Reflections and a Proposal for a Query and Reporting Language for Richly Annotated Multiparallel Corpora

Simon Clematide
Institute of Computational Linguistics, University of Zurich
simon.clematide@cl.uzh.ch

Abstract

Large and open multiparallel corpora are a valuable resource for contrastive corpus linguists if the data is annotated and stored in a way that allows precise and flexible ad hoc searches. A linguistic query language should also support computational linguists in automated multilingual data mining. We review a broad range of approaches for linguistic query and reporting languages according to usability criteria such as expressibility, expressiveness, and efficiency. We propose an architecture that tries to strike the right balance to suit practical purposes.

1 Introduction

There is a large amount (millions of sentences) of open multiparallel text data available electronically: resolutions of the General Assembly of the United Nations (Rafalovitch and Dale, 2009), European parliament documents (Koehn, 2005; Hajlaoui et al., 2014), European administration translation memories and law texts (Steinberger et al., 2012; Steinberger et al., 2006), documents from the European Union Bookstore (Skadiņš et al., 2014), and movie subtitles. See Tiedemann (2012) and Steinberger et al. (2014) for an overview.

Automatic part-of-speech tagging and lemmatization of raw text has become standard procedure, and richer linguistic annotations such as morphological analysis, named entity recognition, base chunking, and dependency analysis are possible for many languages. Further, statistical word alignment can be applied to any parallel language resource. If we want to exploit these large, richly annotated resources and flexibly serve the language-related information needs of translators, terminologists and contrastive linguists, an expressive query language for ad hoc search must be provided. Such a query language will also be useful for automated *linguistic data mining*, a use case of computational linguists. A successful combination of these two different paradigms of linguistic information retrieval (i.e. ad hoc search and precomputed word collocation statistics) has been shown in the case of the text corpus query language CQL within the framework of the *Sketch Engine* (Kilgarriff et al., 2014).

Historically, there are two different strains of linguistic query systems, (a) corpus linguistics tools for text corpora such as CQP (Christ, 1994) with KWIC reporting, and (b) treebank tools such as TGrep2 (Rohde, 2005) for searching through deeply nested structures of syntactically annotated sentences. In recent years, we have seen a convergence of these strains: query languages for text corpora have enriched their search operators in order to cope with syntactic constituents, for example introducing the operators `within` and `contain` in CQL (Jakubicek et al., 2010) or the constituent search construct in Poliquarp (Janus and Przepiórkowski, 2007). On the other hand, treebanking-style query approaches that were bound to context-free tree structures have evolved into more general query systems for structural linguistic annotations, e.g. ANNIS (Krause and Zeldes, 2014) which allows a richer set of the structural relations (multi-layered directed acyclic graphs, including syntactic dependencies or coreference chains across sentences), or the Prague Markup Language Tree Query (PML-TQ) system for multi-layered annotations (Štěpánek and Pajas, 2010), which also covers parallel treebanks.[1]

[1] Unfortunately, it is difficult to access up-to-date information about the query possibilities for alignments of words or syntactic nodes. The documentation, however, describes a general cross-layer, node-identifier-based selector dimension. The parallel Prague Czech-English Dependency Treebank 2.0 (PCEDT 2.0) `http://ufal.mff.cuni.cz/pcedt2.0` illustrates the representation of word-aligned dependency trees.

1.1 Linguistic Information Needs

A *linguistic query* in a general sense is a set of interrelated constraints about linguistic structures. The following paragraphs introduce the structures we want to represent and query.

Monolingual constraints on the primary level of word tokens (the minimal unit of analysis) are dealing with inflected word forms, base forms, part-of-speech tags, and morphological categories. Word tokens have a sequential ordering relation (linear precedence). For our case of orthographically well-formed texts, we assume consistent tokenization for all levels of annotation. Giving up this requirement leads to non-trivial ordering problems (Chiarcos et al., 2009). Sentences are sequences of tokens, and documents are sequences of sentences.[2] Documents or sentences typically have metadata associated with them, for instance indicating whether a document is a translation or not.

Each full or partial dependency analysis of a sentence can be represented as a directed and labeled tree graph where each node is a word token, except for the root of the tree, which we assume to be a virtual node. Nested syntactic constituents (or chunks in the case of partial parsing) introduce a dominance relation between syntactic nodes (non-terminals) or primary token nodes (terminals). Dominated nodes also have a linear precedence ordering, the sibling relation.

Cross-lingual constraints are concerned with word alignments and sub-sentential alignments on the chunk level.[3] Directed bilingual word alignments as produced by statistical word alignment tools such as GIZA++ are 1:*n* (Och and Ney, 2003). Bidirectional alignments are thus relational, in general, we have $m : n$ alignments on the level of words, for example, between a German compound and its corresponding multi-word unit in French, unless we apply a symmetrization technique (Tiedemann, 2011, 75ff.).[4]

1.2 Reporting and Visualization

The set of constraints in a query does not exactly determine the content or format of the search results. All flexible linguistic query languages offer means to select the sub-structures and attributes which the user is interested in.[5] This may also include sorting, aggregating or statistical tabulating of the results, as for instance the excellent reporting functions of PML-TQ allow. In our opinion, reporting also includes the user-configurable export of search results, for example as simple comma-separated data for further statistical processing[6], or as hierarchically structured XML serializations.

The graphical visualization of search results aids end users in quickly browsing complex data structures. Visualizations of syntactic structures or frequency distributions of aligned words should be generated on top of specific textual reporting formats. Interactive behavior (collapsing trees, highlighting of aligned nodes) supports a quick interpretation of search results.

The remainder of this paper is structured as follows. Section 2 describes general usability criteria of linguistic query systems. Section 3 discusses interesting linguistic query languages and their main properties. Section 4 introduces general data query languages that are related to linguistic systems. Section 5 discusses evaluation approaches for linguistic query languages. Finally, section 6 presents our proposals for an efficient linguistic query and reporting system for multiparallel data.

2 Usability Criteria for Linguistic Query Systems

Expressibility How naturally can users express their information need? Can users apply their linguistic concepts to formulate their query (Jakubicek et al., 2010, 743), or do they have to deal with cumbersome constructs?

Non-experts may profit from a visual or menu-based composition of queries. Gärtner et al. (2013) and Mírovský (2008) describe graphical query solutions for dependency trees. ANNIS (Zeldes et al., 2009) offers a graphical query interface for AQL. Nygaard and Johannessen (2004) built a menu-based visual query composition for parallel treebanks that used TGrep2 as its query execution engine.

[2] In order to keep the description simple we do not impose more nesting levels in documents.

[3] Sentence alignments are considered as given in the context of multiparallel corpora, although in practical terms it might require a lot of work to achieve a proper and consistent sentence alignment across multiple languages.

[4] Recently, Baisa et al. (2014) applied Dice coefficients to identify aligned lemmas in parallel sentences.

[5] TGrep2 uses backticks to mark the top node of the subtree that is printed as output.

[6] ANNIS provides a practical export format for the WEKA machine learning framework.

Experts, however, will profit most from text-based queries that allows to abstract common and recurrent functionality in the form of user-definable macros, variables, or functions.

Expressiveness Are there inherent limitations in a query language that systematically prevent the formulation of precise search constraints for certain structures? It is well known since its inception that the fragment of existential first-order logic implemented by the TIGERSearch language does not allow for the search of missing constituents in syntactic graphs (König and Lezius, 2003). Lai and Bird (2010) provide a concise overview on the formal expressiveness of query languages for hierarchical linguistic structures and discuss the fact that transitive closures of immediate dominance or precedence relations formally require the expressiveness of monadic second-order logic. Interestingly, such a high expressiveness does not imply inefficient or impractical execution times as shown by Maryns and Kepser (2009) for context-free treebank structures – if tree automata techniques are used. However, purely logical approaches have not received much attention in practice.

Efficiency How much processing time and memory is needed for the execution of a query? Answers to this question relate to many different parameters. First, *data size* of the corpora matters – dealing with thousands, millions, or billions of sentences makes a big difference. Second, *data model complexity* matters. Third, *query expressiveness and complexity* matters.

Even if a user is dealing with large datasets, complex data models and complicated queries, there are solutions to produce acceptable response times. For instance, by providing a highly parallel computing infrastructure using MapReduce techniques (Schneider, 2013), or by using sophisticated indexing and retrieval techniques (Ghodke and Bird, 2012).

Reporting and exporting Does the query language or query system offer flexible support for the user to configure the data reported in the search results? The selection of sub-structures is typically deeply integrated in the query syntax. For text concordancing tools, Frick et al. (2012) mention the `LINK/ALL` operator of COSMAS II, or bracketed expressions in Poliquarp. The statistical reporting functions of the monolingual treebank search tool TIGERSearch[7] rely on named node specifications, and they can only be accessed and configured by graphical user interface interactions. Other query languages such as PML-TQ offer a proper reporting language with a rich set of functions for sorting, aggregating and exporting (e.g. grammar rules).

Visualization Does the query system offer appealing visualizations of the data or data aggregations? ANNIS3 (Krause and Zeldes, 2014) has an outstanding amount of visualization options.

Availability and accessibility Is a system bound to specific operating systems? Large datasets typically overstrain personal desktop computers. Web-based services can be hosted on dedicated computing infrastructure, and there is typically no client-side software installation necessary given the rendering capabilities of modern web browsers (e.g. interactive SVG graphics). Open web-based services enable easy sharing of query results via URLs (Pezik, 2011).

3 Families of Linguistic Query Languages

As mentioned above, there are two strains of linguistic query languages. Some specific properties of these languages are discussed next.

3.1 Text Corpus Query Languages

CQP The language of the IMS Corpus Query Processing Workbench (Hardie, 2012)[8] has a long history (Christ, 1994). From this common ancestor, CQL (Kilgarriff et al., 2004) and Poliquarp were later developed. Right from the beginning, CQP supported annotated word tokens, structural boundaries (sentences, constituents) and sentence-aligned parallel texts. The core of a query consists of regular expressions that specify matching token sequences. These descriptions can refer to the level of word forms, part-of-speech tags or any other positional (=token-bound) attribute. Non-recursive constituents are indirectly available as structural boundaries and can be used to restrict the search space for regular expression matches on the positional level. The constituent segments also allow for attributes which can be queried, for instance syntactic head information. The main

[7] `http://www.ims.uni-stuttgart.de/forschung/ressourcen/werkzeuge/tigersearch.html`

[8] `http://cwb.sourceforge.net`

Relation	QL	Symbol
Immediate dominance	TGrep2, fsq, TS, AQL LPath	> /
Transitive dominance	TGrep2 fsq TS, AQL LPath	>> >+ >* //
Immediate precedence	TGrep2, fsq, TS, AQL LPath	. ->
Transitive precedence	TGrep2, fsq TS, AQL LPath	.. .* -->
Immediate sibling	TS, AQL TGrep2 LPath	$ $. =>

Table 1: Operators of query languages (QL)

weakness of this query language is the lack of a means to query arbitrary relations between tokens, which would be necessary to properly support the search for dependency relations. Given the fact that dependency labels are bound to words, one could map this information as an attribute on the positional level, for example, attributing the property of being a subject to the head of the subject.

An integrated macro and reporting language distinguishes CQP as a powerful and versatile tool.

CQL The query language behind the commercial corpus query platform Sketch Engine[9] is an extension of CQP (Jakubicek et al., 2010).

Support for identifying word matches across parallel corpora is technically implemented via the `within` operator. For a sentence-aligned parallel corpus (English and German Europarl corpus), a query rooted in the English side might look like:

```
[word="car"] within europarl7_de: [word="Auto"]
```

This finds all occurrences of *car* in sentences where a parallel sentence containing the word *Auto* exists. This kind of query, however, does not allow to explicitly test for word alignment relations. Still, the search patterns on both sides of the `within` operator can be arbitrarily complex.

3.2 Treebank Query Languages

TGrep2 The efficient treebank query tool TGrep2 is limited to context-free parse trees. Lai and Bird (2004) see its strength in the ability to query for non-inclusion or non-existence of constituents. Their information need Q2 "Find sentences that do not include the word *saw*" can be expressed succinctly as `S !<< saw`. Their information need Q5 "Find the first common ancestor of sequences of a noun phrase followed by a verb phrase" leads to a short but intricate query (see Tab. 1 for operators):

```
*=p << (NP=n .. (VP=v >> =p !>>
                 (* << =n >> =p)))
```

3.2.1 Path-based Languages

LPath Bird et al. (2006) developed this query language as a generally applicable extension of the XPath query language for XML[10]. Syntactic trees as well as XML documents are ordered trees. However, the direct use of XPath for querying linguistic trees is limited by the absence of (a) the horizontal axis of *x immediately follows/precedes y*, and (b) *sibling x immediately follows/precedes sibling y*.[11] Q2 from above can be stated as

```
/S[not //_[@lex = 'saw']]
```

Q5 cannot be expressed correctly (Lai and Bird, 2004). A further extension of LPath, called LPath+ (Lai and Bird, 2005), is more expressive and allows for a correct but complex query:

```
//_[/_[(NP or (/_[not(=>_)])*/NP[not(=>_)) and
       => (VP or (/_[not(<=_)])*/VP[not(<=_)])]
```

This is due to the fact that path-based, variable-free languages cannot easily express equality restrictions. Therefore, the following shorter LPath expression does not have the correct meaning because each NP (or VP) may refer to different nodes:

```
//_[{//NP->VP} and not(//_{//NP->VP})]
```

DDDQuery This language is another attempt to extend XPath and to better adapt it for linguistic information needs (Faulstich et al., 2006). Its data model was developed for a multi-layered, linguistically richly annotated representation of historical texts, including transcriptions and aligned translations, which resulted in "non-tree-shaped annotation graphs and multiple annotation hierarchies with conflicting structure". This query language "goes beyond LPath by supporting queries on text spans, on multiple annotation layers, and across aligned texts". The language introduces shared variables for any node set in order to easily express equality restrictions and report the matched nodes as result data.

[9] See Kilgarriff et al. (2014) for a recent description. The *NoSketchEngine*, the open-source part of the Sketch Engine, is available from `http://nlp.fi.muni.cz/trac/noske`.

[10] `http://www.w3.org/tr/xpath`

[11] Note that the transitive closures of these relations are available in XPath.

PML-TQ This query language is also a path-based approach (Štěpánek and Pajas, 2010). A query consists of a Boolean combination of node selector paths and filters. The language allows recursive sub-queries in selectors which evaluate to node sets. The cardinality of these node sets can be tested by numeric quantifiers. A quantifier of zero tests for the non-existence of nodes; therefore, non-existing nodes can be queried in a natural way. A similar technique of extensionalization of sub-queries into node sets was implemented for the TreeAligner language (Marek et al., 2008).

3.2.2 Logic-based Languages

fsq[12] The *Finite Structure Query* language (Kepser, 2003) provides full first-order logic as a query language over syntactic structures of the TIGER data model (Brants et al., 2004). This includes labelled secondary edges between arbitrary nodes and discontiguous children. Therefore, fsq has an outstanding expressiveness. Regular expression support for node labels and response times that are comparable to TIGERSearch make this approach a practical one. Lai and Bird's difficult question Q5 can be expressed as follows in the somewhat inconvenient LISP-like prefix notation for first-order logic of fsq [13]:

```
(E a (E n (E v (&
   (cat n NP) (cat v VP) (>+ a v) (.. n v)
   (! (>+ n v)) (! (>+ v n))
   (A b (-> (& (>+ a b) (>+ b n))
            (! (>+ b v))))))))
```

Compared to the query language of TIGERSearch, there is a lack of special purpose predicates such as the (token) arity of syntactic nodes or precedence or dominance restrictions with numeric distance limits, for example, `>2,5` expressing a indirect dominance relation with a minimal depth of 2 and a maximum of 5.

MonaSearch[14] Maryns and Kepser (2009) extended the logical expressiveness of fsq even further to monadic second-order logic. However, its data model is restricted to context-free parse trees. A main application of such an expressive language are automatic consistency checks in human-created treebanks. However, existentially quantified formulas can be used to effectively query matching structures.

TIGERSearch König and Lezius (2000) introduced this logic-based, syntax graph description language for the TIGER data model. It is a subset of first-order logic, providing only globally existentially quantified variables and limited negation. The language has two layers, namely, node constraints and graph constraints.

Node constraints are either node descriptions or node (relation) predicates. *Node descriptions* are Boolean expressions of feature-value constraints with optional variable decorations for referencing the same node several times in a query, for instance `#v:[word != "saw"]` for a terminal node description, or `#np:[cat = ("NP"|"CNP")]` for a simple or coordinated noun phrase. *Node predicates* constrain selected properties of nodes, such as being the root of a tree (`root(#s)`) or having a certain number of daughter nodes (`arity(#CNP,2)`). *Node relation predicates* express the usual structural relations in a user-friendly operator notation, e.g. `#s >* #np` for a dominance relation. *Graph constraints* are conjunctions or disjunctions of node constraints. Negation is not allowed on the level of graph constraints, which severely limits the expressiveness.

The TIGER language originally specified user-defined macros (templates), however, this part of the language was never implemented.

AQL The query language of ANNIS is an extension of the TIGERSearch language for multi-level graph-based annotations. It offers operators for labelled dependency relations, inclusion or overlap of token spans, corpus metadata information, and namespaces for annotations of the same type produced by different tools[15]. The operator for dependency relations is an instance of the general operator `->` for directed and labelled edges between any two nodes. Such edges can also be used to establish or query alignments between parallel sentences on the level of words or phrases.

TreeAligner The *Stockholm TreeAligner* (Lundborg et al., 2007) introduced an operator for querying bilingual alignments between words or phrases of parallel treebanks, freely combinable with monolingual TIGERSearch-style queries. To overcome some expressiveness limitations of

[12] The Java implementation of fsq also includes a TIGERSearch-like visualization for the matched trees, see `http://www.tcl-sfs.uni-tuebingen.de/fsq`.

[13] Existential (`E`) and universal (`A`) quantification, conjunction (`&`), negation (`!`), implication (`->`).

[14] `http://www.tcl-sfs.uni-tuebingen.de/MonaSearch`

[15] For instance, for different parsers (Chiarcos et al., 2010).

TIGERSearch, Marek et al. (2008) introduced node sets (node descriptions decorated with variables starting with % instead of #). One might try to express Bird and Lai's Q2, that is, find sentences without *saw*, in the following ways:

```
#s:[cat="S"] >* #w:[word!="saw"]   (1)
#s:[cat="S"] !>* #w:[word="saw"]   (2)
#s:[cat="S"] !>* %w:[word="saw"]   (3)
```

(1) actually matches all cases where a sentence dominates any other word than *saw*. (2) searches for occurrences of the word *saw* not dominated by a sentence node. The interpretation of (3) relies on a modified evaluation strategy of the negated dominance if one of the arguments is a node set: only those sentences match where the negated transitive dominance constraint `!>*` is true for any of the nodes with the word attribute *saw*.

4 General Data Query Languages

Complex data structures are not a privilege of linguistics, so obviously many *general* data query languages and data management systems exist. Some of them have been used to represent and query linguistic structures.

XPath/XQuery[16] Bouma and Kloosterman (2007) used these XML technologies in a straightforward manner for querying and mining syntactically annotated corpora. These query languages are also the basis of Nite QL (Carletta et al., 2005), which is targeted at multimodal annotations.

SQL The structured query language for relational databases (RDBMS) is a standard technology with highly efficient implementations. RDBMSs have been widely used to represent large amounts of data, e.g. for text concordancing.[17]

CYPHER[18] Distributed NoSQL graph databases and CYPHER as one of the straightforward query languages seem to be a good match for highly interconnected linguistic data (Holzschuher and Peinl, 2013). Pezik (2013) reports some experiments for corpus representation and corpus query with a pure graph database. Banski et al. (2013) integrate a general text retrieval engine with a graph database for their corpus analysis platform.

SPARQL[19] RDF (Resource Description Framework) triple stores with SPARQL endpoints for querying linked data are fairly standard nowadays. Kouylekov and Oepen (2014) used this technique to represent and query semantic dependencies. However, the queries directly operate on the internal RDF representations and do not meet the criteria of natural expressibility. The authors propose a query-by-example and a template expansion front-end for better usability.

Chiarcos (2012) introduce POWLA, a generic formalism to represent multi-layer annotated corpora in RDF and OWL/DL and to query these structures by SPARQL. In order to improve the expressibility, SPARQL macros for AQL operators are defined. Given the expressiveness of SPARQL, this allows to overcome the query language limitations of AQL or TIGERSearch, which cannot query for missing annotations.

LUCENE[20] Every information retrieval system has an integrated query language. Powerful text indexing and query engines such as LUCENE can be used to manage large amounts of texts. By treating each sentence as an IR document, Ghodke and Bird (2012) implemented a high-performance treebank query system[21] on top of LUCENE.

5 Evaluation Strategies for Linguistic Query Languages

There are essentially two approaches to implement the evaluation of linguistic query languages: either by programming a custom implementation of the execution of the query over a custom implementation of the data management, or by translating the query and the data into a host database system and executing the actual query on the host.

Sometimes, these approaches are mixed; for instance, the TreeAligner uses the relational database SQLite for storing and retrieving the primary data of word tokens, but implements a custom in-memory engine for the evaluation of the Boolean algebra of node predicates and node relations.

5.1 Custom Evaluation Engines

Manatee (Rychlý, 2007) is CQL's back-end for textual data management and query evaluation. It

[16] `http://www.w3.org/XML/Query`

[17] `http://corpus.byu.edu` (Davies, 2005)

[18] `http://neo4j.com/developer/cypher-query-language`

[19] `http://www.w3.org/TR/sparql11-query`

[20] `http://lucene.apache.org`

[21] Their query language, however, does not allow regular expressions over labels, or underspecified node descriptions.

is language and annotation independent and includes efficient implementations of inverted indexes, word compression, etc. in order to cope with extremely large text corpora. Attributes of primary data can be set-valued and support unification-style attribute comparisons. Another interesting feature of Manatee is its support for dynamic attributes of positional primary data. These are implemented as function calls which can be declared at the level of the corpus configuration, for instance, for external lexicon look-up, morphological analysis, or the transformation of tags.

TGrep2, TIGERSearch and fsq are examples for treebank query systems with a fully custom data management and query evaluation engine. Rosenfeld (2010) gives a concise description of the implementation techniques behind TIGERSearch. The corpus import of TIGERSearch includes the construction of many specialized indexes for predicates and attributes. During indexing, statistics on the selectivity of attributes are built, which in turn guide the query execution planner to limit the full evaluation of a query to a subset of syntactic trees. At the stage of corpus indexing, users can provide their own type definitions, that is, short names for subsets of admissible feature values. A definition for genitive or dative case looks as follows:

```
    gen-dat := "gen","dat";
```

Although any query involving this case ambiguity can be expressed by a Boolean disjunction, type definitions lead to both more readable and compact queries and also to more efficient processing due to the type-based data model of TIGERSearch.

5.2 Query Translation Approach

LPath and DDDQuery are both Xpath-style languages that owe much to the hierarchical data model of XML. However, storing and efficient retrieval of large XML data sets turns out to be a technical challenge in general (Grust et al., 2004). One common solution for high-performance XML retrieval is based on a mapping of the hierarchical document structure into a flat relational format, which in turn allows the use of highly efficient RDBMSs. Both linguistic query languages – LPath and DDDQuery – are translated into SQL queries because their XML data model is physically stored in an RDBMS. The implementation of DDDQuery (Faulstich et al., 2006) is especially interesting for us because they first translate into a first-order logic intermediate representation from which the actual SQL queries are derived.

The development of the relational data model of ANNIS (relANNIS) and the corresponding translation of the ANNIS query language AQL into SQL queries by Rosenfeld (2010) was inspired by the DDDQuery translation. In the next section, we propose a linguistic query language which is similar to AQL but has a simpler data model. Therefore, we expect that our query translation component can be built using the techniques of AQL query evaluations.

6 A Proposal for Querying Richly Annotated Multiparallel Text Corpora

Our data model presupposes the following components: (a) multiparallel corpora with sentence, word and sub-sentential alignments across languages; (b) monolingual linguistic annotations such as PoS tags (preferably the same universal tagset across languages), base forms and morphological information; (c) syntactic annotations in the form of dependency relations and (partial) constituents, allowing the output of different tools for the same kind of analysis (multi-annotation). Multi-tokenization is not required for our data and would impose unnecessary complexity for the query component. However, metadata on the level of corpora, documents, or sentences is needed.

The proposed query language should allow to flexibly query all aspects of our data model. However, the search space of the query evaluation will be restricted to the context of a monolingual sentence and its corresponding aligned sentences.[22] The concrete query syntax for monolingual search will be based on TIGERSearch. Additionally, we introduce an alignment operator similar to the bilingual one of the TreeAligner. However, in multiparallel queries the alignment operator can be used to constrain alignments between nodes of any pair of languages. From AQL, we reuse the operator for dependency relations, the support for metadata predicates, and explicit namespaces. From CQP, we import the concept of a non-recursive macro language. Such a facility proved to be extremely useful for large scale linguistic mining in the case of Sketch Grammars of the Sketch Engine (Kilgarriff et al., 2004).

[22] Monolingual searches across sentence boundaries as permitted in CQP-style queries will not be possible. However, this search limit does not preclude reporting contextual information from surrounding sentences.

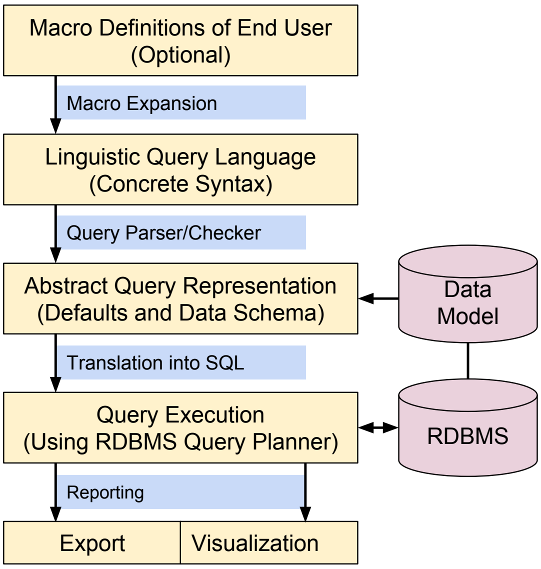
Macro Definitions of End User (Optional)
Macro Expansion
Linguistic Query Language (Concrete Syntax)
Query Parser/Checker
Abstract Query Representation (Defaults and Data Schema)
Data Model
Translation into SQL
Query Execution (Using RDBMS Query Planner)
RDBMS
Reporting
Export
Visualization

Figure 1: Architecture of our proposed system

The predicates needed for expressing the constraints of linguistic queries are different from the reporting functions. After the query execution, reporting functions will be applied to the token IDs, for instance the function *lemma*(#*wordid*) which renders the base form of a terminal node. Flexible reporting expressions similar to PML-TQ have to be defined and implemented. Graphical visualization is just another post-processing step that renders the output of specialized reporting functions.

RDBMSs are stable and efficient data management platforms, and modern, open source implementations such as PostgreSQL[23] support extensions to cope with acyclic graph structures (e.g. recursive SELECT). Therefore, we decided to host our data on an RDBMS and compile our linguistic query language into SQL. The overall architecture of our system is shown in Fig. 1.

One remaining problem is the inability to search for missing elements. The work presented here is part of a contrastive corpus linguistics project which is interested in differences in the use of articles in English and other languages, especially in the case where one language has an article and the other has not. A direct reimplementation of the TreeAligner approach with node set variables seems problematic since the evaluation of a query in the TreeAligner is implemented by iteratively constructing and manipulating node sets in memory. However, the general idea of an extensionalization of intermediate search results is natural.[24] Indeed, SQL itself offers the set operations UNION, INTERSECT, and EXCEPT to combine the results of different queries. In the next section, we present a proposal for searching missing elements using the result set operation EXCEPT.

6.1 Proposal for Query Result Set Operations

If we carefully separate reporting from querying, we can apply result set operations in order to implement the search for missing structures as filtering. We admit that there will be some computing overhead, but conceptually, filtering is easier for end users than full first-order logic.

To illustrate the idea, we informally embed CQP-style macros and TreeAligner constraints into SQL syntax. Bird and Lai's Q2 is easy:

```
SELECT #s FROM corpus WHERE #s:[cat="S"]
EXCEPT
SELECT #s FROM corpus
   WHERE #s:[cat="S"] >* [word="saw"]
```

The information need of Q5 focuses on a triple of an ancestor *a*, an NP *n* and a VP *v*.

```
MACRO a_dom_n_and_v($0=#a,$1=#n,$2=#v)
$0:[] >* $1:[cat="NP"] & $0 >* $2:[cat="VP"] &
$1 .* $2 & $1 !>* $2 & $2 !>* $1 ;

SELECT #a,#v,#n FROM corpus
   WHERE a_dom_n_and_v[#a,#n,#v]
EXCEPT
SELECT #a,#v,#n FROM corpus
   WHERE a_dom_n_and_v[#x,#n,#v] & #a >* #x
```

The first select is too general and includes all ancestors. The second selects the ancestors which dominate such an ancestor. The EXCEPT operator (which calculates the set minus) leaves the very ancestor that does not dominate any other.

A bilingual use case is the search for English noun chunks (nc) *without* article that are aligned to a German chunk *with* an article.[25] The information need are the parallel nouns.

```
MACRO aligned_nc($0=#c,$1=#n,$2=#c2,$3=#n2)
$0:[cat="NC"] > $1:[pos="NOUN"] &
$2:[cat="NC"] > $3:[pos="NOUN"] &
$1 --en,de $3 ;

SELECT #n_en,#n_de FROM corpus
  WHERE aligned_nc[#c_en,#n_en,#c_de,#n_de]
  & #c_de > [pos="DET"]
EXCEPT
SELECT #n_en,#n_de FROM corpus
  WHERE aligned_nc[#c_en,#n_en,#c_de,#n_de]
  & #c_de > [pos="DET"] & #c_en > [pos="DET"]
```

[23] http://www.postgresql.org

[24] Sub-selectors in PML-TQ work in a similar way and their quantifiers are cardinality tests on the matched node sets.

[25] We extend the alignment operator `A -- B` of the TreeAligner with language specifications `A --L1,L2 B`.

In all examples shown above, the resulting nodes of the combined SELECT statements will be fed into the desired reporting functions.

7 Conclusion

We provided a thorough review of linguistic query languages and their implementation approaches and tried to connect them to our use case of richly annotated multiparallel corpora. The usability of linguistic query systems is determined by their expressibility, expressiveness, and efficiency, their support for flexible reporting and exporting – including output for visualization back-ends – and open availability. We decided to host our highly structured data on a RDBMS and to provide a translation from a logic-based query language into SQL.

An important open question for future work is an empirical assessment whether our approach can efficiently deal with the huge amount of annotations and textual material of large multiparallel corpora. Furthermore, a performance comparison with approaches based on RDF triple stores and SPARQL (Chiarcos, 2012) is needed.

Acknowledgments

I wish to thank Johannes Graën for fruitful discussions and Rahel Oppliger for proofreading. This research was supported by the Swiss National Science Foundation under grant 105215 146781/1 through the project "Large-scale Annotation and Alignment of Parallel Corpora for the Investigation of Linguistic Variation".

References

Vít Baisa, Miloš Jakubíček, Adam Kilgarriff, Vojtěch Kovář, and Pavel Rychly. 2014. Bilingual word sketches: the translate button. In *Proc EURALEX*, pages 505–513.

Piotr Banski, J Bingel, N Diewald, E Frick, M Hanl, M Kupietz, P Pezik, C Schnober, and A Witt. 2013. KorAP: the new corpus analysis platform at IDS Mannheim. In *Human Language Technologies as a Challenge for Computer Science and Linguistics. Proceedings of the 6th Language and Technology Conference*.

Steven Bird, Yi Chen, Susan B. Davidson, Haejoong Lee, and Yifeng Zheng. 2006. Designing and evaluating an XPath dialect for linguistic queries. In *Proceedings of the 22nd International Conference on Data Engineering*, pages 52–.

Gosse Bouma and Geert Kloosterman. 2007. Mining syntactically annotated corpora with XQuery. In *Proceedings of the Linguistic Annotation Workshop*, pages 17–24.

Sabine Brants, Stefanie Dipper, Peter Eisenberg, Silvia Hansen-Schirra, Esther König, Wolfgang Lezius, Christian Rohrer, George Smith, and Hans Uszkoreit. 2004. TIGER: Linguistic interpretation of a German corpus. *Research on Language and Computation*, 2(4):597–620.

Jean Carletta, Stefan Evert, Ulrich Heid, and Jonathan Kilgour. 2005. The NITE XML toolkit: Data model and query language. *Language Resources and Evaluation*, 39(4):313–334.

Christian Chiarcos, Julia Ritz, and Manfred Stede. 2009. By all these lovely tokens... merging conflicting tokenizations. In *Proceedings of the Third Linguistic Annotation Workshop*, pages 35–43.

Christian Chiarcos, Kerstin Eckart, and Julia Ritz. 2010. Creating and exploiting a resource of parallel parses. In *Proceedings of the Fourth Linguistic Annotation Workshop*, pages 166–171.

Christian Chiarcos. 2012. A generic formalism to represent linguistic corpora in RDF and OWL/DL. In *Proc LREC 2012*, pages 3205–3212.

Oliver Christ. 1994. A modular and flexible architecture for an integrated corpus query system. In *Proceedings of COMPLEX'95 3rd Conference on Computational Lexicography and Text Research Budapest, Hungary*, pages 23–32.

Mark Davies. 2005. The advantage of using relational databases for large corpora: Speed, advanced queries and unlimited annotation. *International Journal of Corpus Linguistics*, 10(3):307–334.

Lukas Faulstich, Ulf Leser, and Thorsten Vitt. 2006. Implementing a linguistic query language for historic texts. In *Current Trends in Database Technology*, pages 601–612.

Elena Frick, Carsten Schnober, and Piotr Bański. 2012. Evaluating query languages for a corpus processing system. In *Proc LREC 2012*, pages 2286–2294.

Sumukh Ghodke and Steven Bird. 2012. Fangorn: A system for querying very large treebanks. In *Proceedings of COLING 2012: Demonstration Papers*, pages 175–182, December.

Torsten Grust, Maurice Van Keulen, and Jens Teubner. 2004. Accelerating XPath evaluation in any RDBMS. *ACM Trans. Database Syst.*, 29(1):91–131, March.

Markus Gärtner, Gregor Thiele, Wolfgang Seeker, Anders Björkelund, and Jonas Kuhn. 2013. ICARUS – an extensible graphical search tool for dependency treebanks. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60.

Najeh Hajlaoui, David Kolovratnik, Jaakko Väyrynen, Ralf Steinberger, and Daniel Varga. 2014. DCEP - digital corpus of the European Parliament. In *Proc of LREC 2014*, pages 3164–3171.

Andrew Hardie. 2012. CQPweb — combining power, flexibility and usability in a corpus analysis tool. *International Journal of Corpus Linguistics*, 17(3):380–409.

Florian Holzschuher and René Peinl. 2013. Performance of graph query languages: Comparison of Cypher, Gremlin and native access in Neo4J. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 195–204.

Milos Jakubicek, Adam Kilgarriff, Diana McCarthy, and Pavel Rychly. 2010. Fast syntactic searching in very large corpora for many languages. In *Proceedings of the 24th Pacific Asia Conference on Language, Information and Computation*, pages 741–747.

Daniel Janus and Adam Przepiórkowski. 2007. Poliqarp: An open source corpus indexer and search engine with syntactic extensions. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*, pages 85–88.

Stephan Kepser. 2003. Finite structure query: A tool for querying syntactically annotated corpora. In *Proceedings of EACL*, pages 179–186.

Adam Kilgarriff, Pavel Rychlý, Pavel Smrž, and David Tugwell. 2004. The Sketch Engine. In *Proceedings of the Eleventh EURALEX International Congress*, pages 105–116.

Adam Kilgarriff, Vít Baisa, Jan Bušta, Miloš Jakubíček, Vojtěch Kovář, Jan Michelfeit, Pavel Rychly, and Vít Suchomel. 2014. The Sketch Engine: ten years on. *Lexicography*, 1(1):7–36.

Philipp Koehn. 2005. Europarl: A parallel corpus for statistical machine translation. In *Machine Translation Summit*, volume 5, pages 79–86.

Milen Kouylekov and Stephan Oepen. 2014. Semantic technologies for querying linguistic annotations: An experiment focusing on graph-structured data. In *Proc LREC 2014*, pages 4331–4336.

Thomas Krause and Amir Zeldes. 2014. Annis3: A new architecture for generic corpus query and visualization. *Digital Scholarship in the Humanities*.

Esther König and Wolfgang Lezius. 2000. A description language for syntactically annotated corpora. In *COLING 2000, July 31 - August 4, 2000, Universität des Saarlandes, Saarbrücken, Germany*, pages 1056–1060.

Esther König and Wolfgang Lezius. 2003. The TIGER language. A description language for syntax graphs. formal definition. Technical report, Institute for Natural Language Processing, University of Stuttgart.

Catherine Lai and Steven Bird. 2004. Querying and updating treebanks: A critical survey and requirements analysis. In *Proceedings of the Australasian Language Technology Workshop 2004*, pages 139–146.

Catherine Lai and S. G. Bird. 2005. LPath+: A first-order complete language for linguistic tree query. In *Proc PACLIC'19*, pages 1–12.

Catherine Lai and Steven Bird. 2010. Querying linguistic trees. *J. of Logic, Lang. and Inf.*, 19(1):53–73, January.

Joakim Lundborg, Torsten Marek, Maël Mettler, and Martin Volk. 2007. Using the Stockholm TreeAligner. In *Proceedings of the 6th Workshop on Treebanks and Linguistic Theories*, pages 73–78.

Torsten Marek, Joakim Lundborg, and Martin Volk. 2008. Extending the TIGER query language with universal quantification. In *KONVENS 2008: 9. Konferenz zur Verarbeitung natürlicher Sprache*, pages 5–17.

Hendrik Maryns and Stephan Kepser. 2009. Monasearch - a tool for querying linguistic treebanks. In *Treebanks and Linguistic Theories 2009*, pages 29–40.

Jirí Mírovský. 2008. Netgraph - making searching in treebanks easy. In *In Proc. of IJCNLP'08*, pages 945–950.

Lars Nygaard and J.B. Johannessen. 2004. Searchtree - a user-friendly treebank search interface. In *Proc TLT 2004, Tübingen, December 10–11, 2004*, pages 183–189.

Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational linguistics*, 29(1):19–51.

Piotr Pezik. 2011. Providing corpus feedback for translators with the PELCRA search engine for NKJP. In *Explorations across languages and corpora : PALC 2009*, Łódź Studies in Linguistics, pages 135–144, Frankfurt am Main; New York. Peter Lang.

Piotr Pezik. 2013. Indexed graph databases for querying rich TEI annotation. http://digilab2.let.uniroma1.it/teiconf2013/wp-content/uploads/2013/09/Pezik.pdf.

Alexandre Rafalovitch and Robert Dale. 2009. United nations general assembly resolutions: A six-language parallel corpus. In *Proceedings of the MT Summit*, pages 292–299.

Douglas L. T. Rohde. 2005. TGrep2 user manual. http://tedlab.mit.edu/ dr/Tgrep2/tgrep2.pdf.

Viktor Rosenfeld. 2010. *An Implementation of the Annis 2 Query Language*. Student thesis, Humboldt-Universität zu Berlin.

Pavel Rychlý. 2007. Manatee/Bonito – a modular corpus manager. In *Proceedings of Recent Advances in Slavonic Natural Language Processing, RASLAN 2007*, pages 65–70.

Roman Schneider. 2013. KoGra-DB: Using MapReduce for language corpora. In *43. Jahrestagung der Gesellschaft für Informatik (GI)*, pages 140–142.

Raivis Skadiņš, Jörg Tiedemann, Roberts Rozis, and Daiga Deksne. 2014. Billions of parallel words for free: Building and using the EU Bookshop Corpus. In *Proc of LREC 2014*, pages 1850–1855.

Ralf Steinberger, B. Pouliquen, A. Widiger, C. Ignat, T. Erjavec, D. Tufiş, and D. Varga. 2006. The JRC-acquis: A multilingual aligned parallel corpus with 20+ languages. In *Proc of LREC 2006*.

Ralf Steinberger, Andreas Eisele, Szymon Klocek, Spyridon Pilos, and Patrick Schlüter. 2012. DGT-TM: A freely available Translation Memory in 22 languages. In *Proc of LREC 2012*, pages 454–459.

Ralf Steinberger, Mohamed Ebrahim, Alexandros Poulis, Manuel Carrasco-Benitez, Patrick Schlüter, Marek Przybyszewski, and Signe Gilbro. 2014. An overview of the European Union's highly multilingual parallel corpora. *Language Resources and Evaluation*, 48(4):679–707.

Jörg Tiedemann. 2011. *Bitext Alignment*, volume 4 of *Synthesis Lectures on Human Language Technologies*. Morgan & Claypool.

Jörg Tiedemann. 2012. Parallel data, tools and interfaces in OPUS. In *Proc of LREC 2012*, pages 2214–2218.

Amir Zeldes, Anke Lüdeling, Julia Ritz, and Christian Chiarcos. 2009. ANNIS: A search tool for multi-layer annotated corpora. In *Corpus Linguistics 2009*.

Jan Štěpánek and Petr Pajas. 2010. Querying diverse treebanks in a uniform way. In *Proc LREC 2010*, pages 1828–1835.