Converting Latin Treebank Data into an SQL Database for Query Purposes

Christophe Onambélé
CIRCSE Research Centre
Università Cattolica del Sacro Cuore
Largo Gemelli, 1
20123, Milan, Italy
christophe.onambele@unicatt.it

Matyáš Kopp
Charles University in Prague
Institute of Formal and Applied Linguistics
Malostranské náměstí 25
118 00, Prague, Czech Republic
kopp@ufal.mff.cuni.cz

Marco Passarotti
CIRCSE Research Centre
Università Cattolica del Sacro Cuore
Largo Gemelli, 1
20123, Milan, Italy
marco.passarotti@unicatt.it

Jiří Mírovský
Charles University in Prague
Institute of Formal and Applied Linguistics
Malostranské náměstí 25
118 00, Prague, Czech Republic
mirovsky@ufal.mff.cuni.cz

ABSTRACT

This paper describes how to turn a Latin dependency treebank into queryable information so that it can be browsed online using a tree query engine and its web interface. The annotation layers of the treebank are first introduced, then the query system architecture is detailed, and finally the way the treebank is converted into a relational database architecture is described.

CCS CONCEPTS

•**Information systems** → *Database query processing; Query languages;* •**Computing methodologies** → Language resources;

KEYWORDS

Latin, Treebank Data Conversion, Tree Query System, Natural Language Processing

ACM Reference format:
Christophe Onambélé, Matyáš Kopp, Marco Passarotti, and Jiří Mírovský. 2017. Converting Latin Treebank Data into an SQL Database for Query Purposes. In *Proceedings of DATeCH2017, Göttingen, Germany, June 01-02, 2017,* 6 pages.
DOI: http://dx.doi.org/10.1145/3078081.3078087

1 INTRODUCTION

Research works on building dependency treebanks have largely increased over the past decade resulting in the current availability of several such resources. For instance, the last release of Universal Dependencies (UD)[1] consists of 64 dependency treebanks representing 47 languages. Among them there is a number of treebanks for ancient languages, namely the Ancient Greek Dependency Treebank (AGDT [2]) and the subset of Ancient Greek texts in the PROIEL corpus [6], the *Index Thomisticus* Treebank (IT-TB [12], and the Latin Dependency Treebank (LDT [1]). The research areas using such resources range from computational linguistics and NLP to theoretical linguistics and (digital) humanities.

While computational linguists are more prone to use treebanks for NLP purposes, digital classicists and theoretical linguists are interested in browsing data by running queries on them. Making linguistic resources queryable through user-friendly tools is a desideratum, as it is shown by the current availability of a number of query languages and tools specifically built for such purpose. In order to enable researchers to make existing resources accessible for the community in a friendly and easy-to-use interface, this paper describes the main steps to convert a multi-layer dependency treebank (IT-TB) available in a specific format, into a relational database so that it can be browsed online using a generic system for searching annotated corpora. Using a relational database stems from the fact that it allows for faster processing of data than tools relying on XML data.

The paper is organized as follows. Section 2 discusses related work on searching tools for dependency treebanks. Section 3 presents the layers of the treebank. Section 4 provides an overview of the PML-TQ system and software libraries. Section 5 describes the transformation of the IT-TB data into a relational database architecture. Section 6 details and discusses some queries run on different layers of annotation. Finally, Section 7 concludes the paper.

2 RELATED WORK ON SEARCH TOOLS

Several tools exist today that enable researchers to query linguistic resources. These tools differ in complexity of use and expressivity. ICARUS [4] enables users to search dependency treebanks by building queries either graphically or in a text-based mode. INESS [11] provides an open treebanking environment for visualizing and

[1]Version 1.4 released on November 15, 2016. http://universaldependencies.org/.

querying data from (also parallel) treebanks. TüNDRA[2] is a web-based treebank search and visualization application partly based on TIGERSearch [13]. It provides users with the possibility to query dependency treebanks with non-projective dependencies.[3] ANNIS [8] provides the means for searching and visualizing multi-layer annotated linguistic corpora via a web interface. SETS treebank search [9] is a simple query language and tool for searching dependency treebanks.

3 THE *INDEX THOMISTICUS* TREEBANK

The *Index Thomisticus* Treebank (IT-TB) is a dependency annotated corpus which includes texts of Thomas Aquinas (Medieval Latin). Its annotation scheme is based on the Prague Dependency Treebank 2.0 (PDT).[4] At the *morphological layer (m-layer)*, each word of a given sentence is annotated with its lemma (base form of the word) and its part-of-speech tag (keeping the morphological information). Prior to that, the *word layer (w-layer)* simply represents the sequence of words of the input sentence. The *analytical layer (a-layer)* represents the surface syntax of sentences. In analytical trees, every word and punctuation mark of the sentence corresponds to a node of a rooted dependency tree. The edges of the tree represent dependency relations that are labelled with (surface) syntactic functions called "analytical functions" (like Subject, Attribute, etc.). Currently, the a-layer of the IT-TB contains approximately 300,000 a-nodes (around 16,000 sentences). The *tectogrammatical layer (t-layer)*, which represents the underlying structure of the sentence, is conceived as the semantically relevant counterpart of the grammatical means of expression (described by a-trees). Only nodes for autosemantic words occur in tectogrammatical trees; function words and punctuation marks are left out. The t-nodes are labeled with semantic role tags called "functors" indicating the semantic dependency relation between a head and its dependent. Furthermore, t-trees also include ellipsis resolution, coreferential analysis and annotation of information structure (called "topic-focus articulation"). Currently, the the t-layer of the IT-TB features approximately 28,000 nodes (approx. 2,000 sentences).

4 PML-TREE QUERY

PML-TQ [5] is a powerful, generic and open system enabling to search and explore linguistically annotated corpora. The Prague Markup Language[5] (PML [5]) that underpins PML-TQ is an abstract XML-based format applicable to any type of annotation purpose, and in particular to multi-layered treebanks annotations. It follows the stand-off principles [7]. Figure 1 shows the query system architecture and the data flow. Users can access and query data in two ways.

Local Data Querying. This consists of evaluating a PML-TQ query on PML data stored on the user's local hard drive. The evaluation is performed by the `TrEd`[6] extension "PML Tree Query Interface", which provides a graphical interface for querying the data. In this case, the evaluator `PMLTQ::BtredEvaluator`[7] transforms the query to a Perl code and runs it directly on the PML data. The tree structures (representations of the resulting trees) are rendered by TrEd; the appearance of a tree is specified by a stylesheet that is part of the TrEd extension belonging to the queried treebank or can be specified by the user.

Remote Data Querying. The data flow in the client–server and SQL evaluation approach shown in Figure 1 can be divided into three groups. Dotted lines represent both tools and data flow that filter queries by featuring statistical information on a given query. Dashed lines are for tools and data flow helping to select the relevant tree structure matching a specific query. Finally, tools and data flow, represented by solid lines, are coping with both of the aforementioned tasks. There are two user interfaces (the PML-TQ Web and the PML Tree Query Interface for TrEd) which communicate with the PML-TQ Server[8] through a REST API.

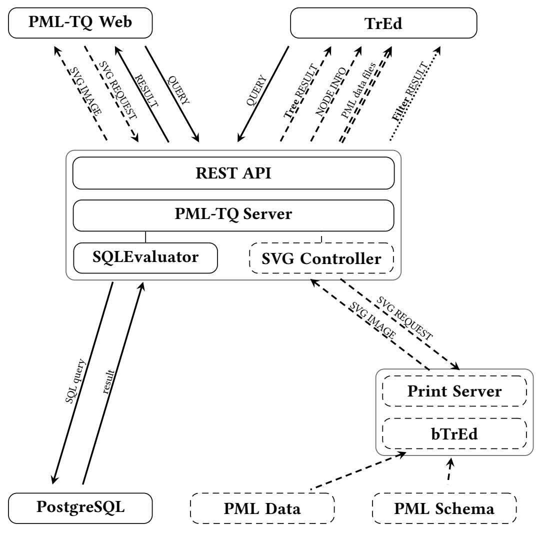


Figure 1: PML-TQ Server Architecture and Data Flow.

Treebank data are stored in two formats: (1) a relational database PostgreSQL used for evaluating the queries, and (2) a PML format used by the Print Server[9] for visualizing the resulting tree structures. The Print Server is a TrEd macro that implements a simple HTTP server using TrEd's tree visualization functionality and provides tree structures of the queries' results in the SVG format.

5 CONVERSION PROCESS

The conversion process is performed through the steps presented below.

The A-layer PML Format. Dependency treebanks in CoNLL-X [3] or CoNLL-U format[10], and Penn like constituency treebanks

[2]Some of its features are available in http://weblicht.sfs.uni-tuebingen.de/weblichtwiki/index.php/Tundra.
[3][10] describes the integration of the IT-TB into TüNDRA.
[4]http://ufal.mff.cuni.cz/pdt2.0/.
[5]http://ufal.mff.cuni.cz/jazz/PML/.
[6]TrEd is one of the client applications to the PML-TQ system, compatible with data in the PML format. Conversion scripts from other formats to PML are provided (e.g., from CoNLL 2009: https://ufal.mff.cuni.cz/conll2009-st/task-description.html).
[7]The `query evaluation` parses each query and determines which nodes satisfy its conditions; then it returns the resulting (tree) nodes or a table of statistics. There are two types of data storage, each with its own query evaluator implemented. The first is the PML data format and its evaluator (`PMLTQ::BtredEvaluator`); the second one is a relational database PostgreSQL and its evaluator (`PMLTQ::SQLEvaluator`) which transforms a PML-TQ query into a SQL query and evaluates it on the database.
[8]The `PMLTQ::Server` is a Perl module written in `Mojolicious` web framework.
[9]https://github.com/ufal/pmltq-print-server.
[10]http://universaldependencies.org/format.html.

can be automatically converted into PML. Here, we describe the PML schema for the a-layer which formalizes two things: (i) the dependency relation between `head` and `dependent`, and (ii) the list of analytical functions. Figure 2 shows an instance of PML in the analytical dependency annotation of the sentence *quod sit officium sapientis* "(What is) the office of the wise man". The annotation file consists of a header, meta data specifying the type of annotation, and the list of trees. In PML, a tree structure is represented by the tree ID stored in the attribute `ID` of the element `LM` under `trees`. The identifier of the sentence is in the element `s.rf` under `trees`. Nodes bear an ID linking to the word form stored in the morphological file (`m.rf`), and the analytical function (`afun`). These nodes also have two technical members: (i) an XML element `ord` referring to the order of the word in sentence (counted from 1, 0 is for root), and (ii) a list of child-nodes represented by the element `children`.

Following this PML schema, one can figure out the dependency structure of the sentence. You start with the root represented by the sentence ID `a-005.SCG*LB1.CP-++1.N.-.1-1.1-4`[11]. This root has two dependents (or `children`): a nested tree representing the sentence, and the punctuation mark labeled `AuxK`. The root of the nested tree is the second word (`W2`) of the sentence whose analytical function is `Pred`, and it heads two dependents: the first word of the sentence (`W1`) which is a predicate nominal (`PNom`), and a nested subtree whose head is the third word of the sentence (`W3`), which is the subject (`Sb`). This node heads that for the fourth word of the sentence (`W4`), which is an attribute (`Atr`). The corresponding dependency tree is shown in figure 3.

From PML Data to SQL Database. The conversion uses information on relevant data types provided by the PML schema (see Figure 4). PML elements are stored in SQL tables, and each record has a unique identifier attribute. The PML format contains roles[12] (`#TREES, #NODES, #CHILDNODES`) enabling users to traverse a tree with various node types (e. g. `a-node`, `a-root`). Mapping a tree into a table (although trees and tables are two different topological objects) is well performed, as a tree structure is basically a special case of relation easily representable in a table without any loss of information.

The use of a relational database[13] is motivated by the fact that it is much faster than XML format (stored in text files). For example, we were not able to open 60 MB IT-TB in one file on PC with 4 GB RAM, because it was not able to parse such a large XML file. Splitting it into multiple files would increase the speed slightly but it would not solve the fact that a huge amount of XML data need to be parsed while executing every single query.

The conversion process is motivated by the fact that there were no better possibilities at the time PML-TQ was implemented. Nowadays we believe that using some document database (such as MongoDB) should have been better[14] since it respects tree structure, or

```
<?xml version="1.0" encoding="UTF-8"?>
<adata xmlns="http://ufal.mff.cuni.cz/pdt/pml/">
  <head>
    <schema href="adata_it_schema.xml" />
    <references>
      <reffile id="m" name="mdata" href="005-SCG.DATI.1.l.m" />
      <reffile id="w" name="wdata" href="005-SCG.DATI.1.l.w" />
    </references>
  </head>
  <meta>
    <annotation_info>
      <desc>Manual annotation</desc>
    </annotation_info>
  </meta>
  <trees>
    <LM id="a-005.SCG*LB1.CP-++1.N.-.1-1.1-4">
      <s.rf>m#m-005.SCG*LB1.CP-++1.N.-.1-1.1-4</s.rf>
      <ord>0</ord>
      <children>
        <LM id="a-005.SCG*LB1.CP-++1.N.-.1-1.1-4W2">
          <m.rf>m#m-005.SCG*LB1.CP-++1.N.-.1-1.1-4W2</m.rf>
          <ord>2</ord>
          <children>
            <LM id="a-005.SCG*LB1.CP-++1.N.-.1-1.1-4W1">
              <m.rf>m#m-005.SCG*LB1.CP-++1.N.-.1-1.1-4W1</m.rf>
              <ord>1</ord>
              <afun>Pnom</afun>
            </LM>
            <LM id="a-005.SCG*LB1.CP-++1.N.-.1-1.1-4W3">
              <m.rf>m#m-005.SCG*LB1.CP-++1.N.-.1-1.1-4W3</m.rf>
              <ord>3</ord>
              <children id="a-005.SCG*LB1.CP-++1.N.-.1-1.1-4W4">
                <m.rf>m#m-005.SCG*LB1.CP-++1.N.-.1-1.1-4W4</m.rf>
                <ord>4</ord>
                <afun>Atr</afun>
              </children>
              <afun>Sb</afun>
            </LM>
          </children>
          <afun>Pred</afun>
        </LM>
        <LM id="a-005.SCG*LB1.CP-++1.N.-.1-1.1-4W5">
          <m.rf>m#m-005.SCG*LB1.CP-++1.N.-.1-1.1-4W5</m.rf>
          <ord>5</ord>
          <afun>AuxK</afun>
        </LM>
      </children>
    </LM>
    ...
   </trees>
</adata>
```

Figure 2: Sample Instance of the Sentence *"quod sit officium sapientis."* encoded in PML Format on Analytical Layer.

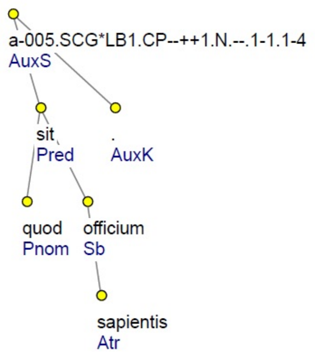
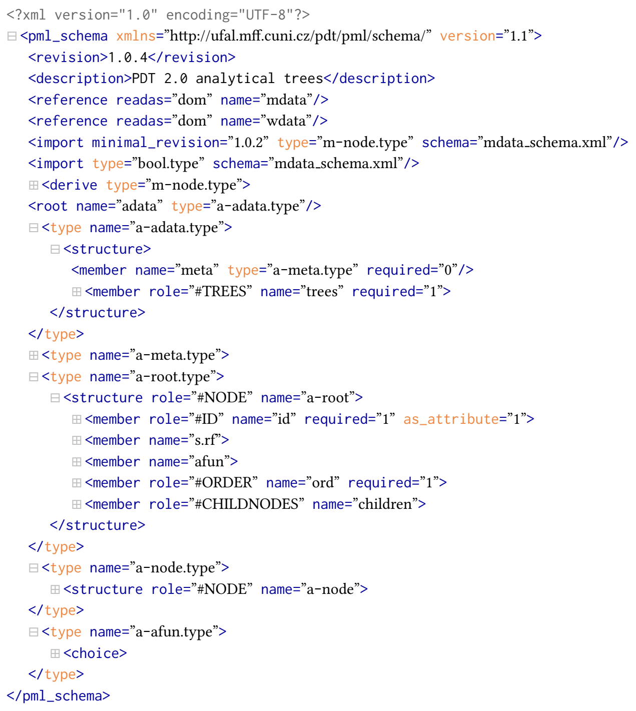


Figure 3: A-tree of the Sentence *"quod sit officium sapientis."*

[11]Textual reference: `a-` (analytical layer), `005` (fifth text registered in the *Index Thomisticus*), `SCG` (*Summa contra Gentiles*), `LB1` (Book 1), `CP-++1` (Chapter 1), `N.-` (no numbered section), `1-1` (sentence starts at line 1, word 1), `1-4` (sentence ends at line 1, word 4). The root node is assigned by default afun `AuxS`.
[12]PML `roles` are pre-defined sets of labels indicating which data refer to the nodes trees, how the nested tree structure are built, data structures carrying unique ID, or links to other layers of annotation.
[13]The database structure is optimized for build-in relations. Every query with a relation needs a `JOIN` in its SQL form. Reducing the number of database `JOIN` increases performance.
[14]better=faster.



```
<?xml version="1.0" encoding="UTF-8"?>
<pml_schema xmlns="http://ufal.mff.cuni.cz/pdt/pml/schema/" version="1.1">
  <revision>1.0.4</revision>
  <description>PDT 2.0 analytical trees</description>
  <reference readas="dom" name="mdata"/>
  <reference readas="dom" name="wdata"/>
  <import minimal_revision="1.0.2" type="m-node.type" schema="mdata_schema.xml"/>
  <import type="bool.type" schema="mdata_schema.xml"/>
  <derive type="m-node.type">
  <root name="adata" type="a-adata.type"/>
  <type name="a-adata.type">
    <structure>
      <member name="meta" type="a-meta.type" required="0"/>
      <member role="#TREES" name="trees" required="1">
    </structure>
  </type>
  <type name="a-meta.type">
  <type name="a-root.type">
    <structure role="#NODE" name="a-root">
      <member role="#ID" name="id" required="1" as_attribute="1">
      <member name="s.rf">
      <member name="afun">
      <member role="#ORDER" name="ord" required="1">
      <member role="#CHILDNODES" name="children">
    </structure>
  </type>
  <type name="a-node.type">
    <structure role="#NODE" name="a-node">
  </type>
  <type name="a-afun.type">
    <choice>
  </type>
</pml_schema>
```

Figure 4: Sample PML Schema for Analytical Trees Representation.

some native XML database (no conversion will be needed). But the most time consuming part of the query are the filters; so if we want to have faster query evaluation, we have to improve the evaluation of aggregation functions. However one can question whether such converted data structures are queryable, since the task may not be quite affordable in a declarative paradigm as the one of SQL. In fact, the PML-TQ language is also a declarative language, so users only need to declare the conditions which the target should satisfy and don't need to care about how it is reached.

6 QUERYING THE DATA

PML-TQ and its extension to TrEd give users the possibility to write and run complex queries on multi-layered annotated resources like ours. For instance, we can compute a "core vocabulary" of the IT-TB. We will start with a simple solution and then proceed to more elaborate ones.

The first and very simple solution is demonstrated by the following query:

```
a-node $a := [ ];
>> for $a.m/lemma give $1, count() sort by $2 desc
```

In its selective part (`a-node $a := [ ]`), the query searches for all nodes on the analytical layer. `a-node` is the type of nodes to search for, the node is named by `$a :=` for later reference (as `$a`), and the properties of the node are defined in squared brackets (`[ ]`), i.e. in this case we have no requirements on the nodes. The part of the query after >> is called an output filter. In this case, it counts a distribution of morphological lemmas of the selected nodes (i.e. all a-nodes in the data). The output filter takes all different lemmas in the results (`for $a.m/lemma`), prints the lemma along with its number of occurrences (`give $1, count()`; `$1` refers to the first expression after `for`, i.e. `$a.m/lemma`) and sorts the resulting list in the descending order by the number of occurrences (`sort by $2 desc`; `$2` refers to the second expression after `give`, i.e. `count()`). A sample of the result is given in Table 1.

Table 1: A sample of the Distribution of Lemmas in the Data

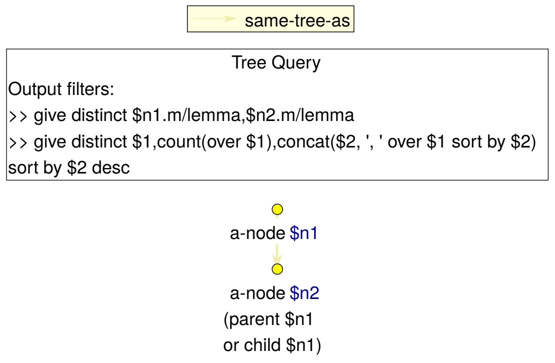
Lemma	Count
sum	9354
qui	3878
in	3805
non	3554
et	3448

The query just lists the most frequent lemmas in the treebank (we excluded punctuation from the resulting table). Instead, we might want to compute the "core syntactic vocabulary" of the treebank, i.e. those words that show the highest number of connections (both as governor and dependent) in the analytical layer of annotation.

The following query demonstrates a solution to the task of computing the "core syntactic vocabulary". The idea is to extract the most connected lemmas in the treebank in terms of dependants and governors:

```
a-node $n1 :=
[ same-tree-as a-node $n2 :=
         [ (parent $n1 or child $n1) ] ];
>> give distinct $n1.m/lemma,$n2.m/lemma
>> give distinct $1,count(over $1),
        concat($2, ', ' over $1 sort by $2) sort by $2 desc
```

The graphical representation of the query is given in Figure 5.



```
Output filters:
>> give distinct $n1.m/lemma,$n2.m/lemma
>> give distinct $1,count(over $1),concat($2, ', ' over $1 sort by $2)
sort by $2 desc
```

Figure 5: A Graphical Query on the Analytical Data.

In its selective part, the query searches for all pairs of analytical nodes that are connected via the parent or child relations, i.e. it searches for all pairs `$n1`, `$n2` of nodes where `$n2` is either the parent or child of `$n1`. For technical reasons given by the syntax of the query language, we first use only a general relation between the two nodes (`same-tree-as`) and then specify it further (`parent $n1 or child $n1`). The first line of the output filter lists all distinct pairs of connected lemmas found by the selective part of the query. The second line of the output filter is applied on the output of the first line and for each lemma prints the lemma (`give distinct $1`), number of different lemmas it was connected to (`count(over $1)`), and a list of all different lemmas it was connected to, sorted alphabetically and separated by commas (`concat($2, ', ' over $1 sort by $2)`); the whole resulting list is sorted in descending order by number of connected lemmas (`sort by $2 desc`). A sample of the result is given in Table 2.

Table 2: A Sample of the Connectiveness of Lemmas in the Data

Lemma	Count	Connections
et	1179	a, abduco, aboleo, ...
sum	1113	a, abeo, abjicio, ...
in	716	abeo, abstractio, ...
possum	505	a, absque, accidens, ...
ad	487	accedo, accidens, ...
...		

For each lemma, Table 2 reports the number of its connections and a small selection of the lemmas it is connected to (again, we excluded punctuation from the lists). If we compare Table 1 with Table 2, we notice that some lemmas are ranked higher in Table 2 than in Table 1. Among them, the coordinating conjunction *et* "and" moves to the first position in Table 2, as the most connected lemma in the treebank. Also, beside the verb *sum* "to be", the verb *possum* "can" occurs in the highest positions of Table 2, while it is not among those ranked highest in Table 1.

In a-trees, prepositions and conjunctions act as bridge nodes linking the effective heads of phrases and clauses standing in governing/subordinate relation. For instance, in the a-tree for the sentence *ex hoc dicitur quod...* (literaly, "from this [it] is said that..."), the node for the preposition *ex* "from" links that for the head verb *dicitur* "is said" with that for the pronoun *hoc* "this". Although in the a-tree *ex* depends on *dicitur* and *hoc* depends on *ex*, the effective dependency relation holds between the head of the governing clause (*dicitur*) and that of the prepositional phrase (*hoc*). Such effective dependencies are not considered by the previous query. In order to face this issue, PML-TQ features two specific relations for searching for effective dependencies. The following query is identical to the previous one with one exception – instead of parent and child relations, it uses relations `eparent` and `echild` (standing for effective parent and effective child):

```
a-node $n1 :=
[ same-tree-as a-node $n2 :=
          [ (eparent $n1 or echild $n1) ] ];
>> give distinct $n1.m/lemma,$n2.m/lemma
>> give distinct $1,count(over $1),
       concat($2, ', ' over $1 sort by $2) sort by $2 desc
```

A sample of the result is given in Table 3. It reports the results for effective relations in the IT-TB. It is worth noticing that only content words occur in the list, while function words are skipped. Only the conjunction *et* still occurs in the highest positions of the list: this reflects the use of *et* as adverb, with the meaning of also. As a matter of fact, the number of connections of *et* in Table 2 is much higher than in Table 3 (1,179 vs 335), because Table 2 includes both the use of *et* as coordinating conjunction and as adverb, while only the latter is considered in Table 3.

Table 3: A Sample of the Connectiveness of Lemmas in the Data Using Effective Relations eparent and echild

Lemma	Count	Connections
sum	1387	abduco, abeo, abjicio, ...
possum	597	absolutus, absum, ...
dico	566	abscessus, absolutus, ...
...		
hic	427	abstraho, accidens, ...
facio	393	absolutus, abstraho, ...
et	335	accidens, actio, actus, ...
...		

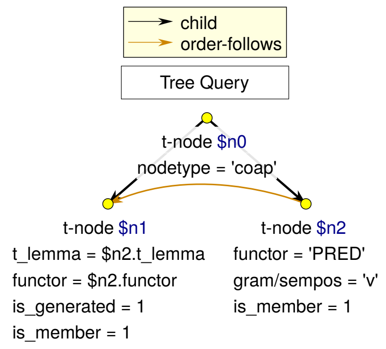


Figure 6: A Graphical Query on the Tectogrammatical Data.

Figure 6 shows the graphical form of a query that searches in tectogrammatical data (t-nodes). The query includes a node `$n0` of type `"coap"` (`nodetype = "coap"`), which is the type assigned to paratactic structure root nodes. `$n0` must have (at least) two direct descendants:

- `$n1`: a `t-node` that (a) is member of the paratactic structure (`is_member = 1`), (b) is newly added in the t-nodes, as it does not correspond to any word in the sentence (`is_generated = 1`) and (c) has both `t_lemma` and `functor` equal to those of node `$n2` (`t_lemma = $n2.t_lemma`; `functor = $n2.functor`).[15]
- `$n2`: a `t-node` that (a) is member of the paratactic structure (`is_member = 1`), (b) is assigned `functor PRED` (main predicate of the sentence) and (c) is a verb (`gram/sempos = "v"`).[16] Furthermore, `$n1` must follow `$n2` in the order of the nodes in t-nodes, as it is represented by the brown arrow directed from `$n2` to `$n1`.[17]

The textual query corresponding to Figure 6 is the following:

[15]T_lemma is the lemma registered at the tectogrammatical layer of annotation. T_lemmas usually correspond to morphological lemmas.
[16]Gram/sempos refers to the grammateme (gram) called "semantic part of speech" (sempos). Grammatemes are attributes capturing meaning of semantically relevant morphological categories such as number and gender for nouns. For instance, pluralia tantum nouns are assigned the singular number grammateme.
[17]Nodes in t-nodes are ordered according to information structure.

```
t-node $n0 :=
[ nodetype = 'coap', t-node $n1 :=
    [ t_lemma = $n2.t_lemma, functor = $n2.functor,
      is_generated = 1, is_member = 1 ],
     t-node $n2 := [ functor = 'PRED', gram/sempos = 'v',
      is_member = 1, order-follows $n1 ] ];
```

Figure 7: A Query on the Tectogrammatical Data.

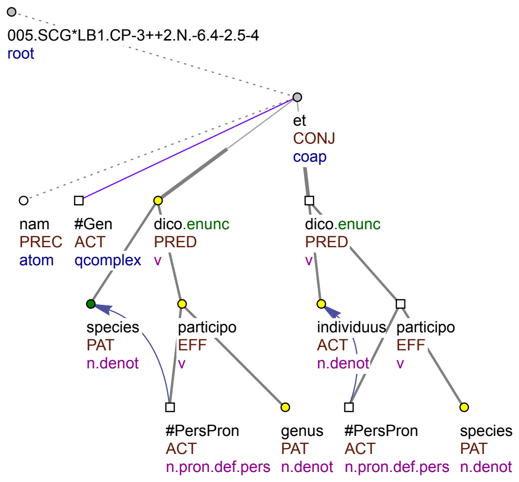


Figure 8: A Tectogrammatical Tree Resulting from a Query.

Figure 8 shows one of the t-trees retrieved by the query (Figure 7) in the IT-TB. The sentence whose t-tree is shown in figure 8 is the following: *nam species participare dicitur genus, et [participare dicitur] individuum speciem* "for the species is said to participate in the genus and the individual [is said to participate] in the species.".[18] Since the second occurrence of the word *dicitur* (`t_lemma:` *dico*) is missing in the sentence, it is replaced in t-trees through ellipsis resolution by adding a new `t-node` with `t_lemma` *dico* (in t-trees, it appears as a white square node). According to the query, this newly added node is a member of a paratactic construction and has the same `t_lemma` and `functor` of another member of the same structure (see the round yellow node for *dico*), which is assigned `functor` `PRED`, it is a verb and it precedes the newly added node in the order of the nodes.

7 CONCLUSION

In this paper, we described our practical experiments in transforming IT-TB data into queryable information. This led us to sketch out a query system designed for dealing with different annotation purposes, and particularly with multi-layer treebanks. We focused on an ancient language in order to cope with an audience (historical linguists, classicists and digital humanists) that needs user-friendly and powerful tools to query annotated corpora and extract empirical evidence from them. In this respect, the graphical interface of PML-TQ is meant to address such requirement. Beside the IT-TB, also part of the Latin Dependency Treebank[19] and the Latin Valency Lexicon Latin-Vallex[20] have been converted into SQL and can thus be browsed online via PML-TQ.

The PML-TQ system is available on the LINDAT/CLARIN[21] repository.

ACKNOWLEDGMENTS

The research reported in the present contribution was supported by the LINDAT/CLARIN project of the Ministry of Education of the Czech Republic (project LM2015071). The *Index Thomisticus* Treebank is partly funded by the Italian Ministry of Education, University and Research (MIUR), FIR-2013 "Developing and Integrating Advanced Language Resources for Latin" (ID: RBFR13EWQN).

REFERENCES

[1] David Bamman and Gregory Crane. 2006. The design and use of a Latin dependency treebank. In *Proceedings of the Fifth Workshop on Treebanks and Linguistic Theories (TLT2006)*. 67–78.
[2] David Bamman, Francesco Mambrini, and Gregory Crane. 2009. An ownership model of annotation: The Ancient Greek dependency treebank. In *Proceedings of the Eighth International Workshop on Treebanks and Linguistic Theories (TLT 8)*. 5–15.
[3] Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*. 149–164.
[4] 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, August 2013, Sofia, Bulgaria*. 55–60.
[5] Jirka Hana and Jan Štěpánek. 2012. Prague markup language framework. In *Proceedings of the Sixth Linguistic Annotation Workshop*. 12–21.
[6] Dag TT Haug and Marius Jøhndal. 2008. Creating a parallel treebank of the old Indo-European Bible translations. In *Proceedings of the Language Technology for Cultural Heritage Data Workshop (LaTeCH 2008), Marrakech, Morocco, 1st June 2008*. 27–34.
[7] Nancy Ide and Laurent Romary. 2004. International standard for a linguistic annotation framework. *Natural language engineering* 10, 3-4 (2004), 211–225.
[8] Thomas Krause and Amir Zeldes. 2016. ANNIS3: A new architecture for generic corpus query and visualization. *Digital Scholarship in the Humanities* 31, 1 (2016), 118–139.
[9] Juhani Luotolahti, Jenna Kanerva, Sampo Pyysalo, and Filip Ginter. 2015. Sets: Scalable and efficient tree search in dependency graphs. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*. 51–55.
[10] Scott Martens and Marco Passarotti. 2014. Thomas Aquinas in the TüNDRA: Integrating the Index Thomisticus Treebank into CLARIN-D.. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation*. 767–774.
[11] Paul Meurer, Helge Dyvik, Victoria Rosén, Koenraad De Smedt, Gunn Inger Lyse, Gyri Smørdal Losnegaard, and Martha Thunes. 2013. The INESS treebanking infrastructure. In *Proceedings of the 19th Nordic Conference of Computational Linguistics (NODALIDA 2013); May 22-24; 2013; Oslo University; Norway. NEALT Proceedings Series 16*. 453–458.
[12] Marco Passarotti. 2011. Language Resources. The State of the Art of Latin and the Index Thomisticus Treebank Project. In *Corpus anciens et Bases de données*. 301–320.
[13] Holger Voormann and Wolfgang Lezius. 2002. TIGERin-Grafische Eingabe von Benutzeranfragen für ein Baumbank-Anfragewerkzeug. In *Proceedings of KONVENS-02), Saarbrücken*.

[18] Textual reference: `005` (fifth text registered in the *Index Thomisticus*), `SCG` (*Summa contra Gentiles*), `LB1` (Book 1), `CP-3++2` (Chapter 32), `N.-6` (Number 6), `4-2` (sentence starts at line 4, word 2), `5-4` (sentence ends at line 5, word 4).
[19] http://itreebank.marginalia.it/ldt/app/form.
[20] http://itreebank.marginalia.it/vallex/app/form.
[21] https://lindat.mff.cuni.cz/en.