CoNLL-RDF: Linked Corpora done in an NLP-friendly way

Christian Chiarcos and Christian Fäth

Goethe-University Frankfurt, Germany,
(chiarcos@informatik|faeth@em).uni-frankfurt.de,
WWW home page: http://acoli.informatik.uni-frankfurt.de

Abstract. We introduce *CoNLL-RDF*, a direct rendering of the *CoNLL* format in *RDF*, accompanied by a formatter whose output mimicks CoNLL's original TSV-style layout. CoNLL-RDF represents a middle ground that accounts for the needs of NLP specialists (easy to read, easy to parse, close to conventional representations), but that also facilitates LLOD integration by applying off-the-shelf Semantic Web technology to CoNLL corpora and annotations. The CoNLL-RDF infrastructure is published as open source. We also provide SPARQL update scripts for selected use cases as described in this paper.

1 Motivation and Background

Representing NLP resources and linguistic annotations as Linked Data has become an established technique in the context of both the development of the Linguistic Linked Open Data (LLOD) cloud [8] and the advances on Knowledge Extraction in Artificial Intelligence, resp. their adaptation in different fields of practice, such as, e.g., Biomedical IE. Yet, current formalisms usually applied for the purpose [17, 18, 12] find limited resonance in the core NLP community: RDF serializations are often seen as needlessly complex, possible benefits of Linked Data for Natural Language Processing (and of standards in general) are not widely recognized yet,[1] and NLP specialists are used to work with established de-facto standards for most aspects of NLP.

The tradition of shared tasks organized by the Conference of Natural Language Learning (CoNLL)[2] helped to establish such task-specific de-facto standards for core aspects of NLP and corpus linguistics. While these formats were not generic, they are easy to parse (or at least, commonly known) and supported by many tools and corpora. An NLP researcher may thus rightfully ask why we should exceed beyond CoNLL if it is technically well supported, easy to process and established in existing workflows – given the complexity of the alternatives proposed.

[1] As summarized by Mark Johnson in his ACL-IJCLNP 2012 keynote on the future of computational linguistics, "[s]tandard data formats (...) I'm not sure these are important: if someone can use a parser, they can probably also write a Python wrapper" [14, slide 8].

[2] http://www.signll.org/conll

We argue that there *is* added value in using RDF-based formalisms – namely in interoperability, interpretability, resource integration and infrastructural support [7], as well as eliminating the need for potentially error-prone transformations –, and that their apparent disadvantages can be compensated by establishing a middle ground between the CoNLL format family and RDF: *CoNLL-RDF* is a shallow, human-readable and easily processable, RDF compliant format which provides a compromise between established CoNLL representations used in NLP and more advanced and semantically well-defined formalisms brought forward in the context of Linguistic Linked Open Data [17, 18, 12]. CoNLL-RDF allows NLP researchers and engineers to seamlessly integrate corpus data with lexical resources, terminology bases and knowledge graphs.

We present the mapping of CoNLL to a lossless and isomorphic RDF representation and its canonical serialization. Taken together, both aspects, the RDF data model and the associated format conventions, define CoNLL-RDF. We describe infrastructure, use cases, related research and perspectives.

1.1 Turtle: A human-readable RDF serialization

RDF data can be serialized in and losslessly converted between different standard formats – depending on the intended use. As such, Turtle provides a human readable instantiation [2]. It represents RDF statements (labeled edges in the graph) as triples of source node ("subject"), edge ("property"/"relation") and target ("object"), concluded with a dot.

```
<http://example.org/sbj>
    <http://www.w3.org/2000/01/rdf-schema#subClassOf>
        <http://example.org/SomeClass> .
<http://example.org/sbj> <http://www.w3.org/2000/01/rdf-schema#label> "example"  .
<http://example.org/sbj> <http://www.w3.org/2000/01/rdf-schema#label> "Beispiel" .
```

Abbreviations are possible, e.g., using prefixes; also, if several triples refer to the same subject, the subject can be dropped if the triples are separated with ";". If they share the same subject and the same predicate, the objects can be enumerated in ","-separated sequences:

```
@prefix ex: <http://example.org/>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
ex:sbj rdfs:subClassOf ex:SomeClass ;
       rdfs:label      "example", "Beispiel" .
```

Whitespaces are irrelevant in Turtle, and the format allows to provide explanatory comments following #. Turtle is an attractive representation formalism because of its close relationship with the SPARQL query language [4]. With basic understanding of Turtle, users of CoNLL-RDF can thus directly apply SPARQL and SPARQL end points for querying, storing and manipulating annotations. Because of its flexibility, compactness, and readability we decided to design CoNLL-RDF as a sublanguage of Turtle with additional serialization constraints.

1.2 The CoNLL format family

Most CoNLL shared tasks used variations of tab-separated columns as format for testing and evaluation with one word per line, annotations represented as column values in these various lines, sentences being separated by empty lines, and comments marked by #. Over time, different column separators were used (1999-2001, 2004-2005 spaces, 2002-2003 single space), but ultimately settled on tabulators (since 2006, CoNLL-X).

CoNLL-X (2006)[3] introduced dependency annotations with every word being identified by its position in the sentence (explicit `ID` column), pointing to its syntactic `HEAD`, with a particular dependency relation assigned to the word (abbreviated `EDGE` here). In addition, columns for word-level annotations of `LEMMA` and morphosyntactic features (`FEAT`) were added.

```
# ID WORD    LEMMA POS  POS2 FEAT                HEAD EDGE  HEAD2 EDGE2
  1  Cathy   Cathy N    N    eigen|ev|neut        2    su    _     _
  2  zag     zie   V    V    trans|ovt|1of2of3|ev 0    ROOT  _     _
  3  hen     hen   Pron Pron per|3|mv|datofacc    2    obj1  _     _
  4  wild    wild  Adj  Adj  attr|stell|onverv    5    mod   _     _
  5  zwaaien zwaai N    N    soort|mv|neut        2    vc    _     _
  6  .       .     Punc Punc punt                5    punct _     _
```

Shared tasks in 2008 and 2009 combined this format with annotations for semantic roles, with one *additional column per frame instance* found in the sentence. This is an important extension of the original TSV format, because it means a departure from the tabular data model: Within a corpus, sentences differ in the number of columns. Subsequent tasks introduced further additions, e.g. additional columns for document and document parts (`DOCID`, `PART`), speaker identification (`AUTHOR`), and word sense (`WSENSE`), as well as a column for co-indexing coreferential referring expressions (`COREF`) throughout the text.[4]

2 CoNLL-RDF: LLOD with NLP-friendly flavor

CoNLL-RDF comprises the following components: A shallow and extensible RDF data model, its serialization(s), a core API for parsing, manipulating and formatting CoNLL (resp., CoNLL-RDF) and a library of exemplary SPARQL update scripts. It is designed to remain minimal, focusing on Un*x shell, but paving the way for more elaborate solutions in the future. In the following, we presume CoNLL to be represented by tab-separated values (TSV); we also concentrate in our presentation on the popular CoNLL-X format and its derivatives. We do, however, support all TSV dialects used for CoNLL Shared Tasks until 2015.

2.1 The CoNLL-RDF data model

The **original data model** of CoNLL is a sequence of variable-width tables, each representing a sentence, with rows in a fixed (sequential) order. Because of

[3] `http://ilk.uvt.nl/conll/`
[4] `http://conll.cemantix.org/2011/data.html`

the variable width, CoNLL is not directly translatable to a relational database format, it is, however, conceptually close. Aiming for a light-weight and loss-less conversion of CoNLL to RDF, we roughly follow a trivial mapping approach comparable to R2RML [10]: Map rows to individuals, columns to properties, values to literals; cross-references (foreign keys) receive special treatment and are mapped to object properties.

In order to provide a generic converter of CoNLL data, we must not rely on a fixed set or order of columns. Instead, given a **user-provided list of column labels** at conversion time, we create a datatype property for each of them. These novel properties are assigned to a designated `conll` namespace and take literals as their values. In case these need to be transformed into object properties, or their annotations are to be parsed (e.g., decoupling morphosyntactic features, originally parsed as compact string, only), this can be easily implemented by SPARQL update. However, as most instantiations of the CoNLL format since 2006 comprise dependency annotations, the `HEAD` column must receive special treatment, and will *always* be converted to an object property. This convention enables special handling of intra-sentential cross-references and is suitable for, but not restricted to dependency syntax. For CoNLL RDF formatting, the surface form of a token is expected in a column with the label `WORD` (original terminology; since 2006 `FORM`), and labels for intrasentential relations defined in `HEAD` are drawn from the column `EDGE`. No additional a priori naming conventions apply within the CoNLL-RDF core API. Individual SPARQL update scripts, however, are usually defined with reference to specific CoNLL-RDF properties, imposing task-specific constraints on portability and applicability.

Unlike TSV, RDF is a conceptual model and does not impose any **sequential structure**. This is, however, necessary to represent words and sentences in linguistic corpora. For this purpose, we rely on NIF [12]:[5] Every word (a CoNLL row) is identified by its URI and defined as an instance of (`rdf:type` or `a`) `nif:Word`. Within a sentence, every word is connected to its successor by `nif:nextWord`. Every newline-separated table is identified by a sentence URI and defined as `a nif:Sentence`. Every sentence is connected to its successor by `nif:nextSentence`. The sentence is implicitly identified with the virtual root of the syntactic annotation, hence, for every word that does not take another word as its `conll:HEAD`, we set `conll:HEAD` to the sentence URI.

For **generating URIs**, we identify sentences by their respective position in the data, starting with 1. Within a sentence, a word (or a multi-word expression, if encoded as a separate line) is identified either by its position (starting with 1) or by the value of its `ID` column, if provided. Following conventions of TIGER [3, 15] the local name of the URI is formed by "`s`"+$SENTID$+"_"+$WORDID$, e.g., `s1_1` for the first word in the first sentence. Following conventions adopted

[5] For the sake of processability, we use only a minimal fragment of NIF. We do neither adopt its full semantic model nor its URI formation constraints; yet, it is possible to transform CoNLL-RDF to NIF using SPARQL update and to provide NIF-compliant URIs if information about the original spacing (which is not preserved in CoNLL) is provided externally.

for CoNLL dependency annotations, the word `ID` 0 is reserved for the (virtual root of the) sentence, identified here with the sentence itself, the corresponding local name of the first sentence is thus `s1_0`. The local name is concatenated with a user-provided base URI that identifies the corpus or document that the current CoNLL file represents.

For successfully working with CoNLL-RDF, it is not necessary that corpus **URIs resolve**; still, it can be beneficial to facilitate sustainability and reusability: Resolvable URIs allow references to CoNLL-RDF data provided at a particular location in the web of data, and content negotiation offers the possibility to distribute both CoNLL and CoNLL-RDF data over the same URL.

2.2 Parsing and manipulating CoNLL-RDF

For the en-bloc conversion of CoNLL data we provide the JAVA class `CoNLL2RDF`. To facilitate processing data streams, `CoNLLStreamExtractor` reads CoNLL from `stdin` and applies `CoNLL2RDF` sentence by sentence to return semantically valid (albeit not necessarily canonically formatted) CoNLL-RDF.

When parsing CoNLL, every sentence is transformed into a `nif:Sentence`, every word as a `nif:Word`, and their order preserved by `nif:nextWord` resp. `nif:nextSentence`. As described above, every column is mapped to a datatype property in the `conll:` namespace and with a user-provided label as local name.

In addition to this mere conversion functionality, `CoNLLStreamExtractor` supports data manipulation by means of SPARQL update statements. It takes as additional arguments a list of files containing SPARQL update statements, marked by the flag `-u`. Every individual file represents a module, which can be stacked to form a processing pipeline. Successively, these SPARQL updates are applied to the individual sentences, thereby rewriting the RDF graph. We also support iterated applications, marked by an integer that can follow the respective SPARQL file in curly brackets.

Finally, a SPARQL select query can be applied to aggregate information from sentences and to produce a TSV table as output that may then be used for further evaluation, classifier training or database population. Without a select query, `CoNLLStreamExtractor` produces CoNLL-RDF as output.

2.3 CoNLL-RDF syntax and its canonical format

CoNLL-RDF is an extensible RDF vocabulary that can be serialized in any RDF format. Yet, to facilitate processability in NLP applications at a level comparable to the original CoNLL TSV format, it is accompanied with specifications for its canonical format: CoNLL-RDF syntax is a highly constrained subset of Turtle. However, these constraints represent layout conventions that the CoNLL-RDF formatter can enforce upon *any RDF data* from the `conll` namespace.

A document in **canonically formatted CoNLL-RDF** begins with a header that enlists all properties from the `conll` namespace used in the file. This header is a #-introduced comment, containing space-separated local names of properties meant to facilitate CoNLL-RDF parsing. It is not normative, but informational.

The header is followed by an empty line, then a block of `PREFIX` declarations that define the namespace mappings. The following typical prefixes will be assumed predefined for all examples in the text:

```
PREFIX nif: <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#>
PREFIX conll: <http://ufal.mff.cuni.cz/conll2009-st/task-description.html#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <BASE_URI>                # URI of the data source, provided by user
```

The prefixes are followed by the actual content of the corpus. Sentences within the corpus and words within a sentence are ordered sequentially. Sentences are separated by empty lines. The first line in a sentence refers to the sentence URI and defines it as a `nif:Sentence`. For all except the first sentence of a corpus, the first line is preceded by a `nif:nextSentence` statement marking its position.

The second line holds the first content word, defines it as a `nif:Word`, followed by its `conll:WORD`, then other annotations in desired order, finally concluded with a `nif:nextWord` statement pointing to the next word in the sentence (if available). The relation between words and sentences is established via `conll:HEAD`. CoNLL-RDF uses the Turtle separator ";" to enumerate the annotations assigned to a particular `nif:Word`, and "," to enumerate multiple values for the same annotation. CoNLL-RDF follows CoNLL in that all triples referring to one `nif:Word` are written in one line, concluded with ".". An example of CoNLL-RDF is given in Fig. 1. For the sake of brevity and readability, one token is rearragend while the others are cut off in their canonical in-line format.

The resulting format adopts many basic CoNLL conventions (comments starting with #, sentences separated by empty lines, one token per line, strictly ordered fields) and others modified (annotations separated by ";" and identified by property names rather than position). Most importantly, canonically formatted CoNLL-RDF can be as easily processed with low-level string manipulations as the original CoNLL format (see Sec. 3.5). Yet, it is still possible to process CoNLL-RDF in any other RDF format or tool without losing semantic information. The canonical format can be easily restored using the provided API.

2.4 Serializing CoNLL-RDF

`CoNLLRDFFormatter` reads CoNLL-RDF data in any common RDF serialization from standard input and converts it to one of several advanced output options:

`-rdf` canonically formatted CoNLL-RDF output (= default)
`-debug` augment `-rdf` output with color-highlighting (on Un*x shells)
`-conll FIELD1[.. FIELDn]` generates CoNLL TSV output according to the columns (fields) in their given order.
`-sparqltsv` provided a custom SPARQL select query, this also generates TSV output.
`-grammar` human-readable linearization with special formatting for dependency syntax (`conll:HEAD`+`conll:EDGE`), see Fig. 2
`-semantics` visualization of object properties and labels of `conll:WORD`s, with `conll:` properties removed; useful for visualizing extracted knowledge graphs

```
# sent_id dev-s4/de
:s3_0 nif:nextSentence :s4_0 .
:s4_0 a nif:Sentence .
:s4_1 a nif:Word; conll:WORD "Nach"; conll:ID "1"; conll:LEMMA "nach"; conll:UPOS "ADI
:s4_2 a nif:Word; conll:WORD "einem"; conll:ID "2";
    conll:LEMMA "ein"; conll:UPOS "DET"; conll:POS "ART";
    conll:FEAT "Case=Dat|Definite=Ind|Gender=Masc,Neut|Number=Sing|PronType=Art";
    conll:HEAD :s4_4; conll:EDGE "det"; nif:nextWord :s4_3 .
:s4_3 a nif:Word; conll:WORD "viertel"; conll:ID "3"; conll:LEMMA "Viertel"; conll:UP(
:s4_4 a nif:Word; conll:WORD "Jahr"; conll:ID "4"; conll:LEMMA "Jahr"; conll:UPOS "NOU
:s4_5 a nif:Word; conll:WORD "hielt"; conll:ID "5"; conll:LEMMA "halten"; conll:UPOS '
:s4_6 a nif:Word; conll:WORD "ich"; conll:ID "6"; conll:LEMMA "ich"; conll:UPOS "PRON'
:s4_7 a nif:Word; conll:WORD "ein"; conll:ID "7"; conll:LEMMA "ein"; conll:UPOS "DET";
:s4_8 a nif:Word; conll:WORD "duftendes"; conll:ID "8"; conll:LEMMA "duftend"; conll:U
:s4_9 a nif:Word; conll:WORD "Wunder"; conll:ID "9"; conll:LEMMA "Wunder"; conll:UPOS
:s4_10 a nif:Word; conll:WORD "in"; conll:ID "10"; conll:LEMMA "in"; conll:UPOS "ADP";
:s4_11 a nif:Word; conll:WORD "den"; conll:ID "11"; conll:LEMMA "der"; conll:UPOS "DET
:s4_12 a nif:Word; conll:WORD "Händen"; conll:ID "12"; conll:LEMMA "Hand"; conll:UPOS
:s4_13 a nif:Word; conll:WORD "."; conll:ID "13"; conll:LEMMA "."; conll:UPOS "PUNCT";
```

Fig. 1. CoNLL-RDF generated from the German UD corpus

```
s4_1  . . . / case-- Nach       ID 1 LEMMA nach POS APPR UPOS ADP
s4_2  . . . / det--- einem      FEAT Case=Dat|Definite=Ind|Gender=Masc,Neut|Number=Sing|PronType=Art ID 2 LEMMA ein POS ART UPOS DET
s4_3  . . . / nummod viertel    FEAT NumType=Card ID 3 LEMMA Viertel POS ADJA UPOS NUM
s4_4  . . / nmod---- Jahr       FEAT Case=Dat|Gender=Masc,Neut|Number=Sing ID 4 LEMMA Jahr POS NN UPOS NOUN
s4_5  . \ root       hielt      FEAT Number=Sing|Person=1|VerbForm=Fin ID 5 LEMMA halten POS VVFIN UPOS VERB
s4_6  . . \ nsubj--- ich        FEAT Case=Nom|Number=Sing|Person=1|PronType=Prs ID 6 LEMMA ich POS PPER UPOS PRON
s4_7  . . . / det--- ein        FEAT Case=Acc|Definite=Ind|Gender=Masc,Neut|Number=Sing|PronType=Art ID 7 LEMMA ein POS ART UPOS DET
s4_8  . . . / amod-- duftendes  FEAT Case=Acc|Degree=Pos|Gender=Masc,Neut|Number=Sing ID 8 LEMMA duftend POS ADJA UPOS ADJ
s4_9  . . \ dobj---- Wunder     FEAT Case=Acc|Gender=Masc,Neut|Number=Sing ID 9 LEMMA Wunder POS NN UPOS NOUN
s4_10 . . . / case-- in         ID 10 LEMMA in POS APPR UPOS ADP
s4_11 . . . / det--- den        FEAT Case=Dat|Definite=Def|Number=Plur|PronType=Art ID 11 LEMMA der POS ART UPOS DET
s4_12 . . \ nmod---- Händen     FEAT Case=Dat|Number=Plur ID 12 LEMMA Hand POS NN UPOS NOUN
s4_13 . . \ punct--- .          ID 13 LEMMA . POS $. UPOS PUNCT
```

Fig. 2. Human-readable dependency visualization (`CoNLLRDFFormatter -grammar`)

3 Use Cases

A key feature of CoNLL-RDF is that comes with SPARQL 1.1 support, an easy, expressive and powerful mechanism for graph rewriting. Any piece of annotation can thus be transformed and enriched by pipelines of stacked, reusable SPARQL scripts. This modular approach offers a wide range of use cases. Here, we describe sample use cases of both core functionality and SPARQL-based enrichment operations. All examples are taken from the German UD corpus.[6] In future publications we plan to address SRL syntax with variable column width.

3.1 SPARQL update pipelines

Together with the core API for parsing, manipulating and formatting CoNLL, resp., CoNLL-RDF, we provide a library of SPARQL update scripts and a number of Un*x shell scripts that illustrate their usabilities for pipelines to manipulate, format and aggregate over CoNLL annotations for different NLP tasks.

As such, Fig. 3 shows a basic conversion pipeline for CoNLL-U files: (1) `CoNLLStreamExtractor` reads CoNLL-U input from standard in, with base URI and CoNLL column labels as arguments (line 1-2). (2) A number of SPARQL

[6] `https://github.com/UniversalDependencies/UD_German`, part of [16]

update operations (details below) are called after `-u` (line 3-5). (3) `CoNLLRDF-Formatter` (line 7) produces canonically formatted CoNLL-RDF output (cf. Fig. 1). Alternative output options can be specified as arguments of the shell script and will be passed on to `CoNLLRDFFormatter`.

```
1 java CoNLLStreamExtractor https://github.com/UniversalDependencies/UD_German# \
2                           ID WORD LEMMA UPOS POS FEAT HEAD EDGE DEPS MISC \
3                           -u sparql/link-UPOS.sparql \
4                              sparql/link-stanford-EDGE.sparql \
5                              sparql/gloss-de-to-en-DBnary.sparql \
6     | \
7 java CoNLLRDFFormatter $*
```

Fig. 3. Example pipeline script

3.2 Enriching a corpus with dictionary glosses

Line 5 in Fig. 3 calls a SPARQL update script that illustrates enrichment of a corpus with information drawn at runtime from a remote dictionary in the LLOD cloud. One of the most important advantages of RDF and the semantic web standards is the possibility to easily integrate and aggregate such external information. DBnary [19] is a dictionary database covering 16 languages, applied here to heuristically generate English glosses from German lemmata:

```
PREFIX lemon: <http://lemon-model.net/lemon#>
PREFIX dbnary_deu: <http://kaiko.getalp.org/dbnary/deu/>
PREFIX dbnary: <http://kaiko.getalp.org/dbnary#>

INSERT { ?a conll:EN_GLOSS ?eng. }
WHERE { ?a conll:LEMMA ?lemma.
        SERVICE <http://kaiko.getalp.org/sparql> {
          SELECT ?lemma ?eng
          WHERE {
            GRAPH <http://kaiko.getalp.org/dbnary/eng> {
                ?tr dbnary:targetLanguage <http://lexvo.org/id/iso639-3/deu>.
                ?tr dbnary:writtenForm ?deTr.       FILTER(str(?deTr)=?lemma)
                ?tr dbnary:isTranslationOf/lemon:canonicalForm/lemon:writtenRep ?enTr.
                BIND(str(?enTr) AS ?eng)
            }
          } LIMIT 1
    }}
```

The nested select query uses DBnary's SPARQL service to retrieve one English translation for each `conll:LEMMA`. Albeit the SPARQL updates appear relatively complex, they are easy to adapt. For example, they are portable to other resources as long as the same representation formalism is used. DBnary uses the lemon/Monnet[11] vocabulary and the query refers to the German `lexvo:deu` glosses in the English DBnary, but with minimal modifications on `SERVICE` and `GRAPH` (to point to the locations of other dictionaries), this script can be applied to *any* dictionary provided in this format via any SPARQL endpoint. An update to dictionaries using the more recent lemon/OntoLex model [9] requires similarly minimal adjustments on the vocabulary used.

3.3 Annotation interpretation

Similarly to dictionaries, other LLOD resources can be integrated with corpus data and thus open new areas of application. In the context of pattern-based extraction, for example, one may want to develop patterns that are applicable to different part-of-speech (POS) tagsets. One strategy to achieve this is to translate POS tags into conceptual descriptions grounded in an external ontology. A SPARQL update script that performs this operation for the German data under consideration is called in Fig. 3, line 3 (and a similar one for dependency labels in line 4):

```
CREATE SILENT GRAPH <http://purl.org/olia/ud-pos-all.owl>;

# load OLiA Annotation Model for universal dependency POS tags
LOAD <http://purl.org/olia/ud-pos-all.owl> INTO GRAPH <http://purl.org/olia/ud-pos-all.owl>;

# load OLiA Linking Model for universal dependency POS tags
LOAD <http://purl.org/olia/ud-pos-all-link.rdf>
                                          INTO GRAPH <http://purl.org/olia/ud-pos-all.owl>;

PREFIX oliasys: <http://purl.org/olia/system.owl#>
INSERT { ?a a ?oliaConcept. }
WHERE {  ?a conll:UPOS ?pos.
         GRAPH <http://purl.org/olia/ud-pos-all.owl> {
           ?x oliasys:hasTag ?pos.
           ?x a ?concept.

           # complex property path to retrieve (possible) superclasses for ?x
           ?concept (rdfs:subClassOf|owl:equivalentClass|^owl:equivalentClass|
                     ((owl:intersectionOf|owl:unionOf)/rdf:rest*/rdf:first))*
                                                                  ?oliaConcept.

           # limit results to concepts from http://purl.org/olia/olia.owl
           FILTER(strstarts(str(?oliaConcept), "http://purl.org/olia/olia.owl"))
            }};

DROP GRAPH <http://purl.org/olia/ud-pos-all.owl>
```

Similar to the dictionary lookup using `SERVICE`, the keyword `LOAD` allows to retrieve a terminological resource,[7] and then to query it locally: These scripts match `conll:UPOS` values against tags as defined in the corresponding OLiA [5] annotation model, and then retrieve the OLiA Reference Model concepts with a complex property path that follows the `a/rdfs:subClassOf*` subsumption hierarchy (extended here to support `owl:equivalentClass` and set operations). After parsing the format-specific string patterns in the `FEAT` column this approach can also be used for linking to OLiA features. Fig. 4 shows two sample tokens with DBnary glosses and OLiA parts-of-speech generated by the pipeline. For better readability, the output has been slightly highlighted and rearranged.

Such an annotation interoperability pipeline is relevant beyond NLP: The *Lin|gu|is|tik* portal recently extended the thesaurus-based indexing of linguistic literature to annotations of LLOD-native annotated corpora and other language resources [6]. CoNLL-RDF now allows extending the indexing to CoNLL corpora provided by the Universal Dependency community.

[7] This example is informative, only. A scalable implementation requires external resources to be loaded in advance rather than for every sentence individually.

```
:s4_1 a nif:Word, olia:Adposition;  conll:UPOS "ADP";
            conll:EN_GLOSS "like";  conll:LEMMA "nach";
            conll:WORD "Nach";      nif:nextWord :s4_2 .
:s4_2 a nif:Word, olia:Determiner;  conll:WORD "einem";
            conll:EN_GLOSS "an";    conll:UPOS "DET";
            conll:LEMMA "ein";      nif:nextWord :s4_3 .
```

Fig. 4. Sample output with OLiA POS-concepts and glosses from DBnary

3.4 Generating TSV output and back-transformation

With the flag `-conll`, followed by a list of column names (which are resolved to properties in the `conll:` namespaces), `CoNLLRDFFormatter` generates CoNLL output. This approach is particularly convenient to perform elementary operations such as dropping or switching columns in CoNLL annotations. Fig. 5 shows the results of back-transformation. To generate problem-specific output, SPARQL update scripts may be applied to wrap this information in the corresponding `conll:` properties before exporting them via `-conll`.

```
# sent_id dev-s4/de
# ID    WORD    LEMMA   UPOS    POS FEAT    HEAD    EDGE    DEPS    MISC
1   Nach    nach    ADP APPR    _   4   case    _   _
2   einem   ein DET ART Case=Dat|Definite=Ind|Gender=Masc,Neut|Number=Sing|PronType=Art 4   det _   _
3   viertel Viertel NUM ADJA    NumType=Card    4   nummod  _   _
4   Jahr    Jahr    NOUN    NN  Case=Dat|Gender=Masc,Neut|Number=Sing   5   nmod    _   _
5   hielt   halten  VERB    VVFIN   Number=Sing|Person=1|VerbForm=Fin   0   root    _   _
6   ich ich PRON    PPER    Case=Nom|Number=Sing|Person=1|PronType=Prs  5   nsubj   _   _
7   ein ein DET ART Case=Acc|Definite=Ind|Gender=Masc,Neut|Number=Sing|PronType=Art 9   det _   _
8   duftendes   duftend ADJ ADJA    Case=Acc|Degree=Pos|Gender=Masc,Neut|Number=Sing    9   amod    _   _
9   Wunder  Wunder  NOUN    NN  Case=Acc|Gender=Masc,Neut|Number=Sing   5   dobj    _   _
10  in  in  ADP APPR    _   12  case    _   _
11  den der DET ART Case=Dat|Definite=Def|Number=Plur|PronType=Art  12  det _   _
12  Händen  Hand    NOUN    NN  Case=Dat|Number=Plur    5   nmod    _   _
13  .   .   PUNCT   $.  _   5   punct   _   _
```

Fig. 5. German UD corpus back-transformed from CoNLL-RDF

3.5 Preserving low-level processability

One factor of the popularity of CoNLL formats has been that they can be very easily processed using regular expressions and built-in data structures. In fact, any stream of CoNLL data (without comments) can be transferred into a three-dimensional, ragged array by coupling three regular expressions to split sentences (`\n\n+`), words (`\n`) and columns (`\t`). This style of parsing requires less than 10 lines of code in any modern programming language and provides a data structure which can be effectively navigated. With *canonically formatted* CoNLL-RDF, this advantage persists: sentence and word splitting remain identical; the different fields, however, should be parsed into a hash map (hashtable) rather than a plain array, using the following regular expression:

```
^([^\s]+)\s(.*\s)?([^\s]+)\s+([^;]+)(\s*;.*)?.$
    $1       $2      $3          $4     $5
 id (URI)          key/att     value
```

Handling annotations in a hashtable has the advantage of human-readable keys, and it also eliminates the threat of varying array dimensionality. Beyond this, however, CoNLL-RDF users can now benefit from other RDF technology, e.g., the availability of higher-level APIs, off-the-shelf data base solutions, and a standardized query language. In this way, canonically formatted CoNLL-RDF as produced by the `CoNLLRDFFormatter` provides a middle ground by supporting both well-known low-level functionalities akin to that of CoNLL and higher-level technologies that come with using RDF standards.

4 Discussion and related research

The primary goal of CoNLL-RDF is to provide NLP-friendly and human readable text representations of CoNLL data that can be directly processed/combined with off-the-shelf Linked Data technology. CoNLL-RDF provides an RDF view on CoNLL corpora, yet its canonical serialization maintains the low-level processability of the original CoNLL. In this section we provide an overview over advantages and limitations in comparison to other RDF representations and processing options.

4.1 CoNLL/TSV corpora in the Semantic Web

CoNLL-RDF is built on a minimal fragment of NIF to provide base datatypes, `nif:Word`, `nif:Sentence` and `nif:nextWord`, resp. `nif:nextSentence`. However, as mentioned above, we do not follow NIF URI formation principles nor its substring relations (e.g., `nif:referenceContext`, `nif:sentence` or `nif:word`). We deliberately omitted both characteristics (the actual core of NIF) because CoNLL words are not necessarily strings *in the original text data*, but may also represent empty elements; moreover, exact character positions are not consistently represented in CoNLL. Such URIs, however, can be generated by SPARQL update scripts if they de-tokenize, remove empty elements and recover the character offsets in the generated data. As evident from its name, NIF is, however, not focusing on the representation of corpora, but rather for use in NLP pipelines, its coverage of linguistic annotations is thus biased towards certain, frequent but less complex types of annotation. NIF is thus particularly well-suited for corpora with "simple" word-level, phrase-level and sentence-level annotations. It lacks support for representing annotation layers[8], non-branching syntactic nodes, empty elements (traces), labeled edges, conflicting tokenizations, semantic annotation, etc. It is well-suited for word-level NLP and entity linking.

The *Open Annotation (OA)* model [17] and the related *Web Annotation (WA)* specifications [18] were originally developed for expressing metadata about web content, but have occasionally been applied to represent Biomedical IE annotations in text. OA relies on reification and uses a naming scheme which is

[8] CoNLL-RDF provides a clear representation of annotation layers: CoNLL has been described as a 'hybrid standoff format' [13] in the sense that every column represents a self-contained annotation layer that refers to a common segmentation (tokens).

intransparent from an NLP perspective. Given the verbose model and the (from a linguistic point of view) counterintuitive terminology (`hasTarget`, `hasBody`), it is doubtful whether it will be accepted widely in the NLP and language resource community. However, like NIF, also OA properties can be generated from CoNLL-RDF using SPARQL.

POWLA[9] is another vocabulary for modeling linguistically annotated corpora. Whereas NIF and OA have been grown in a bottom-up fashion and extended/adapted as needed, POWLA has been designed in a top-down perspective as a full-blown reconstruction of the ISO TC 37/SC4 LAF/GrAF model in RDF/OWL, specifically designed to facilitate corpus querying. Unlike NIF, POWLA can express arbitrarily complex annotations. Unlike OA, it is terminologically transparent. Similar to OA, it suffers from a certain degree of verbosity in comparison to NIF and has thus not widely applied yet.

All these ontological models have their own advantages and limitations but share one common feature: a complex interconnected structure and relatively rich semantics. In order to convert a generic CoNLL file to fit into one of these vocabularies considerable restructuring is required that must be specific for every individual CoNLL dialect. In comparison to these, CoNLL-RDF provides a low-level view on the data, highly generic and with shallow semantics isomorphic to that of the underlying CoNLL data, that may serve as an intermediate step towards generating NIF, OA or POWLA data, but that is already sufficient (and more comprehensible) for researchers accommodated with classical CoNLL.

4.2 CoNLL-RDF and LOD wrapping frameworks

Several provider-side frameworks for the transformation and publication of tabular data in the Semantic Web exist. Among these, *RDB to RDF (R2RML)* [10] and *RDB Direct Mapping (RDB-DM)* [1] are of particular importance. They do build, however, on relational databases rather than TSV data, and thus impose an additional level of complexity when it comes to the mere transformation and manipulation of tabular data.

Conceptually closer to CoNLL-RDF is *CSV2RDF* [20]. Partially grounded in RDB-DM, CSV2RDF formalizes the direct rendering of tabular data to RDF in a similar way as implemented here. A crucial difference, however, is that CSV2RDF regards tables as unstructured objects, meaning that it neither supports distinguishing individual sentences nor preserves the order of words. Another difference is that CoNLL data does not necessarily come as a fixed-width TSV table, but that SRL or discourse annotations lead to ragged arrays not supported by CSV2RDF.

4.3 CoNLL APIs

Another strand of related research is concerned with manipulating and visualizing CoNLL data. Unlike most of these tools (which are countless, but barely

[9] `http://purl.org/powla` and 'Interoperability of Corpora and Annotations' in [8]

documented in scientific literature), CoNLL-RDF is highly generic, it is applicable to *all* TSV formats ever used for a CoNLL Shared Task, as well as several other, related specifications, e.g., SketchEngine,[10] Corpus Work Bench[11] and TreeTagger.[12] It focuses on processing one CoNLL file or stream at a time (no merging).

In comparison to popular APIs such as NLTK,[13] CoNLL-RDF is unique in that it introduces a clear and transparent distinction between minimal core functionality (parsing, manipulation and formatting, implemented in JAVA) and advanced manipulations (e.g., interpreting annotations, implemented in SPARQL). The API already allows column dropping or reordering; more advanced manipulations, however, are *not* part of the core infrastructure but subject to an extensible and reusable repository of SPARQL update scripts and select queries and thus portable across platforms, and highly reusable. In particular, SPARQL update operations can be stacked into modular pipelines – whose modules, however, can be re-used to apply to other data sources. Thereby, it achieves a high degree of genericity that preserves its capabilities for low-level processing (as known from many APIs) but comes with native support by the rich infrastructure developed and maintained by the Semantic Web community.

5 Summary and outlook

We introduce CoNLL-RDF, comprising of the following components:

- an *extensible, shallow RDF vocabulary* grounded on existing LLOD specifications (NIF),
- its *canonical format* that facilitates low-level processability in a similar fashion as known from CoNLL,
- a *core infrastructure* for parsing, manipulating and formatting CoNLL-RDF,
- a library of *SPARQL update scripts* to manipulate CoNLL-RDF, and
- *sample pipelines* illustrating their application to various use cases.

These are available under the Apache license 2.0 on our website.[14] In addition to this, we provide an LLOD-edition of the Universal Dependency corpora [16, v.1.4, 47 languages, 12 mio tokens] using CoNLL-RDF.

For the moment, we anticipate four primary uses of CoNLL-RDF:

- off-the-shelf corpus infrastructure providing database, query language, APIs, web services, etc.,

[10] `https://www.sketchengine.co.uk/`
[11] `http://cwb.sourceforge.net/`
[12] `http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/`
[13] NLTK provides numerous corpus readers specialized for different CoNLL variants, cf. `http://www.nltk.org/howto/corpus.html`.
[14] `http://acoli.informatik.uni-frankfurt.de/resources.html`

– a new format to publish CoNLL-style corpora, still parseable with 10 lines of Python, but compliant with W3C standards and thus massively improved technical support,
– advanced manipulations of CoNLL data using the SPARQL modules
– facilitated aggregation across distributed and local resources using resolvable URIs, federated search (`SERVICE`), and standardized access (`LOAD`)

Important for the practical application of CoNLL-RDF is to improve usability for researchers not too familiar with Semantic Web technologies. While developing SPARQL update scripts will always require special expertise, their mere use, rearrangement and adaptation for NLP problems does not.

Even though it is very easy to employ existing SPARQL modules already with the command-line based approach taken here, identifying existing SPARQL modules is not a trivial task: With a growing number of SPARQL update scripts and endless possibilities to combine them, dependencies must be taken into account when developing stacked SPARQL pipelines. Fortunately, it is easy to automatically fetch their dependencies among data sources and SPARQL queries (by tracing URIs which are sought, deleted and/or inserted by a given SPARQL module), a convenient and user-oriented visualization of this information is, however, yet to be developed and a goal of active research.

Acknowledgments

The research of Christian Chiarcos was supported by the BMBF-funded Research Group 'Linked Open Dictionaries (LiODi)' (2015-2020). Christian Fäth conducted his research in the context of DFG-funded projects 'Virtuelle Fachbibliothek' (2015-2016) and 'Fachinformationsdienst Linguistik' (2017-2019).

References

1. Arenas, M., Bertails, A., Prud'hommeaux, E., Sequeda, J.: A Direct Mapping of Relational Data to RDF (2012), `https://www.w3.org/TR/rdb-direct-mapping`, W3C Recommendation 27 September 2012
2. Beckett, D., Berners-Lee, T., Prud'hommeaux, E., Carothers, G.: RDF 1.1 Turtle (2014), `https://www.w3.org/TR/turtle/`, W3C Recommendation 25 February 2014
3. Brants, S., Hansen, S.: Developments in the TIGER Annotation Scheme and their Realization in the Corpus. In: Zampolli, A., et al. (eds.) Proceedings of the Third International Conference on Language Resources and Evaluation (LREC 2002). Las Palmas, Canary Islands, Spain (2002)
4. Buil Aranda, C., Corby, O., Das, S., Feigenbaum, L., Gearon, P., Glimm, B., Harris, S., Hawke, S., Herman, I., Humfrey, N., Michaelis, N., Ogbuji, C., Perry, M., Passant, A., Polleres, A., Prud'hommeaux, E., Seaborne, A., Williams, G.: SPARQL 1.1 Overview (2013), `https://www.w3.org/TR/sparql11-overview`, W3C Recommendation 21 March 2013
5. Chiarcos, C., Sukhreva, M.: OLiA - Ontologies of Linguistic Annotation. Semantic Web Journal 518, 379–386 (2015)

6. Chiarcos, C., Fäth, C., Renner-Westermann, H., Abromeit, F., Dimitrova, V.: Lin|gu|is|tik: Building the Linguist's Pathway to Bibliographies, Libraries, Language Resources and Linked Open Data. In: Calzolari, N., et al. (eds.) Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016). Portorož, Slovenia (2016)
7. Chiarcos, C., McCrae, J., Cimiano, P., Fellbaum, C.: Towards open data for linguistics: Linguistic Linked Data. In: Oltramari, A., Vossen, P., Qin, L., Hovy, E. (eds.) New Trends of Research in Ontologies and Lexical Resources, pp. 7–25. Springer, Berlin, Heidelberg, Germany (2013)
8. Chiarcos, C., Nordhoff, S., Hellmann, S. (eds.): Linked Data in Linguistics. Springer, Berlin, Heidelberg, Germany (2012)
9. Cimiano, P., McCrae, J., Buitelaar, P.: Lexicon Model for Ontologies (2016), https://www.w3.org/2016/05/ontolex/, W3C Community Report, 10 May 2016
10. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language (2012), https://www.w3.org/TR/r2rml, W3C Recommendation 27 September 2012
11. Declerck, T., Buitelaar, P., Wunner, T., McCrae, J., Montiel-Ponsoda, E., de Cea, A.: lemon: An Ontology-Lexicon model for the Multilingual Semantic Web. In: W3C Workshop: The Multilingual Web - Where Are We? Madrid, Spain (Oct 2010)
12. Hellmann, S., Lehmann, J., Auer, S., Brümmer, M.: Integrating NLP using Linked Data. In: Welty, C., et al. (eds.) Proc. 12th International Semantic Web Conference (ISWC 2013). Sydney, Australia (2013), also see http://persistence.uni-leipzig.org/nlp2rdf/
13. Ide, N., Chiarcos, C., Stede, M., Cassidy, S.: Designing annotation schemes: From model to representation. In: Ide, N., Pustejovsky, J. (eds.) Handbook of Linguistic Annotation. Text, Speech, and Language Technology, Springer, Dordrecht, Netherlands (in press)
14. Johnson, M.: Computational Linguistics. Where do we go from here? Invited talk at the 50th Annual Meeting of the Association of Computational Linguistics (ACL-IJCNLP 2012), Jeju, Korea. http://web.science.mq.edu.au/~mjohnson/papers/Johnson12next50.pdf (accessed 2016/07/13) (2012)
15. Lezius, W., Biesinger, H., Gerstenberger, C.: TigerXML Quick Reference Guide (2002), https://nats-www.informatik.uni-hamburg.de/pub/CDG/TigerCorpus/tigerxml_manual.ps.gz, IMS, University of Stuttgart
16. Nivre, J., Agić, Ž., Ahrenberg, L., et. al.: Universal dependencies 1.4 (2016), http://hdl.handle.net/11234/1-1827, LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University
17. Sanderson, R., Ciccarese, P., Van de Sompel, H.: Open Annotation Data Model (2013), http://www.openannotation.org/spec/core, W3C Community Draft, 08 February 2013
18. Sanderson, R., Ciccarese, P., Young, B.: Web Annotation Data Model (2017), https://www.w3.org/TR/annotation-model, W3C Recommendation 23 February 2017
19. Sérasset, G.: DBnary: Wiktionary as a Lemon-Based Multilingual Lexical Resource in RDF. Semantic Web Journal 648 (2014), also see http://kaiko.getalp.org/about-dbnary/
20. Tandy, J., Herman, I., Kellogg, G.: Generating RDF from Tabular Data on the Web (2015), https://www.w3.org/TR/csv2rdf, W3C Recommendation 17 December 2015