Semi-Automatic Annotation Tool to Build Large Dependency Tree-Tagged Corpus[*]

Eun-Jin Park[a], Jae-Hoon Kim[b], Chang-Hyun Kim[a], and Young-Kill Kim[a]

[a]1 Dongsam-Dong, Youngdo-gu, Busan, 606-791, KOREA,
[b]161 Gajeong-Dong, Yuseong-gu, Daejeon. 305-700, KOREA
jhoon@mail.hhu.ac.kr, {ejpark,chkim,kimyk}@etri.re.kr

Abstract. Corpora annotated with lots of linguistic information are required to develop robust and statistical natural language processing systems. Building such corpora, however, is an expensive, labor-intensive, and time-consuming work. To help the work, we design and implement an annotation tool for establishing a Korean dependency tree-tagged corpus. Compared with other annotation tools, our tool is characterized by the following features: independence of applications, localization of errors, powerful error checking, instant annotated information sharing, user-friendly. Using our tool, we have annotated 100,904 Korean sentences with dependency structures. The number of annotators is 33, the average annotation time is about 4 minutes per sentence, and the total period of the annotation is 5 months. We are confident that we can have accurate and consistent annotations as well as reduced labor and time.

Keywords: Annotation (tagging) tool, Workbench, Error Editor, Corpus construction.

1. Introduction

More recently, a large corpus annotated with linguistic information is used in natural language processing. By using this corpus, natural language processing systems have learn some linguistic phenomena automatically. Building such a corpus, however, is an expensive, labor-intensive and time-consuming work. Furthermore, maintaining consistency of a constructed corpus is difficult. Therefore, we need an annotation tool for improving annotation efficiency and maintaining consistency. To help such work, some annotation tools (Atalay, 2003; Lim, 2002; Morton, 2003; Day, 1997; Brants T. and Plaehn, 2000) have already been used. In this paper, we describe an annotation tool for building a Korean dependency tree-tagged corpus with linguistic information about the segmentation of word phrases (called eojeols in Korean), part-of-speech (POS) tags, boundaries of chunks, and dependency links and relations. We design an annotation tool so that an annotator can carefully investigate them and edit errors on them through a GUI. We also design it so that errors in low level processing like POS tagging might not be propagated to higher level processing step like parsing. Moreover, the tool is characterized by the following features; 1) It is independent of special applications like information extraction. 2) It focuses on localizing errors to modules as far as possible, such as morphological analyzers. It can make annotators find and pay attention to errors related to the modules easily. 3) It has an error checking function to make possible that errors can not be stored as it can be. 4) It promptly shares annotated information among annotators so that annotators can keep consistency with others' annotation within a working group. 5) It has a user-friendly interface.

This paper is organized as follows: In Section 2, we introduce other annotation tools for establishing corpora In Section 3, we describe the architecture of our annotation tool for

[*] Copyright 2007 by Eun-Jin Park, Jae-Hoon Kim, Chang-Hyun Kim, and Young-Kill Kim

building a Korean dependency tree-tagged corpus. In Section 4 and 5, we explain the implementation details of our tool and guide process of the annotation using our tool, respectively. Finally, we draw conclusions, and discuss future works in Section 6.

2. Related Works

Several annotation tools (Atalay, 2003; Day, 1997; Lim, 2002; Morton, 2003; Day, 1997; Brants T. and Plaehn, 2000; Carletta, 2002) have been developed and used in several projects (KIBS, 2005; Marcus, 1994; SEJONG, 2005). In this section, we briefly introduce such annotation tools for building a tagged corpus: Alembic workbench (Day, 1997), WorkFreak (Morton, 2003) and a semi-automatic tree annotating workbench (Lim, 2002) developed in the Sejong Project (SEJONG, 2005).

2.1.Alembic Workbench

Alembic Workbench (Day, 1997) developed at MITR[1] is the system which annotates named-entity for an effective information extraction system. It supports multi-languages and SGML formats. Also it learns the user's working pattern to construct the corpus semi-automatically. It helps annotators by graphic user-interface. This system had been upgraded and released as Callisto[2] in 2004. In spite of such upgrade, this system is not yet for general purpose, only information extraction. Furthermore, adapting it to Korean requires preprocessing like morphological analysis and POS tagging.

2.2.WordFreak

WordFreak[3] (Morton, 2003) is a java-based linguistic annotation tool designed to support automatically annotating linguistic data and it employs active-learning for the human correction of automatically annotated data. It annotates several linguistic information like syntactic structure, named-entity and anaphoric information, etc. It provides automatic taggers for tokenization, POS tagging, chunking, full parsing, and name finding through OpenNLP[4] project and also automatically annotates linguistic information by learning the user pattern of work. And it can extend its component to other languages like Korean easily, but also requires preprocessing like morphological analysis and POS tagging for each language.

2.3.Korean semi-automatic tree annotation workbench

In this section, we will describe the workbench (Lim, 2002) which is building a Korean dependency tree-tagged corpus. It extracts various syntactic patterns from the constructed corpus based on the selected features, and automatically applies the extracted syntactic patterns to the appropriate states. It provides an integrated environment for searching, converting and editing Korean parsing tree corpus in the Sejong project (SEJONG, 2005). However, it is for a stand-alone system, but not for a multi-user system, and then cannot share annotated information instantly. It is improper for building a large-scale corpus.

3. Annotation Tool for Building a Korean Dependency Tree-Tagged Corpus

Our work annotates naturally-occurring text for linguistic structure. Most notably, we produce skeletal dependency trees with links and relations showing rough syntactic and semantic information called Korean dependency tree-tagged corpus. We also annotate text with

[1] http://www.mitre.org/tech/alembic-workbench/
[2] http://callisto.mitre.org/
[3] http://wordfreak.sourceforge.net/
[4] http://opennlp.sourceforge.net/

segmentation of word phrases (eojeols in Korean), POS tags, and chunk annotation. In this section, we describe the architecture of our annotation tool, called PPeditor, for establishing the Korean dependency tree-tagged corpus. It is designed for editing dependency trees generated from a Korean dependency parser (Kim, 1994), that is, our method for building a corpus is semi-automatic. It also is designed for sharing annotated results through a database (DB) promptly so that many annotators can work simultaneously to keep consistency of dependency tree-tagged corpus.

3.1.Architecture of PPeditor

Figure 1 shows the overall architecture of PPeditor consisting of a sentence analyzer, an annotation tool, and a DB. The sentence analyzer comprises four components: a morphological analyzer, a POS tagger, a partial parser and a dependency parser. The morphological analyzer segments a sentence into a sequence of morphemes and the POS tagger assigns POS tags to morphemes reflecting their syntactic category. The partial parser called a chunker uses a sequence of POS tags provided by a tagger and identify boundaries of syntactic groups like noun and verb groups having linear structures. The chunker preserves all the previously added information in the sentence and only creates the boundaries of constituents called chunks. Finally the dependency parser generates explicit dependency links that show the head-dependent relations between chunks.

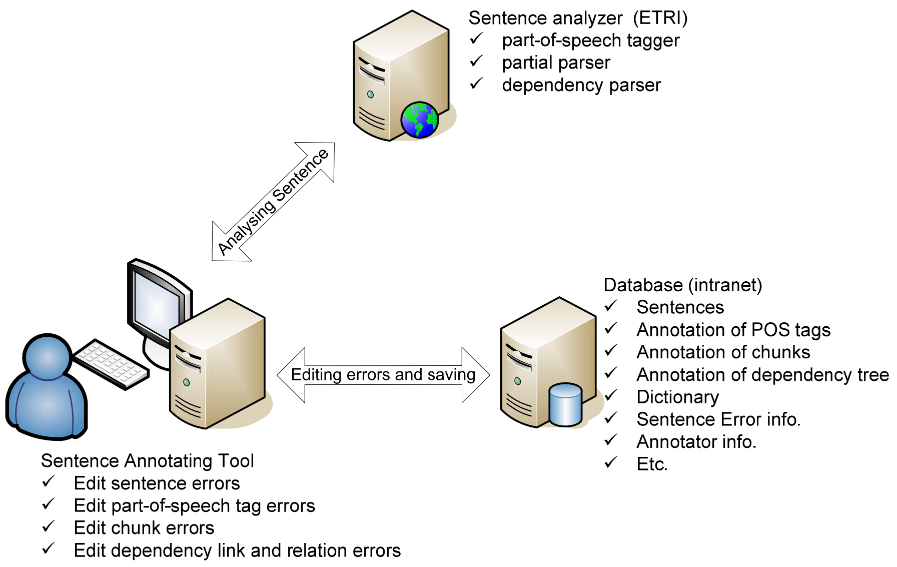


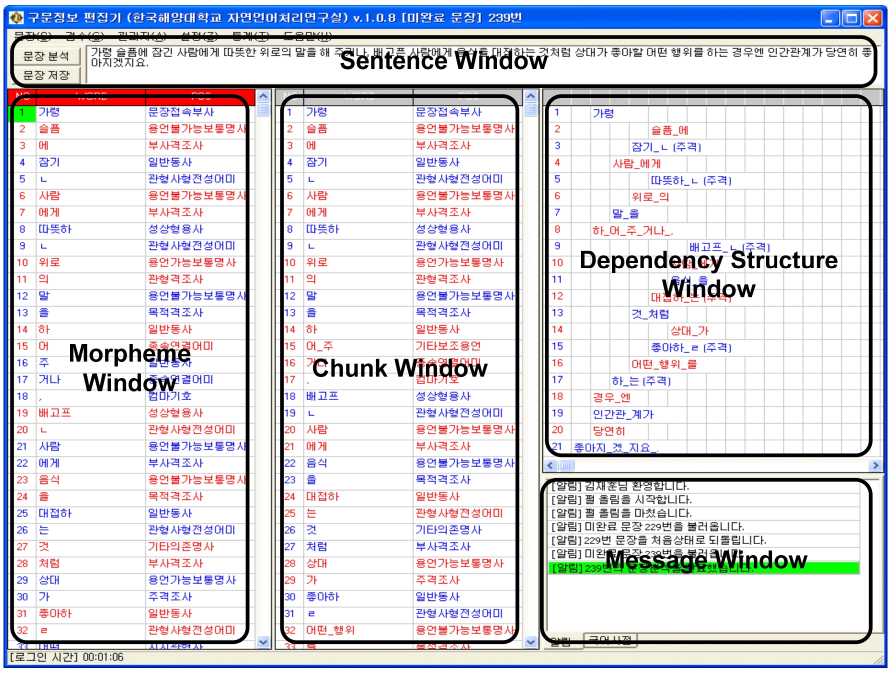
Figure 1: Architecture of PPeditor

The annotation tool helps annotators with editing several kinds of errors (spelling errors, spacing errors, segmentation errors, POS tagging errors, chunking errors, dependency structure errors), and so it is called an error editor also. The DB saves several kind of information on original sentences, annotations for the sentences, dictionaries, annotators, etc.

The annotation flow is as follows;
1. select a sentence from the DB.
2. send it to the sentence analyzer.
3. analyze it by the sentence analyzer.
4. receive analysis results from the sentence analyzer.
5. display the results in GUI.
6. observe errors on the results.
7. edit the errors.
8. repeat 2 through 7 until all the errors are corrected.
9. save the annotation results to the DB.

Step 8 is very important because the errors are propagated to the higher levels. Namely, errors of POS tagging is reflected into the partial parsing and syntactic parsing. The higher level analysis of sentences must be processed again if errors in the low level analysis are corrected. By doing this, the propagated errors are automatically disappeared and then the efficiency is improved greatly.

3.2.Graphic user interface of the PPeditor



구문정보 편집기 (한국해양대학교 자연언어처리연구실) v.1.0.8 [미완료 문장] 239번

가령 슬픔에 잠긴 사람에게 따뜻한 위로의 말을 해 주거나, 배고픈 사람에게 음식을 대접하는 것처럼 상대가 좋아할 어떤 행위를 하는 경우엔 인간관계가 당연히 좋아지겠지요.

NO	WORD	POS
1	가령	문장접속부사
2	슬픔	용언불가능보통명사
3	에	부사격조사
4	잠기	일반동사
5	ㄴ	관형사형전성어미
6	사람	용언불가능보통명사
7	에게	부사격조사
8	따뜻하	성상형용사
9	ㄴ	관형사형전성어미
10	위로	용언가능보통명사
11	의	관형격조사
12	말	용언불가능보통명사
13	을	목적격조사
14	하	일반동사
15	어	종속연결어미
16	주	일반동사
17	거나	종속연결어미
18	,	컴마기호
19	배고프	성상형용사
20	ㄴ	관형사형전성어미
21	사람	용언불가능보통명사
22	에게	부사격조사
23	음식	용언불가능보통명사
24	을	목적격조사
25	대접하	일반동사
26	는	관형사형전성어미
27	것	기타의존명사
28	처럼	부사격조사
29	상대	용언가능보통명사
30	가	주격조사
31	좋아하	일반동사
32	ㄹ	관형사형전성어미

NO	WORD	POS
1	가령	문장접속부사
2	슬픔	용언불가능보통명사
3	에	부사격조사
4	잠기	일반동사
5	ㄴ	관형사형전성어미
6	사람	용언불가능보통명사
7	에게	부사격조사
8	따뜻하	성상형용사
9	ㄴ	관형사형전성어미
10	위로	용언가능보통명사
11	의	관형격조사
12	말	용언불가능보통명사
13	을	목적격조사
14	하	일반동사
15	어_주	기타보조용언
16	거나	종속연결어미
17	,	컴마기호
18	배고프	성상형용사
19	ㄴ	관형사형전성어미
20	사람	용언불가능보통명사
21	에게	부사격조사
22	음식	용언불가능보통명사
23	을	목적격조사
24	대접하	일반동사
25	는	관형사형전성어미
26	것	기타의존명사
27	처럼	부사격조사
28	상대	용언가능보통명사
29	가	주격조사
30	좋아하	일반동사
31	ㄹ	관형사형전성어미
32	어떤_행위	용언불가능보통명사

1	가령
2	슬픔_에
3	잠기_ㄴ (주격)
4	사람_에게
5	따뜻하_ㄴ (주격)
6	위로_의
7	말_을
8	하_어_주_거나_,
9	배고프_ㄴ (주격)
10	사람_에게
11	음식_을
12	대접하_는 (주격)
13	것_처럼
14	상대_가
15	좋아하_ㄹ (주격)
16	어떤_행위_를
17	하_는 (주격)
18	경우_엔
19	인간관_계가
20	당연히
21	좋아지_겠_지요_.

[알림] 김재훈님 환영합니다.
[알림] 펄 올림을 시작합니다.
[알림] 펄 올림을 마쳤습니다.
[알림] 미완료 문장 229번을 불러옵니다.
[알림] 229번 문장을 처음상태로 되돌립니다.
[알림] 미완료 문장 239번을 불러옵니다.
[알림] 239번의 문장분석을 완료했습니다.

[로그인 시간] 00:01:06

Figure 2: GUI of the error editor

GUI (Graphical User Interface) of the annotation tool is consisting of a main menu, a sentence window, a morpheme window, a chunk window, a dependency structure window, and a message window. Each window displays information on sentences, morphemes, chunks, dependency structures, and messages, respectively. Table 1 shows relations between kinds of errors and windows. On the sentence window, errors on spelling and spacing of words are observed and corrected. On the morpheme window, errors on POS tags and segmentation of eojeols are edited. On the chunk window and the dependency structure window, errors on boundaries of chunks and on dependency structures, which comprise dependency links and relations, are rectified.

Table 1: Relations between kinds of errors and windows

Error	Window
spelling	sentence
spacing	
segmentation	morpheme
POS tag	morpheme / chunk
boundary of chunks	chunk
link with head	dependency structure
dependency relation	

Basic functions of an editor are insertion, deletion, and substitution of objects, and these functions are also basic in editing. For expert users, our system adopts hot keys for every function to improve the efficiency, for example, CTRL-I for insertion of a morpheme or a chunk. All mappings of hot keys and their functions are skipped due to limitation of space. To minimize typing errors on each window, combo-boxes are employed for editing POS tags and dependency relations.

3.3.Functions of PPeditor

Figure 3 shows the structure of PPeditor. The function of PPeditor is classified as the communication function, error edit function, guideline function, and corpus release function. All of the functions are connected with the user interface.

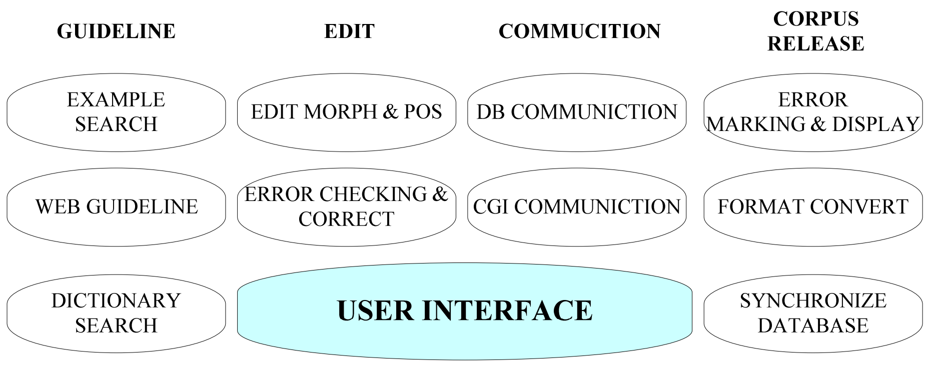


Figure 3: Functions of the PPeditor

The communication function is to communicate with the sentence analyzer and the data base. The communication of the sentence analyzer, which uses the CGI (Common Gateway Interface), is to send the sentence to the analyzer and receive the result of it. Therefore, it can freely set up in the different operating system or the computer system. The MySQL, which is connected with client by ODBC, was used as the data base which store sentences and the results of the annotation.

It is the basic function of PPeditor to correct the errors. It provided the functions of general editor which are insertion, deletion, and addition. But, it has some features which are not provided in the general editor. First, it automatically modified the errors which are repetitively caused by the sentence analyzer. Second, if the error of low step is removed, the result of correction will be reflected to high step in this system. Because the errors of the low step (ex: morphological analysis) are delivered to the high step (ex: syntax analysis). For example, the spelling errors at the sentence affect the morphological analysis and the syntax analysis. Third, it provides user to searching the similar language phenomenon or the similar contexts. There are a lot of ambiguous expressions in natural language. The annotator is able to keep consistency about them by using this function. Fourth, errors are not stored in the data base by checking the errors. The annotator may ignore simple error when he worked lots of time. Fifth, it helps the annotator search the dictionary and examples which are annotated by other annotators. He needs data such as a dictionary although he is familiar to his mother tongue. Sixth, there is focus function which highlights the current position. There is a lot of information on the screen, so he loses his position easily. Plate (i.e., photograph) captions appear underneath the figure, as can be seen in Plate 1. The plates and photographs should be centered.

The guideline function is studying skills for annotating sentences. This includes function of example search, web guideline, and dictionary search. Web guideline is online tutorial to help annotator. The example search is searching examples which are annotated sentences by other annotators. This function is very useful to keep constancy by searching the result of annotation.

The corpus release function is to exchange the results of annotation among annotation groups. It includes the function of format conversion which converts from database to text, and the function of synchronizing data among databases. And also it includes the function of error display which had marked by gaugers who review the dependency tree-tagged corpus. This helps annotator find error position easily.

4. Annotation of the Korean Dependency Tree-Tagged Corpus

Using our tool, we had annotated 100,904 Korean sentences of which all are over 20 words with syntactic structures, which comprise segmentation of eojeols, POS tags of morphemes and chunks, boundaries of chunks, dependency links and relations. 33 annotators, who are trained over 1 month, had worked for 5 months. There are 3 annotation subgroups, which worked 20,000, 40,000, and 40,000 sentences, respectively. Each group has local database and synchronizes their annotated sentences to central database server every 2 weeks. Figure 4 shows the organization of annotation subgroups and a central database server.

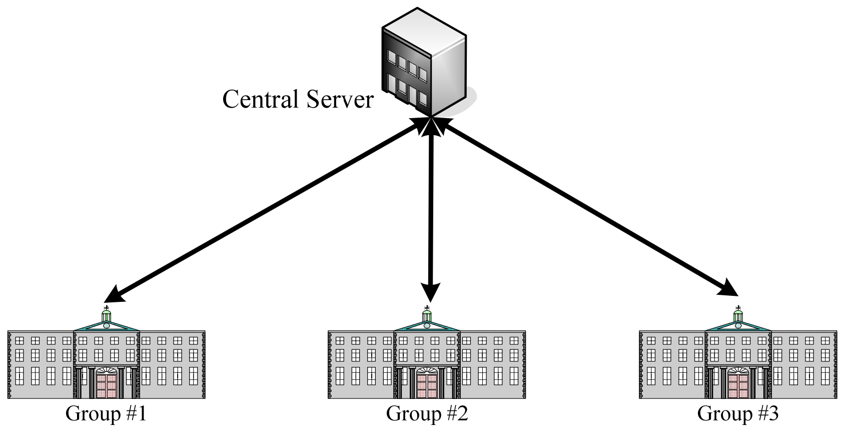


Figure 4: The organization of annotation subgroups

4.1.A scenario for annotating a corpus

The user level consists of gauger, advisers, annotators, and administrators. Most important users for annotating a corpus are annotators and advisers. In the normal case of annotation, an annotator works according to the workflow as mentioned in Section 3.1. In the case, some sentences extracted from running text are very difficult to annotate linguistic information; the annotator asks helps to advisers who are experts on annotations or well-educated corpus linguists. The advisers should hold profound knowledge in linguistics and could explain complex linguistic phenomena. Mostly one among senior annotators is in charges of the advice. Figure 5 shows this flow.

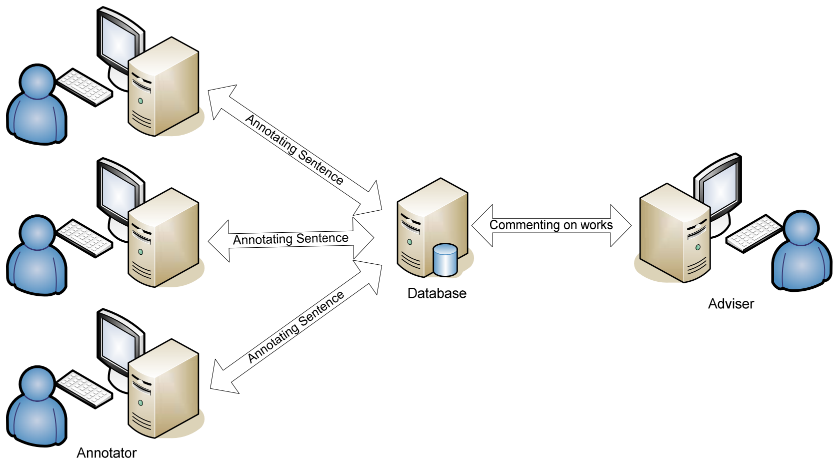


Figure 5: The scenario for annotating a sentence

Several annotators can access one record of the DB simultaneously. This causes problems when one sentence is annotated by two or more annotators. To avoid this problem, we use the 'status' field of the 'sent' table. If one annotator holds a sentence to work, the status of the sentence must be changed into 'reserved'. If the work is done, the status is recovered to the original status or is changed into other statuses.

4.2.The scenario for gauging a corpus

There is one gauger in the project, and each group has one adviser. The adviser examines annotated sentences so that errors can be reduced. If the adviser finds any errors on annotations, they explain the errors, give a guideline for removing the errors, and then ask the annotator of the annotations to remove the errors. We randomly selected the samples of annotated sentences every two weeks at the ratio of 1% and examine the quality of them. If an annotator cannot satisfy the accuracy rates of 99.9%, the adviser commands him to review all his sentences. Otherwise the adviser sends them to the gauger who gauges the sentences once more. This cross validation keeps the high reliability of the corpus. Figure 6 shows the scenario for gauging a corpus.

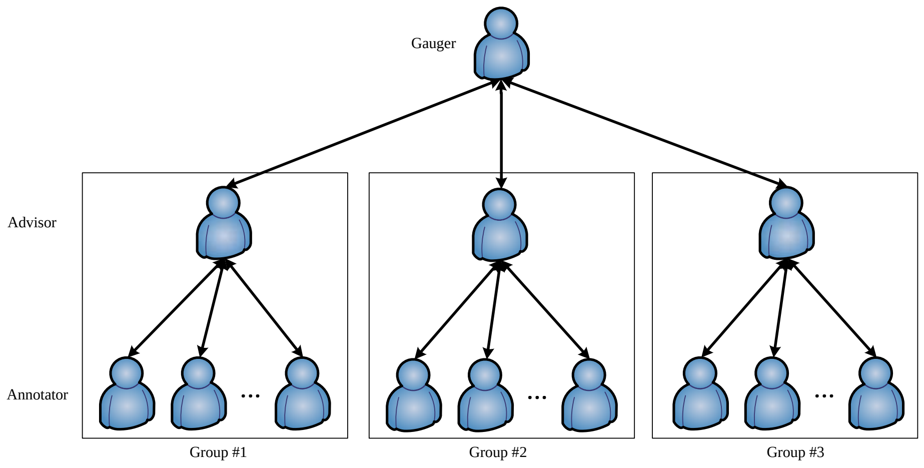


Figure 6: The scenario for gauging a corpus

4.3.Annotating the sentences

Table 2 shows the statistics on our annotated corpus, which consists of 100,904 Korean sentences with dependency structures. The average number of eojeols, morphemes, and chunks per sentence is 22, 43, and 36, respectively. A sentence has 22 eojeols on average and each eojeol has 1.9 parts-of-speech and 1.6 chunks on average. We commit 33 annotators to this work, which has taken 5 months. Each annotator has annotated 3,317 sentences on average and the average working time is 4 minutes and 32 sec on average (see Table 3). As we consider the length of the sentence, over 22 eojeols, the result is remarkable.

Table 2: Statistics on our annotated corpus

	Total	Average
eojeols	2,186,060	22
morphemes	4,344,200	43
chunks	3,644,790	36

Table 3: The Working Time & Num. Of sentence

Annotator No.	Num.	Average Time	Annotator No.	Num.	Average Time	Annotator No.	Num.	Average Time
#1	3,791	3:11	#12	2,018	3:14	#23	1,833	4:23
#2	6,780	3:36	#13	950	3:59	#24	2,364	3:40
#3	3,831	4:56	#14	4,565	4:16	#25	5,189	3:56
#4	3,193	4:25	#15	4,179	3:01	#26	2,757	4:28
#5	2,067	2:38	#16	5,131	3:28	#27	3,116	3:58
#6	3,082	4:43	#17	2,719	3:06	#28	2,100	3:46
#7	1,082	7:38	#18	1,770	4:36	#29	1,997	3:53
#8	2,415	4:10	#19	5,762	3:36	#30	4,567	3:48
#9	2,574	4:52	#20	1,548	4:47	#31	3,157	4:39
#10	7,017	4:30	#21	2,516	4:36	#32	1,131	3:11
#11	6,534	4:57	#22	6,454	3:02	#33	1,280	5:10

5. Conclusions and Future works

In this paper, we describe PPeditor which is the annotation tool for building Korean dependency tree-tagged corpus. The tree-tagged corpus contains many kind of linguistic information about segmentation of eojeols, POS tags of morphemes, boundaries of chunks, and dependency structures with heads and relations. Annotation of linguistic information using PPeditor means correcting the errors occurred in the sentence analyzer. Compared with other annotation tools, the tool is characterized by the following features: independence of applications, localization of errors, powerful error checking, instant annotated information sharing, and user-friendly.

Using our tool, we have annotated 100,904 Korean sentences with dependency structures. The number of annotators is 33, the average annotation time is 4 minutes per sentence, and the total period of the annotation is 5 months. We are confident that we can have accurate and consistent annotations as well as reduced labor and time.

In the near future, we will improve the function of automatic error-correction using machine learning techniques like transformation-based learning (Brill, 1995) and also add the function of detecting error-prone context automatically. We expect that these functions will not only reduce repetitive works, but also improve efficiency and effectiveness of annotations.

References

Atalay, N. B., Oflazer, K. and Say, B.: The Annotation Process in the Turkish Treebank. *Proceedings of the EACL Workshop on Linguistically Interpreted Corpora, Budapest, Hungary* .2003.

Brants T, and Plaehn, O.: Interactive corpus annotation. *Proceedings of the Second International Conference on Language Resources and Engineering* (LREC 2000) .2000. 453-459

Brill, E.: Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging. *Computation Linguistics Vol. 21(4):* .1995. 543-565

Carletta, J., McKelvie, D., Isard, A., Mengel, A., Klein, M. and Mller, M. B.: *A generic approach to software support for linguistic annotation using XML. G. Sampson & D. McCarth Eds., Readings in Corpus Linguistics, Continuum International* .2002.

Day, D., Aberdeen, J., Hirschman, L., Kozierok, R., Robinson, P. and Vilain, M.: Mixed-Initiative Development of Language Processing Systems. *Proceedings of the ANLP* .1997. 348-355

Day, D., Aberdeen, J., Hirschman, L., Kozierok, R., Robinson, P., and Vilain, M.: Mixed-Initiative Development of Language Processing Systems. *Proceedings of the Applied Natural Language Processing* .1997. 348-355

KIBS, http://kibs.kaist.ac.kr/, Korea Information Base System .2005.

Kim, C.-H., Kim, J.-H., Seo, J. and Kim, G. C.: A Right-to-Left Chart Parsing for Dependency Grammar using Headable Path. *Proceeding of the 1994 International Conference on Computer Processing of Oriental Languages* .1994. 175-180

Lim, J.-H., Park, S.-Y., Kwak, Y.-J., Rim, H.-C., Kim, U.-S. and Kang, B.-M.: Semi-Automatic Tree Annotators Using Statistical Syntactic Patterns. *Proceedings of the 14th Conference of Hangul and Korean Information Processing* .2002. 343-350

Marcus, M., Kim, G., Marcinkiewicz, M., MacIntyre, R., Bies, A., Ferguson, M., Katz, K. and Schasberger, B.: The Penn Treebank: Annotating Predicate Argument Structure. *Proceedings of ARPA Human Language Technology Workshop* .1994.

Morton, T. and LaCivita, J.: WordFreak: An Open Tool for Linguistic Annotation. *Proceedings. of the NAACL* (2003) 17-18

Ngai, G., and Florian, R.: Transformation-Based Learning in the Fast Lane. *Proceedings of the NAACL* (2001) 40-47

SEJONG, http://sejong.or.kr/english/, The 21st Century Sejong Project .2005.