие
✓ NAME	✓ смърт
✓ ORG	✓ раждане
✓ TIME	✓ занимание
✓ LOC-GPE	✓ основаване
✓ COREF	✓ преместване
✓ EVT	✓ промяна
✓ LANG	✓ характеризация
✓ PUBLISHER	✓ унищожение
✓ EDITOR	✓ каузация
✓ SOURCE	✓ напускане
✓ PAGES	☐ Track cursor
✓ DATE	
✓ ISBN	
✓ MSC	
✓ MSR	
✓ SUM	

Александър Стамболийски Александър Стоименов Стамболийски (1 март 1879 – 14 юни 1923) е български политик, водач на Българския земеделски народен съюз (БЗНС). Той е министър-председател на България в правителството на БЗНС (1919 – 1923). Биография Ранни години Роден е на 1 март 1879 г. в Славовица, Пазарджишко. Александър Стамболийски учи в земеделското училище в Садово (1893 – 1895) и завършва Лозаро-винарското училище в Плевен (1895 – 1897). В Плевен е ученик на основателя на Българския земеделски съюз (БЗС) Янко Забунов. През 1899 участва в учредителния конгрес на БЗНС. През следващите години учи философия в Хале и агрономия в Мюнхен, но прекъсва образованието си, поради заболяване от туберкулоза. Стамболийски като студент в Хале Навлизане в политиката След завръщането си в България се занимава с политическа дейност. През 1908 година Александър Стамболийски е избран за народен представител от БЗНС. Той се превръща във фактически водач на БЗНС и под негово влияние съюзът е преобразуван от съсловна организация в политическа партия. Народен представител в XIV (1908 – 1911) и XVI-XX (1913 – 1923) обикновено и в V (1911) Велико народно събрание. Под негово въздействие БЗНС провъзгласява лозунга за самостоятелна селска власт, която трябва да защитава „ общоселските интереси". Стамболийски подписва Ньойския договор Страницата с подписите на Борис III, Александър Стамболийски и Михаил Маджаров на ратификацията на Ньойския договор Разговор със съветския представител Кръстьо Раковски по време на Конференцията в Генуа през 1922 година С членове на българската делегация на Конференцията в Генуа За разлика от марксизма Стамболийски пропагандира, че обществото се дели не на класи, а на съсловия. Тъй като най-голямото съсловие в България е селското и тъй като държавата съществува и се развива благодарение на неговия труд, то и

1908 година
TIME

Австро-Унгарска империя

Александър Стоименов Стамболийски
български политик

Class:: български политик

Class:: Стамболийски в Париж през 1921 г. Източник: ДА „Архиви"

Роден: 1 март 1879 г.Славовица, Източна

Figure 3: A screenshot of the UI showing an annotated document. The left column displays the annotation coloring toggles. The center window is the interface of the extended TinyMCE editor. The right column shows information from the knowledge base for the entities in the document.

match the order in which they were opened. Listing 1 is a small example of misnesting. In (Simov et al., 2001) a similar issue is presented, but in XML, typically called "not well-formed XML document". The issues that may arise when styling the document or using another functionality, even built in the editor, are linked to the tags representing them. It may not be possible to style the desired chunks of text, for example: parts of heading text, paragraphs, and others, because there are multiple tags representing annotations, sentences, and others which overlap, leading to tag misnesting when styling parts of sentence with a lot of overlapping of annotations, when representing annotations with custom tags. Consider the example in Listing 2. If we need to style only the class-1 annotated part of the text, it becomes impossible because annotated class-2 is started in class-1. If styling is done with XHTML tags (spans, divs, italic, and other tags) due to limitations in tag nesting in XHTML the entire sentences must be styled, so the XHTML remains valid. Other custom mechanisms for styling, rules, or CSS can be used, which also depend on the chosen text editor.

In the implemented solution with a single new tag, the annotations are represented as a multiclass attribute, each token has a unique ID per document. We did not find any major responsiveness issues while working with longer documents. Other functionalities like sentences, tooltip, etc. are implemented using custom attributes to store desired information and behavior. In that context, changing annotation of text is actually a change in the class attributes of the tokens. (Listing 3)

Annotation suggestions are displayed to the user, which can be generated with database queries, LLMs or by traversal of the knowledge graph. The user can also review stored information for the suggestion in order to make the best decision for annotation and URL linking. Figure 2 shows an example of the suggestions.

We need to point out a subtle but crucial detail: TinyMCE uses a non-standard XHTML attributes internally, which may not show in the "View source" option, but causes confusion during development. One example of that is the usage of "data-mce-href" hidden attribute (instead of the direct usage of the "href" attribute) which is used to keep track of the original link, during editing, or transformations for different reasons.

3.5 Automatic annotation

The automatic annotation pipeline is a core feature of the system. When the user uploads a brand new document to the database, the NLP pipe first extracts the text, tokenizes it, and segments it into sentences.

The system pipeline then applies the language models that annotate the named entities and the event structure in the text, providing the initial annotations of the documents. We use our own pretrained and later fine-tuned BERT (Devlin et al., 2019) and T5 (Raffel et al., 2019) models for the tasks. The models were pretrained on 20B and 35B

Bulgarian corpora respectively.

```
<tok id="1" class="PER" neurl="Alexander
    Stamboliyski" titlenes="PER"
    sentence="1">Alexander</tok>

<tok id="2" class="PER"
    neurl="Alexander_Stamboliyski" titlenes="PER"
    sentence="1">Stamboliyski</tok>

<tok id="3" class="death birth occupation"
    titlenes="death birth occupation"
    sentence="2">(</tok>

<tok id="4" class="TIME death birth occupation"
    titlenes="TIME death birth occupation"
    sentence="2">1</tok>

<tok id="5" class="TIME death birth occupation"
    titlenes="TIME death birth occupation"
    sentence="2">March</tok>

<tok id="6" class="TIME death birth occupation"
    titlenes="TIME death birth occupation"
    sentence="2">1879</tok>

<tok id="7" class="death birth occupation"
    titlenes="death birth occupation"
    sentence="2">-</tok>

<tok id="8" class="TIME death birth occupation"
    titlenes="TIME death birth occupation"
    sentence="2">14</tok>

<tok id="9" class="TIME death birth occupation"
    titlenes="TIME death birth occupation"
    sentence="2">June</tok>

<tok id="10" class="TIME death birth occupation"
    titlenes="TIME death birth occupation"
    sentence="2">1923</tok>

<tok id="11" class="death birth occupation"
    titlenes="death birth occupation"
    sentence="2">)</tok>

<tok id="12" class="occupation"
    titlenes="occupation" sentence="2">is</tok>

<tok id="13" class="occupation"
    titlenes="occupation"
    sentence="2">bulgarian</tok>

<tok id="14" class="occupation"
    titlenes="occupation"
    sentence="2">politician.</tok>
```

Listing 3: An annotated sentence in the CLaDA-BG-HTML format. (The original sentence is in Bulgarian.)

The processing by the models is done on the sentence level. The BERT model is used for the recognition of named entities and classifies tokens in the classic BOI format. The names are later mapped to their specific URLs in the Knowledge Base. The best model we created achieves a macro-F1 score of 81.23%. Experiments regarding entity disambiguation with fine-tuning models for retrieval (bi-encoders and cross-encoders) are also made, but are still in an early stage and will be presented in the future.

The event extraction is done with the T5 model which processes the sentences and generates the event structure into a JSON compatible format. The output contains a list of events described by event type, event text span, and a list of roles and their text spans. The predicted texts are fuzzy matched to the input tokens of the sentence, in order to get the token ids of the spans. The model achieves an F1 score of 84.29% in the extraction of test events. The models are fine-tuned on the latest version of the Bulgarian Event Corpus (Osenova et al., 2022). The development of the corpus and the models is described in more detail in (Simov et al., 2025). The annotation subsystem is designed as an internal REST API which is called by the back-end server on every update of the documents. The annotation returned by the pipeline is then stored in the database and later cladaHTML is generated from it.

4 Workflow

The main workflow of the system is:

- The user uploads a document from docx/markdown or creates a plain document which is represented internally in cladaHTML.[10]
- The user can edit or style the document.
- When the document is saved on the server, it is sent to the LLM server for automatic annotation, then it is returned to the back-end and saved in the relational database form.
- After the document is processed, it is returned in cladaHTML to the UI, the user can edit, style, edit annotations, create a new annotation, etc. and of course save it again, create a new version for the document, download it locally, or share it with another user. A diagram is presented in Figure 4.
- When the user is working on a document, he/she could perform different types of search in the database for additional information related to the annotations, saved in the BGKG, or to access other documents.

5 Conclusions and Future Work

In the paper, we presented a web based annotation system that allows for editing and stylization of documents in a user friendly way. The system leverages the use of LLMs for automatic annotations and initial annotation suggestions. Our main contributions are: (1) implementation of an extension of

[10]The uploading of a set of many documents will be implemented as an offline services in the system.

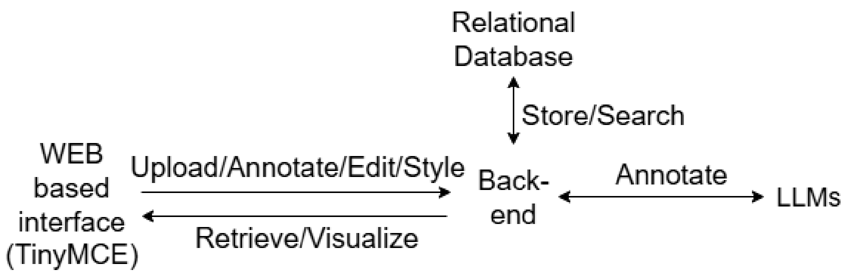


Figure 4: Interaction diagram of the subsystems.

XHTML to incorporate a token-based annotation of XHTML documents; (2) The TinyMCE rich text editor was extended to visualize the annotation of such documents and to allow for manual annotation. We are also working on manual modification of the automatic annotation; (3) A mechanism for annotation of XHTML with named entities and events. We think that in this way we provide the NLP technologies to the end users without a need for them to know the details of these technologies.

The specified internal document format cladaHTML is compatible with the standard features of the core TinyMCE editor, extending its functionality. The system is web-based, no installation is needed, easy to work with, and not computationally demanding. Developers should watch out for rich text editors adding hidden tags and merging multiple same-type tags, which may not appear even in source view.

In the future, we plan to work in two directions: (1) Integration with other components of the whole architecture, presented in the introduction; (2) Extension of the functionalities presented in the paper.

Plans for future work include: Support for importing from PDF, ability to do OCR and support for older or ancient languages. Support for exporting in docx format and as interactive document for embedding in web-pages in the format of: XHTML, CSS and JavaScript document, so some interaction with the document is possible outside the editor. Stylization of plain documents with LLMs.

We plan to extend the LLMs to support the editing process for spell checking, linguistic ambiguity, and others as needed. Although in the paper we referred to CoNLL in the context of Universal Treebanks we believe that a format based on tokens could incorporate not only syntactic annotation, but any annotations over text.

In the paper we provide integration of the implemented editor with a selected document. The more complicated searches that are represented shortly in the introduction. Currently we are working on a creation of RAG (Retrieval-Augmented Generation) system — see (Gao et al., 2024) for a Survey. Such a system will provide a more flexible way of searching the document database. In this way we will be able to rank the appropriate documents as mentioned earlier. We plan to implement a system to extract new knowledge from selected documents. The form of the knowledge will depend on the conceptual knowledge in BGKG — the ontological knowledge, the instance information and syntactic structure of the text. The significance of this new knowledge with respect to the selected documents will be determined by evaluating the extracted new facts as key ones.

Open-source version is considered after the production phase is achieved. For now, the system is tied to our requirements, but a modular approach can be implemented.

Acknowledgments

The reported work has been supported by CLaDA-BG, *the Bulgarian National Interdisciplinary Research e-Infrastructure for Resources and Technologies in favor of the Bulgarian Language and Cultural Heritage, part of the EU infrastructures CLARIN and DARIAH*. We also acknowledge the provided access to *the e-infrastructure of the Centre for Advanced Computing and Data Processing* (the Grant No BG05M2OP001-1.001-0003).

We would like to thank the CLaDA-BG team for the nice working atmosphere and their comments on the earlier versions of the text. Also, we would like to thank the reviewers for their valuable and insightful comments.

References

Kalina Bontcheva, Hamish Cunningham, Ian Roberts, Angus Roberts, Valentin Tablan, Niraj Aswani, and Genevieve Gorrell. 2013. Gate teamware: a web-based, collaborative text annotation framework. *Language Resources and Evaluation*, 47(4):1007–1029.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-augmented generation for large language models: A survey.

Jan-Christoph Klie, Michael Bugert, Beto Boullosa, Richard Eckart de Castilho, and Iryna Gurevych. 2018. The INCEpTION platform: Machine-assisted and knowledge-oriented interactive annotation. In *Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*, pages 5–9, Santa Fe, New Mexico. Association for Computational Linguistics.

Mariana Neves and Jurica Ševa. 2019. An extensive review of tools for manual annotation of documents. *Briefings in Bioinformatics*, 22(1):146–163.

Petya Osenova, Kiril Simov, Iva Marinova, and Melania Berbatova. 2022. The Bulgarian event corpus: Overview and initial NER experiments. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 3491–3499, Marseille, France. European Language Resources Association.

Colin Raffel, Noam M. Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Kiril Simov, Nikolay Paev, Petya Osenova, and Stefan Marinov. 2025. Bulgarian event extraction with llms. Presented at RANLP2025.

Kiril Simov, Zdravko Peev, Milen Kouylekov, Alexander Simov, Marin Dimitrov, and Atanas Kiryakov. 2001. CLaRK — an XML-Based System for Corpora Development. In *Proc. of the Corpus Linguistics 2001 Conference*, pages 558–560.