Off-Road LAF: Encoding and Processing Annotations in NLP Workflows

Emanuele Lapponi,[♣] Erik Velldal,[♣] Stephan Oepen,[♣♡] and Rune Lain Knudsen[♣♢]
[♣] University of Oslo, Department of Informatics
[♡] Potsdam University, Department of Linguistics
[♢] University of Bergen, Department of Linguistic, Literary, and Aesthetic Studies
{emanuel|erikve|oe|runelk}@ifi.uio.no

Abstract

The Linguistic Annotation Framework (LAF) provides an abstract data model for specifying interchange representations to ensure interoperability among different annotation formats. This paper describes an ongoing effort to adapt the LAF data model as the interchange representation in complex workflows as used in the Language Analysis Portal (LAP), an on-line and large-scale processing service that is developed as part of the Norwegian branch of the Common Language Resources and Technology Infrastructure (CLARIN) initiative. Unlike several related on-line processing environments, which predominantly instantiate a distributed architecture of web services, LAP achives scalability to potentially very large data volumes through integration with the Norwegian national e-Infrastructure, and in particular job sumission to a capacity compute cluster. This setup leads to tighter integration requirements and also calls for efficient, low-overhead communication of (intermediate) processing results with workflows. We meet these demands by coupling the LAF data model with a lean, non-redundant JSON-based interchange format and integration of an agile and performant NoSQL database, allowing parallel access from cluster nodes, as the central repository of linguistic annotation.

Keywords: Linguistic Annotation Framework, LAF, GrAF, annotation interchange, interoperability, portal, NoSQL database

1. Introduction

This paper describes an effort to adopt the data model of the Linguistic Annotation Framework (LAF; Ide & Romary, 2001; Ide & Suderman, 2013) for the interchange representation in NLP workflows. In general terms, a workflow will integrate separate language analysis components in potentially complex processing pipelines. A standard example of a workflow could be a sequence of tasks such as sentence segmentation, tokenization, part-of-speech tagging, syntactic parsing, and so on – requiring a collection of tools to be chained together in an inter-connected sequence.

We demonstrate how LAF can be used to represent the input and output data for each component in a workflow and observe that, within the general parameters of the data model, there are often degrees of freedom in the design of specific annotation layers. Seeing as our focus is on parameterizable, efficient, and scalable processing (rather than on static annotation for resource creation), this study evaluates the LAF approach from a new perspective, also discussing aspects of storage and retrieval efficiency in a large, homogeneous LAF database that supports parallel processing and full versioning of intermediate analysis layers.

The LAF encoded analysis workflows we describe are implemented in the Language Analysis Portal (LAP) currently being developed at the University of Oslo (UiO), as further presented in Section 2 below. After also reviewing LAF in some detail in Section 3, we discuss the concrete use and implementation of LAF in LAP in Section 4. Some related efforts are then presented in Section 5.

2. Background: Language Analysis Portal

The Language Analysis Portal (LAP; Lapponi, Velldal, Vazov, & Oepen, 2013b) is an ongoing initiative forming part of the CLARINO infrastructure project, the Norwegian branch of the pan-European CLARIN federation. The objective of LAP is to provide an easily accessible web interface that eliminates technological barriers to entry for non-expert users. At the same time, by integrating a wide range of NLP tools and transparent access to high-performance computing (HPC), the portal will enable execution of complex workflows and ensure scalability to very large data sets. A core component of the current pilot implementation is Galaxy (Giardine et al., 2005; Blankenberg et al., 2010; Goecks et al., 2010), a web-based workflow management system initially developed for data-intensive research in genomics and bioinformatics. In these fields, Galaxy portals allow biologists with no programming skills to access and configure processing tools, conduct experiments, and share both the results and the processing steps associated with them. By adapting Galaxy to the context of NLP, the vision of LAP is to ensure the same kind of access and ease of use of language technology tools for researchers from the humanities and the social sciences (and also for researchers within the field of NLP itself).

Rather than creating *ad-hoc* processing tools for LAP, existing NLP software is adapted and made available from within Galaxy. Note that rather than primarily employing web-services, LAP will offer a centralized repository of tools, built upon a backbone of a large high-performance computing cluster.[1] Through the simple and uniform Galaxy interface, a user will be able to combine the various tools into workflows, have them executed on the cluster, and retrieve the results. One important challenge here, however, concerns how to make diverse NLP tools work together seamlessly through workflows designed dynamically by the user. The large collection of pre-existing tools made available by the NLP community comes with

[1] Abel, the high performance computing facility at UiO, hosted by the USIT Research Computing group, is a powerful Linux cluster boasting more than 600 machines and totaling more than 10.000 cores (CPUs). For more information on the Abel cluster, see http://uio.no/hpc/abel/.

a wide range of different, mutually incompatible representation formats for encoding input and output data. If one wants to chain together different processing components that use different encoding formats, some way of translating between one and the other is required. To preserve the dynamic configuration functionality offered by Galaxy, LAP implements an interchange format that is used as a *lingua franca* to ensure interoperability between components. This means that the NLP tools integrated with LAP must be augmented with the capability of decoding from and encoding to this shared format. In an earlier development version of LAP (Lapponi et al., 2013a), we experimented with file-based interchange, where annotations produced by different tools would be accumulated in files formatted using various XML, JSON, or tabulated approaches. However, this proved not to be ideal both in terms of communication efficiency and the degree of expressivity provided by the representation. This paper documents the efforts to instead use a representation based on the Linguistic Annotation Framework, with annotations stored in a database for more scalable interchange.

3. The Linguistic Annotation Framework

LAF is a graph-based model for representing multi-modal linguistic annotations that aims at providing full interoperability among annotation formats. One of the fundamental principles that guided the development of LAF is that all annotation information should be explicitly represented, i.e., the intepretation of annotations should not require implicit knowledge about particular categories and relations (Ide & Suderman, 2013). Another fundamental principle is that one should observe a strict separation between annotation *structure* and annotation *content*. The focus of LAF is only on the structural part, and it is important to realize that LAF itself is not a format as such. It does not come with pre-specified linguistic labels or categories etc. Rather, it is a general framework for how to represent the annotation structure itself; an abstract data model specifying how to relate annotations to data and how to relate annotations to other annotations.

As a complement to the abstract model, the Graph Annotation Format (GrAF) is intended to be used as an XML serialization for the purposes of interchange (by specifying mappings to/from other existing formats) (Ide & Suderman, 2013). We further detail both the abstract data model and the serialization format below.

The LAF data model for annotations can be seen to comprise an acyclic directed graph decorated with feature structures (Ide & Suderman, 2013). More specifically, the LAF data model is comprised of three basic components;

- *Regions*: Standoff references to the media being annotated (further detailed below).
- *Graph Elements*: A graph structure consisting of nodes, edges and links to regions, organizing the structure and the relations of the linguistic information.
- *Annotations*: Feature structures containing the actual linguistic information, associated with nodes and edges.

```
<region xml:id="seg-r0" anchors="54 55"/>

<node xml:id="ptb-n00003">
   <link targets="seg-r0"/>
</node>

<a label="tok" ref="ptb-n00003" as="PTB">
   <fs>
      <f name="msd" value="DT"/>
   </fs>
</a>
```

Figure 1: GrAF serialization of part of a text in the ANC, representing token and part of speech annotations for the first word ("A") in one of the available texts.

Importantly, LAF falls under the category of *standoff annotation* (sometimes called *remote markup*), where the object of description and its annotations are not interspersed (in contrast to *inline* annotation). Regions are defined in terms of so-called *anchors* that reference locations in the primary data – the media being annotated. In the case of text, the regions are "anchored" in terms of character indices, for acoustic data it might be time intervals, and so. The LAF graph elements then simply link to these regions, leaving the primary data untouched.

As mentioned above, GrAF is an XML serialization of LAF. Although not many annotated resources are yet available as GrAF, one notable example is the Manually Annotated Sub-Corpus (MASC) of the American National Corpus (ANC), for which software is also made available to extract the annotations.[2] Figure 1 shows an excerpt of the ANC annotations serialized as GrAF, and provides an example of a possible approach to modeling relations among annotations using the LAF data model. The region element with id `seg-r0` references a token found starting at character position 54 and ending at 55; node `ptb-n00003` contains a link to said region and is referenced by a part-of-speech annotation (an `a` element).

LAF was recently approved as an ISO standard (ISO 24612:2012).[3] While having been under continuous development for about a decade, LAF in its current form is still a fairly new standard. This also means that there is not much to look to in terms of existing best practices when in comes to encoding various annotations in LAF or GrAF. This is particularly relevant given the somewhat underspecified nature of the data model. Being simply a general framework rather than a format, LAF leaves much room for variation in exactly how a given annotation should be represented. There is typically more than one LAF-compliant way to represent a given annotation. This observation is practically evidenced by the fact that we were not abel to parse the ANC files mentioned above using a standard GrAF API like the open-source Python Poio toolkit.[4]

Issues like these are what we turn to in the next section. Part of the mission of the paper will be to provide some

[2] `http://www.americannationalcorpus.org/.`

[3] `http://www.iso.org/iso/catalogue_detail.htm?csnumber=37326`

[4] `http://media.cidles.eu/poio.`

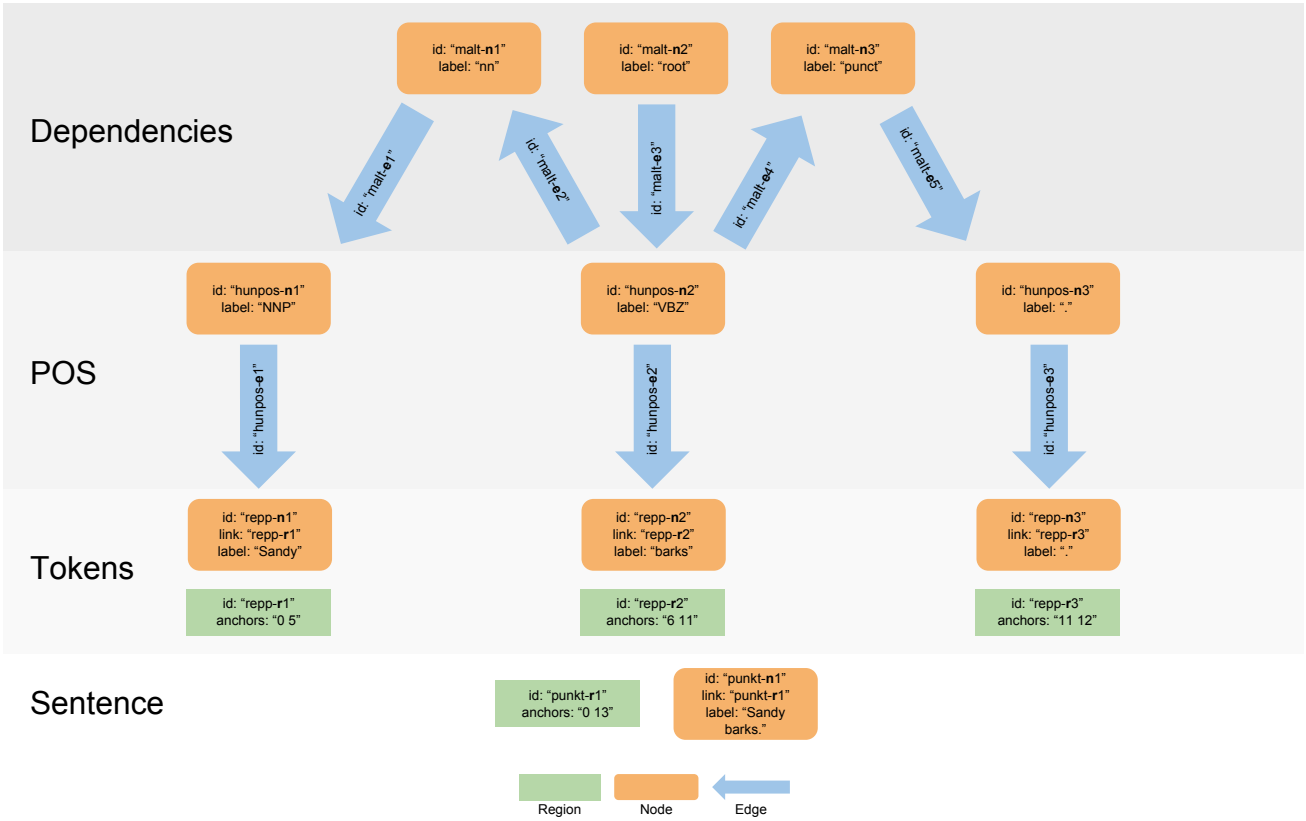

Figure 2: Diagram of LAP records representing a LAF graph for the annotation of a single example sentence; "Sandy barks." Unlike the ANC example, tokens are represented with their own node and annotations, while also segmenting the text and producing regions. The same goes for sentence annotations, which produce regions with their own anchors to character offsets. POS tag annotations are associated to their own nodes and point to their respective tokens by instantiating edges. Dependency relations are encapsulated in nodes, and head-dependent directionality is preserved within the LAF model by splitting dependency arches at node-points.

concrete examples of LAF used in practice – in particular emphasizing its use as an interchange representation in processing pipelines as opposed to a static, serialized annotation format.

4. LAF in LAP

In the current implementation of LAP, annotations are related to each other and to the original text using the LAF data model. Running a tool on some input (either text or LAP annotations, according to the tool dependencies) populates a NoSQL database with *records*. These are modeled after the core elements of a LAF graph and can be of three different types: region, node or edge. Since record instances are produced as a result of tool execution, their type and characteristics depend on those of the tool itself; e.g., tools that affect text segmentation output both regions and nodes.

Note that adding new annotations always means creating new graph elements, rather than editing and augmenting existing ones. While this approach results in graphs that are less compact than the ones seen in ANC, it also allows tools to take advantage of the graph structure when invoking input annotations, for instance in case only specific parts of the graph built by certain tools is required. Annotations produced by a given tool are associated to node- and edge-records created during execution in the form of feature structures: Rather than existing separately, they are encapsulated in the corresponding records and are retrieved as the graph is being traversed, in order to avoid unnecessary queries to the database. In this section we illustrate how graph elements are added to the database by working through the steps involved in an example use case, making comments on implementation and design decisions along the way.

A simple example of a use case for the system could be that of annotating running 'raw' text with dependency annotations, export them to some format and download them to analyze them or process them further. This can be achieved in LAP by creating a workflow consisting of tools that produce the annotations required by the dependency parser; minimally a sentence segmenter, a tokenizer and a POS tagger, in the case of MaltParser (Nivre et al., 2007). The example corpus for this hypothetical session consists of a single document containing the string "`Sandy barks.`". The first tool in the chain is the punkt sentence splitter (Kiss & Strunk, 2006), which acts both as a segmenter, producing regions (in this case, the output consists of the region `punkt-r1` with anchors 0 and 13), and as an annotator, instantiating nodes containing the output string

of the tool (punkt-n1, "Sandy barks."). Here, the output consists of the region punkt-r1 with anchors 0 and 13.

The second layer in the chain is the REPP tokenizer (Dridan & Oepen, 2012), which behaves similarly to the sentence splitter, producing both regions and nodes containing annotations. The information enclosed in the items produced by these first two annotators (visually represented in Figure 2) is different from that which we observed in Ide and Suderman (2013), where the anchors belonging to sentence regions consist of lists of token region ids; this approach assumes that the lists of token produced by different tokenizers comprise a set of "base segmentations" that are anchored to character offsets. This means that other region-producing nodes don't get access to the offsets.

While this is a viable solution for representing the annotations, we found that it was not ideal for retaining LAP's modular approach, as it binds the sentence annotations to the tokens, making it harder for another tokenizer to re-use the information provided by the sentence splitter. In LAP, both tokenizers and sentence segmenters are dependency-less tools, which means that they can be run at the beginning of a workflow; what sets them apart is that tokenizer can exploit the sentence information if it is available in the database. The REPP-wrapper simply assesses their availability (or whether the user explicitly required a specific sentence-splitting tool), calls a get_sorted_regions() function using the splitter id as a parameter and prepares the input for the tokenizer based on the sentence information. Tokenizers, working as segmenters as well as annotators, produce both regions and nodes. Nodes are annotated with references to the instantiated regions, as well as with feature structures with the token value (which may present a normalization of the underlying region, or not, depending on the tokenizer).

The third layer in the annotation chain is the HunPos part-of-speech tagger. Rather than associating new annotations to the token nodes, the tagger produces a new set of nodes, each of which linked to one token node through edge records. This approach is different from the one we observed in the ANC representations, where token annotations are implicitly expressed by the regions, and POS tags are associated to token-level nodes. Running multiple tokenizers and POS taggers on the same primary data adds new regions, nodes, and edges to the same pool of records available to LAP via the database, where different segmentations and alternative annotations thus can co-exist.

Edge-annotations adhere to the LAF specifications in Ide and Suderman (2013), who state that "[...] annotations on edges specify the structural role of the edge" rather than linguistic content. This affects our internal representation of dependency graphs, where dependency relations have their own node and the head-dependent directionality of the arches is preserved within the LAF model by splitting each arch in two edges. In the top layer of Figure 2, "barks" is the head of "Sandy" with relation "nn", so there is an edge malt-e2 going from node hunpos-n2 (the part-of-speech node for "barks") to node malt-n1 (which contains the dependency relation information) and an edge malt-e1 that goes from malt-n2 to hunpos-n1 (the part-of-speech node for "Sandy").

An example of the stored JSON representation, showing the database entries for REPP- and HunPos-nodes depicted in Figure 2, is provided in Figure 3. At the end of a workflow, users can select which annotations they wish to serialize and export them using the available modules in LAP, for example by formatting records from the database into ANC-style GrAF documents.

5. Related Efforts

In this section we will look briefly at how our approach of using LAF in the LAP context, compares to how related formats are put to use in related settings. We will review a number of interchange formats that are used in similar (albeit typically web service–based) environments for annotating text.

Travelling Object (TO2) is a stand-off annotation format used primarily in the Panacea project,[5] a web service–based platform for machine translation and general language technological applications. It is directly based on the GrAF serialization of LAF and follows the ANC approach. It represents documents with a set of files: one or more primary data documents, up to two segmentation files (sentences and tokens), a header document with metadata conforming to the CES format, and one or more annotation documents.

The Format for Linguistic Annotation (FoLiA) is the annotation format used in the Dutch CLARIN initiative for the TTNWW workflow platform,[6] among others. Unlike TO2, it is a mixed format and represents one annotated document as one XML file containing all the information. It is designed with human readability in mind, and supports a set of core annotation types using explicit XML tags. This set of annotation types are divided into four categories: Structure, token, span and subtoken. From the summary on the project web site, FoLiA has its focus more on expressivity rather than on computing efficiency, and is as such not ideal for real-time or resource-constrained applications.

The KYOTO Annotation Format (KAF) is the annotation format used in the collaborative Asia–Europe KYOTO project.[7] It too is inspired by the LAF paradigm, but represents annotation types the same way as FoLiA does, using explicitly named XML tags (e.g. <wf>, <term>, etc.). It can thus be said to be specialized, XML-based implementation of the LAF data model.

The NLP Interchange Format (NIF) stands out to a certain degree in this comparison due to its core reliance on the Resource Description Framework (RDF) and Web Ontology Language (OWL), i.e., so-called semantic technologies. As such, NIF (like LAF) makes a clear distinction between the underlying data model and a range of pre-existing, fully standardized serializations, including XML and the more compact Turtle. In our view, NIF is to a large degree complementary to both LAF and several of the related initiatives reviewed here: It puts strong emphasis on the complete standardization of its basic building blocks,

[5] http://www.panacea-lr.eu/.

[6] http://yago.meertens.knaw.nl/apache/TTNWW/

[7] http://adimen.si.ehu.es/~rigau/publications/gl09-kaf.pdf

for example the representation of arbitrary Unicode strings and addressing of substrings through Internationalized Resource Identifiers (IRI), interactions with HTTP content negotiation, or aspects of versioning and persistence. For the structure and labeling of linguistic annotation, on the other hand, NIF leans heavily on existing standards and vocabularies, including LAF.

The Text Corpus Format (TCF), finally, is used by the Weblicht workflow platform, which is a part of the German CLARIN ecosystem. It is perhaps the most verbose of all the interchange formats, containing explicitly named XML tags for the annotations it can represent. It does not refer to character positions in the source material, but rather makes use of references to token IDs when anchoring higher layers of annotations.

In LAP, we have chosen to focus primarily on defining an internal representation of the LAF model that lies very close to the database structure, rather than on some existing serialization format (e.g. GrAF, NIF etc.). Maintaining a strict adherence to the LAF paradigm, we have no restrictions on the type of annotations available since all annotations are represented as general features contained in feature sets for nodes or edges. The internal LAF representation is not specifically designed with readability in mind as we deem this to be the responsibility of the export functionality of LAP. Still, the extended JSON format used by MongoDB allows for a fairly transparent representation, making queries, modifications, and serialization transformations a viable task.

Each tool to be integrated needs a thin transformation layer/wrapper that makes use of the appropriate format converter to handle insertion into, and queries from, the database. While this requires the party responsible for integrating a tool to master the knowledge of LAP's inner database abstraction layer rather than a general serialization format, it enables us to transform any supported input/output format into any other. By treating LAF serialization formats in the same way as any other input/output format, LAP becomes somewhat agnostic with regards to the format of the source material, which greatly simplifies the process of treating data from a multitude of sources in a uniform manner. Every format subsumed by LAF will be implementable in LAP with a reasonable amount of work, as well as various ad-hoc formats implemented in NLP tools.

We have currently implemented import/export functionality for GrAF as defined in (Ide & Suderman, 2013), TCF, CoNNL/TSV data, the CG3 format, and the tabular format used for generating corpora in the IMS Corpus Workbench (CWB) toolset. This means that a user can upload e.g. a TCF-formatted document, run it through a processing workflow, and retrieve the results in the CWB format ready to be inserted into the corpus search tool.

Another practical difference between the LAP approach and these pre-existing interchange formats relates to the 'granularity' of inter-component communication: In the file-based TO2 and TCF format, annotations are accumulated into ever-growing documents that have to be (re-)parsed and (re-)serialized before and after each processing step. Thus, even when a tool only requires a certain type of annotation found in the document, it still needs to decipher *all* elements of the interchange file. With our approach, a tool can target specific record types by submitting queries to the database. This property is also practical in terms of parallelization, in that different jobs can invoke separate parts of the database, eliminating the need for wrestling with the bookkeeping of splitting and merging files.

6. Ongoing and Future Work

As observed in Section 1 above, existing tools need to be adapted and augmented with means to communicate with the LAP database, i.e., retrieve and decode annotations from the LAF graph and/or encode their output before adding it to the graph. A functional pilot of LAP with an integrated, interoperable ensemble of common NLP components for English and Norwegian and transparent access to the Norwegian national HPC eInfrastructure will be launched for public testing in the second quarter of 2014.

Acknowledgments

We are grateful to our colleagues at the Oslo Language Technology Group and at the Research Computing Services at the University of Oslo—in particular Hans A. Eide, Bjørn-Helge Mevik, Thierry Toutain, and Nikolay A. Vazov—for fruitful discussions and suggestions, as well as to three anonymous reviewers for insightful comments. Large-scale experimentation and engineering is made possible though access to the Abel high-performance computing facilities at the University of Oslo, and we are indebted to the staff of the University Center for Information Technology, to the Norwegian Metacenter for Computational Science, and to the Norwegian tax payer.

References

Blankenberg, D., Von Kuster, G., Coraor, N., Ananda, G., Lazarus, R., Mangan, M., ... Taylor, J. (2010). Galaxy. A web-based genome analysis tool for experimentalists. *Current Protocols in Molecular Biology*, 19.10.1 – 21.

Dridan, R., & Oepen, S. (2012). Tokenization. Returning to a long solved problem. A survey, contrastive experiment, recommendations, and toolkit. In *Proceedings of the 50th Meeting of the Association for Computational Linguistics* (p. 378 – 382). Jeju, Republic of Korea.

Giardine, B., Riemer, C., Hardison, R. C., Burhans, R., Elnitski, L., Shah, P., ... Nekrutenko, A. (2005). Galaxy. A platform for interactive large-scale genome analysis. *Genome Research*, *15*(10), 1451 – 5.

Goecks, J., Nekrutenko, A., Taylor, J., & Team, T. G. (2010). Galaxy. A comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, *11*(8:R86).

Ide, N., & Romary, L. (2001). A common framework for syntactic annotation. In *Proceedings of the 39th Meeting of the Association for Computational Linguistics* (p. 306 – 313). Toulouse, France. Retrieved from `http://www.aclweb.org/anthology/P01-1040` doi: 10.3115/1073012.1073052

Ide, N., & Suderman, K. (2013). The Linguistic Annotation Framework: A standard for annotation interchange and merging. *Language Resources and Evaluation*, *(forthcoming)*.

Kiss, T., & Strunk, J. (2006). Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, *32*(4), 485 – 525.

Language resource management – Linguistic Annotation Framework (LAF) (Norm No. ISO 24612:2012). (2012). ISO, Geneva, Switzerland.

Lapponi, E., Velldal, E., Vazov, N. A., & Oepen, S. (2013a). HPC-ready language analysis for human beings. In *Proceedings of the 19th Nordic Conference of Computational Linguistics* (p. 447 – 452). Oslo, Norway.

Lapponi, E., Velldal, E., Vazov, N. A., & Oepen, S. (2013b). Towards large-scale language analysis in the cloud. In *Proceedings of the 19th Nordic Conference of Computational Linguistics: Workshop on Nordic Language Research Infrastructure* (p. 1 – 10). Oslo, Norway.

Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryığıt, G., Kübler, S., ... Marsi, E. (2007). MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, *13*(2).

```
{
    "origin" : "repp",
    "index" : 0,
    "out_edges" : [ ],
    "links" : [
        "repp-r1"
    ],
    "annotation_spaces" : [
        "repp"
    ],
    "type" : "node",
    "id" : "repp-n1",
    "in_edges" : [
        "hunpos-e1"
    ],
    "annotations" : {
        "repp" : {
            "class" : "token",
            "label" : "Sandy"
        }
    }
}
{
    "origin" : "repp",
    "index" : 1,
    "out_edges" : [ ],
    "links" : [
        "repp-r2"
    ],
    "annotation_spaces" : [
        "repp"
    ],
    "type" : "node",
    "id" : "repp-n2",
    "in_edges" : [
        "hunpos-e2"
    ],
    "annotations" : {
        "repp" : {
            "class" : "token",
            "label" : "barks"
        }
    }
}
{
    "origin" : "repp",
    "index" : 2,
    "out_edges" : [ ],
    "links" : [
        "repp-r3"
    ],
    "annotation_spaces" : [
        "repp"
    ],
    "type" : "node",
    "id" : "repp-n3",
    "in_edges" : [
        "hunpos-e3"
    ],
    "annotations" : {
        "repp" : {
            "class" : "token",
            "label" : "."
        }
    }
}
```

```
{
    "origin" : "hunpos",
    "index" : 0,
    "links" : [ ],
    "annotation_spaces" : [
        "hunpos"
    ],
    "id" : "hunpos-n1",
    "out_edges" : [
        "hunpos-e1"
    ],
    "type" : "node",
    "annotations" : {
        "hunpos" : {
            "class" : "pos_tag",
            "label" : "NNP"
        }
    },
    "in_edges" : [
        "maltparser-e1"
    ]
}
{
    "origin" : "hunpos",
    "index" : 1,
    "links" : [ ],
    "annotation_spaces" : [
        "hunpos"
    ],
    "id" : "hunpos-n2",
    "out_edges" : [
        "hunpos-e2",
        "maltparser-e2"
    ],
    "type" : "node",
    "annotations" : {
        "hunpos" : {
            "class" : "pos_tag",
            "label" : "NNS"
        }
    },
    "in_edges" : [
        "maltparser-e3"
    ]
}
{
    "origin" : "hunpos",
    "index" : 2,
    "links" : [ ],
    "annotation_spaces" : [
        "hunpos"
    ],
    "id" : "hunpos-n3",
    "out_edges" : [
        "hunpos-e3"
    ],
    "type" : "node",
    "annotations" : {
        "hunpos" : {
            "class" : "pos_tag",
            "label" : "."
        }
    },
    "in_edges" : [
        "maltparser-e5"
    ]
}
```


Figure 3: JSON representations for the REPP- and HunPos-nodes (left and right respectively) in Figure 2.