Interoperability of Annotation Schemes: Using the Pepper framework to display AWA documents in the ANNIS interface

Talvany Carlotto∗, Zuhaitz Beloki†, Xabier Artola†, Aitor Soroa†
∗Center for Language and Cognition Groningen, University of Groningen, Groningen, Netherlands,
†IXA NLP Group, University of the Basque Country, Donostia, Basque Country
∗talvanynet@gmail.com, †xabier.artola, zuhaitz.beloki, a.soroa@ehu.es

Abstract

Natural language processing applications are frequently integrated to solve complex linguistic problems, but the lack of interoperability between these tools tends to be one of the main issues found in that process. That is often caused by the different linguistic formats used across the applications, which leads to attempts to both establish standard formats to represent linguistic information and to create conversion tools to facilitate this integration. Pepper is an example of the latter, as a framework that helps the conversion between different linguistic annotation formats. In this paper, we describe the use of Pepper to convert a corpus linguistically annotated by the annotation scheme AWA into the relANNIS format, with the ultimate goal of interacting with AWA documents through the ANNIS interface. The experiment converted 40 megabytes of AWA documents, allowed their use on the ANNIS interface, and involved making architectural decisions during the mapping from AWA into relANNIS using Pepper. The main issues faced during this process were due to technical issues mainly caused by the integration of the different systems and projects, namely AWA, Pepper and ANNIS.

Keywords: Interoperability, AWA, Pepper, relANNIS, ANNIS

1. Introduction

Natural language processing (NLP) applications demand the integration of many basic linguistic processing modules (Tokenization, POS tagging, Named Entity Recognition and Classification, Syntactic Parsing, Coreference Resolution, etc.) that perform linguistic analysis at several levels. Sophisticated applications such as event extraction systems, for example, are built upon those basic modules, which are typically combined to build complex processing workflows. However, application developers face many challenges when combining NLP modules, which are often tailored to the particular needs of the task at hand and, thus, difficult to integrate. Arguably, one of the main problems encountered when combining NLP modules is the lack of interoperability among them, as each module uses different mechanisms and formats to represent their output, i.e., their annotations.

Nowadays, there exist many linguistic annotation schemes that define logical levels of annotation independent of the annotation's physical format, and are thus able to represent linguistic annotations on many levels. For instance, GATE (Cunningham et al., 1996), AWA (Artola et al., 2009), PAULA (Dipper and Götze, 2005) or NAF (Fokkens et al., 2014) are some of the most relevant projects where standoff annotation is used to deal with the combination of multiple overlapping hierarchies characteristic of the multidimensional nature of linguistic information.

There have been also attempts to establish standards for representing linguistic information with the aim of facilitating effective resource management and integration of NLP tools (Ide and Romary, 2004). However, as desirable as it might be, expecting such a unique representation standard to be adopted by the NLP community is unrealistic. The reasons for this are numerous, including, among others, the following:

- Linguistic standards are developed following diverse goals, often contradictory ones. A linguistic standard has to be able to represent a great variety of linguistic phenomena, yet it has to do so in a rigid, unambiguous way, so that users know how to map their internal representation to the standard representation.
- NLP is a very active area of research, where new techniques and applications are constantly developed. Devising a uniform representation scheme that is able to cover all the linguistic levels required today is very difficult, if not impossible.
- There exist many representation schemes which lack proper toolsets, libraries and APIs upon which to effectively build applications. For instance, the ISO/TC37/SC 4 standard on language resource management lacks, as far as we know, any library or API that allows its use. Likewise, tools for visualizing annotations, or tools for aiding corpus manual annotation are missing.
- Many representation schemes have been designed with a particular language (or language family) in mind, which hinders its adoption for languages with different morphosyntactic structures.

In this paper we study the interoperability among annotation schemes by using the Pepper architecture (Zipser et al., 2015). Pepper is a framework for the conversion of linguistic annotations among formats. It uses a data model named Salt as a *lingua franca* among linguistic schemes, and the conversion among formats is achieved by mapping each one into Salt. Pepper can be used to convert documents into the relANNIS format, which in turn is supported by ANNIS (Krause and Zeldes, 2014), a corpus search tool that supports diverse multilayer corpora with appropriate search and visualization .

We describe experiments of converting a corpus linguistically annotated by following the annotation scheme AWA

(described in section 2.) into the Pepper framework. As a result, we can benefit from the libraries and visualization tools offered by ANNIS. Our experiment not only allows us to visualize and search for AWA annotations in an interface known for its accessibility, performance and scalability features (Krause and Zeldes, 2014), but it also reveals some challenges and practical issues one has to face when dealing with interoperability aspects on linguistic annotation schemes.

2. AWA

The Annotation Web Architecture (AWA) representation scheme was presented by Artola et al. (2009), with the aim of facilitating the communication between linguistic processors in a variety of NLP applications. AWA is specifically designed to represent morphologically-rich languages such as Basque, and also to represent interpretational ambiguity at many levels. AWA guarantees that linguistic annotations are properly represented, and avoids information redundancy. The AWA scheme defines a layered annotation format. If a process is meant to add information that cannot be described by existing layers, it simply adds a new annotation layer. Any previous layers remain intact and can still be used by other processes. Layers are connected by references from one layer to items in another (lower-level) layer.

The design goals and main characteristics of the AWA scheme enable multi-level interoperability among linguistic processors, and nowadays there exist several NLP tools that produce and consume AWA documents, such as *Morpheus* (Alegria et al., 1996), a wide-coverage morphosyntactic analyzer for Basque, and *Eustagger* (Ezeiza et al., 1998), a general-purpose tagger/lemmatizer. The representation schemes corresponding to the annotations of the following processors have also been defined: *Ixati* (Alegria et al., 2006), a shallow parser, which includes the NERC tool Eihera+; *Edgk/MaltIxa* (Aranzabe et al., 2012), a dependency grammar parser; *EusWN* (Agirre et al., 2014), a word-sense disambiguator that uses an algorithm based on random walks over the Basque WordNet; and a named-entity disambiguator that links entity names with the corresponding Wikipedia pages (Fernandez et al., 2011). In addition, several tools have been developed to manage annotations and corpora represented in the AWA scheme: *EULIA* (Artola et al., 2009), an environment to manually annotate AWA documents; *Armiarma*[1], a web interface to search documents in corpora annotated in AWA and to disambiguate them at segmentation, morphosyntactic or lemmatization level; and *Abar-Hitz* (Díaz De Ilarraza et al., 2004), a graphical and language-independent tool for the creation and management of the Basque Dependency Treebank.

This extensive application of the model shows that AWA constitutes a coherent and flexible representation scheme that serves as the basis for the exchange of information at different levels of analysis.

All the elements in AWA belong to three basic types (see Figure 1):

`Anchor`: these elements represent nuggets of information to which linguistic information is attached. They can vary from physical elements found in the input text (character offsets or XPointer expressions referring to specific points or ranges within an XML document) to annotation items resulting from earlier annotation processes. In particular, AWA allows linguistic annotations produced in a given step to be later used as anchors in a subsequent linguistic annotation phase. For instance, in AWA we can create an annotation that attaches information to a specific analysis of a word, rather than to the word itself. This is a powerful feature, which allows an accurate representation of a very broad range of linguistic phenomena. Structural ambiguity is represented by annotations that correspond to overlapping anchors.

`AnnotationItem` (links): Links are the annotation outcomes of linguistic analysis processes, and each link connects one single linguistic interpretation to an anchor. Several links can be connected to one same anchor, which allows the representation of interpretation ambiguity. In this case, the disambiguation is carried out by marking only one of the links as correct, and considering the other ones as incorrect.

`LingInfo`: these elements represent the different types of linguistic content resulting from the analysis processes. In AWA, linguistic content is represented by typed, TEI-encoded feature structures. In morphological segmentation, lemmatization, and other processes, the linguistic content is composed of actual word forms, which allows the use of massive common libraries that contain them as feature structures. This saves both processing time and storage space, as words only need to be analyzed when they first occur.

The previously mentioned classes are the main types of the AWA representation scheme. However, different specializations of the main classes are available in AWA to represent the following levels of analysis: tokenization, multiwords, segmentation, morphosyntax, dependency, coreference, shallow syntax and constituent trees.

Sometimes, an anchor is not a single annotation generated in a preceding step, but a more complex structure of annotation groups. AWA accepts this kind of anchors in a natural manner, allowing new anchor classes to inherit from the main `Anchor`. This is especially useful when an annotation must refer to a group of annotations representing a semantic interpretation.

3. Integrating AWA into ANNIS

ANNIS (ANNotation of Information Structure) is a browser-based tool for searching and visualizing complex and diverse linguistic annotations. Among others, it is able to draw tree-structured annotations. ANNIS requires data to be represented in the relANNIS format, so for a specific format to be used in ANNIS, there must be a conversion from the format into relANNIS. Our documents were in the AWA format, so we had to convert them into the relANNIS format, so that they would be supported by the ANNIS interface.

[1] http://ixa2.si.ehu.es/armiarma

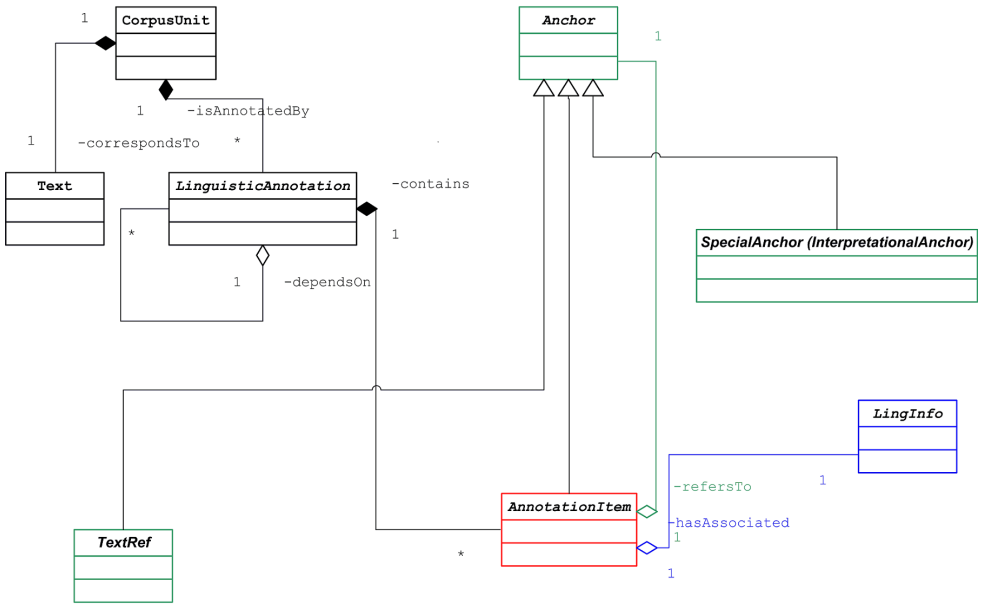

Figure 1: Abstract view of the Annotation Web Architecture.

Pepper is a framework for converting between different linguistic annotation formats. Pepper supports many formats by default, such as EXMARalDA, Tiger XML, MMAX2, RST, TCF, TreeTagger format, relANNIS and PAULA. Nevertheless, the framework can be extended to support custom formats by implementing a plugin module that maps the new format to and from any of the supported formats. As AWA is not one of the formats supported by the Pepper framework, we had to develop a module in the Pepper framework, so we could integrate AWA annotations into Pepper, and therefore convert AWA documents into relANNIS documents.

The creation of a new module in Pepper has no relation to the target format of the desired conversion. New modules perform conversions from the source format (AWA in our case) to an intermediate format called Salt. Salt is a graph-based, abstract model, general enough to allow the representation of almost any linguistic format. The conversion from the Salt format into the target format (relANNIS in our case) is carried out by Pepper as default. When a new module is integrated in Pepper, the conversion of documents in formats supported by the new module can be performed by the Pepper interface. The user provides the Pepper interface with a flow document containing instructions about the source and target formats and folders, and Pepper performs the conversion.

In AWA, each annotation layer is composed of several TEI-like files, which represent the annotations corresponding to one given document[2]. As traditionally source documents used in Pepper are composed of one single file per document, Pepper tries to convert every file containing the extension supported by the source format into a new document. In our module, we had to filter documents during this documents creation, and only execute it for the first file of every AWA document. Since all AWA files are compressed with the *.bz2* extension, we defined *.bz2* as the supported file extension for AWA in Pepper.

The organization of corpora and of corpus documents is different between AWA and relANNIS. Take, for instance, the basic corpora file structure in both formats. In AWA, each document has a system folder containing all the files that represent the annotations of that specific document. A corpus in AWA is composed by a variety of these folders. In relANNIS, however, a corpus is one list of files corresponding to the different layers of the corpus. Every file contains information about a specific layer regarding all the documents in the corpus. One of the outcomes of that difference is that one corpus in AWA has many more files than a corpus in relANNIS, but files in relANNIS can become considerably large, while in AWA they tend to maintain a size limit. Figure 2 shows a diagram depicting a simplified version of the conversion process from an AWA corpus into a relANNIS corpus.

To give another example of differences between the formats, let us take a look in a simple annotation both in AWA and relANNIS. The word *gaelikoa* is an example token in a document of our corpus. In AWA, the annotation for that specific word in that specific document is XML-encoded, as displayed in Figure 3. In relANNIS, however, the word receives a code (23176, in the example of Figure 4), which is mapped in a separate document, and the annotations can be found in the tabulated format depicted in Figure 4.

Tokenization of words in AWA-annotated corpora results in a file that stores, for each token, its initial position in the original text and its offset. That approach is not compatible for converting AWA into Salt. Some special characters that are present in some AWA documents, as, for instance, the * * (non-breaking space character), are not recognized by the Pepper engine, and the resultant offset is incorrect.

[2] Although, since 2015, Pepper also supports the conversion of TEI documents, the conversion provided by Pepper is not enough to convert a group of interrelated AWA/TEI documents.

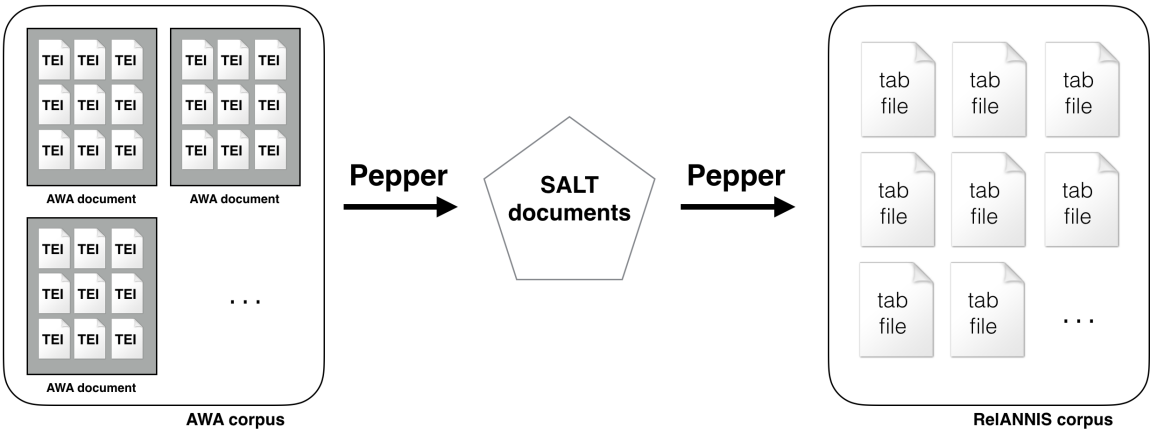

Figure 2: A diagram representing the conversion from an AWA corpus into a relANNIS corpus.

```
<?xml version="1.0"?>
<p>
    <fs id="L-A-IZE-ARR-2454" type="lemmatization">
        <f name="form">
            <str>gaelikoa</str>
        </f>
        <f name="assembled-lemma">
            <str>gaeliko</str>
        </f>
        <f name="morpho-features">
            <fs type="toplevel-feature-list">
                <f name="POS">
                    <sym value="Noun"/>
                </f>
                <f name="SUBCAT">
                    <sym value="Common"/>
                </f>
                <f name="Animate">
                    <minus/>
                </f>
                <f name="CASE">
                    <sym value="Absolute"/>
                </f>
                <f name="NUMBER">
                    <sym value="Singular"/>
                </f>
                <f name="DEFINITENESS">
                    <sym value="Definite"/>
                </f>
            </fs>
        </f>
    </fs>
</p>
```

Figure 3: Example of annotation in AWA. The annotations for the word *gaelikoa* (Basque for *gaelic*) are highlighted by the blue rectangles.

In our Pepper model, we did not use the offsets stored in the AWA documents. The tokenization process had to be reimplemented inside the model with the creation of new offsets. The new offsets were obtained by looking for each token in the original document using a starting search point, initially the beginning of the document. When the token is found, its last position becomes the new starting search point for the next token, and so on.

The actual amount of code that had to be written for the experiment was not considerably large - it consisted of two Java classes containing around 100 lines. Most of the work involved taking the architectural decisions described above while mapping AWA into Pepper and dealing with the setup and integration of Pepper and ANNIS.

```
23175   default_ns  POS                 Noun
23175   default_ns  SUBCAT              Common
23176   default_ns  id                  Xw1100
23176   default_ns  assembled-lemma     gaeliko
23176   default_ns  POS                 Noun
23176   default_ns  SUBCAT              Common
23176   default_ns  CASE                Absolutive
23176   default_ns  NUMBER              Singular
23176   default_ns  DEFINITENESS        Definite
23177   default_ns  id                  Xw1101
23177   default_ns  assembled-lemma     eta
23177   default_ns  POS                 Junction
```

Figure 4: Example of annotation in relANNIS. The annotations for the word *gaelikoa* are highlighted by the blue rectangle.

4. Experiments and testing

Three basic steps are required to use an AWA corpus in the ANNIS interface. First, (1) the new Pepper module we developed is used to make the conversion from the AWA documents into the relANNIS format; then (2) the newly converted files have to be imported into the ANNIS database; and finally (3) we can execute queries in the ANNIS interface. Steps one and two have to be executed only once, i.e., it is not necessary to make the conversions every time you wish to execute queries in the corpus.

We used a corpus of one thousand AWA-annotated documents to evaluate performance, corresponding to approximately 40 megabytes of files. Each of the three steps was executed three times to have an average time, and all executions were processed by a computer equipped with a 2.6 GHz processor and 8GB of RAM memory. The first step (conversion from AWA into relANNIS) took in average fourteen minutes to finish for the one thousand document corpus. The second step (importing the documents into the ANNIS database) took in average five minutes to finish. The performance of the third step will be explained with an example in the following paragraphs.

Figure 5 shows the result for a query that looks for occurrences of an adjective (in red in the image) followed by a non-adjective word (in purple in the image), followed by two any words followed by a word with lemma *etxe* (Basque for *house*, in green in the image). The query

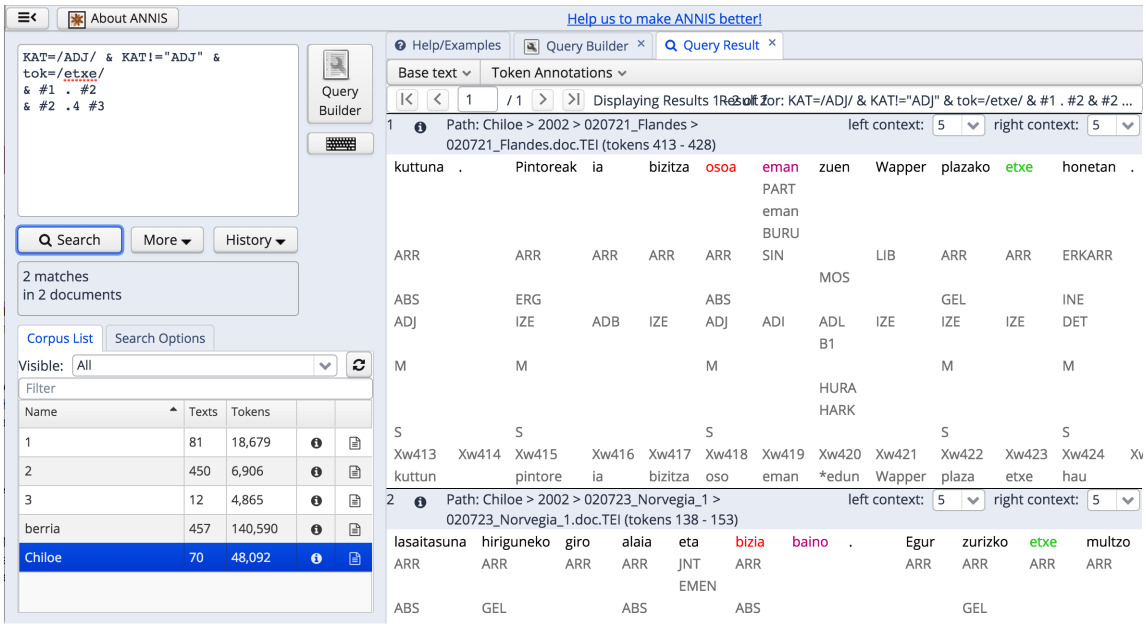

Figure 5: Example query in the ANNIS interface.

described here was executed on the ANNIS interface and looked for results in the one thousand documents of the corpus. It took less than one second for the results start to show up on the screen. Queries can be simpler or more complex than this one, but we should mention that, for all queries we tried in this corpus (less complex than the example or more complex than the example), it took less than one second for the first results to appear.

5. Conclusions

This paper addresses the problem of interoperability among annotation schemes. We describe a practical experiment to map AWA annotations into the relANNIS format and we show the feasibility of our approach. As a practical result, corpora annotated following AWA can be graphically visualized and consulted using ANNIS.

From the experiment, we have learned lessons we believe to be a valuable contribution for anyone interested in integrating NLP tools that use heterogeneous annotation formats.

Both Salt and AWA are very general schemes that follow conceptually sound principles, avoid information duplication, and properly represent linguistic annotations at many levels. In our opinion, this is corroborated by the fact that the conversion has been possible without major problems, and that no information has been lost in the way.

The main problems faced were due to technical issues such as the use of different character encoding by the systems, which complicate practical aspects such as computing the right offsets of textual anchors.

The lack of technical expertise using the Pepper framework has been another important obstacle, as well as the fact that the documentation was not up to date. However, the Pepper developers have been eager to help and they have offered us excellent technical support.

Based on our experience, we believe that similar experiments of integrating specific annotation schemes as AWA into ANNIS would face a similar situation to ours. We consider Pepper a powerful tool in that process, and believe that other researchers would benefit from its use. However, the issues that are described in this paper should be taken into account when trying to replicate the same experiment.

6. Bibliographical References

Agirre, E., de Lacalle, O. L., and Soroa, A. (2014). Random walks for knowledge-based word sense disambiguation. *Computational Linguistics, 40:1. ISSN 0891-2017*, 40(1):forthcoming.

Alegria, I., Artola, X., Sarasola, K., and Urkia, M. (1996). Automatic morphological analysis of Basque. *Literary & Linguistic Computing*, 11(4):193–203.

Alegria, I., Arregi, O., Ezeiza, N., and Fernandez, I. (2006). Lessons from the Development of a Named Entity Recognizer. *Procesamiento del Lenguaje Natural*, (36):25–37.

Aranzabe, M., Bengoetxea, K., Díaz de Ilarraza, A., Ezeiza, N., Goenaga, I., and Gojenola, K. (2012). Combining rule-based and statistical syntactic analyzers. In *ACL 2012 Joint Workshop on Statistical Parsing and Semantic Processing of Morphologically Rich Languages (SP-Sem-MRL 2012), Association for Computational Linguistics (ACL), USA, ISBN: 978-1-937284-30-5*, pages 48–54, Jeju Island, Republic of Korea.

Artola, X., Díaz de Ilarraza, A., Soroa, A., and Sologaistoa, A. (2009). Dealing with complex linguistic annotations within a language processing framework. *IEEE Transactions on Audio, Speech and Language Processing*, 17(5):904–915.

Cunningham, H., Wilks, Y., and Gaizauskas, R. J. (1996). GATE: a General Architecture for Text Engineering. In *Proceedings of the 16th conference on Computational linguistics*, pages 1057–1060. Association for Computational Linguistics.

Díaz De Ilarraza, A., Garmendia, A., and Oronoz, M. (2004). Abar-hitz: An annotation tool for the basque dependency treebank. In *Proceedings of LREC2004*, pages 251–254.

Dipper, S. and Götze, M. (2005). Accessing heterogeneous linguistic data. generic XML-based representation and flexible visualization. In V. Saraswat et al., editors, *In Proceedings of the 2nd Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics*, pages 206–210. MIT Press, Poznan, Poland.

Ezeiza, N., Aduriz, I., Alegria, I., Arriola, J. M., and Urizar, R. (1998). Combining stochastic and rule-based methods for disambiguation in agglutinative languages. In *Proc. of COLING-ACL'98*, pages 10–14, Montreal (Canada).

Fernandez, I., Alegria, I., and Ezeiza, N. (2011). Semantic relatedness for named entity disambiguation using a small wikipedia. In *Text, Speech and Dialog, TSD 2011*, pages 276–283.

Fokkens, A., Soroa, A., Beloki, Z., Ockeloen, N., Rigau, G., van Hage, W. R., and Vossen, P. (2014). NAF and GAF: Linking linguistic annotations. In *To appear in Proceedings of 10th Joint ACL/ISO Workshop on Interoperable Semantic Annotation (ISA-10)*.

Ide, N. and Romary, L. (2004). International standard for a linguistic annotation framework. *Natural Language Engineering*, 10(3-4):211–225.

Krause, T. and Zeldes, A. (2014). Annis3: A new architecture for generic corpus query and visualization. *Digital Scholarship in the Humanities*.

Zipser, F., Krause, T., Lüdeling, A., Neumann, A., Stede, M., and Zeldes, A. (2015). ANNIS, saltnpepper & PAULA: A multilayer corpus infrastructure. In *Final Conference of the SFB 632 Information Structure: Advances in Information Structure Research*, Berlin, Germany, Mai.