UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Departamento de Lingüística Aplicada
Departament de Lingüística Aplicada

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dept. of Applied Linguistics

Comparative study of computer tools for linguistic annotation of corpora

Master's Thesis

Master's Degree in Languages and Technology

AUTHOR: Gou , Hongling

Tutor: Periñán Pascual, José Carlos

ACADEMIC YEAR: 2021/2022

Departamento de Lingüística Aplicada
Departament de Lingüística Aplicada

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MÁSTER EN LENGUAS Y TECNOLOGÍA

Curso Académico: 2021/2022

TÍTULO TRABAJO FIN DE MÁSTER:

Comparative study of computer tools for linguistic annotation of corpora

AUTORA: Hongling Gou

Declaro que he redactado el Trabajo de Fin de Máster "Comparative study of computer tools for linguistic annotation of corpora" para obtener el título de Máster en Lenguas y Tecnología en el curso académico 2021-2022 de forma autónoma, y con la ayuda de las fuentes consultadas y citadas en la bibliografía (libros, artículos, tesis, etc.). Además, declaro que he indicado claramente la procedencia de todas las partes tomadas de las fuentes mencionadas.

Firmado: Hongling Gou

DIRIGIDO POR: Dr. Carlos Periñán Pascual

ACKNOWLEDGEMENTS

Upon finishing this thesis, I would like to express my great gratitude to all those who have offered me sincere assistance during this year.

First and foremost, my hearty thanks go to my tutor, Professor Dr. Carlos Periñán Pascual, who has given me insightful suggestions and constant encouragement both in my study and in my life.

Also, I owe my thanks to all the professors who have taught and enlightened me during my study at UPV, for guiding me in the field of research work, which is both challenging and fantastic.

The experience and profit I obtained will be of great importance to my further studies.

ABSTRACT

Computer tools play a significant role in corpus annotation, since software suitable for linguists improves the efficiency of annotation, increases the user experience, and contributes to generating professional results. This research compares current corpus-annotation tools from a qualitative perspective to evaluate availability, interactivity, and functionality. In particular, this research presents an innovative method to compare this type of computer tools using the XML, XSL, and HTML technologies. In this regard, XSL is applied to XML files generated by corpus-annotation software so that HTML files can be automatically created to allow linguists to display different annotations marked with different colors on the browser.

Key words: computer tool, corpus annotation, XML, XSL, HTML

RESUMEN

Las herramientas informáticas desempeñan un papel significativo en la anotación de corpus, ya que un programa informático adecuado para los lingüistas mejora la eficiencia de la anotación, aumenta la experiencia de los usuarios y contribuye a generar resultados más profesionales. Esta investigación compara herramientas actuales de anotación de corpus desde una perspectiva cualitativa con el fin de evaluar su disponibilidad, interactividad y funcionalidad. En concreto, esta investigación presenta un método innovador para comparar este tipo de herramientas informáticas a través del uso de las tecnologías XML, XSL y HTML. En este sentido, XSL se aplica a los archivos XML generados por los programas de anotación de corpus de forma que se puedan construir automáticamente archivos HTML que permitan a los lingüistas mostrar en el navegador las diversas anotaciones marcadas con colores diferentes.

Palabras clave: herramienta informática, anotación de corpus, XML, XSL, HTML

RESUM

Les eines informàtiques exerceixen un paper significatiu en l'anotació de corpus, ja que un programa informàtic adequat per als lingüistes millora l'eficiència de l'anotació, augmenta l'experiència dels usuaris i contribueix a generar resultats més professionals. Aquesta investigació compara eines actuals d'anotació de corpus des d'una perspectiva qualitativa amb la finalitat d'avaluar la seua disponibilitat, interactivitat i funcionalitat. En concret, aquesta investigació presenta un mètode innovador per a comparar aquest tipus d'eines informàtiques a través de l'ús de les tecnologies XML, XSL i HTML. En aquest sentit, XSL s'aplica als arxius XML generats pels programes d'anotació de corpus de manera que es puguen construir automàticament arxius HTML que permeten als lingüistes mostrar en el navegador les diverses anotacions marcades amb colors diferents.

Paraules clau: eina informàtica, anotació de corpus, XML, XSL, HTML

TABLE OF CONTENTS

INDEX OF FIGURES

TABLE INDEX

1. INTRODUCTION

Corpus annotation is recognized as the most fundamental academic research worldwide. Through corpus annotation, linguists can not only analyze linguistics but, most importantly, they can facilitate international communication by analyzing different languages. As a result of the rapid development of computer science in recent years, linguistic annotators have been applying corpus computer tools in line with that development.

Availability, interactivity, and functionality are the three basic aspects of computer tools that need to be evaluated. Although most computer tools are free-access and can be easily downloaded from official websites, installation in different operating systems can be complicated. In addition, the interactivity of the computer tool is crucial. For example, computers tools can be very user-friendly for inexperienced corpus annotators if they include a user manual and guides users step by step. The functionality of the computer tool is of most importance. For example, for an annotator with no experience in corpus annotation, it is more appropriate to choose a corpus computer tool with a simple interface and guidelines. However, for a professional linguistic annotator, the more advanced features of the corpus computer tool are more important, because there are a large number of corpus annotations that need to be manually annotated over time, and it is more important that annotators can go back to the previous annotation interface to continue annotation.

Therefore, a suitable computer tool plays a very important role in corpus annotation, not only to improve the efficiency of annotation, but also to increase the user experience and to generate professional annotation files as a basis for further academic research. For this reason, this study designed two academic tasks to test these three aspects of the computer tool. Task 1 designed a sample corpus and implemented the corpus annotation using three computer tools. All steps were recorded to compare the three computer tools. Table 1 shows the results of comparing these three computer tools in different aspects. Task 2 creates the XSL file, converts the generated XML file into an HTML file, and displays the annotations marked with different colors in the browser. As a result, Table 2 shows the results of the comparison. This study applies the technologies XML, DTD, XSL and HTML to propose a corpus annotation based on morphosyntactic and syntactic annotations, where the two levels of corpus annotation are part of speech (POS) and verb tense.

In summary, this research has three parts. Section 2 provides a literature review of corpus levels, standards and computer tools, Section 3 describes the two tasks and compares the corpus tools and draws conclusions in the form of two tables describing the comparison between the tools MAE, UAM and DEXTER. Finally, Section 4 concludes with a general review of the theoretical and practical parts of the research.

2 CORPUS ANNOTATION

This section introduces four parts about corpus annotation: the levels of corpus annotation, standards for corpus annotation, theoretical techniques, and corpus computer tools.

2.1 Levels

Linguistic information about grammatical categories of words, syntactic structures of sentences, speech acts and semantic information can be annotated, bringing added value to professional research in academic and social fields. The classification of the different linguistic levels of corpus annotation is crucial as a foundational theory for annotation.

Today, modern corpora are created in electronic format, making it easier to store linguistic resources. In this technological context, corpus computer tools are increasingly used in linguistics. Therefore, it is important to present a comparative study of computer tools applied to the field of linguistic annotation. For this reason, this section describes the theoretical background of the level of corpus annotation.

With the rapid development of information technology, corpus annotation at different linguistic levels has been applied in various related research fields. According to previous research, there are seven levels of corpus annotation:

> The different sounds used by a language are described at the level of phonology. The writing system is described at the level of orthography. Morphology describes the formation and inflection of individual words. Syntax describes the ordering of words and their combination into phrases and sentences. Semantics analyzes the meaning of individual words (lexical semantics) and the meaning of phrases and sentences (compositional semantics). How words and phrases are used to make things happen is the level of pragmatics. How people and things are introduced as topics and subsequently referred to in later utterances is the level of discourse. (Wilcock, 2009, p. 29)

Therefore, this section provides an overview of the main concepts and practical examples related to different levels and standards of corpus annotation. This section also presents the status of corpus annotation at different levels, including but not limited to definitions of levels, examples of corpus computer tools, and an overview of future developments in the field.

2.1.1 Phonological and orthographic annotation

Wilcock (2009) explained that phonology and orthography deal with the smallest units, that is, individual sounds and letters, respectively. Thus, orthographic and phonology levels of annotations are related according to the direction of research. According to Gires (2017), an orthographic transcription can generate the phonemic annotation automatically by utilizing a pronunciation lexicon and/or rule-based algorithms. He argued that orthographic annotation provided a methodology for solving linguistic research in phonological annotation. The orthographic level of annotation focused on the written words and the spellings, and the annotation tasks were typically tokenization and sentence boundary detection (Wilcock, 2009). As explained by Attia (2017), tokenization is used to identify token boundaries and is automatically performed with a limited set of token delimiters: space and punctuation symbols.

There were annotation tasks on phonological level. For example, various phonology applications can be distinguished at different levels of representation for acoustics and linguistic functions. Hirst (2006) argued that, between the physical acoustic signal and a functional representation of linguistic meaning, there were three intermediate levels: phonetic representation, surface phonological representation, and underlying phonological representation.

However, Gries (2017) claimed that research on paralinguistic aspects of speech is in the initial steps, and prosodical corpus annotation is still not mainstream in corpus linguistics.

2.1.2 Morphology level annotation

According to Wilcock (2009), morphological annotations focus the formation and inflection of every single word. For example, part-of-speech (POS) tagging is a task under the theoretical basis of morphology.

From a more general perspective, morphology and lexicography are, in a sense, related and cannot be separated. Thus, when linguistic hierarchies are applied to categorize corpus annotations, such annotated linguistic corpora are similar even though they are under different levels of linguistic annotations. In other words, when a human annotator selects a linguistic annotation level and then builds a corpus based on the selected linguistic level, other linguistic categorizations are also reflected in some practical parts of research and are shown in the final results. More specifically, for example, when creating a lexical corpus, POS tagging is also used, and the lexical-level annotated corpus that was initially created to build a lexical corpus for lexical meaning analysis may also yield morphological results in its final analysis. For instance, the topic chosen by a research group can be to analyze grammars utilizing the syntactic level of corpus annotation, but (i) we can study and illustrate syntactic phenomena morphologically, deriving some indices of the influence of different grammar on word formation, or (ii) we can study the composition of sentences at the syntactic level efficiently.

According to the classification of linguistic levels (Wilcock, 2009), this article classifies corpus annotation into seven linguistic levels. However, corpora are not annotated at one particular linguistic level in practical tasks. Of course, this is not only a problem encountered in our research, Starko (2020) found that semantic annotations are a natural complement to morphological annotations, and after the whole process of semantic annotation, he claimed that the application of the combination of morphological (POS) and semantic tagging could be a significant development of linguistic annotation and a variety of natural language processing (NLP) applications. In corpus annotation, there are many similar cases. For example, the TIGER Corpus Navigator also used syntax annotation to identify target classifications even though it is a semantic website system (Hellmann, 2010).

2.1.3 Syntax level annotation

Syntax describes the ordering of words and their combination into phrases and sentences. Syntactic annotation is at the level of words, phrases, clauses, and the combination of each part. Based on this basic theoretical knowledge, those annotation projects related to the formation of a whole sentence and the relationship between each part of a sentence belong to the level of syntactic annotation. As mentioned above, annotation can be applied to an interdisciplinary field of study. For example, Tateisi (2005) and his team focused on the syntax level of annotation with the study of relations between proteins and genes in cooperation with biologists for further scientific development in the biomedical field. Based on original texts in the biomedical domain, the corpus containing syntactic annotation was created for improving NLP software in bio-text mining for information extraction. Following the Penn Treebank II (PTB) scheme, the primary texts of the GENIA Corpus were manually annotated with an XML editor and then converted to a PTB format before being merged with the POS annotation of the GENIA Corpus (Tateisi, 2005).

While syntactic ways of annotation have the advantage of simplifying the process of annotation, those surface syntactic annotation methods are essential to provide deeper information using semantically oriented annotation, and when "deeper" is mentioned, there are various aspects in different linguistic frameworks. Candito (2014) and his linguistic team define a deep syntactic representation scheme for French based on the SEQUOIA corpus. Focused on a deep annotation scheme and a deeply annotated corpus by analyzing the surface dependency trees of the Sequoia corpus, the free-access DS (Deep Sequoia) corpus was created for corpus linguistics studies and further semantic analysis. The DS corpus, developed by team members located in two different French towns (i.e. Paric and Nancy), was iteratively and collaboratively produced. In particular, a complete annotation of the corpus was constructed independently in both towns and then the final results were collaboratively agreed on.

There are also numerous creative and practical ways of using syntactic annotation in different fields. For example, also based on the SEQUOIA corpus, Fort (2014) presented the design of Zombilingo, a Game With A Purpose (GWAP) that provided ways of syntactic annotation. With the view that using a GWAP is an alternative way to the traditional and costly human annotation, after comparing several games such as PhraTris6 and Phrase Detectives, Fort (2014) decided to create a new GWAP, despite the discouraging results of previous projects. Furthermore, to build the corpus annotated at syntax level and free access, the resources were freely available texts by selecting Wikipedia articles from public domain resources. For this reason, applying syntactic annotation to make an free-access corpus for information extraction for initial study in different domains of research is an typical task in syntactic annotation.

2.1.4 Semantic level annotation

Semantics analyzes the meaning of individual words and the meaning of phrases and sentences, and semantic annotation is the level of dealing with the medium-sized units such as words, phrases, and sentences. However, according to Wilcock (2009, p. 30): "at the higher linguistic levels of semantics, pragmatics and discourse, there are numerous different theories and it is difficult to find a clear consensus for use in the practical tasks of corpus annotation." Palmer (2005) and his team have clearly explained the difference between semantic and syntactic analyses in the aspect of parsers that represent the meaning of texts. By comparing these two sentences "John broke the window." and "The window broke," Palmer (2005, p. 1) explained that: "a syntactic analysis will represent the window as the verb's direct object in the first sentence and its subject in the second but does not indicate that it plays the same underlying semantic role in both cases". Therefore, semantic annotation and syntactic annotation focus on different aspects of linguistic annotation.

During academic research in recent decades, semantic annotation played a significant role in annotating natural language texts. For example, in the study by Starko (2020), semantic annotations were used for Ukrainian with a taxonomic approach and were solved by insights from human natural language categorization. The project put forward a way of creating a semantic lexicon via annotating semantic annotation and then matching it with VESUM (Large Electronic Dictionary of Ukrainian). Starko (2020, p. 1) stated that: “the semantic lexicon will be used by the TagText tagger (both tools developed by the r2u team) to add the semantic annotation to the GRAC corpus”. The semantic annotation provided opportunities for other scholars to further explore semantic classes and other tasks in NLP software and applications. This research played an essential role in the development of semantic annotation application of solutions in the social field. Semantic tags were helpful for applications that could be applied in corpus linguistics and other domains, such as named entity recognition and information extraction. In addition, GRAC, the General Regionally Annotated Corpus of Ukrainian, was not only used in the academic field of linguistics but also NLP communities.

2.1.5 Pragmatic/discourse level annotation

According to Wilcock (2009, p. 29): “how words and phrases are used to make things happen is the level of pragmatics. How people and things are introduced as topics and subsequently referred to in later utterances is the level of discourse”. Although Wilcock separately defined the pragmatical level of annotation and the discourse-level of annotation, they were not separated clearly in many practical projects of corpus annotation.

The Penn Discourse Treebank (PDTB), a classical discourse-level annotation project, aimed to produce a large-scale corpus. In the project, connectives and arguments were annotated to expose a clearly defined level of discourse structure (Miltsakaki, 2004). Four classes of connectives, subordinating and coordinating conjunctions, adverbials and implicit connectives, and arguments were defined and illustrated to prove that a structure-clarified corpus could be annotated at the discourse level. Most discourse annotations were used to provide linguistic information and statistical support in NLP basing on PDTB.

For decades, the word-level annotation has been applied to artificial intelligence, machine translation, data analysis, segment source code, medical schedules, and other new sciences of interdisciplinary fields to provide practical solutions to social and scientific research problems.

However, unlike PDTB, which focused more on the linguistic markers of coherence relations, “Cognitive approach to Coherence Relations (CCR) was originally proposed as a set of cognitively plausible primitives to order coherence relations, but is also increasingly used as a discourse annotation scheme” (Hoek, 2019, p. 1). Recently, Hoek (2019) provided a new CCR to illustrate the application of the cognitive approach to coherence relations, and Hoek's new research significantly contributes to adding “disjunction” as a new distinction to CCR.

Corpus annotation has seen significant diachronic changes from annotating standardized text in journals to relatively informal text in novels, from written text annotation to spoken dialogues. Despite those manually annotated text corpus projects, the application of discourse annotation in the field of videos and their transcribed texts is the mainstream usage of discourse annotation in the new decades. For instance, a discourse-level annotation based on planned spoken monologues applying the style of PDTB was executed by Long (2020) to study and display the annotation results of distributions of discourse relations and senses, which differed from the corpus containing standard written text only.

2.2 Standards

The standard of establishing a corpus is an essential aspect during the initial step of corpus creation as the annotation standard helps to standardize the annotation tasks. Scholars were researching how to automatically adapt one annotation to another in order to reduce human input as the annotation tasks are always under different annotation standards (Jiang, 2009). In his research, Jiang provided a strategy for an automatic adaptation of annotation standards. The adaptation between the different standards of annotation tasks, such as treebanks and sequence labelling, was still hot research in the NLP domain, and it would be a tremendous support for saving human input on corpus annotation.

In recent decades, there have been numerous ideas for designing schemes for testing the annotation standards automatically. For example, Flickinger (2008) believed that the significant role of a target annotation scheme was that it should be under the theoretical and practical aspects of facilitating the automatic comparison of results across frameworks and supporting the final evaluation of results against the corpus annotation standards.

According to Müller(2002, p. 1): “The result is that in many research projects software is created from scratch, used once for a particular corpus processing task, and is then discarded”. Corpus annotation standards are very important because annotation files that adhere to annotation standards can be reused. In view of the foregoing, this section provides the theoretical basis of corpus annotation standards and describes two annotation standards, Text Encoding Initiative (TEI) and Expert Advisory Groups on Language Engineering Standards (EAGLE).

2.2.1 TEI

Adherence to corpus annotation standards is an important prerequisite for creating corpus annotations. A well-known and widely used corpus annotation standard is the Text Encoding Initiative (TEI). TEI promotes coding standardization, it allows more scholars to use the same coding principles and markup language to facilitate standardization and validation of the accuracy and consistency of XML files, as well as to provide a means for information software to access the corpus for further applications. Wynne (2005) also mentioned four significant parts of the TEI Header, a file description, an encoding description, a profile description, and a revision description.

The TEI standard is one of the important corpus annotation standards, which is characterized by comprehensibility, flexibility and extensibility. The TEI standard mainly provides a standard format for data exchange and guidance for encoding texts in this format. Burnards (1994) explained that the TEI encoding scheme is formulated with numerous models and DTD fragments, which can also be called tag sets. According to Burnadas (1994, p. 55): “the DTD fragments from which the main TEI DTD is constructed may be classified as core DTD fragments, base DTD fragments, and additional DTD fragments”.

Nowadays, the TEI standard is widely used by linguists for corpus annotation. For example, Maraoui (2017, p. 1) developed his research under the TEI standard because of its simplicity and concrete structure. The author declared that: “we choose to work with TEI because it has clear, simple and concrete structure. Further, the text encoding with TEI bases upon a well-maintained schema and offers a large spectrum of possible elements”. For this reason, a TEI-based model is presented for normalizing and improving the structure of the Hadith corpus in his research. Before presenting the experiment on the corpus, Maraoui (2017) stated an overview of TEI standards, the original goal of TEI is to support the development of a comprehensive set of standards for the preparation and transcription of electronic texts for corpus annotation. In addition, the author present the following sample for TEI.

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
<teiHeader>
  <!-- Header properties -->
</teiHeader>
<text>
<front> <!-- front information --> </front>
<body> <!-- main body --> </body>
<back> <!-- back information --> </back>
</text>
</TEI> (Maraoui, 2017, p. 2)
```

The elements <TEI>, <teiHeader>, <text>, <front>, <body> and <back> are structured by TEI standards. The simplified structure presents two parts: the TEI header and the text. First, according to Maraoui (2017, p. 2): the <TEI> element “contains all the information analogous to that provided by the title page of a printed text” and the <teiHeader> marked the metadata of the document. Second, the <text> element has three child elements <front>, <body> and <back>. The <front> element contains the front information, the <body> element contains the main body and the <back> element contains the back information.

The following is a sample of book information applying TEI standards:

```
<TEI xmlns="http://www.tei-c.org/ns/1.0">
<teiHeader>
    <fileDesc>[FileInformation]</fileDesc>
</teiHeader>
<book>
    <title>[BookTitle] </title>
        <publication>[JournalName]</publication>
    </book>
</TEI>
```

Complying with the TEI standard, this sample presents two parts: the TEI header and the book information. First, the <teiHeader> element contains the header of the document, the <fileDesc> presents the information of the file itself. Second, the <book> element presents detailed book information, i.e., the title of the book and the name of the journal that published the book. The <book> element contains subelements <title> and <publication> elements that present this book information respectively.

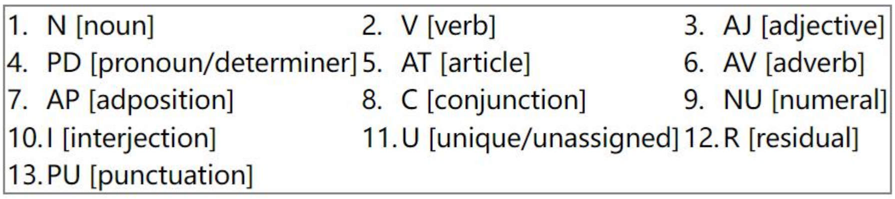
In conclusion, the above paragraph contains an introduction to TEI standards. According to the TEI consortium, there is a set of Guidelines that specify encoding methods for “machine-readable texts” in the field of linguistics and others.

We applied the TEI standard of corpus annotation for three main reasons to comply with our research:

a. Its basic knowledge of corpus annotation language is XML, and this research is creating the corpus based on XML language;
b. Its original goals are to simplify the corpus structure, and there is precisely a demand for improving the structure.
c. The generated XML files by computer tools follow the TEI standard.

2.2.2 EAGLE

Expert Advisory Groups on Language Engineering Standards (EAGLE) is a standard for different levels of corpus annotation. According to guidelines on the Recommendations for the Morphosyntactic Annotation of Corpora, two types of annotation most commonly applied to a text are morphosyntactic annotation and syntactic annotation. According to EAGLE (1996a), the guidelines of morphosyntactic annotation[1] is also defined as grammatical tagging, “whereby a label or tag is associated with each word token in the text, to indicate its grammatical classification”. The following Figure 1 shows the tagset guidelines.



1. N [noun]	2. V [verb]	3. AJ [adjective]
4. PD [pronoun/determiner]	5. AT [article]	6. AV [adverb]
7. AP [adposition]	8. C [conjunction]	9. NU [numeral]
10. I [interjection]	11. U [unique/unassigned]	12. R [residual]
13. PU [punctuation]		

Figure 1. Tagset guidelines of EAGLE.

Figure 1 shows tagsets. The capital letters represent tags, and the contents in square brackets explain the type of annotations indicated by tags. For example, capital N refers to noun tag, capital V refers to verb tag, capital AJ refers to the adjective tag, and so on. In addition, on EAGLE’S official website, there are explanations for the last three tags.

EAGLE (1996b) proposed guidelines for syntactic annotation[2]. For example, EAGLE (1996b) proposed several annotation layers And the following shows annotation eight layers:

Bracketing of segments: layer (a)

Labelling the category of segments: layer (b)

[1] The guidelines of morphosyntactic annotation can be found on the website of EAGLE. URL: http://www.ilc.cnr.it/EAGLES96/annotate/annotate.html

[2] The guidelines of syntactic annotation can be found on the website of EAGLE. URL: http://www.ilc.cnr.it/EAGLES96/segsasg1/segsasg1.html

Showing dependency relations: layer (c)

Indicating syntactic function labels: layer (d)

Marking subclassification of syntactic segments: layer (e)

Logical relations of various kinds: layer (f)

Information about the rank of a syntactic unit: layer (g)

Spoken language non-fluency phenomena: layer (h) (EAGLE, 1996b)

The A layer proposed the "syntactic integrity (sentences, clauses, phrases, words)" that is delimitated by square brackets. B layer described that "Segments can be labelled according to their syntactic function, such as Subject, Object, Adjunct..." C layer showed the dependency relations, which EAGLE (1996) explained as "head-dependent relations between words, e.g. adjectives and the nouns they modify". D layer proposed the annotation for syntactic functions, such as Subject, Object, and Adjunct. E layer "involves assigning feature values to phrases or words, e.g. marking a Noun Phrase as singular, or a Verb Phrase as past tense." According to EAGLE, "a singular (proper) noun phrase (Nns), and past tense verb phrase (Vd)". F layer "includes a variety of syntactic phenomena, such as co-referentiality (e.g. in control structures), cross-reference (or substitution), ellipsis, control, traces and syntactic discontinuity." G layer described the rank of annotation, which is "applied to general categories of constituents, words being of lower rank than phrases, phrases being of lower rank than clauses, and clauses being of lower rank than sentences." The last H layer is defined to annotate "a range of phenomena that do not normally occur in written language corpora, such as blends, false starts, reiterations and filled pauses." (EAGLE, 1996b)

Apart from these recommendations for morphosyntactic annotation, EAGLE also provided a Corpus Encoding Standard (CES)[3] based on XML. According to EAGLE (2000), the CES standard complies with TEI guidelines.

> The CES specifies a minimal encoding level that corpora must achieve to be considered standardized in terms of descriptive representation (marking of structural and typographic information) as well as general architecture (to be maximally suited for use in a text database). (EAGLE, 2000)

EAGLE (2000) proposed recommendations for corpus annotation in many aspects. It proposed the definition of the basic terms, such as “encoding”, “text” and “markup”. For example, according to EAGLE (2000), a text “is a piece of human language communication in the broader sense, that one has reason to consider as a whole.” and markup or tags “represent the interpretation of segments of text”. Furthermore, it proposed three levels of standardization, the metalanguage level, syntactic level and semantic level standardization. According to EAGLE (2000), a better way to “standardize texts is to specify precise tag names and syntactic rules for using the tags as well as constraints on content”. Finally, it also defined the types of information and several criteria for encoding.

2.3 Technology

This section describes five computational languages and specifications: XML, DTD, XML Schema, XSL, and HTML.

2.3.1 XML

XML is a computational language that is widely used in corpus annotation, and

[3] The Corpus Encoding Standard (CES) can be found on the website of EAGLE. URL: https://www.cs.vassar.edu/CES/

this research uses XML as the main encoding language to create a corpus for morphosyntactic level annotation. Indeed, the “familiarity with SGML or XML” is an essential technical principle for understanding the structure of the TEI scheme (Burnard, 1994). Thus, having a basic understanding of the structures of XML is also an essential theoretical basis in order to start to establish any corpus for any usage for various levels of corpus annotation.

XML, Extensible Markup Language, is a markup language that can be used to create customized tags. Furthermore, tags, elements, and attributes are three terminologies and basic structures used to describe XML (Tidwell, 1999). Below is a modified sample of a part of the XML document with the structure cited from Tidwell's (1999) introduction of an XML language for an initial understanding of XML.

```
<address>
    <name>
        <first-name>Hongling</first-name>
        <last-name>Gou</last-name>
    </name>
    <street>27 Street</street>
    <city state="VL">Valencia</city>
    <postal-code>46000</postal-code>
</address>
```

These tags should be structured by the left angle bracket (<) and the right angle bracket (>), and there should be two tags: starting tag and ending tag, for example, <address> and </address>; These elements are the starting tags and ending tags mentioned above and are structured by the brackets. <first-name> and <last-name> are the child elements of <name>.

Within the element, attributes provide information concerning the element, and the information provided needs to be enclosed in quotation marks. For example, “state” is the attribute of the <city> element, containing the “VL” value. In the following we present two samples for introducing the attribute in an element:

```
Sample A)
<Book color= “red”>
    <BookTitle>Poems of Shakespeare</BookTitle>
</Book>

Sample B)
<Book>
    <BookColor>Red</BookColor>
    <BookTitle>Poems of Shakespeare</BookTitle>
</Book>
```

Sample A has two elements: <Book> and <BookTitle>, in which the <BookTitle> element is the child element of the <Book> element, “color” is the attribute of the <Book> element, providing the information of book color. On the contrary, sample B has three elements: <Book>, <BookColor> and <BookTitle>, in which <BookColor> and <BookTitle> are the child elements of the <Book> element. The <BookColor> element provides the book color information. In sample A, the color of the book is presented in the attribute “color” while in sample B the value of color is presented in element <BookColor>, so the annotator can use both attribute and element to provide the same information.

The following paragraphs present different types of XML of annotated corpus samples:

Sample C)

```
<?xml version='1.0' encoding='utf-8'?>
<document>
    <header>
        <file_name>Mycorpus.txt</file_name>
    </header>
    <body>
        <annotation id='1' features='grammar'>Hmmm</annotation>
        <annotation id='2' features='grammar'>Emmm</annotation>
        ...
        <annotation id='200' features='grammar'>Ahh</annotation>
    </body>
</document>
```

Regarding sample C), this XML document generated from corpus annotation has two parts to its structure. In the first part, the <header> tag contains the child tag <file_name>, which contains the information of the name of this XML file.

The second part is the <body> tag, which contains two hundred <annotation> child elements. The <annotation> tag has two attributes. First, the attribute id defines the serial number of the annotation, which is normally auto-generated by the computer tools. Second, the features attribute presents the layer of annotation, it is “grammar” in this case, and it could be POS or verb tenses in this task. The value of the <annotation> tag is the annotated text. For example, the <annotation> tag whose id presents “200” and has been annotated with the grammar layer containing the value “Ahh”, and “Ahh” is the annotation text.

Sample D)

```
<?xml version="1.0" encoding="UTF-8"?>
<dexter_code_set>
```

```
  <metadata>
    <file>Text3.xml</file>
    <modified>22-5-9 12:07</modified>
  </metadata>
  <data>
        <type name="verb" red="255" green="128" blue="255" visible="true">
              <token>
                    <start_id>b.10.1</start_id>
                    <start_index>106</start_index>
                    <start_string>If</start_string>
                    <end_id>b.10.1</end_id>
                    <end_index>174</end_index>
                    <end_string>one</end_string>
              </token>
<token>
...
</token>
        </type name="noun" red="128" green="255" blue="255" visible="true">
<type>
...
</type>
  </data>
</dexter_code_set>
```

Regarding sample D, there are also two parts in the structure of XML files, the <metadata> and the <data>. In the first part, the <metadata> tag contains the child element <file> and <modified>, the <file> defines the name of this XML file, and <modified> element presents the time when this XML file is modified.

In the second part, the <data> element contains all the data of annotation, which is the main body of this XML document. In the <data> element, there are child elements <type>. <type> elements refer to different annotation tags. For example, the attribute name of the first <type> has the value “verb”, which defines that this <type> element contains the information of annotation for verbs. The attribute name of the second <type> element has the value “noun”, which defines that this <type> element contains the information of annotation for nouns.

Sample D contains the annotation information in the <token> element, which has six child elements, <start_id>, <start_index>, <end_id>, <end_index> and <end_string>. These six child elements contain the annotation information , but they present the position of the annotation in the original XML files instead of presenting the annotated text. The <start_id> and <end_id> contain the serial number information of the start and end positions; the <start_index> and <end_index> contain the information of the position of annotated text in the original XML document; the <start_string> and </end_string> present the string before and after the annotation.

In this way, the computer locates the position of the annotated text. For example, in this <token> element, its child elements define the location of annotation. The id number is “b.10.1”, the location index from 106 to 107 in the XML document, and the annotated text starts with the text “if” and ends with “one”.

2.3.2 DTD

Document Type Definition (DTD) plays an essential role in forming the basic technical knowledge of corpus annotation. As defined by Goldberg, “A DTD, or Document Type Definition, is an older, but widely used system with a peculiar and limited syntax” (Goldberg, 2010, p. 94). DTD principles help standardize and validate the markup languages and clarify the structure. It is also a personalized and customized standard that can be applied in different fields of corpus annotation.

According to Goldberg (2010), DTD can define various elements, other occasions of choice, and occurrences. Below we present five kinds of elements and examples for DTD rules.

a. DTD principles can define an element that contains text, which means its content and value should only be text, but no other form of value.

Eg. <!ELEMENT author (#PCDATA)>

<!ELEMENT name (#PCDATA)>

b. DTD principles can define an empty element, which means its value should be empty.

E.g. <!ELEMENT image EMPTY>

<!ELEMENT date EMPTY>

c. DTD principles can define an element that contains a child element, which is an element that belongs to its father element. Thus, in the example below, the "color" element should be the child element of <book_colors> and with these DTD rules, the <book_colors> element only exists as one child element.

Eg. <!ELEMENT book_colors (color)>

d.If one element has many child elements, such as book name, author, date of publication, book color, and book image. DTD can also be used to define it as below:

E.g. <!ELEMENT book_info (name, author, date, color, image)>

d. In some cases, DTD can be used to define an element that contains anything, which means no rules apply to the value of this element. In other words, the value can be text, data, or symbol.

Eg. <!ELEMENT book_content ANY>

<!ELEMENT book_classification ANY>

The following shows sample A about student information.

A.

```
<!ELEMENT student (university_name,gender,age,nationality,emai_addl)>
<!ELEMENT university_name (#PCDATA)>
<!ELEMENT gender(#PCDATA)>
<!ELEMENT age(#PCDATA)>
<!ELEMENT nationality(#PCDATA)>
<!ELEMENT email_add(#PCDATA)>
```

Sample A has an element <student>, which has five child elements describing the student information in the aspect of university name, gender, age, nationality and email address.

The following shows sample B of a DTD file standardizing the XML document about book information.

B.

```
<!ENTITY name "book">
<!DOCTYPE book[
<!ELEMENT book_info (title, abstract)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT abstract (short, long)>
<!ELEMENT short (#PCDATA)>
<!ELEMENT long (#PCDATA)>
<!ATTLIST book id CDATA #REQUIRED>
]>
<document id="123">
    <title>Un título</title>
    <abstract>
        <short>short abstract</short>
        <short>long abstract</short>
    </abstract>
</document>
```

As for sample B, the first line describes the name of this entity as “book”. The codes from line 3 to line 8 define the content of book information, from which the <book_info> element contains the title and abstract. In addition, there are two types of abstract, which are long and short types defined in the fifth line. The data type is also defined in DTD. The element <title>, <short> and <long> are followed by “#PCDATA”, which means the data type of these elements are the plain texts.

In addition, there are more detailed principles for DTD standards in Goldberg's (2010) quick start guide to explain how DTD works to validate that the XML documents are deemed valid, and the value of each element in the XML documents is well-formed, strictly as the DTD documents require.

2.3.3 XSD

According to Campbell (2003), the XML Schema Definition (XSD) “represents the metadata for the associated XML document or class of XML documents.” He also pointed out that “XML Schema’s goal was to extend the capabilities of the

DTD (Data Type Definition).” On the W3C XML Schema Web Site, two guidelines can be found for XML Schema, the structure of XML Schema[4] and the datatypes of XML Schema[5].

On the one hand, XML Schema and DTD are both computational languages that provide a format to validate the content of XML documents. On the other hand, there are differences between them. First, “XML Schema is more verbose than a DTD that performs the same function.” This could be the advantage of DTD in some cases, but XML Schema could provide more solutions in more complicated cases where “data typing is an important issue”. Second, Campbell also explained that “the major difference between the DTD and an XML Schema is the fact that the DTD is not an XML document; it is written in Extended BNF notation. By contrast, the XML Schema is itself an XML document.”

This research applies DTD technology as a simplified DTD file is required to validate the generated XML sample. According to Campbell's theory, the use of XSD technology will result in more verbose coding in this case.

2.3.4 XSL

According to Clark (1999), XSL “specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary.”

XSL is a computational language that modifies the XML document. Each computing language has its own syntax and rules, and XSL also has a concise and strict syntax and rules. The most essential step in creating an XSL file is to make a declaration by presenting a namespace at the top of the XSL file.

[4] The website of structure of XML Schema. URL: https://www.w3.org/TR/xmlschema11-1/

[5] The website of the datatypes of XML Schema URL: https://www.w3.org/TR/xmlschema11-2/

According to Clark (1999), “XSLT processors must use the XML namespaces mechanism [XML Names] to recognize elements and attributes from this namespace”. Apart from the namespaces, syntax and rules are most important. The following paragraphs describe several samples of XSL syntax and rules.

Clark (1999) defined that “the content of the xsl:template element is the template that is instantiated when the template rule is applied.” Sample A describes the syntax of the <xsl:template> element.

A.

```
<xsl:template match = pattern name = qname>
        <!-- Content: (xsl:param*, template) -->
    </xsl:template>
```

Clark (1999) defined that “an xsl:call-template element invokes a template by name; it has a required name attribute that identifies the template to be invoked.” Sample B describes the syntax of <xsl:call-template> instruction:

B.

```
<xsl:call-template name = qname >
        <!-- Content: xsl:with-param* -->
</xsl:call-template>
```

Clark (1999) also described that “the xsl:apply-templates element recursively processes the children of the source element,” and if there is no attribute “select”, “the xsl:apply-templates instruction processes all of the children of the current node, including text nodes.” Sample C describes the syntax of <xsl:apply-templates> instruction:

C.

```
<xsl:apply-templates>
        <xsl:apply-templates select = node>
        <!-- Content: (xsl:sort | xsl:with-param)* -->
```

```
    </xsl:apply-templates>
```

Clark (1999) described that “The xsl:for-each instruction contains a template, which is instantiated for each node selected by the expression specified by the select attribute.” and the attribute “select” is required for this function. Sample D describes the syntax of <xsl:for-each> instruction:

```
D.  <xsl:for-each>
            <xsl:for-each select = node-set-expression >
            <!-- Content: (xsl:sort*, template) -->
    </xsl:for-each>
```

In XSL, variables and parameters are two instructions to which values can be assigned. Clark (1999) explained the difference between variables and parameter that “the value specified on the xsl:param variable is only a default value for the binding.” The attribute name is required for both instructions. The following samples E and F describes these two instructions:

```
E.
<xsl:variable>
            <xsl:variable name = qname select = expression >
            <!-- Content: template-->
</xsl:variable>
```

```
F.
<xsl:param>
            <xsl:param name = qname select = expression >
            <!-- Content: template -->
</xsl:param>
```

According to Clark (1999), the main purpose of the <xsl:key> element is to declare keys and he explained that “xsl:key element gives information about the keys of any node that matches the pattern specified in the match attribute." There are three attributes required in key elements, the name attribute, match attribute and use attribute. Sample G shows the rules of element key:

G.

```
<xsl:key name = qname
    match = pattern
    use = expression />
```

According to Clark (1999), "sorting is specified by adding xsl:sort elements as children of an xsl:apply-templates or xsl:for-each element" and the attribute of data-type defines two types of data:

> - text specifies that the sort keys should be sorted lexicographically in the culturally correct manner for the language specified by lang
>
> - number specifies that the sort keys should be converted to numbers and then sorted according to the numeric value; the sort key is converted to a number as if by a call to the number function; the lang attribute is ignored. (Clark, 1999:40)

During the process of programming, sorting is an essential step to set orders for the generated values. Sample H describes part of the rules of element xsl:sort that applied in this research.

H.

```
<xsl:sort
    select = string-expression
    data-type = { "text" | "number"} />
```

In XSL, there are two instructions to perform conditional processing, the xsl:if and xsl:choose. The difference between these two functions is that the xsl:if element defines conditional processing with only one condition while xsl:choose has optional conditions. In this research, we design the XSL with the xsl:choose element.

The xsl:choose element has three parts of code, xsl:choose, xsl:when and xsl:otherwise, where the xsl:otherwise element executes the optional conditions when the xsl:when is not true. As for this point, Clark (1999) explained in detail that “the content of the first, and only the first, xsl:when element whose test is true is instantiated. If no xsl:when is true, the content of the xsl:otherwise element is instantiated.” Sample I describes the syntax of xsl:choose instruction.

I.

```
<xsl:choose>
        <xsl:when test = boolean-expression >
            <!-- Content: template -->
        </xsl:when>
        <xsl:otherwise>
            <!-- Content: template -->
        </xsl:otherwise>
</xsl:choose>
```

The <xsl:when> element and <xsl:otherwise> elements are child element of the <xsl:choose> element. By testing the “boolean-expression”, the result presents a value that can only contain the value of “true” or “false”. This step decides this conditional processing.

Apart from these instructions, two functions have been applied in this research. The count function and substring function. The XSL count function is defined to count elements with specific attribute names defined in the XML document. In addition, the count function adopts transformation by means of matching the templates in a style sheet against an XML document. Sample J shows the syntax of the count function.

```
J.    <xsl:value-of select="count(root/child/line[name='attribute value'])" />
```

2.3.5 HTML

Hyper Text Markup Language (HTML) is computational language that provides the technology for publishing information. According to W3C[6], the publishing language used by the World Wide Web is HTML. According to the guidelines, there are four means that HTML offers to publish information and this research complies with the first method. The task in the next chapter applies HTML language to “publish online documents with headings, text...”

The following sample presents the structure of a HTML file that is related to this task.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <h1>This is the most important heading</h1>
<h2>This is heading two</h2>
        <p>This is paragraph one.</p>
<sentence>This is sentence one </sentence>
<sentence>This is sentence two</sentence>
...
```

[6] The World Wide Web Consortium (W3C) is the main international standards organization for the World Wide Web. According to W3C, there are guidelines for HTML. URL: https://www.w3.org/TR/html401/

```
<p>This is paragraph two.</p>
</body>
</html>
```

Having regard to the above code, <html>, <head>, <body> and <p> tags are the basic four tags of HTML. HTML separate the header and the body into two parts in the structure, the head part and the body part.

According to the above sample, the child element <title> of the <head> tag contains the value of page title. The <body> tag contains the main body of the HTML file and its child element <h1> contains the heading of the HTML file; its child elements <p> tags contain the <sentence> tags, which present the original text.

2.4 Software

There is a diversity of computer tools, and different types of computer tools meet different academic needs. This section presents some of the computer tools as theoretical basis.

Choosing an appropriate tool for corpus annotation research is essential. Nowadays, the creation and analysis of the corpus has been improved and optimized by the development of information technology. Corpora was first generated by handwriting and were entirely manually compiled. Corpora has become more popular because scholars recognized that corpora significantly contributes to the storage, integration, and analysis of data and text, information sharing and extraction with information technology. However, linguists prefer to obtain technical support from computer tools, allowing them to spend more time on anticipatory annotation rather than learning how to prepare and use the software. With all of the information and free-access software emerging on the

Internet, there is a wide range of computer tools to be discovered and evaluated, so it is difficult to identify each one.

Neves (2021) argued that computer tools were critical for creating and assessing new NLP and information extraction methods in specific instances. Therefore, he chose 78 computer tools and evaluated them based on four criteria:

> Publication criteria. These criteria describe features related to both the tools ’ publications and to other publications referencing the use of the tool.
>
> Technical criteria. This group of criteria evaluates technical aspects of the software itself, such as source code availability and the easiness of installation.
>
> Data criteria. These criteria assess the input and output format of documents, schema and annotations. Functional criteria. In this group, we evaluated various criteria related to the functionality of the tools.
>
> Functional criteria. In this group, we evaluated various criteria related to the functionality of the tools. (Neves 2021:4)

Finally, Neves's (2021) proposed one annotation list for annotation tools: BioAnnotate, Callisto, Ellogon, Glozz, MMAX2, MAE, UAM Corpus, and WordFreak. This section briefly describes these standalone annotation tools.

2.4.1 BioAnnotate

According to a recent study, BioAnnotate is an effective software platform for automated word annotation in biomedical applications (Fernández, 2013). Fernández stated that BioAnnotate has three primary significant features that make it a versatile and adaptable open-source computer tool:

> (i) a rich client enabling users to annotate multiple documents in a user-friendly environment,
>
> (ii) an extensible and embeddable annotation meta-server allowing for the annotation of documents with local or remote vocabulary and
>
> (iii) a simple client/server protocol that facilitates the use of our

> meta-server from any other third-party application. (Fernández, 2013:1)

BioAnnotate is available on its official website[7].

2.4.2 Callisto

Callisto is a Java-based computer tool that is multilingual and multi-platform, with a modular design that allows for extensive customization. “It is built on the ATLAS architecture to promote extensibility and abstraction across the diversity of linguistic signals and their associated annotations” (Day, 2004). The computer tool and its modules are free access on the website[8].

2.4.3 Ellogon

The Ellogon computer tool is defined as a text engineering platform by Petasis.

> Ellogon is a new text engineering platform developed by the Software and Knowledge Engineering Laboratory of the Institute of Informatics and Telecommunications, N.C.S.R. “Demokritos”, Greece. Ellogon is a multi-lingual, cross-platform, general-purpose text engineering environment, developed to aid both researchers who are doing research in the natural language field or computational linguistics, as well as companies that produce and deliver language engineering systems. (Petasis, 2002, p. 1)

According to Petasis' research, Ellogon comprises three subsystems: a Collection and Document Manager (CDM), a sophisticated and user-friendly graphical user interface (surface), and a modular pluggable component system. Furthermore, according to Petasis (2002), the Ellogon contains eight features that make it an effective computer tool:

[7] The official website of BioAnnotate, URL: https://www.sing-group.org/bioannote/)

[8] The website of Callisto, URL: http://callisto.mitre.org

a. Support of multiple languages
b. Portability
c. Advanced surface
d. Modular Architecture
e. Interoperability with other platforms
f. Memory compression
g. Execution server
h. HTTP Server

Ellogon is also available on its website[9].

2.4.4 Glozz

According to Widlöcher (2012), “a formal framework supporting description of heterogeneous linguistic objects and structures, appropriate representation formats, and adequate manual computer tools” is required to satisfy the demand for creating and distributing reference annotations. Glozz, a comprehensive corpus computer tool, was designed to address specific demands. The essential feature of Glozz is its versatility, as other annotation systems do not allow for simultaneous editing, reading, and mining of annotations. Glozz is a complex and adaptive tool in that it incorporates an advanced tool, namely GlozzQL, that simplifies annotation tasks and allows for simultaneous annotating and querying. Glozz's website (www.glozz.org) has a detailed introduction and Yann Mathet published “Glozz User's Manual” in 2011.

2.4.5 MMAX2

According to Müller (2006), “MMAX2 is a highly customizable tool for creating, browsing, visualizing and querying linguistic annotations on multiple levels.” Müller (2006) explains how MMAX2 may be used to define an annotation scheme, create an annotation, assess inter-annotator agreement, and perform

[9] The website of Glozz, URL: https://www.ellogon.org/

the corpus query and transformation. Furthermore, when the MMAX2 tool is used for advanced NLP tasks, the MMAX2 Discourse API allows MMAX2 documents from the programming language Java to be processed for more complex operations than basic queries and modifications.

Müller (2006) demonstrated that the MMAX2 tool had previously been used in annotation tasks such as "POS tags, word senses, coreferences, disfluencies (in transcribed spoken language), grammatical dependency relations, and others."

In response to the requirement for a more streamlined operating procedure, three standalone computer tools are briefly described below for reference: MAE, WordFreak, and UAM Corpus.

2.4.6 MAE

According to Stubbs (2011), "MAE and MAI are lightweight annotation and adjudication tools for corpus creation." MAI is another computer tool that is frequently nominated with MAE. MAE is for Multi-purpose Annotation Environment, while MAI stands for Multi-document Adjudication Interface (MAI). Furthermore, due to their versatility, from the beginning they were both designed to be used for a multiplicity of annotation tasks.

The MAI and the MAE are similar in that they both employ the Java programming language and the SQLite database to store and extract annotation data (Stubbs, 2011). MAE and MAI can be utilized more efficiently for linguistic annotation by inexperienced annotators.

Furthermore, MAE may be used for document-level annotation. Document-level annotations are utilized for activities such as document categorization, and recognizing documents based on hallmarks and phrases (Neves, 2021).

2.4.7 Wordfreak

According to Morton (2003), “WordFreak is a natural language computer tool that has been designed to be easy to extend to new domains and tasks.” WordFreak was launched as a fundamental computer tool for linguistic annotations. Furthermore, as a Java-based and open-source program, WordFreak distributes the original code of WordFreak on its website[10], as well as screenshots and a deployable online version of this computer tool (Morton, 2003).

2.4.8 UAM Corpus

According to O’Donnell (2008), the UAM Corpus tool “allows the user to annotate a corpus of text files at a number of linguistic layers, which are defined by the user”, besides, O’Donnell (2008) explained that UAM has a lot of functions of “corpus search, automatic tagging based on lexical pattern matching, and production of statistics”, which makes it a useful and valuable software for annotating a text corpus.

One of the reasons for choosing UAM is that “UAM CorpusTool is free, and works on Macintosh and Windows” (O’Donnell, 2008). Our operating systems are mostly Windows. As a result, it is more convenient for us to initiate this research with a tool that is compatible with the operating system to facilitate the installation of the computer tool. There is considerable demand for a simplified computer tool because it would encourage inexperienced human annotators to focus on the annotation process rather than on core computer science such as operating systems or environment settings.

[10] The website of Wordfreak, URL: http://wordfreak.sourceforge.net

Furthermore, “stand-off annotation provides far better support for projects with multiple annotation layers of the same text, or annotations by alternative users” (O’Donnell, 2008). Using standalone annotation software in our study is critical in selecting computer tools. There are two types of annotations: in-line and stand-off. When XML annotations are inserted into the text, this is called in-line markup. In this way, all the text and annotations are mixed in the plain text, making it complicated to read and modify. On the other hand, the stand-off annotations are not annotated in the exact plain text but annotated for separating the text and annotations utilizing a stand-off markup.

> The great advantage of using stand-off annotations is that whatever specific annotation types are to be made, other annotation types can be added later because the original text is still available. (Wilcock, 2009, P. 28)

This program also provides a Corpus Search tool, which “allows the user to search for instances in the annotated corpus which matches some criteria.” (O'Donnell, 2008) According to the introduction on its website, “you can also search for segments CONTAINING another segment type, or containing one or more words of text.” There is a major trend today in social and academic requirements for information transmission and research. Providing a tool for searching directly within the computer tool saves time and effort for new annotators. In addition, the resources are available on its website[11].

2.4.9 Other Software

ANVIL, which was designed in 2000 for gesture research, is currently one of the most prominent open-source tools for video annotation research. Since ANVIL’s fundamental language is Java, its essential coding background requirement is JavaSE 8 or higher. Color-coded elements, cross-level connections, 3D

[11] The website of UAM, URL: http://www.corpustool.com/index.html

visualizations of motion capture data, and project tools can be used to facilitate the annotation process throughout the coding process.

DEXTER is a collection of software tools that aid in annotating linguistic data and it is user-friendly, free, and cross-platform. DEXTER comprises two primary components: Dexter Converter and Dexter Coder. The Dexter Converter offers technical assistance for converting the source text into XML documents. The Dexter Coder concentrates on changing tags and performing annotations. DEXTER includes a transform tool, allowing the user to focus on annotation instead of the time-consuming technology of learning XML language. The following experiment task will demonstrate the academic research of DEXTER. DEXTER is available on its official website[12].

3 RESEARCH TASKS

This research designs two tasks to test the availability, interactivity, and functionality of MAE, UAM and DEXTER. In Task #1, we designed a corpus sample and performed corpus annotation using three computer tools. In Task #2, we created XSL files to convert generated XML files into HTML files, which display annotations marked with different colors on the browser. The procedures are described so that these three computer tools can be compared.

3.1 Analysis of computer tools

Over the last few decades, corpus annotation software has gained much importance as a research instrument for linguists, and an appropriate annotation tool is important for linguists. This section includes the qualitative research of three different computer tools (i.e. MAE, UAM, and Dexter) to give an in-depth analysis of their advantages and disadvantages. There are two reasons why

[12] The website of DEXTER, URL: http://www.dextercoder.org

these three computer tools were chosen for examination. Firstly, they technically comply with the requirements of this research. Secondly, they are free-access computer tools.

3.2 Methodology

This section provides a qualitative analysis of exploratory research that used these three computer tools to capture the operational phases in real-time and compare the annotation process.

There is a wide range of software available today, and the software prerequisites differ depending on various academic requirements. First, since the technical requirements of the computer tools differ, preparing them for undertaking research before actually starting to annotate differs. For example, the MAE tool requires preparing the DTD document before the annotation procedure, whereas Dexter tools require plain text to be transformed into XML documents.

Second, apart from different academic requirements, evaluating the criteria of different is an essential procedure before annotation. For example, the annotation program has various criteria for generating tags. Some annotation technologies require the generation of annotation tags by hand, while others create the tags automatically. Furthermore, the software supports a variety of annotation methods. For example, some software contains shortcut keys that help to annotate with only one key whereas other software requires the text to be selected and then a click is needed to annotate.

This section analyzes and reports each step to explore and compare the availability, interactivity, and functionality of computer tools, thus, several aspects are evaluated, including downloading, installing, utilizing, and eventually exporting the results. The testing steps start with preparing the technology for the download, installation, annotation process, and output results. Each aspect of the annotation task was tested and recorded during the development of the work. Additionally, the testing results of the computer tools are finally displayed in a table. The final analysis and comparison of the suggested procedure will focus on the features above, and the difficulties that occurred during the task are also explained.

In detail, there are six steps for testing and comparing the computer tools in the first part of the task. Figure 2 shows the procedure.

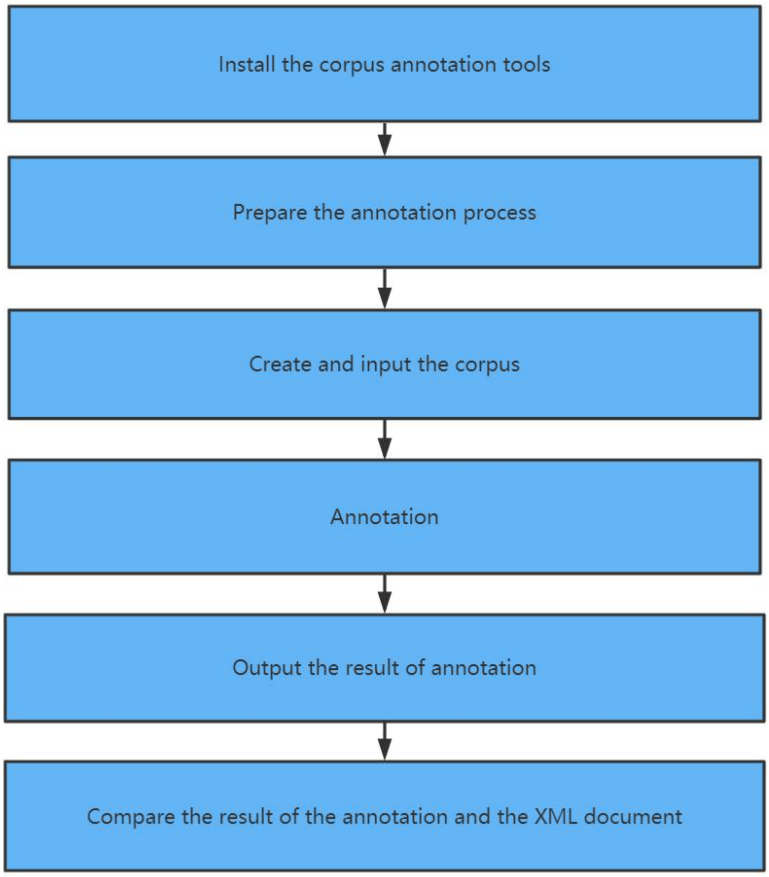

Install the corpus annotation tools
Prepare the annotation process
Create and input the corpus
Annotation
Output the result of annotation
Compare the result of the annotation and the XML document

Figure 2. Steps in analyzing corpus-annotation software.

The remainder of this section explains the steps of annotation and how to investigate each step, although the actual task process of applying different computer tools is diverse due to different requirements.

Figure 2 shows that the first step is to download and install the annotation tools MAE, UAM and DEXTER. The computer tools also have varied hardware requirements, network requirements for download and installation; nevertheless, the three annotation software packages are free access that are conveniently implemented on the Windows operating systems.

Second, the requirements are different in the preparation of annotations. For example, MAE requires a DTD document to create tags before the annotation process; UAM has no technique requirements as MAE, and the process of creating tags is automatically executed within the tool, and DEXTER has an internal jar program that converts plain text into an XML document, so there are no technical requirements for manually creating XML or DTD documents either.

Third, we create a corpus, and input the corpus into the annotation tools. Typically, only one corpus is necessary to test the three computer tools at a time. However, it is not sufficient to generate the corpus only once in the final stage of exporting XML documents because different computer tools differ in steps and storage space. Thus, we create a corpus sample and save them in the packages in each software.

The next step is to input the corpus. The corpus input method varies in each tool. Some tools can only input one plain text, while others can input dozens of texts. When the text input process is executed, this will be the testing point. Scholars are then required to manually annotate the plain text.

And then, we annotate by using these three tools and saving the generate annotation results. This is the essential step since scholars can analysis the procedure by manually annotating it. For example, the surface of the computer tool, surface features of the annotated results, manual annotation ease, and many other elements will be examined and documented for future comparison. The results of the annotation will be produced as XML documents. And then the characteristics of the generated XML document will be checked in this step.

Finally, the computer tools are compared through testing and documenting the procedures. The annotation findings and XML documents are the most important aspects of the comparison. These three corpus computer tools are used to generate an XML document for subsequent activity. The primary features of the test throughout the annotation process are the availability, interactivity, and functionality of the computer tools for linguistic annotators. The following points are intended to present the features of these three tools during the task:

A. Is the software easy to install?

B. Does the software have a single interface or multi-interface?

C. Does the software include guidelines?

D. How many steps are required for annotation?

E. Is there a shortcut key for annotation?

F. Can the corpus be loaded from a single text file or or a folder of text files?

G. Can the software allow the annotation of multiple layers?

H. Can the software add comments for annotations?

I. Can the software highlight annotations?

J. How many colors does the software have for highlighting annotations?

K. Are the annotations of two layers separated in different XML documents?

L. Are the annotations and the original text separated into two XML documents?

M. Is it possible to perform secondary annotation?

Another important aspect of this task is the level of linguistic annotation. In this task, the morphosyntactic level of linguistic annotation is chosen to test and compare the functionality of each computer tool. In particular, we designed two layers to identify basic word attributes at this linguistic level: POS and VERBTENSES. In the layer of VERBTENSES, there are four tags to show the verb tenses: simple present, present continuous, present perfect, and simple past. In the POS layer, there are three tags to show the part of speech: verb, noun, and preposition.

In conclusion, there are two tasks in total. To test the availability, interactivity, functionality and professionalism, the first task is aimed at testing the software throughout the annotation process. The second task is aimed at testing the software by analyzing the structure of the generated XML file and the complexity of converting the generated XML files to HTML files and then presenting the HTML file on the browser.

3.3 Task #1

In Task #1, we designed a corpus sample and performed corpus annotation using three computer tools. The goal of Task #1 is to test and compare MAE, UAM and DEXTER by analyzing the annotation steps . In the end, Table 1 shows the comparison results. As for English level, we chose A1-level English and collected ten plain texts from the website[13] of A1-level English, which contains several listening lessons. To find and save the original text, the researcher should enter into each lesson, and the plain text of the lesson shows up under

[13] The website of of A1-level English. URL: https://learnenglish. britishcouncil.org

“Transcript”. In Task #1, we collected and saved ten plain texts, as the content of the A1-level English corpus.

3.3.1 MAE

This subsection proposes Task #1 for linguistic annotation relying upon MAE. MAE is a simplified computer tool for linguistic annotation which has only one surface. The annotation steps in MAE are tested and recorded for further study. In general, MAE requires six steps to execute the whole annotation procedure:

1. Download and install MAE
2. Prepare a DTD document
3. Load the DTD document
4. Create a linguistic corpus
5. Load the text
6. Annotate
7. Output the XML document

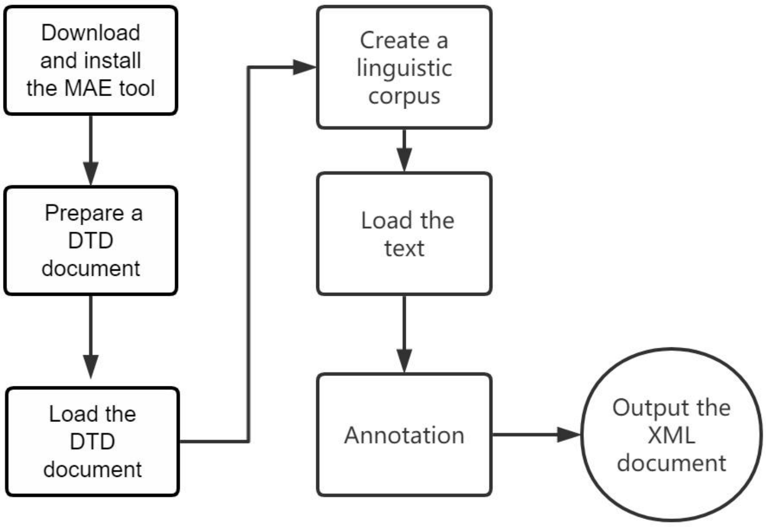


Figure 3. Annotation procedures of MAE.

To test MAE and record each step, we divide the MAE annotation task into three parts.

Part one: Annotate one plain text on one level of annotation.
Part two: Annotate one plain text on two levels of annotation.
Part three: Annotate ten plain texts on two levels of annotation.

Following the general steps in Figure 3, there are six steps recorded in part one. First, MAE should be downloaded and installed in one folder. MAE is free-access computer tool, which can be downloaded from the github platform[14]. The jar document can be operated conveniently for annotation, so there are no more steps for installation. Additionally, there is a free user guide manual that explains how to use the software in detail.

Second, a DTD document should be prepared for annotation after installation. Below is a sample DTD document for the task:

```
<!ENTITY name "ANNOTATIONDTD">
<!ELEMENT VERBTENSES ( #PCDATA ) >
<!ATTLIST VERBTENSES start #IMPLIED >
<!ATTLIST VERBTENSES type (simple present | present continuous | present perfect | simple past) #IMPLIED "other" >
<!ATTLIST VERBTENSES comment CDATA "default value" >
```

The ENTITY name is “ANNOTATIONDDTD”. To start the initial test on a syntactic level, there is one element of DTD documents named “VERBTENSES”. The data type of the element should only be text, but no other form of value, so it shows “#PCDATA” with brackets. The grammar tags are “simple present”, “present continuous”, “present perfect”, and “simple past”, adding in the attribute of the “VERBTENSES” element.

Third, after the full installation and preparation of the DTD document, MAE requires to “load a file”. In the next step, after loading the DTD document and TXT document, MAE will automatically generate one tag named grammar. The tag has different types for annotation as it is settled in the DTD documents. Following preparation, the scholar can now start the annotation process:

[14] Github is mainly used for the creation of computer program source code. The MAE tool can be downloaded in https://github.com/keighrim/mae-annotation.

First, select the text on the interface, and annotate the plain text. Choose “VERBTENSES” if it is annotated in grammar level.

Second, the marked annotation results will be displayed at the bottom of the window. There are five types of annotation: “simple present”, “present continuous”, “present perfect”, and “simple past”. Choose one type of annotation so it is matched and annotated in this type.

Finally, after the original texts are marked up and annotated, choose “save file as XML” to output the results, and then the result is generated as an XML file and saved. When outputting the final documents, the documents are saved in the current corpus file directory. The other two computer tools can choose the directory for final storage, but the final documents will disappear if we operate MAE. In the methodology, we created a corpus directory before the annotation, so we could use it to test and compare the three computer tools. However, when we used MAE, the final documents were saved in the original corpus directory. That is why there should be a new step to create a corpus only for annotation with MAE.

To test the computer tool in different linguistic annotation levels, after the first XML documents are generated, we used another DTD document to test the computer tool. As it was already installed, in part two we started from the second step. A new DTD document was created for annotation in two linguistic annotation levels. Below is a sample DTD document for the task:

```
<!ENTITY name "ANNOTATIONT2">
<!ELEMENT VERBTENSES ( #PCDATA ) >
<!ATTLIST VERBTENSES start #IMPLIED >
<!ATTLIST VERBTENSES type ( simple present | present continuous | present perfect | simple
past ) #IMPLIED "other" >
```

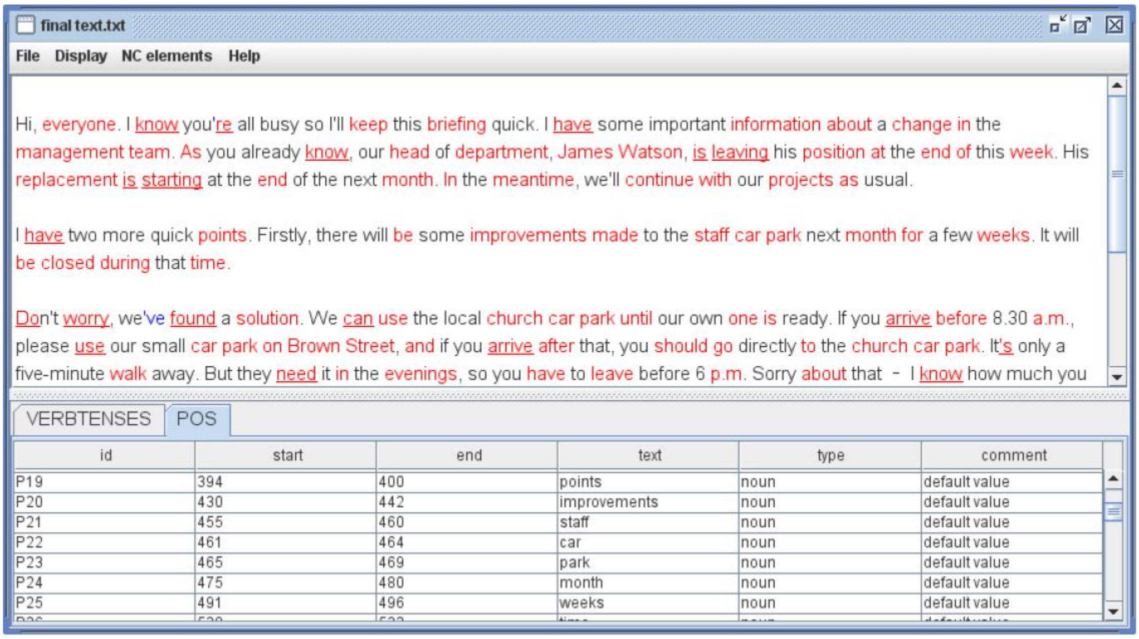
```
<!ATTLIST VERBTENSES comment CDATA "default value" >


<!ELEMENT POS ( #PCDATA ) >
<!ATTLIST POS start #IMPLIED >
<!ATTLIST POS type ( verb | noun | preposition ) #IMPLIED "other" >
<!ATTLIST POS comment CDATA "default value" >
```

The ENTITY name is “ANNOTATIONT2”. There are two elements of DTD documents to test the software at two linguistic levels: “VERBTENSES” and “POS”. The first part of this DTD document is the same as the previous DTD document, but we produce three new elements for POS layer, “verb”, “noun”, and “preposition”.

In the next step, after loading the DTD document and the prepared TXT documents in part one, MAE will automatically generate two tags: grammar and function. Each tag has different types for annotation. When preparation has been completed, the scholar can start the annotation process as below:

First, select the text in the interface, and mark the different language level tags. Choose “VERBTENSES” if it is annotated at the grammar level, or choose “POS” if annotated at the pragmatic level.

Second, the marked annotation results are displayed at the bottom of the window. In POS layer, there are three types of annotations: “noun”, “verb”, and “preposition”. In the grammar layer, there are four types of annotations: “simple present”, “present continuous”, “present perfect”, and “simple past”. Choose one type of annotation, so it is matched and annotated in this type.



Hi, everyone. I know you're all busy so I'll keep this briefing quick. I have some important information about a change in the management team. As you already know, our head of department, James Watson, is leaving his position at the end of this week. His replacement is starting at the end of the next month. In the meantime, we'll continue with our projects as usual.

I have two more quick points. Firstly, there will be some improvements made to the staff car park next month for a few weeks. It will be closed during that time.

Don't worry, we've found a solution. We can use the local church car park until our own one is ready. If you arrive before 8.30 a.m., please use our small car park on Brown Street, and if you arrive after that, you should go directly to the church car park. It's only a five-minute walk away. But they need it in the evenings, so you have to leave before 6 p.m. Sorry about that - I know how much you

id	start	end	text	type	comment
P19	394	400	points	noun	default value
P20	430	442	improvements	noun	default value
P21	455	460	staff	noun	default value
P22	461	464	car	noun	default value
P23	465	469	park	noun	default value
P24	475	480	month	noun	default value
P25	491	496	weeks	noun	default value

Figure 4. The MAE interface.

Figure 4 shows the interface of the annotation step in MAE. There are three main parts to the annotation process: text, tags, and annotation records. The original text is in the center of the whole interface, occupying most of the interface, and the annotated phrases are in-line and highlighted by a different color. The VERBTENSES and POS layers are generated automatically by the DTD document and are located in the middle of the interface. Finally, the annotated records are at the bottom of the interface.

Therefore, there are three parts: the original text, the tags, and the annotation records. The structure of each part has its own characteristics. The whole original texts are shown in the center of MAE; the two tags are below the original texts. The annotated records are at the bottom of MAE, where you can find six attributes for each grammatical tag, i.e. “d”, “start”, “end”, “text”, “type”, and “comment”. When we selected the original text and selected the tags, MAE automatically generated the annotation records. The function of “id” is to differentiate these annotation records, the function of “start” and “end” is to record the position of the annotated text, and the “text” shows the value corresponding to the grammatical tag. The option of “type” shows the different tag types; The function of “comment” is to manually add any comments by the user. Below is a sample of the XML documents, where the complete document can be found in Appendix xxxx.

```
<?xml version="1.0" encoding="UTF-8" ?>
<ANNOTATIONDTD>
<TEXT><![CDATA[
Hi, everyone. I know...
...
OK. That's it? Are there any questions?
]]></TEXT>
<TAGS>
<VERBTENSES id="V1" start="203" end="214" text=" is leaving" type="present continuous" comment="default value" />
<VERBTENSES id="V2" start="269" end="281" text="is starting " type="present continuous" comment="default value" />
<VERBTENSES id="V3" start="550" end="559" text="'ve found" type="present perfect" comment="default value" />
<VERBTENSES id="V4" start="1023" end="1042" text="has now introduced " type="present perfect" comment="default value" />
<VERBTENSES id="V5" start="978" end="984" text="wanted" type="simple past" comment="default value" />
<POS id="P2" start="94" end="105" text="information" type="noun" comment="default value" />
<POS id="P4" start="114" end="120" text="change" type="noun" comment="default value" />
<POS id="P98" start="541" end="546" text="worry" type="verb" comment="default value" />
<POS id="P99" start="575" end="578" text="can" type="verb" comment="default value" />
<POS id="P100" start="579" end="582" text="use" type="verb" comment="default value" />
```

```
<POS id="P101" start="627" end="629" text="is" type="verb"
...
</TAGS>
</ANNOTATIONDTD>
```

The generated XML document is well structured. There are mainly two parts: original text and tags. The original plain text is created in the element of <TEXT>. The annotated text is created in the element of <TAGS>, in which each child element shows the final annotation. The value of the element complies with the annotation records at the bottom of MAE. For example, we do not add comments for the annotation in this task, so it shows “default value”.

In part three, we decided to annotate ten plain texts on two layers of morphosyntactic annotation. However, as MAE cannot load the whole directory, but only the single text documents, part three is executed following the same steps as part two. Finally, ten original plain texts are manually annotated, and MAE separately generates ten different XML documents for each plain text.

3.3.2 UAM

In this subsection, UAM is applied to test linguistic annotation. The steps are recorded to demonstrate the availability, interactivity, functionality, and professionalism of the corpus annotation.

When applying UAM, the user is required to prepare different DTD documents to set tags on the two layers of POS and verb tenses, so there are two tasks designed to differentiate the single level annotation and multi-level linguistic annotation. However, different DTD documents set two levels of linguistic tags, so there is only one task designed to test it. In general, there are six steps to perform the annotation under two linguistic levels of corpus annotation.

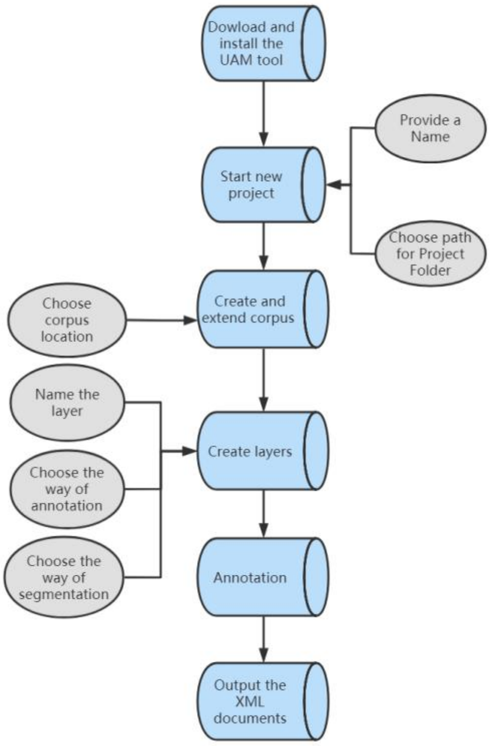


Figure 5. UAM annotation procedures.

Figure 5 shows the six steps of the annotation procedure. First, the UAM Corpus annotation tool is free-access, so it is free to download from website. Installation is relatively easy as there are no more steps after the file is unzipped, with the EXE file running the tool.

Second, the UAM Corpus requires a new project for corpus annotation to be created. The start-up screen of UAM requires creating a project, opening a project, or importing a project. In this paper, we created a new annotation project. In general, there are mainly three steps to start a new project.

1.Start a new project

2.Name the new project

3.Choose a project folder for the project

After the new project is created, there are four folders in the annotation project: Annotations, Corpus, Results, and Schemes. UAM is user-friendly for inexperienced human annotators as there is a guideline for each step. Moreover, there is the option “Back” to modify the previous step.

Thirdly, UAM requires creating and extending the corpus. In this step, there are also some guidelines for corpus creation. Following the steps of the UAM, the most significant part is to choose the corpus location to add the original text. There are two steps to add the corpus to this computer tool:

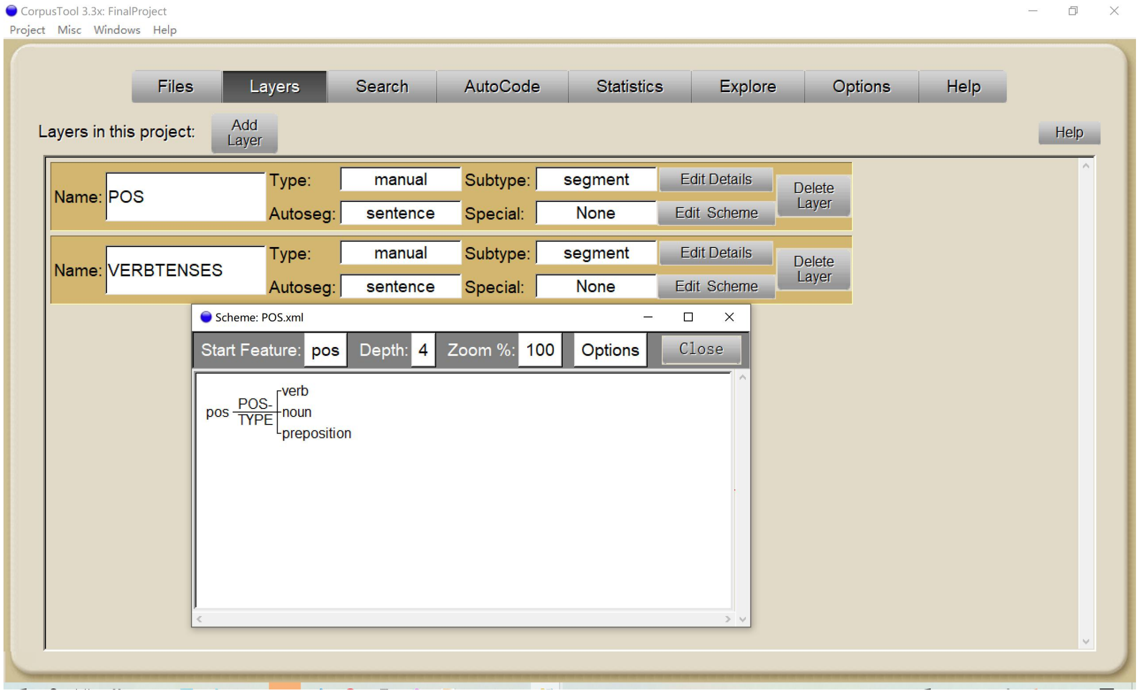
1.Extend corpus: choose the location of a single text file, or a folder of text files, or paste it directly from the clipboard.
2.Incorporate the file.

In the first step, there are three ways of adding the original documents into the software. After completing this procedure and testing it several times, we found two features of UAM. First, UAM can add several corpus folders. Second, it can add several times. These features are important as they simplify the procedure of corpus annotation when there are several documents to be annotated.

In the second step, the user can choose to incorporate the text files one by one, but can also press “Incorporate all” to load all the text files as a batch. There are also options for previously added documents, for example, incorporating files and deleting files.

After the texts are loaded, UAM requires layers to be created. There are several options for a customized annotation design. For example, the user can choose manual annotation or automatic annotation. Moreover, the user can segment text by paragraphs, sentences, clauses, etc.

In this task, we chose manual annotation and sentence segmentation. In the UAM Corpus tool, the two “function” and “grammar” tags are called layers, while the different types are structured by the scheme's design. After the function and grammar layers have been created, the UAM requires the scheme to be edited in order to show the different types of linguistic levels. This step is shown in Figure 6.



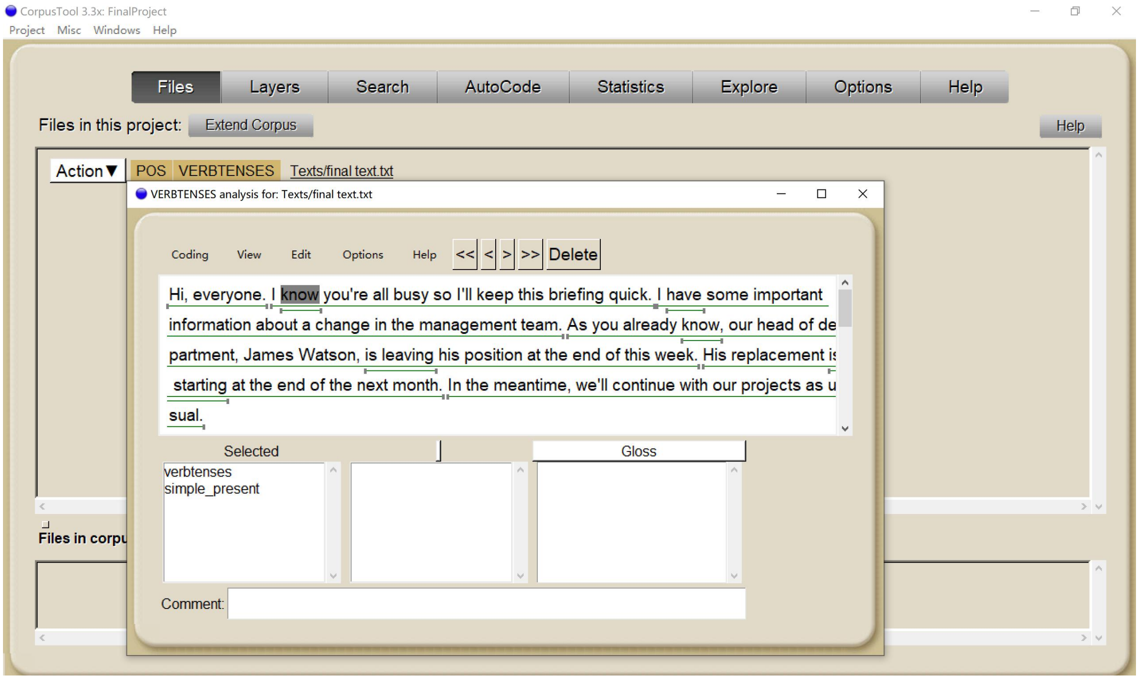
Name: POS Type: manual Subtype: segment Autoseg: sentence Special: None

Name: VERBTENSES Type: manual Subtype: segment Autoseg: sentence Special: None

Start Feature: pos Depth: 4 Zoom %: 100

Figure 6. Design scheme for layers.

After the corpus and layer are created, the next step is annotation. UAM cannot be applied to annotation on two linguistic levels at the same time, so the user needs to annotate the same plain text twice. First, after opening the function layer , and then selecting the original text and the different types at the bottom of the surface, the pragmatic annotation is successfully executed. Secondly, the syntactic annotation repeats the same procedure, but it is executed in the grammar layer .



Hi, everyone. I know you're all busy so I'll keep this briefing quick. I have some important
information about a change in the management team. As you already know, our head of de
partment, James Watson, is leaving his position at the end of this week. His replacement is
starting at the end of the next month. In the meantime, we'll continue with our projects as u
sual.

verbtenses
simple_present

Figure 7. Annotation of UAM Corpus.

Finally, the last step is to generate the annotation results. Unlike MAE, the operation is convenient and efficient, but the button option is on the surface of the annotation.

As the annotation process is executed separately for the POS and VERBTENSES layers, the final annotation results are generated in two different XML documents by UAM. The original text and the annotation results are in the same <segment> element, but the original texts are the content and the annotation results are in brackets as an attribute of an element. For example, below is one part of the XML documents of the POS layer:

I <segment id='102' features='pos;verb' state='active' parent='3'>have</segment>some important <segment id='25' features='pos;noun' state='active' parent='3'> information</segment>

This sample demonstrates that the annotation results are in the element

“<segment id='102' features='pos; verb' state='active' parent='3'>”, the “features” element presents the annotation results in the form of a POS layer and it is annotated as “verb”.

3.3.3 DEXTER

The Dexter computer tool requires transferring TXT documents into XML documents one by one, so the linguistic annotation will be complemented on two levels. This task consisted of testing and annotating one document and then repeating the process for the entire corpus. For linguistic annotation using DEXTER, there are ten steps in general:

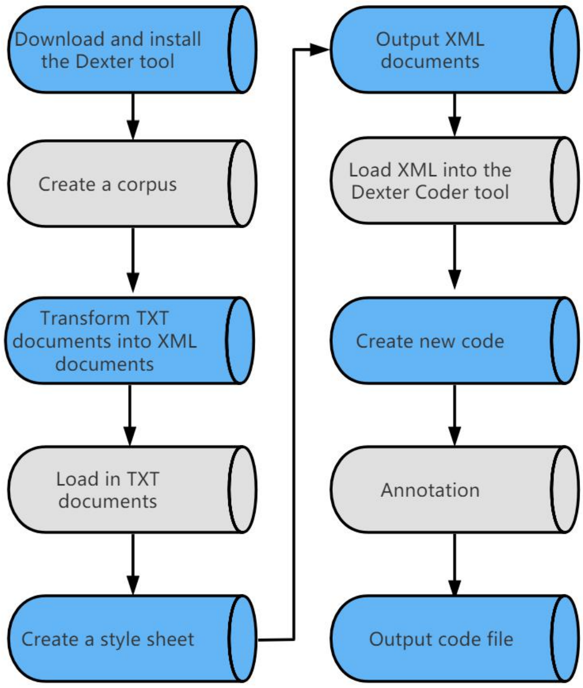


Figure 8. Annotation procedures in the Dexter tool.

First, the Dexter software is free-access, so it can be downloaded from the official website. The primary tools for this task are Dexter Coder and Dexter

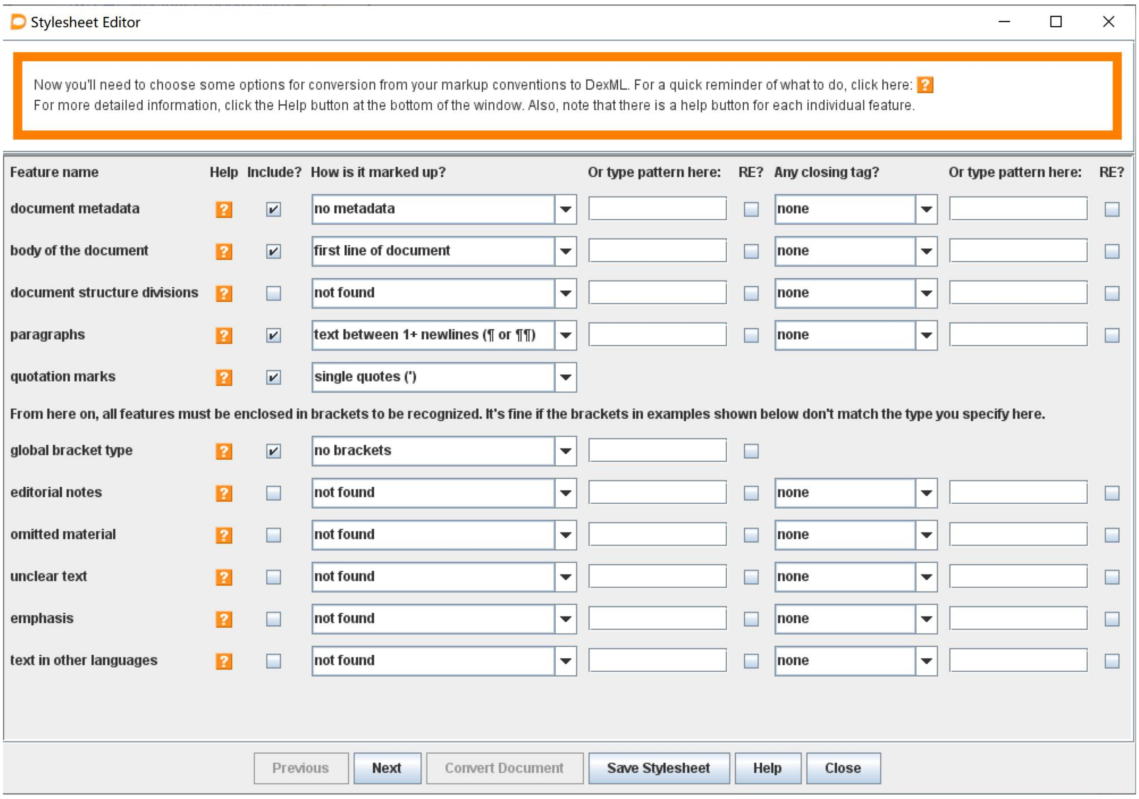
Converter.

Second, to ensure the results are the same across tests, we tested the source texts at the English A1 level and with the same linguistic annotation levels, part-of-speech and verb tenses. Using the same TXT documents from the MAE corpus, we created a new corpus folder and added it to the Dexter folder.

Unlike MAE, no preparation of the DTD file is required before starting annotation other than installing and creating the corpus. Instead, after loading the original text, DEXTER requires the user to create a style sheet for each original text. The user can select different requirements for spoken or written language.

DEXTER can only load a single piece of text, not an entire folder of texts, and then transfer it to XML after applying the stylesheet. In this step, we select the written language and follow the guidelines in the Dexter converter.

Third, after the original plain text is transformed into an XML document, Dexter requires the user to create a style sheet according to special requirements. The style sheet consists of personalized choices for the user to set the form of the original texts. Figure 9 shows a sample of the DEXTER style sheet. On the left, Dexter shows each item of the style sheet, and the introductions of each item are displayed after clicking on the orange question mark. Although the Dexter computer tool has many guidelines inside the software, the inexperienced human annotator may become confused with the requirements of setting the style sheet. On the other hand, style sheets make it more convenient for professional linguistic annotators to choose and set the structure of documents in a personalized method.



Now you'll need to choose some options for conversion from your markup conventions to DexML. For a quick reminder of what to do, click here: ?
For more detailed information, click the Help button at the bottom of the window. Also, note that there is a help button for each individual feature.

Feature name	Help	Include?	How is it marked up?	Or type pattern here:	RE?	Any closing tag?	Or type pattern here:	RE?
document metadata	?	☑	no metadata		☐	none		☐
body of the document	?	☑	first line of document		☐	none		☐
document structure divisions	?	☐	not found		☐	none		☐
paragraphs	?	☑	text between 1+ newlines (¶ or ¶¶)		☐	none		☐
quotation marks	?	☑	single quotes (')					

From here on, all features must be enclosed in brackets to be recognized. It's fine if the brackets in examples shown below don't match the type you specify here.

global bracket type	?	☑	no brackets		☐			
editorial notes	?	☐	not found		☐	none		☐
omitted material	?	☐	not found		☐	none		☐
unclear text	?	☐	not found		☐	none		☐
emphasis	?	☐	not found		☐	none		☐
text in other languages	?	☐	not found		☐	none		☐

Figure 9. The Dexter style sheet.

After the style sheet is created, the XML documents can be generated complying with the different style sheet settings. Below is the sample of the XML documents converted by the Dexter Converter:

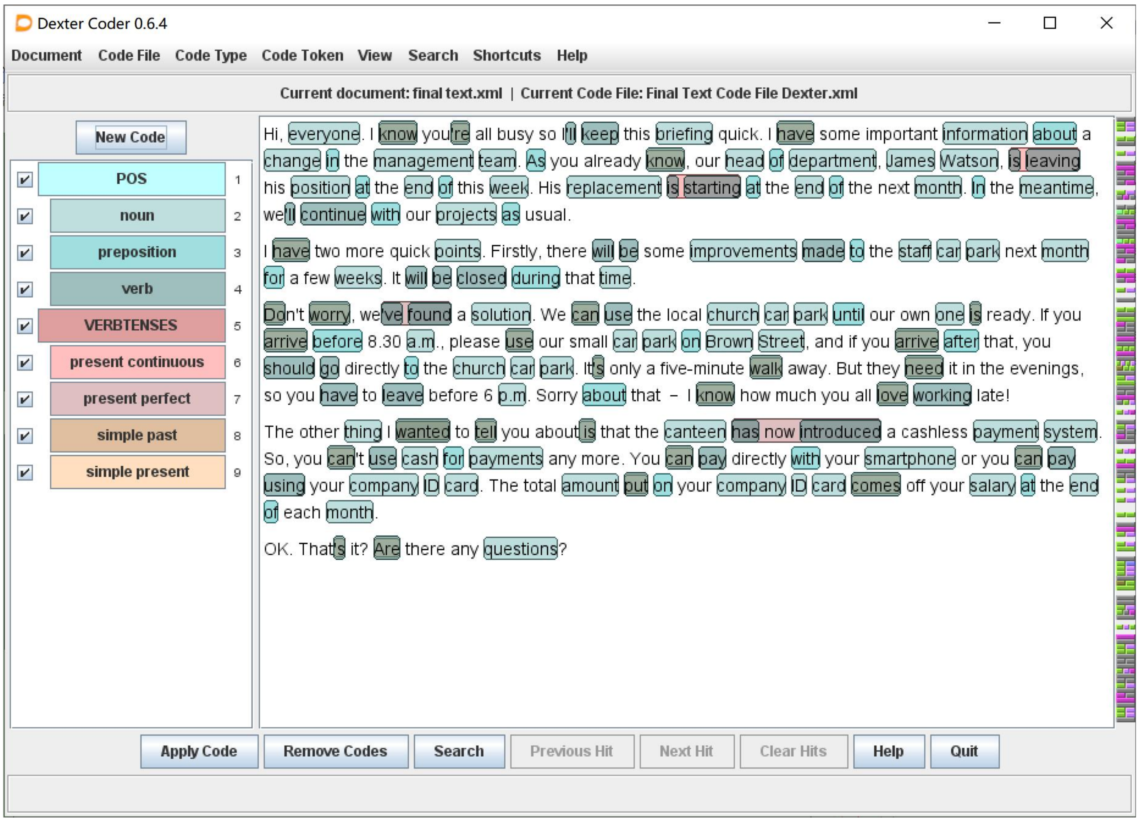
```
<?xml version="1.0" encoding="UTF-8"?>
<TEI.2>
  <teiHeader>
...
  </teiHeader>
  <text>
    <body id="b">
      <p id="b.1">
          <seg id="b.1.1">Hi, everyone. I know you're all busy so I'll keep this briefing quick. I have some important information about a change in the management team. As you already know, our head of department, James Watson, is leaving his position at the end of this week. His
```

```
replacement is starting at the end of the next month. In the meantime, we'll continue with our projects as usual. </seg>
          </p>
     ...
        </body>
      </text>
     </TEI.2>
```

The generated XML document has a simplified structure. Each paragraph is separated by the <seg> element and has its own “id” number for identification.

With the generated documents, the Dexter Converter tool performs the steps for creating different tags on different linguistic levels and executing annotations. The Open Document option on the interface helps load the XML documents into the Dexter Coder tool. Unlike the other two computer tools, the original texts cannot be loaded into the Dexter Coder tool for annotation without conversion by the Dexter Converter tool.

The next step is to create new code. There are two tags, i.e., “POS” and “VERBTENSES”, used to execute linguistic annotations in two layers. These tags are called codes in the Dexter Coder and named layers in UAM, and they refer to the different linguistic tags for the “POS” and “VERBTENSES” of the annotation.



Hi, everyone. I know you're all busy so I'll keep this briefing quick. I have some important information about a change in the management team. As you already know, our head of department, James Watson, is leaving his position at the end of this week. His replacement is starting at the end of the next month. In the meantime, we'll continue with our projects as usual.

I have two more quick points. Firstly, there will be some improvements made to the staff car park next month for a few weeks. It will be closed during that time.

Don't worry, we've found a solution. We can use the local church car park until our own one is ready. If you arrive before 8.30 a.m., please use our small car park on Brown Street, and if you arrive after that, you should go directly to the church car park. It's only a five-minute walk away. But they need it in the evenings, so you have to leave before 6 p.m. Sorry about that – I know how much you all love working late!

The other thing I wanted to tell you about is that the canteen has now introduced a cashless payment system. So, you can't use cash for payments any more. You can pay directly with your smartphone or you can pay using your company ID card. The total amount put on your company ID card comes off your salary at the end of each month.

OK. That's it? Are there any questions?

Figure 10. Annotation of Dexter tool.

On the left of Figure 10, there are nine codes in total. The POS code has three sub-codes and the VERBTENSES code has four sub-codes. The position and structure are well-performed. In addition, the user can randomly select different colors of the code. For example, in our task, the POS codes were designed in calm blue tones, and VERBTENSES codes were in warm red tones. Different colors make the layers more recognizable, so it improves the user experience and the efficiency of corpus annotation.

Figure 10 shows the interface of the annotation of Dexter. The structure has two parts. On the left, a column of tags shows the different codes, and on the right of the interface, we find the original text of the loaded XML documents.

After creating the codes for two levels of linguistic annotation, the user can start to annotate by selecting the code and the correspondingly presented texts. The annotator can use the shortcut "A" on the keyboard to annotate the text. This makes it convenient for annotation tasks requiring linguistic annotations involving large amounts of manual annotation for different tags.

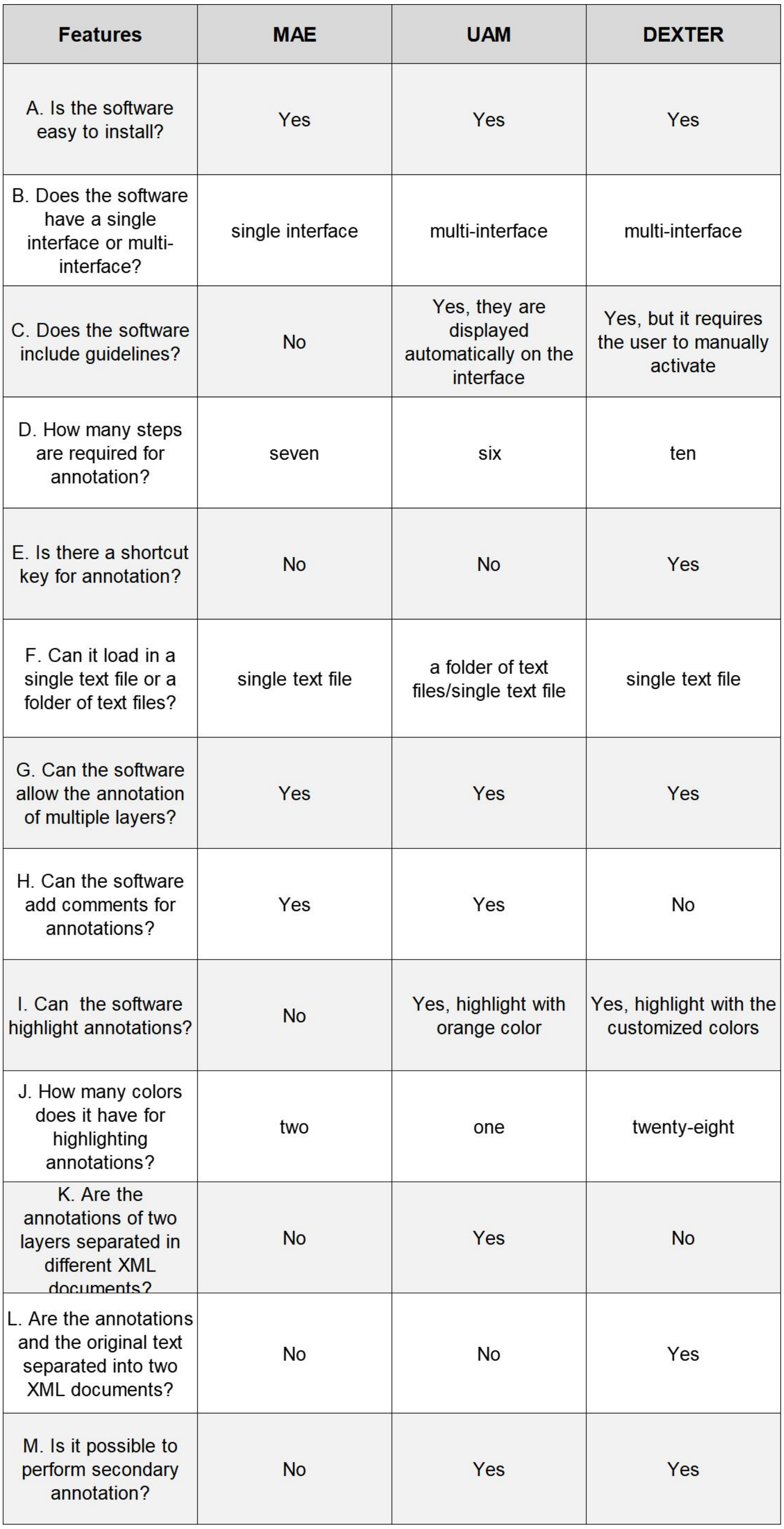
Finally, the last step is to output the annotation results. In Dexter, the final annotated XML document is called “Code File.” We chose the option “Save Code File” on the interface, so the annotated records could be output. In the end, the outcome is also saved as another XML document. The following is part of the example of the annotated XML document of Dexter. The annotated XML document shows the different types of tags in the <type> element, the name attribute shows the name of the type, and the attribute “red,” “green,” and “blue” indicate the color. Also, we discover that the <token> elements show only the data of the position of texts instead of the original texts. In this aspect, the generated XML documents' structure differs from MAE and UAM. The following is a sample of XML document:

```
      <data>
    ...
     <type name="simple past" parent="VERBTENSES" red="128" green="128" blue="255"
visible="true">
          <token>
            <start_id>b.4.1</start_id>
            <start_index>19</start_index>
            <start_string>I wan</start_string>
            <end_id>b.4.1</end_id>
            <end_index>24</end_index>
            <end_string>ted t</end_string>
          </token>
        </type>
    </data>
```

The <data> element presents the data of the position of annotations. In the <token> element, there are six child elements: <start_id>, <start_index>, <start_string>, <end_id>, <end_index>, and <end_string>. The first three elements show the identification number, location in the text and the start position of the annotation text.

3.3.4 Result of Task #1

After accomplishing all the research tasks, we recorded them and made a table to compare these three computer tools in their availability, functionality, and availability.



Features	MAE	UAM	DEXTER
A. Is the software easy to install?	Yes	Yes	Yes
B. Does the software have a single interface or multi-interface?	single interface	multi-interface	multi-interface
C. Does the software include guidelines?	No	Yes, they are displayed automatically on the interface	Yes, but it requires the user to manually activate
D. How many steps are required for annotation?	seven	six	ten
E. Is there a shortcut key for annotation?	No	No	Yes
F. Can it load in a single text file or a folder of text files?	single text file	a folder of text files/single text file	single text file
G. Can the software allow the annotation of multiple layers?	Yes	Yes	Yes
H. Can the software add comments for annotations?	Yes	Yes	No
I. Can the software highlight annotations?	No	Yes, highlight with orange color	Yes, highlight with the customized colors
J. How many colors does it have for highlighting annotations?	two	one	twenty-eight
K. Are the annotations of two layers separated in different XML documents?	No	Yes	No
L. Are the annotations and the original text separated into two XML documents?	No	No	Yes
M. Is it possible to perform secondary annotation?	No	Yes	Yes

Table 1. Comparison of computer tools by Task #1.

Concerning feature A, this table compares the convenience of downloading and installing the three corpus tools; MAE, UAM Corpus, and Dexter are all open-source software and installation packages found on the official website. All three computer tools proved to be easy to download and install.

Concerning feature B, this table compares whether the three corpus tools have a simplified interface. MAE has only one interactive interface, and all executions operate on the same page. In contrast to MAE, the UAM Corpus software has multiple interactive interfaces. There are new interfaces for the guideline and the annotation. In particular, the UAM corpus can annotate several texts and different linguistic levels of annotation simultaneously during the annotation steps, so there could be several annotation interfaces simultaneously. Dexter has two computer tools, each of which has several interfaces for different uses.

As for feature C, this table compares whether the three corpus tools have guidelines inside the software. As for MAE, there is a manual script for the new annotator, but there is no guideline inside the software for introducing the functions of MAE. The UAM Corpus has guidelines and instructions inside the software, and it provides several options for the user to choose in different steps according to their specifically designed annotation tasks. In addition, all the guidelines of the UAM corpus tool are presented automatically at each step, which the user follows to complete the entire project. These guidelines are also available in the Dexter software. However, unlike UAM, Dexter requires the user to click on them manually rather than presenting them actively and giving the user a choice. Dexter and UAM have more advantages in terms of guidelines; both tools offer functions that can be selected and set by the user. For example, the interface shows the same structure as the original text with the loaded plain text; there are no other functionalities to delimit the segments of the text. However, UAM software has the functionalities of choosing how the text is segmented, but it is selected and set by the user. The original text can be segmented into paragraphs, sentences, clauses, tokens, etc. Dexter does not have options for segmenting the original text, but when setting the style sheet, there are personalized options for specific academic requirements.

Feature D describes the annotation steps. The process of annotating with MAE consists of seven steps from start to end, while UAM requires six steps and Dexter requires ten.

Regarding feature E, MAE and UAM have no shortcut key for annotation, but Dexter has shortcut "A" to apply code to XML documents. Therefore, it is more convenient to apply Dexter if many annotation tasks require manual annotation.

Regarding features F and G, MAE and DEXTER can only load in one single file and only once, whereas the UAM Corpus tool provides the functions of loading single files and a folder. Moreover, the UAM software can load several times. In this way, if the original texts are separated into several documents, it is more convenient to use UAM as the computer tool for annotation.

About feature H, MAE can test two linguistic levels simultaneously by designing the DTD documents. If two linguistic levels of annotation are required, another element can be added in the DTD documents, and the new tag will be generated automatically in the interface after loading the DTD documents. UAM software also has the functionality of annotating on two linguistic levels, but it requires manually generating a “scheme” and setting the structure of each linguistic level. The user must manually set the different types of each tag in the “scheme system.” Like UAM, Dexter requires the user to create the tags, but in UAM, it is named “layer,” and in Dexter, it is named “code.”

About feature I, the MAE and UAM software both have functions for recording comments for annotation, but DEXTER has no comments function.

Regarding the characteristic of tags in feature J, MAE has no highlight on the two function and grammar tags, UAM highlights the tags in orange, and Dexter has 28 kinds of color that can be customized. As for the annotated texts, there are two colors in MAE to highlight different annotations, UAM has only one color to underline the original text and the annotated text, and Dexter has twenty-eight colors according to the designed code. This is an aspect where DEXTER has a clear advantage in that it emphasizes the annotated texts and the correlation between content and tags, allowing users to identify the grade of annotated texts directly by color. On the one hand, DEXTER allows for a closer correlation between the annotation tags and the text being annotated. On the other hand, optimizing the user interface makes the computer tool more attractive and allows the annotator to become more involved in the annotation.

As for feature K, two linguistic levels of annotation results are displayed together in the same generated XSL file by MAE. When applying DEXTER for annotation, the annotation results for the two different levels are also in the same XML file, but the final XML file output from MAE shows the original text and the annotation text, whereas DEXTER only shows the annotation data, not the actual annotation text. On the other hand, UAM separates two levels of linguistic annotation, so the final XML file is separated.

Regarding feature L, we find that Dexter separates the annotation records and the original texts into two different XML documents, while UAM and MAE contain the annotated text and original text in the same XML documents.

Finally, feature M in the table, describes that MAE cannot open the annotated documents again after it is executed. Dexter can review and annotate the annotated text by simultaneously loading the converted XML documents and the annotated code file. Nevertheless, the user must find the path where the XML files converted using Dexter and the code file of the annotated texts are stored; otherwise, the software will report an error. However, the UAM Corpus can open the last project directly on the initial interface by choosing the latest project options. In this regard, UAM is a better choice for tasks requiring secondary annotation when a lot of annotation work is involved.

This section provides a way of critical thinking and approaching the exploration and research of corpus tools. After the comparison, we conclude that an inexperienced human annotator would prefer the MAE software, which only requires a few steps compared with the other two tools, a complete user manual, and a simplified interactive interface. Professional linguists involved in a long-term and complex research task may choose the UAM or DEXTER as they provide customized options and secondary annotation.

3.4 Task #2

Task #2 creates XSL files to convert generated XML files into HTML files, displaying annotations marked with different colors on the browser. The goal of the second task is to test UAM and DEXTER by analyzing the structure of the generated XML and the complexity of converting the generated XML to HTML. In the end, Table 1 shows the comparison results.There are four steps to complete this task:

1.Use the XSL document to clean the data of the original XML document

2.Mark the annotation in the XML document

3.Convert the XML document into HTML document

4.Present the HTML document in the browser

However, we did not design Task #2 for MAE software for three reasons according to the generated XML file by MAE: First, the original text is a whole text and does not separate into paragraphs or sentences. Second, the structure does not contain the elements of an HTML document. There are no <title>, <body> or <paragraph> elements. Finally, there is only “start,” “end,” and “text” data information about the position of annotation. The lacking data information makes it more complicated to complete Task #2. For example, DEXTER has six elements to present the detailed information on the position of the annotation in the text, and the <start_string> element and <end_string> element help to locate the annotation in the text. Consequently, Task #2 does not test MAE.

3.4.1 UAM

To test UAM, we analyzed the structure of the generated XML and then created an XSL document to clean the data and convert the XML document into an HTML document.

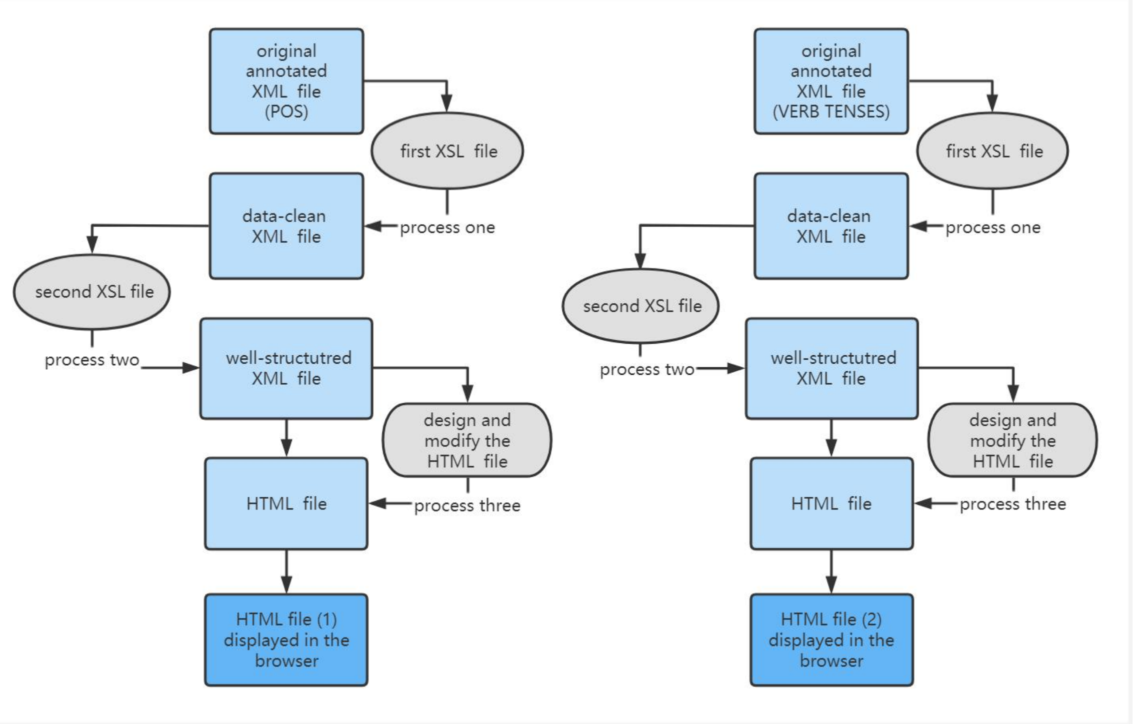

original annotated XML file (POS)
first XSL file
data-clean XML file
process one
second XSL file
process two
well-structutred XML file
design and modify the HTML file
HTML file
process three
HTML file (1) displayed in the browser
original annotated XML file (VERB TENSES)
first XSL file
data-clean XML file
process one
second XSL file
process two
well-structutred XML file
design and modify the HTML file
HTML file
process three
HTML file (2) displayed in the browser

Figure 11. Procedures of Task #2 for UAM.

The structure of the XML document should be modified. On the one hand, the generated XML has redundant data, for example, the <lang> element, which refers to the annotation language but is unnecessary for the task. These elements should be deleted by using XSL technology. On the other hand, as the final goal is to transform XML files into HTML files, the structure of XML documents should conform with the structure of HTML. The XML file already has the element <body> and <header>, but there is no <p> element. Therefore, the necessary elements should be added to the XML files by using XSL technology.

Figure 11 shows the detailed procedures. For preparation, there is one original XML file annotated by UAM and two XSL files that we created to modify the XML documents.

This task executes twice as there are two generated XML documents with two layers presenting part of speech annotation and verb tenses annotation. Therefore, the procedures for the two annotated XML documents are the same, but the documents and results are different. For this reason, the following paragraphs only describe the procedure on the left of Figure 11. In general, there are three main steps to execute the task.

process one: the first XSL file is used to clean the data and mark out the annotated text

process two: the second XSL file is used to set colors for each tag

process three: the XML file is designed and modified and is then converted to an HTML file

In process one, the XSL file modifies the original XML document. Appendix 1 shows the code of the first XSL file. From line 1 to line 3, the code defines the namespace of this document. Every HTML file requires the declaration and namespace. From line 5 to line 21, the code modifies the XML document structure by deleting the unnecessary elements and adding the required elements. When the XSL code matches some nodes and does nothing, it deletes the unnecessary elements, and when the XSL code matches the nodes and adds an element in a bracket, it will add a required element to the XML file. For example:

A.

```
<xsl:template match="lang" />
```

The above code deletes the <lang> element by matching the node of the <lang> element and does nothing about it.

From line 23 to line 43, the code finds the node and marks out the annotation to prepare the next step. The XSL code applies the <xsl: choose> element to match the node where the value of "features" is "pos" and then sets the value for each tag of "pos": verb, noun, and preposition. For example:

B.

```
<xsl:when test="@features='pos;verb'">verb</xsl:when>
```

The above code matches the node where the value of features is "pos" and "verb," then marks out this node and sets the value "verb" to differentiate it from other tags.

After the XML document is processed with the first XSL file, the result of the generated new XML file is saved as the XML file for the next step. The generated XML file shows in Appendix 2.

In process two, the second XSL file is used to modify the XML file into well-structured HTML format and then changes the colors for each of the POS tags. Appendix 3 shows the code of the second XSL file. From line 5 to line 21, the XSL code converts the XML structure into HTML format structure by changing the name of the elements. The following shows part of the code:

C.

```
<xsl:template match="//paragraph">
<p><xsl:apply-templates select="node()" /></p>
</xsl:template>
```

As for this part of the code, the element <paragraph> changes the name and converts it into element <p> by using the <xsl:template> element.

From line 23 to line 33, the code matches the special node and sets the color for the node by using the <font> tags of HTML. The following shows part of the code:

D.

```
<xsl:template match="//segment[@pos='verb']">
    <font color="olive"><xsl:apply-templates select="node()" /></font>
</xsl:template>
```

The generated XML file of process two is in Appendix 4. The code matches the <segment> element where the value of "pos" is "verb" and then sets the olive color for the whole node by selecting a node and using the <font> tags.

The next step is process three in Firgue 11. The well-structured XML generated by process two is used to design and modify an HTML file. As process one and process two have already converted the XML file into the format of an HTML file, process three requires adding a title for the document, which can be present on the browser.

E.

```
<h1>Part of Speech</h1>
```

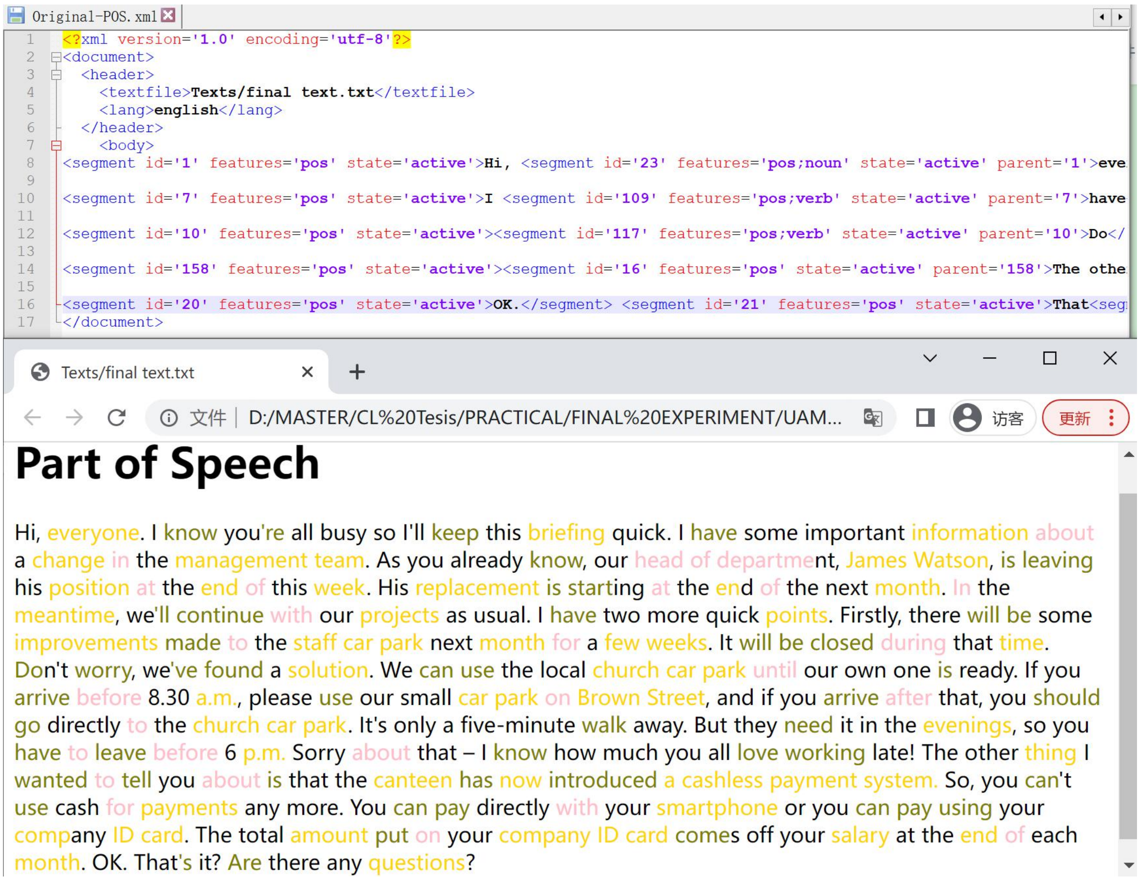
Sample E applies the <h> HTML tag to add the title value. This title will be present on the browser.

Finally, the declaration of the HTML is added (sample D) and is saved as an HTML file, then it is opened by the browser. The annotation results are present in the browser showing its annotated texts with the different colors that are set in process two.

D.

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

Sample D shows the declaration of HTML. As a result, the following Figure 12 shows the results of Task #2 for UAM. The XML document is converted into the HTML file, which can be present on the browser.



```
<?xml version='1.0' encoding='utf-8'?>
<document>
  <header>
    <textfile>Texts/final text.txt</textfile>
    <lang>english</lang>
  </header>
    <body>
<segment id='1' features='pos' state='active'>Hi, <segment id='23' features='pos;noun' state='active' parent='1'>eve

<segment id='7' features='pos' state='active'>I <segment id='109' features='pos;verb' state='active' parent='7'>have

<segment id='10' features='pos' state='active'><segment id='117' features='pos;verb' state='active' parent='10'>Do</

<segment id='158' features='pos' state='active'><segment id='16' features='pos' state='active' parent='158'>The othe

<segment id='20' features='pos' state='active'>OK.</segment> <segment id='21' features='pos' state='active'>That<seg
</document>
```

Part of Speech

Hi, everyone. I know you're all busy so I'll keep this briefing quick. I have some important information about a change in the management team. As you already know, our head of department, James Watson, is leaving his position at the end of this week. His replacement is starting at the end of the next month. In the meantime, we'll continue with our projects as usual. I have two more quick points. Firstly, there will be some improvements made to the staff car park next month for a few weeks. It will be closed during that time. Don't worry, we've found a solution. We can use the local church car park until our own one is ready. If you arrive before 8.30 a.m., please use our small car park on Brown Street, and if you arrive after that, you should go directly to the church car park. It's only a five-minute walk away. But they need it in the evenings, so you have to leave before 6 p.m. Sorry about that – I know how much you all love working late! The other thing I wanted to tell you about is that the canteen has now introduced a cashless payment system. So, you can't use cash for payments any more. You can pay directly with your smartphone or you can pay using your company ID card. The total amount put on your company ID card comes off your salary at the end of each month. OK. That's it? Are there any questions?

Figure 12. Result of Task #2 with UAM.

The top of Figure 12 shows the original XML generated by UAM, and at the bottom, it presents the annotation results with different colors in the browser.

3.4.2 DEXTER

Following the steps of Task #2, the general procedures of testing DEXTER are the same. First, the structure of the generated XML is analyzed. Second, design Task #2 by applying XSL technology to modify the XML files and converting the modified XML file into an HTML file.

The structure of the original XML files generated by DEXTER is simplified. In Task #1, DEXTER generates two XML documents: the original XML file, the original code file, and the original converted XML file. The code file contains all the annotation data of the two layers, and the text file contains the content of the text. In order to set colors for annotations, it is essential to generate an XML file with the annotation text instead of the annotation data, which are the numbers and strings for the end and start positions. Thus, an essential step in this task is to extract the annotation information from the code file and then put the extracted annotation data in the original XML file.

As for the main steps of completing Task #2, we designed it to separate Task #2 into two layers for two reasons. On the one hand, the XSL codes could be more verbose in processing two layers of annotation together. On the other hand, since DEXTER annotates the exact text repeatedly for two layers simultaneously and Task #2 is designed to apply colors to each annotation, separately displaying the HTML file for each layer is more visually precise.

The following Figure 13 presents the procedures of DEXTER.

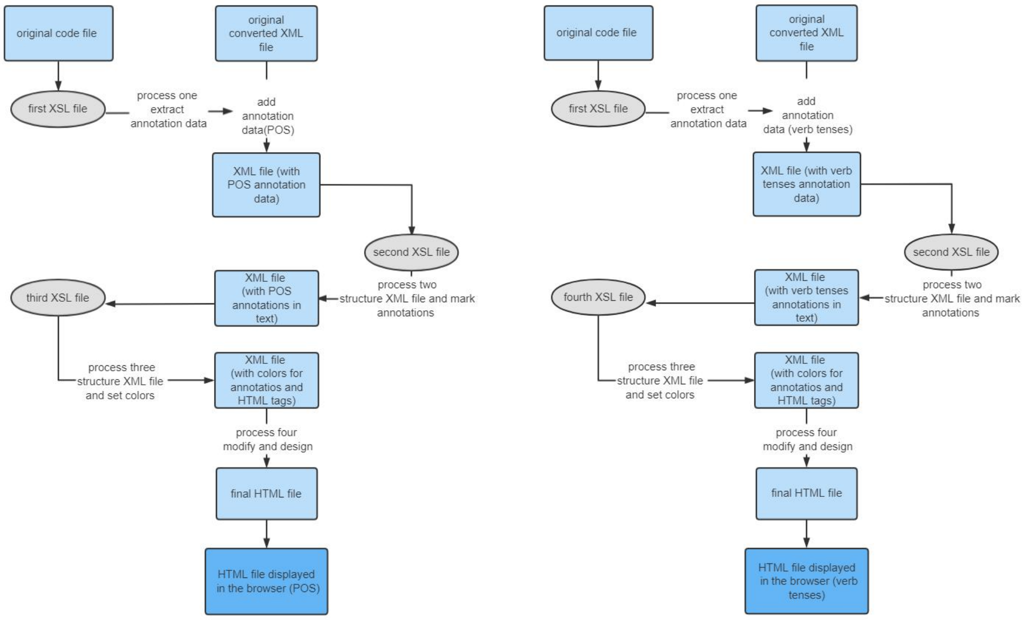

original code file
original converted XML file
first XSL file
process one extract annotation data
add annotation data(POS)
XML file (with POS annotation data)
second XSL file
process two structure XML file and mark annotations
XML file (with POS annotations in text)
third XSL file
process three structure XML file and set colors
XML file (with colors for annotatios and HTML tags)
process four modify and design
final HTML file
HTML file displayed in the browser (POS)
original code file
original converted XML file
first XSL file
process one extract annotation data
add annotation data (verb tenses)
XML file (with verb tenses annotation data)
second XSL file
process two structure XML file and mark annotations
XML file (with verb tenses annotations in text)
fourth XSL file
process three structure XML file and set colors
XML file (with colors for annotatios and HTML tags)
process four modify and design
final HTML file
HTML file displayed in the browser (verb tenses)

Figure 13. Procedures of Task #2 for DEXTER.

On the left of Figure 13, it presents the procedures of Task #2 regarding the POS layers. In general, there are four main steps to complete the task.

process one uses the first XSL file to extract data from the original code file and then adds the extracted annotation data into the original converted XML file.

process two uses the second XSL file to structure the XML file and marks the annotations.

process three uses the third XSL file to structure the XML file by the format of HTML and sets colors for each annotation.

process four: designs and modifies the XML file and then converts it to an HTML file.

The main purpose of process one is to extract data from the original code file using XSL technology. Appendix 5 shows the XSL file. This XSL file has one template from line 5 to line 19, which uses <xsl:for-each> function to extract annotation information from the original code file containing all the annotation data. The <data> element contains a <xsl:for-each> element and the <xsl:for-each> element has a <sent> element, which defines four attributes, the attributes name are id, function, start and end. The following code is part of the first XSL file.

a.

```
<sent>
    <xsl:attribute name="id"><xsl:value-of select="start_id"/></xsl:attribute>
    ...
</sent>
```

The <sent> element has four attributes, the id attribute extracts the annotation id information, the function attribute extracts the POS tags information (verb, noun, and preposition), and the start and end attributes extract the position of the annotation.

As a result, this step extracts the data from the code file and generates the annotation data in the <data> element. The following shows part of the results of process one in Appendix 6.

b.

```
<data>
        <sent id="b.1.1" function="verb" start="17" end="20"/>
        <sent id="b.1.1" function="verb" start="74" end="77"/>
    ...
</data>
```

The <data> element contains several child elements <sent>. Each <sent> element contains four attributes, where id attributes present the id of each annotation, function attributes present three POS tags, verb, noun, and preposition, start attributes present the start position of annotations, and end attributes present the end position of annotations. However, there are limitations in this step.

After extracting the annotation data, we manually select and copy the two layers of data and paste them into the original XML before proceeding to the next step. This process of manual selection of replication risks generating unnecessary errors and affects the results since the extracted annotation data is added to the converted XML file in the next step. Finally, the generated XML file contains POS annotation data showing in Appendix 7.

In process two, the second XSL file structures the XML file and marks the annotations within the original text. Appendix 8 shows the second XSL file, which has three templates. The first template structures the XML file by adding <document>, <body>, <paragraph> and <sentence> elements, and sets a framework for data extraction.

From line 6 to line 55, this template extracts the original text, sentence id from <seg> element and adds all the information into the <sentence> element. In addition, this template calls two templates “CountTags” and “TagText” to add the annotation text by using the <xsl:call-template> function. The following c sample shows the <xsl:call-template> function.

c.

```
<xsl:variable name="count">
          <xsl:call-template name="CountTags">
               <xsl:with-param name="id" select="$sentence_id"/>
          </xsl:call-template>
</xsl:variable>
```

The <xsl:variable> element defines a variable, it is named “count” by the attribute “name”. The variable element has a child element <xsl:call-template>, which calls the template “CountTags”. The child element <xsl: with-template> has two attributes, the “name” attribute specifies the name of the parameter and the “select” attribute provides the value of the parameter.

The code from line 57 to line 60 defines the “CountTags” template that is called <xsl:call-template>. The following d sample shows the template.

d.

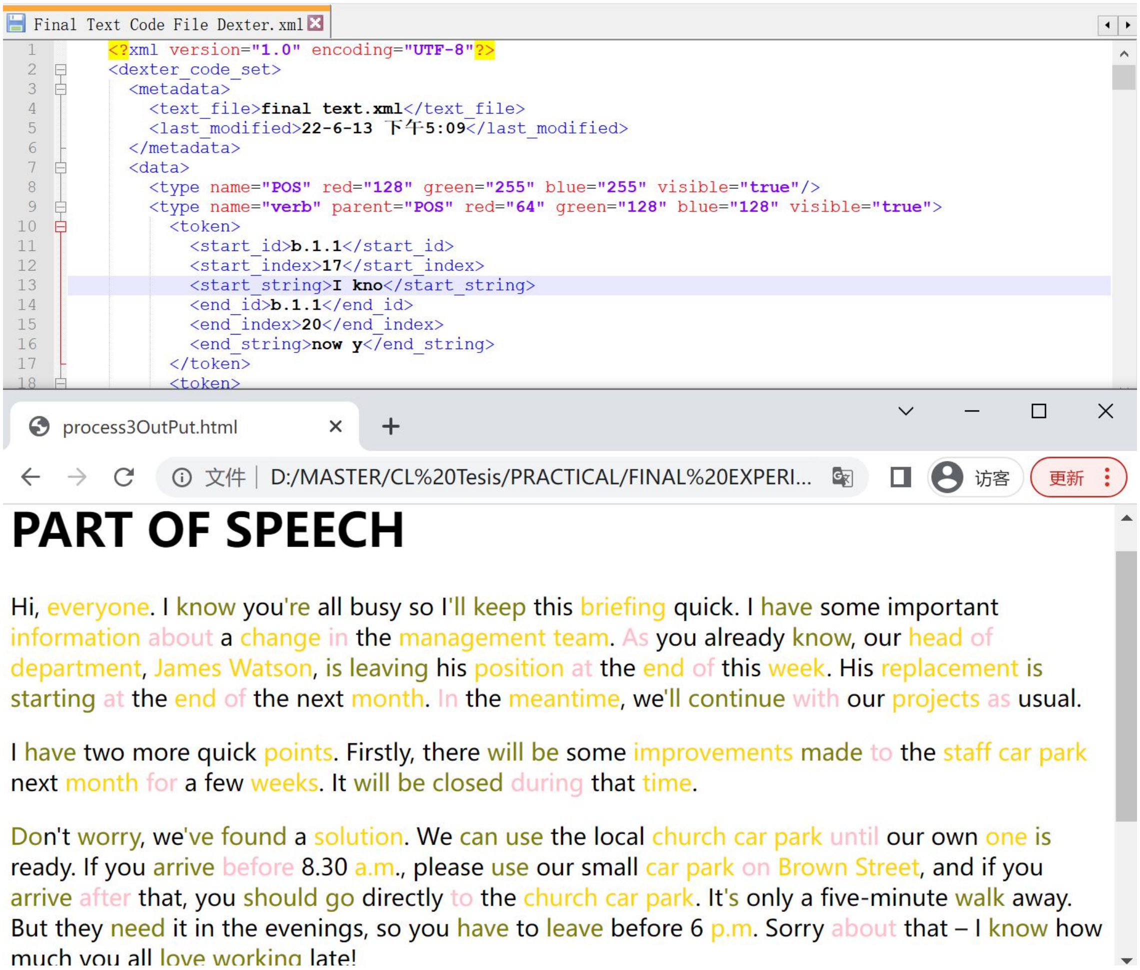
```
<xsl:template name="CountTags">
        <xsl:param name="id"/>
        <xsl:value-of select="count(//data/sent[@id = $id])"
</xsl:template>
```

This template is named as “CountTags” by attribute name. The child element <xsl:param> defines a parameter that is named as “id”. And then, the element <xsl:value-of> provides the value of this parameter through the attribute “select”.

Using the same functions and programming logic, the function of code from line 36 to 43 calls another template “TagText” to add annotation information. From line 62 to line 102, the “TagText” template defines six parameters and extracts the data of attribute “function,” “start” and “end” from <sent> element into “TagText” templates.

The “TagText template” marks out the annotations using the substring function to set <segment> element to locate before and after the tag text. Finally, process two generates the XML file that marks out all POS annotations within the sentences showing in Appendix 9. The generated XML file has the <document>, <body>, <paragraph>, <sentence> and <segement> element, and the <segement> element contains all the annotation.

In process three, in Task #2 of DEXTER shares the same procedures as process two with UAM, and the code uses the same functions and structures. Thus, the XSL file of process three is also shown in Appendix 3. As a result, we transform the XML files into HTML files, the following figure shows the final results of transformation by DEXTER.


Final Text Code File Dexter.xml
process3OutPut.html
文件 | D:/MASTER/CL%20Tesis/PRACTICAL/FINAL%20EXPERI...
访客
更新

```
<?xml version="1.0" encoding="UTF-8"?>
<dexter_code_set>
  <metadata>
    <text_file>final text.xml</text_file>
    <last_modified>22-6-13 下午5:09</last_modified>
  </metadata>
  <data>
    <type name="POS" red="128" green="255" blue="255" visible="true"/>
    <type name="verb" parent="POS" red="64" green="128" blue="128" visible="true">
      <token>
        <start_id>b.1.1</start_id>
        <start_index>17</start_index>
        <start_string>I kno</start_string>
        <end_id>b.1.1</end_id>
        <end_index>20</end_index>
        <end_string>now y</end_string>
      </token>
      <token>
```

PART OF SPEECH

Hi, everyone. I know you're all busy so I'll keep this briefing quick. I have some important information about a change in the management team. As you already know, our head of department, James Watson, is leaving his position at the end of this week. His replacement is starting at the end of the next month. In the meantime, we'll continue with our projects as usual.

I have two more quick points. Firstly, there will be some improvements made to the staff car park next month for a few weeks. It will be closed during that time.

Don't worry, we've found a solution. We can use the local church car park until our own one is ready. If you arrive before 8.30 a.m., please use our small car park on Brown Street, and if you arrive after that, you should go directly to the church car park. It's only a five-minute walk away. But they need it in the evenings, so you have to leave before 6 p.m. Sorry about that – I know how much you all love working late!

Figure 14. Result of Task #2 with DEXTER.

The top of Figure 14 shows the original XML file generated by DEXTER, and on the bottom, it presents the annotation results with different colors in the browser. In the end, the XML files, XSL files, and HTML files are packaged and uploaded to the GitHub platform[15] as an open-source project.

3.4.3 Result of Task #2

In general, Task #2 is designed to use these three computer tools to convert the

[15] the XML files, XSL files, and HTML files are packaged and uploaded to the GitHub platform, URL: https://github.com/GarritaGou/XSL_Project

XML files into HTML files and present the results in the browser.

After implementing the task, the recorded procedures analyze and compare the software. Task #2 proposes five questions concerning the comparison of these three computer tools.

N. Does the structure of XML files suit Task #2?

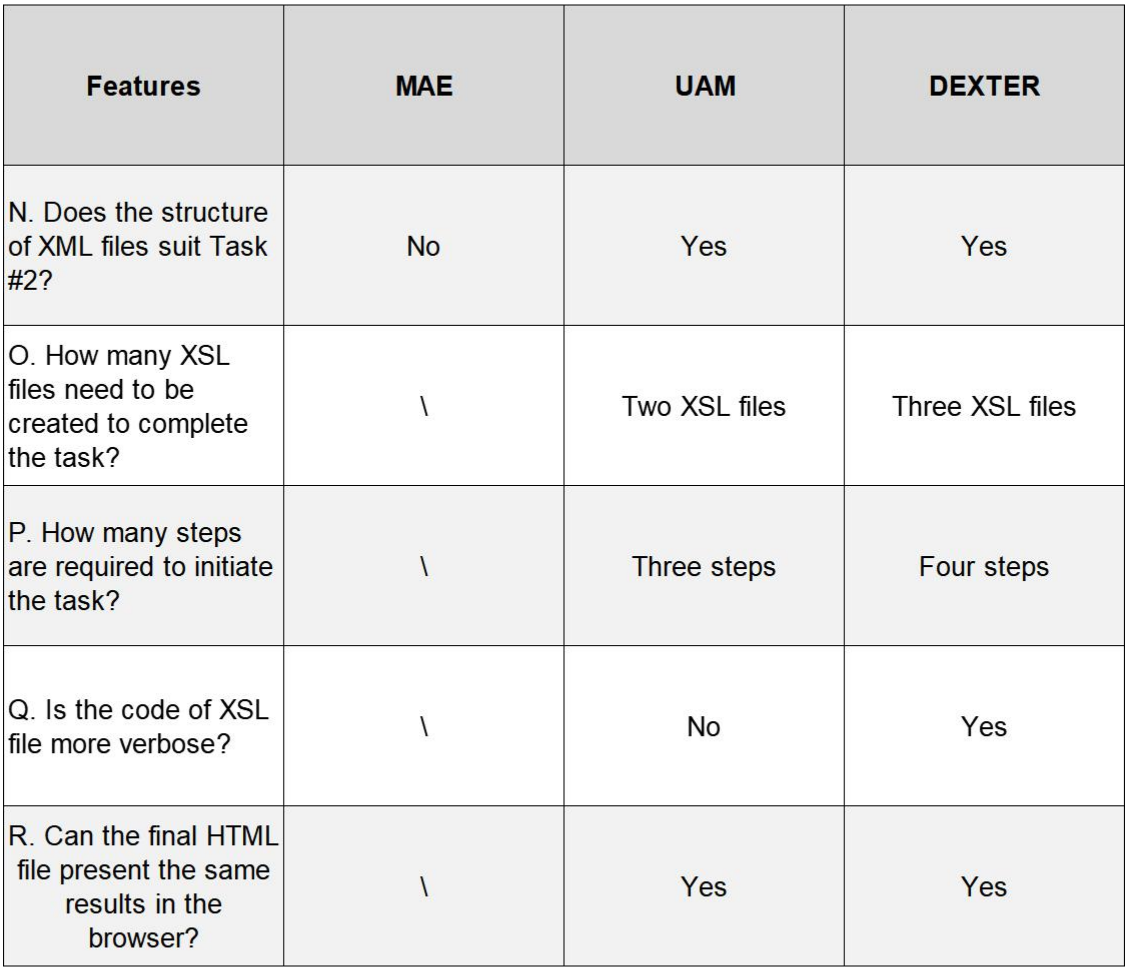
O. How many XSL files need to be created to complete the task?

P. How many steps are required to initiate the task?

Q. Is the code of XSL file more verbose?

R. Can the final HTML file present the same results in the browser?

The following Table 2 shows the results.



Features	MAE	UAM	DEXTER
N. Does the structure of XML files suit Task #2?	No	Yes	Yes
O. How many XSL files need to be created to complete the task?	\	Two XSL files	Three XSL files
P. How many steps are required to initiate the task?	\	Three steps	Four steps
Q. Is the code of XSL file more verbose?	\	No	Yes
R. Can the final HTML file present the same results in the browser?	\	Yes	Yes

Table 2. Comparison result two.

As for the N characteristic, the XML files created by UAM and DEXTER both suit Task #2. In addition, the original XML file generated by UAM contains the information of the original text and the annotation information, which shows the annotated text instead of the annotation data. So UAM has a better XML structure for initiating Task #2 than DEXTER.

As for the O and P characteristics, there are two XSL files and three steps for initiating Task #2 in UAM, while DEXTER requires three XSL files and four steps. From the aspects of the procedure and required XSL files, UAM has more advantages in converting its original XML file into an HTML over DEXTER.

Concerning the Q characteristic, DEXTER requires more verbose XSL code in the third XSL file because it needs to extract annotation data from the code file before proceeding with the following steps. In the first UAM XSL file, the code only uses one template to mark out the annotations in the original sentences, whereas Task #2 in DEXTER, the code in the third XSL file uses three templates to mark the annotations. In this aspect, to complete the procedure of converting the XML file into an HTML file, DEXTER needs more verbose XSL codes than UAM.

The characteristic R describes that both UAM and DEXTER can execute Task #2 successfully and present the same annotation results in the browser.

In conclusion, UAM and DEXTER have more advantages in converting their annotation results into the HTML file than MAE in regard to the XML file structure. UAM has more advantages than DEXTER in this task regarding the contents of distributing annotation data in the XML file.

4. CONCLUSION

This research provides theoretical basis and makes contributions to compare MAE, UAM and DEXTER by completing two academic tasks, where Task #1 designed a corpus sample and performed corpus annotation and Task #2 created XSL files to convert generated XML files into HTML files and showed them in the browser. First, the evaluation results derived from Task #1 suggest that MAE is a computer tool with a simplified interface and interactivity that suits inexperienced annotators according to features A and B. In addition, MAE and DEXTER are more suitable for professional linguists in resolving long-term and complex annotation tasks, in which DEXTER offers a higher user experience according to features E, I and J. Second, Task #2 proposes that UAM and DEXTER have more advantages in transforming annotation results into the HTML file than MAE in the aspect of the structure of XML files, in which UAM has more advantages in transformation than DEXTER according to feature O, P and Q.

Most importantly, this research contributes to the open source code uploading on the Github platform, so that linguists can implement the conversion themselves.

There is a limitation to the current research. XSL files in Task #2 can be improved to proceed automatically to make scientific results. In the future, we will create and update the XSL codes so that the extracted data is automatically divided into two layers and stored in two separate XML files to provide scientific results.

5. BIBLIOGRAPHY

Attia, M., Mcdonald, R., Petrov, S., & Kayadelen, T. 2017. PoS, Morphology and Dependencies Annotation Guidelines for Arabic.

Candito, M., Perrier, G., Guillaume, B., Ribeyre, C., Fort, K., Seddah, D., & de La Clergerie, É. V. (2014). *Deep syntax annotation of the sequoia french treebank*. In International Conference on Language Resources and Evaluation (LREC). Reykjavik, Iceland. https://hal.inria.fr/hal-00969191v2

Campbell, C. E., Eisenberg, A., & Melton, J. (2003). Xml schema. *ACM SIGMOD Record*, 32(2), pp.96-101.

Clark, J. (1999). *Xsl transformations (xslt).* World Wide Web Consortium *(W3C*). URL http://www. w3. org/TR/xslt, 103.

Day, D. S., McHenry, C., Kozierok, R., & Riek, L. D. (2004). Callisto: *A Configurable Annotation Workbench.* International Conference on Language Resources and Evaluation (LREC). Massachusetts, USA. http://www.lrec-conf.org/proceedings/lrec2004/pdf/612.pdf

Expert Advisory Groups on Language Engineering Standards (EAGLE).(1996a). *Recommendations for the Morphosyntactic Annotation of Corpora.* http://www.ilc.cnr.it/EAGLES96/annotate/annotate.html

Expert Advisory Groups on Language Engineering Standards (EAGLE).(1996b). *Recommendations for the Syntactic Annotation of Corpora.* http://www.ilc.cnr.it/EAGLES96/segsasg1/segsasg1.html

Expert Advisory Groups on Language Engineering Standards (EAGLE).(2000). *Corpus Encoding Standard.* https://www.cs.vassar.edu/CES/

Flickinger, D. (2008). *Toward a cross-framework parser annotation standard.* Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation. Manchester, UK. https://aclanthology.org/W08-13.

Fort, K., Guillaume, B., & Chastant, H. (2014). *Creating Zombilingo, a Game With A Purpose for dependency syntax annotation*. Proceedings of the First International Workshop on Gamification for Information Retrieval. Amsterdam, The Netherlands. https://doi.org/10.1145/2594776.2594777.

Goldberg, K. H. (2010). *XML: Visual quickstart guide*. Berkeley: Peachpit Press.

Gries, S. T., & Berez, A. L. (2017). Linguistic annotation in/for corpus linguistics. In *Handbook of linguistic annotation.* (pp.379-409). USA: Springer.

Hellmann, S., Chiarcos, C., & Ngomo, A. C. N. (2010). *The tiger corpus navigator.* Proc. 9th International Workshop on Treebanks and Linguistic Theories TLT-9. Germany. http://hdl.handle.net/10062/15891.

Hirst, D. (2006). *Phonetic and phonological annotation of speech prosody*. Analisi prosodica. teorie, modelli e sistemi di annotazione, Atti del II Convegno Nazionale della Societa Italiana di Scienze della Voce (AISV). Torriana.

Hoek, J., Evers-Vermeul, J., & Sanders, T. J. (2019). Using the cognitive approach to coherence relations for discourse annotation. *Dialogue & Discourse*, 10(2), pp.1-33.

Jiang, W., Huang, L., & Liu, Q. (2009). Automatic adaptation of annotation standards: Chinese word segmentation and pos tagging–a case study. *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. Suntec, Singapore. https://aclanthology.org/P09-1059.

Leech, G., & Wynne, M. (2005). Developing linguistic corpora: A guide to good practice. M. Wynne (Éd.)*. Developing linguistic corpora: a guide to good practice*, *92*, pp.17-29.

Long, W., Cai, X., Reid, J. E., Webber, B., & Xiong, D. (2020). Shallow discourse annotation for chinese ted talks. *arXiv preprint arXiv*.

López F. H., Reboiro J. M., Glez P. D., Aparicio, F., Gachet, D., Buenaga, M., & Fdez R. F. (2013). BioAnnote: A software platform for annotating biomedical documents with application in medical learning environments. *Computer methods and programs in biomedicine*, *111*(1), pp.139-147.

Maeda, K., & Strassel, S. M. (2004). *Computer tools for Large-Scale Corpus Development: Using AGTK at the Linguistic Data Consortium.* Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC). PA, USA.http://www.lrec-conf.org/proceedings/lrec2004/pdf/761.pdf

Maraoui, H., Haddar, K., & Romary, L. (2017). *Modeling of Al-Hadith Al-Shareef with TEI*. 2017 International Conference on Engineering & MIS (ICEMIS). Monastir, Tunisia. https://doi.org/

Morton, T. S., & LaCivita, J. (2003). *WordFreak: an open tool for linguistic* *annotation*. Companion Volume of the Proceedings of HLT-NAACL 2003-Demonstrations. Philadelphia, USA. https://aclanthology.org/N03-4

Müller, C., & Strube, M. (2002). A*n API for Discourse-level Access to XML-encoded Corpora*. Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC). Heidelberg, Germany.http://lrec-conf.org

Müller, C., & Strube, M. (2006). Multi-level annotation of linguistic data with MMAX2. *Corpus technology and language pedagogy: New resources, new tools, new methods*, 3, pp.197-214.

Neves, M., & Ševa, J. (2021). An extensive review of tools for manual annotation of documents. *Briefings in bioinformatics*, *22*(1), pp.146-163.

Ngonga Ngomo, A. C., & Lyko, K. (2012). *Eagle: Efficient active learning of link specifications using genetic programming*. Extended semantic web conference. Berlin, Heidelberg. https://aksw.org/Projects/LIMES.html

O'Donnell, M. (2008). *The UAM CorpusTool: Software for corpus annotation and exploration*. Proceedings of the XXVI Congreso de AESLA.Almeria, Spain. http://www.aesla.uji.es/congresoxxvii/

Palmer, M., Gildea, D., & Kingsbury, P. (2005). The proposition bank: An annotated corpus of semantic roles. *Computational linguistics*, 31(1), pp.71-106.

Miltsakaki, E., Prasad, R., Joshi, A. K., & Webber, B. L. (2004). *The Penn Discourse Treebank*. Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC). Philadelphia, USA. http://lrec-conf.org

Petasis, G., Karkaletsis, V., Paliouras, G., Androutsopoulos, I., & Spyropoulos, C. D. (2002). *Ellogon: A new text engineering platform*. Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC). Las Palmas, Canary Islands, Spain. http://www.lrec-conf.org/proceedings/lrec2002/pdf/211.pdf

Sperberg-McQueen, C. M., & Burnard, L. (Eds.). (1994). *Guidelines for electronic text encoding and interchange*. *Chicago and Oxford: Text Encoding Initiative*. Oxford, UK: Humanities Computing Unit, University of Oxford.

Starko, V. (2020). *Semantic Annotation for Ukrainian: Categorization Scheme, Principles, and Tools*. Proceedings of the 4th International Conference on

Computational Linguistics and Intelligent Systems (*COLINS).* Lviv, Ukraine. http://ceur-ws.org/

Stubbs, A. (2011). *MAE and MAI: lightweight annotation and adjudication tools.* Proceedings of the 5th linguistic annotation workshop. Massachusetts, USA. https://aclanthology.org/W11-0416

Tateisi, Y., Yakushiji, A., Ohta, T., & Tsujii, J. I. 2005. *Syntax Annotation for the GENIA corpus*. Companion Volume to the Proceedings of Conference including Posters/Demos and tutorial abstracts. Manchester, UK. https://aclanthology.org/I05-2038

Tidwell, D. (1999). Tutorial: XML programming in Java. *Cyber Evangelist. developer Works XML Team*.

Widlöcher, A., & Mathet, Y. (2012). *The glozz platform: A corpus annotation and mining tool*. Proceedings of the 2012 ACM symposium on Document engineering. Caen, France. https://doi.org/10.1145/2361354.2361394

Wilcock, G. (2009). Introduction to linguistic annotation and text analytics. *Synthesis Lectures on Human Language Technologies*, 2(1), pp.1-159.

APPENDICES

Appendix 1. First XSL file of process one for UAM

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" version="1.0" encoding="iso-8859-1" indent="yes"/>

    <!-- copy everything -->
    <xsl:template match="@*|node()">
        <xsl:copy>
            <xsl:apply-templates select="@*|node()"/>
        </xsl:copy>
    </xsl:template>

    <!-- matches some nodes and do nothing, i.e. delete them -->
    <xsl:template match="lang" />

     <xsl:template match="body">
        <body><paragraph><xsl:apply-templates select="node()" /></paragraph></body>
    </xsl:template>

    <xsl:template match="//sentence">
        <sentence><xsl:apply-templates select="node()" /></sentence>
    </xsl:template>

    <xsl:template match="//segment">
        <xsl:choose>
            <xsl:when test="@features='pos'">
                <sentence>
                    <xsl:apply-templates select="node()" />
                </sentence>
            </xsl:when>
            <xsl:otherwise>
                <segment>
                    <xsl:attribute name="pos">
                        <xsl:choose>
                            <xsl:when test="@features='pos;verb'">verb</xsl:when>
                            <xsl:when test="@features='pos;noun'">noun</xsl:when>
                            <xsl:when
test="@features='pos;preposition'">preposition</xsl:when>
                        </xsl:choose>
                    </xsl:attribute>
                    <xsl:apply-templates select="node()" />
                </segment>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:template>
</xsl:stylesheet>
```

Appendix 2. Generated XML file of UAM in process one

```
<?xml version="1.0" encoding="iso-8859-1"?>
<document>
  <header>
      <textfile>Texts/final text.txt</textfile>

  </header>
    <body>
      <paragraph>
         <sentence>Hi, <segment pos="noun">everyone</segment>.</sentence>
         <sentence>I <segment pos="verb">know</segment> you<segment
pos="verb">'re all busy so I'll <segment pos="verb">keep</segment> this <segment
pos="noun">briefing</segment> quick.</sentence>
         <sentence>I <segment pos="verb">have</segment> some important <segment
pos="noun">information</segment>
            <segment pos="preposition">about</segment> a <segment
pos="noun">change</segment>
            <segment pos="preposition">in</segment> the <segment
pos="noun">management</segment>
            <segment pos="noun">team</segment>.</sentence>
         <sentence>
            <sentence>As</sentence> you already <segment pos="verb">know</segment>,
our <segment pos="preposition">head</segment>
            <segment pos="preposition">of</segment>
            <segment pos="preposition">departme</segment>nt, <segment
pos="noun">James Watson</segment>, <segment pos="verb">is</segment>
            <segment pos="verb">leaving</segment> his <segment
pos="noun">position</segment>
            <segment pos="preposition">at</segment> the <segment
pos="noun">end</segment>
            <segment pos="preposition">of</segment> this <segment
pos="noun">week</segment>.</sentence>
         <sentence>His <segment pos="noun">replacement</segment>
            <segment pos="verb">is</segment>
            <segment pos="verb">start</segment>ing <segment
pos="preposition">at</segment> the <segment pos="noun">en</segment>d <segment
pos="preposition">of</segment> the next <segment
pos="noun">month</segment>.</sentence>
         <sentence>
```

```
          <segment pos="preposition">In</segment> the <segment
pos="noun">meantime</segment>, we<segment pos="verb">'ll</segment>
          <segment pos="verb">continue</segment>
          <segment pos="preposition">with</segment> our <segment
pos="noun">projects</segment>
          <sentence>as</sentence> usual.</sentence>

       <sentence>I <segment pos="verb">have</segment> two more quick <segment
pos="noun">points</segment>.</sentence>
       <sentence>Firstly, there <segment pos="verb">will</segment>
          <segment pos="verb">be</segment> some <segment
pos="noun">improvements</segment>
          <segment pos="verb">made</segment>
          <segment pos="preposition">to</segment> the <segment
pos="noun">staff</segment>
          <segment pos="noun">car</segment>
          <segment pos="noun">park</segment> next <segment
pos="noun">month</segment>
          <segment pos="preposition">for</segment> a <segment pos="noun">few
weeks</segment>.</sentence>
       <sentence>It <segment pos="verb">will</segment>
          <segment pos="verb">be</segment>
          <segment pos="verb">closed</segment>
          <segment pos="preposition">during</segment> that <segment
pos="noun">time</segment>.</sentence>

       <sentence>
          <segment pos="verb">Do</segment>n't <segment pos="verb">worry</segment>,
we<segment pos="verb">'ve</segment>
          <segment pos="verb">found</segment> a <segment
pos="noun">solution</segment>.</sentence>
       <sentence>We <segment pos="verb">can</segment>
          <segment pos="verb">use</segment> the local <segment
pos="noun">church</segment>
          <segment pos="noun">car</segment>
          <segment pos="noun">park</segment>
          <segment pos="preposition">until</segment> our own one <segment
pos="verb">is</segment> ready.</sentence>
       <sentence>If you <segment pos="verb">arrive</segment>
          <segment pos="preposition">before</segment> 8.30 <segment
```

```
pos="noun">a.m.</segment>, please <segment pos="verb">use</segment> our small
<segment pos="noun">car</segment>
        <segment pos="noun">park</segment>
        <segment pos="preposition">on</segment>
        <segment pos="noun">Brown</segment>
        <segment pos="noun">Street</segment>, and if you <segment
pos="verb">arrive</segment>
        <segment pos="preposition">after</segment> that, you <segment
pos="verb">should</segment>
        <segment pos="verb">go</segment> directly <segment
pos="preposition">to</segment> the <segment pos="noun">church</segment>
        <segment pos="noun">car</segment>
        <segment pos="noun">park</segment>.</sentence>
      <sentence>It's only a five-minute <segment pos="verb">walk</segment>
away.</sentence>
      <sentence>But they <segment pos="verb">need</segment> it in the <segment
pos="noun">evenings</segment>, so you <segment pos="verb">have</segment>
        <segment pos="preposition">to</segment>
        <segment pos="verb">leave</segment>
        <segment pos="preposition">before</segment> 6 <segment
pos="noun">p.m.</segment>
      </sentence>
      <sentence>Sorry <segment pos="preposition">about</segment> that &#x2013; I
<segment pos="verb">know</segment> how much you all <segment
pos="verb">love</segment>
        <segment pos="verb">working</segment> late!</sentence>

      <sentence>
        <sentence>The other <segment pos="noun">thing</segment> I <segment
pos="verb">wanted</segment>
          <segment pos="preposition">to</segment>
          <segment pos="verb">tell</segment> you <segment
pos="preposition">about</segment>
          <segment pos="verb">is</segment> that the <segment pos="noun">canteen
<segment pos="verb">has</segment> now <segment pos="verb">introduced</segment> a
cashless <segment pos="noun">payment</segment>
            <segment pos="noun">system</segment>.</segment>
          <sentence>So, you <segment pos="verb">can</segment>'t <segment
pos="verb">use</segment> cash <segment pos="preposition">for</segment>
            <segment pos="noun">payments</segment> any more.</sentence>
```

```
                    <sentence>You <segment pos="verb">can</segment>
                    </sentence>
                    <segment pos="verb">pay</segment> directly <segment
pos="preposition">with your <segment pos="noun">smartphone</segment> or you
<segment pos="verb">can</segment>
                    <segment pos="verb">pay</segment>
                    <segment pos="verb">using</segment> your <segment
pos="noun">comp</segment>any</sentence>
                <segment pos="noun">ID</segment>
                <segment pos="noun">card</segment>.</sentence>
            <sentence>The total <segment pos="noun">amount</segment>
                <segment pos="verb">put</segment>
                <segment pos="preposition">on</segment> your <segment
pos="noun">company</segment>
                <segment pos="noun">ID</segment>
                <segment pos="noun">card</segment>
                <segment pos="verb">come</segment>s off your <segment
pos="noun">salary</segment> at the <segment pos="noun">end</segment>
                <segment pos="preposition">of</segment> each <segment
pos="noun">month</segment>.</sentence>

            <sentence>OK.</sentence>
            <sentence>That<segment pos="verb">'s</segment> it?</sentence>
            <sentence>
                <segment pos="verb">Are</segment> there any <segment
pos="noun">questions</segment>?</sentence>
        </paragraph>
    </body>
</document>
```

Appendix 3. Second XSL file of process two for UAM and process three for DEXTER

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" version="4.0" encoding="iso-8859-1" indent="yes"/>
    <!-- identity template - copies all elements and its children and attributes -->

    <xsl:template match="@*|node()">
        <xsl:copy>
            <xsl:apply-templates select="@*|node()" />
        </xsl:copy>
    </xsl:template>

    <xsl:template match="//document">
        <html><xsl:apply-templates select="node()" /></html>
    </xsl:template>

    <xsl:template match="//textfile">
        <title><xsl:apply-templates select="node()" /></title>
    </xsl:template>

    <xsl:template match="//paragraph">
        <p><xsl:apply-templates select="node()" /></p>
    </xsl:template>

    <xsl:template match="//segment[@pos='verb']">
        <font color="olive"><xsl:apply-templates select="node()" /></font>
    </xsl:template>

    <xsl:template match="//segment[@pos='noun']">
        <font color="Gold"><xsl:apply-templates select="node()" /></font>
    </xsl:template>

    <xsl:template match="//segment[@pos='preposition']">
        <font color="pink"><xsl:apply-templates select="node()" /></font>
    </xsl:template>
</xsl:stylesheet>
```

Appendix 4. Original generated XML file of UAM in process two

```
<?xml version="1.0" encoding="iso-8859-1"?>
<html>

    <header>

        <title>Texts/final text.txt</title>


    </header>

    <body>

        <p>

            <sentence>Hi, <font color="Gold">everyone</font>.
            </sentence>

            <sentence>I <font color="olive">know</font> you<font color="olive">'re</font> all
busy so I'll <font color="olive">keep</font> this <font color="Gold">briefing</font> quick.
            </sentence>

            <sentence>I <font color="olive">have</font> some important <font
color="Gold">information</font>
                <font color="pink">about</font> a <font color="Gold">change</font>
                <font color="pink">in</font> the <font color="Gold">management</font>
                <font color="Gold">team</font>.
            </sentence>

            <sentence>

                <sentence>As</sentence> you already <font color="olive">know</font>, our <font
color="pink">head</font>
                <font color="pink">of</font>
                <font color="pink">departme</font>nt, <font color="Gold">James Watson</font>,
<font color="olive">is</font>
                <font color="olive">leaving</font> his <font color="Gold">position</font>
                <font color="pink">at</font> the <font color="Gold">end</font>
                <font color="pink">of</font> this <font color="Gold">week</font>.
```

```
</sentence>

<sentence>His <font color="Gold">replacement</font>
    <font color="olive">is</font>
    <font color="olive">start</font>ing <font color="pink">at</font> the <font color="Gold">en</font>d <font color="pink">of</font> the next <font color="Gold">month</font>.
</sentence>

<sentence>
    <font color="pink">In</font> the <font color="Gold">meantime</font>, we<font color="olive">'ll</font>
    <font color="olive">continue</font>
    <font color="pink">with</font> our <font color="Gold">projects</font>

    <sentence>as</sentence> usual.
</sentence>


<sentence>I <font color="olive">have</font> two more quick <font color="Gold">points</font>.
</sentence>

<sentence>Firstly, there <font color="olive">will</font>
    <font color="olive">be</font> some <font color="Gold">improvements</font>
    <font color="olive">made</font>
    <font color="pink">to</font> the <font color="Gold">staff</font>
    <font color="Gold">car</font>
    <font color="Gold">park</font> next <font color="Gold">month</font>
    <font color="pink">for</font> a <font color="Gold">few weeks</font>.
</sentence>

<sentence>It <font color="olive">will</font>
    <font color="olive">be</font>
    <font color="olive">closed</font>
    <font color="pink">during</font> that <font color="Gold">time</font>.
</sentence>


<sentence>
    <font color="olive">Do</font>n't <font color="olive">worry</font>, we<font
```

```
color="olive">'ve</font>
                <font color="olive">found</font> a <font color="Gold">solution</font>.
            </sentence>

            <sentence>We <font color="olive">can</font>
                <font color="olive">use</font> the local <font color="Gold">church</font>
                <font color="Gold">car</font>
                <font color="Gold">park</font>
                <font color="pink">until</font> our own one <font color="olive">is</font> ready.
            </sentence>

            <sentence>If you <font color="olive">arrive</font>
                <font color="pink">before</font> 8.30 <font color="Gold">a.m.</font>, please <font color="olive">use</font> our small <font color="Gold">car</font>
                <font color="Gold">park</font>
                <font color="pink">on</font>
                <font color="Gold">Brown</font>
                <font color="Gold">Street</font>, and if you <font color="olive">arrive</font>
                <font color="pink">after</font> that, you <font color="olive">should</font>
                <font color="olive">go</font> directly <font color="pink">to</font> the <font color="Gold">church</font>
                <font color="Gold">car</font>
                <font color="Gold">park</font>.
            </sentence>

            <sentence>It's only a five-minute <font color="olive">walk</font> away.
            </sentence>

            <sentence>But they <font color="olive">need</font> it in the <font color="Gold">evenings</font>, so you <font color="olive">have</font>
                <font color="pink">to</font>
                <font color="olive">leave</font>
                <font color="pink">before</font> 6 <font color="Gold">p.m.</font>

            </sentence>

            <sentence>Sorry <font color="pink">about</font> that &#x2013; I <font color="olive">know</font> how much you all <font color="olive">love</font>
                <font color="olive">working</font> late!
            </sentence>
```

<sentence>

<sentence>The other <font color="Gold">thing</font> I <font color="olive">wanted</font>
<font color="pink">to</font>
<font color="olive">tell</font> you <font color="pink">about</font>
<font color="olive">is</font> that the <font color="Gold">canteen <font color="olive">has</font> now <font color="olive">introduced</font> a cashless <font color="Gold">payment</font>
<font color="Gold">system</font>.</font>

<sentence>So, you <font color="olive">can</font>'t <font color="olive">use</font> cash <font color="pink">for</font>
<font color="Gold">payments</font> any more.
</sentence>

<sentence>You <font color="olive">can</font>

</sentence>
<font color="olive">pay</font> directly <font color="pink">with</font> your <font color="Gold">smartphone</font> or you <font color="olive">can</font>
<font color="olive">pay</font>
<font color="olive">using</font> your <font color="Gold">comp</font>any
</sentence>
<font color="Gold">ID</font>
<font color="Gold">card</font>.
</sentence>

<sentence>The total <font color="Gold">amount</font>
<font color="olive">put</font>
<font color="pink">on</font> your <font color="Gold">company</font>
<font color="Gold">ID</font>
<font color="Gold">card</font>
<font color="olive">come</font>s off your <font color="Gold">salary</font> at the <font color="Gold">end</font>
<font color="pink">of</font> each <font color="Gold">month</font>.
</sentence>

```
            <sentence>OK.</sentence>

            <sentence>That<font color="olive">'s</font> it?
            </sentence>

            <sentence>
                <font color="olive">Are</font> there any <font color="Gold">questions</font>?
            </sentence>

        </p>

    </body>

</html>
```

Appendix 5. First XSL file of process one for DEXTER

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema" exclude-result-prefixes="xs" version="2.0">
    <xsl:output method="xml" version="1.0" encoding="iso-8859-1" indent="yes"/>

    <xsl:template match="/">
        <data>
            <xsl:for-each select="//data/type">
                <xsl:variable name="function" select="@name"/>
                <xsl:for-each select="token">
                    <sent>
                        <xsl:attribute name="id"><xsl:value-of
select="start_id"/></xsl:attribute>
                        <xsl:attribute name="function"><xsl:value-of
select="$function"/></xsl:attribute>
                        <xsl:attribute name="start"><xsl:value-of
select="start_index"/></xsl:attribute>
                        <xsl:attribute name="end"><xsl:value-of
select="end_index"/></xsl:attribute>
                    </sent>
                </xsl:for-each>
            </xsl:for-each>
        </data>
    </xsl:template>
</xsl:stylesheet>
```

Appendix 6. Original generated XML file of process one for DEXTER

```
<?xml version="1.0" encoding="iso-8859-1"?>
<data>
    <sent id="b.1.1" function="verb" start="17" end="20"/>
    <sent id="b.1.1" function="verb" start="74" end="77"/>
    <sent id="b.1.1" function="verb" start="160" end="163"/>
    <sent id="b.1.1" function="verb" start="207" end="213"/>
    <sent id="b.3.1" function="verb" start="20" end="24"/>
    <sent id="b.3.1" function="verb" start="45" end="47"/>
    <sent id="b.3.1" function="verb" start="110" end="115"/>
    <sent id="b.3.1" function="verb" start="142" end="144"/>
    <sent id="b.3.1" function="verb" start="223" end="224"/>
    <sent id="b.3.1" function="verb" start="303" end="306"/>
    <sent id="b.3.1" function="verb" start="343" end="347"/>
    <sent id="b.3.1" function="verb" start="384" end="387"/>
    <sent id="b.3.1" function="verb" start="406" end="409"/>
    <sent id="b.4.1" function="verb" start="19" end="24"/>
    <sent id="b.4.1" function="verb" start="29" end="32"/>
    <sent id="b.4.1" function="verb" start="72" end="81"/>
    <sent id="b.4.1" function="verb" start="124" end="126"/>
    <sent id="b.4.1" function="verb" start="164" end="166"/>
    <sent id="b.4.1" function="verb" start="209" end="211"/>
    <sent id="b.4.1" function="verb" start="213" end="217"/>
    <sent id="b.3.1" function="verb" start="41" end="43"/>
    <sent id="b.2.1" function="verb" start="3" end="6"/>
    <sent id="b.2.1" function="verb" start="51" end="52"/>
    <sent id="b.2.1" function="verb" start="72" end="75"/>
    <sent id="b.2.1" function="verb" start="135" end="136"/>
    <sent id="b.2.1" function="verb" start="138" end="143"/>
    <sent id="b.3.1" function="verb" start="7" end="11"/>
    <sent id="b.3.1" function="verb" start="193" end="198"/>
    <sent id="b.3.1" function="verb" start="335" end="338"/>
    <sent id="b.1.1" function="verb" start="46" end="49"/>
    <sent id="b.1.1" function="verb" start="42" end="44"/>
    <sent id="b.1.1" function="verb" start="204" end="205"/>
    <sent id="b.1.1" function="verb" start="269" end="270"/>
    <sent id="b.1.1" function="verb" start="272" end="279"/>
    <sent id="b.1.1" function="verb" start="330" end="332"/>
    <sent id="b.1.1" function="verb" start="334" end="341"/>
```

<sent id="b.2.1" function="verb" start="46" end="49"/>
<sent id="b.2.1" function="verb" start="130" end="133"/>
<sent id="b.3.1" function="verb" start="1" end="2"/>
<sent id="b.3.1" function="verb" start="16" end="18"/>
<sent id="b.3.1" function="verb" start="93" end="94"/>
<sent id="b.3.1" function="verb" start="216" end="221"/>
<sent id="b.3.1" function="verb" start="283" end="286"/>
<sent id="b.3.1" function="verb" start="261" end="262"/>
<sent id="b.4.1" function="verb" start="43" end="45"/>
<sent id="b.4.1" function="verb" start="64" end="66"/>
<sent id="b.4.1" function="verb" start="118" end="120"/>
<sent id="b.4.1" function="verb" start="160" end="162"/>
<sent id="b.4.1" function="verb" start="205" end="207"/>
<sent id="b.4.1" function="verb" start="258" end="260"/>
<sent id="b.4.1" function="verb" start="286" end="290"/>
<sent id="b.5.1" function="verb" start="9" end="10"/>
<sent id="b.5.1" function="verb" start="16" end="18"/>
<sent id="b.1.1" function="verb" start="25" end="27"/>
<sent id="b.3.1" function="verb" start="411" end="417"/>
<sent id="b.1.1" function="noun" start="94" end="104"/>
<sent id="b.1.1" function="noun" start="5" end="12"/>
<sent id="b.1.1" function="noun" start="56" end="63"/>
<sent id="b.1.1" function="noun" start="114" end="119"/>
<sent id="b.1.1" function="noun" start="128" end="137"/>
<sent id="b.1.1" function="noun" start="139" end="142"/>
<sent id="b.1.1" function="noun" start="170" end="173"/>
<sent id="b.1.1" function="noun" start="178" end="187"/>
<sent id="b.1.1" function="noun" start="190" end="194"/>
<sent id="b.1.1" function="noun" start="196" end="201"/>
<sent id="b.1.1" function="noun" start="219" end="226"/>
<sent id="b.1.1" function="noun" start="247" end="250"/>
<sent id="b.1.1" function="noun" start="257" end="267"/>
<sent id="b.1.1" function="noun" start="288" end="290"/>
<sent id="b.1.1" function="noun" start="304" end="308"/>
<sent id="b.1.1" function="noun" start="318" end="325"/>
<sent id="b.1.1" function="noun" start="352" end="359"/>
<sent id="b.2.1" function="noun" start="23" end="28"/>
<sent id="b.2.1" function="noun" start="59" end="70"/>
<sent id="b.2.1" function="noun" start="84" end="88"/>
<sent id="b.2.1" function="noun" start="90" end="92"/>

```
<sent id="b.2.1" function="noun" start="94" end="97"/>
<sent id="b.2.1" function="noun" start="104" end="108"/>
<sent id="b.2.1" function="noun" start="120" end="124"/>
<sent id="b.2.1" function="noun" start="157" end="160"/>
<sent id="b.3.1" function="noun" start="28" end="35"/>
<sent id="b.3.1" function="noun" start="59" end="64"/>
<sent id="b.3.1" function="noun" start="66" end="68"/>
<sent id="b.3.1" function="noun" start="70" end="73"/>
<sent id="b.3.1" function="noun" start="89" end="91"/>
<sent id="b.3.1" function="noun" start="242" end="247"/>
<sent id="b.3.1" function="noun" start="249" end="251"/>
<sent id="b.3.1" function="noun" start="253" end="256"/>
<sent id="b.3.1" function="noun" start="168" end="172"/>
<sent id="b.3.1" function="noun" start="174" end="179"/>
<sent id="b.3.1" function="noun" start="156" end="158"/>
<sent id="b.3.1" function="noun" start="358" end="360"/>
<sent id="b.3.1" function="noun" start="129" end="131"/>
<sent id="b.4.1" function="noun" start="11" end="15"/>
<sent id="b.4.1" function="noun" start="56" end="62"/>
<sent id="b.4.1" function="noun" start="94" end="100"/>
<sent id="b.4.1" function="noun" start="102" end="107"/>
<sent id="b.4.1" function="noun" start="187" end="196"/>
<sent id="b.4.1" function="noun" start="137" end="144"/>
<sent id="b.4.1" function="noun" start="128" end="131"/>
<sent id="b.4.1" function="noun" start="270" end="276"/>
<sent id="b.4.1" function="noun" start="278" end="279"/>
<sent id="b.4.1" function="noun" start="281" end="284"/>
<sent id="b.4.1" function="noun" start="301" end="306"/>
<sent id="b.4.1" function="noun" start="315" end="317"/>
<sent id="b.4.1" function="noun" start="327" end="331"/>
<sent id="b.5.1" function="noun" start="30" end="38"/>
<sent id="b.3.1" function="noun" start="160" end="163"/>
<sent id="b.1.1" function="noun" start="235" end="237"/>
<sent id="b.4.1" function="noun" start="224" end="230"/>
<sent id="b.4.1" function="noun" start="232" end="233"/>
<sent id="b.4.1" function="noun" start="235" end="238"/>
<sent id="b.4.1" function="noun" start="251" end="256"/>
<sent id="b.1.1" function="preposition" start="106" end="110"/>
<sent id="b.1.1" function="preposition" start="121" end="122"/>
<sent id="b.1.1" function="preposition" start="145" end="146"/>
```

```
<sent id="b.1.1" function="preposition" start="175" end="176"/>
<sent id="b.1.1" function="preposition" start="228" end="229"/>
<sent id="b.1.1" function="preposition" start="239" end="240"/>
<sent id="b.1.1" function="preposition" start="281" end="282"/>
<sent id="b.1.1" function="preposition" start="292" end="293"/>
<sent id="b.1.1" function="preposition" start="311" end="312"/>
<sent id="b.1.1" function="preposition" start="343" end="346"/>
<sent id="b.1.1" function="preposition" start="361" end="362"/>
<sent id="b.2.1" function="preposition" start="77" end="78"/>
<sent id="b.2.1" function="preposition" start="110" end="112"/>
<sent id="b.2.1" function="preposition" start="145" end="150"/>
<sent id="b.3.1" function="preposition" start="117" end="122"/>
<sent id="b.3.1" function="preposition" start="165" end="166"/>
<sent id="b.3.1" function="preposition" start="200" end="204"/>
<sent id="b.3.1" function="preposition" start="75" end="79"/>
<sent id="b.3.1" function="preposition" start="235" end="236"/>
<sent id="b.3.1" function="preposition" start="369" end="373"/>
<sent id="b.4.1" function="preposition" start="133" end="135"/>
<sent id="b.4.1" function="preposition" start="177" end="180"/>
<sent id="b.4.1" function="preposition" start="262" end="263"/>
<sent id="b.4.1" function="preposition" start="319" end="320"/>
<sent id="b.4.1" function="preposition" start="308" end="309"/>
<sent id="b.1.1" function="simple present" start="17" end="20"/>
<sent id="b.1.1" function="simple present" start="25" end="27"/>
<sent id="b.1.1" function="simple present" start="74" end="77"/>
<sent id="b.1.1" function="simple present" start="160" end="163"/>
<sent id="b.2.1" function="simple present" start="3" end="6"/>
<sent id="b.3.1" function="simple present" start="1" end="2"/>
<sent id="b.3.1" function="simple present" start="7" end="11"/>
<sent id="b.3.1" function="simple present" start="41" end="43"/>
<sent id="b.3.1" function="simple present" start="110" end="115"/>
<sent id="b.3.1" function="simple present" start="142" end="144"/>
<sent id="b.3.1" function="simple present" start="193" end="198"/>
<sent id="b.3.1" function="simple present" start="261" end="262"/>
<sent id="b.3.1" function="simple present" start="283" end="286"/>
<sent id="b.3.1" function="simple present" start="303" end="306"/>
<sent id="b.3.1" function="simple present" start="384" end="387"/>
<sent id="b.3.1" function="simple present" start="406" end="409"/>
<sent id="b.4.1" function="simple present" start="43" end="45"/>
<sent id="b.4.1" function="simple present" start="29" end="32"/>
```

```
    <sent id="b.4.1" function="simple present" start="118" end="120"/>
    <sent id="b.4.1" function="simple present" start="160" end="162"/>
    <sent id="b.4.1" function="simple present" start="205" end="207"/>
    <sent id="b.4.1" function="simple present" start="258" end="260"/>
    <sent id="b.4.1" function="simple present" start="286" end="290"/>
    <sent id="b.5.1" function="simple present" start="9" end="10"/>
    <sent id="b.5.1" function="simple present" start="16" end="18"/>
    <sent id="b.3.1" function="simple present" start="93" end="94"/>
    <sent id="b.1.1" function="present continuous" start="204" end="213"/>
    <sent id="b.1.1" function="present continuous" start="269" end="279"/>
    <sent id="b.3.1" function="present perfect" start="16" end="24"/>
    <sent id="b.4.1" function="present perfect" start="64" end="81"/>
    <sent id="b.4.1" function="simple past" start="19" end="24"/>
</data>
```

Appendix 7. Final generated XML file of process one for DEXTER

```
<?xml version="1.0" encoding="UTF-8"?>
<TEI.2>
  <teiHeader>
    <fileDesc>
      <titleStmt>
        <title/>
      </titleStmt>
      <publicationStmt/>
      <sourceDesc>
        <recordingStmt/>
      </sourceDesc>
    </fileDesc>
    <encodingDesc/>
    <profileDesc>
      <creation/>
      <langUsage/>
      <textClass/>
      <textDesc/>
      <particDesc/>
      <settingDesc>
        <setting/>
      </settingDesc>
    </profileDesc>
    <revisionDesc/>
  </teiHeader>
  <text>
    <body id="b">
      <p id="b.1">
        <seg id="b.1.1">Hi, everyone. I know you're all busy so I'll keep this briefing quick. I have some important information about a change in the management team. As you already know, our head of department, James Watson, is leaving his position at the end of this week. His replacement is starting at the end of the next month. In the meantime, we'll continue with our projects as usual. </seg>
      </p>
      <p id="b.2">
        <seg id="b.2.1">I have two more quick points. Firstly, there will be some improvements made to the staff car park next month for a few weeks. It will be closed during that time. </seg>
      </p>
      <p id="b.3">
        <seg id="b.3.1">Don't worry, we've found a solution. We can use the local church car park until our own one is ready. If you arrive before 8.30 a.m., please use our small car park on Brown Street, and if you arrive after that, you should go directly to the church car park. It's only a five-minute walk away. But they need it in the evenings, so you have to leave before 6 p.m. Sorry about that – I know how much you all love working late! </seg>
      </p>
      <p id="b.4">
        <seg id="b.4.1">The other thing I wanted to tell you about is that the canteen has now introduced a cashless payment system. So, you can't use cash for payments any more. You can pay directly with your smartphone or you can pay using your company ID card. The total amount put on your company ID card comes off your salary at the end of each month. </seg>
      </p>
      <p id="b.5">
        <seg id="b.5.1">OK. That's it? Are there any questions? </seg>
      </p>
    </body>
    <data>
   <sent id="b.1.1" function="verb" start="17" end="20"/>
   <sent id="b.1.1" function="verb" start="74" end="77"/>
   <sent id="b.1.1" function="verb" start="160" end="163"/>
   <sent id="b.1.1" function="verb" start="207" end="213"/>
   <sent id="b.3.1" function="verb" start="20" end="24"/>
   <sent id="b.3.1" function="verb" start="45" end="47"/>
   <sent id="b.3.1" function="verb" start="110" end="115"/>
   <sent id="b.3.1" function="verb" start="142" end="144"/>
   <sent id="b.3.1" function="verb" start="223" end="224"/>
   <sent id="b.3.1" function="verb" start="303" end="306"/>
   <sent id="b.3.1" function="verb" start="343" end="347"/>
   <sent id="b.3.1" function="verb" start="384" end="387"/>
   <sent id="b.3.1" function="verb" start="406" end="409"/>
   <sent id="b.4.1" function="verb" start="19" end="24"/>
   <sent id="b.4.1" function="verb" start="29" end="32"/>
   <sent id="b.4.1" function="verb" start="72" end="81"/>
   <sent id="b.4.1" function="verb" start="124" end="126"/>
   <sent id="b.4.1" function="verb" start="164" end="166"/>
   <sent id="b.4.1" function="verb" start="209" end="211"/>
   <sent id="b.4.1" function="verb" start="213" end="217"/>
   <sent id="b.3.1" function="verb" start="41" end="43"/>
   <sent id="b.2.1" function="verb" start="3" end="6"/>
```

```
<sent id="b.2.1" function="verb" start="51" end="52"/>
<sent id="b.2.1" function="verb" start="72" end="75"/>
<sent id="b.2.1" function="verb" start="135" end="136"/>
<sent id="b.2.1" function="verb" start="138" end="143"/>
<sent id="b.3.1" function="verb" start="7" end="11"/>
<sent id="b.3.1" function="verb" start="193" end="198"/>
<sent id="b.3.1" function="verb" start="335" end="338"/>
<sent id="b.1.1" function="verb" start="46" end="49"/>
<sent id="b.1.1" function="verb" start="42" end="44"/>
<sent id="b.1.1" function="verb" start="204" end="205"/>
<sent id="b.1.1" function="verb" start="269" end="270"/>
<sent id="b.1.1" function="verb" start="272" end="279"/>
<sent id="b.1.1" function="verb" start="330" end="332"/>
<sent id="b.1.1" function="verb" start="334" end="341"/>
<sent id="b.2.1" function="verb" start="46" end="49"/>
<sent id="b.2.1" function="verb" start="130" end="133"/>
<sent id="b.3.1" function="verb" start="1" end="2"/>
<sent id="b.3.1" function="verb" start="16" end="18"/>
<sent id="b.3.1" function="verb" start="93" end="94"/>
<sent id="b.3.1" function="verb" start="216" end="221"/>
<sent id="b.3.1" function="verb" start="283" end="286"/>
<sent id="b.3.1" function="verb" start="261" end="262"/>
<sent id="b.4.1" function="verb" start="43" end="45"/>
<sent id="b.4.1" function="verb" start="64" end="66"/>
<sent id="b.4.1" function="verb" start="118" end="120"/>
<sent id="b.4.1" function="verb" start="160" end="162"/>
<sent id="b.4.1" function="verb" start="205" end="207"/>
<sent id="b.4.1" function="verb" start="258" end="260"/>
<sent id="b.4.1" function="verb" start="286" end="290"/>
<sent id="b.5.1" function="verb" start="9" end="10"/>
<sent id="b.5.1" function="verb" start="16" end="18"/>
<sent id="b.1.1" function="verb" start="25" end="27"/>
<sent id="b.3.1" function="verb" start="411" end="417"/>
<sent id="b.1.1" function="noun" start="94" end="104"/>
<sent id="b.1.1" function="noun" start="5" end="12"/>
<sent id="b.1.1" function="noun" start="56" end="63"/>
<sent id="b.1.1" function="noun" start="114" end="119"/>
<sent id="b.1.1" function="noun" start="128" end="137"/>
<sent id="b.1.1" function="noun" start="139" end="142"/>
<sent id="b.1.1" function="noun" start="170" end="173"/>
```

```
<sent id="b.1.1" function="noun" start="178" end="187"/>
<sent id="b.1.1" function="noun" start="190" end="194"/>
<sent id="b.1.1" function="noun" start="196" end="201"/>
<sent id="b.1.1" function="noun" start="219" end="226"/>
<sent id="b.1.1" function="noun" start="247" end="250"/>
<sent id="b.1.1" function="noun" start="257" end="267"/>
<sent id="b.1.1" function="noun" start="288" end="290"/>
<sent id="b.1.1" function="noun" start="304" end="308"/>
<sent id="b.1.1" function="noun" start="318" end="325"/>
<sent id="b.1.1" function="noun" start="352" end="359"/>
<sent id="b.2.1" function="noun" start="23" end="28"/>
<sent id="b.2.1" function="noun" start="59" end="70"/>
<sent id="b.2.1" function="noun" start="84" end="88"/>
<sent id="b.2.1" function="noun" start="90" end="92"/>
<sent id="b.2.1" function="noun" start="94" end="97"/>
<sent id="b.2.1" function="noun" start="104" end="108"/>
<sent id="b.2.1" function="noun" start="120" end="124"/>
<sent id="b.2.1" function="noun" start="157" end="160"/>
<sent id="b.3.1" function="noun" start="28" end="35"/>
<sent id="b.3.1" function="noun" start="59" end="64"/>
<sent id="b.3.1" function="noun" start="66" end="68"/>
<sent id="b.3.1" function="noun" start="70" end="73"/>
<sent id="b.3.1" function="noun" start="89" end="91"/>
<sent id="b.3.1" function="noun" start="242" end="247"/>
<sent id="b.3.1" function="noun" start="249" end="251"/>
<sent id="b.3.1" function="noun" start="253" end="256"/>
<sent id="b.3.1" function="noun" start="168" end="172"/>
<sent id="b.3.1" function="noun" start="174" end="179"/>
<sent id="b.3.1" function="noun" start="156" end="158"/>
<sent id="b.3.1" function="noun" start="358" end="360"/>
<sent id="b.3.1" function="noun" start="129" end="131"/>
<sent id="b.4.1" function="noun" start="11" end="15"/>
<sent id="b.4.1" function="noun" start="56" end="62"/>
<sent id="b.4.1" function="noun" start="94" end="100"/>
<sent id="b.4.1" function="noun" start="102" end="107"/>
<sent id="b.4.1" function="noun" start="187" end="196"/>
<sent id="b.4.1" function="noun" start="137" end="144"/>
<sent id="b.4.1" function="noun" start="128" end="131"/>
<sent id="b.4.1" function="noun" start="270" end="276"/>
<sent id="b.4.1" function="noun" start="278" end="279"/>
```

```
   <sent id="b.4.1" function="noun" start="281" end="284"/>
   <sent id="b.4.1" function="noun" start="301" end="306"/>
   <sent id="b.4.1" function="noun" start="315" end="317"/>
   <sent id="b.4.1" function="noun" start="327" end="331"/>
   <sent id="b.5.1" function="noun" start="30" end="38"/>
   <sent id="b.3.1" function="noun" start="160" end="163"/>
   <sent id="b.1.1" function="noun" start="235" end="237"/>
   <sent id="b.4.1" function="noun" start="224" end="230"/>
   <sent id="b.4.1" function="noun" start="232" end="233"/>
   <sent id="b.4.1" function="noun" start="235" end="238"/>
   <sent id="b.4.1" function="noun" start="251" end="256"/>
   <sent id="b.1.1" function="preposition" start="106" end="110"/>
   <sent id="b.1.1" function="preposition" start="121" end="122"/>
   <sent id="b.1.1" function="preposition" start="145" end="146"/>
   <sent id="b.1.1" function="preposition" start="175" end="176"/>
   <sent id="b.1.1" function="preposition" start="228" end="229"/>
   <sent id="b.1.1" function="preposition" start="239" end="240"/>
   <sent id="b.1.1" function="preposition" start="281" end="282"/>
   <sent id="b.1.1" function="preposition" start="292" end="293"/>
   <sent id="b.1.1" function="preposition" start="311" end="312"/>
   <sent id="b.1.1" function="preposition" start="343" end="346"/>
   <sent id="b.1.1" function="preposition" start="361" end="362"/>
   <sent id="b.2.1" function="preposition" start="77" end="78"/>
   <sent id="b.2.1" function="preposition" start="110" end="112"/>
   <sent id="b.2.1" function="preposition" start="145" end="150"/>
   <sent id="b.3.1" function="preposition" start="117" end="122"/>
   <sent id="b.3.1" function="preposition" start="165" end="166"/>
   <sent id="b.3.1" function="preposition" start="200" end="204"/>
   <sent id="b.3.1" function="preposition" start="75" end="79"/>
   <sent id="b.3.1" function="preposition" start="235" end="236"/>
   <sent id="b.3.1" function="preposition" start="369" end="373"/>
   <sent id="b.4.1" function="preposition" start="133" end="135"/>
   <sent id="b.4.1" function="preposition" start="177" end="180"/>
   <sent id="b.4.1" function="preposition" start="262" end="263"/>
   <sent id="b.4.1" function="preposition" start="319" end="320"/>
   <sent id="b.4.1" function="preposition" start="308" end="309"/>
   </data>
  </text>
</TEI.2>
```

Appendix 8. Second XSL file of process two for DEXTER

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xs="http://www.w3.org/2001/XMLSchema" exclude-result-prefixes="xs" version="2.0">
    <xsl:output method="xml" version="1.0" encoding="iso-8859-1" indent="yes"/>

    <xsl:key name="grammar" match="sent" use="@id"/>

    <xsl:template match="/">
        <document>
            <body>
                <xsl:for-each select="//p">   >
                    <paragraph>
                        <sentence>
                            <xsl:variable name="text" select="seg"/>
                            <xsl:variable name="sentence_id" select="seg/@id"/>
                            <xsl:attribute name="id">
                                <xsl:value-of select="$sentence_id"/>
                            </xsl:attribute>

                            <xsl:variable name="count">

                                <xsl:call-template name="CountTags">
                                    <xsl:with-param name="id" select="$sentence_id"/>
                                </xsl:call-template>
                            </xsl:variable>



                            <xsl:choose>
                                <xsl:when test="$count > 0">

                                    <xsl:variable name="constructions"
select="key('grammar', $sentence_id)" />
                                    <xsl:variable name="sorted_start" as="element()*">
                                        <xsl:for-each select="$constructions">

                                            <xsl:sort select="@start"
data-type="number"/>
                                            <index><xsl:value-of
```

```
select="@start"/></index>
                                        </xsl:for-each>
                                    </xsl:variable>


                                    <xsl:call-template name="TagText">

                                        <xsl:with-param name="text" select="$text"/>

                                        <xsl:with-param name="constructions"
select="$constructions"/>
                                        <xsl:with-param name="sorted_start"
select="$sorted_start"/>
                                        <xsl:with-param name="num" select="1"/>

                                        <xsl:with-param name="count"
select="$count"/>
                                        <xsl:with-param name="new_start" select="0"/>

                                    </xsl:call-template>

                                </xsl:when>
                                <xsl:otherwise>
                                    <xsl:value-of select="$text"/>
                                </xsl:otherwise>
                            </xsl:choose>

                        </sentence>
                    </paragraph>
                </xsl:for-each>
            </body>
        </document>
    </xsl:template>

    <xsl:template name="CountTags">
        <xsl:param name="id"/>
        <xsl:value-of select="count(//data/sent[@id = $id])"/>
    </xsl:template>

    <xsl:template name="TagText">
        <xsl:param name="text" />
        <xsl:param name="constructions" />
        <xsl:param name="sorted_start" />
        <xsl:param name="num" />
        <xsl:param name="count" />
        <xsl:param name="new_start" />

        <xsl:variable name="pos" select="$sorted_start[$num]"/>

        <xsl:variable name="input" select="$constructions[@start=$pos]"/>

        <xsl:variable name="function" select="$input/@function"/>
        <xsl:variable name="start_index" select="$input/@start"/>
        <xsl:variable name="end_index" select="$input/@end"/>
        <xsl:variable name="length" select="$end_index - $start_index + 1"/>
        <xsl:variable name="target" select="substring($text, $start_index - $new_start, $length)"/>

        <xsl:value-of select="substring-before($text, $target)"/>
        <segment>
            <xsl:attribute name="function">
                <xsl:value-of select="$function"/>
            </xsl:attribute>
            <xsl:value-of select="$target"/>
        </segment>

        <xsl:choose>
            <xsl:when test="$num=$count">
                <xsl:value-of select="substring-after($text, $target)"/>
            </xsl:when>
            <xsl:otherwise>
                <xsl:call-template name="TagText">
                    <xsl:with-param name="text" select="substring-after($text, $target)"/>
                    <xsl:with-param name="constructions" select="$constructions"/>
                    <xsl:with-param name="sorted_start" select="$sorted_start"/>
                    <xsl:with-param name="num" select="$num + 1"/>
                    <xsl:with-param name="count" select="$count"/>
                    <xsl:with-param name="new_start" select="$end_index"/>
                </xsl:call-template>
            </xsl:otherwise>
```

```
            </xsl:choose>
        </xsl:template>

    </xsl:stylesheet>
```

Appendix 9. Generated XML file of DEXTER in process two

```
<?xml version="1.0" encoding="iso-8859-1"?>
<document>
    <body>
        <paragraph>
            <sentence id="b.1.1">Hi, <segment function="noun">everyone</segment>. I <segment function="verb">know</segment> you<segment function="verb">'re</segment> all busy so I<segment function="verb">'ll</segment>
                <segment function="verb">keep</segment> this <segment function="noun">briefing</segment> quick. I <segment function="verb">have</segment> some important <segment function="noun">information</segment>
                <segment function="preposition">about</segment> a <segment function="noun">change</segment>
                <segment function="preposition">in</segment> the <segment function="noun">management</segment>
                <segment function="noun">team</segment>. <segment function="preposition">As</segment> you already <segment function="verb">know</segment>, our <segment function="noun">head</segment>
                <segment function="preposition">of</segment>
                <segment function="noun">department</segment>, <segment function="noun">James</segment>
                <segment function="noun">Watson</segment>, <segment function="verb">is</segment>
                <segment function="verb">leaving</segment> his <segment function="noun">position</segment>
                <segment function="preposition">at</segment> the <segment function="noun">end</segment>
                <segment function="preposition">of</segment> this <segment function="noun">week</segment>. His <segment function="noun">replacement</segment>
                <segment function="verb">is</segment>
                <segment function="verb">starting</segment>
                <segment function="preposition">at</segment> the <segment function="noun">end</segment>
                <segment function="preposition">of</segment> the next <segment function="noun">month</segment>. <segment function="preposition">In</segment> the <segment function="noun">meantime</segment>, we<segment function="verb">'ll</segment>
                <segment function="verb">continue</segment>
                <segment function="preposition">with</segment> our <segment function="noun">projects</segment>
                <segment function="preposition">as</segment> usual. </sentence>
        </paragraph>
        <paragraph>
            <sentence id="b.2.1">I <segment function="verb">have</segment> two more quick
      <segment function="noun">points</segment>. Firstly, there <segment
      function="verb">will</segment>
                <segment function="verb">be</segment> some <segment
      function="noun">improvements</segment>
                <segment function="verb">made</segment>
                <segment function="preposition">to</segment> the <segment
      function="noun">staff</segment>
                <segment function="noun">car</segment>
                <segment function="noun">park</segment> next <segment
      function="noun">month</segment>
                <segment function="preposition">for</segment> a few <segment
      function="noun">weeks</segment>. It <segment function="verb">will</segment>
                <segment function="verb">be</segment>
                <segment function="verb">closed</segment>
                <segment function="preposition">during</segment> that <segment
      function="noun">time</segment>. </sentence>
        </paragraph>
        <paragraph>
            <sentence id="b.3.1">
                <segment function="verb">Do</segment>n't <segment
      function="verb">worry</segment>, we<segment function="verb">'ve</segment>
                <segment function="verb">found</segment> a <segment
      function="noun">solution</segment>. We <segment function="verb">can</segment>
                <segment function="verb">use</segment> the local <segment
      function="noun">church</segment>
                <segment function="noun">car</segment>
                <segment function="noun">park</segment>
                <segment function="preposition">until</segment> our own <segment
      function="noun">one</segment>
                <segment function="verb">is</segment> ready. If you <segment
      function="verb">arrive</segment>
                <segment function="preposition">before</segment> 8.30 <segment
      function="noun">a.m</segment>., please <segment function="verb">use</segment> our small
      <segment function="noun">car</segment>
                <segment function="noun">park</segment>
                <segment function="preposition">on</segment>
```

```
            <segment function="noun">Brown</segment>
            <segment function="noun">Street</segment>, and if you <segment
function="verb">arrive</segment>
            <segment function="preposition">after</segment> that, you <segment
function="verb">should</segment>
            <segment function="verb">go</segment> directly <segment
function="preposition">to</segment> the <segment function="noun">church</segment>
            <segment function="noun">car</segment>
            <segment function="noun">park</segment>. It<segment
function="verb">'s</segment> only a five-minute <segment function="verb">walk</segment>
away. But they <segment function="verb">need</segment> it in the evenings, so you <segment
function="verb">have</segment> to <segment function="verb">leave</segment> before 6
<segment function="noun">p.m</segment>. Sorry <segment
function="preposition">about</segment> that &#x2013; I <segment
function="verb">know</segment> how much you all <segment function="verb">love</segment>
            <segment function="verb">working</segment> late! </sentence>
      </paragraph>
      <paragraph>
         <sentence id="b.4.1">The other <segment function="noun">thing</segment> I
<segment function="verb">wanted</segment> to <segment function="verb">tell</segment> you
about<segment function="verb"> is</segment> that the <segment
function="noun">canteen</segment>
            <segment function="verb">has</segment> now <segment
function="verb">introduced</segment> a cashless <segment
function="noun">payment</segment>
            <segment function="noun">system</segment>. So, you <segment
function="verb">can</segment>'t <segment function="verb">use</segment>
            <segment function="noun">cash</segment>
            <segment function="preposition">for</segment>
            <segment function="noun">payments</segment> any more. You <segment
function="verb">can</segment>
            <segment function="verb">pay</segment> directly <segment
function="preposition">with</segment> your <segment
function="noun">smartphone</segment> or you <segment function="verb">can</segment>
            <segment function="verb">pay</segment>
            <segment function="verb">using</segment> your <segment
function="noun">company</segment>
            <segment function="noun">ID</segment>
            <segment function="noun">card</segment>. The total <segment
function="noun">amount</segment>
```

```
                <segment function="verb">put</segment>
                <segment function="preposition">on</segment> your <segment function="noun">company</segment>
                <segment function="noun">ID</segment>
                <segment function="noun">card</segment>
                <segment function="verb">comes</segment> off your <segment function="noun">salary</segment>
                <segment function="preposition">at</segment> the <segment function="noun">end</segment>
                <segment function="preposition">of</segment> each <segment function="noun">month</segment>. </sentence>
        </paragraph>
        <paragraph>
            <sentence id="b.5.1">OK. That<segment function="verb">'s</segment> it? <segment function="verb">Are</segment> there any <segment function="noun">questions</segment>? </sentence>
        </paragraph>
    </body>
</document>
```