Chapter 6
Modelling Linguistic Annotations

Abstract This chapter describes how linguistic annotations can be represented in RDF. Web Annotation and NIF provide the means to reference text segments on the web. Yet, representing linguistic annotations requires appropriate vocabularies. We discuss relevant vocabularies and illustrate how they can be applied to support annotation at different levels.

6.1 Introduction

In the previous chapter, we discussed mechanisms to address text segments using URIs. While mechanisms to address text segments provide the basis for representing annotations in RDF, an important ingredient is still missing: vocabularies that allow to describe the linguistic phenomena annotated. In this chapter, we describe OWL- and RDF-based vocabularies to represent annotations which provide the basis for semantic interoperability.

With the rising importance of linguistic annotations in linguistics and language technology, the band-width and amount of linguistic annotations is continuously increasing in complexity and heterogeneity, and this is directly reflected in the number and diversity of annotation formats [1].

Two relatively widely used corpus formalisms have been introduced in Sect. 5.1, already. They are two representative examples for the variability of representation formalisms available, but suffer either from restrictions in terms of expressivity (CoNLL-TSV, TEI/inline XML) or processability (TEI/standoff XML):

- CoNLL-TSV (Sect. 5.1.1) annotates one word per line; it can neither adequately represent segments smaller than words (as necessary for morphology), nor nested structures (as necessary for phrase structure grammar). Instead, morphological and phrase-structure grammars are reduced to opaque strings whose semantics are not defined by the format, but left to client software.
- Inline XML (e.g. in TEI, Sect. 5.1.2, Fig. 5.4) can represent annotations at arbitrary granularity, but is limited to tree structures.

P. Cimiano et al., *Linguistic Linked Data*,
https://doi.org/10.1007/978-3-030-30225-2_6

- Standoff XML (e.g. in TEI, Sect. 5.1.2, Fig. 5.5) can represent arbitrary graph structures (and thus any kind of linguistic annotation). This genericity, however, comes at the price of insufficient readability and limited technological support outside the language resource community.

As a result, interoperability between and across formats becomes a major concern, and has traditionally been addressed by graph-based data models and XML standoff formats. Standoff formats have a longer history in the language resource community, in particular for annotations with overlapping segments. While XML had been considered a solution for language resource interoperability during the late 1990s, such constellations required extensions of the original XML data model, i.e. the support for directed multi-graphs in annotation.[1] Standoff XML has been considered state of the art for multi-layer corpora during the early 2000s. As it forms the basis for the Linguistic Annotation Framework [3, LAF]—i.e. ISO 24612:2012 [4]—and its instantiations in the GrAF format [5], PAULA XML [6] and KAF [7], it is still considered technologically relevant [1], albeit it declined in popularity because of poor support by off-the-shelf XML technology [8] and the rise of JSON.

More recently, standoff XML in human language technology is thus being increasingly replaced by LLOD-compliant representations, often based on JSON-LD, e.g. Web Annotation (Sect. 5.2) or LIF (Sect. 11.4). Like earlier standoff XML formats, RDF implements labelled directed multi-graphs, but does benefit from a richer technological ecosystem and a broader developer community. Burchardt et al. [9] demonstrated that syntactic and semantic annotations can be integrated and jointly queried on grounds of an RDF formalization. Cassidy et al. [10] developed an early RDF serialization of LAF, and the standoff XML format NAF [11] has been explicitly designed with the goal to facilitate its transformation to RDF. Subsequently, various NLP infrastructures have adopted RDF-native exchange formats, including TELIX [12], NIF [13] and LIF [14].

In this chapter, we describe two approaches to represent linguistic annotations in RDF and as linked data: CoNLL-RDF and POWLA. Based on a fragment of NIF, CoNLL-RDF provides a semantically shallow and isomorphic reconstruction of TSV formats in RDF and thus represents a technological bridge between the most popular format family in NLP and LLOD technologies. POWLA is an OWL2/DL vocabulary that defines generic linguistic data structures, which can be used in combination with CoNLL-RDF, Web Annotation or NIF to formalize any kind of linguistic annotation.

[1] In the language resource community, it is generally assumed that labelled directed (multi-) graphs can represent *every* kind of linguistic annotation [2].

6.2 Transforming Legacy Annotation Formats into RDF

In NLP and DH, established annotation formats such as CoNLL are still much more prevalent than native RDF-based annotation formats. Even in the case that RDF backends are used for storing or transforming annotations, legacy formats are used as input/output to maintain compatibility with existing NLP pipelines. However, RDF excels at information integration, and where annotations from different sources are to be combined with each other, to be transformed or to be linked with other pieces of information, RDF-native formalisms enjoy increasing popularity (see Chap. 11).

Depending on whether developers see their priority in backward compatibility with legacy formats or in interoperability and re-usability, there are two approaches to transform legacy annotation formats into RDF:

- **Shallow transformation:** In this approach, a direct mapping of the original legacy data source to RDF is created. The mapping is shallow in that it provides a 1:1 mapping of the data elements of the original data source to RDF properties and classes, preserving by and large the original data model. There is thus no semantic integration or interoperability as the resulting vocabulary is proprietary for the source data model. Yet, this allows to transform, query and process annotations with RDF technology and to link them with LOD resources as required. The resulting RDF is specific to a given legacy data format and thus lacks interoperability to other RDF datasets.
- **Alignment with RDF/OWL native annotation vocabularies:** In applications where semantic integration and interoperability is important, an approach should be followed in which the source data model is mapped to an existing OWL vocabulary. This approach requires aligning/lifting the data format of the legacy data source to existing RDF/OWL vocabularies. The advantage of this approach is that by reuse of existing vocabularies in machine-readable formats, interoperability across datasets is ensured.

Here, we illustrate the shallow transformation approach using CoNLL as a popular legacy format for linguistic annotations.

6.2.1 CoNLL-RDF: Shallow Transformation of CoNLL into RDF

In NLP, the CoNLL formats (see Sect. 5.1.1) represent a large and diverse family of formats based on tab-separated values (TSV), and used for a wide range of linguistic phenomena. CoNLL-compatible formats have also been the basis for the development of corpus infrastructures, in particular, the Corpus WorkBench [15] and SketchEngine [16], widely used in corpus linguistics and lexicography, respectively. Individual CoNLL formats (‘dialects’) posit specific constraints on columns and their content.

```
#ID WORD      LEMMA    UPOS  POS FEATS         HEAD EDGE
1   James     James    PROPN NNP Number=Sing  2    name
2   Baker     Baker    PROPN NNP Number=Sing  3    nsubj
3   told      tell     VERB  VBD Mood=Ind|... 0    root
4   reporters reporter NOUN  NNS Number=Plur  3    obj
5   Friday    Friday   PROPN NNP Number=Sing  3    nmod:tmod
6   :         :        PUNCT :   _            3    punct
```

Fig. 6.1 CoNLL-U annotation for Fig. 5.1 (p. 62)

CoNLL-RDF [17] is a semantically shallow approach to render CoNLL data structures in a generic way in RDF. Building on a minimal core vocabulary drawn from NIF (Sect. 5.3), it provides a generic mechanism to create user-defined datatype properties in the `conll` namespace, special handling for `HEAD` and `ARG` columns to represent dependency syntax and semantic roles, and a basic infrastructure for parsing, transforming, visualizing and converting CoNLL-RDF. Converting CoNLL-RDF includes import from and export to CoNLL TSV, but also to human-readable and graphical representations, as well as to a canonical serialization in Turtle/RDF.

Figure 6.1 shows a CoNLL fragment in the **CoNLL-U** dialect, the CoNLL format used by the Universal Dependencies (UD) initiative [18][2]: `ID` is the number of the word in the sentence, `WORD` is the form of the word,[3] `LEMMA` its lemma, `UPOS` its UD part-of speech tag, `POS` its original part-of-speech tag, `FEATS` its morphosyntactic features, `HEAD` the ID of its parent word in dependency annotation (or 0 for the root), and `EDGE` the label of its dependency relation. The final columns `DEPS` and `MISC` are not used for this example.

To provide a generic conversion of CoNLL data, CoNLL-RDF expects column labels to be provided at conversion time. For each column, an RDF property is generated using the user-provided label as local name in the `conll` namespace. As these properties are provided by the user, they lack any alignment to existing RDF/OWL vocabularies. It is in this sense that CoNLL-RDF is shallow as properties are specific for a certain CoNLL format and lack interoperability with other vocabularies.

[2] http://universaldependencies.org/, last accessed 09-07-2019.

[3] In CoNLL-U terminology, the columns `WORD` and `EDGE` are termed `FORM` and `DEP`, respectively. While CoNLL-RDF does not require to use these labels, it provides specialized visualization for them in a human-readable export. We thus recommend following CoNLL-RDF terminology rather than CoNLL-U terminology. Likewise, the column `POS` should be `XPOS` in CoNLL-U, but we follow Fig. 5.1 in using the same column label for the same information.

The following URI schema is used[4]:

- During conversion time, the user provides a base URI, e.g. identifying the corpus or document that the current CoNLL file represents.
- The abstract sentence URI consists of the base URI, followed by `#s` and the number of the sentence in the corpus, starting with 1.
- The word URI consists of the abstract sentence URI, followed by `_` and the number of the word in the sentence, starting with 1. If an explicit `ID` column is provided, this value is used instead.
- The sentence URI consists of the abstract sentence URI, followed by `_0`. It is thus the 0th word of the sentence, i.e. its virtual root (following a practice used in CoNLL dependency parsing).

Basic data structures are `nif:Sentence` and `nif:Word`. Their respective sequential structure is made explicit by `nif:nextSentence` and `nif:nextWord`. CoNLL properties are generated from user-provided column labels for every non-empty cell:

- **Syntactic dependencies:** Column `HEAD` ⇒ object property `conll:HEAD` pointing to the URI of the parent word, resp. the sentence. If no `HEAD` column is provided, `conll:HEAD` points to the sentence.
- **Semantic roles:** In semantic role annotation, one column per frame in the sentence is created, so that the *n*th `SRL-ARGs` column contains the roles of the *n*th predicate in the sentence (i.e. annotated in the `PRED` column).[5] For a word annotated with the value, say `ARG0` in the third `SRL-ARGs` column, the predicate is the third word in the sentence that has an annotation for `PRED` ⇒ object property `conll:ARG0` pointing from predicate URI to argument URI.
- **Other:** Other column labels yield datatype properties containing an untyped literal. CoNLL-RDF does not require specific column labels, but it provides enhanced visualizations for the columns `WORD` and `EDGE`.

As a result, we obtain a shallow rendering of the original CoNLL data structure in RDF (Fig. 6.2), which can be effectively queried, manipulated and serialized back into CoNLL using off-the-shelf RDF technology. The `conll:` namespace used here is not backed by an ontology, but populated by properties as defined by the user (column labels) or in the data (values for *X*`-ARGs` columns). One of the

[4]CoNLL formats do not preserve the original whitespaces, we thus cannot produce valid NIF URIs for a CoNLL-annotated text. (CoNLL-U does provide mechanisms to express the *absence* of whitespaces after a token, but this does not preserve information about the type of whitespace and is specific to a single dialect.) Instead, CoNLL-RDF follows the naming conventions of [19, 20].

[5]More precisely, this is a pattern for every *X*`ARGs` column label for which the corresponding *X* column does exist. For CoNLL-RDF, `ARGs` column must not be followed by other annotations, and their values (not their labels) must by valid URI characters.

```
:s1_1 a nif:Word;
    conll:ID "1";
    conll:WORD "James";
    conll:LEMMA "James";
    conll:UPOS "PROPN";
    conll:POS "NNP";
    conll:FEATS "Number=Sing";
    conll:HEAD :s1_2;
    conll:EDGE "name";
    nif:nextWord :s1_2.
```

Fig. 6.2 Turtle fragment for the first row in Fig. 6.1

design goals of CoNLL-RDF has been seamless round-tripping from CoNLL-TSV to CoNLL-RDF and back to TSV. While TSV export from CoNLL-RDF can be easily accomplished with SPARQL SELECT, such queries only support a fixed number of columns. Export of semantic roles is somewhat more complex, as it involves nested `SELECT` for every word using `GROUP_CONCAT` with tabulator as separator.

Beyond format specifications and converters, the CoNLL-RDF library permits the iterative application of sequences of SPARQL updates (resp., files that contain these) to documents or data streams providing CoNLL(-RDF) data. Even though it lacks formal semantics by design, the CoNLL RDF model can thus also serve as a basis to transform CoNLL data into semantically richer formalisms such as the NIF ontology or POWLA (Sect. 6.3). The motivation of CoNLL-RDF as a representation formalism is to facilitate the transformation of annotation graphs *provided in a commonly used format* on the basis of available Semantic Web technologies for its querying, manipulation, storage, etc. in a backward-compatible fashion, such that the results can be serialized back into the original format.

6.2.2 *Querying and Manipulating CoNLL-RDF Annotations*

Figure 5.3 introduced semantic role annotations for the example given above, replicated in Fig. 6.3, with a CoNLL-RDF rendering in Fig. 6.4. NIF data structures identify words and their structural relations, and user-provided column labels (`WORD`, `POS`, `NER`, `SRL`) yield the corresponding properties in the `conll:` namespace. For the label `SRL-ARGs` we have one column only in the example, corresponding to semantic arguments of the first word with `SRL` annotations, i.e. `:s1_4`, and accordingly, the predicate `:s1_4` is annotated with properties generated from the annotations in the first `SRL-ARGs` column.

For named entity types, we see that the CoNLL-RDF preserves the original IOBES annotation, with `B-PERSON` marking the beginning of a person name, and `E-PERSON` marking the end of the span annotated with `PERSON`. With SPARQL

```
# WORD     POS NER      SRL        SRL-ARGs
James      NNP B-PERSON _          ARG0
Baker      NNP E-PERSON _          ARG0
told       VBD O        tell.v.01 rel
reporters  NNS O        _          ARG2
Friday     NNP S-DATE   _          ARGM-TMP
:          :   O        _          _
```

Fig. 6.3 POS, NER and PropBank (SRL) annotations (replicated from Fig. 5.3)

```
:s1_1 a nif:Word; conll:WORD "James";   conll:POS "NNP";
                  conll:NER "B-PERSON"; nif:nextWord :s1_2.
:s1_2 a nif:Word; conll:WORD "Baker";   conll:POS "NNP";
                  conll:NER "E-PERSON"; nif:nextWord :s1_3.
:s1_3 a nif:Word; conll:WORD "told";    conll:POS "VBD";
                  conll:SRL "tell.v01"; conll:rel :s1_3;
                  conll:ARG0 :s1_1, :s1_2; conll:ARG2 :s1_4;
                  conll:ARG-TMP :s1_5;  nif:nextWord :s1_4.
:s1_4 a nif:Word; conll:WORD "reporters"; conll:POS "NNS";
                  nif:nextWord :s1_5.
:s1_5 a nif:Word; conll:WORD "Friday";  conll:POS "NNP";
                  conll:NER "S-DATE";   nif:nextWord :s1_6.
:s1_6 a nif:Word; conll:WORD ":";       conll:POS ":".
```

Fig. 6.4 CoNLL-RDF/Turtle fragment for Fig. 6.3

Update, this can be effectively transformed into a more compact representation. In a first step, we eliminate the IOBES codes from `conll:NER`:

```
DELETE {
  ?w conll:NER ?iobes.
} INSERT {
  ?w conll:NER ?short.
} WHERE {
  ?w conll:NER ?iobes.
  BIND(replace('^[IOBES]-','') AS ?short)
};
```

In a second step, we use the dependency annotation from Fig. 6.2 to restrict the NER annotation to the syntactic head of the corresponding phrase:

```
DELETE {
  ?w conll:NER ?ner.
} WHERE {
  ?w conll:NER ?ner.
  ?w conll:HEAD/conll:NER ?ner.
};
```

This update eliminates all `NER` annotations if the syntactic head carries the same annotation. In the same way, semantic role annotations (exemplified for the example of `ARG0` here) can be reduced to the syntactic head:

```
DELETE {
  ?pred ?role ?arg.
```

```
} WHERE {
  ?pred conll:SRL [].
  ?pred ?role ?argHead, ?argDep.
  ?argDep conll:HEAD ?argHead.
  FILTER(?role in (conll:ARG0,conll:ARG1,conll:ARG2,...))
};
```

The result can be rendered in a somewhat simplified CoNLL TSV format using SPARQL SELECT (we leave `SRL-ARGs` as an exercise):

```
SELECT ?word ?pos ?ner ?srl
WHERE {
  ?w conll:WORD ?word; conll:POS ?pos.
  OPTIONAL { ?w conll:POS ?ner_raw. }
  OPTIONAL { ?w conll:SRL ?srl_raw. }
  BIND(IF(BOUND(?ner_raw),?ner_raw,'_') as ?ner)
  BIND(IF(BOUND(?srl_raw),?srl_raw,'_') as ?srl)
}
```

This query is trivial for properties where annotations are obligatory (`WORD`, `POS`). For properties where empty annotations are possible, CoNLL compliance requires to insert _. Here, _ is inserted for missing (non-bound) values of optional properties.

From this query, the CoNLL RDF libraries (or any other SPARQL engine) will produce the following table:

```
James      NNP _       _
Baker      NNP PERSON  _
told       VBD _       tell.v.01
reporters  NNS _       _
Friday     NNP DATE    _
:          :   _       _
```

The CoNLL RDF libraries guarantee proper sentence segmentation and word order, but this query can also be run using any general SPARQL engine. Reproducing the original word order requires slight modifications in this case[6]:

```
SELECT ?word ?pos ?ner ?srl
WHERE {
  ?w conll:WORD ?word; conll:POS ?pos.
  OPTIONAL { ?w conll:POS ?ner_raw. }
  OPTIONAL { ?w conll:SRL ?srl_raw. }
  BIND(IF(BOUND(?ner_raw),?ner_raw,'_') as ?ner)
  BIND(IF(BOUND(?srl_raw),?srl_raw,'_') as ?srl)
  { SELECT ?w (COUNT(DISTINCT ?tmp) AS ?id)
```

[6]Sentence-level segmentation is left as an exercise. This requires retrieving the sentence URI for every word (by means of `?w conll:HEAD+ ?s. ?s a nif:Sentence`) and `GROUP BY` (and `ORDER BY`) sentence URIs before ordering words. If a word has no `nif:nextWord` predecessor, set the value of the `WORD` column to `concat('\\n',?word)` rather than to `?word`.

```
    WHERE {
      ?tmp nif:nextWord* ?w
    } GROUP BY ?w
  }
} ORDER BY ?w
```

This query uses an embedded `SELECT` statement to count the number of preceding NIF words to create a numerical id (alternatively, the optional `conll:ID` property can be used) and performs an ordering of words on this basis. The original query can also be extended to include the dependency annotations used for disambiguating the `NER` annotations:

```
SELECT ?id ?word ?pos ?head ?dep ?ner ?srl
WHERE {
  ?w conll:WORD ?word; conll:POS ?pos.
  OPTIONAL { ?w conll:POS ?ner_raw. }
  BIND(IF(BOUND(?ner_raw),?ner_raw,'_') as ?ner)
  OPTIONAL { ?w conll:SRL ?srl_raw. }
  BIND(IF(BOUND(?srl_raw),?srl_raw,'_') as ?srl)
  BIND(replace(str(?w),'.*_','') AS ?id)
  ?w conll:HEAD ?h.
  BIND(replace(str(?h),'.*_','') AS ?head)
  ?w conll:EDGE ?dep.
};
```

For this query, we rely on the CoNLL-RDF URI scheme introduced above, i.e. the URI ends with the original (or implied) `ID` value for every word (and its head) is the last, separated with `_`. However, this information cannot be drawn from a URI directly but must rather be cast as a string (`str()`), before a replacement can be applied. These examples show how CoNLL-RDF and SPARQL can be effectively used to perform complex transformations of CoNLL data. While export to CoNLL TSV is somewhat more challenging (unless the CoNLL-RDF libraries are used), the transformation itself is simple and straight-forward. In addition, it is possible to consult external knowledge sources as part of the transformation. As an example, consider the LLOD edition of VerbNet [21] provided as part of LemonUby [22]. The semantic role annotations of PropBank are grounded in VerbNet. Having both corpus and dictionary available as RDF, resp. LLOD data, we can now produce an alignment of VerbNet classes and PropBank predicates.

6.3 Top-Down Modelling: Generic Data Structures

Common annotation formats have been designed to either fulfil requirements of a specific task (e.g. as training data for syntactic or semantic parsing), the phenomenon they address (e.g. annotation of words and or trees) or with the goal to adopt existing technical solutions for their processing. By rendering linguistic data structures in directed graphs, RDF provides a means to facilitate syntactic

or structural interoperability between such formats (resp., annotations expressed by them), but only in the sense that they can be stored and accessed uniformly. Neither the Web Annotation nor the NLP Interchange Format aim for semantic interoperability of annotations. While NIF provides a vocabulary for annotations, it is specific to the types of annotation currently supported, including parts of speech, lemmatization, phrase structure syntax, entity linking and sentiment analysis. For representing linguistic annotations in a way that semantic interoperability is ensured, the above-mentioned models need to be complemented with an explicit vocabulary that allows to describe annotations at a content level. Here, we describe POWLA [23], a small but generic vocabulary developed to facilitate the exchange and querying of multi-layer corpora with arbitrary annotations. As it aims to support *any* kind of linguistic annotation, POWLA is the most general of the vocabularies suggested so far.

6.3.1 Linguistic Annotations in POWLA

POWLA[7] [24] is a small OWL2/DL-based vocabulary grounded in the Linguistic Annotation Framework [4, LAF] and thus capable of representing *any kind* of text-oriented annotation. This genericity sets it apart from earlier approaches on LOD-based corpus representations in that POWLA is not tied to specific types of annotations, e.g. constituent syntax and frame semantics [9], for syntax [25] or selected NLP tasks [26].

POWLA is an OWL2/DL serialization of PAULA [27, 28], an early implementation of LAF [29] and serialized in a standoff XML format comparable to GrAF [30]. PAULA and PAULA XML have been the basis for developing integrative NLP pipelines [31], a corpus information system for multi-layer corpora [32], and a generic converter suite for linguistic annotations [33]. In these applications, the robustness of the PAULA data model has been demonstrated as well as its capabilities to integrate annotations from different sources. POWLA has been applied to modelling multi-layer corpora, including case studies on the Manually Annotated Sub-Corpus of the American National Corpus [34, MASC], syntactic and coreference-annotated corpora [24] and for high-precision information extraction [35], annotation engineering [36] and syntactic parsing [37].

POWLA aims to formalize linguistic annotations by building on existing standards with respect to their anchoring in the original document. In PAULA XML, XLink/XPointer references served this purpose. POWLA is underspecified with respect to this, but both NIF URIs and Web Annotation selectors can be employed. POWLA thus serves to complement existing NIF, Web Annotation or application-specific RDF renderings of linguistic annotations with interoperable linguistic data structures.

[7] http://purl.org/powla/powla.owl.

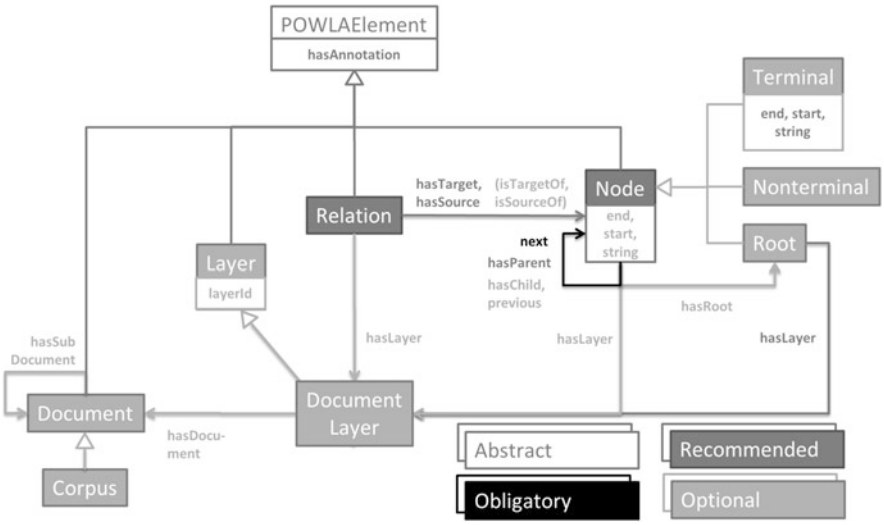

Fig. 6.5 POWLA data model, obligatory, abstract, recommended and optional properties shown in different shades of grey

The POWLA data model is illustrated in Fig. 6.5. It provides a scalable vocabulary in that it defines a minimum core of obligatory and recommended properties and concepts concerned with linguistic annotation, along with a number of optional concepts and properties particularly relevant for corpus organization, or querying.

6.3.1.1 POWLA Nodes

Units of annotation are formalized as `powla:Node`. By means of the property `powla:hasParent` (and its inverse `powla:hasChild`), the hierarchical composition of nodes can be expressed, e.g. for syntax trees or discourse structure. The property `powla:next` (and its inverse `powla:previous`) is used to express the sequential order of two adjacent nodes. It is recommended that `powla:next` is used to link nodes which have the same parent, as this facilitates navigation in tree structures. POWLA does not provide its own mechanism for document linking, but can be applied in combination with NIF or Web Annotation, e.g. a NIF URI (representing, e.g., a `nif:Word`) can be linked via `powla:hasParent` with a `powla:Node` that represents a syntactic phrase. The domain declaration of `powla:hasParent` RDFS entails that the NIF URI represents a `powla:Node`, and thus, `powla:next` transitions can (and should) be created to connect it with adjacent NIF words of the same phrase or, if necessary, with an empty element (represented by a `powla:Node` without an associated NIF URI) to be inserted between string-adjacent NIF words. It is recommended to use `powla:next` transitions for `powla:Nodes` with the same parent not only because this facilitates

navigation, but also because it allows to define an order of tree elements that is independent from the order of strings. This may be necessary, e.g., to represent a form of 'deep' or 'canonical' syntax that deviates from the actual strings, e.g. in spoken discourse, but also in complex structures that violate syntactic projectivity. One example here is preposition stranding as in (3):

(3) $[\textit{What}]_{\text{PP}_1}$ *are you talking* $[\textit{about}]_{\text{PP}_1}$ *?*

Here, *what* is placed in preverbal position ('WH-movement'), but it still forms a syntactic unit with the preposition that remained in its 'base position'. For such constellations, the Penn Treebank uses empty elements and a complex coindexing scheme, but as an alternative POWLA would also allow to specify a direct `powla:next` link between both parts of the PP—independently from the `nif:nextWord` transitions that express the *actual* sequence of words.[8]

POWLA defines three subclasses of nodes, albeit their use is optional, as they can be inferred `powla:hasParent` annotations:

- `powla:Terminal`: A `powla:Node` which is not the object of a `powla:hasParent` property. The notion of 'terminal node' has little practical relevance for modelling and querying corpora in RDF, but a single 'base segmentation', resp., the '(privileged) tokenization layer' have been fundamental concepts in XML standoff formats. We assume that the privileged tokenization layer is the sequence of minimal segments, thereby identifying terminals by the absence of `powla:hasParent` properties. As an alternative to the minimal segmentation principle, explicit `nif:Word` and `nif:nextWord` annotations can be used for querying token sequences, but this is beyond POWLA.
- `powla:Nonterminal`: Every `powla:Node` which is not a terminal.
- `powla:Root`: A root node is a `powla:Node` which is not a subject of a `powla:hasParent` property. For corpus querying and corpus organization, `powla:Root` is an important concept as it provides an 'entry point' into linguistic annotations. In particular, a number of properties that organize linguistic annotations into documents and corpora are based on the notion of root nodes as this limits the number of necessary links between data points and data sets.

[8] In fact, we may even express the 'canonical'—albeit in this case, somewhat unnatural—order directly on NIF words. In the absence of any hierarchical annotation, we can specify the following `powla:next` transitions (represented by →):

about → *What* → *are* → *you* → *talking* → *?*

While there are no restrictions regarding the ordering of blank nodes defined as `powla:Nodes`, the antisequential ordering *of externally defined URIs* must be handled with care in order to prevent cycles when combining annotations coming from different sources. It is thus recommended to follow the sequential order and to connect nodes with the same parent via `powla:next` if and only if no sibling does exist that is intermediate in terms of the original string order.

POWLA nodes carry the following object properties:

- `powla:next`: (obligatory) object property (inverse `powla:previous`, optional) used for expressing the sequential order of sibling nodes.
- `powla:hasParent`: (recommended) object property (inverse `powla:hasChild`, optional) used for representing hierarchical annotations.

POWLA defines several datatype properties for nodes, including:

- `powla:start`, `powla:end`: (optional, recommended for terminals) specify numerical indices, e.g. an offset (as in NIF) or another, structure-sensitive index [38]. Interpretation is implementation specific, but the end value needs to be greater than the start value.
- `powla:hasAnnotation`: abstract property applying to every POWLA element, should be instantiated with annotation-specific subproperties following the schema `has`*XY*.
- `powla:string`: (optional, recommended for terminals) carries the string value of the tokens to which this annotation unit (node) applies.

6.3.1.2 POWLA Relations

POWLA nodes represent units of annotation and their sequential and hierarchical structure. For representing annotated relations between nodes as labelled edges, POWLA uses a reified representation by means of `powla:Relation`. By means of `powla:hasSource` and `powla:hasTarget` (inverse of `powla:isSourceOf`, resp. `powla:isTargetOf`) POWLA relations are linked with the nodes between which the relation holds.

Note that POWLA relations do not represent whether they encode hierarchical or non-hierarchical relations. Hierarchical (dominance) relations are characterized by coverage inheritance, i.e. the target (parent) node covers the same stretch of terminals as the source (child) nodes. A prototypical example is phrase structure syntax. Non-hierarchical relations do not impose such constrains and hold between nodes independently from their hierarchical structure; prototypical examples are dependency syntax or coreference annotation. In POWLA, this difference is not marked at the relation type, but independently from the reified relation by an accompanying `powla:hasParent` (`powla:hasChild`) property.

In the same way as POWLA nodes, relations can carry `powla:hasAnnotation` subproperties.

6.3.1.3 OLiA Links

The Ontologies of Linguistic Annotation (OLiA, Sect. 8.4.2) provide OWL2/DL formalizations of annotation schemes for a plethora of linguistic annotations and languages. Unlike NIF, POWLA does not provide a specialized property for linking annotations with OLiA concepts. Instead, nodes and relations can be directly assigned an OLiA class (including anonymous classes composed by description-logic operators $\sqcup$, $\sqcap$, $\neg$ and/or property constraints) as an `rdf:type`. POWLA nodes and POWLA relations can thus be modelled as instances of OLiA concepts or OLiA annotation model concepts.

In fact, this has been a motivating factor in providing a reified representation of linguistic annotations rather than ad hoc properties (as in NIF extensions for dependency syntax). To a certain extent, dependency labels and morphosyntactic annotations overlap,[9] so that an a priori, i.e. tagset-independent, split between relational (property) and non-relational (concept) annotations is not possible. Instead, morphological, syntactic and semantic features are represented in the OLiA Reference Model in terms of a single concept hierarchy. A reified representation of relational annotations avoids type-shifting between annotation properties and OLiA concepts. This is necessary to stay within OWL(2) semantics and thus allows the application of OWL(2) reasoners to OLiA/POWLA data.

Unlike NIF, POWLA does thus not provide a vocabulary for specific linguistic categories. OLiA is indeed the recommended vocabulary to represent specific categories. In this way, POWLA is both more minimalistic than NIF annotation classes and properties, and more expressive, as it relies on a rich background vocabulary.

6.3.1.4 Corpus Organization

Aside from modelling linguistic annotations, annotated corpora may require an explicit representation of the organization of annotations into documents and layers. In principle, RDF graphs can be used for this purpose, but an explicit vocabulary is nevertheless necessary to identify what kind of information a particular RDF graph contains and how this relates to other RDF graphs (e.g. pertaining to the same source document, or representing the same kind of annotation for different source documents).

Beyond formalizing linguistic annotations, POWLA provides such a vocabulary to organize corpora. This includes two orthogonal dimensions:

- **dataset structure**: A corpus is composed of individual *documents* and their associated annotations. Documents can be organized into *collections* (e.g. sub-

[9] For example, the Universal Dependencies provide the POS tag `DET` and the dependency label `det`, and '[m]ost commonly, a word of POS `DET` will have the relation `det` and vice versa.' http://universaldependencies.org/u/dep/all.html#al-u-dep/det, accessed 09-07-2019.

corpora). A corpus is regarded as a special case of a collection. Furthermore, a document may be composed of different *parts*, which may in turn be considered independent documents in their own right (e.g. books in the Bible).
- **annotation structure:** Typically, a corpus features multiple types of annotations, e.g. parts of speech along with dependency syntax. Typically, these are produced by independent annotation efforts or using different annotation tools. Annotations of the same type should be organized into a single annotation layer.

For data set structure, POWLA provides two basic concepts, `powla:Document` and `powla:Corpus`. The POWLA document corresponds to an annotation project in PAULA. An annotation project may contain other annotation projects as subdocuments (`hasSubDocument`). If it does not contain other annotation projects, it represents a collection of documents (e.g., a subcorpus, or a pair of texts in a parallel corpus). Otherwise, it contains the annotations of one particular text. In this case, it aggregates different document layers (see below). A corpus is a document that is not a subdocument of another document.

Annotation structure is rendered by means of `powla:Layer`, which serves to group together annotations of the same kind independently from their document. A `powla:DocumentLayer` is a specific layer *within a document*, as expressed by the property `powla:hasDocument`. `DocumentLayer` is a nexus for linking documents and annotations. Nodes and relations can be assigned a document layer (and thus, a document) via `powla:hasLayer`. The property `powla:hasLayer` is recommended for root nodes. A root may have at most one layer.

POWLA layers and documents can be assigned labels by means of subproperties of `powla:hasAnnotation` that correspond to metadata rather than annotations, e.g. date of creation or name of the annotator.

6.3.2 Complementing NIF with POWLA

We illustrate POWLA with annotations for phrasal syntax (Fig. 6.6), named entities (Fig. 6.7) and semantic relations (Fig. 6.8) for an unabridged fragment of `wsj_0655` and its annotation according to the Penn Treebank [39, phrase structure syntax], PropBank [40, semantic roles], OntoNotes named entity types [41], RST Discourse Treebank [42] and automated annotations with DBpedia Spotlight [43]. For presentational reasons, annotation layers are merged and structural annotations are superimposed over each other (Fig. 6.9). Note that the annotations can still be distinguished by means of `powla:hasAnnotation` subproperties. In addition, external vocabulary can be used to encode additional information, e.g. the ITS vocabulary for entity linking[10] or the NERD ontology for entity types.[11]

[10] https://www.w3.org/TR/its20/, last accessed 09-07-2019.

[11] http://nerd.eurecom.fr/ontology, last accessed 09-07-2019.

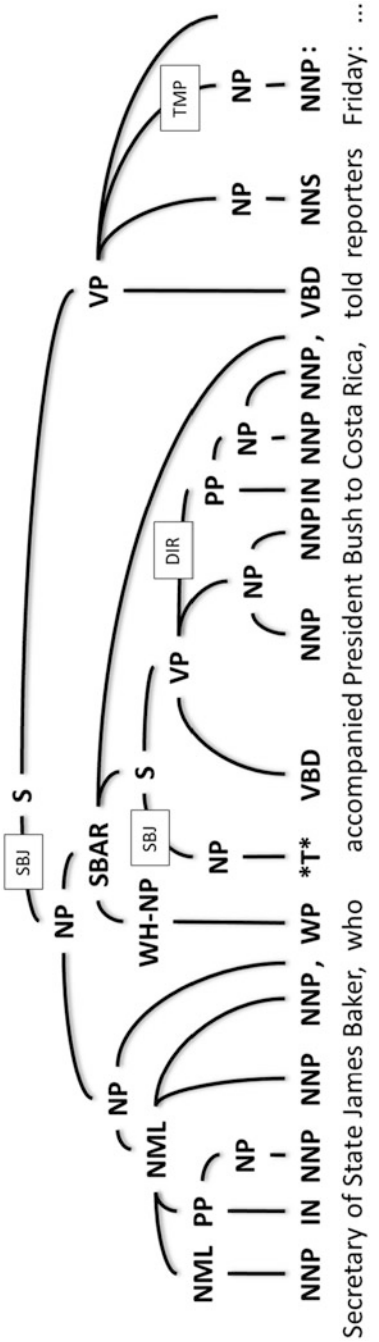

Fig. 6.6 Graph view on syntax annotation of ex. 2 (wsj_0655, OntoNotes **syntax**)

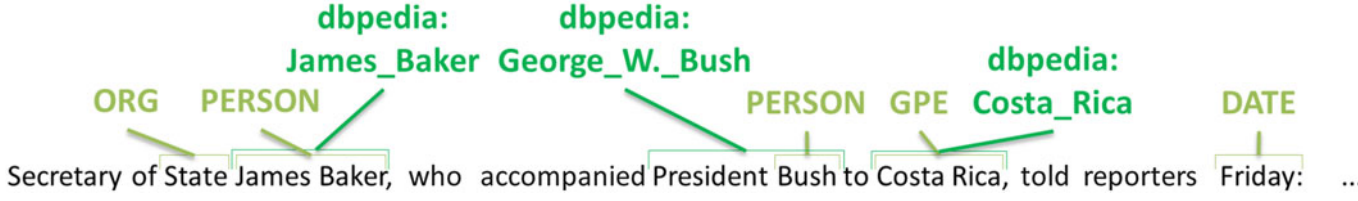

Fig. 6.7 Graph view on named entity annotation of ex. 2 (wsj_0655, manual **named entities** from OntoNotes and automated [and partially incorrect] **entity linking** from DBPedia Spotlight)

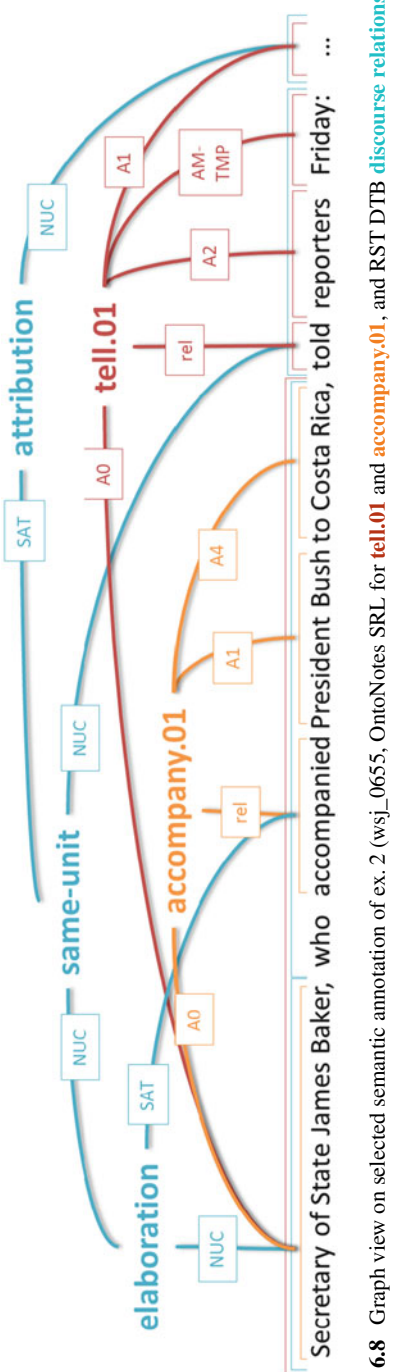

Fig. 6.8 Graph view on selected semantic annotation of ex. 2 (wsj_0655, OntoNotes SRL for **tell.01** and **accompany.01**, and RST DTB **discourse relations**)

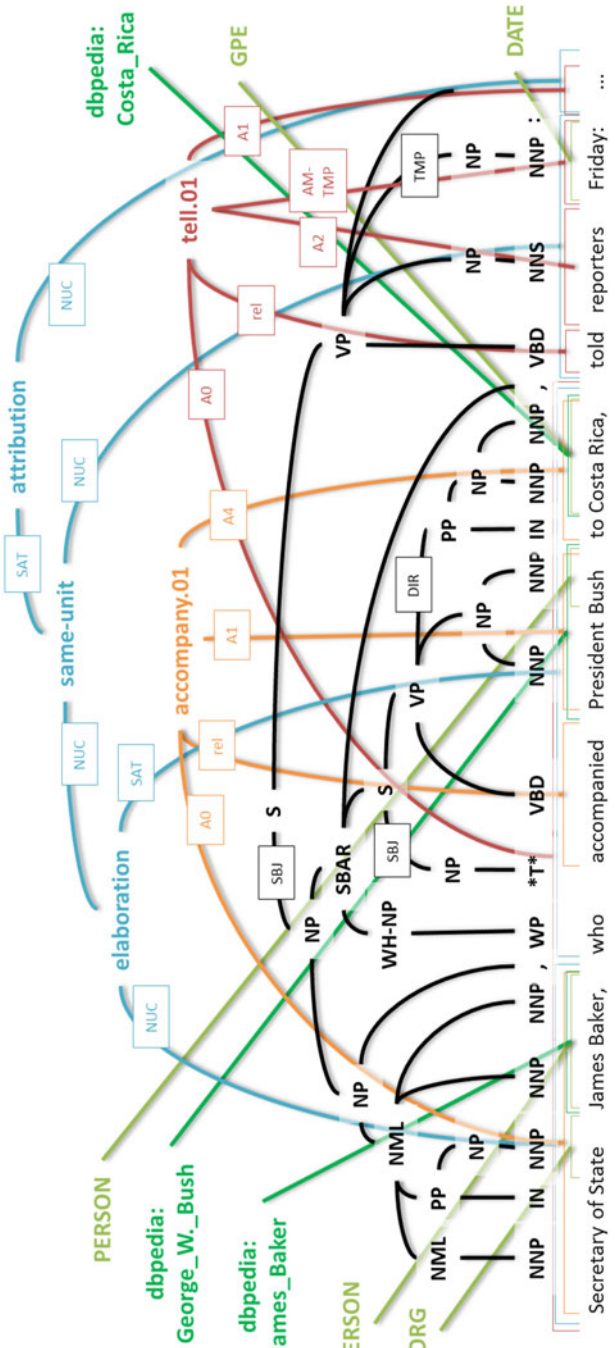

Fig. 6.9 Merged graph from Figs. 6.6, 6.7, and 6.8

```
PREFIX nif: <http://persistence.uni-leipzig.org/nlp2rdf/ontologies/nif-core#>
PREFIX doc: <https://catalog.ldc.upenn.edu/docs/LDC95T7/raw/06/wsj_0655.txt#>
PREFIX powla: <http://purl.org/powla/powla.owl#>
PREFIX itsrdf: <http://www.w3.org/2005/11/its/rdf#>
PREFIX : <https://catalog.ldc.upenn.edu/docs/LDC2007T21/ontonotes-1.0-documentation.pdf#>

# user-defined datatypes
:hasPOS rdfs:subPropertyOf powla:hasAnnotation. # cf. nif:posTag
:hasNER rdfs:subPropertyOf powla:hasAnnotation. # cf. on:EMANEX
:hasCAT rdfs:subPropertyOf powla:hasAnnotation. # phrase labels
:hasNUC rdfs:subPropertyOf powla:hasAnnotation. # NUC/SAT labels
:hasSRL rdfs:subPropertyOf powla:hasAnnotation. # semantic roles
```

Before providing its data, a POWLA data set may need to introduce user-specific subproperties of `powla:hasAnnotation` as shown above.

```
# data
doc:offset_0_9 a powla:Node;
    powla:string "Secretary" ;
    :hasPOS "NNP";
    powla:hasParent _:syntax1;
    nif:nextWord doc:offset_10_12.
```

The first POWLA terminal is based on the NIF part-of-speech annotation sample above (Sect. 5.3.1, p. 79)—albeit NIF-specific information (string class, context, offsets) being omitted. The recommended offsets have been omitted as they can be inferred from the NIF URI. The recommended POWLA type `powla:Node` can in fact be RDFS-inferred from `powla:hasParent` (etc.) and is thus omitted in the following data points. For the syntactic annotation, we create an anonymous (blank) POWLA node `_:syntax1`. This phrase covers exactly this token (and would be conflated with the terminal node in NIF), so that no siblings exist which could be linked by `powla:next`.

```
_:syntax1 powla:string "Secretary";
    :hasCAT "NML";
    powla:hasParent _:syntax2;
    powla:next _:syntax3.
```

The `NML` phrase covers exactly one token, albeit optional, we provide its string value to facilitate readability. Again, `powla:hasParent` links to a POWLA nonterminal, but also, `powla:next` points to the following sibling (relative to the same parent node).

```
doc:offset_10_12 powla:string "of" ;
    :hasPOS "IN";
    powla:hasParent _:syntax3;
    nif:nextWord doc:offset_13_18.

doc:offset_13_18 powla:string "State";
    :hasPOS "NNP";
    :hasNER "ORG";
    powla:hasParent _:syntax4;
    nif:nextWord doc:offset_19_24.

_:syntax4 powla:string "State";
    :hasCAT "NP";
    powla:hasParent _:syntax3.
```

These nodes illustrate additional annotations for named entities (`:hasNER`, corresponding to `on:ENAMEX` above; `:hasCAT` for phrase labels). It should be noted that the decision whether to create a separate nonterminal for every type of annotation or whether to unify coextensional nonterminals is application specific. Here, we opt for unification as this yields a sparse representation.[12]

```
_:syntax3 powla:string "of State";
    :hasCAT "PP";
    powla:hasParent _:syntax2;
    powla:next doc:offset_19_24.
```

This is the first POWLA nonterminal in our data with more than one child node.

```
doc:offset_19_24 powla:string "James";
    :hasPOS "NNP" ;
    powla:hasParent _:syntax2, _:ner1;
    nif:nextWord doc:offset_25_30;
    powla:next doc:offset_25_30.

doc:offset_25_30 powla:string "Baker";
    :hasPOS "NNP" ;
    powla:hasParent _:syntax2, _:ner1;
    nif:nextWord doc:offset_30_31.

_:ner1 powla:string "James Baker";
    :hasNER "PERSON";
    itsrdf:taIdentRef <http://dbpedia.org/resource/
        James_Baker> ;
    a dbo:Agent, dbo:Person, dbo:OfficeHolder .
    # this is a root node not connected to _:syntax2

_:syntax2 powla:string "Secretary of State James Baker";
```

[12] Although not illustrated in the data snippet, coextensional nodes connected by `powla:hasParent` would not be unified. Only the 'highest' non-branching nonterminals per annotation layer would be unified with each other.

```
62      :hasCAT "NML";
63      powla:isSourceOf      # semantic role annotation,
            pointing to predicate
64              [ :hasSRL "A0"; powla:hasTarget doc:offset_36_47
                    ];
65      powla:hasParent _:syntax5;
66      powla:next doc:offset_30_31.
```

For the syntactic phrase and the token span annotated for named entities and entity linking, different nonterminals need to be created as they are not coextensional. In both annotations, however, the sibling structure is identical.

The blank node `_:ner1` carries named entity annotations, but also entity linking (using an external vocabulary in accordance with NIF recommendations). It is important to note here a semantic difference between POWLA and NIF. In NIF, only a *string* can be defined as having the type `dbo:Agent`. This is rather imprecise as it conflates two levels of abstraction: The textual level and the referential level. In POWLA, both levels can be more clearly separated, as POWLA nonterminals are abstract entities (like annotations in Web Annotation), and are thus not a priori incompatible with referential semantics. It is still semantically weak as it enforces a reading that `dbo:Agent` (etc.) is in fact a class of descriptors rather than entities themselves—an interpretation possibly valid in the context of DBpedia, but probably counterintuitive.

The nonterminal `_:ner1` is a root node; it does not have a following sibling (as it does not have a parent) and thus remains isolated from `_:syntax2`.[13] The blank node `_:syntax2` is coextensional with the `A0` span of the predicate `accompany.01`. This is modelled here as a (non-hierarchical) relation pointing from `_:syntax2` to the verb *accompanied*.

```
68  doc:offset_30_31 powla:string ",";
69      :hasPOS ",";
70      powla:hasParent _:syntax5;
71      nif:nextWord doc:offset_32_35.
72
73  _:syntax5 powla:string "Secretary of State James Baker,";
74      :hasCAT "NP";
75      powla:hasParent _:syntax6;
76      powla:next _:syntax8;   # next relative to syntax parent
77      powla:hasParent _:rst1;
78      powla:isSourceOf                # relation label "NUC"
79              [ :hasNUC "NUC"; powla:hasTarget _:rst1 ];
80      powla:next _:syntax8.   # next relative to RST parent
```

[13] It is possible to infer that `_:ner1` is enclosed by `_:syntax2` via the following SPARQL fragment:

```
?ner1 ^powla:hasParent+/powla:hasParent+ ?syntax2.
# at least one descendant of ?ner1 is in ?syntax2
MINUS { ?tmp powla:hasParent+ ?ner1.
MINUS { ?x powla:hasParent ?tmp }
MINUS { ?tmp powla:hasParent+ ?syntax2 }
# there is no terminal descendant of ?ner1 that is not a descendant of ?syntax2
```

This last snippet illustrates a labelled hierarchical relation (i.e. a relation accompanied by `powla:hasParent`). It also shows how the same POWLA node may carry two `powla:hasParent` statements. In principle, a node can also have different `powla:next` statements relative to each parent, illustrated by two `powla:next` triples. In this particular case, the RST sibling and the syntactic sibling are coextensional and have been merged.

6.3.3 Transforming CoNLL-RDF to POWLA

As mentioned above, CoNLL-RDF can be used as an intermediate representation between a specific format and a full-fledged linked data representation. A generic transformation of CoNLL to POWLA can be easily accomplished with CoNLL-RDF and six SPARQL Update rules.

1. Define words and sentences as `powla:Nodes`, but otherwise retain their original annotation.

```
INSERT {
  ?w a powla:Node.
  ?w powla:hasParent ?s.
  ?s a powla:Node.
} WHERE {
  ?w a nif:Word.
  ?w conll:HEAD+ ?s.
  ?s a nif:Sentence.
};
```

   As CoNLL(-RDF) is based on word-level annotations, and supports phrase-level annotations only in the form of string fragments, we define the sentence of any particular word as its parent node.
2. Define the sequential order of nodes. As CoNLL does not provide native support for phrase nodes, map `nif:nextWord` and `nif:nextSentence` to `powla:next`:

```
INSERT {
    ?x powla:next ?y
} WHERE {
    ?x ?rel ?y
    FILTER(?rel in (nif:nextWord,nif:nextSentence))
};
```

3. Define string values and offsets for terminal nodes[14]:

```
INSERT {
    ?w powla:string ?word ;
       powla:start ?start ;
       powla:end ?end .
} WHERE {
    ?w conll:WORD ?word.
    { SELECT ?w (COUNT(DISTINCT ?pre) as ?start)
      WHERE {
        ?pre nif:nextWord+ ?w
      } GROUP BY ?w
    }
    BIND( (?start + 1) as ?end)
};
```

4. Define annotations: Datatype properties from the `conll` namespace are (re)defined as subproperties of `powla:hasAnnotation`. As we cannot directly check for the prefix, we perform a string match on the underlying URI:

```
INSERT {
 ?annotation rdfs:subPropertyOf powla:hasAnnotation
} WHERE {
 ?w ?annotation ?x.
 FILTER(literal(?x) && strstarts(str(?prop),
 'http://ufal.mff.cuni.cz/conll2009-st/task-description.html
     #')
};
```

5. Create relations for dependency syntax: Dependency annotations in CoNLL-RDF are represented by two properties: the obligatory `conll:HEAD` annotation and the optional `conll:EDGE` annotation. In POWLA, these can be complemented by `powla:Relation`:

```
INSERT {
  ?dependent powla:isSourceOf [
    rdfs:label ?edge;
    powla:hasTarget ?head ]
} WHERE {
  ?dependent conll:HEAD ?head.
  OPTIONAL { ?dependent conll:EDGE ?edge }
};
```

[14] We follow CoNLL-RDF conventions for naming the column containing the string value `WORD`. If a different column label is used, the rule needs to be adjusted accordingly. Optionally, the `conll:WORD` property may be deleted to facilitate backward compatibility; we leave it intact here.

Note that POWLA does not define the type of offsets and that we assume the number of preceding tokens. Alternatively, one may use the `STRLEN` function to get an approximate character offset.

In order to avoid confusion between the original `conll:EDGE` property and the annotation attached to the relation; this is represented here by means of an `rdfs:label`.

6. Create relations for semantic roles: Create a novel annotation property and to map relevant CoNLL properties by means of a `FILTER`:

```
INSERT {
  :hasSRL rdfs:subPropertyOf powla:hasAnnotation.
  ?arg powla:isSourceOf [
    :hasSRL ?role;
    powla:hasTarget ?pred ].
} WHERE {
  ?pred ?srl ?arg.
  FILTER(?srl in (syn:ARG0,syn:ARG1,syn:ARG2)) # etc
  BIND(replace(str(?srl),'.*#','') as ?role)
};
```

The resulting representation is compliant with POWLA data structures and thus provides the basis for semantic interoperability. POWLA allows to infer recommended and optional properties and classes from obligatory (and recommended) POWLA properties and classes. With this information, the resulting data structure is suitable for effective navigation.

6.4 Querying Annotated Corpora

One motivation for modelling linguistic annotations in RDF is to facilitate accessibility and integration of linguistic annotations from different sources and across processing tools. Other applications include information integration in NLP pipelines (cf. Chap. 11) and enrichment with external knowledge repositories, which may provide annotation terminology (Sect. 8.4.2) or lexical information about a particular domain.

For demonstrating access to and information retrieval from linguistic annotations provided in RDF, we focus on corpus querying using the POWLA vocabulary as a basis. We demonstrate the capability of SPARQL and POWLA for querying corpora by addressing a number of critical problems used to assess the expressivity of corpus query languages. Lai et al. [44] designed seven reference queries to evaluate the extent to which a query language is capable to match and return specific nodes, to navigate trees, sequences and other relations, and their transitive closure, as well as to demonstrate the need for an update mechanism (cf. our solution to Q4 below). We illustrate the application of (expanded) POWLA+NIF and SPARQL for these queries, slightly adapted to match the example data from Sect. 6.3.2, cf. Fig. 6.9 (p. 107) for a visualization of the data.

Q1 Find sentences that include the word *told*.

For identifying sentences, we resort to syntax annotations, i.e., POWLA nodes without parents and with `:hasCat "S"`.[15]

The corresponding SPARQL query would look as follows:

```
SELECT ?s
WHERE {
    ?s a powla:Root; :hasCAT "S".
    ?told powla:string "told"; powla:hasParent+ ?s.
}
```

As a result, we return the sentence URI. Resolving this URI should point to the underlying string, using the means of Web Annotation or NIF to address text fragments. However, we can also reconstruct the full string of the sentence from POWLA alone:

```
SELECT ?s ?string
WHERE {
    ?s a powla:Root; :hasCAT "S".
    ?told powla:string "told"; powla:hasParent+ ?s.
    { SELECT ?s
      (GROUP_CONCAT(?word; separator=" ") as ?string)
      WHERE {
        ?w a powla:Terminal; powla:hasParent+ ?s;
           powla:start ?start; powla:string ?word.
      } GROUP BY ?s ORDER BY ?s ?start
    }
}
```

Note that this reconstruction is approximative only as a whitespace is inserted between two tokens whereas the original distribution and number of whitespaces between tokens is unknown (and may already have been lost in the source formats, e.g. if provided in CoNLL TSV).

Q2 Find sentences that do not include the word *told*.

Some corpus query languages do not permit existential negation or universal quantification. In SPARQL this is possible as shown in the following query:

```
SELECT ?s
WHERE {
    ?s a powla:Root; :hasCAT "S".
    MINUS { ?told powla:string "told"; powla:hasParent+ ?s }
}
```

Q3 Find noun phrases whose rightmost child is a noun.

This query exploits the fact that `powla:next` holds between siblings only. However, as siblings are defined only in relation to a particular parent node and

[15] Alternatively, one may use NIF data structures and retrieve `nif:Sentences`.

multiple parent nodes are possible, we need to make sure that the next element is also a child of the same noun phrase. For identifying nouns from Penn Treebank annotations, we need to check against different parts of speech and hence use a corresponding `FILTER` expression.

```
SELECT ?np
WHERE {
  ?np :hasCAT "NP"; powla:hasChild ?noun.
  ?noun :hasPOS ?pos FILTER(?pos="NNP" || ?pos="NNS").
  MINUS {
    ?noun powla:next/powla:hasParent ?np
  }
}
```

Q4 Find verb phrases that contain a verb immediately followed by a noun phrase that is immediately followed by a prepositional phrase.

Verbs are identified by several tags in Penn Treebank annotation, but characterized with a 'V' as a first character. This is used here to define a filter condition. A challenge in this query is that it is not restricted to direct child nodes of verb phrases (between `powla:next` would hold), but to any descendant. This is solved here by using `nif:nextWord` (alternatively, start and end indices of terminal nodes may be used). As we compare phrases, we need to rule out that one phrase is contained in another.

```
SELECT ?vp
WHERE {
  ?vp :hasCAT "VP"; powla:hasParent+ ?v, ?np, ?pp.
  ?v  :hasPOS ?pos FILTER(strstarts(?pos,'V')).
  ?np :hasCAT "NP".
  ?pp :hasCAT "PP".
  ?v nif:nextWord/powla:hasParent+ ?np.
  ?np powla:hasChild+/nif:nextWord/powla:hasParent+ ?pp.
  MINUS { ?np powla:hasParent* ?pp }
  MINUS { ?pp powla:hasParent* ?np }
  MINUS { ?v powla:hasParent+ ?np }
  MINUS { ?v powla:hasParent+ ?pp }
}
```

As an alternative, a SPARQL update may be applied to the data beforehand to assign every nonterminal its left- and right-most token and use this precompiled information to formulate the query in easier terms:

```
INSERT {
    ?node :left ?left; :right ?right.
} WHERE {
    ?node powla:hasChild+ ?left, ?right.
    ?left a nif:Word. ?right a nif:Word.
    MINUS { ?node powla:hasChild+/nif:nextWord+ ?left }
    MINUS { ?right nif:nextWord+/powla:hasParent+ ?node }
};
```

Using these task-specific properties, we can simplify the query:

```
SELECT ?vp
WHERE {
  ?vp :hasCAT "VP"; powla:hasParent+ ?v, ?np, ?pp.
  ?v  :hasPOS ?pos FILTER(strstarts(?pos,'V')).
  ?np :hasCAT "NP".
  ?pp :hasCAT "PP".
  ?v nif:nextWord/^:left ?np.
  ?np :right/nif:nextWord/^:right ?pp.
}
```

Q5 Find the first common ancestor of sequences of a noun phrase followed by a verb phrase.

This query requires a result set of all ancestors and its subsequent reduction to the first common ancestor. The variable `?tmp` is introduced to eliminate every descendant of a result candidate which is ancestor to *both* the verb phrase and the noun phrase.

```
SELECT ?ancestor
WHERE {
  ?np :hasCAT "NP".
  ?np powla:hasChild+/nif:nextWord/powla:hasParent+ ?vp.
  MINUS { ?np powla:hasParent* ?vp }
  MINUS { ?vp powla:hasParent* ?np }
  ?ancestor powla:hasChild+ ?np, ?vp.
  MINUS {
    ?ancestor powla:hasChild+ ?tmp.
    ?tmp powla:hasChild+ ?np, ?vp
  }
}
```

Q6 Find a clause which dominates a verb phrase, restricted to cases in which this verb phrase is dominated by an RST nucleus.

For this query, we assume an integrated representation of RST and syntax annotations, i.e. that RST segments point to the largest constituents they contain, not directly to strings (cf. lines 73–80 in the listing on p. 110). In Penn Treebank annotation, clauses are identified by category labels starting with `S`.[16]

```
SELECT ?clause
WHERE {
  ?clause :hasCAT ?cat FILTER(strstarts(?cat,'S')).
  ?clause powla:hasChild ?vp.
  ?vp :hasCAT "VP".
```

[16] Q6 is reformulated from an example in phonology. It is designed to show that queries can be run across different annotation layers. In fact, our version is slightly more complex than Lai et al.'s original as we do not just need to check the annotations of the RST node, but the annotation of its relation.

```
  ?vp powla:hasParent ?rst.
  ?rstrel powla:hasSource ?vp;
          powla:hasTarget ?rst;
          :hasNUC "NUC".
}
```

Q7 Find a noun phrase dominated by a verb phrase. Return the subtree dominated by that noun phrase only.

```
SELECT ?np
WHERE {
  ?vp :hasCAT "VP"; powla:hasChild ?np.
  ?np :hasCAT "NP".
}
```

The challenge here is that corpus query languages often do not allow to constrain the returned result, but that they rather return the entire context for a match to the query. Given the expressivity of SPARQL, it should not come as a surprise that each of these seven queries can be expressed in a relatively straightforward fashion, i.e. with a little more complexity than the solutions in existing query languages provided by Lai et al. [44]. Unlike the languages compared by Lai et al., however, *every single* query can be expressed by SPARQL.

A clear benefit of the combined application of SPARQL and LLOD-native vocabularies to represent linguistic data structures in RDF is that queries can be more easily ported over different data sources, and that information conveyed in corpora can be flexibly combined with external knowledge sources.

Beyond expressivity, portability and resource integration, a great benefit in comparison to common corpus query languages is that query fragments can be precompiled in the data (cf. Q4), so that query-specific optimizations can be developed on the fly. Indeed, the direct application of SPARQL (and other RDF query languages) to linguistic corpora has been suggested repeatedly [9, 24, 45, 46].

Yet, SPARQL also has disadvantages in comparison to existing corpus query languages:

- The semantic restrictions of existing corpus language queries are motivated by the need to guarantee adequate response times. With SPARQL being semantically unconstrained, the user is responsible to keep an eye on run time.[17]

[17] In fact, advanced users need to acquire some insight into the inner mechanics of SPARQL to optimize queries and to guard themselves against unexpected results. As such, SPARQL is evaluated in a bottom-up fashion: {}-enclosed graph fragments are compiled first, and then joined with the context set. In order to prevent large result sets for optional or alternative statements, users should take care to make the relation between all variables in, say, an `OPTIONAL` fragment explicit—even if redundant with information expressed before. Moreover, SPARQL engines normally evaluate the query left to right. This means that filters on a variable should only be expressed after this variable is bound, and queries can be optimized by positing the most restrictive conditions first.

- Most RDF databases do not come with graphical visualizations that are appealing to linguists. For this purpose, external tools can be used, but shifting between different applications is relatively inconvenient.
- Many corpus information systems provide user-friendly graphical query builders. It would be possible to develop such a system for SPARQL, but for linguistics-specific visualization, the supported vocabulary needs to be restricted.
- SPARQL has a rather technical appeal in that it is neither developed by nor normally taught to linguists.

Given the state of the art in corpus linguistics, an appealing solution to these problems is to continue to use existing corpus query infrastructures, but to provide SPARQL generators that allow to replicate them against a triple store if data from different sources is to be integrated or a specific problem requires a level of semantic expressivity that the original corpus query does not provide:

- Such a hybrid solution can be used by anyone familiar with existing query languages.
- SPARQL queries generated from semantically constrained corpus query languages lead to more constrained (i.e. more effective) queries.
- If a specific visualization is needed (e.g. to fine-tune query results), the same query can be run against existing infrastructure.

In this way, a linguist can design a query, e.g. to extract features to be fed into an NLP pipeline, and the generated SPARQL query can then be refined if a higher level of expressivity is needed than provided by the original query language. Such reconstructions of corpus query languages in SPARQL have been described by Chiarcos et al. [23] and [46]. Given the increasing shift in the language resource community from XML-based to graph-based corpus infrastructures [47, 48], as well as the growing maturity of RDF-based NLP service architectures (Chap. 11), we expect an increasing interest in this line of research.

6.5 Summary and Further Reading

This chapter has described mechanisms and data models to represent linguistic annotations in RDF, be it for processing and integration with Semantic Web technology and resources or for publication of corpora as Linked Open Data. Both functions overlap, but have very different requirements:

- For applications where backward compatibility with existing community standards is important, a direct reconstruction of the respective format (resp., its semantics) in RDF(S) provides a viable way to facilitate the processing and publication of data. This was illustrated for the CoNLL format and the use of CoNLL-RDF for annotation manipulation. CoNLL-RDF may be serialized back into CoNLL, or, alternatively, transformed into a full-fledged LLOD representation as required by the second use case.

- For applications aiming to facilitate interoperability and resource integration, and for publication of annotated data as LLOD, we recommend the use of generic data models for linguistic data structures. At the time of writing, no widely used community standard for this purpose has emerged yet. We described the POWLA vocabulary which may represent a basis from which such a vocabulary may eventually evolve, and its application for querying linguistic annotations.

In the wider context of LLOD vocabularies, CoNLL-RDF and POWLA complement NIF (resp., Web Annotation) in that they extend them with application-specific, resp., general data structures for representing linguistic annotations. Aside from other applications, it is thus possible to model, to share and to access linguistically annotated corpora on the web of data in an interoperable way. At the time of writing, it is not uncommon to encounter ad hoc formats to model annotations in RDF,[18] or to devise novel, application-specific formats to integrate specific annotations into a coherent representation.[19] In the longer perspective, however, we expect a tendency towards greater harmonization, and ultimately, convergence towards more widely used conventions, most probably based on widely used community standards rather than proprietary or task-specific formats. For open data and data which can be related to LLOD resources, RDF represents a useful publication format, also because it allows to separate the data model from its serialization—be it RDF/Turtle, RDF/XML, JSON-LD, RDF-Thrift or a TSV rendering as produced from CoNLL-RDF. Depending on the respective use case, other serializations can be generated in a lossless fashion.

As for further reading on linguistic annotations, we suggest consulting the *Handbook of Linguistic Annotation* [51] which provides a general overview and a detailed discussion of most representative formats currently being used for linguistic annotation in various modalities. With respect to linguistic annotations in the context of linguistic linked (open) data, it is hard to give a specific recommendation at this time, as the field is evolving at a fast pace. Aside from the present volume, another upcoming publication on the topic includes the book edited by Pareja-Lora et al. [52]. As a general suggestion, the interested reader may want to follow up on proceedings and associated publications of the workshop series on Linked Data in Linguistics (LDL, biannually, since 2012) and the conference series on Language, Data and Knowledge (LDK, biannually, since 2017), as these take the most specific focus on semantic technologies in application to language technology. Beyond this, it is possible to get directly involved, e.g., in the context of the W3C community group on Linked Data for Language Technology Community Group (LD4LT),[20] which provides a mailing list for discussing these issues.

[18] For example, the MATE SRL system [49] provides an ad hoc representation in RDF/N3 under http://barbar.cs.lth.se:8081/, accessed 09-07-2019.

[19] For example, the Concrete format [50], designed as an application of Apache Thrift, or JSON-NLP [53], designed as a JSON replacement of earlier generic formats for NLP.

[20] https://www.w3.org/community/ld4lt/, accessed 09-07-2019.

References

1. N. Ide, C. Chiarcos, M. Stede, S. Cassidy, Designing annotation schemes: from model to representation, in *Handbook of Linguistic Annotation*, ed. by N. Ide, J. Pustejovsky. Text, Speech, and Language Technology (Springer, Berlin, 2017)
2. S. Bird, M. Liberman, A formal framework for linguistic annotation. Speech Commun. **33**(1–2), 23 (2001)
3. N. Ide, K. Suderman, The Linguistic Annotation Framework: a standard for annotation interchange and merging. Lang. Resour. Eval. **48**(3), 395 (2014)
4. ISO, ISO 24612:2012. Language resource management—Linguistic Annotation Framework. Technical Report, ISO/TC 37/SC 4, Language resource management (2012). https://www.iso.org/standard/37326.html
5. N. Ide, K. Suderman, GrAF: a graph-based format for linguistic annotations, in *Proceedings of the 1st Linguistic Annotation Workshop (LAW 2007)*, Prague, 2007, pp. 1–8
6. C. Chiarcos, S. Dipper, M. Götze, U. Leser, A. Lüdeling, J. Ritz, M. Stede, A flexible framework for integrating annotations from different tools and tag sets. TAL (Traitement Automatique des Langues) **49**(2), 217 (2008)
7. W. Bosma, P. Vossen, A. Soroa, G. Rigau, M. Tesconi, A. Marchetti, M. Monachini, C. Aliprandi, KAF: a generic semantic annotation format, in *Proceedings of the 5th International Conference on Generative Approaches to the Lexicon GL 2009*, Pisa, 2009
8. R. Eckart, Choosing an XML database for linguistically annotated corpora, in *Sprache und Datenverarbeitung. Proceedings of the KONVENS 2008 Workshop on Datenbanktechnologien für Hypermediale Linguistische Anwendungen*, Berlin, 2008
9. A. Burchardt, S. Padó, D. Spohr, A. Frank, U. Heid, Formalising multi-layer corpora in OWL/DL—Lexicon modelling, querying and consistency control, in *Proceedings of the 3rd International Joint Conference on NLP (IJCNLP)*, Hyderabad, 2008, pp. 389–396
10. S. Cassidy, An RDF realisation of LAF in the DaDa annotation server, in *Proceedings of the 5th Joint ISO-ACL/SIGSEM Workshop on Interoperable Semantic Annotation (ISA-5)*, Hong Kong, 2010
11. A. Fokkens, A. Soroa, Z. Beloki, N. Ockeloen, G. Rigau, W.R. van Hage, P. Vossen, NAF and GAF: linking linguistic annotations, in *Proceedings of the 10th Joint ISO-ACL SIGSEM Workshop on Interoperable Semantic Annotation* (2014), pp. 9–16
12. E. Rubiera, L. Polo, D. Berrueta, A. El Ghali, TELIX: an RDF-based model for linguistic annotation, in *Proceedings of the 9th Extended Semantic Web Conference (ESWC 2012)*, Heraklion, 2012
13. S. Hellmann, J. Lehmann, S. Auer, M. Brümmer, Integrating NLP using linked data, in *Proceedings of the 12th International Semantic Web Conference (ISWC)*. Lecture Notes in Computer Science, vol. 8219 (Springer, Heidelberg, 2013), pp. 98–113
14. N. Ide, K. Suderman, E. Nyberg, J. Pustejovsky, M. Verhagen, LAPPS/Galaxy: current state and next steps, in *Proceedings of the 3rd International Workshop on Worldwide Language Service Infrastructure and 2nd Workshop on Open Infrastructures and Analysis Frameworks for Human Language Technologies (WLSI/OIAF4HLT2016)* (2016), pp. 11–18
15. O. Christ, A modular and flexible architecture for an integrated corpus query system, in *Proceedings of the 3rd Conference on Computational Lexicography and Text Research (COMPLEX'94)*, Budapest, 1994
16. A. Kilgarriff, V. Baisa, J. Bušta, M. Jakubíček, V. Kovář, J. Michelfeit, P. Rychlý, V. Suchomel, The Sketch Engine: ten years on. Lexicography **1**(1), 7 (2014). https://doi.org/10.1007/s40607-014-0009-9
17. C. Chiarcos, C. Fäth, CoNLL-RDF: Linked corpora done in an NLP-friendly way, in *Proceedings of the 1st International Conference on Language, Data, and Knowledge, LDK 2017*, ed. by J. Gracia, F. Bond, J.P. McCrae, P. Buitelaar, C. Chiarcos, S. Hellmann (Springer, Cham, 2017), pp. 74–88. https://doi.org/10.1007/978-3-319-59888-8_6

18. J. Nivre, Ž. Agić, L. Ahrenberg, et al., Universal dependencies 1.4 (2016). http://hdl.handle.net/11234/1-1827
19. S. Brants, S. Hansen, Developments in the TIGER annotation scheme and their realization in the corpus, in *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC)*, Las Palmas, 2002, pp. 1643–1649
20. W. Lezius, H. Biesinger, C. Gerstenberger, TigerXML quick reference guide. Technical Report, IMS, University of Stuttgart (2002)
21. K.K. Schuler, VerbNet: a broad-coverage, comprehensive verb lexicon. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA (2005). AAI3179808
22. J. Eckle-Kohler, J. McCrae, C. Chiarcos, *lemonUby*—a large, interlinked, syntactically-rich resource for ontologies. Semantic Web J. **6**(4), 371 (2015)
23. C. Chiarcos, Interoperability of corpora and annotations, in *Linked Data in Linguistics*, ed. by C. Chiarcos, S. Nordhoff, S. Hellmann (Springer, Heidelberg, 2012), pp. 161–179
24. C. Chiarcos, POWLA: modeling linguistic corpora in OWL/DL, in *Proceedings of the 9th Extended Semantic Web Conference (ESWC-2012)*, Heraklion, 2012, pp. 225–239
25. N. Mazziotta, Building the syntactic reference corpus of medieval French using NotaBene RDF annotation tool, in *Proceedings of the 4th Linguistic Annotation Workshop* (Association for Computational Linguistics, Stroudsburg, 2010), pp. 142–146
26. S. Hellmann, J. Lehmann, S. Auer, M. Brümmer, Integrating NLP using linked data, in *Proceedings of the 12th International Semantic Web Conference, 21–25 October 2013*, Sydney, 2013. Also see http://persistence.uni-leipzig.org/nlp2rdf/
27. S. Dipper, M. Götze, Accessing heterogeneous linguistic data—generic XML-based representation and flexible visualization, in *Proceedings of the 2nd Language & Technology Conference 2005*, Poznan, 2005, pp. 23–30
28. M.G. Stefanie Dipper, 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*, Trento, 2006
29. N. Ide, L. Romary, International standard for a Linguistic Annotation Framework. Nat. Lang. Eng. **10**(3–4), 211 (2004)
30. N. Ide, K. Suderman, GrAF: a graph-based format for linguistic annotations, in *Proceedings of the Linguistic Annotation Workshop*. Prague (Association for Computational Linguistics, Stroudsburg, 2007), pp. 1–8
31. M. Stede, H. Bieler, S. Dipper, A. Suriyawongk, Summar: combining linguistics and statistics for text summarization, in *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI)*, Riva del Garda, 2006, pp. 827–828
32. A. Zeldes, J. Ritz, A. Lüdeling, C. Chiarcos, ANNIS: a search tool for multi-layer annotated corpora, in *Corpus Linguistics*, Liverpool, 2009, pp. 20–23
33. F. Zipser, L. Romary, A model oriented approach to the mapping of annotation formats using standards, in *Proceedings of the Workshop on Language Resources and Language Technology Standards, collocated with LREC (LR&LTS 2010)*, Valetta, 2010
34. N. Ide, C.F. Baker, C. Fellbaum, C.J. Fillmore, R. Passonneau, MASC: the manually annotated sub-corpus of American English, in *Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC-2008)*, Marrakech, 2008, pp. 2455–2461
35. D.A. de Araujo, S.J. Rigo, J.L.V. Barbosa, Ontology-based information extraction for juridical events with case studies in Brazilian legal realm. Artif. Intell. Law **25**(4), 379 (2017)
36. C. Chiarcos, C. Fäth, Graph-based annotation engineering: towards a gold corpus for Role and Reference Grammar, in *Proceedings of the 2nd Conference on Language, Data and Knowledge (LDK)*. OpenAccess Series in Informatics (Schloss Dagstuhl, Leibniz-Zentrum fuer Informatik, 2019)
37. C. Chiarcos, B. Kosmehl, C. Fäth, M. Sukhareva, Analyzing Middle High German syntax with RDF and SPARQL, in *Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC)* (Miyazaki, Japan, 2018)
38. T. Krause, U. Leser, A. Lüdeling, graphANNIS: a fast query engine for deeply annotated linguistic corpora. J. Lang. Technol. Comput. Linguist. **31**(1), 1 (2016)

39. M. Marcus, B. Santorini, M.A. Marcinkiewicz, Building a large annotated corpus of English: the Penn Treebank. Comput. Linguist. **19**(2), 313 (1993)
40. P. Kingsbury, M. Palmer, From TreeBank to PropBank, in *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC)*, Las Palmas, 2002
41. E. Hovy, M. Marcus, M. Palmer, L. Ramshaw, R. Weischedel, OntoNotes: the 90% solution, in *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology (HLT-NAACL)* (Association for Computational Linguistics, New York, 2006), pp. 57–60
42. L. Carlson, D. Marcu, M.E. Okurowski, Building a discourse-tagged corpus in the framework of Rhetorical Structure Theory, in *Current and New Directions in Discourse and Dialogue*, ed. by J. van Kuppevelt, R. Smith. Text, Speech, and Language Technology, vol. 22, chap. 5 (Kluwer, Dordrecht, 2003)
43. P. Mendes, M. Jakob, A. García-Silva, C. Bizer, DBpedia SpotLight: shedding light on the web of documents, in *Proceedings of the 7th International Conference on Semantic Systems (I-Semantics 2011)*, Graz, 2011
44. C. Lai, S. Bird, Querying and updating treebanks: a critical survey and requirements analysis, in *Proceedings of the Australasian Language Technology Workshop* (2004), pp. 139–146
45. M. Kouylekov, S. Oepen, Semantic technologies for querying linguistic annotations: an experiment focusing on graph-structured data, in *Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC)* (Reykjavik, Iceland, 2014)
46. A. Frank, C. Ivanovic, Building literary corpora for computational literary analysis—a prototype to bridge the gap between CL and DH, in *Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC)*, Miyazaki, May 7–12, 2018
47. P. Banski, J. Bingel, N. Diewald, E. Frick, M. Hanl, M. Kupietz, P. Pezik, C. Schnober, A. Witt, KorAP: the new corpus analysis platform at IDS Mannheim, in *Proceedings of the 6th Language & Technology Conference on Human Language Technology Challenges for Computer Science and Linguistics*, December 7–9, 2013, Poznan, (2014), pp. 586–587
48. T. Krause, U. Leser, A. Lüdeling, graphANNIS: a fast query engine for deeply annotated linguistic corpora. JLCL **31**(1), 1 (2016)
49. B. Bohnet, J. Kuhn, The best of both worlds: a graph-based completion model for transition-based parsers, in *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics* (Association for Computational Linguistics, Stroudsburg, 2012), pp. 77–87
50. F. Ferraro, M. Thomas, M.R. Gormley, T. Wolfe, C. Harman, B. Van Durme, Concretely annotated corpora, in *Proceedings of the AKBC Workshop at NIPS* (2014)
51. N. Ide, J. Pustejovsky (eds.), *Designing Annotation Schemes: From Model to Representation*. Text, Speech, and Language Technology (Springer, Berlin, 2017)
52. A. Pareja-Lora, M. Blume, B. Lust, C. Chiarcos (eds.), *Development of Linguistic Linked Open Data Resources for Collaborative Data-Intensive Research in the Language Sciences* (MIT Press, Cambridge, 2019)
53. D. Cavar, O. Baldinger, U.M. Joshua Herring, Y. Zhang, S. Bedekar, S. Panicker, An annotation encoding schema for natural language processing using JSON: NLP JSON schema version 0.1, November 2018. Technical Report, Indiana University (2018)