Recent Advances in a Feature-rich Framework for Treebank Annotation

Petr Pajas
Charles Univ. in Prague, MFF ÚFAL
Malostranské nám. 25
118 00 Prague 1 – Czech Rep.
pajas@ufal.ms.mff.cuni.cz

Jan Štěpánek
Charles Univ. in Prague, MFF ÚFAL
Malostranské nám. 25
118 00 Prague 1 – Czech Rep.
stepanek@ufal.ms.mff.cuni.cz

Abstract

This paper presents recent advances in an established treebank annotation framework comprising of an abstract XML-based data format, fully customizable editor of tree-based annotations, a toolkit for all kinds of automated data processing with support for cluster computing, and a work-in-progress database-driven search engine with a graphical user interface built into the tree editor.

1 Introduction

Constructing a treebank is a complicated process. Among other things it requires a good choice of tools, varying from elementary data conversion scripts over annotation tools and tools for consistency checking, to tools used for semi-automatic treebank building (POS taggers, syntactic parsers). If no existing tool fits the needs, a new one has to be developed (or some existing tool adapted or extended, which, however, seldom happens in practice). The variety of tools that exist and emerged from various treebanking projects shows that there is no simple solution that would fit all. It is sometimes a small missing feature or an incompatible data format that disqualifies certain otherwise well-established tools in the eyes of those who decide which tools to use for their annotation project.

This paper presents an annotation framework that was from its very beginning designed to be extensible and independent of any particular annotation schema. While reflecting the feedback from several treebanking projects, it evolved into a set of generic tools that is open to all kinds of annotations that involve tree structures. By this paper we would like not only to promote this framework, but also show that due to its open nature, it may be easily extended to fit new requirements. The first three sections describe base components of the framework, an abstract data format, a versatile annotation tool for tree-oriented annotations, and a framework for automatic annotation processing; some of these components have been mentioned in earlier publications, but the framework has neither been published in its integrity nor described in much detail. The last section describes a query engine that is a newest addition to the framework, first presented by this paper.

2 Data format

The base data format selected for the present annotation framework, both for data exchange and as a memory-model reference, is PML (Pajas and Štěpánek, 2006). PML is an abstract XML-based format intended to be generally applicable to all types of annotation purposes, and especially suitable for multi-layered treebank annotations following the stand-of principles. A notable feature that distinguishes PML from other encoding schemes, like Tiger-XML (Mengel and Lezius, 2000), XCES (Ide and Romary, 2003), or maybe even SynAF by ISO TC37/SC4 (Declerck, 2006), is its generic and open nature. Rather than being targeted to one particular annotation schema or being a set of specifically targeted encoding conventions, PML is an open system, where a new type of annotation can be introduced easily by creating a simple XML file called PML schema, which describes the annotation by means of declaring the relevant data types and possibly assigning certain roles to these data types. The roles in the con-

Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008), pages 673–680
Manchester, August 2008

text of PML are just labels from a pre-defined set that can be used to mark the declarations according to their purpose. For instance, the roles indicate which data structures represent the nodes of the trees, how the node data structures are nested to form a tree, which field in a data structure carries its unique ID (if any), or which field carries a link to the annotated data or other layers of annotation, and so on. PML schema can define all kinds of annotations varying from linear annotations through constituency or dependency trees, to complex graph-oriented annotation systems. The PML schema provides information for validating the annotation data as well as for creating a relevant data model for their in-memory representation.

To give an example, the annotation of the Prague Dependency Treebank 2.0 (PDT) (Hajič and others, 2006), which was published in the PML format, consists of four annotation layers, each defined by its own PML schema: a lowest word-form layer consisting of tokenized text segmented just into documents and paragraphs; a morphological layer segmenting the token stream of the previous layer to sentences and attaching morphological form, lemma, and tag to each token; an analytical layer building a morpho-syntactic dependency tree from the words of each sentence (morphologically analyzed on the previous layer); and a tectogrammatical layer consisting of deep-syntactic dependency trees interlinked in a N:M manner with the analytical layer and a valency lexicon and carrying further relational annotation, such as coreference and quotation sets. All these features are formally described by the respective PML schemas.

The fundamental toolkit for PML comprises of a validator (based on compiling PML schemas to RelaxNG grammars accompanied by Schematron rules), and API, consisting of a Perl library (basic interfaces for Java and C++ are planned). The input/output functions of the library are modular and can work with local files as well as with remote resources accessible via HTTP, FTP or SSH protocols (with pluggable support for other protocols). Additionally, the library supports on-the-fly XSLT-based format conversions that can be easily plugged in via a simple configuration file. Consequently, the API can transparently handle even non-PML data formats. Currently there are about a dozen input/output conversion filters available, covering various existing data formats including the TigerXML format, the formats of the Penn Treebank (Marcus et al., 1994), the CoNLL-X shared task format (Buchholz and Marsi, 2006), and the formats of the Latin Dependency (Bamman and Crane, 2006), Sinica (Chu-Ren et al., 2000), Slovene Dependency (Džeroski et al., 2006) (SDT), and Alpino (van der Beek et al., 2002) treebanks. Support for XCES formats is planned as soon as a final release of XCES is available.

This basic toolkit is further supplemented by various auxiliary tools, such as `pmlcopy` which allows one to copy, move, rename, or GZip sets of interconnected PML data files without breaking the internal URL-based references.

3 Tree Editor

The heart of the annotation framework is a multiplatform graphical tree editor called TrEd, (Hajič et al., 2001).

TrEd was from the beginning designed to be annotation-schema independent, extensible and configurable. TrEd can work with any PML data format whose PML schema correctly defines (via roles) at least one sequence of trees. Beside PML format, TrEd can work with many other data formats, either by means of the modular input/output interface of the PML library or using its own input/output backends.

The basic editing capabilities of TrEd allow the user to easily modify the tree structure with drag-and-drop operations and to easily edit the associated data. Although this is sufficient for most annotation tasks, the annotation process can be greatly accelerated by a set of custom extension functions, called macros, written in Perl. Macros are usually created to simplify the most common tasks done by the annotators. They can be called either from menu or by keyboard shortcuts.

Although TrEd ensures that the result of the annotation is in accord with the related PML schema, there is still a chance that an annotator errs in some other aspect of the annotation. For this reason TrEd offers the possibility to write macros that incorporate custom consistency tests into the built-in editing commands of TrEd. Such tests can prevent the user from making accidental mistakes (like assigning a case to a verb or subordinating a Subject to a particle). Macros can also completely disable some dangerous editing commands (for example, the PDT annotation modes in TrEd disable the pos-

sibility to add or delete tokens or trees).

While macros provide means to extend, accelerate and control the annotation capabilities of TrEd, the concept of style-sheets gives users control over the visual presentation of the annotated data. Style-sheets, among other, offer the possibility to: visually differentiate nodes and edges by color, shape, size or line style according to arbitrary criteria; assemble the data associated with nodes and edges to node and edge labels; alter node positioning and padding; visualize additional edges and cross-structure relations by means of arrows or other types of connections; control the content and styling of the text (usually the annotated sentence) displayed in a box above the tree. TrEd can also balance trees, visualize disconnected groups of nodes, zoom the tree view arbitrarily, and display trees in a vertical mode, see Fig. 1.

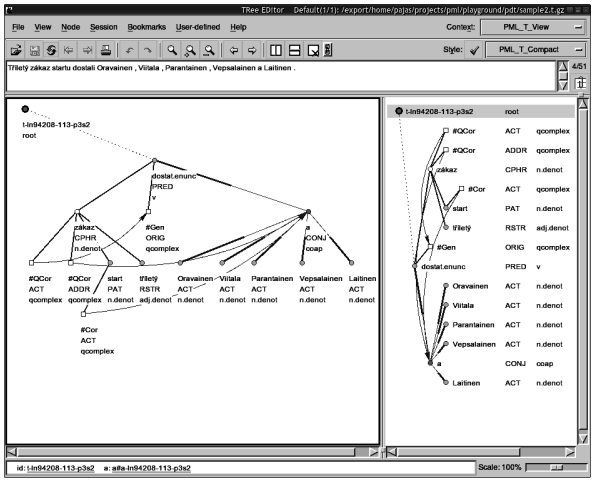

Figure 1: The same tree displayed using horizontal and vertical display mode in TrEd

Due to model/view separation, TrEd provides means for controlling which nodes are actually displayed (it is thus possible to write modes that collapse subtrees, hide auxiliary nodes, completely skip some levels of the tree, display multiple trees at once (Fig. 2), or even display additional "virtual" nodes and edges that are not actually present in the underlying data structures).

So far, TrEd has been selected as annotation tool for PDT and several similarly structured treebanking projects like Slovene (Džeroski et al., 2006), Croatian (Tadić, 2007), or Greek Dependency Treebanks (Prokopidis et al., 2005), but also for Penn-style Alpino Treebank (van der Beek et al., 2002), the semantic annotation in the Dutch

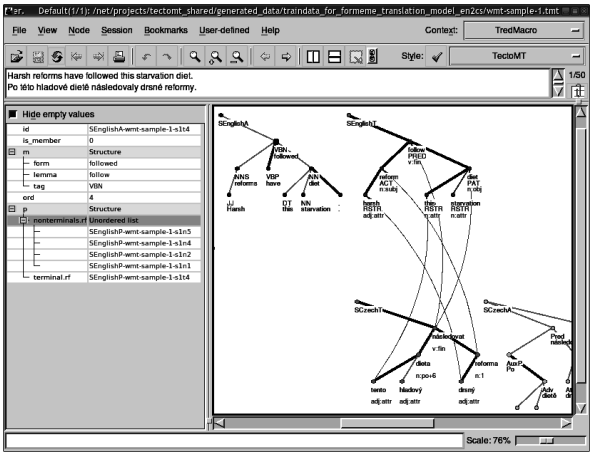

Figure 2: The main window visualizing node-to-node alignment of trees in the TectoMT project (Žabokrtský et al., 2008); side-bar shows data associated with the selected node.

language Corpus Initiative project (Trapman and Monachesi, 2006), as well as for annotation of morphology using so-called MorphoTrees (Smrž and Pajas, 2004) in the Prague Arabic Dependency Treebank (where it was also used for annotation of the dependency trees in the PDT style). While most other projects use off-line conversion

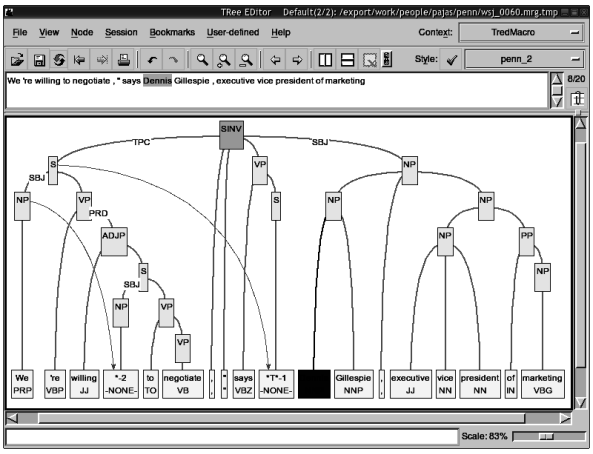

Figure 3: One possible way of displaying Penn Treebank data in TrEd

to some format directly supported by TrEd, for Alpino Treebank a better approach has been selected: the PML library has been configured to convert between Alpino XML format and PML transparently on-the-fly using two simple XSLT stylesheets created for this purpose.

Like some other annotation tools, for example DepAnn (Kakkonen, 2006), TrEd provides means for comparing two (or more) annotations and vi-

sually marking the differences. This functionality is currently provided by macros tailored especially for the PDT annotations. Modifying these macros for specific needs of other tree-based annotations should be easy.

4 Automated processing

The same code-base that runs TrEd (except for the GUI) is used in a command-line tool for automated processing of the annotated data called BTrEd. This tool allows one to search, transform or modify the data by means of small programs written in Perl known from TrEd as macros. Given a list of files, the tool opens the files one after another and applies a given macro on each of them (or, if one chooses, to each tree or each node). With the powerful API of TrEd and the expressiveness of the Perl programming language at hand, one can very easy prepare scripts that gather information, create reports, or automate some parts of the annotation process; in some cases the script can be as short to fit on the command-line.

It is often the case that one has to process a large amount of data repeatedly with one or more scripts. To avoid the need of reading the data into memory upon each execution, BTrEd is augmented by a client-server interface and a client tool called NTrEd. NTrEd, given a list of computer hostnames or IP addresses and a list of files, starts remotely on each of the computers a server-mode instance of BTrEd and distributes the supplied data among these servers (either equally or according to CPU load). Since only file-names are transferred in the communication, some form of shared data storage is assumed. The BTrEd servers read their respective share of the treebank, keep it in their RAM and await client connections. Once the servers are started in this way, one can pass a script to NTrEd as one would do with BTrEd; the tool forwards the script to the servers and collecting the results, outputs them in the same way as a stand-alone BTrEd would. If the script modifies the data, the user can send a request to the servers to either save the changed files or reload them, discarding all modifications. Security of NTrEd is achieved by means of SHA-based authentication (completely transparent to the user) and, optionally, by SSH tunneling.

Since one machine can run multiple instances of BTrEd server, each user of a computer cluster can run its own set of BTrEd servers without interfering with other users. Concurrent work of several users on the same data is supported by TrEd and BTrEd by a simple system of file locks, similar to that of GNU Emacs.

This kind of processing was exploited heavily during the post-annotation checking phases of PDT 2.0 production (Štěpánek, 2006). Employing a cluster consisting from about ten computers, a typical consistency-checking script processed the whole amount of PDT 2.0 (1.5 million analytical nodes and 700 thousand tectogrammatical nodes) in just a few seconds. This was particularly helpful for rapid prototyping or testing hypotheses and it accelerated the whole process enormously.

The NTrEd system, keeping the data in RAM of the servers, is sufficient for small to medium-sized corpora. For huge corpora in scale of terabytes it may not be the case. For processing such huge amounts of data, another tool called JTrEd was recently added to the framework. JTrEd is a wrapper script that simplifies distribution of BTrEd tasks over a computer cluster controlled by the Sun Grid Engine (SGE).

The BTrEd machinery is not intended just for small scripts. A project building a full MT engine on top of this framework is in progress (Žabokrtský et al., 2008).

5 Data-base driven query engine

One of the reasons for which tree-banks are created is that they cover and capture a representative number of syntactic constructions of the particular language. However, to be able to identify them effectively, one has to employ some querying system, consisting of a sufficiently expressive query language and an engine that can search the treebank and present the user with all occurrences matching the constraints of the query.

While for complex queries the tools described in the previous section serve well to users with basic programming skills,'every-day' querying by linguistic public requires a more accessible user-interface. In this section we describe a working prototype of a new query engine and its user interface, based on the data representation and tools described in the preceding sections. First, however, we briefly review some existing solutions.

For searching over PDT a tool called NetGraph (Mírovský, 2006) is traditionally used. This tool's graphical interface allows the users to formulate their queries in a very natural way, namely

as trees whose structures correspond to the structures of the desired search results (although one may specify, for example, that an edge in the query tree should actually match a path in the result tree). Each node in the query tree can carry a set of attributes that match or otherwise constrain the attributes of the corresponding node in the result tree. The query can further put some cardinality constraints on the matching nodes; these are formulated using a special set of labels on the query tree. A great advantage of NetGraph is its web-enabled user interface (Java applet). The underlying query engine is written in C, and although relatively simplistic (i.e. no indexing or planning techniques are used), for PDT-sized corpus it offers reasonable speed for the interactive use. Certain disadvantages of the NetGraph system in our view are: lack of support for querying relations between two or more trees; no support for multi-layered annotations; limited means of expressing attribute constraints and their boolean combinations; restriction to a limited legacy data format.

Probably the best-known query languages for tree structures nowadays are XPath and XQuery, promoted by (and in case of the latter bound to) the XML technology. The advantage of these query languages is that there are several implementations to choose from. Beside searching, some tools (e.g. XSH2 (Pajas, 2005)) provide means for XPath-based data modification. For these reasons, XPath searches over XML-encoded treebank data are promoted (Bouma and Kloosterman, 2002). The disadvantage is, however, that being restricted to the XML data model, users of such tools have to query over a particular XML encoding of the data which often in some way or other obscures the actual annotation schema and relations the annotation represents. Besides, it can be argued that XPath alone does not provide sufficient expressiveness for typical linguistic queries.

As a remedy for the last deficiency, Steven Bird et al. (Bird et al., 2006) proposed a concise query language named LPath, which, while extending core XPath, was designed with the needs of linguistic queries in mind. Their query system is powered by a relational database in which the queries are translated from LPath to SQL. To enable efficient evaluation of constraints on horizontal and vertical relationships between two nodes of a tree by the relational database, the database representation of the trees uses a simple labeling scheme which labels each node with several integers so that the relationship constraints translate in SQL to simple comparisons of the respective integer labels.

It has been shown (Lai and Bird, 2005) that further extension to the LPath language, known as LPath+, is already 1st-order complete. It should, however, be noted that 1st-order completeness has little to do with the practical expressiveness of the language; certain queries, easily expressed in 1st-order logic, only translate to LPath+ at the cost of combinatorial explosion in the size of the query. For example, like XPath, the only way LPath+ offers to match two non-identical sibling nodes is to reach one by the child axis and the other using the following-sibling or preceding-sibling axes from the first one; thus for a query with n sibling nodes whose constraints do not necessarily imply inequality and which can appear in the tree in arbitrary order, the LPath+ query must, in general, enumerate a disjunction of all the $n!$ possible permutations. This may not be a problem when querying over English treebanks, but is a serious handicap for querying over treebanks for languages with free word-order.

There are several other tools for querying over treebanks, we are aware at least of TIGERSearch (Lezius, 2002) for annotations in the TigerXML format, and TGrep2 (Rohde, 2001) for Penn Treebank and similar, which we shall not describe here in detail as they are well known.

For the PML-based annotation system presented in this paper, we have developed a prototype of a new querying system, referred to, just for the purposes of this paper, as PML Tree Query (PML-TQ). The new system attempts to equal the qualities of the above mentioned systems and additionally provide

- a query language with sufficient expressiveness yet without complex formalisms
- unified treatment of structural and non-structural, inter- and cross-layer relationships
- basic reporting capabilities (computing number or distribution of occurrences, etc.)
- a graphical query editor built into TrEd
- a scriptable and extensible interface

At the current stage, PML-TQ provides a prototype query language supporting arbitrary logical

conditions on attributes of nodes and their inter- and cross-layer relations, optional nodes, and basic cardinality constraints. A result of an evaluation of a PML-TQ can be either a set of matches, each match being a set of nodes in the treebank corresponding to the nodes in the query, or a report with some information computed from these node sets. The reporting capabilities of PML-TQ allow one to perform various aggregations on the result node sets and compute statistics over the aggregated groups. Thus, one may easily formulate queries such as *"what is the maximum, minimum, and average depth of a tree in the treebank"*, *"what preposition forms correspond on the surface layer to tectogrammatical nodes with the functor* `DIR3` *and what is their distribution"*, *"what is the most common functor for a child node of a node with functor* `PRED`*"*, *"what is the joint distribution of functors for nodes in the parent-child relation"*, etc.

In the graphical representation of PML-TQ, relations between nodes are represented by arrows. Each PML-TQ query forms a tree or forest whose edges represent basic relations between nodes and possibly nesting of subqueries and whose nodes can be interconnected by additional arrows representing further relations.

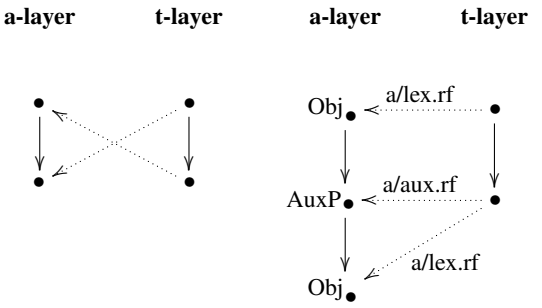

Figure 4: Examples of cross-layer queries in PML-TQ. Query on the left finds dependency that got reversed on the PDT tectogrammatical layer compared to the analytical layer; the query on the right finds tectogrammatical nodes corresponding to an analytical object governing a prepositional object.

Apart from the graphical representation, PML-TQ queries can be formulated in a textual form with syntax resembling XPath (but substantially more expressive). A query is parsed from this syntax into a syntactic tree encoded in PML; in this representation the queries can be stored, visualized, and graphically manipulated in TrEd.

There are presently two engines that can evaluate PML-TQ queries. To utilize the modern RDBMS technology for performance and scalability, we have created a translator of PML-TQ to SQL. One can thus query over a static treebank stored in a database (for encoding tree-structures into database tables, we have adopted a labeling system similar to that described in (Bird et al., 2006)). For querying over data that change (e.g. a file currently open in TrEd or a bunch of files with annotation in progress), we have implemented a simple, yet still relatively fast, evaluator in Perl with a basic planner that can perform PML-TQ searches over PML data sequentially in either TrEd, BTrEd, NTrEd, or JTrEd.

Having two engines behind our query interface in TrEd has several benefits. The annotators will be able to perform identical queries over a corpus stored in a remote SQL database as well as to search in their own locally stored data. The developers of scripts for BTrEd will be able to formulate parts of their scripts briefly as PML-TQ queries whose correctness they will be able to verify independently on a treebank using the SQL backend using an interactive environment.

The SQL-based execution system has currently two database backends for feature and performance comparison: Oracle Express 10g and Postgres SQL. We use Perl DBI modules to interconnect these backends with the TrEd toolkit.

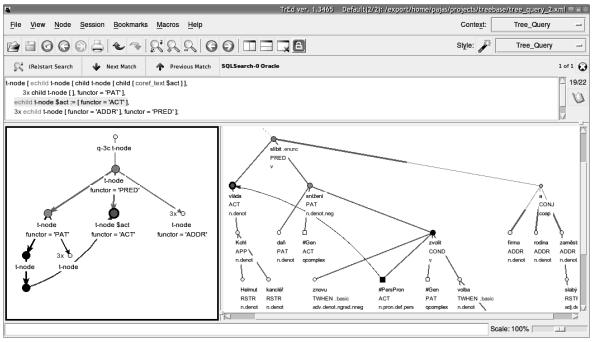

Figure 5: PML-TQ interface in TrEd. Top: the query as text, bottom left the query tree, bottom right a matching tree from PDT 2.0.

The search results can be presented in TrEd in one of several ways. Results of queries that make use of the report-generating facilities of PML-TQ are displayed simply as tables in a text window. Most PML-TQ queries, however, return matching nodes or trees. To display them, TrEd retrieves

corresponding URLs and node positions from the database and reads the actual data from the PML files that must currently be accessible locally or remotely via HTTP, FTP or SSH. Since there are situations when the original data cannot be accessed in this way, we are working on a solution that would allow TrEd to fetch and reconstruct the trees directly from the database.

For populating the database system with treebanks, we have developed a tool that can transfer arbitrary PML-encoded data into a set of database tables suitable for PML-TQ querying. The information about available inter-layer and cross-layer relations is automatically extracted from the PML schema and can be further adjusted by a few command-line parameters.

We have evaluated our database search engine using queries obtained by several means. We had the luck that the developer of NetGraph kindly provided us with all queries collected from real NetGraph users over the past few years in the server logs. We thus obtained almost 9000 queries for the analytical layer and about 5000 queries for the tectogrammatical layer of PDT 2.0. By translating them to PML-TQ with a simple Perl script, we obtained a large collection for testing the basic functionality and performance of our system. To that we added a set of queries that test more advanced PML-TQ features and, for comparison, several queries analogous to the LPath query examples given in (Bird et al., 2006).

When we first run our complete query collection on the Oracle database with 1.5 million nodes and about 88 thousand trees from the analytical layer of PDT, we were surprised to see that out of 8188 queries, 8102 computes in a fraction of second, further 33 in less then 2 seconds, further 36 in less than 10 seconds, 14 in less than 20 seconds and only 5 in more than one minute. Four of these, however, took extremely long time to compute (from hours to days). We observed that all these time-consuming queries were rather similar: they imposed either no or too week constraints on the nodes and sometimes the query tree had a rather large number of auto-morphisms (there was a query consisting of a node with six identical child-nodes none of which carried any constraints). We then found a tree in our data set that contained a node with 85 children. This gives roughly 10^{12} solutions to the query with six siblings on this tree alone.

In some cases the queries can be rewritten using cardinality constraints ("find all nodes with at least 6 children"), which avoids the combinatorial explosion. Since we felt this may not always be possible, we also tried to remove from our data set all trees with more than 20 siblings (44 trees from 70K) that turned out to be mostly TV listings anyway. After that, the performance for the four of the problematic queries improved dramatically: first 100 matches were found in a few seconds and fist 10^6 matches in less than 10 minutes.

Although we have modified the query compiler to suggest cardinality constraints were it seems appropriate and to automatically eliminate some types of automorphisms on the query tree by imposing a strict ordering on the permutable query nodes, we think it is neither possible to completely secure the query system against time-exhaustive queries nor to reliably detect such queries automatically. The querying interface therefore gives the users the option to select a reasonable maximum number of results and allows them to cancel the query evaluation at any time.

6 Conclusion

Over last few years our annotation framework made a considerable leap, from a simple annotation tool to a feature-rich system with several interoperating components. The complete framework is publicly available, either under the General Public License License (GPL), the Perl Artistic License or other GPL-compatible free license. A public release of the tree query interface described in the previous section is scheduled for mid to end of 2008.

7 Acknowledgment

This paper as well as the development of the framework is supported by the grant Information Society of GA AV ČR under contract 1ET101120503.

References

Bamman, David and Gregory Crane. 2006. The design and use of a Latin dependency treebank. In *Proceedings of the Fifth International Workshop on Treebanks and Linguistic Theories (TLT 2006)*, pages 67–78, Prague.

Bird, Steven, Yi Chen, Susan B. Davidson, Haejoong Lee, and Yifeng Zheng. 2006. Designing and evaluating an XPath dialect for linguistic queries. In

ICDE '06: Proceedings of the 22nd International Conference on Data Engineering, page 52, Washington, DC, USA. IEEE Computer Society.

Bouma, Gosse and Geert Kloosterman. 2002. Querying dependency treebanks in XML. In *Proceedings of the Third international conference on Language Resources and Evaluation (LREC)*, Gran Canaria.

Buchholz, Sabine and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings CoNLL-X*.

Chu-Ren, Huang, Keh-Jiann Chen, Feng-Yi Chen, Keh-Jiann Chen, Zhao-Ming Gao, and Kuang-Yu Chen. 2000. Sinica treebank: Design criteria, annotation guidelines, and on-line interface. In *Proceedings of 2nd Chinese Language Processing Workshop (Held in conjunction with ACL-2000)*, pages 29–37, Hong Kong, October 7.

Declerck, Thierry. 2006. Synaf: Towards a standard for syntactic annotation. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006)*, pages 209–232.

Džeroski, Sašo, Tomaž Erjavec, Nina Ledinek, Petr Pajas, Zdeněk Žabokrtský, and Andreja Žele. 2006. Towards a slovene dependency treebank. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006)*, pages 1388–1391.

Hajič, Jan, Barbora Vidová-Hladká, and Petr Pajas. 2001. The Prague Dependency Treebank: Annotation Structure and Support. In *Proceedings of the IRCS Workshop on Linguistic Databases*, pages 105–114, Philadelphia, USA. University of Pennsylvania.

Hajič, Jan et al. 2006. The Prague Dependency Treebank 2.0. CD-ROM. CAT: LDC2006T01.

Ide, Nancy and R. Romary. 2003. Encoding syntactic annotation. In Abillé, A., editor, *Building and Using Parsed Corpora*. Kluwer, Dordrecht.

Kakkonen, Tuomo. 2006. Depann - an annotation tool for dependency treebanks. In *Proceedings of the 11th ESSLLI Student Session at the 18th European Summer School in Logic, Language and Information*, pages 214–225, Malaga, Spain.

Lai, Catherine and Steven Bird. 2005. LPath+: A first-order complete language for linguistic tree query. In *Proceedings of the 19th Pacific Asia Conference on Language (PACLIC)*, Information and Computation, pages 1–12, Taipei, Taiwan. Academia Sinica.

Lezius, Wolfgang. 2002. *Ein Suchwerkzeug für syntaktisch annotierte Textkorpora*. Ph.D. thesis, IMS, University of Stuttgart, December. Arbeitspapiere des Instituts für Maschinelle Sprachverarbeitung (AIMS), volume 8, number 4.

Marcus, Mitchell P., Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The penn treebank: Annotating predicate argument structure. In *HLT*. Morgan Kaufmann.

Mengel, A. and W. Lezius. 2000. An XML-based representation format for syntactically annotated corpora.

Mírovský, Jiří. 2006. Netgraph: A tool for searching in prague dependency treebank 2.0. In Hajič, Jan and Joakim Nivre, editors, *Proceedings of the Fifth Workshop on Treebanks and Linguistic Theories (TLT)*, pages 211–222, Prague, Czech Republic.

Pajas, Petr and Jan Štěpánek. 2006. XML-based representation of multi-layered annotation in the PDT 2.0. In *Proceedings of the LREC Workshop on Merging and Layering Linguistic Information (LREC 2006)*, pages 40–47.

Pajas, Petr. 2005. XSH - XML Editing Shell (an introduction). In *Proceedings of XMLPrague conference on XML 2005*, pages 69–78, Prague.

Prokopidis, P, E Desypri, M Koutsombogera, H Papageorgiou, and S Piperidis. 2005. Theoretical and practical issues in the construction of a greek dependency treebank. In *In Proc. of the 4th Workshop on Treebanks and Linguistic Theories (TLT*, pages 149–160.

Rohde, D. 2001. TGrep2 the next-generation search engine for parse trees. http://tedlab.mit.edu/ dr/-Tgrep2/.

Smrž, Otakar and Petr Pajas. 2004. MorphoTrees of Arabic and Their Annotation in the TrEd Environment. In Nikkhou, Mahtab, editor, *Proceedings of the NEMLAR International Conference on Arabic Language Resources and Tools*, pages 38—41, Cairo. ELDA.

Štěpánek, Jan. 2006. Post-annotation checking of prague dependency treebank 2.0 data. In *Proceedings of the 9th International Conference, TSD 2006*, number 4188 in Lecture Notes In Computer Science, pages 277–284. Springer-Verlag Berlin Heidelberg.

Tadić, Marko. 2007. Building the croatian dependency treebank: the initial stages. In *Contemporary Linguistics*, volume 63, pages 85–92.

Trapman, Jantine and Paola Monachesi. 2006. Manual for the. annotation of semantic roles in D-Coi. Technical report, University of Utrecht.

van der Beek, Leonoor, Gosse Bouma, Robert Malouf, and Gertjan van Noord. 2002. The alpino dependency treebank. In *Computational Linguistics in the Netherlands CLIN 2001*, Rodopi.

Žabokrtský, Zdeněk, Jan Ptáček, and Petr Pajas. 2008. TectoMT: Highly modular hybrid MT system with tectogramatics used as transfer layer. (To appear).