CCASH: A Web Application Framework for Efficient, Distributed Language Resource Development

Paul Felt, Owen Merkling, Marc Carmen, Eric Ringger,
Warren Lemmon, Kevin Seppi, Robbie Haertel
Department of Computer Science
Brigham Young University
Provo, Utah 84602 USA
E-mail: pablofelt@gmail.com, omerkling@gmail.com, marc.carmen@gmail.com, ringger@cs.byu.edu,
lemmon.warren@gmail.com, kseppi@byu.edu, robbie_haertel@byu.edu

Abstract

We introduce CCASH (Cost-Conscious Annotation Supervised by Humans), an extensible web application framework for cost-efficient annotation. CCASH provides a framework in which cost-efficient annotation methods such as Active Learning can be explored via user studies and afterwards applied to large annotation projects. CCASH's architecture is described as well as the technologies that it is built on. CCASH allows custom annotation tasks to be built from a growing set of useful annotation widgets. It also allows annotation methods (such as AL) to be implemented in any language. Being a web application framework, CCASH offers secure centralized data and annotation storage and facilitates collaboration among multiple annotations. By default it records timing information about each annotation and provides facilities for recording custom statistics. The CCASH framework has been used to evaluate a novel annotation strategy presented in a concurrently published paper, and will be used in the future to annotate a large Syriac corpus.

1. Introduction

The current success and widespread use of data-driven techniques in language-related fields make annotated corpora an often essential language resource. For instance, many popular Natural Language Processing (NLP) algorithms require significant amounts of human-annotated training data in order to perform effectively. Also, annotated text can be useful in its own right as a means of qualitatively exploring the annotated text. For example, one might use part-of-speech (POS) annotations to study the syntax of a language, or morphological annotations to study the formation of words in a morphologically rich language.

Along with the need for annotated corpora comes the need for tools capable of creating these corpora. However, the process of creating annotated corpora is not trivial. For one thing, employing human specialists to annotate each instance in a corpus by hand can be prohibitively costly. A general purpose annotation tool should make use of existing cost-efficient annotation methods such as automatic annotation and Active Learning (see Section 2). However, cost-efficient annotation is an area of active research, so annotation tools should also be sufficiently flexible to encourage novel methods to be implemented and explored. Indeed, since the effectiveness of various annotation methods may vary across tasks and domains, even projects interested only in applying known annotation methods to a large corpus may wish to conduct exploratory studies to compare the efficiency of several annotation methods before proceeding on a large scale. In addition to cost, many other problems must be dealt with. If the annotation task being conducted is uncommon, project developers may need to customize an existing annotation tool or create their own custom tool to implement that annotation task. Annotation projects that employ multiple annotators must solve problems of data distribution and consistency. Such projects must somehow distribute views of the corpus to each annotator and collect annotations into a central location, handling any conflicts among the annotations.

Although this discussion by no means exhausts the demands that might be made of a general-purpose annotation tool, we believe they are an important subset. Ideally then, an annotation tool would offer at a minimum the following high-level features:

- Accommodate proven cost-efficient annotation methods
- Encourage novel cost-efficient annotation methods
- Facilitate exploratory studies and comparisons of annotation methods (e.g. measure annotation costs)
- Accommodate custom annotation tasks
- Coordinate the efforts of multiple annotators

In this paper we introduce CCASH (Cost-Conscious Annotation Supervised by Humans), a web application framework for corpus annotation designed to implement this feature set by using familiar programming paradigms, open standards technologies, and by providing reasonable default implementations whenever possible, always allowing those with unique requirements to define their own features from the ground up.

The remainder of this paper is organized as follows: in Section 2 we describe annotation projects, studies, and tools that influenced CCASH's design and implementation. In Section 3 we explain our decision to implement CCASH as a web application. In Sections 4 and 5 we describe CCASH's architecture and

implementation details. In Section 6 we describe the process of customizing CCASH. In Section 7 we outline a case study in which CCASH was used, and in Section 8 we discuss conclusions and future work.

2. Related Work

Here we present previous work that helped to motivate the feature set outlined in Section 1 and to inform the way that CCASH implements those goals. Due to the importance of cost efficiency to those goals, a large portion of the work we cite consists of annotation projects, studies, and tools that were used to develop cost-efficient methods of annotation.

Automatic annotation, or pre-labeling, consists of using NLP algorithms to automatically annotate each instance before it is presented to an expert annotator. Expert annotators then need only review and correct the proposed annotations, which can be much quicker than annotating from scratch. Marcus et al. (1994) evaluated automatic annotation using an interface embedded in the GNU Emacs Editor to annotate the Penn Treebank. They manually timed four annotators and found that automatic annotation more than doubled annotation speed and also increased accuracy and inter-annotator agreement. Chiou et al. (2001) manually timed two annotators and reported a 70% increase in annotation speed using automatic annotation on a Chinese Treebank annotation task. They did not report the tool they used. Ganchev et al. (2007) used a custom web-based tool to do named entity recognition (NER). They evaluated an automatic annotator that presented annotators with a set of plausible guesses instead of a single best guess. They manually recorded the time of a single annotator, reporting a more than 50% increase in speed compared with a manual baseline.

The dramatic time savings reported in these studies underscore the importance of providing proven annotation methods in any general-use annotation framework. Also, notice that each study evaluates automatic annotation by manually timing a very small number of annotators. These results are convincing, but relatively informal. This suggests a need for annotation tools that automatically record cost in such a way as to facilitate exploratory studies, allowing significant user studies to be run without much overhead. Also, flexibility and customization were shown to be important to annotation tools. For example, Ganchev et al. (2007) found it necessary to tweak the simple concept of automatic annotation in order to make it successful in the domain of NER.

Many automatic annotators require annotated training data. Annotated data are cheaply available for common tasks in major languages. However, in order to apply automatic annotation to a new task or to a new language, expert annotators must be paid to annotate training data, reducing the cost-efficiency of automatic annotation.

Active Learning (AL) is a technique that addresses this problem by reducing the cost of annotating useful amounts of training data (Ringger et al. 2007; Haertel et al. 2008a; Haertel et al. 2008b; Settles 2009). AL controls which data instances an expert is asked to annotate, presenting them with instances likely to be most informative for learning algorithms. The resulting annotations may be used to train an automatic annotation algorithm.

Ngai and Yarowski (2000) evaluated the effectiveness of AL for noun phrase chunking using an hourly cost model. For this study, seven annotators used a custom-built Java annotation client communicating with a server to enable centralized AL and record timing information. Tomanek et al. (2007) evaluated the performance of AL on the task of NER in immunogenetics. They developed and used JANE (the Jena ANnotation Environment), a Java program built on MMAX2 (Müller and Strube, 2006), to record annotators' timing information. JANE uses a client-server architecture, allowing distributed annotation and multi-annotator AL.

Both of these studies deal with multiple annotators by centralizing their data and developing tools with client-server architectures. They also extend AL to the multi-annotator case, again underlining the variety of implementations possible for each established annotation method.

Ringger et al. (2008) conducted an AL study with 47 annotators doing English POS tagging using a custom-built web application that collected timing information. They used that information to derive an hourly cost model for English POS tagging, which Haertel et al. later incorporated into a cost-conscious version of AL (2008b). This is a case where cost measurements were not just used to provide evidence for the effectiveness of a particular method of annotation, but were actually incorporated into an annotation method. In other words, there are some cost-efficient annotation methods that cannot be implemented with an annotation tool that does not record and provide access to cost information, making real-time cost measurement essential.

Representative general-use annotation platforms that influenced CCASH's design include GATE (Cunningham 2002), Word-Freak (Morton & Lacivita 2003), MMAX2 (Müller & Strube 2006), Knowtator (Ogren 2006), and JANE (Tomanek et al. 2007). These tools all support common annotation tasks and also allow for the creation of custom annotation tasks with different degrees of flexibility. GATE is a Java tool that uses a client-server architecture to coordinate multiple annotators, allows timing information to be recorded, and uses a modular design to promote customization. Knowtator allows users to define complex annotation schemas, making it exceptionally configurable and reducing the need for customized plug-ins. MMAX2, like GATE, is a highly modular Java application with a client-server architecture. JANE is a Java application built on MMAX2 that, as mentioned before, provides a form of AL. Word-Freak supports both automatic annotation and searching based

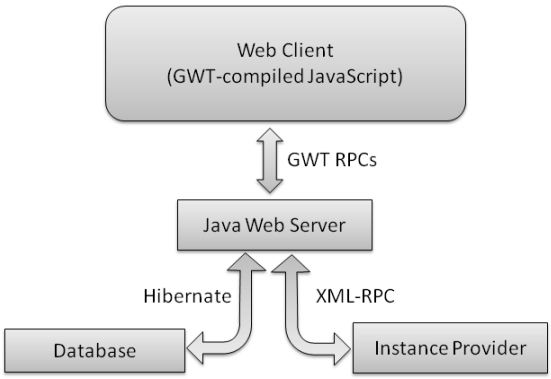

Figure 1: CCASH Architecture

on annotation confidence, which allows annotators to engage in a kind of manual AL.

3. Web Application Framework

Although the features outlined in Section 1 could be implemented in a variety of ways, CCASH designers felt that a web application framework was most fitting for a number of reasons. Previous annotation tools have tended toward client-server relationships in order to centralize data and facilitate multiple annotator collaboration. Web applications make client-server architecture easy and natural. Among the tools described in Section 2, GATE and MMAX2 seem to be the most popular due in large part to their support for extensive programmatic customization. A web application seemed a good choice for a customizable architecture, since Internet architecture has a tradition of being extremely customizable, even allowing modules written in different languages and running on different platforms to interoperate.

Being a web application gives CCASH other key advantages in a distributed annotation project. The overhead of configuring a collaborative annotation project can be handled by a single administrator with access to the server. Annotators can then immediately begin annotating texts from any GWT-supported web browser with virtually no per-user configuration time. Since web applications are reloaded every time a user revisits the site or refreshes the browser, there is no difficulty associated with distributing software or project configuration updates. Any updates to the annotation task or to the CCASH framework are instantly and transparently available to all annotators.

4. CCASH Architecture

CCASH's architecture consists of four parts: a web client, a web server, a database, and an instance provider (see Figure 1).

4.1 Instance Provider

Instance providers are processes with a single responsibility: to provide instances to annotators. In this context an instance is a piece of text, such as a word or sentence, which requires expert annotation. The instances that an instance provider returns may optionally be pre-annotated. Instance providers are largely independent from the rest of CCASH. They make themselves available as web services at some address by implementing a simple XML-RPC interface (see Section 5.3). In CCASH, part of setting up an annotation project is giving it the address of a valid instance provider. Because instance providers are decoupled across the network from the rest of CCASH, they may be implemented in any language. This is particularly valuable since instance providers are a prime target for making use of NLP algorithms such as pre-labeling and AL. Algorithm libraries and custom research tools exist in many languages besides Java, and may be reused as part of implementing an instance provider. Because of this network decoupling, instance providers may also be located at anywhere in the world, although because of network latency issues we anticipate that they will commonly be located either on the same machine as the web server, or nearby.

For convenience, CCASH provides Java instance providers that use generics to return any type of instance in sequential and random order. We are also working on including Java instance providers that implement several varieties of pre-labeling and AL.

4.2 Data Model

Deciding how to represent and store instances and annotations was a difficult design decision in CCASH. Ideally, one would invent a data structure that is both efficient and also able to encode every instance and annotation type that might be required. For example, a POS tagging task might require annotations to be a sequence of tags. Dependency parsing, on the other hand, might require annotations to contain sets of directed connections between word pairs in the corresponding instance. One can imagine that a data structure able to represent both of these annotations (not to mention a multitude of other possible annotation tasks) would run the risk of being bulky and cumbersome. However, if the data structure were not sufficiently general, it would lose the ability to represent certain tasks and the framework would be unusable for them. Also, if a data structure required users to radically alter their own data schemas in order to fit CCASH's structures, it might discourage them from using the framework.

Recall that one of our high-level design goals is to provide reasonable default implementations whenever possible, always allowing those with unique requirements to define their own functionality. Guided by this principle, we decided to provide some reasonable default data representations and separate the CCASH framework from task-specific data structures as much as possible, allowing developers to use their own data structures, if desired, with minimal interference from the framework.

The two parts of CCASH that need to work with task-specific instance and annotation structures are the web client and the instance provider. The web client must

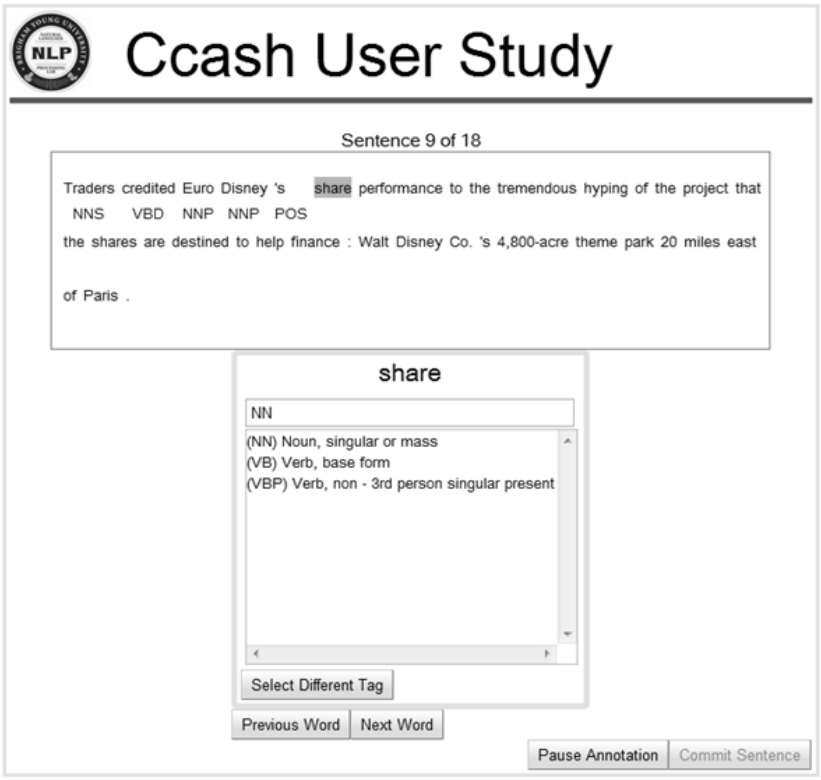

Figure 2: English POS Task in CCASH

know how to appropriately display the data instances and collect the desired annotations. The instance provider must select instances, possibly pre-label them, and then send them to the client. It receives new annotations from the client, updating its models with new annotations and recording the annotations alongside the data. The web client and the instance provider share a common method of serialization, and between those two endpoints, CCASH is ignorant of instance and annotation values. CCASH simply passes the serialized value along as a member variable of wrapper objects that CCASH uses to maintain records in its own database.

4.3 Web Client and Server

The web client is the portion of the application that runs in a user's browser using a combination of HTML and JavaScript. The CCASH client-side framework is written using the Google Web Toolkit (see Section 5.1), and we recommend that CCASH developers extending that framework or implementing new annotation tasks (see Section 6) do the same. While annotating, the web client's principle responsibility consists of requesting instances from the web server, displaying them to the user, and collecting annotations. The client then sends those annotations back to the web server.

The web server is in charge of facilitating client interactions with other components such as instance providers and the database. It passes on client requests for new instances to the appropriate instance provider and notifies the same instance provider when annotations are completed, giving the instance provider a chance to update its models given this new information.

The CCASH framework uses a combination of client-side interfaces and server-side storage to provide out-of-the-box user account management and project management. It also maintains a database containing information about each annotation (see Section 4.5), allowing access to project statistics.

4.4 Widget Libraries

In order to make new tasks as easy as possible to implement and customize, we implement default tasks by creating and assembling re-usable GWT widgets. For example, the English POS task in Figure 2 is a combination of a sequential annotation widget (allowing navigation over a sequence of instances), an instance annotation widget (highlighting the current instance in a box) and an English POS instance annotation widget which makes use of an auto-completion widget populated with the Penn Treebank tag set. The auto-completion widget allows users to type in any part of the tag or description, narrowing down selection options to entries that match any part of the selection.

Because CCASH is intended to be used for research as well as large-scale annotation projects, the framework includes widgets useful for building user studies. These currently include widgets for instructions, surveys, and tutorial annotations with feedback.

In addition to the widgets offered by CCASH, many

widgets come standard with GWT, and other third party GWT widget libraries are freely available. Because GWT can interface with native JavaScript, even third-party JavaScript libraries can be used with some additional overhead.

4.5 Evaluation

Previous work suggests a strong need for measuring the cost of each annotation in terms of time (Haertel et al. 2008a). This is not, however, the only possible measure of cost. Culotta et al. (2006), for example, measure cost in terms of the number of required user actions to fix an annotation in a given user interface. This is a reasonable surrogate for time, since more interactions generally mean more time, and it enjoys the benefit of being easy to predict. CCASH provides a flexible mechanism for measuring cost by collecting events fired by the web client into a simple sequence analogous to a timeline. Each timeline event has a name and a timestamp, allowing calculation of cumulative time, number of interactions, and other desired statistics. CCASH by default fires events when an annotation instance is requested, when it is presented to an annotator, when an annotation is completed, and when an annotation task is paused or resumed. If more granularity is required, for example if each user interaction needs to be recorded, CCASH developers implementing new tasks in CCASH can fire custom events at any point.

This cost information can be used to evaluate cost-reduction strategies *post hoc*. But it can also be used by an annotation method that learns from annotation costs. Haertel et al. (2008b) and Settles et al. (2008) have both proposed methods of incorporating cost models into the AL process, helping to offset traditional AL's bias towards long, costly instances.

5. Core Technologies

CCASH makes use of several supporting technologies. This section briefly describes what they are and how they are used.

5.1 Google Web Toolkit (GWT)

CCASH's web client component is implemented with the Google Web Toolkit (GWT). GWT allows developers to build user interfaces in Java using familiar Swing-like widgets. GWT provides a cross-compiler that compiles Java code into optimized JavaScript which communicates with a Java web server using remote procedure calls. GWT packages this entire bundle—JavaScript for the client and Java code for the server—into a Web Archive (WAR) which can be hosted on any compatible Java web server like Apache's Tomcat or Red Hat's jBoss.

We chose to use GWT to implement the web client portion of CCASH for several reasons. Most importantly, GWT helps user interface developers abstract away from the browser-specific idiosyncrasies that can make web programming difficult for newcomers. CCASH is designed with the assumption that future researchers who create new tasks for CCASH will likely be familiar with Java programming and at least one of the two major interface design paradigms that GWT supports: assembling Swing-like graphical components programmatically, or else defining XML interfaces bound to Java objects (similar to more traditional web-page design). GWT code compiles to JavaScript that is compatible with most major modern web browsers including IE, Firefox, Safari, and Opera. GWT also provides several mechanisms for creating localizable web applications. This helps CCASH support Unicode and right-to-left languages as well as locale-specific text and styles. Also, GWT facilitates using the browser history buttons to navigate through locations within a web application by encoding some application state in a history token embedded in the browser's address bar.

5.2 Hibernate

The Java Persistence API is a robust and standard way to manage permanent data storage in Java. We chose to use Hibernate to implement this API and to interface with the database layer of CCASH. This means that if developers find that they need to persist their own custom data objects, they can do so by simply annotating their data object classes in compliance with the Java Persistence API. It also means that framework users are free to use any of the many database implementations that are supported by Hibernate.

Using the Java Persistence API makes it easy to place the database either on the same machine that is running the web server or on any other machine that is network-accessible. Note that storing annotations in a database makes them efficiently accessible without precluding the possibility of exporting them to other formats such as XML.

5.3 XML-RPC

XML-RPC is a simple protocol for making remote procedure calls over the network. Because of the simplicity of the protocol, implementations exist in many programming languages including C, C++, C#, Java, Python, Ruby, Lisp, and many more. This makes it easy for the instance provider to be implemented in almost any language in order to reuse existing algorithms implementations or libraries for automatic annotation and AL.

6. Defining Custom Tasks

The process of adding a new annotation task to CCASH consists principally of creating a client-side user interface for the task and then connecting it to an appropriate instance provider. The following subsections describe this process in more detail.

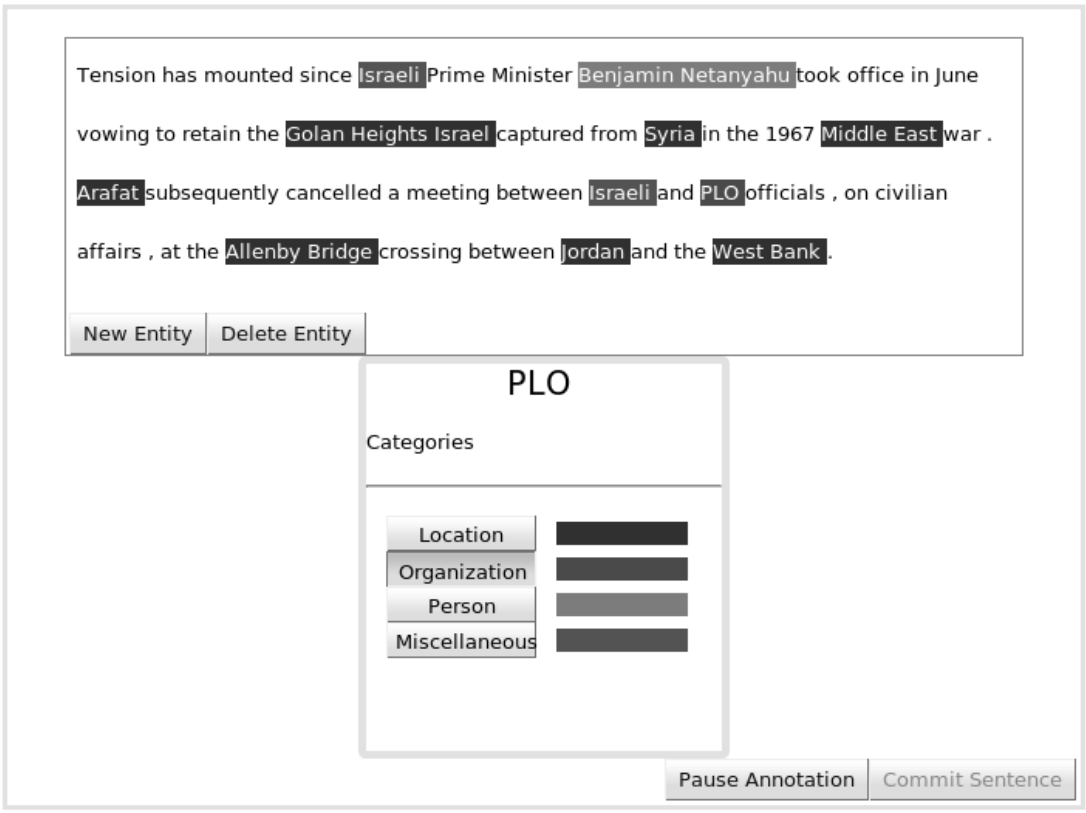

Figure 3: NER Task in CCASH

6.1 Building a Client-side User Interface

In the CCASH framework we have implemented an English part of speech (POS) annotation task (Figure 2) and a named entity recognition (NER) annotation task (Figure 3). In both of these tasks, the user is presented with an interface that gives context at the top of the page and a more focused inspector that we call the "lens" below. When implementing a new task in the web client, developers may either take advantage of this pre-existing layout or else build their own layout.

To build custom client-side interfaces, developers extend a helper class that takes care of bookkeeping such as firing standard timeline events (see Section 4.5). They then create and assemble the widgets necessary to implement their task. As mentioned before, third party widget libraries are available for GWT. CCASH also provides widgets for handling common high-level tasks such as navigating within a sequence of instances and highlighting the instance currently in focus. If no helper widgets fit a given task, a developer is free to implement that task from scratch.

Finally, a task designer who is interested in task-specific timing information will want throw custom timeline events in the web client at the appropriate times.

6.2 Building an Instance Provider

Creating an instance provider consists of implementing an XML-RPC interface whose most important method allows clients to get the next instance for a particular annotator. As explained in Section 5.3, instance providers need not be implemented in Java. However, if a new instance provider is implemented in Java, it can make use of convenience methods for filtering timing events, serialization, and XML-RPC implementation.

CCASH includes Java helper classes with generic instance and annotation types that may be used to implement instance providers with a variety of data types. These helper classes currently support only trivial instance orderings (sequential and random), but we hope to soon provide a complete AL and pre-labeling framework.

7. Case Study

One of CCASH's principle objectives is to facilitate exploratory studies and comparisons of different annotation methods. CCASH has been used for that purpose in a recent user study conducted by Carmen et al. (2010). In this study CCASH was used to record the times of a group of thirty-three linguistic graduate students as they annotated Penn Treebank sentences with English POS tags. They were given additional help in the form of suggestions from a POS tag dictionaries, which consist of simple mappings from each word type to tags that were previously applied to that type. The coverage level of such dictionaries was shown to have an impact on annotation time and accuracy.

Carmen et al. had some non-trivial constraints on study

organization. The study presented each participant with a common set of 18 sentences in the same order with one of six different POS tag dictionary coverage levels. Additionally, the study ensured that each user encountered each coverage level exactly three times, and also that each coverage level was applied to a sentence of significantly different length.

These constraints affected both the order in which sentences were provided to different users and the quality of suggestions offered to the participants. Because of this, we feel that this study provides some evidence for CCASH's ability to handle diverse annotation methods in practice.

Additionally, after setting up CCASH for the study, very little effort was required to run it to completion. Subjects worked from a variety of locations using a variety of web browsers. Administrators were able to monitor the progress of the study from the administrator interface, downloading and reviewing statistics periodically. When one user encountered a minor bug, it was fixed without requiring the participants to reinstall or update any software. Also, data and annotations were collected centrally, eliminating any need to distribute data or collect resulting annotation or timing information.

8. Conclusions and Future Work

CCASH is a web application framework designed to give researchers and corpora builders a common platform for developing cost-efficient annotation methods and for applying them in annotation projects. CCASH currently shows promise in meeting these goals by supporting two common tasks: POS tagging and NER labeling. It has also been successfully used as a platform for a user study evaluating the effectiveness of using POS tag dictionaries to speed up English POS tagging.

We are making the entire CCASH project public on SourceForge.net (http://sourceforge.net/projects/ccash). As we improve the process of extending CCASH with new annotation tasks, we hope that the language resources community will begin to contribute their own annotation tasks, share useful widgets, and collaborate on the framework. At the same time we plan to release a Java framework for AL and automatic annotation.

Additionally we plan to extend CCASH to implement the OpenID protocol (http://www.openid.net) so that users can log in with any OpenID provider, avoiding the annoyance of creating a dedicated CCASH account.

Finally, we are currently implementing a Syriac morphological annotation task in CCASH. Because Syriac is a low-resource language and Syriac morphological annotation is a non-trivial task, expert annotators are expensive. It will be important to quickly determine which annotation methods are most cost-effective, and CCASH will be a good means to accomplish this. This Syriac annotation task will involve a number of annotators dispersed around the world. We are interested in experimenting with different cost-conscious methods for coordinating the efforts of multiple annotators and in building successful approaches into the CCASH framework.

9. Acknowledgements

We would like to thank Jeremy Sandberg for his contributions to the code base for this project.

10. References

Carmen, M., Felt, P., Haertel, R., Lonsdale, D., McClanahan, P., Merkling, O., Ringger, E., Seppi, K. (2010). Tag Dictionaries Accelerate Manual Annotation. In *Proceedings of LREC 2010*. Malta.

Chiou, F.-D., Chiang, D., & Palmer, M. (2001). Facilitating Treebank Annotation with a Statistical Parser. In *Proceedings of the Human Language Technology (HLT) Conference*. San Diego: ACL.

Culotta, A. & McCallum, A. (2005). Reducing labeling effort for stuctured prediction tasks. In *The Twentieth National Conference on Artificial Intelligence (AAAI)*. Pittsburgh, PA: pp. 746-751.

Cunningham, H., Maynard, D., Kalina, B., & Tablan, V. (2002). GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the ACL*. Philadelphia.

Ganchev, K., Pereira, F., & Mandel, M. (2007). Semi-automated Named Entity Annotation. In *Proceedings of the Linguistic Annotation Workshop*. Prague: ACL, pp. 53-56.

Haertel, R., Ringger, E., Seppi, K., Carroll, J., & McClanahan, P. (2008a). Assessing the Costs of Sampling Methods in Active Learning for Annotation. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies*. Columbus, Ohio: ACL Short Papers, pp. 65-68.

Haertel, R. A., Seppi, K. D., Ringger, E. K., & Carroll, J. (2008b). Return on Investment for Active Learning. In *Proceedings of the NIPS Workshop on Cost-Sensitive Learning*. ACL Press.

Lowe, J., Jacobson, M., & Michailovsky, B. (2004). Interlinear Text Editor Demonstration and Project Archivage Progress Report. In *4th EMELD Work-shop on Linguistic Databases and Best Practice*. Detroit.

Marcus, M., Santorini, B., & Marcinkiewicz, M. (1994). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics* 19(2), pp. 313-330.

Morton, T., & Lacivita, J. (2003). Word-Freak: An Open Tool for Linguistic Annotation. In *Proceedings of HLT-NAACL*, pp. 17-18.

Müller, C., & Strube, M. (2006). Multi-Level Annotation of Linguistic Data with MMAX2. In S. Braun, K. Kohn, & J. Mukherjee (Eds.), *Corpus Technology and Language Pedagogy. New Resources, New Tools, New Methods*. English Corpus Linguistics Vol. 3. Frankfurt: Peter Lang, pp. 197-214.

Ngai, G., & Yarowsky, D. (2000). Rule Writing or

Annotation: Cost-efficient Resource Usage for Base Noun Phrase Chunking. In *Proceedings of ACL*, pp. 117-125.

Ogren, P. V. (2006). Knowtator: A Protégé plug-in for Annotated Corpus Construction. In *Proceedings of the 2006 Conference of the NAACL-HLT*. New York: Companion Volume, Demonstrations, pp. 273-275.

Ringger, E., McClanahan, P., Haertel, R., Busby, G., Carmen, M., Carroll, J., et al. (2007). Active Learning for Part-of-speech Tagging: Accelerating Corpus Annotation. In *Proceedings of the Linguistic Annotation Workshop at ACL*, pp. 101-108.

Ringger, E., Carmen, M., Haertel, R., Seppi, K., Lonsdale D., McClanahan P., Carroll, J., & Ellison, N.. (2008). Assessing the Costs of Machine-assisted Corpus Annotation through a User Study. In *Proceedings of LREC 2008*. Morocco.

Settles, B., Craven, M., & Friedland L. 2008. Active Learning with Real Annotation Costs. In *Proceedings of the NIPS Workshop on Cost-Sensitive Learning*, pp. 1069-1078.

Settles, B. (2009). Active Learning Literature Survey. University of Wisconsin-Madison. Computer Sciences Technical Report 1648.

Tomanek, K., Wermter, J., & Hahn, U. (2007). Efficient Annotation with the Jena ANnotation Environment (JANE). In *Proceedings of the ACL 2007 Linguistic Annotation Workshop–A Merger of NLPXML 2007 and FLAC*. Prague.