Proceedings of the 16th Linguistic Annotation Workshop (LAW-XVI) @LREC 2022, pages 170–181
Marseille, 24 June 2022

GRAIL—A Generalized Representation and Aggregation of Information Layers

Sameer Pradhan[1,4] **and Mark Liberman**[1,2,3]
[1]Linguistic Data Consortium,
[2]Department of Linguistics,
[3]Department of Computer Science,
University of Pennsylvania, Philadelphia, USA
[4]cemantix.org
spradhan@upenn.edu
cemantix.org

Abstract

This paper identifies novel characteristics necessary to successfully represent, search, and modify natural language information shared simultaneously across multiple modalities such as text, speech, image, video, etc. We propose a multi-tiered system that implements these characteristics centered around a declarative configuration. The system facilitates easy incremental extension by allowing the creation of composable workflows of loosely coupled components, or plugins. This will allow simple initial systems to be extended to accommodate rich representations while providing mechanisms for maintaining high data integrity. Key to this is leveraging established tools and technologies. We demonstrate using a small example.

Keywords: Annotation, Representation, Corpora, Framework

1. Introduction

In this paper, we propose a novel representation that is capable of addressing some frequent use cases that arise during the manipulation of data and annotations spanning multiple modalities. To the best of our knowledge, none of the existing systems are capable of gracefully addressing them. The proposed approach is capable of handling multiple modalities of information. However, for the purposes of this article, we will restrict ourselves to areas of research that deal with three modalities: i) Natural Language Processing (NLP) (a.k.a. Computational Linguistics (CL)); ii) Automatic Speech Recognition (ASR); iii) Computer Vision (CV). A fortunate side-effect of neural network methods is an exponential growth in research on multimodal data across various disciplines (Ramachandram and Taylor, 2017). In addition, the availability of large datasets, and fast GPUs has made it possible for an individual without explicit linguistic, acoustic, image processing, or other forms of knowledge, to assemble a system demonstrating state of the art performance across a wide array of "understanding" tasks. However, all these advances have not (yet) made redundant the need for some level of task-specific supervision. Such supervision is typically provided through a combination of gold standard and predicted annotation layers. Over the past two or three decades, each community has made significant progress in terms of the tools and representations that allow the capture of multiple layers of annotations *within* their subdomain. However, the problem identified by Bird and Liberman (2001), the lack of standards to guarantee interoperable representations across the input signals and associated annotation remains *largely unsolved*.

Many existing tools and methods tend to be fragmented and brittle. Small changes in the information aggregate can impose a substantial toll on orchestrating the harmony across multiple annotation layers. The typical approach for addressing this disconnect is an *ad-hoc* manipulation of information (either content or annotations) at a stage lying somewhere *after* it is captured and *before* being used for training; or pre-processing information *before* the application of trained models to unseen cases in order to ensure maximum compatibility with the assumptions made during training.

We start by reviewing the state of frameworks in Section 2. In Section 3 we look at the collection of serializations that have been proposed over the years with a quick look at the available tooling in Section 4 before presenting the need for a generalized architecture in Section 5, followed by details of the architecture in Section 6 with a Subsection 6.3 demonstrating some core capabilities using concrete use case. We conclude in Section 7.

2. State of Frameworks

Over time many types of annotations have been created within as well as across the three modalities. While some layers of annotations can be independent of others, they typically tend to be interdependent. These dependencies can range from very simple to very complex. Many annotation frameworks[1] have been proposed over the years to enable the capture, storage and manipulation of this information aggregate. Due to space limitation, we will highlight only some of them[2]. Following are a few notable frameworks de-

[1]A framework is a collection of (software) tools, libraries and methodologies to help manage the data and annotations.

[2]For a more detailed information on the evolution of various frameworks, the reader can refer to the Handbook of Lin-

veloped over the past two decades:

- **ATLAS**: A Flexible and Extensible Architecture Linguistic Annotations (Bird et al., 2000; Bird and Liberman, 2001; Maeda et al., 2006)
- **GATE**: General Architecture for Text Engineering (Cunningham, 2002)
- **UIMA**: Unstructured Information Management Architecture (Ferrucci and Lally, 2004)
- **LAF**: The Linguistic Annotation Framework: A Standard for Annotation Interchange and Merging (Ide and Suderman, 2014)
- **ELAN**: A Professional Framework for Multimodality Research (Wittenburg et al., 2006)
- **EMU**: Advanced Speech Database Management and Analysis (Winkelmann and Raess, 2014; Winkelmann et al., 2017; Jochim, 2017)
- **ANNIS**; Complex Multilevel Annotations in a Linguistic Database (Dipper et al., 2004; Götze and Dipper, 2006; Zeldes et al., 2009; Rosenfeld, 2010; Zipser et al., 2015; Krause and Zeldes, 2016)

GATE was probably one of the first comprehensive suite of tools that could be used to annotate and tag linguistic information on text. It was created during the heyday of the Java programming language. The GATE ecosystem has evolved over time and is currently being overhauled to use cloud architecture. Unfortunately, the new version is not available for testing yet.

As for UIMA, its strong coupling with the Java programming language has had a severely negative impact on its user base as Python has emerged as the language of choice for most popular frameworks across NLP, speech and video. The underlying Common Annotation Structure (CAS) which claimed to address various interoperability issues through the creation of a type system did not live up to the hype.

The Linguistic Annotation Framework (which includes GrAF) is designed with the assumption that all annotations should be represented as graphs and manipulated using various graph algorithms of minimization, transduction, etc. LAF (and GrAF) framework is not being actively developed but is being adopted by the TextFabric[3] that is using it for curating corpora of ancient texts.

ANNIS, in combination with PAULA XML and Salt-NPepper has an active, large community. The Salt-NPepper modules play a similar role in the ANNIS framework—somewhat akin to the role SQL plays in the database landscape. It handles multiple modalities using a *pluriverse* approach where multiple disparate layers of different annotations and variations within annotation schemata for similar phenomena.

One other notable example is the OntoNotes corpus (Weischedel et al., 2011) which used a relational data model (Pradhan et al., 2007), to capture inter- and intra- layer connections and delegated constraint checks to its ACID[4] conformant engine. Individual layers of annotations were independently serialized in separate files with minimum inter-layer data coupling (Pradhan and Ramshaw, 2017). Unfortunately, it did not see adoption outside the project itself. Recent introduction of data versioning systems (DVS) and the use of Data Frames for representing such information, seem to reinforce the importance of an underlying relational data model.

3. State of Representations

In the previous section we looked at some annotation frameworks. Over the years, there have been large scale initiatives such as the Text Encoding Initiative (TEI) (Ide and Véronis, 1995) and international standardization efforts such as the ISO TC37 SC4. The NLP community has seen numerous annotation formats over the years, with the general consensus that they are best represented using some graph formalism. The requirements of such formats can vary quite a bit depending on whether it is being used during the creation of complex annotations or whether a stable version of this is used for training machine learning models, or for purposes of teaching. In the first case where the users are creators of some complex set of annotation, it is important to have a rich set of tools and representations to address the issues that creep over a lifetime of an annotation project, such as evolution in guidelines which can necessitate retroactive updates to annotations in order to create a consistent body of annotations. On the other hand, (typically read-only) consumers of annotations don't need to understand or deal with data complexities that don't impact its use. We cannot cover a complete history of work in this area, but will discuss a few notable cases.

- **The LAF, GrAF, TCF and LIF family**—The Linguistic Annotation Format (LAF) (Ide and Romary, 2004) and its successor—Graph Annotation Format (GrAF) (Ide and Suderman, 2007) primarily used XML.
- **NXT**—Short for NITE XML Toolkit (Calhoun et al., 2010; Carletta et al., 2005), where NITE stands for Natural Interactivity Tools Engineering, is a multi-level, cross-level and cross-modality annotation representation, retrieval and exploitation of multi-party natural interactive human-human and human-machine dialogue data.
- **EAF**—ELAN Annotation Format (EAF), is an XML based data serialization format is part of a larger Abstract Corpus Model[5] (ACM).
- **AG**—This is the annotation graph XML format used by various tools to create and manipulate in-

guistic Annotation (Ide and Pustejovsky, 2017)

[3] https://github.com/annotation/text-frabric

[4] In computer science, ACID (atomicity, consistency, isolation, durability) is a set of properties of database transactions intended to guarantee data validity despite errors, power failures, and other mishaps.

[5] http://emeld.org/workshop/2003/brugman-paper.html

ternally to store various corpora by the Linguistic Data Consortium which are typically released as simpler representations.

- **TextGrid**—This is the underlying format for files created and used by the Praat tool[6] (Boersma and others, 2014)—probably the most popular tool used by researchers and students for the study of computational phonetics.
- **CHAT**—This is the serialization used by the CLAN programs that have evolved over the years as part of the CHILDES project (MacWhinney, 2014) which has grown to become a larger collection—the TalkBank. This also has a task specific nature.
- **The CoNLL-* family**—The Computational Natural Language Learning (CoNLL) shared tasks initiated a culture of yearly international evaluations, starting in 2002, to promote consistent and replicable research. The initial data representation was in the form of a space (or, tab) separated table of columns one of which being the words and the other being a sequence of labels that identified various annotation classes such as base phrases, named entities, etc. Each year, a new task *typically* added one or more columns to this table creating what came to be widely recognized as CoNLL (column) format.

  The Universal Dependencies effort (De Marneffe et al., 2021) started as a project for representing dependencies across many languages in a consistent fashion. This group embellished the CoNLL format, starting with version that represented dependency trees, and gave it a new moniker CoNLL-U (Universal). The Universal Dependencies effort has spawned off-shoots in coreference, morphological layers, named entities, etc. and has become the consumer favorite. Notable *extensions* to this are CoNLL-UA (Universal Anaphora) and CoNLL-UP (Universal Propositions). There have been recent updates to this format to allow the injection of useful metadata.
- **Symbolic Expressions**—One of the oldest, large scale and successful annotation projects—The Penn Treebank (Marcus et al., 1993) used Lisp-like symbolic expressions (S-Expressions or `sexp`) to represent syntax trees. A variation of such formalism called the PENMAN (Kasper, 1989) notation was used for defining the Suggested Upper Merged Ontology (SUMO) (Bateman, 1990; Bateman et al., 1990). This has seen a recent revival in the Abstract Meaning Representation (AMR) project (Banarescu et al., 2013).

We can see that over the years many task agnostic formats were based on a larger ecosystem of serialization technologies such as XML, and have recently seen some evolution to use JSON, as a result of the growth and significance of the world wide web. Two of these formats—TextGrid and CHAT—addressed specific tasks in the humanities discipline. They were not designed to be extensible which led to some backwards incompatibilities. Also since they don't have a formal grammar, it is harder to write tools to manipulate them. Finally, somewhat surprisingly, defying all principles of database theory, the CoNLL family of formats, which are essentially a collection of single *unnormalized*, tables of data, has become the most widely used format by most NLP researchers. And we are seeing some resurgence in the use of symbolic-expressions to represent rich graph structures.

The data formats for storing binary data such as audio and video signals is a completely different branch that has seen various proprietary and open source standardizations somewhat akin to the evolution of the Unicode standard for text. Most of the annotation formats that deal with audio and video information use offsets into this data typically as time points/intervals possibly along with a spatial specification commonly in the form of a pair of coordinates bounding box (or, bounding rectangle) for two dimensional signals which are the most commonly used ones[7].

4. State of Tooling

Although we are going to focus on *architecture* in this article, it is very important to acknowledge the fact that availability of *right tools* and *libraries* plays a crucial role in minimizing the inertia in its adoption and evolution. Unfortunately, creation of novel architectures and toolings is also one of the least funded areas[8]. For a very recent and thorough survey of of tools that are available for various document annotation phenomena we refer the reader to Neves and Seva (2021). They list some 60 tools and thoroughly evaluate 15 of them using 26 criteria that cover multiple aspects of the annotation and tooling requirements. It is evident that many tools have been moving to use the web and cloud based architectures but are mostly centered on a graphical user interface. One tool—SLATE- (Kummerfeld, 2019)–that stands out from others by catering to a niche user base—*an expert*—someone who prefers a command line interface.

5. Case for a Generalized Architecture

It would be helpful to reiterate that one of the important lessons that the community–specifically the ones evolving a science of annotation–has learned over the past couple of decades is that the most robust abstract representation of a many different kinds of (or, layers of) annotations has roots in a graph formalism. The LAF framework attempts at a representation that can capture conflicting variations in annotation schemas for

[6] `https://praat.org`

[7] The discussion of three dimensional signals such as lidar data used for autonomous driving is beyond the scope of this discussion.

[8] We can only speak from experience in the area of tooling in natural language processing research

a given layer of annotation—the classic example being that of difference in word segmentation across guidelines. They propose a way forward for merging across layers with such variations through the use of *dummy nodes* that can be resolved in multiple ways while reading or serializing a specific version. The LAF architecture, however, only deals with text sources. Chiarcos et al. (2012) provide an elaborate discussion on the potential complexities introduced by minor differences in representational decisions made by individual annotation schemas in a multi-layered annotation corpus. They provide an algorithm for merging annotations using the case of conflicting token representations across such layers. The problem gets further complicated when the notion of temporal intervals is introduced as described in the ExMARaLDA effort (Schmidt, 2004). Both these efforts address special case of a class of problems that are expected to multiply with the addition of additional modalities such as speech signals that are a function of time; and visual information which adds a spatial dimension to the mix. The algorithms presented in these are engineered for many such eventualities on an as-needed basis. This approach is likely to prove prohibitive in the long run. Most approaches poorly address the need for capturing metadata associated with the data itself.

Using *declarative constraint specifications*, for example, an *interval algebra* in the temporal domain, or using constraints on transformation of *graphic primitives* in a spatial domain, could allow one to generalize the solutions at a higher level of abstraction which could allow the creation of a class of solutions that would reasonably manage a potential explosion of checks across possible constraint violations. Furthermore leveraging developments in *version control* and *fully persistent data structures* (Driscoll et al., 1989) and *conflict-free replicated data types* (Preguiça, 2018) which have standard implementations in many languages that are very efficient in time and space could allow room for better integration across schema evolutions. One other issue with the existing frameworks is their typically monolithic nature that results in a steep learning curve. An architecture that attempts to decompose the typical domains into smaller sub-domains can facilitate selective and *incremental adoption* by end users and also allow for creation of flexible extensions to address edge cases specific to a particular sub-domain.

6. Proposed Architecture

A way to handle various slices of the representations, maybe even an individual layers locally while still allowing global consistency guarantees could substantially relive the cognitive load on the user, or in other words could go a long way in managing the *incidental* and *accidental complexities* of the tooling, which would be even more important given the significant *essential* complexities arising from the integration across multiple modalities.

The architecture we propose here does claim to be a new invention. Rather our design approach can be compared with the evolution of the concept of the *blockchain*, which as detailed by Narayanan (2017) is a careful selection and assembling of a collection of conceptual and technological innovations that happened over the past fifty some years. The UNIX operating system designers made a very similar claim[9] (Ritchie and Thompson, 1974). We have identified existing tools, technologies and propose to follow well established design principles such as, for example, *separation of concerns*, the liberal use of *open/closed principle*, and a decomposition of the domain into modules that can be composed together in various declarative configurations, as opposed to a monolithic design.

6.1. Design Requirements

All the design requirements that we will discuss are assumed to operate over a corpus with the following general characteristic of the underlying data and annotations:

- Multiple layers of span-*ed* or span-*less*[10]; time-*ed* or time-*less*[11]; annotations
- Multiple media types and encodings
- Different annotation guidelines
- Produced using different tools

6.1.1. Functional Requirements

Here we list some functional requirements that the formalism absolutely has to satisfy.

- **Selective Disassembly and Reassembly**—This is an important requirement that we address in our architecture as the example we discuss later will highlight.
- **Structured Querying Capability**—One should be able to perform structured queries spanning media and layers.
- **Ensure Synchronization after Modifications to Layer(s)**—For example, it should be *reasonably easy* to propagate changes in one layer to other layers while maintaining certain core constraints,

[9] "The success of UNIX lies not so much in new inventions but rather in the full exploitation of a carefully selected set of fertile ideas, and especially in showing that they can be keys to the implementation of a small yet powerful operating system."—Ritchie and Thomson (1974)

[10] A span-*ed* annotation is one that is associated with a specific text span. Named-entities, base phrases, sentences, etc. fall in this category. Whereas span-*less* annotations are ones that are not *directly* associated with one specific span. Typically they tend to capture relation between two or more annotations that themselves may be span-*ed* or span-*less*. For example, an identify coreference relation between a set of span-*ed* entities and/or events in a text.

[11] A classic example of time-*less* annotation is punctuation in a transcript; The space between words in a transcript on the other hand can represent many different time durations. It can be almost negligible (given some lower duration threshold) with an effective duration of zero, or could range from several milliseconds to several seconds or more with a positive value of time duration.

- **Customized Aggregation**—Allow modular and customized information aggregation strategies.

6.1.2. Non-Functional Requirements

we have identified various capabilities that would be expected of this architecture. We have identified a few of these that we consider to be salient and concepts that have not *so far* been sufficiently exploited by existing frameworks.

- Prefer **convention** and **configuration** over writing custom code.
- A focus on **functional decomposition** across different modalities at both the level of data and multiple layers to promote incremental adoption.
- Allow the capture of **metadata** at various levels—including metadata on the annotations themselves.
- **Retain rich source context** to allow for its potential regeneration.
- Allow **declarative specification** of entities, constraints and transformations.
- Rely on **functional data structures** which are the underpinnings of modern **version control** systems
- **Delegate** complex constraint satisfaction requirements to tools like **relational database engines**
- Build on an ecosystem of established **data abstractions and libraries** rather than from scratch.
- Allow customizations through special **modes**, **hooks** and **plugins** .
- Adopt **literate annotation** practices
- Easy but powerful **data importing and exporting** mechanism.

6.2. An Implementation

In this section we will cover some details of the choices we made over possible implementations that let us adhere to the list of criteria that we listed in the earlier section.

6.2.1. Convention and Configuration over Code

Convention can go a long ways in keeping information easily understandable and shareable. Our architecture makes very few assumptions about the data, and allows the creation of a *multi-tier configuration* with sensible defaults for a small class of typical set of roles expected of a user, or a typical combination of modalities. The user can decide to tune the configuration as they become more comfortable using the system and in a way that allows them to be most effective at a given task. We will look at two example abstraction that can go a long way in reducing task complexity and allowing for better data consistency.

A case specialized MIME

When dealing with multiple modalities of data and a mix of text, binary or mixed content in files, it becomes important to use some notations that allow the interpretation of the file content. This is one of the main reasons for the creation of a Multipurpose Internet Mail Extensions (MIME) standard. The purpose for creating this standard was initially to identify multimedia contents and to support non-ASCII text characters. However, the degree of specification that such formalism provides globally across all data can be too general for a specific domain. In specialized domains as in the case of natural language processing, indication that a file contains text does not add very much information. In the absence of standard mechanisms, the differentiation between file formats containing various information is made through the use of various conventional names or multiple file extensions. Let's take a look at a few historical cases: i) The ATLAS XML files were traditionally named with an `.aif.xml` file extension; ii) The original *merged* representation of the Penn Treebank parses were stored in files with extension `.mrg`; iii) The CoNLL shared task tabular format used a `.conll` extension. The variability introduced using (sometimes) arbitrary naming conventions flies in the face of the concept of *namespaces* with a likely origin in programming language literature, but significant enough to have been exported in other fields of study such as computer networking, *area codes* and *country codes* in phone numbers, *zip codes*, etc. The cases are too numerous and common place to need further justification. However, so far as we are aware, there has not been a way of specifying important information of the quality, source, version, etc. of information found in various annotation files—either gold standard or manual. Typically a file containing a syntactic parse is commonly named with the `.parse` or `.tree` extension. There was a time when the landscape of parses was limited to Penn Treebank parses, and a few more bits of information was enough to disambiguate the contents for the end user.

It could be a predicted parse, or a gold standard parse; a constituent parse, or a dependency parses, etc. Even if one knows the answers to these questions the precise provenance might be impossible to trace as it could be predicted parse using a specific version of a specific parser trained on a specific corpus and which (as is traditionally the case with off-the-shelf parses) was trained on parses after removing *empty category* nodes from them.

We propose the use of a system of *tags* and such taxonomy itself can be grouped under the meta tag (prefix) "NLP-" to form a category of MIME types called NLP-MIME and possibly ASR-MIME for speech data and CV-MIME for vision data. Most of the data represented in other modalities such as image, videos, etc. tends to be containers of binary data, and the mechanism that has been in use for decades is by creating a plethora of file types such as `.wav, .au, .mp3` for audio, and `.png, jpeg`, etc. for images, and so on and so forth. A common solution for such content was the specification of a header at the beginning of the file which conveyed the salient invariants for that representation. For example, a `.wav` file would have a header specifying the sampling rate to be 16K, a bit precision of 16-bit and containing a single channel. We propose

a content hash-like framework of 10 character hashes which can capture the important characteristic of a file, say a `.parse` file.

We will use an extension of signature `.uughtzzzzz_parse` to indicate exactly what kind of parse it contains. In this case we use a ten character coding scheme where the first two characters map to a table indicating the source file—if any. And the following few indicate the value of one specific property each as shown below:

01 (word formed using first two characters **[0,1]**)
This is reserved for the tag for the file that was used as the source file and was transformed—either automatically or manually—to form the current version. If this is the source file, then these have a special value of uu. Letter u standing for *unset*.

2 (character at index **2**)
Whether the parse is a gold standard (**g** or an automatic **c, C, d**

- **g** Gold standard Penn Treebank parse
- **c** Output of Charniak Parser
- **b** Output of Berkeley Parser
- . ...

3 (character at index **3**)
How the hyphenization was represented in the schema for these parses

- **h** Tokens split at most hyphens (e.g., Treebank parses using guidelines version ...)
- **s** Tokens split at some hyphens (e.g., an intermediate inconsistent version)
- **n** Tokens split at none of the hyphens (e.g., the original Treebank v3 parses)
- . ...

4 (character at index **4**)
Whether the parse used the NML phrase tag which was added in later versions of Treebank guidelines

- **t** Yes, it did
- **f** No, it did not.

...
...

If one devises a reasonable strategy of creating such tags, and once the crucial properties of the contents are specified in the six character tag, then the user of the data can make several sensible assumptions about the contents. In fact, when there is a one-to-one conversion between two such tags, it can be used in a build system that would provide various guarantee—whether the verifiable features in the contents match the tag; exactly what function or transformations one needs to apply to a source tag to generate the contents for another tag, likely within the same layer. Such a system can substantially reduce the cognitive overhead on uses of the system and also allow modular functions to be written that only rely on the specific localized information for a particular layer.

If it feels like creating a whole category of new MIME types is going too far, then one might want consider the creation of *molecular file formats*[12] which includes a `chemical/MIME` specification.

Line No.	A Typical Editor	Emacs
1	`# coding=utf-8`	`#!/usr/bin/python`
2		`# -*- coding: utf-8 -*-`

Figure 1: An established convention (from the early days of UNIX, and further expanded over time) for adding useful metadata on the first two lines of source code.

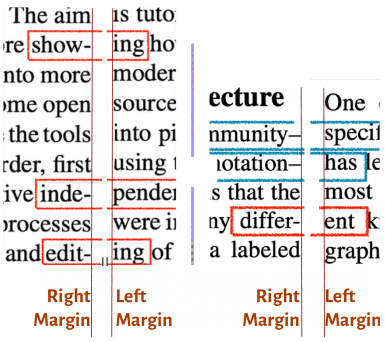

Figure 2: Example of hyphenation near right margin in a typical typeset document. There are two parts or columns in this figure. The left column is a snapshot of portion of text adjacent to the right page margin. And the right column shows the part that is adjacent to the left margin and on the following line. Three out of nine lines have these hyphenation artifacts marking the continuation of words on the following line.

A case for magic comments

We recommend the use of magic comments to provide more detailed information, potentially richer and complimentary to the tag classes. One of the practices or conventions that goes long back in time is the concept known as *shebang*, which would be immediately recognized by UNIX/GNU-Linux users as the interpreter that should be involved to run the contents of that file as a *script* provided the *executable bit* is set for that file. This concept, which can be considered somewhat akin to the headers in binary files, has found its way into being used for many other scenarios—one of them being the specification of the text encoding used in a file as part of the UNICODE standard. A few bits of (visually) *invisible* sequence of bytes, called the Byte Order Marker[13] (BOM) is used to indicate the specific encoding used by a text file. The mechanics in various situations are complex and described in the UNICODE standard. The same design principle was used in other systems such as by the Python programming language to indicate the text encoding of the source code used in a Python script.

It matters where and in what form data originates

NLP is a relatively new field that has seen an explosion in interest over the past several years. Most researchers made a very simplifying assumption that source text is born as tokens. This recently raised interesting issues leading to the introduction of shared tasks that started with raw text.

[12] https://en.wikipedia.org/wiki/Chemical_file_format
[13] https://www.unicode.org/faq/utf_bom.html#BOM

Raw text in its unsegmented/untokenized form is still not usually the root source of the text. Much of the text that is part of the word layer of various corpora typically is in the form of some markup which is interpreted by the end application and is not visible on the interface. To take a few examples, most PubMed articles are available as a XML documents that adhere to a specific schema. This text can contain various details such as emphasis markers, subscripts, superscript, formula, tables, etc. However, most annotation projects strip that out while preparing data for annotation. A result of this is that the consuming learning algorithm or system often does not have access to all the information encoded in the source document. This can be good in some cases but in many cases it results in the prediction algorithm having to re-learn structure and properties of the text which it could have otherwise used to learn useful patterns. Recent iterations of CoNLL-U files have started keeping such information in the headers.
Figure 2 demonstrates another special case that is typically encountered when annotating scanned text. When scanned text is used as a source of annotation, it is a typical practice to *clean up* such artifacts. However that results in loss of useful information.
What we propose to do is keep track of all such information in a way that it can be cleaned when necessary, but can also be accessible to the learning algorithms.

6.2.2. Appropriate Level of Functional Decomposition

We use the *command design pattern* that has been central to generations of version control systems, but was likely made popular, and has been expanded by the `git` version control tool. This allows for a creation of tools that focus on several top-level domain decomposed (potentially hierarchically) reasonably independently of each other while ensuring that the resulting artifacts can be aggregated to form a consistent whole. One can design custom workflows that take into consideration the nature of segmentation of a typical user base such as the annotator, the data consumer (e.g., for training machine learning models), a linguist, a phonetician, the schema designer (with less programming expertise), the power user (who can write new plugins and custom workflows), to name a few. For the power user the architecture allows for creating and storing frequently used or infrequent but complex stages of data transformations or searches that can be executed easily later.

6.2.3. Built on top of Giants—Emacs, Emacs-Lisp, Org-mode, Babel, etc.

We decided to use the time tested decisions on representing text and other media that went into the design of the **programmable editor**—Emacs[14]. It is important to clarify that the architecture is not tied to the Emacs editor. Emacs-Lisp[15] (Monnier and Sperber, 2020) is a dialect of lisp with a special focus on programmatic text editing capabilities. Its extensive documentation[16] details the numerous text and data encoding decisions that were made with its evolution and which we can simply adopt.
We try to highlight some aspects of emacs-lisp that are of particular import in this context:

- **Homoiconicity**—This is a fortunate side effect of using a lisp dialect. The ability to treat data and code interchangeably can be very powerful.
- **Extreme Configurability**—As part of its core design principle, the data represented and processed using emacs-lisp is extremely configurable. There are several layers of configurability that can be a very powerful tool.
- **Hooks**—One of the fundamental design principles which also is its strength is the care taken in capturing various events that alter states of data and which allow for insertion of *hooks* that get automatically triggered helping one create an automated way of describing and ensuring validity of constraints.

Another important component is the *orgmode*[17] library that can be used to represent *active documents* which is touted to be a great format/library for conducting reproducible research(Schulte and Davison, 2011). It's core functionality can provide many features that could a general framework such as this one. Given the space limitations, We will highlight a few functionalities that directly contribute towards our goal.

- **Rich Document Representation**—orgmode is sometimes referred to as Org Document as one can consider it to be a set of tools to create rich, structured documents.
- **(Programmable) Structured Editing**—It is a kind-of markdown language that is designed with structured editing in mind
- **Rich API**—The `org-element.el` library provides a rich set of functions and allows for customizing connections between various pieces of information through a mechanism of mixing and matching (hierarchical) inheritance of information (properties or key-value pairs) with a hierarchical *tag* structure that can provide an immensely powerful representation of information.
- **Rich Set of Plugins**—It has a very rich set of plugins that provide a rich collection of search and filtering libraries that can be used to search and manipulate the data structures.
- **Literate Programming (and therefore Literate Annotation)**—Another sub-ecosystem of plugins are based on the babel library (another important component of the emacs ecosystem.) This combination opens up potential for a practice of *literate annotation* where one can potentially *annotate*

[14] `https://emacs.org`
[15] `https://cemantix.org/links/emacs-lisp.html`
[16] The emacs-lisp manual is very comprehensive spanning some 1200 pages and regularly maintained.
[17] `https://orgmode.org`

Rich Transcript	`Um {lipsmack} and that's it. {laugh}`
Input to Syntactic Parser	`Um and that 's it .`

Output of the Parser	`(S (INTJ (UH Um))`
	`(CC and)`
	`(NP (DT that))`
	`(VP (VBZ 's) (NP (PRP it)))`
	`(. .))`

Figure 3: Level of information from a syntactic parse as expected by a syntactician.

annotations—among other capabilities.

We refer to the specification using a recursive acronym YAMR[18]—YAMR Ain't Meaning Representation.

6.2.4. Relational Data Model—The Force is Still Strong

Separation of concerns is another important design decision that plays a part in this architecture. One can incorporate local constraints easily through the use of `hooks` but complex constraints are best delegated to a database engine using a database schema. There was a time when NoSQL database seemed to promise the world, but history has shown that an absence of schema does not make schema go away. It just reappears in places where it is not convenient to maintain and share. The move by Google engineers to switch Spanner, their distributed database, from NoSQL to SQL (Bacon et al., 2017) architecture is a good indicator that relational models still have their place in the world of distributed computing.

6.3. Illustrating the Architecture

We will try to illustrate the richness and flexibility of our architecture using two short sentences.

6.3.1. An Example Task

Let's assume that a syntactician would like to parse the utterance shown on the first line of Figure 3 (among many others). In its most raw form, this string reflects the guidelines used for transcription that marks *non-word* sounds in curly braces as shown. Most of the off-the-shelf parsers are not trained on such special tokens representing *non-word*s. Let's assume that the parser expects pre-tokenized text as input. The second line shows a cleaned, tokenized string that can be fed to the parser. Following that is its likely parse.

Let's say that a phonetician would like to take a closer look at the relation between syntax and the duration of words, pauses, etc. In order to accomplish that, first the utterance needs to be processed through a *forced alignment* routine that tries to align audio segments to words, non-words in the transcript and also pauses that are longer than a certain threshold. A typical aligner does not care about the *punctuation's* in the input, and some even expect that to be part of the data *clean up* process. The forced aligner produces two new *layers* of information—a sequence of *timed words* and a *phone-level word alignment* of the audio with respect to the transcript. The next step is to integrate the *syntax* layer with the *phone-level alignment*. Figure 4 depicts the same transcript but in a tabular form.

[18] `https://yamr.org`

For the purposes of this discussion the top level header represents four *layers* of information. The caption describes the notations used in the table. A closer look at the table tells us that the alignment of sequence of *symbols* across the layers is not quite straight forward. Especially the fact that the *time aligned* words layer does not insert an `sp` (pause duration) marker when the duration is below a certain threshold. It should also be noted that the *phone-level alignment* layer uses a different scale than the use of *seconds* by the *timed word* layer. It is not hard to convince the reader that some non-trivial mechanism needs to be in place for one to integrate the syntax layer in this richer layer of information. One could write one time script to deal with a particular set of examples, but as far as we know there is no good general purpose solution available to anyone wanting to do such analysis.

The set of tools that we provide makes it easier for one to address such cases only by filling in some configurational parameters, such as the fact that `{LG}` and `{laugh}` are to be considered equivalent. In the worst case one might have to write a few lines of custom code if all such cases are not addressable using the configuration.

6.3.2. A Serialized Representation

In this section we will try to describe and illustrate how the serialization of the information looks like through snapshots of the file view. One important piece to know about `orgmode` is that it started as outline mode and so there is an innate ability (when opened in `emacs`) of folding (or, hiding) various levels of hierarchical information under a specific *node* (called *entry* or *headline*) in the `orgmode` lingua. Figure 5 shows a small cross section of a file with a top-level *Session* node and its first child that is the first *utterance*. The utterance itself has multiple children nodes. The child nodes are represent either a *word*, a *token*, a *whitespace* or a *metanode*. Whitespace is shown as a single underscore (`_`). These are identified using *tags* in the `orgmode` lingua which is a alphanumeric sequence within two colons (`:`). When a word and token are the same, the node can have both tags `:word:token:`. Tags can be concatenated to one other to indicate a set.

This structure somewhat represents a variation of an abstract syntax tree. The nodes that are tagged as `:metanode:` have an associated operator. In this case it is shows as `[0R]` and `[AND]` They are used when traversing the tree to get a sequence of tokens that match the users requirement. For example the three nodes (`I`, `_`, `'m`) that are children of `[AND]` will all be selected during the traversal provided each of them represents a specific signature—in this case a `:token:` tag—if one is trying to read the tokenized version of the utterance.

One of the powerful features of the *orgmode* data structure is that one can assign arbitrary number of (potential *hierarchy* of) *tags* AND *property*, *name-value* tuples and configure their *inheritability*. A node with many associated properties is shown in Figure 6. We

Tokens				Timed Words			Phone Alignment					Transcript
Token (index)	Start (char)	End (char)	Word	Start Time (seconds)	End Time (seconds)	Word	Start (sample)	End (sample)	Phone	Score	Word	
0	0	2	Um	30.082	30.952	um	300700000	302300000	AH1	-597.189941	UM	um
							302300000	309400000	M	517.626770		
	2	3	sp			sp	309400000	309400000	sp	-0.156736	sp	
			{LS}	30.952	31.312	{LS}	309400000	313000000	ls	-896.665771	{LS}	{lipsmack}
			sp	31.312	31.502	sp	313000000	314900000	sp	-76.367462	sp	
0	3	6	and	31.502	32.102	and	314900000	316300000	AE1	-131.814972	AND	and
							316300000	320600000	N	248.685242		
							320600000	320900000	D	-30.130604		
	6	7	sp			sp	320900000	320900000	sp	-0.156736	sp	
1	7	11	that	32.102	32.402	that's	320900000	321700000	DH	-34.681316	THAT'S	that's
2	11	13	's				321700000	322000000	AE1	-109.149544		
							322000000	322500000	T	-150.791061		
							322500000	323900000	S	45.310963		
	13	14	sp	32.402	32.702	sp	323900000	326900000	sp	-517.184387	sp	
3	14	16	it	32.702	33.182	it.	326900000	328400000	AH0	-507.279114	IT	it.
4	16	17	.				328400000	331700000	T	-731.196716		
			sp			sp	331700000	331700000	sp	-0.156736	sp	
			{LG}	33.182	33.692	{LG}	331700000	336800000	lg	-160.172989	{LG}	
				33.692	33.782	sp	336800000	337700000	sp	1.512673	sp	

Figure 4: Further details (in addition to the syntactic parse) expected by a phonetician. `sp` represents tokens identifying space character. These are explicitly marked in the output of the aligner; {`LS`} and {`LG`} respectively are equivalent to {`lipsmack`} and {`laugh`} as understood by the aligner. Greyed out tokens represent that they are missing from that layer. We have not identified space characters in the transcript column as they are typically invisible to the eye.

```
* Session              :session:...
  * What I'm I'm tr- # telling now. :utterance:...
    * What             :word:token:...
    * _                :whitespace:...
    * [OR]             :metanode:...
      * [AND]          :metanode:...
        * I'm          :word:...
      * [AND]          :metanode:...
        * I            :token:...
        * _            :whitespace:...
        * 'm           :token:...
    * tr-              :word:token:...
    * _
    * #                :word:token:...
    * _
    * [OR]             :metanode:...
      * [AND]          :metanode:...
        * I'm          :word:...
        * _
      * [AND]          :metanode:...
        * I            :token:...
        * _
        * 'm           :token:...
        * _
    * telling          :word:token:...
    * _
    * [OR]             :metanode:
      * [AND]          :metanode:
        * now.         :word:...
      * [AND]          :metanode:
        * now          :token:...
        * _
        * .            :token:...
```

Figure 5: The abstract tree representation of an utterance with both the raw and tokenized versions available using appropriate means of traversal.

have added some comments (not part of the org syntax) along with the properties so that the reader can better interpret their meaning. This framework makes use of the concept that hypergraphs are better than just graphs for modeling relational data (Wolf et al., 2016).

Finally, Figure 7 shows how one can switch to a tabular view where a selected group of properties are displayed as columns and are editable similar to a spreadsheet. Lack of space prohibits us to go into much details but one can customize the values for a given property to belong to a certain list of values which makes it easier to change them while adhering to those constraints.

In essence this forms a stream of rich nodes interconnected to form *hypergraphs* where a *hyperedges* can represent the sequence of nodes satisfying a specific use case *without destroying* its *relation with other annotations, data and metadata*, thus allowing one to potentially re-insert a transformed/enriched version of those nodes into the (richer) consistent whole.

7. Conclusion

In this article we have outlined a novel architecture that uses established tools and technologies as well as various time tested design principles that could streamline and simplify the management of multiple layers of annotations across various media while keeping the barrier to entry quite minimal without sacrificing future extensibility while allowing multiple versions of data and annotations to stay alongside each other and allow easy export of meaningful slices of the whole that are of interest to the end user.

8. Acknowledgements

We would like to thank Prof. Mitch Marcus for providing invaluable feedback on various design decisions. He suggested the acronym GRAIL connecting this work with the Treebank parser evaluation metric—PARSEVAL[19]. Thanks also to Dr. Richard Stallman (`rms`), founder of the Free Software Foundation, Chief GNUisance of the GNU Project for creating GNU Emacs, one of the oldest, extensible *free* software still under active development. This work builds on its many, carefully designed data structures and rich design concepts. Numerous discussions with him influenced the design of this representation.

9. Bibliographical References

Bacon, D. F., Bales, N., Bruno, N., Cooper, B. F., Dickinson, A., Fikes, A., Fraser, C., Gubarev, A., Joshi, M., Kogan, E., et al. (2017). Spanner: Becoming a sql system. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 331–343.

[19]Percival was one of the Grail knights in numerous medieval and modern stories of the Grail quest.

```
* Session                                                    :session:...
  * What I'm I'm tr- # telling now.                          :uid_00:utterance:...
    * What                                                   :word:token:
      :PROPERTIES:
      :IS_WORD:                  t               # t/f     -- Whether node is a span of type WORD
      :IS_TOKEN:                 t               # t/f     -- Whether node is a span of type TOKEN
      :WORD_INDEX:               0               # int     -- Value of  word index if node is a WORD
      :TOKEN_INDEX:              0:0             # int:int -- Value of token index if node is a TOKEN
      :IS_WHITESPACE:            f               # t/f     -- Whether WORD (or, TOKEN) represents whitespace (ws)
      :WHITESPACE_VALUE:         -               # string  -- Type of WHITESPACE (ws) represented by span
      :NODE_ID:                  nid_02          # string  -- Unique ID of the node
      :NODE_TYPE:                0               # int     -- Type of node: 0) non_ws; 1) ws; 2) inserted_ws; 3) or_...
      :NODE_DESCRIPTION:         non_whitespace  # string  -- Description of the node type
      :SPAN_START:               0               # int     -- Value of start character offset for WORD (or TOKEN)
      :SPAN_END:                 4               # int     -- Value of   end character offset for WORD (or TOKEN)
      :INCOMPLETE_WORD:          f               # t/f     -- Whether WORD (or, TOKEN) is incomplete (e.g., partially spoken)
      :RESTART_MARKER:           f               # t/f     -- Whether WORD (or, TOKEN) is a marks a resetart event
      :PARSE_IGNORE:             f               # t/f     -- Whether WORD (or, TOKEN) should be ignored during parsing
      :IS_REPEATED:              f               # t/f     -- Whether WORD (or, TOKEN) is repeated
      :IS_HIDDEN:                t               # t/f     -- Whether WORD (or, TOKEN) should be hidden from the view by default
      :CAN_HAVE_DURATION:        t               # t/f     --     Can WORD (or, TOKEN) has a time duration
      :HAS_DURATION:             t               # t/f     --    Does WORD (or, TOKEN) have a time duration
      :START_TIME:               5.01            # float   -- Start time in fraction of seconds
      :END_TIME:                 7.56            # float   --   End time in fraction of seconds
      :COMPUTED:                 f               # t/f     -- Whether the value of this node was computed or present in the source
      :END:
    * _                                                      :whitespace:
      :PROPERTIES:
      :IS_WHITESPACE:            t
      :WHITESPACE_VALUE:         " "
      :NODE_ID:                  nid_03
      :NODE_TYPE:                1
      :NODE_DESCRIPTION:         whitespace
      :CAN_HAVE_DURATION:        t
      :HAS_DURATION:             f
```

Figure 6: The properties associated with one particular node—the word "What"

```
ITEM                          | NODE_ID | NODE_TYPE | NODE_DESCRIPTION    | WORD_INDEX | TOKEN_INDEX | INCOMPLETE_WORD | IS_REPEATED | ALLTAGS
* Session                     | nid_00  | 0         | non_whitespace      | .          | .           |                 |             | :session:
 * What I'm I'm tr- # tellin.. | nid_01  | 0         | non_whitespace      | .          | .           |                 |             | :uid_00:utterance:
  * What                      | nid_02  | 0         | non_whitespace      | 0          | 0:0         |                 |             | :uid_00:word:token:
  * _                         | nid_03  | 1         | whitespace          | .          | .           |                 |             | :uid_00:whitespace:
  * [OR]                     | nid_04  | 3         | metanode_or         | .          | .           |                 |             | :uid_00:metanode:
   * [AND]                   | nid_05  | 4         | metanode_and        | .          | .           |                 |             | :uid_00:metanode:
    * I'm                    | nid_06  | 0         | non_whitespace      | 1          | .           |                 |             | :uid_00:metanode:word:
   * [AND]                   | nid_07  | 3         | metanode_and        | .          | .           |                 |             | :uid_00:metanode:
    * I                      | nid_08  | 0         | non_whitespace      | .          | 1:0         |                 |             | :uid_00:metanode:token:
    * _                      | nid_09  | 3         | inserted_whitespace | .          | .           |                 |             | :uid_00:metanode:whitespace:
    * 'm                     | nid_10  | 0         | non_whitespace      | .          | 1:1         |                 |             | :uid_00:metanode:token:
  * [OR]                     | nid_11  | 3         | or_metanode         | .          | .           |                 |             | :uid_00:metanode:
   * [AND]                   | nid_12  | 4         | and_metanode        | .          | .           |                 |             | :uid_00:metanode:
    * I'm                    | nid_13  | 0         | non_whitespace      | 2          | .           |                 | t           | :uid_00:metanode:word:
   * [AND]                   | nid_14  | 4         | and_metanode        | .          | .           |                 |             | :uid_00:metanode:
    * I                      | nid_15  | 0         | non_whitespace      | .          | 2:0         |                 | t           | :uid_00:metanode:token:
    * _                      | nid_16  | 2         | inserted_whitespace | .          | .           |                 | t           | :uid_00:metanode:whitespace:
    * 'm                     | nid_17  | 0         | non_whitespace      | .          | 2:1         |                 | t           | :uid_00:metanode:token:
  * tr-                      | nid_18  | 0         | non_whitespace      | 3          | 3:0         | t               |             | :uid_00:word:token:
  * _                         | nid_19  | 3         | inserted_whitespace | .          | .           |                 |             | :uid_00:whitespace:
  * #                         | nid_20  | 0         | non_whitespace      | 4          | 4:0         |                 |             | :uid_00:word:token:
  * _                         | nid_21  | 1         | whitespace          | .          | .           |                 |             | :uid_00:whitespace:
  * _                         | nid_22  | 1         | whitespace          | .          | .           |                 |             | :uid_00:whitespace:
  * telling                   | nid_23  | 0         | non_whitespace      | 5          | 5:0         |                 |             | :uid_00:word:token:
  * _                         | nid_24  | 1         | whitespace          | .          | .           |                 |             | :uid_00:whitespace:
  * [OR]                     | nid_25  | 3         | and_metanode        | .          | .           |                 |             | :uid_00:metanode:
   * [AND]                   | nid_26  | 4         | and_metanode        | .          | .           |                 |             | :uid_00:metanode:
    * now.                   | nid_27  | 0         | non_whitespace      | 6          | .           |                 |             | :uid_00:metanode:word:
   * [AND]                   | nid_28  | 4         | and_metanode        | .          | .           |                 |             | :uid_00:metanode:
    * now                    | nid_29  | 0         | non_whitespace      | .          | 6:0         |                 |             | :uid_00:metanode:token:
    * _                      | nid_30  | 3         | inserted_whitespace | .          | .           |                 |             | :uid_00:metanode:whitespace:
    * .                      | nid_31  | 0         | non_whitespace      | .          | 6:1         |                 |             | :uid_00:metanode:token:
```

Figure 7: The `column view` which allows a columnar representation of the nodes and properties. A hyperedge with set of node IDs 02, [03,] 06, [22,] 23, [24,] 27 represent the untokenized sentence and 02, [03,] 08, [09,] 10, [22,] 23, [24,] 29, [30,] 31 represent the tokenized version. The node IDs inside square brackets represent the whitespace nodes.

Banarescu, L., Bonial, C., Cai, S., Georgescu, M., Griffitt, K., Hermjakob, U., Knight, K., Koehn, P., Palmer, M., and Schneider, N. (2013). Abstract meaning representation for sembanking. In *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*, pages 178–186.

Bateman, J. A., Kasper, R. T., Moore, J. D., and Whitney, R. A. (1990). A general organization of knowledge for natural language processing: the penman upper model. Technical report, Technical report, USC/Information Sciences Institute, Marina del Rey, CA.

Bateman, J. A. (1990). Upper modeling: Organizing knowledge for natural language processing. Technical report, University of Southern California Marina del Rey Information Sciences Institute.

Bird, S. and Liberman, M. (2001). A formal framework for linguistic annotation. *Speech Commun.*, 33(1-2):23–60.

Bird, S., Day, D., Garofolo, J. S., Henderson, J., Laprun, C., and Liberman, M. (2000). ATLAS: A flexible and extensible architecture for linguistic annotation. In *Proceedings of the Second International Conference on Language Resources and Evaluation, LREC 2000, 31 May - June 2, 2000, Athens, Greece*.

Boersma, P. et al. (2014). The use of praat in corpus research. *The Oxford handbook of corpus phonology*, pages 342–360.

Calhoun, S., Carletta, J., Brenier, J. M., Mayo, N., Jurafsky, D., Steedman, M., and Beaver, D. (2010). The nxt-format switchboard corpus: a rich resource for investigating the syntax, semantics, pragmatics and prosody of dialogue. *Language resources and evaluation*, 44(4):387–419.

Carletta, J., Evert, S., Heid, U., and Kilgour, J. (2005). The nite xml toolkit: data model and query language. *Language resources and evaluation*, 39(4):313–334.

Chiarcos, C., Ritz, J., and Stede, M. (2012). By all these lovely tokens... merging conflicting tokenizations. *Language resources and evaluation*, 46(1):53–74.

Cunningham, H. (2002). Gate, a general architecture for text engineering. *Computers and the Humanities*, 36(2):223–254.

De Marneffe, M.-C., Manning, C. D., Nivre, J., and Zeman, D. (2021). Universal dependencies. *Computational linguistics*, 47(2):255–308.

Dipper, S., Götze, M., Stede, M., and Wegst, T. (2004). Annis. *Interdisciplinary studies on information structure: ISIS; working papers of the SFB 632*, (1):245–279.

Driscoll, J. R., Sarnak, N., Sleator, D. D., and Tarjan, R. E. (1989). Making data structures persistent. *Journal of computer and system sciences*, 38(1):86–124.

Ferrucci, D. and Lally, A. (2004). Uima: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348.

Götze, M. and Dipper, S. (2006). Annis: Complex multilevel annotations in a linguistic database. In *Proceedings of the 5th Workshop on NLP and XML (NLPXML-2006): Multi-Dimensional Markup in Natural Language Processing*.

Ide, N. and Pustejovsky, J. (2017). *Handbook of linguistic annotation*, volume 1. Springer.

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

Ide, N. and Suderman, K. (2007). Graf: A graph-based format for linguistic annotations. In *proceedings of the Linguistic Annotation Workshop*, pages 1–8.

Ide, N. and Suderman, K. (2014). The linguistic annotation framework: a standard for annotation interchange and merging. *Language Resources and Evaluation*, 48(3):395–418.

Ide, N. and Véronis, J. (1995). *Text encoding initiative: Background and contexts*, volume 29. Springer Science & Business Media.

Jochim, M. (2017). Extending the emu speech database management system: Cloud hosting, team collaboration, automatic revision control. In *INTERSPEECH*, pages 813–814.

Kasper, R. T. (1989). A flexible interface for linking applications to penman's sentence generator. In *Speech and Natural Language: Proceedings of a Workshop Held at Philadelphia, Pennsylvania, February 21-23, 1989*.

Krause, T. and Zeldes, A. (2016). Annis3: A new architecture for generic corpus query and visualization. *Digital Scholarship in the Humanities*, 31(1):118–139.

Kummerfeld, J. K. (2019). SLATE: A superlightweight annotation tool for experts. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 7–12, Florence, Italy, July. Association for Computational Linguistics.

MacWhinney, B. (2014). *The CHILDES project: Tools for analyzing talk, Volume II: The database*. Psychology Press.

Maeda, K., Lee, H., Medero, J., and Strassel, S. M. (2006). A new phase in annotation tool development at the linguistic data consortium: The evolution of the annotation graph toolkit. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation, LREC 2006, Genoa, Italy, May 22-28, 2006*, pages 1570–1573.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of english: The penn treebank. *Comput. Linguistics*, 19(2):313–330.

Monnier, S. and Sperber, M. (2020). Evolution of emacs lisp. *Proceedings of the ACM on Programming Languages*, 4(HOPL):1–55.

Narayanan, A. and Clark, J. (2017). Bitcoin's academic pedigree: The concept of cryptocurrencies is built from forgotten ideas in research literature. *Queue*, 15(4):20–49.

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

Pradhan, S. and Ramshaw, L. (2017). Ontonotes: Large scale multi-layer, multi-lingual, distributed annotation. In *Handbook of linguistic annotation*, pages 521–554. Springer.

Pradhan, S. S., Hovy, E., Marcus, M., Palmer, M., Ramshaw, L., and Weischedel, R. (2007). Ontonotes: A unified relational semantic representation. In *International Conference on Semantic Computing (ICSC 2007)*, pages 517–526. IEEE.

Preguiça, N. (2018). Conflict-free replicated data types: An overview. *arXiv preprint arXiv:1806.10254*.

Ramachandram, D. and Taylor, G. W. (2017). Deep multimodal learning: A survey on recent advances and trends. *IEEE signal processing magazine*, 34(6):96–108.

Ritchie, D. M. and Thompson, K. (1974). The unix time-sharing system. *Commun. ACM*, 17(7):365–375, jul.

Rosenfeld, V. (2010). An implementation of the annis 2 query language. *Berlin: Humboldt-Universität zu Berlin*.

Schmidt, T. (2004). Transcribing and annotating spoken language with exmaralda. In *Proceedings of the LREC-Workshop on XML based richly annotated corpora*, Lisbon.

Schulte, E. and Davison, D. (2011). Active documents with org-mode. *Computing in Science & Engineering*, 13(3):66–73.

Weischedel, R., Hovy, E., Marcus, M., Palmer, M., Ramshaw, L., Belvin, R., Pradhan, S., and Xue, N. (2011). Ontonotes: A large training corpus for enhanced processing. *Joseph Olive, Caitlin Christianson, andJohn McCary, editors, Handbook of Natural LanguageProcessing and Machine Translation: DARPA GlobalAutonomous Language Exploitation*.

Winkelmann, R. and Raess, G. (2014). Introducing a web application for labeling, visualizing speech and correcting derived speech signals. In *Proceedings of the Ninth International Conference on Language*

Resources and Evaluation (LREC'14), pages 4129–4133.

Winkelmann, R., Harrington, J., and Jänsch, K. (2017). Emu-sdms: Advanced speech database management and analysis in r. *Computer Speech & Language*, 45:392–410.

Wittenburg, P., Brugman, H., Russel, A., Klassmann, A., and Sloetjes, H. (2006). Elan: A professional framework for multimodality research. In *5th international conference on language resources and evaluation (LREC 2006)*, pages 1556–1559.

Wolf, M. M., Klinvex, A. M., and Dunlavy, D. M. (2016). Advantages to modeling relational data using hypergraphs versus graphs. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7. IEEE.

Zeldes, A., Lüdeling, A., Ritz, J., and Chiarcos, C. (2009). Annis: a search tool for multi-layer annotated corpora. In *Proceedings of Corpus Linguistics*.

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*, volume 632.