Ontology-Based XQuery'ing of XML-Encoded Language Resources on Multiple Annotation Layers

Georg Rehm[1], **Richard Eckart**[2], **Christian Chiarcos**[3], **Johannes Dellert**[1]

University of Tübingen, Germany[1]
SFB 441: Linguistic Data Structures
Nauklerstr. 35, Tübingen

Technische Universität Darmstadt, Germany[2]
Department of English Linguistics
Hochschulstr. 1, Darmstadt

University of Potsdam, Germany[3]
SFB 632: Information Structure
Karl-Liebknecht-Str. 24-25, Potsdam

Corresponding author: georg.rehm@uni-tuebingen.de

Abstract

We present an approach for querying collections of heterogeneous linguistic corpora that are annotated on multiple layers using arbitrary XML-based markup languages. An OWL ontology provides a homogenising view on the conceptually different markup languages so that a common querying framework can be established using the method of ontology-based query expansion. In addition, we present a highly flexible web-based graphical interface that can be used to query corpora with regard to several different linguistic properties such as, for example, syntactic tree fragments. This interface can also be used for ontology-based querying of multiple corpora simultaneously.

1. Introduction

Annotated linguistic corpora can be used in several different scenarios: they can be employed in machine learning contexts to serve as training data, they can be used to build language models based on statistical properties, or they can serve as resources in computer-assisted language learning software. In fact, there are so many possible ways in which corpora can be used effectively that, nowadays, their initial purpose has become overshadowed. Traditionally, linguists compiled corpora in order to answer research questions on the basis of empirical evidence. After a corpus had been compiled, it could be analysed using statistical methods.

We are concerned with devising a web-based corpus platform for a collection of ca. 60 heterogeneous linguistic corpora. One of the obstacles we are confronted with is providing homogeneous means of accessing this very large collection of diverse and complex linguistic resources. The user interface does not only have to generalise over several heterogeneous annotation formats, it has to be intuitively usable for linguists without expertise in XML, querying standards such as XQuery, or even the original markup languages. In other words, we want to lay a technical foundation for the interoperability and reusability of annotated linguistic corpora. We would like to enable academics who are not interested in the corpus annotation specifics to log onto the platform and to explore as well as to query the available corpora in an efficient and simple way.

This article is a follow-up to (Rehm et al., 2007) and presents several new results. Section 2 briefly highlights the most important properties of data formats for linguistic corpora and our generic data model. Section 3 sketches the general approach, our system architecture, and the process flows. The following sections discuss the platform's query interface: first, we illustrate the technical aspects of querying multi-rooted trees (section 4). We subsequently introduce an ontology-based approach for homogenising the heterogeneous markup languages (section 5). Finally, we describe the browser-based graphical interface and the output and visualisation modules (section 6).

2. A Homogeneous Data Model

Since the late 1990s, practically all corpus annotation formats have been realised as XML markup languages (Ide et al., 2000; Sperberg-McQueen and Burnard, 2002; Lehmberg and Wörner, 2007). They come in two different flavours: traditionally, most corpus markup languages form hierarchies that are expressed by nested XML element trees (e. g., for the representation of syntactic constituents or document structures). In stark contrast to hierarchical data formats are markup languages that anchor a data set to a timeline, primarily used for spoken language (Bird and Liberman, 2001). In timeline-based formats such as Exmaralda (Schmidt, 2005), arcs can be drawn from one anchor to another point on the timeline. However, these structures are not represented by nested XML element-trees, but with the help of attribute-value pairs. At the same time, both approaches usually encode several annotation layers concurrently, for example, information on morphological, syntactic, semantic, and pragmatic structures.

In our project we have to deal with both hierarchical and timeline-based corpora and we have to provide the means for enabling users to query both types of resources in a uniform way. In fact, the original annotation format is more or less irrelevant to the user, as the graphical user interface and the underlying technology abstract from any idiosyncrasies and peculiarities of the original data formats. We use an approach that is able to achieve the abovementioned goals (Dipper et al., 2006; Schmidt et al., 2006; Wörner et al., 2006) and that can be compared to the NITE Object Model (Carletta et al., 2003). We developed a tool that semi-automatically splits hierarchically annotated corpora that typically consist of a single XML document instance into individual XML files, so that each file represents all the information related to a single annotation layer (Witt et al., 2007; Rehm et al., 2008); this approach guarantees that overlapping structures can be represented straightforwardly. Timeline-based corpora are processed using another tool in order to separate the graph annotations that are also stored in individual XML files (Witt et al., 2007). Our approach enables us to represent arbitrary types of XML-annotated corpora as individual XML element trees. These

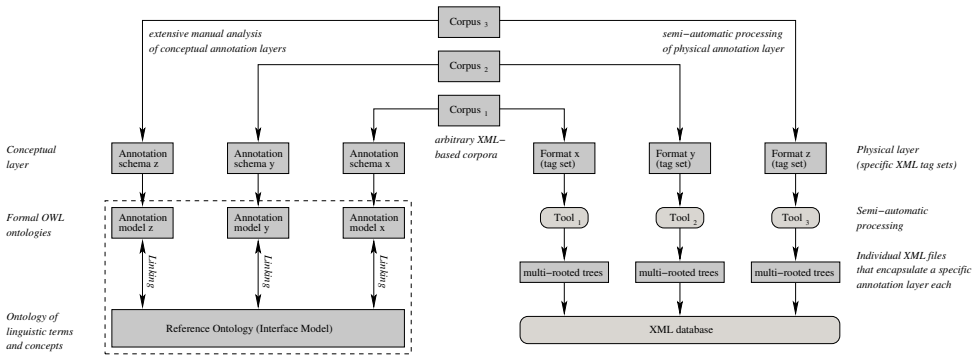

Figure 1: The two main corpus processing workflows

multi-rooted trees are represented as regular XML document instances, but, as a single corpus comprises *multiple* files, there is a need to go beyond the functionality offered by typical XML tools in order to enable us to process multiple files, as regular tools work with single files only.

3. System Architecture

A corpus to be imported into our corpus platform has to be analysed manually first (figure 1). Depending on its markup language, the XML document instance is transformed into multi-rooted trees.

Some corpora can be transformed using simple XSLT stylesheets, while other corpora have to be processed using a custom set of tools: corpora annotated based on the hierarchical model are analysed by a tool that enables us to map XML elements, attributes and textual PCDATA content onto one or more annotation layers (Witt et al., 2007). As soon as this mapping exists, the annotation layers can be exported as XML documents. A second tool can be used to split timeline-based corpora into a set of multi-rooted trees. Finally, these XML files are imported into an XML database (we currently use eXist but plan to evaluate MonetDB/XQuery, Qizx/db, and Sedna soon). A third tool anchors all files to a set of primary data in order to allow query-time coordination between these individual files that represent a single-rooted tree each.

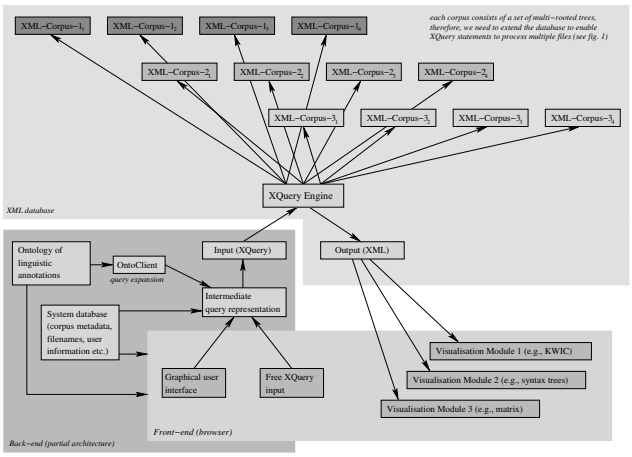

Figure 2: Architecture of the web-based query interface

At the same time, the elements and attributes used in the markup languages are analysed and incorporated into a set of ontologies that encapsulate knowledge about linguistic terms and concepts. The ontologies are used to generalise over the specific and, at times, idiosyncratic names and labels used in the corpus markup languages and to provide a coherent, unified, and homogeneous perspective on the large set of heterogeneous corpora.

The web-based query interface we are currently developing has several requirements. For this paper the two most important issues are the implementation of a mechanism that enables XQuery queries that work on multi-rooted trees (section 4) and the integration of the ontologies of linguistic annotations into the process of building an XQuery statement (section 5). Our web-based GUI can be intuitively applied by linguists and other interested parties who know neither XML, XQuery, nor the XML-based markup languages used in the original corpora (section 6). Figure 2 shows the architecture of the query interface. Figures that explain additional aspects of the system can be found in (Rehm et al., 2008).

4. Querying Multi-Rooted Trees

As each annotation layer is contained in one XML document, a corpus represents a special form of a multi-rooted tree, i. e., a collection of trees that do not share nodes except for the leaves that contain the annotated primary data. AnnoLab (Eckart and Teich, 2007) is an XML/XQuery-based corpus query and management framework that was specifically designed to deal with multi-rooted trees.

To avoid problems regarding projectiveness and overlapping segments, AnnoLab uses a stand-off adaptation of the XML data-model. This adaptation substitutes the text-nodes from the XML data-model using *segments* serving as placeholders for the *signal* (content), thus functioning as stand-off *anchors*. A segment addresses a signal using start and end offsets as well as a *signal identifier*. The remainder of the XML data-model remains untouched and compatible with standard XPath and XQuery.

The native XML database eXist, written in Java, is currently used as a storage and query host because it is Open Source software and it offers an easy integration with AnnoLab which is also implemented in Java.

4.1. XQuery Extensions

To access signals and to perform queries across multiple layers, XQuery extensions are necessary. These fall into the categories *signal access* and *segment coordination*.

4.1.1. Signal Access

Signal access extensions deal with retrieving the signal addressed by a particular segment and with finding parts of signals that correspond to a specific search pattern. There are three typical usage scenarios:

- Fetch the content addressed by segments. `get-text(N)` gets all content addressed by segment descendants of `N`.
- Retrieve segments that address content that matches a pattern. `find-text(S, p)` generates segments for matches of `p` within the signals `S`.
- Match annotation elements against content. By repeating text inside segments, this is directly supportable by the eXist native full-text index which is accessible through some eXist-specific operators, e. g., `//tok[. &= 'will']` searches for all `tok` elements that contain the word `will`. These operators speed up search but use an inverted index that is generated using a tokenizer which does not necessarily match the tokenization defined by the annotation.

4.1.2. Segment Coordination

These functions perform comparisons and calculations on *segments*. The extensions can be used to specify the desired relations between *segments* originating from different annotation layers and, thus, to coordinate different layers. The function `containing(X, Y)` illustrates the general principle: it takes two sets of elements `X` and `Y`. For each element in the two lists a cover segment is generated starting at the smallest offset and ending at the largest offset referenced by any segment below them. Finally, any element from `X` is returned which contains any of the elements in `Y`.

4.2. Test Corpus

To illustrate querying and to gather initial performance results we devised eight queries based on the German treebank TüBa-D/Z (Telljohann et al., 2004). For this experiment our version of TüBa-D/Z consisting of 1285 documents was split into six layers (7710 XML files): (1) *Clause* – sentences, clauses; (2) *Discourse* – anaphora, coreference relations; (3) *Field* – topological fields; (4) *Lexical* – tokenization, parts-of-speech; (5) *Named Entities*; (6) *Phrase* – phrase structure.

4.3. Example Queries

The queries PQ1–PQ5, based on tests presented by (Dipper et al., 2007a), are single-layer queries. PQ1 and PQ2 operate on the *Lexical* layer, the others use the *Phrase* layer.

- PQ1 – Find all sentences that contain the word `kam`.
- PQ2 – Find all sentences that do not contain `kam`.
- PQ3 – Find all noun phrases. Return the references to those noun phrases.
- PQ4 – Find all noun phrases. Return the subtrees dominated by these noun phrases.
- PQ5 – Find all noun phrases dominated by a verb phrase. Return the subtrees dominated by those NPs.

Figure 3 shows PQ5. First, all XML documents of the *Phrase* layer are fetched. The function `ds:layer()` is a shortcut to fetch all XML documents that belong to the specified layer. Layers are stored in the database as one XML file per layer per signal. Next, the noun phrase nodes (`NX`) that are dominated by verb phrase nodes (`VXFIN` and `VXINF`) are fetched. Finally, the branches dominated by the `NX` nodes are returned, each encapsulated in a `match` tag.
AnnoLab pre-declares several namespace prefixes to house extension functions, e. g., `ds` (datastore functions), `seq` (segment coordination functions), and `oc` (OntoClient related functions).

```
declare namespace leveler="urn:xmlns:sfb441:leveler";
element result {
  let $l := ds:layer('Phrase')
  for $s in $l//ntNode[
      @leveler:category='VXFIN' or
      @leveler:category='VXINF'
    ]/ntNode[@leveler:category='NX']
  return element match { $s }
}
```

Figure 3: PQ5: Find all NPs dominated by a VP

The queries TUEBA1 and TUEBA2 are simple multi-layer queries. They involve the *Lexical* and *Field* layers and use the custom XQuery function `seg:containing()` to relate annotations from both layers to one another. TUEBA2 additionally uses the function `oc:expand()` to retrieve the possible verb tags from an embedded OntoClient.

- TUEBA1 – Find all occurrences of the verb `will` in the Vorfeld.
- TUEBA2 – TUEBA1 using OntoClient (see section 5) to expand the concept `Verb` into possible verb tags.

The query TUEBA2 (figure 4) first fetches all instances of the verb `will` from the *Lexical* layer because token and part-of-speech information is stored there. The function `oc:expand()` is used to get all possible POS tags for verbs (see section 5). In addition, we use the eXist full-text index operator `&=` to find instances of the word `will`. Next, all *Vorfeld* annotations (`VF` tag) are fetched from the *Field* layer. Finally, all *Vorfeld* annotations are filtered out that contain one of the `will` instances we found in the first step and returned surrounded by a `match` tag.

```
declare namespace leveler="urn:xmlns:sfb441:leveler";
let $verb := ds:layer('Lexical')//tok
    [pos/@leveler:text = oc:expand('Verb')]
    [.//orth &= 'will']
let $vf := ds:layer('Field')//ntNode
    [@leveler:category='VF']
let $res := seq:containing($vf, $verb)
return element result {
    for $r in $res return element match { $r }
}
```

Figure 4: TUEBA2: Find the verb `will` in the Vorfeld using an embedded OntoClient

The query BQ2 (figure 5) is based on Q2 presented by (Bird et al., 2006). It makes use of the following-sibling axis.

- BQ2 – Find noun phrases that are immediate following siblings of a verb.

Since POS and phrase structures are kept in different layers, we cannot directly search for noun phrases following a verb. First we fetch all verb tokens from the *Lexical* layer. Next we fetch all VPs from the *Phrase* layer. The TüBa-D/Z guidelines mandate that any verb is located under a VP which would then be sibling to the NP. To make sure that there is no additional content in the VP we use `seq:same-extent` to join the VPs on the verbs we fetched before. On the resulting nodes we search for all immediately following siblings that are NPs. We use the custom function `tree:following-sibling` because in eXist the following sibling axis is rather slow (section 4.4). Finally, the resulting NPs are returned, wrapped in `match` tags.

```
declare namespace lvlr="urn:xmlns:sfb441:leveler";
element result {
  let $verbs := ds:layer('Lexical')//tok[
      starts-with(pos/@lvlr:text, "V")]
  let $VPs := ds:layer('Phrase')//orth
      /parent::ntNode[starts-with(@lvlr:category,'VX')]
  let $candVPs := seq:same-extent($VPs, $verbs)
  let $res := tree:following-sibling($candVPs,1)[
      @lvlr:category = 'NX']
  for $s in $res
  return element match { $s }
}
```

Figure 5: BQ2: Find NPs that are immediate following siblings of a verb

4.4. Performance Evaluation

Using the queries described above we carried out performance measurements on a Pentium IV 3.0 GHz with two 80 GB SATA 7200 rpm hard disks (RAID 1) and 2 GB memory of which 1 GB was allocated to AnnoLab running an embedded eXist.

Table 1 shows the normalized run-time results. The queries were run on sets of 100, 200, 300, 400, 500, and 1000 documents to see how they scale. Numbers were normalized to milliseconds per 1000 tokens (t/kT). Each query was run ten times in a row after a fresh start of the database and the minimum, average, and maximum run-times were taken.

The maximum time was always obtained in the first run which included the initial start-up of the database and initial caching. In an optimally scaling case, the minimum t/kT would remain constant for the different test-set sizes. For the queries BQ2, PQ1, PQ3, PQ4, TUEBA1, and TUEBA2 we observe relatively stable minimum times – they scale quite well. Interestingly for some of these queries the average time even drops with a growing test-set size. The query PQ2 scales very bad. Its maximum time grows rapidly with growing test-set size (up to almost three hours for 1000 documents!) even though its minimum time remains rather low (about 30 seconds for 1000 documents).

Table 2 shows the absolute run-times of the queries. We would like to have near interactive (at most 3 seconds) responses for queries on the whole TüBa-D/Z corpus (1285 documents), but the results show clearly that no query returns within this desired timeframe.

The queries are supported by QName indices defined on the relevant attributes and by the eXist native full-text engine. The segment coordination functions are currently not backed by a specialized index as described by, for example, (Alink et al., 2006), however, we use a fast linear algorithm for `seg:containing` and `seg:same-extent`.

We believe that part of the poor performance is due to eXist currently not having a statistics-backed query optimizer. For example, to find instances of the word `will` eXist uses the full-text index if we specify the `&=` operator and then uses the element QName index on `orth` to find the element by name, or it only uses the QName index if we use the traditional `=` operator. However, eXist does not check if there might be only very few instances of `orth` and many instances of `will`, so it does not prefer searching for `orth` first and falling back to traditional `=` comparison which might be faster. The lack of a sophisticated query optimizer also makes it difficult to write efficient queries. eXist can optimize XPath predicates but completely fails to optimize XQuery `WHERE` statements which would allow for more readable queries.

About one third of BQ2 is spent navigating the following-sibling axis. Using `following-sibling::*[1]` to access the immediate following sibling, eXists wastes a lot of time expanding the complete following-sibling axis and only then selecting the first one. The `tree:following-sibling` function alleviates this by selecting only the *n* following siblings, but instead of doing a batch lookup needs to go to the database once for each VP node instead.

A challenge not covered by our example queries concerns the marking of matches in the query result. The query TUEBA2, for example, results in *Vorfeld* annotations. To the user, however, we want to display whole sentences with the matching *Vorfeld* tags highlighed within them. That means we actually need to query for a sentence containing a *Vorfeld* containing the verb `will` and then mark the *Vorfeld* node – a process which we do not know how to perform efficiently in terms of XQuery. We would need an extension function closely working together with the XML serializer returning the results.

Because of issues such as highlighting of matches, indexed full-text searching and indexed segment coordination an efficient implementation of an XML-based corpus query tool is hardly conceivable using W3C XQuery functionality only. To improve performance, we have already started evaluating alternative XML databases.

5. Creating XQuery Constraints

In order to provide a consistent approach for documentation and to enable a uniform query interface that applies to different annotation formats, we built an ontology that serves as a terminological reference, represented in OWL DL. This *reference model* is based on the EAGLES recommendations for morphosyntax, the general ontology for linguistic description (Farrar and Langendoen, 2003), and the SFB632 annotation standard (Dipper et al., 2007b). It covers reference specifications for word classes, and morphosyntax (Chiarcos, 2007), and is currently extended to syntax and information structure.

For the sustainable operationalisation of existing annotation schemes which then enables us to define annotation-independent corpus queries, we employ the structured model of ontologies of linguistic annotation (OLiA). The core idea of the OLiA architecture is a clear separation

	minimum						*average*						*maximum*					
	100	200	300	400	500	1000	100	200	300	400	500	1000	100	200	300	400	500	1000
BQ2	251	435	421	391	379	418	307	466	438	411	397	428	558	604	537	489	458	474
PQ1	3	14	15	30	30	31	29	43	51	90	110	138	239	284	372	618	810	1093
PQ2	45	56	65	71	74	78	95	395	861	1231	1563	2897	429	3376	7961	11629	14931	28253
PQ3	32	38	36	37	37	38	71	59	51	54	52	51	297	195	165	160	147	133
PQ4	42	47	44	47	49	50	71	62	54	56	57	55	209	132	107	106	95	78
PQ5	5	8	8	8	9	12	24	22	16	16	18	16	95	70	52	53	52	40
TUEBA1	15	21	24	24	23	23	56	45	41	42	40	38	261	180	146	159	164	138
TUEBA2	113	120	118	118	116	119	153	142	135	131	129	130	361	261	215	214	207	179

Table 1: Query performance in milliseconds per 1000 tokens

	minimum						*average*						*maximum*					
	100	200	300	400	500	1000	100	200	300	400	500	1000	100	200	300	400	500	1000
BQ2	7.7s	31.9s	47.2s	55.9s	67.9s	2.5m	9.4s	34.2s	49.1s	58.6s	1.1m	2.6m	17.1s	44.3s	1.0m	1.2m	1.4m	2.9m
PQ1	86ms	1.0s	1.6s	4.3s	5.4s	11.4s	0.9s	3.2s	5.8s	12.9s	19.7s	50.5s	7.3s	20.9s	41.7s	1.5m	2.4m	6.7m
PQ2	1.3s	4.1s	7.3s	10.2s	13.2s	28.4s	2.9s	29.0s	1.6m	2.9m	4.7m	17.6m	13.1s	4.1m	14.9m	27.7m	44.6m	2.9h
PQ3	1.0s	2.8s	4.0s	5.3s	6.7s	13.8s	2.2s	4.3s	5.8s	7.7s	9.4s	18.5s	9.0s	14.3s	18.5s	22.9s	26.4s	48.s
PQ4	1.3s	3.4s	4.9s	6.7s	8.8s	18.4s	2.2s	4.5s	6.1s	8.0s	10.3s	20.0s	6.4s	9.7s	12.0s	15.1s	17.0s	28.5s
PQ5	0.1s	0.6s	0.9s	1.2s	1.7s	4.4s	0.7s	1.6s	1.7s	2.4s	3.3s	5.9s	2.9s	5.1s	5.9s	7.6s	9.3s	14.7s
TUEBA1	0.5s	1.5s	2.7s	3.4s	4.2s	8.4s	1.7s	3.3s	4.6s	6.0s	7.2s	13.7s	8.0s	13.2s	16.4s	22.8s	29.3s	50.4s
TUEBA2	3.5s	8.8s	13.2s	16.8s	20.9s	43.6s	4.7s	10.4s	15.2s	18.7s	23.2s	47.4s	11.0s	19.1s	24.1s	30.5s	37.0s	1.1m

Table 2: Absolute query run time results

between the information drawn from the annotation documentation and its interpretation with respect to a reference terminology. This conceptual separation guarantees transparency and sustainable maintenance of the mapping between annotations and reference terminology.

The *reference model* formalises the reference terminology, in that it represents an overarching terminological backbone that different annotations originating from different formats and annotation schemes are linked to. It consists of three components: a taxonomy of linguistic categories (modelled as OWL classes, e. g., NOUN, COMMONNOUN), a taxonomy of grammatical features (OWL classes, e. g., ACCUSATIVE), and relations (OWL properties, e. g., HASCASE). An *annotation model* is an ontology that represents one specific annotation scheme. We built, among others, annotation models for the SFB632 annotation format (Dipper et al., 2007b) used in typological research, TIGER/STTS (Schiller et al., 1999; Brants et al., 2003), two tag sets for Russian and five tag sets for English, e. g., Susanne (Sampson, 1995), and PTB (Marcus et al., 1993). The linking between annotation models and the reference model is specified in a separate OWL file that imports the reference model and one annotation model.

5.1. Annotation Model

For every tag set, a separate annotation model is specified which can be seen as a formal interpretation of the annotation scheme and its documentation (figure 1). As an example, consider the Susanne tag `APPGf` for `her` used as an attributive pronoun and its presentation in the corresponding annotation model in the lower part of figure 6. The original tag corresponds to an individual that has the tag (`hasTag`) `APPGf`. Moreover, it is characterised by several properties specifying grammatical features, such as case, gender, person and number. Beyond this relational structure, the annotation model also has a hierarchical structure, which is motivated by the systematics of tag formation and by conceptual considerations. Tags beginning with `APPG` are grouped together under the class `AttributivePossessivePronoun`, and tags beginning with `AP(P)` under `AttributivePronoun`. The class `Tag` serves as top-level class for part-of-speech tags in

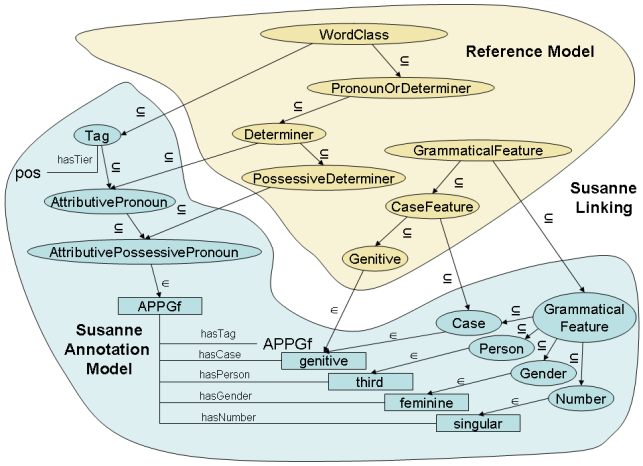

Figure 6: The Susanne tag `APPGf`, its representation within the annotation model and linking with the reference model

the Susanne annotation model, while the top-level class `GrammaticalFeature` dominates a similar hierarchy of classes of grammatical features.

To align tags with specific layers of annotation, the class `Tag` is assigned the property `hasTier` with the value `pos` which is inherited by all POS tags. `APPGf` is, thus, characterised by grammatical features, a specific position in a taxonomy of grammatical categories, and by properties specifying its actual form and the annotation layer.

5.2. Reference Model

The reference model is the terminological backbone of the ontology. It is intended to be a generalisation over the different annotation models, and for the Susanne tag set and `APPGf` as an example, the conceptualisations of the reference model and the annotation model are basically identical (figure 6) – though more complicated constellations occur.

An important difference between the reference model and similar proposals (e. g., GOLD) is that concepts in the reference model may be overlapping, as different conceptualisations are possible. As such, an attributive possessive pronoun can be syntactically regarded as a determiner, but morphologically (and semantically) as a pronoun. Here,

different annotation schemes rely on different conceptualisations – often undocumented – but due to the possible overlap between reference model concepts, the reference model is compatible with either of these possibilities.

5.3. Linking Annotation Model and Reference Model

The reference model and the annotation models are self-contained ontologies. Therefore, the linking between them has to be made explicit. We apply separate OWL files which import the reference model and annotation models. For every annotation model a link file exists that represents the relationship between the annotation model concepts and the reference model concepts by means of `rdfs:Description`s pertaining to `rdfs:subClassOf`-statements. References between annotation model concepts and reference model concepts are possible, thus making instances of annotation model concepts indirect instances of reference model concepts.

In figure 6, the $\subseteq$ edges between reference and annotation model represent the linking. However, it is slightly simplified. As an attributive pronoun serves semantically as a determiner, but syntactically as a pronoun, the annotation model concept `AttributivePronoun` is characterised as a subclass of both `Determiner` and `Pronoun`, etc.

The reference model integrates a limited number of annotation schemes, and, thus, it is specific to the needs of the corresponding projects. Therefore, a potential user may be interested to operate on a terminological resource other than the reference model described above, e. g., GOLD (Farrar and Langendoen, 2003), OntoTag (de Cea et al., 2004), or an OWL version of the DCR (Monachini et al., 2005).

For this purpose, the same linking mechanism may be applied, and these resources may be integrated as an *external reference model*. For the morphosyntactic components of GOLD, OntoTag, and the Data Category Registry, this linking has been implemented. The internal reference model mediates between resource or language-specific annotation models and an external upper model. For the specification of queries, definitions provided by an external reference model may decrease the initial reluctance a user might have to work with the ontology.

5.4. Ontology-Based Corpus Querying

Any tag used in an annotation scheme corresponds to an indirect instance of a class in the reference model. Accordingly, any tag from an annotation model can be retrieved by a description in terms of OWL classes and properties from the reference model. If multiple annotation models are considered, such a description may be expanded into a disjunction of tags from different tag sets.

For this task, the OntoClient was developed, a query preprocessor implemented in Java. OntoClient uses Pellet, an Open Source OWL DL reasoner, to retrieve the set of individuals conforming to a particular description in terms of concepts and properties of the reference model (or an external reference model). From these individuals, the values of the `hasTag` and the `hasTier` properties are determined that represent the actual tag to be queried for and the annotation layer on which it is to be searched. Individuals that lack these properties, e. g., the individual `genitive` in figure 6, merely express grammatical features, and are excluded from the result set.

The function `oc:expand()` applied in figure 4 calls OntoClient's *expand* method with an ontological expression. From the result set, a disjunction of tags (the values of the `hasTag` property) are retrieved and processed. Therefore, the query `[pos/@leveler:text = oc:expand('Verb')]` in figure 4 returns the same results as a query for any single tag for a verbal category as shown in figure 7. The argument of `oc:expand()` is interpreted as a reference model concept, but combinations of object properties and grammatical feature concepts from the reference model (`hasCase(Accusative)`), datatype properties, and string values (`hasTier("pos")`) are also allowed, so are complex expressions formed from these and the set operators $\cap$ (`and`), $\cup$ (`or`) and $\setminus$ (`without`).

```
[pos/@leveler:text = 'VB' or pos/@leveler:text= 'VBZ'
 or ... or pos/@leveler:text= 'VAFIN' or ... or
 pos/@leveler:text='V' or ...]
```

Figure 7: Expanded `oc:expand()` from figure 4

With complex queries, large disjunctions can be avoided by restricting their scope. The outcome of `oc:expand()` can be restricted to POS tags as in (i) below. Also, as in (ii), grammatical features can delimit the scope such as person, number for `will` (1st/3rd.sg.ind.), queried in figure 4.

(i) `oc:expand("Verb and hasTier('pos')")`

(ii) `oc:expand("Verb and (hasPerson(First) or hasPerson(Third)) and hasNumber(Singular)")`

This does not, however, rule out queries across different tag sets. Figure 7 shows result tags originating from tag sets such as PTB and STTS, while the query in figure 4 solely requires information from the STTS tag set and its annotation model. This restriction is handled by metadata for any particular corpus. For corpora with different annotation models, then, separate instantiations of the OntoClient are initialized with exactly those annotation models that are relevant for a particular corpus.

6. The Graphical Interface

We cannot expect our target users (i. e., linguists) to be proficient in XML-related querying languages such as XQuery. Instead, we provide an intuitive user interface that generalises as much as possible from the underlying data structures and querying methods actually used. Our system makes heavy use of Ajax technologies (Asynchronous JavaScript and XML) so that a dynamic, interactive, drag-and-drop-enabled query interface can be provided. The ontology of linguistic annotations (section 5) enables us to provide abstract representations of linguistic concepts (e. g., *noun*, *verb*, *preposition* etc.) that may have a specific set of features; operands can be used to glue together the linguistic concepts by dragging and dropping these graphical representations onto a specific area of the screen, building a query step by step. We also provide several output and visualisation modules for query results, e. g., queried corpus subsets that contain syntactic trees can be visualised as trees, and data that is modelled using a timeline-based approach is displayed in a tabular fashion.

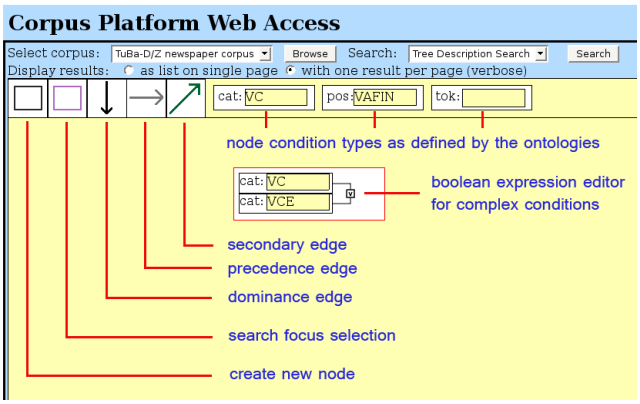

Figure 8: The tree fragment query editor

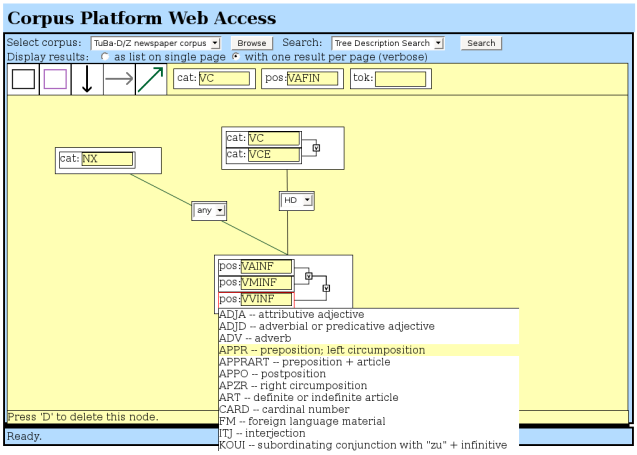

Figure 9: The tree fragment query editor

The graphical front-end is completely implemented in JavaScript extended by the Prototype Ajax Framework (`http://www.prototypejs.org`) and the effects library script.aculo.us (`http://script.aculo.us`). One of its central components is a graphical tree fragment query editor that allows the user to submit complex queries without any knowledge of XQuery. The graphical queries are then interpreted and translated into XQuery. The front-end communicates with the backend via Ajax, posting requests in XQuery to a servlet running on the backend. The servlet responds with the matches encoded in an XML format, which is then interpreted by a variety of display modules.

The tree fragment query editor (see figures 8 and 9) involves dragging and dropping elements on an assembly pane, so that queries can be constructed in a step-by-step fashion. At the moment, structural nodes can be combined by dominance, precedence, and secondary edge relations. The structures defined by these graphs mirror the structures to be found. Each node may contain one or more conditions linked by boolean connectives that help to refine the node classes allowed in the structures. Any structural element in this representation may be given search focus, determining which parts of the structure are returned by the query. This is a helpful function for users who are not interested in seeing entire trees, but who only want to extract lists of, for example, verbs that occur in a specific construction. We plan to realise a set of functions that can be roughly compared to

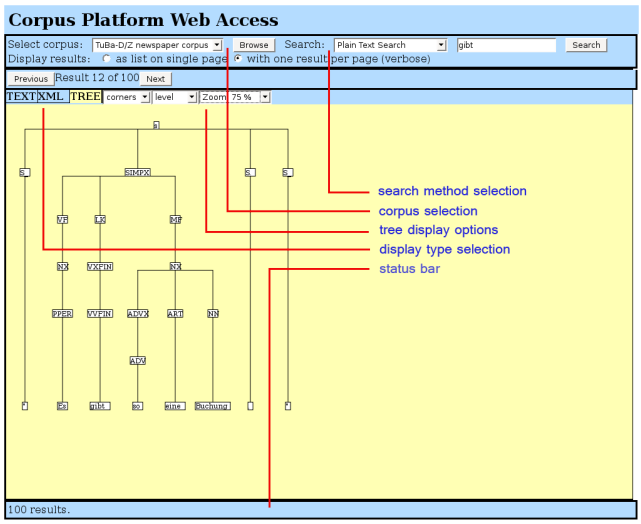

Figure 10: The front-end in tree display mode

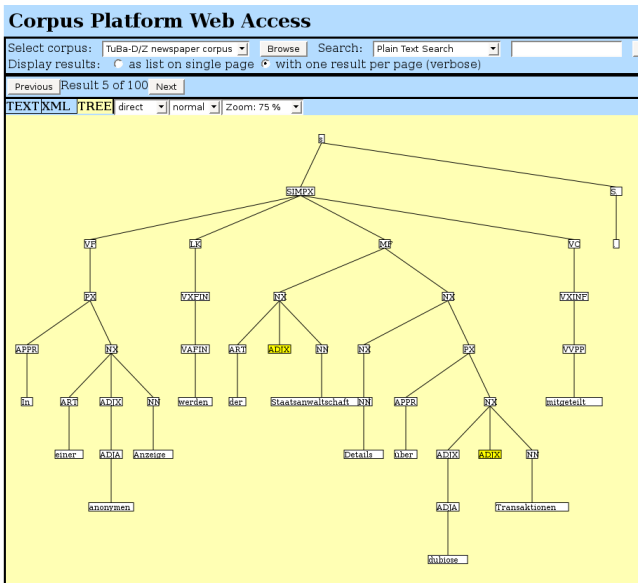

Figure 11: Browsing a corpus (yellow nodes are collapsed)

TIGERSearch's feature set (Lezius, 2002) enhanced by our specific requirements, i. e., multi-layer querying and query expansion through ontologies.

The types of conditions that can be imposed onto structure nodes are determined by the currently selected ontology. For ontology-based querying the graphical front-end needs a list of tag or class names that are formally modelled by the ontology and corresponding descriptions. For all OWL ontologies used in the system we compile these lists that are represented in a simple XML format. When the user chooses an ontology, the front-end retrieves the corresponding file and locally stores the annotation model that determines which condition elements are at the user's disposal.

Tree fragment queries are not the only type of queries allowed by the front-end. It also allows plain text and regular expression queries. Experienced users can formulate their queries in XQuery directly, or they can fine-tune queries initially generated graphically. Query templates for other applications will add to the usability of the platform.

Our aim is to give the user a variety of options to view and to explore results. Four different major display modes are already implemented: plain text view, XML view, graphical tree view and timeline view. Hyperbolic trees and KWIC will be added soon. Display modes are designed to be modular, which means that additional display modules may easily be added if the need arises. The graphical tree display allows collapsing nodes (see figure 10) to prune irrelevant parts of the tree for a better overview, different zoom levels and different display styles. Currently, users can choose whether they want squared or direct edges and whether they want the leaf nodes to be displayed on one level.

The bulk of the development efforts so far has gone into the widgets for tree fragment queries and tree viewing. There were a few major technological obstacles caused by a lack of support for dynamic SVG in most mainstream browsers and the fact that their implementations of the `<canvas>` element are still far from mature (in particular, character treatment can be considered inconsistent – if present at all), but these problems have been solved by now. With the graphical user interface largely in place, development has now turned towards the non-trivial parts of XQuery generation. For performance reasons, as many query components as possible have to be represented using XPath expressions, which can cause problems because secondary edges might result in structures that are less constrained than trees.

7. Conclusions and Future Work

We presented an approach to querying XML-annotated corpora using standard techniques such as XPath and XQuery. As modern corpora are annotated on several layers, we extended the standard functionality of an XML database so that multi-rooted trees, representing one such annotation layer each, can be queried. As our web-based sustainability platform has to cope with arbitrary annotation formats, we built an OWL ontology that encapsulates knowledge about the tag sets used in these multiple and heterogeneous annotation schemes. The ontology can be used for query expansion, so that knowledge of the underlying data formats is not required when querying the corpora using the graphical user interface. This GUI provides an intuitive, modern, flexible, and powerful search interface with several different query and visualisation modes.

As our project is still work in progress, we plan to extend the functionality of the platform in several ways. Currently under construction is a web-interface that enables users to explore extensive sets of metadata associated with the corpora. While these files are also stored in an XML database, we use a relational database to represent user accounts, resources, and access control lists, see (Rehm et al., 2008).

Furthermore, we plan to upgrade and enhance several aspects of the GUI. In addition to a substantial overhaul of the interface in order to improve its usability, we will integrate graphical query templates and saved searches that act like bookmarks in a web browser. For their representation we will use an XML-based format to store all necessary data in one place: the query template itself (i. e., an XQuery fragment), the search result datatype, the annotation layer the query refers to, and a general description of the query.

Acknowledgments Part of the research on AnnoLab was supported by a grant from *Deutsche Forschungsgemeinschaft* (DFG) within the project *Linguistische Profile interdisziplinärer Register* (Technische Universiät Darmstadt). Part of the research presented was supported by a grant from *Deutsche Forschungsgemeinschaft* within the project *Nachhaltigkeit linguistischer Daten*.

8. References

W. Alink, R. Bhoedjang, A. de Vries, and P. Boncz. 2006. Efficient XQuery Support for Stand-Off Annotation. In *Proc. of the Int. Workshop on XQuery Implementation, Experience and Perspectives (XIME-P 2006)*, Chicago, June.

S. Bird and M. Liberman. 2001. A Formal Framework for Linguistic Annotation. *Speech Communication*, 33(1/2):23–60.

S. Bird, Y. Chen, S. B. Davidson, H. Lee, and Y. Zheng. 2006. Designing and Evaluating an XPath Dialect for Linguistic Queries. In *Proc. of the 22nd Int. Conf. on Data Engineering (ICDE 2006)*, pages 52–61.

S. Brants, S. Dipper, P. Eisenberg, S. Hansen-Schirra, E. König, W. Lezius, C. Rohrer, G. Smith, and H. Uszkoreit. 2003. TIGER: Linguistic Interpretation of a German Corpus. *Journal of Language and Computation*.

J. Carletta, J. Kilgour, T. J. O'Donnell, S. Evert, and H. Voormann. 2003. The NITE Object Model Library for Handling Structured Linguistic Annotation on Multimodal Data Sets. In *Proc. of the EACL Workshop on Language Technology and the Semantic Web (3rd Workshop on NLP and XML)*.

C. Chiarcos. 2007. An Ontology of Linguistic Annotation: Word Classes and Morphology. In *Proc. of DIALOG 2007*.

G. A. de Cea, G.-P. Asunción, I. Álvarez de Mon, and A. Pareja-Lora. 2004. OntoTag's Linguistic Ontologies: Improving Semantic Web Annotations for a Better Language Understanding in Machines. In *Proc. of ITCC'04*, pages 124–128.

S. Dipper, E. Hinrichs, T. Schmidt, A. Wagner, and A. Witt. 2006. Sustainability of Linguistic Resources. In *Proc. of the LREC 2006 Satellite Workshop Merging and Layering Linguistic Information*, pages 48–54, Genoa, May.

S. Dipper, M. Götze, U. Küssner, and M. Stede. 2007a. Representing and Querying Standoff XML. In G. Rehm, A. Witt, and L. Lemnitzer, editors, *Data Structures for Linguistic Resources and Applications*, pages 337–346. Narr, Tübingen.

S. Dipper, M. Götze, and S. Skopeteas, editors. 2007b. *Information Structure in Cross-Linguistic Corpora: Annotation Guidelines for Phonology, Morphology, Syntax, Semantics, and Information Structure*, volume 7 of *ISIS*.

R. Eckart and E. Teich. 2007. An XML-Based Data Model for Flexible Representation and Query of Linguistically Interpreted Corpora. In G. Rehm, A. Witt, and L. Lemnitzer, editors, *Data Structures for Linguistic Resources and Applications*. Narr, Tübingen.

S. Farrar and D. T. Langendoen. 2003. A Linguistic Ontology for the Semantic Web. *GLOT International*, 3:97–100.

N. Ide, P. Bonhomme, and L. Romary. 2000. XCES: An XML-based Standard for Linguistic Corpora. In *Proc. of the Second Language Resources and Evaluation Conf. (LREC)*, pages 825–830, Athens.

T. Lehmberg and K. Wörner. 2007. Annotation Standards. In A. Lüdeling and M. Kytö, editors, *Corpus Linguistics*, Handbücher zur Sprach- und Kommunikationswissenschaft (HSK). de Gruyter, Berlin, New York.

W. Lezius. 2002. *Ein Suchwerkzeug für syntaktisch annotierte Textkorpora*. Ph.D. thesis, University of Stuttgart.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. 1993. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

M. Monachini, C. Soria, and M. Ulivieri. 2005. Evaluation of Existing Standards for NLP Lexica. Draft 1.1. Technical report, LIRICS, October, 31st.

G. Rehm, R. Eckart, and C. Chiarcos. 2007. An OWL- and XQuery-Based Mechanism for the Retrieval of Linguistic Patterns from XML-Corpora. In *Int. Conf. Recent Advances in Natural Language Proc. (RANLP 2007)*, pages 510–514, Borovets, September.

G. Rehm, O. Schonefeld, A. Witt, T. Lehmberg, C. Chiarcos, H. Bechara, F. Eishold, K. Evang, M. Leshtanska, A. Savkov, and M. Stark. 2008. The Metadata-Database of a Next Generation Sustainability Web-Platform for Language Resources. In *Proc. of the 6th Language Resources and Evaluation Conf. (LREC 2008)*, Marrakech, May.

G. Sampson. 1995. *English for the Computer. The SUSANNE Corpus and Analytic Scheme*. Clarendon, Oxford.

A. Schiller, S. Teufel, and C. Stockert. 1999. Guidelines für das Tagging deutscher Textcorpora mit STTS. Technical report, Univ. of Stuttgart, Univ. of Tübingen.

T. Schmidt, C. Chiarcos, T. Lehmberg, G. Rehm, A. Witt, and E. Hinrichs. 2006. Avoiding Data Graveyards: From Heterogeneous Data Collected in Multiple Research Projects to Sustainable Linguistic Resources. In *Proc. of the E-MELD 2006 Workshop on Digital Language Documentation: Tools and Standards – The State of the Art*, East Lansing, June.

T. Schmidt. 2005. Time Based Data Models and the TEI's Guidelines for Transcription of Speech. *Working Papers in Multilingualism, Series B*, 62.

C.M. Sperberg-McQueen and L. Burnard, editors. 2002. *TEI P4: Guidelines for Electronic Text Encoding and Interchange*. TEI Consortium.

H. Telljohann, E. Hinrichs, and S. Kübler. 2004. The TüBa-D/Z Treebank – Annotating German with a Context-Free Backbone. In *Proc. of the Fourth Int. Conf. on Language Resources and Evaluation (LREC 2004)*, Lisbon.

A. Witt, O. Schonefeld, G. Rehm, J. Khoo, and K. Evang. 2007. On the Lossless Transformation of Single-File, Multi-Layer Annotations into Multi-Rooted Trees. In B. T. Usdin, editor, *Extreme Markup Languages 2007*, Montréal.

K. Wörner, A. Witt, G. Rehm, and S. Dipper. 2006. Modelling Linguistic Data Structures. In B. T. Usdin, editor, *Extreme Markup Languages 2006*, Montréal.