情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2010-NL-199 No.19
2010/11/19

汎用アノテーションツール **Slate**

徳　永　健　伸[†1]　　Dain Kaplan[†1]　　飯　田　　龍[†1]

必要な情報がアノテーションされたコーパスは，統計的自然言語処理にとって必要不可欠な資源である．これまで多くのコーパス作成支援ツールが作られてきたが，そのほとんどは特定のアノテーションに特化しており，そのアノテーションのためのイン タフェースの工夫が中心的な関心であった．しかし，コーパスの構築には，アノテーションするドキュメントの管理，アノテータの管理などコーパス構築の過程全般に関係する種々のオブジェクトの管理も重要である．本稿では，まず，コーパス作成に必要な機能を洗い出し，それらを実現するための枠組について述べる．そして，その実装例として，コーパス構築の全行程を視野に入れ，ユーザの定義により広範な種類のアノテーションが可能な，汎用アノテーションツール Slate を紹介する．

Slate – A multi-purpose annotation tool

TOKUNAGA TAKENOBU,[†1] DAIN KAPLAN[†1] and IIDA RYU[†1]

Proper annotation process management is crucial to the construction of corpora, which are indispensable to the data-driven techniques that have come to the forefront in NLP during the last two decades. This paper first raises a list of 10 needs that any general purpose annotation system should address, such as user & role management, delegation & monitoring of work, multilingual support, and so on. A framework to address these needs is then proposed. Then an introduction of Slate (Segment and Link-based Annotation Tool *Enhanced*), the second iteration of a web-based annotation tool, which is being rewritten to implement the proposed framework follows, accompanied by a real world example demonstrating how the tool works.

†1 東京工業大学 大学院情報理工学研究科
Department of Computer Science, Tokyo Institute of Technology

1. Introduction

Corpus-based approaches have risen in popularity over the last two decades in the field of natural language processing (NLP); in many areas corpus-based approaches equal in performance, or rival traditional rule-based methods. With the increase in computational power and the advancement and ease of use of machine-learning (ML) techniques, it is no wonder that corpus-based approaches continue to gain in popularity. In the late 1990s a sizable textbook[5)] was published attesting to their merits, and even a decade before, research had appeared pursuing such techniques[1)]. Corpus-based approaches also allow for more language and domain independence within a model, important in today's multilingual world.

The advent of corpus-based approaches also meant that the creation of corpora was required — corpus-based approaches are obviously impossible without them. The individual creation of custom tools for annotation tasks is a tremendous investment of time and labor, which when viewed in a wholistic manner shows how repetitive and wasteful such an investment can be. These custom fitted tools do of course address the demands of the project for which they were born, but also because of this tend towards inflexibility and disposability. Regardless of the problems of interoperability that arise when different tools are used with different data formats (something desirable when trying to cross-test methods with different pre-existing data-sets), a cycle forms of creating new tools for trivial tasks, or perhaps even worse conforming and compromising annotation criteria to fit within the limits of an existing tool made for a previous project, which impinges on the quality of the resource. Corpus-based methods will only grow in scale and complexity, and so it is crucial the annotation tool does not stand in the way.[⋆1]

The bottom line is that corpus creation (and also management) is time consuming enough without having to spend more resources in battling the development of a custom tool. In addition, previous studies have shown that the early phases of a project are the most volatile[6)], resulting in rapid changes to the annotation schema. This can also cause delays and cost additional resources if the annotation tool is too rigid or otherwise unable to adapt to the changes.

A survey[3)] proposed seven categories desirable for any annotation tool: diversity of data, multi-

⋆1 In addition, as data sets continue to grow in size and methods mature, it will be necessary to compare them, meaning a standardized format will also be necessary.

level annotation, diversity of annotation, simplicity, customizability, quality assurance and convertibility. The goal of these categories is to remove the obstacles outlined as problems above. They are well thought out, but there is one caveat to them: they are document-centric, or rather, they simply do not address the bigger scope of managing the annotation process. Furthermore, though they do raise concerns about areas that need to be addressed, they do not propose how these areas are to be tackled. In simple corpora this may not be much of a concern, but in larger projects – and as corpus-based techniques continue to grow and advance so do the sizes of the corpora[2] – it becomes a serious issue.

In order to create a tool suitable for use in the future, it is important to pinpoint the needs of an annotation project. In Section 2 we outline what we think these needs are, and then in Section 3 propose a framework for addressing them. We next introduce Slate (Segment and Link-based Annotation Tool *Enhanced*), the second iteration of a web-based annotation tool under development that utilizes this framework, with a real world example. We then conclude the paper and discuss future work.

2. Annotation Needs

Creating corpora is becoming a serious endeavor; it is an ordeal entirely separate from, and from some viewpoints secondary to, the technique that will utilize a given language resource. Such a trend will only worsen as techniques, and therefore the corpora they use, continue to grow in scale and complexity.

Thus it is not only that the annotation system must be flexible enough to accommodate a wide variety of annotation tasks, but that it must also provide for a means to *manage* the tasks themselves. Otherwise, the creation of the resources will hinder the development of the technique, the reason behind why they were made.

Let us concretize the definitions of terms important for defining annotation needs: annotation tasks, annotation projects, and annotation systems. An annotation task has a specific goal in mind, such as annotating all predicate-argument dependencies in a set of raw resources, or identifying all coreference-chains in a collection of news articles, etc. An annotation task is more generally a *type* of annotation work to be done, whereas an annotation project, is an *instance* of that work. For example, if we wish to perform two tasks on the same dataset, such as predicate-argument dependencies and then coreference-chains, we should consider these as two separate annotation projects, as well. An annotation system is software that allows a user to create annotation projects for annotation tasks.

Let us then specify what is needed from an annotation system. Dipper *et al.* proposed seven categories for what is needed from an annotation tool[3], but they operate mostly at the document-level. Our list of needs below focuses instead at a more macro, annotation project management level. We therefore skip needs related only to the act of annotation (such as appearance), though it is covered by our framework introduced in Section 3.

(1) **User and role management**. Annotation projects are too complex to have a single user type with no roles assigned to them. The system must distinguish between annotators and administrators in order to prevent annotators from unwittingly altering a key part of the project they should not have access to in the first place.

(2) **Delegation and monitoring of work**. The system must allow for an administrator to assign/reassign work to annotators, and to monitor their progress. It is important to forestall an annotator falling behind due to difficulties or other factors as it may also delay the completion of the corpus if the work is not reassigned, or the obstacle stopping the annotator not resolved.

(3) **Adaptability to new annotation tasks**. The system must be flexible enough to easily accommodate a new annotation task. If administrators cannot easily create a new project, and define the annotation requirements then the system will not be useful to them.

(4) **Adaptability within the current annotation task**. During the lifespan of any given project, it often meets with many changes, especially during the initial phases of a project. It is crucial that the system allow for the adjustment of annotation guidelines in such a way as to (1) facilitate the correction of any already annotated items by identifying those resources, and (2) to have the flexibility to accommodate the changes themselves.

(5) **Diffing and merging**. Creating a corpus often entails the *diffing and merging* of data from multiple annotators on a single resource to create a gold standard. Annotator agreement is an important statistic in general for corpora, but in order to have an end product, it may be important to resolve any differences between different annotators. The system must allow an administrator to diff all annotated resources with multiple sets of annotations, and then to merge any differences that are found. In the event that multiple sets of annotations on the same resource are desired however, the system should also not stand in the way.

(6) **Versioning of corpora**. A corpus is a product, an end result. But just as any other product goes through life cycles, so too may the corpus. In other words, after all work has been completed (and the gold standard created), there must be a way to package the result and label it uniquely for use. After a release there are often fixes, amendments, etc. and these changes must be tracked so that an additional version can be released. Without management of versioning, the "current state" of the annotation project is all that is known; for large projects, especially, it is important to identity milestones or to "tag" a given state so that one may be able to go back to it later. This way changes made after the "tagging" will not unknowingly be included into a release.

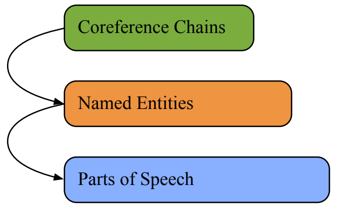


Fig. 1 Higher layers access lower layers as a base for more stringent annotation

(7) **Extensibility in terms of layering**. As corpora are continuing to grow in size, it is no wonder that they are also fundamentally becoming more complex. We have seen recently the stacking, or layering, of corpora upon one another. Attempts at diversifying a single corpus with various types of annotations are on the rise, such as the Discourse TreeBank on top of the Penn TreeBank[7)], or the NAIST Text Corpus atop the Kyoto Text Corpus[4)]. The system must allow for adding new layers upon previous ones, seamlessly. Often the lower layers are directly referenced by the new ones. It must allow for this as well *without* jeopardizing the quality of the lower levels. This idea is shown in Figure 1, where if we consider each level being a separate project, the one above directly references annotations in the preceding layer.

(8) **Extensibility in terms of tools**. Many annotation tasks these days involve (semi-)automatic tagging, followed by an annotator reviewing and correcting any mistakes in the output. The system should ideally allow the annotator to call plugins to the system to tag a given resource to facilitate this. This step could be done prior to data import, but allowing the user to do so afterwards has several benefits, such as enabling them to quickly view the results of the automatic tagging, and to rerun it with different parameters again should it not have provided the expected output. The plugins should be able to be created by anyone (not only the creators of the system).

(9) **Extensibility in terms of importing/exporting**. The system should also allow the user to either define rules or call plugins to convert the data from one format to another. This allows the system to be somewhat agnostic to the data format. This is important as there are a variety of formats, though the system should provide its own that is capable of handling any supported annotation scheme. Including generated comments may facilitate in human verification of the exported resource. This step could be done after export, but having the ability to have the data processed automatically increases productivity for menial tasks that should not require human intervention (and which may also introduce error).

(10) **Support for multiple languages**. Research today is carried out in a variety of languages, and the system should support them.

3. Framework Overview

The ten items listed above can be subdivided into those that require a framework for support, and those that rely on an implementation of a given framework. Without a proper framework in place, (1)–(7) are impossible; (8)–(10) reside entirely with the implementation. We address needs (1) & (2) (User & Project Management, and work delegation) in Section 3.1, (3) & (4) (Adaptability to current and future annotation tasks) in Sections 3.3 and 3.4. Needs (5) & (6) require the necessary data within a framework, but also rely heavily on the implementation to supply such features; they appear as the topic is relevant, below. Figure 2 shows a simplified representation in UML of some of the more major entities of the proposed framework. It may be beneficial to refer to it while reading through the explanation.

3.1 User & Project Management

A large part of proper project management involves proper user management (needs (1) & (2)). It is important to encapsulate user responsibilities in a way that facilitates productivity, rather than hinders it due to complexity. User roles are nothing new to software systems, but choosing the right granularity can be tough. We think it is best to keep things as simple as possible while maintaining the needed functionality; thus, we propose having minimally two types of users: administrators and annotators. (Note that user roles are not present in Figure 2.)

An administrator oversees projects, configures them, adds/removes annotators, and assigns annotation tasks. The administrator should also be able to check the progress of the annotators to make sure no roadblocks are preventing them from finishing their assigned work, or if they become incapacitated for some reason, allow the administrator to easily reassign outstanding work. As mentioned in Section 2, the administrator should also be able to create versions of the project (need (6)), import, export, and in all other ways administer the project. They should not, however, be able to annotate resources.

The annotator, therefore, is solely responsible for this task of annotation. They can select a resource that has been assigned to them and annotate it; they are sandboxed from other users to prevent biases from interfering with their work (meaning they do not see others' work).

This separation between administrator and annotator roles is simple, but it is intuitive; it also provides the necessary separation to properly manage a project and still get the annotation work done. Later on an administrator can diff the resources and merge them as necessary for creating a gold standard. More details about project configuration are explained as the remaining entities and annotation methodology are elaborated on below.

3.2 Entities

As Figure 2 shows, there are a number of entities interconnected to one another. In this section we will focus on the more macro-level entities, Project, Document, Document Set, and then touch upon Tagsets, which are explained more in the next section. The framework is based around the concept of a *document*, which is the only entity that can be directly annotated (shown in the upper right corner of Figure 2). It can represent any kind of data you wish (e.g. a news article, paragraph, book). Since dealing directly with large quantities of documents can be daunting, they are grouped into a more macro-level *document set*. A document can be a member of multiple document sets, so a document set could be created to encapsulate both raw resources from different sources, *and* groups of work to delegate to annotators. As the UML figure shows, annotators are assigned document sets, not documents; again, this is to make coordinating large volumes of documents easy, while keeping them coherently managed to some degree.

The highest level entity in the framework is the *project*. A project represents an annotation task, such as predicate-argument dependency or coreference-chains, and contains a reference to one or more document sets. An important point here is that a project contains a *reference to* one or more

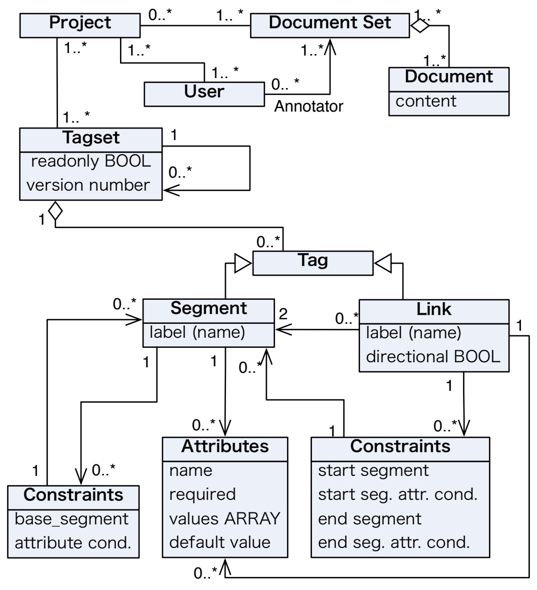


Fig. 2 Simplified relationship diagram for entities

document sets; the sets themselves exist beyond, or outside of, the project. This means that they can (and should) be reused for multiple projects.

More formally, we define a *user* as an annotator or administrator with access to the system, belonging to one or more projects (a user may be working on multiple tasks, concurrently, or in sequence); an annotator will have access to zero or more document sets within any given project, while the administrator can do all the actions outlined in Section 3.1. Since the point of task delegation is to split the work among many individuals, the framework allows for this by enabling the administrator to assign different document sets to different users.

The majority of the UML diagram, however, focuses on *tagsets* and their relations. This is the fundamental mechanism behind generalized annotation in the framework, and is explained in the next section.

3.3 Abstracting Annotation

So far we have outlined several of the main entities that make up the framework. But the crux

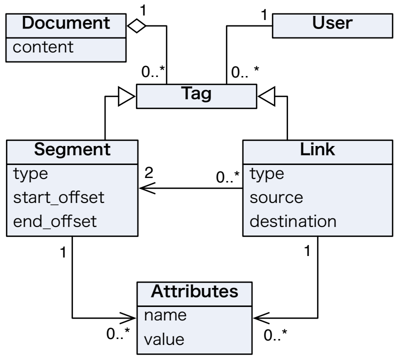


Fig. 3 Simplified relationship diagram for annotation instances

of being able to support multiple annotation tasks (needs (3) & (4)) is the notion of abstracting annotation. Abstracting annotation means that we remove any definitions for what types of entities can be annotated from the system, and instead create a framework that allows administrators to define them themselves. Conceptually, all an annotation is, is either a label placed on a span of text, or a label placed on a relationship between such spans, as is shown in Figure 4. So we allow the administrator to create as many *definitions* for types of annotations as they like, and then during annotation, the annotator can create *instances* of these admin-defined types, shown in Figures 2 and 3, respectively. It is important to understand there is a difference between a definition of a type, and an instance of that type. Also note that the framework itself is agnostic to the definitions; to the system it sees only different types of segments and links.

The annotations on text-spans we call *segments*, and the relationships between them, *links*[8),9)]. Links may be directional, such as in the predicate-argument dependency example shown in Figure 4, or undirected, such as when annotating coreference-chains. As Figure 2 shows, segments and links both extend from an abstract *tag*. For the remainder of this paper, if segment or link is not specified, then the explanation applies to both.

However, simple labels and their relations are limiting; we might want to store additional information both about segments, and about any links that connect them. For this, *attributes* are necessary. Say we wish to supplement a segment type definition for "Noun" with information about its plurality; for this we could create an attribute with the possible values of "singular" and

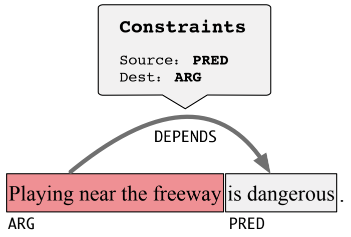


Fig. 4 An example showing two segments, and a link between them

"plural". We can then formalize the definition of a tag T to be a label $label$ and a set of attributes a_i composed of a name $name$ and a value type val_type, along with a possible default value def_val, and a flag req indicating whether or not the attribute is required:

$$T = (label,\ \{a_0, a_1, \ldots, a_{n-1}\}),$$
$$\text{where } a_i = (name, val\_type, def\_val, req).$$

The value type could be an enumeration, all possible character strings (i.e. free input), number ranges, etc. We can then group these segment and link type definitions into a *tagset*, and assign the tagset to a project. Projects may need to contain multiple tagsets, depending on the task. In the case that the project is extending another previous project (need (7)), it will be important to disable creation of instances from that tagset, and instead only allow the annotator to view them (reflected in Figure 2).

3.4 Tagset Management

Especially during the early phases of a project, the annotation specification is victim to changes in design (need (4)). By creating versions/revisions of tagsets, we can enable the system to keep track of what resources are annotated with which version of each type; allowing an administrator to verify that all documents are annotated with the current spec. Examples include an administrator adding an attribute to verbs, or adding a value to an existing attribute, etc. In the framework all instances of annotations are marked with the version/revision of the tagset that was used when they were annotated. It may also be beneficial to denote the difference between changes that complement a current version (e.g. accommodating a new case found during annotation that does not affect other annotations), i.e. revisions, and changes that require reannotating resources (versions).

4. A Real World Example

In this section we introduce Slate, a web-based annotation tool that implements the framework explained above, with a real world example. Though Slate is not yet available to the general public, we have been collaborating with several laboratories to tune Slate to real scenarios. Here we briefly introduce the software using data provided by one such collaborator. The example here illustrates annotating errors within non-native Japanese speakers' prose. A deep analysis of such annotation is beyond the scope of this paper, but we here briefly outline what this task involves.

First, a corpus of non-native Japanese speakers' prose is necessary. Annotators then annotate sentences by marking textspans were errors result, such as using the wrong tense of a verb, or the wrong particle ('*ga*' for '*wa*', and so on). The annotators further correct the span to what the proper usage is conceived to be, and further classify the error into one or more of several predfined categories, such as grammatical, tense, honorifics, etc. Since errors may be noncontiguous but intrinsically related, it is also desirable to be able to link these textspans together, such as the use of '*mettani*' with '*arimasu*', instead of '*arimasen*', which is the correct usage.

Refer to the following basic workflow when following along with the example below.

Project Manager does

(1) Define a project

(2) Upload documents for annotation

(3) Create annotators / assign work

Annotator does

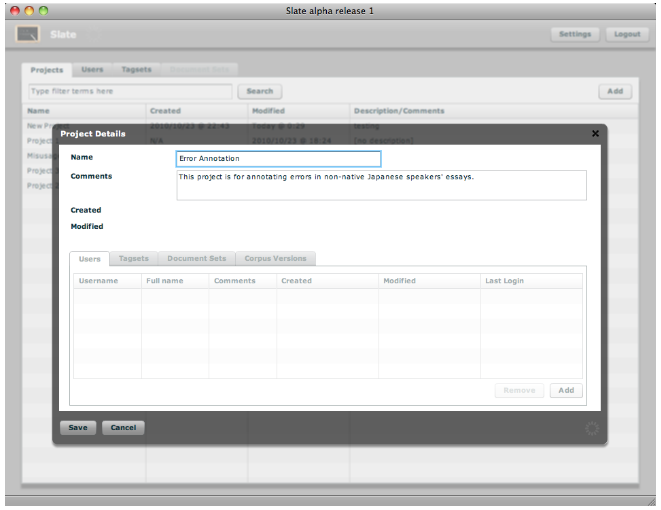
(1) Select an assigned document

(2) Begin / continue annotation

4.1 Defining a project and its tagset(s)

The first step is to define a project for the error annotation task. Figure 5 shows the project definition screen. Once this has been done, we define the tagset or tagsets we need for the task. In this case, a single tagset will suffice, but more than one is equally feasible. Figure 6 shows the tagset definition screen. Here we create a segment that will be used to annotate non-native speakers written errors. We then must create the attributes needed for this segment, as shown in Figure 7. In this case we create attributes for the correction (how the sentence should read), which we make freeform, for any additional comments the annotator may wish to make, also as freeform, and finally for the error type, which we want to make a predefined list annotators can easily select from at the time of annotation.

The segment after the creation of these attributes is shown in Figure 8. We then create a link that allows us to mark two or more segments as being part of the same error semantically, or lexically, as shown in Figure 9. It is also possible at this point to define the default colors (i.e. appearance) of the segment and link as it will be displayed to the annotator. This can be seen in Figures 8 and 9. The link does not need any attributes in this example.



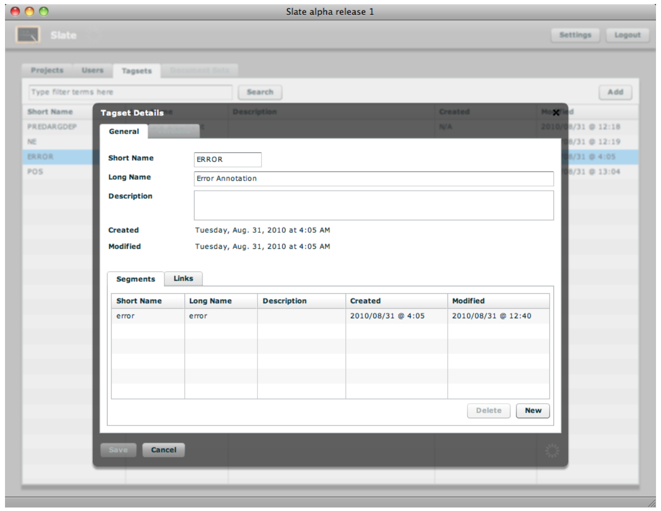
Name: Error Annotation
Comments: This project is for annotating errors in non-native Japanese speakers' essays.
Created
Modified

Username	Full name	Comments	Created	Modified	Last Login

Fig. 5 Creating a new project

4.2 Importing data into a project

The next step is to import the data so that it can be annotated. The data is either entirely new and unannotated, or as is more likely the case, already annotated with some data from some other task in the past. In the latter case, the data must be reformatted into the proper format; a future feature will allow a conversion plugin to do the work at import time as well. Figure 10 shows the import screen for adding new data to an existing project.

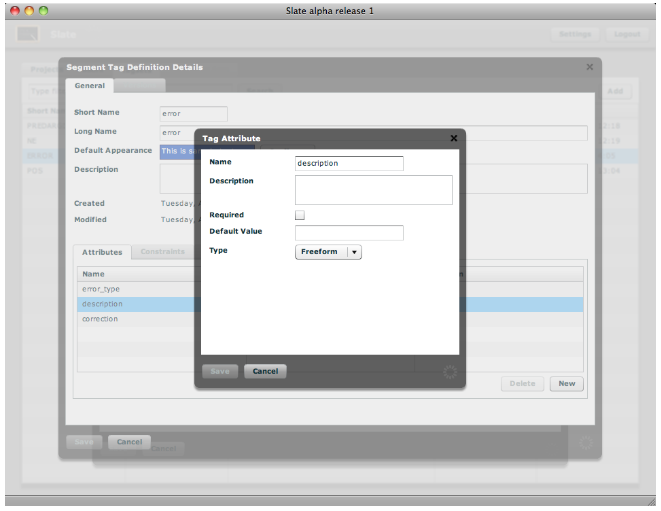


Short Name
PREDARGDEP
NE
ERROR
POS

Short Name	ERROR
Long Name	Error Annotation
Description	
Created	Tuesday, Aug. 31, 2010 at 4:05 AM
Modified	Tuesday, Aug. 31, 2010 at 4:05 AM

Short Name	Long Name	Description	Created	Modified
error	error		2010/08/31 @ 4:05	2010/08/31 @ 12:40

Fig. 6 Creating a new tagset

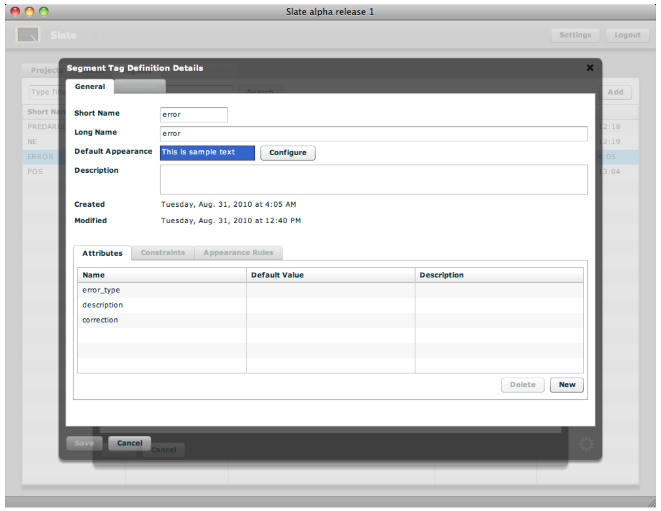


Short Name	error
Long Name	error
Default Appearance	This is sa
Description	
Created	Tuesday,
Modified	Tuesday,

Name
error_type
description
correction

Name	description
Description	
Required	☐
Default Value	
Type	Freeform

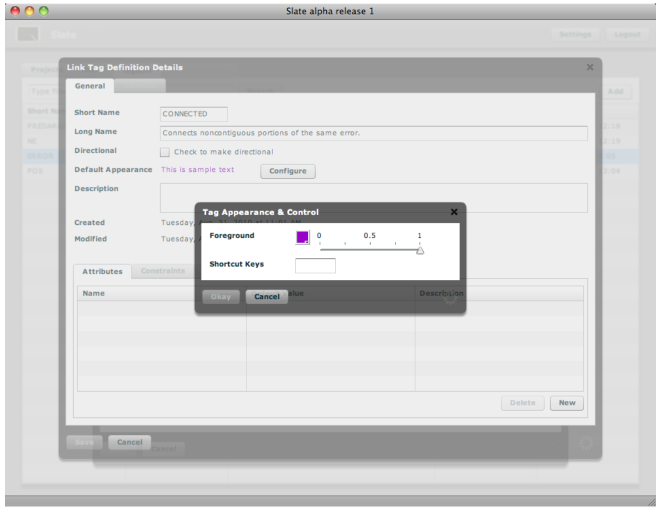
Fig. 7 Adding an attribute to a segment definition



Short Name	error
Long Name	error
Default Appearance	This is sample text
Description	
Created	Tuesday, Aug. 31, 2010 at 4:05 AM
Modified	Tuesday, Aug. 31, 2010 at 12:40 PM

Name	Default Value	Description
error_type		
description		
correction		

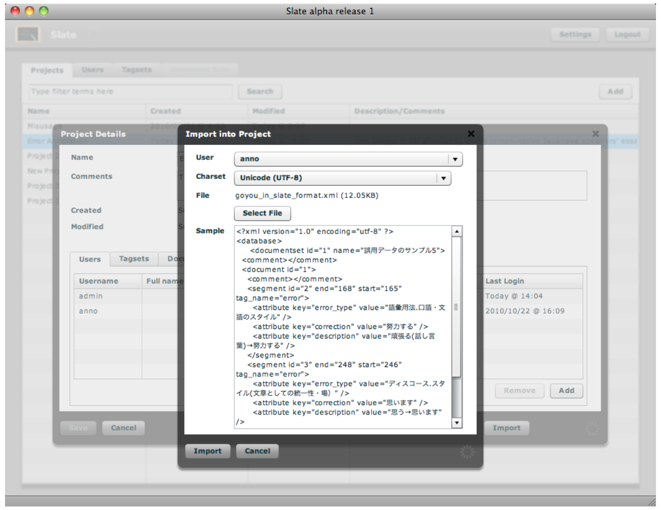
Fig. 8 Segment definition after adding attributes



Short Name	CONNECTED
Long Name	Connects noncontiguous portions of the same error.
Directional	☐ Check to make directional
Default Appearance	This is sample text
Description	
Created	Tuesday,
Modified	Tuesday,

Foreground	■
Shortcut Keys	

Fig. 9 Configuring link definition appearance

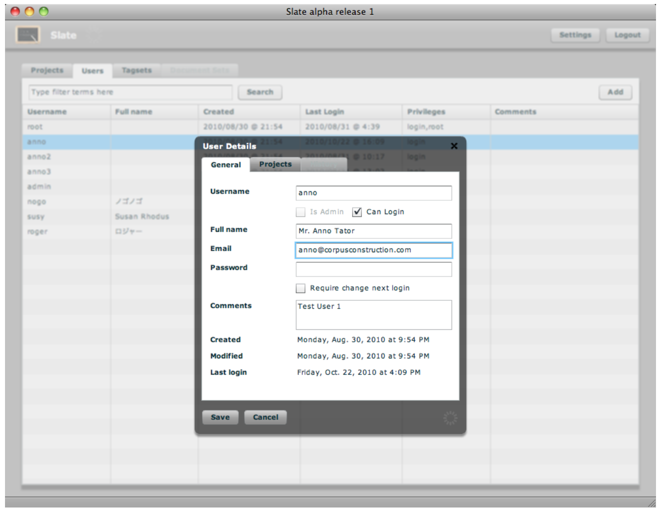


User	anno
Charset	Unicode (UTF-8)
File	goyou_in_slate_format.xml (12.05KB)

Sample

```
<?xml version="1.0" encoding="utf-8" ?>
<database>
    <documentset id="1" name="誤用データのサンプル5">
  <comment></comment>
  <document id="1">
   <comment></comment>
   <segment id="2" end="168" start="165"
tag_name="error">
     <attribute key="error_type" value="語彙用法.口語・文
語のスタイル" />
     <attribute key="correction" value="努力する" />
     <attribute key="description" value="頑張る(話し言
葉)→努力する" />
   </segment>
   <segment id="3" end="248" start="246"
tag_name="error">
     <attribute key="error_type" value="ディスコース.スタ
イル(文章としての統一性・場) " />
     <attribute key="correction" value="思います" />
     <attribute key="description" value="思う→思います"
/>
```

Fig. 10 Importing data into a project



Username	anno
	Is Admin [✓ Can Login]
Full name	Mr. Anno Tator
Email	anno@corpusconstruction.com
Password	
	Require change next login
Comments	Test User 1
Created	Monday, Aug. 30, 2010 at 9:54 PM
Modified	Monday, Aug. 30, 2010 at 9:54 PM
Last login	Friday, Oct. 22, 2010 at 4:09 PM

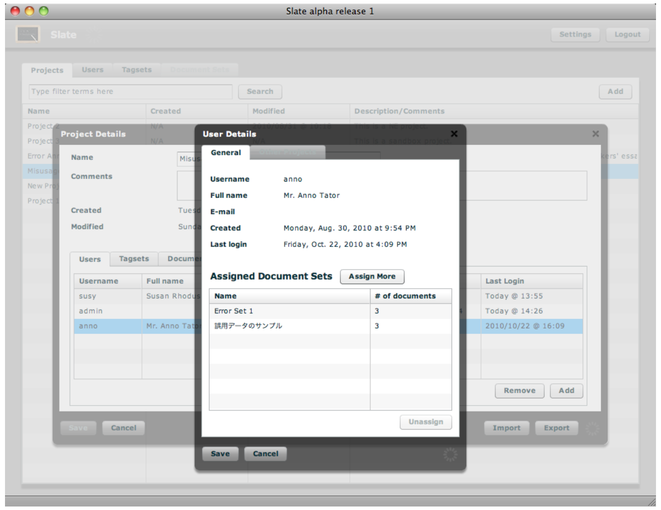
Fig. 11 Creating a new annotator

4.3 Creating and assigning work to annotators

After data has been imported, it must be assigned to annotators so they can perform the work. This is accomplished by assigning document sets, composed of documents as explained in Section 3.2, to annotators. Figure 11 shows the screen for creating a new annotator account, and Figure 12 shows the screen for assigning one or more document sets to a given annotator.

4.4 Annotating documents

Once an annotator has been assigned one or more document sets, they can log in and begin the work. Figure 13 shows the screen an annotator is presented with upon logging in. Here they can select a document from those that they have been assigned to begin/continue work. Information about the selected project, document set, and document appear to the right. Selecting a document results in the main annotation screen, shown in Figure 14. The left panel is the main annotation panel; the right side provides a set of tool panels that show document information, the available tagsets for annotation, and a list of segments/links that exist within the current document. Here the annotator annotates the text by finding errors, and then clicking and dragging the mouse, the same as selecting text in almost any text editor, and then letting go of the mouse button. After they



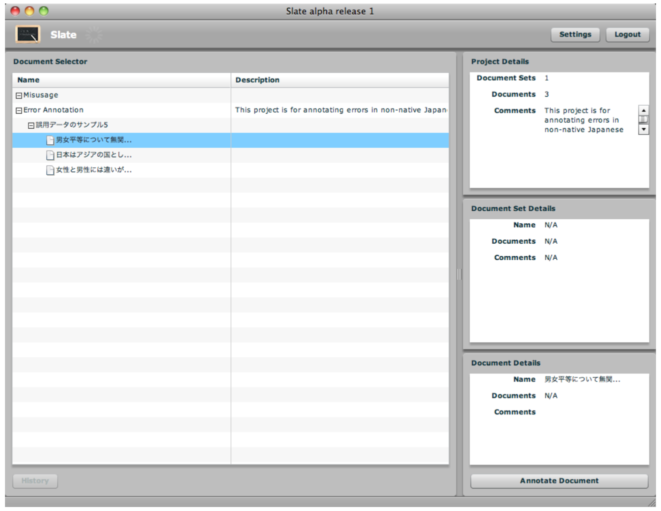
Username	anno
Full name	Mr. Anno Tator
E-mail	
Created	Monday, Aug. 30, 2010 at 9:54 PM
Last login	Friday, Oct. 22, 2010 at 4:09 PM

Assigned Document Sets

Name	# of documents
Error Set 1	3
誤用データのサンプル	3

Fig. 12 Asssigning work to an annotator

have defined the textspan, they can click on the annotation to set attribute values, in this case the corrected text, the error type, and any additional comments.



Document Selector

Name	Description
⊟Misusage	
⊟Error Annotation	This project is for annotating errors in non-native Japan
⊟誤用データのサンプル5	
男女平等について無関...	
日本はアジアの国とし...	
女性と男性には違いが...	

Project Details

Document Sets	1
Documents	3
Comments	This project is for annotating errors in non-native Japanese

Document Set Details

Name	N/A
Documents	N/A
Comments	N/A

Document Details

Name	男女平等について無関...
Documents	N/A
Comments	

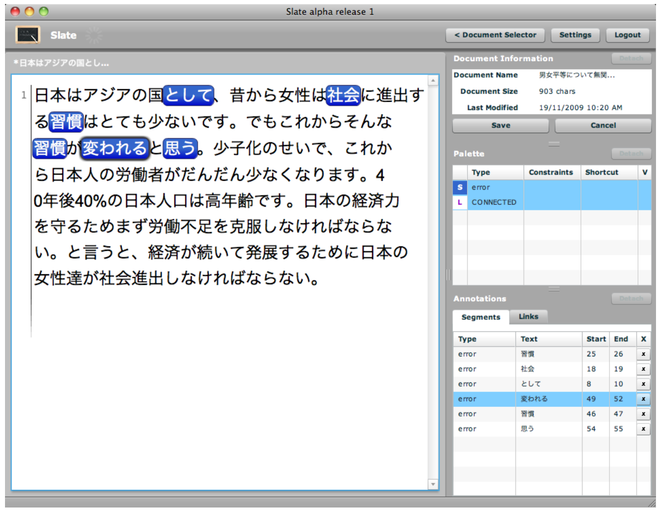
Fig. 13 Annotator home screen

4.5 Exporting annotated data

Once all the annotation work has been completed, it is necessary to export the data so that it can be utilized – the point of creating resources is in their use, not in their creation. In a future release, Slate will support exporting in any format for which a plugin exists.

5. Conclusion

This paper presented a list of 10 needs for annotation systems in order to support corpus-based NLP research moving into the future. The needs fall into two groups: framework-level (needs (1)–(7)), and implementation-level ((8)–(10)). A framework was introduced to tackle the framework-level needs, and a system Slate built on top of the framework, to handle the implementation-level needs. At its core, the framework provides a mechanism for abstracting annotation and the creation of layered resources through its use of tagsets, containing segments and links with attributes



*日本はアジアの国とし...

日本はアジアの国として、昔から女性は社会に進出する習慣はとても少ないです。でもこれからそんな習慣が変われると思う。少子化のせいで、これから日本人の労働者がだんだん少なくなります。40年後40%の日本人口は高年齢です。日本の経済力を守るためまず労働不足を克服しなければならない。と言うと、経済が続いて発展するために日本の女性達が社会進出しなければならない。

Document Information

Document Name	男女平等について無関...
Document Size	903 chars
Last Modified	19/11/2009 10:20 AM

Palette

	Type	Constraints	Shortcut	V
S	error			
L	CONNECTED			

Annotations

Type	Text	Start	End	X
error	習慣	25	26	x
error	社会	18	19	x
error	として	8	10	x
error	変われる	49	52	x
error	習慣	46	47	x
error	思う	54	55	x

Fig. 14 Annotation screen

as well as constraints. The framework further provides a foundation for managing annotation resources and multiple annotation tasks. The framework allows an administrator to define a task and then monitor its progress, letting the annotator do what they were intended to do: annotate. Appearance settings allow the annotator to quickly know what types of segments/links they are working with, and the constraints reduce negligent annotation errors (such as not allowing predicate-argument dependency to join two arguments), though the appearance alone helps with this. Future work can be divided into two categories: framework-level, and implementation-level. For the former, it may be beneficial to more concretely specify how corpus versioning should be carried out. For the latter, a number of features are still left unimplemented. We plan to release an alpha version soon.

Acknowledgments

This work is partially supported by the Grant-in-Aid for Scientific Research Priority Area Program "Japanese Corpus" (2006–2010), sponsored by MEXT (Ministry of Education, Culture, Sports, Science and Technology–Japan).

References

1) Charniak, E.: *Statistical Language Learning*, The MIT Press (1993).
2) Davies, M.: The 385+ million word Corpus of Contemporary American English (1990–2008+) Design, architecture, and linguistic insights, *International Journal of Corpus Linguistics*, Vol.14, No.2, pp.159–190 (2009).
3) Dipper, S., Götze, M. and Stede, M.: Simple Annotation Tools for Complex Annotation Tasks: An Evaluation., *Proceedings of the LREC Workshop on XML-based Richly Annotated Corpora*, pp.54–62 (2004).
4) Iida, R., Komachi, M., Inui, K. and Matsumoto, Y.: Annotating a Japanese Text Corpus with Predicate-Argument and Coreference Relations, *Proceedings of the Linguistic Annotation Workshop*, pp.132–139 (2007).
5) Manning, C.D. and Schuetze, H.: *Foundations of Statistical Natural Language Processing*, The MIT Press (1999).
6) Marcus, M.P., Santorini, B. and Marcinkiewicz, M.A.: Building a Large Annotated Corpus of English: The Penn Treebank, *Computational Linguistics*, Vol.19, No.2, pp.313–330 (1993).
7) Miltsakaki, E., Prasad, R., Joshi, A. and Webber, B.: The Penn Discourse Treebank, *Proceedings of 4th International Conference on Language Resources and Evaluation (LREC 2004)*, pp.2237–2240 (2004).
8) Noguchi, M., Miyoshi, K., Tokunaga, T., Iida, R., Komachi, M. and Inui, K.: Multiple purpose annotation using SLAT – Segment and link-based annotation tool, *Proceedings of 2nd Linguistic Annotation Workshop*, pp.61–64 (2008).
9) Takahashi, T. and Inui, K.: A multi-purpose corpus annotation tool: Tagrin, *Proceedings of the 12th Annual Conference on Natural Language Processing*, pp.228–231 (2006).