Prague Markup Language Framework

Jirka Hana and **Jan Štěpánek**
Charles University in Prague, Faculty of Mathematics and Physics
lastname@ufal.mff.cuni.cz

Abstract

In this paper we describe the Prague Markup Language (PML), a generic and open XML-based format intended to define format of linguistic resources, mainly annotated corpora. We also provide an overview of existing tools supporting PML, including annotation editors, a corpus query system, software libraries, etc.

1 Introduction

Constructing a linguistic resource 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 NLP 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 annotation projects, it evolved into a set of generic tools that is open to all kinds of annotations.

The first section describes the Prague Markup Language and the way it is used to define format of linguistic resources; follows a section on annotation tools, a query engine and programming libraries. Finally, we discuss related work and future plans.

2 Data Format

The base data format selected for the described annotation framework, both for data exchange and as a memory-model reference, is Prague Markup Language (PML, Pajas and Štěpánek, 2006). While designing PML, we have followed the following set of desiderata:

- Stand-off annotation principles: Each layer of the linguistic annotation should be cleanly separated from the other annotation layers as well as from the original data. This allows for making changes only to a particular layer without affecting the other parts of the annotation and data.
- Cross-referencing and linking: Both links to external document and data resources and links within a document should be represented coherently. Diverse flexible types of external links are required by the stand-off approach. Supposed that most data resources (data, tagsets, and dictionaries) use the same principles, they can be more tightly interconnected.
- Linearity and structure: The data format ought to be able to capture both linear and structure types of annotation.
- Structured attributes: The representation should allow for associating the annotated

units with complex and descriptive data structures, similar to feature-structures.

- Alternatives: The vague nature of language often leads to more than one linguistic interpretation and hence to alternative annotations. This phenomenon occurs on many levels, from atomic values to compound parts of the annotation, and should be treated in a unified manner.

- Human-readability: The data format should be human-readable. This is very useful not only in the first phases of the annotation process, when the tools are not yet mature enough to reflect all evolving aspects of the annotation, but also later, especially for emergency situations when for example an unexpected data corruption occur that breaks the tools and can only be repaired manually. It also helps the programmers while creating and debugging new tools.

- Extensibility: The format should be extensible to allow new data types, link types, and similar properties to be added. The same should apply to all specific annotation formats derived from the general one, so that one could incrementally extend the vocabulary with markup for additional information.

- XML based: XML format is widely used for exchange and storing of information; it offers a wide variety of tools and libraries for many programming languages.

Thus 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-off principles. A notable feature that distinguishes PML from other encoding schemes existing at the time of its creation (see Section 4) 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 (see Figure 1 for an example).

The types used by PML include the following:

Attribute-value structures (AVS's), i.e. structures consisting of attribute-value pairs. For each pair, the name of the attribute and the type of the value is specified. The type can be any PML type, including an AVS. A typical usage example of an AVS structure is, for example a structure gathering the annotation of several independent morphological categories (lemma, case, gender, number). A special type of AVS is a *container*, a structure with just one non-attribute member.

Lists allowing several values of the same type to be aggregated, either in an ordered or unordered manner. For example, a sentence can be represented as an ordered list of tokens, whereas a set of pointers to an ontology lexicon could be captured as an unordered list.

Alternatives used for aggregating alternative annotations, ambiguity, etc. For example, `noun` and `verb` can be alternative values of the part-of-speech attribute in the morphological analysis of the word *flies*: only one of them is the correct value, but we do not know (yet) which one.

Sequences representing sequences of values of different types. Unlike list members, the members of a sequence do not need to be of the same type and they may be further annotated using XML attributes. There is also a basic support for XML-like mixed content (a sequence can contain both text and other elements). A simple regular expression might be used to specify the order and optionality of the members. To give a typical usage example, consider the phrase structure tree: each node has a sequence of child nodes of two data types, terminals and non-terminals. The content of each node would typically be an AVS capturing the phrase type for non-terminals and morphological information for terminals.

Links providing a uniform method for cross-referencing within a PML instance, referencing among various PML instances (e.g. between layers of annotation), and linking to other external resources (lexicons, audio data, etc.).

Enumerated types which are atomic data types whose values are literal strings from a fixed fi-

```
<?xml version="1.0"?>
<pml_schema version="1.1" xmlns="http://ufal.mff.cuni.cz/pdt/pml/schema/">
  <description>Example of constituency tree annotation</description>
  <root name="annotation">
    <sequence role="#TREES" content_pattern="meta, nt+">
      <element name="meta" type="meta.type"/>
      <element name="nt" type="nonterminal.type"/>
    </sequence>
  </root>
  <type name="meta.type">
    <structure>
      <member name="annotator"><cdata format="any"/></member>
      <member name="datetime"><cdata format="any"/></member>
    </structure>
  </type>
  <type name="nonterminal.type">
    <container role="#NODE">
      <attribute name="label" type="label.type"/>
      <sequence role="#CHILDNODES">
        <element name="nt" type="nonterminal.type"/>
        <element name="form" type="terminal.type"/>
      </sequence>
    </container>
  </type>
  <type name="terminal.type">
    <container role="#NODE">
      <cdata format="any"/>
    </container>
  </type>
  <type name="label.type">
    <choice>
      <value>S</value>
      <value>VP</value>
      <value>NP</value>
        <!-- etc. -->
    </choice>
  </type>
</pml_schema>
```

Figure 1: A PML schema defining a simple format for representation of phrase structure trees

```
<?xml version="1.0"?>
<annotation xmlns="http://ufal.mff.cuni.cz/pdt/pml/">
  <head>
    <schema href="example_schema.xml"/>
  </head>
  <meta>
    <annotator>John Smith</annotator>
    <datetime>Sun May 1 18:56:55 2005</datetime>
  </meta>
  <nt label="S">
    <nt label="NP">
      <form>John</form>
    </nt>
    <nt label="VP">
      <form>loves</form>
      <nt label="NP">
        <form>Mary</form>
      </nt>
    </nt>
  </nt>
</annotation>
```

Figure 2: A sample phrase structure encoded in the format defined in Figure 1

nite set. A typical example is a boolean type with only two possible values, 0 and 1.

CData type representing all character-based data without internal structure or whose internal structure is not expressed by means of XML. For improved validation and optimal in-memory representation, the cdata type declaration can be accompanied by a simple format specification (identifier, reference, and the standard W3C XML Schema simple types for numbers, date, time, language, . . .).

A PML schema can also assign roles to particular annotation constructions. The roles are labels from a pre-defined set indicating the purpose of the declarations. 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.

A new PML schema can be derived from an existing one by just mentioning the reference to the old one and listing the differences in special PML elements.

A PML schema can define all kinds of annotations varying from linear annotations of morphology, through constituency or dependency trees, to complex graph-oriented annotation systems (coreference, valency, discourse relations). The schema provides information for validating the annotation data as well as for creating a relevant data model for their in-memory or database representation.

To give a complex example, the annotation of the Prague Dependency Treebank 2.0 (PDT 2.0, Hajič et al., 2006) was published in the PML format. It 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 into sentences and attaching the appropriate morphological form, lemma, and tag to each token;
- an analytical layer building a morpho-syntactic dependency tree from the words of each sentence (morphologically analysed on the previous layer);
- a tectogrammatical layer consisting of deep-syntactic dependency trees interlinked in a $m{:}n$ manner with the analytical layer and a valency lexicon and carrying further relational annotation, such as coreference and quotation sets.

Many other corpora were encoded in the format, including the Prague English Dependency Treebank,[1] the Prague Arabic Dependency Treebank,[2] the Prague Dependency Treebank of Spoken Language,[3] the Prague Czech-English Dependency Treebank,[4] Czesl (an error tagged corpus of Czech as a second language, (Hana et al., 2010)), the Latvian Treebank,[5] a part of the National Corpus of Polish,[6] the Index Thomisticus Treebank,[7] etc.

Moreover, several treebanks were converted into the PML format, mostly to be searchable in the query tool (see Section 3.3); e.g. the Penn Treebank 3, the TIGER Treebank 1.0, the Penn – CU Chinese Treebank 6.0, the Penn Arabic Treebank 2 – version 2.0, the Hyderabad Treebank (ICON 2009 version), the Sinica Treebank 3.0 (both constituency and CoNLL dependency trees), the CoNLL 2009 ST data, etc. The conversion programs are usually distributed as “extensions” (plug-ins) of TrEd (see Section 3.2), but they can be run without the editor as well.

3 Tools

A data format is worthless without tools to process it. PML comes with both low level tools (validation, libraries to load and save data) and higher level tools like annotation editors or querying and conversion tools. Since the last published description of the framework (Pajas and Štěpánek, 2008), the

[1] http://ufal.mff.cuni.cz/pedt2.0/

[2] http://ufal.mff.cuni.cz/padt/PADT_1.0/docs/index.html

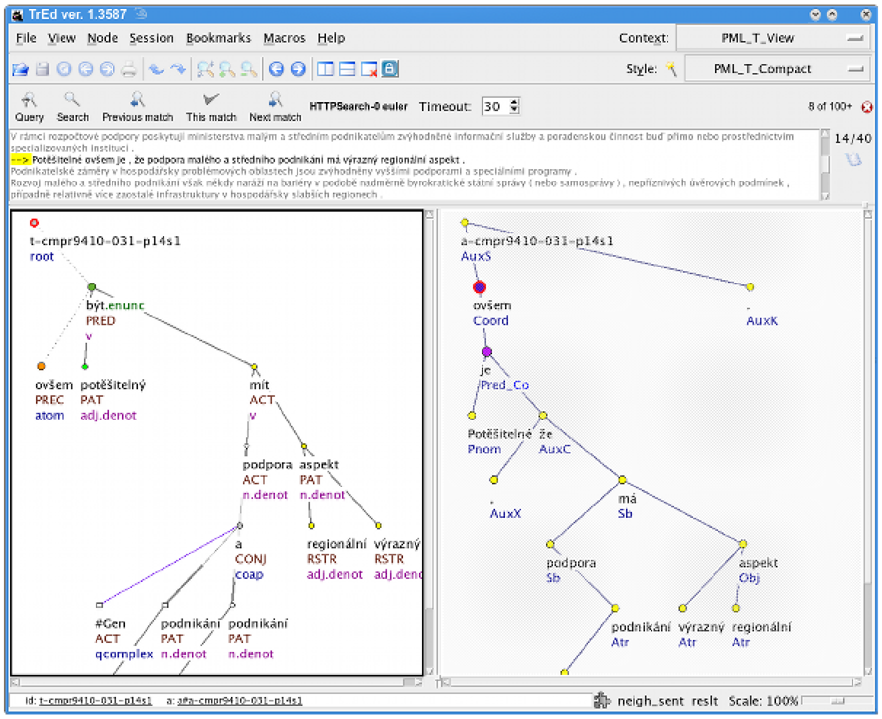
[3] http://ufal.mff.cuni.cz/pdtsl/

[4] http://ufal.mff.cuni.cz/pcedt2.0/

[5] http://dspace.utlib.ee/dspace/bitstream/handle/10062/17359/Pretkalnina_Nespore_etal_74.pdf

[6] http://nkjp.pl/

[7] http://itreebank.marginalia.it/


TrEd ver. 1.3587
File View Node Session Bookmarks Macros Help
Context: PML_T_View
Style: PML_T_Compact
Query Search Previous match This match Next match
HTTPSearch-0 euler Timeout: 30
8 of 100+
14/40
neigh_sent reslt Scale: 100%

V rámci rozpočtové podpory poskytují ministerstva malým a středním podnikatelům zvýhodněné informační služby a poradenskou činnost buď přímo nebo prostřednictvím specializovaných institucí .
--> Potěšitelné ovšem je , že podpora malého a středního podnikání má výrazný regionální aspekt .
Podnikatelské záměry v hospodářsky problémových oblastech jsou zvýhodněny vyššími podporami a speciálními programy .
Rozvoj malého a středního podnikání však někdy naráží na bariéry v podobě nadměrně byrokratické státní správy (nebo samosprávy) , nepříznivých úvěrových podmínek , případně relativně více zaostalé infrastruktury v hospodářsky slabších regionech .

Figure 3: Sample sentence in the TrEd tree annotation tool

tools were further improved and several new ones emerged.

3.1 Low Level Tools

PML documents can be easily validated against their schemas. The validation is implemented by translating the PML schema into a Relax NG schema (plus some Schematron rules) and then validating the documents using existing validation tools for Relax NG. For schemas themselves, there exists another Relax NG schema that can validate them.

Most PML-related tools are written in Perl. The `Treex::PML` package (available at CPAN[8]) provides object-oriented API to PML schemas and documents. The library first loads the schema and then generates API tailored to the instances of the schema.

Applications written in Java can build on a library providing objects supporting basic PML types and utilities for reading and writing them to streams, etc. Moreover, additional libraries provide support for several PML instances (e.g. the PDT corpus and the Czesl corpus (Hana et al., 2010)). While adding support for additional instances is rather straightforward, it must be done manually, as we have not yet implemented an automatic API generator as we did for Perl.

[8]`http://www.cpan.org/`

3.2 Tree Editor TrEd

TrEd, a graphical tree editor, is probably the most frequently used tool from the PML framework. It is a highly extensible and configurable multi-platform program (running on MS Windows, Max OS and Linux). TrEd can work with any PML data[9] whose PML schema correctly defines (via roles) at least one sequence of trees. Besides the 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

[9]TrEd can open data in other formats, too, because it is able to convert the data to PML and back on the fly, the conversion can be implemented as an XSLT transformation, Perl code or executable program.

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. For example, by pressing "(", the annotator toggles the attribute `is_parenthesis` of the whole subtree of the current node.

While macros provide means to extend, accelerate and control the annotation capabilities of TrEd, the concept of *style-sheets* gives users full control over the visual presentation of the annotated data.

So far, TrEd has been used as an annotation tool for PDT 2.0 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 language Corpus Initiative project (Trapman and Monachesi, 2006), the annotation of French sentences with PropBank information (van der Plas et al., 2010), 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 2.0 style).

TrEd is also one of the client applications to the querying system, see Section 3.3.

The editor can also be used without the GUI just to run macros over given files. This mode supports several types of parallelization (e.g. Sun Grid Engine) to speed up processing of larger treebanks. This inspired the Treex project (Popel and Žabokrtský, 2010), a modular NLP software system implemented in Perl under Linux. It is primarily aimed at machine translation, making use of the ideas and technology created during the Prague Dependency Treebank project. It also significantly facilitates and accelerates development of software solutions of many other NLP tasks, especially due to re-usability of the numerous integrated processing modules (called blocks), which are equipped with uniform object-oriented interfaces.

3.3 Tree Query

Data in the PML format can be queried in a query tool called PML-Tree Query (PML-TQ, Pajas and Štěpánek, 2009). The system consists of three main components:

- an expressive *query language* supporting cross-layer queries, arbitrary boolean combinations of statements, able to query complex data structures. It also includes a sub-language for generating listings and non-trivial statistical reports, which goes far beyond statistical features of e.g. TigerSearch.
- client interfaces: a graphical user interface with a graphical query builder, a customizable visualization of the results, web-client interface, and a command-line interface.
- two interchangeable engines that evaluate queries: a very efficient engine that requires the treebank to be converted into a relational database, and a somewhat slower engine which operates directly on treebank files and is useful especially for data in the process of annotation.

The PML-TQ language offers the following distinctive features:

- selecting all occurrences of one or more nodes from the treebanks with given properties and in given relations with respect to the tree topology, cross-referencing, surface ordering, etc.
- support for bounded or unbounded iteration (i.e. transitive closure) of relations[10]
- support for multi-layered or aligned treebanks with structured attribute values
- quantified or negated subqueries (as in "find all clauses with exactly three objects but no subject")
- referencing among nodes (find parent and child that have the same case and gender but different number)
- natural textual and graphical representation of the query (the structure of the query usually corresponds to the structure of the matched subtree)

[10]For example, `descendant{1,3}` (iterating parent relation) or `coref_gram.rf{1,}` (iterating coreference pointer in PDT), `sibling{-1,1}` (immediately preceding or following sibling), `order-precedes{-1,1}` (immediately preceding or following node in the ordering of the sentence)

- sublanguage for post-processing and generating reports (extracting values from the matched nodes and applying one or more layers of filtering, grouping, aggregating, and sorting)

- support for regular expressions, basic arithmetic and string operations in the query and post-processing

For example, to get a frequency table of functions in the Penn Treebank, one can use the following query:

```
nonterminal $n := []
>> for $n.functions
       give $1, count()
       sort by $2 desc
```

Which means: select all non-terminals. Take their functions and count the number of occurrences of each of them, sort them by this number. The output starts like the following:

```
        738953
SBJ     116577
TMP     27189
LOC     19919
PRD     19793
CLR     18345
  ...
```

To extract a grammar behind the tree annotation of the Penn Treebank is a bit more complex task:

```
nonterminal $p := [ * $ch := [ ] ]
>> give $p, $p.cat,
   first_defined($ch.cat,$ch.pos),
   lbrothers($ch)
>> give $2 & " -> "
   & concat($3," " over $1 sort by $4)
>> for $1 give count(),$1
   sort by $1 desc
```

Which means: search for all non-terminals with a child of any type. Return the identifier of the parent, its category, the category or part-of-speech of the child, and the number of the child's left brothers. From this list, return the second column (the parent's category), add an arrow, and concatenate the third column (child's category or part-of-speech) of all the children with the same parent (first column) sorted according to the original word order. In this list, output number of occurrences of each line plus the line itself, sorted by the number of occurrences.

Running it on the Penn Treebank produces the following output:

```
189856    PP -> IN NP
128140     S -> NP VP
 87402    NP -> NP PP
```

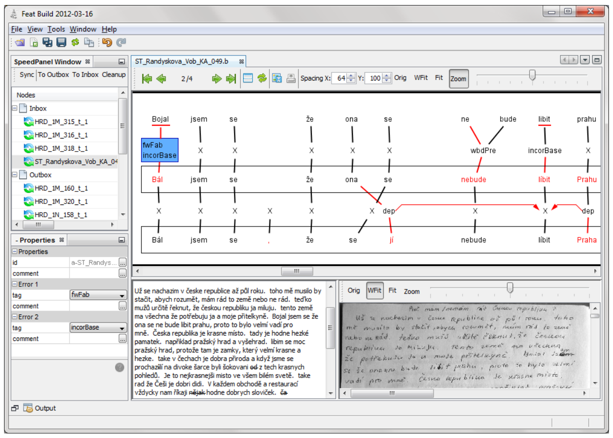


Figure 4: Sample sentence in the *feat* annotation tool

```
72106     NP -> DT NN
65508      S -> NP VP .
45995     NP -> -NONE-
36078     NP -> DT JJ NN
31916     VP -> TO VP
28796     NP -> NNP NNP
23272   SBAR -> IN S
 ...
```

More elaborate examples can be found in (Pajas and Štěpánek, 2009; Štěpánek and Pajas, 2010).

3.4 Other Tools

In addition to the versatile TrEd tree editor, there are several tools intended for annotation of non-tree structures or for specific purposes:

MEd is an annotation tool in which linearly-structured annotations of text or audio data can be created and edited. The tool supports multiple stacked layers of annotations that can be interconnected by links. MEd can also be used for other purposes, such as word-to-word alignment of parallel corpora.

Law (Lexical Annotation Workbench) is an editor for morphological annotation. It supports simple morphological annotation (assigning a lemma and tag to a word), integration and comparison of different annotations of the same text, searching for particular word, tag etc. It natively supports PML but can import from and export to several additional formats.

[11]http://ufal.mff.cuni.cz/~hana/feat.html

Feat[11] is an environment for layered error annotation of learners corpora (see Figure 4). It has been used in the Czesl project (e.g. Hana et al., 2010; Hana et al., 2012) to correct and annotate texts produced by non-native speakers of Czech. The corpus and its annotation is encoded in several interconnected layers: scan of the original document, its transcription, tokenized text encoding author's corrections and two layers of error correction and annotation. Tokens on the latter three layers are connected by hyper-edges.

Capek[12] (e.g. Hana and Hladká, 2012) is an annotation editor tailored to school children to involve them in text annotation. Using this editor, they practice morphology and dependency-based syntax in the same way as they normally do at (Czech) schools, without any special training.

The last three of the above tools are written Java on top of the Netbeans platform, they are open and can be extended via plugins. Moreover, the Capek editor also has an iOS version for iPad.

4 Related Work

TEI The Text Encoding Initiative (TEI) provides guidelines[13] for representing a variety of literary and linguistic texts. The XML-based format is very rich and among other provides means for encoding linguistic annotation as well as some generic markup for graphs, networks, trees, feature-structures, and links. On the other hand, it lacks explicit support for stand-off annotation style and makes use of entities, an almost obsoleted feature of XML, that originates in SGML. There are no tools supporting the full specification.

ISO LAF, MAF, SynAF, GrAF The Linguistic Annotation Format (LAF; Ide and Romary, 2004; Suderman and Ide, 2006) was developed roughly at the same time as PML. It encodes linguistics structures as directed graphs; both nodes and edges might be annotated with feature structures. LAF is very similar to PML, they both support stand-off annotations, feature structures, alternatives, etc. The following points are probably the main differences between the frameworks:

- LAF is an abstract format, independent of its serialization to XML, which is specified by the Graph Annotation Format (GrAF; Ide and Suderman, 2007). PML is an XML based format, but in principle it could be encoded in other structured languages such as JSON.
- While PML allows encoding general graphs in the same way as GrAF, for certain specific graphs it is recommended to use encoding by XML structures: simple paths by a sequence of XML elements and trees by embedding. This greatly simplifies parsing and validation and prevent lots of errors. (In theory, these errors should be prevented by the use of appropriate applications, but in practice the data are often modified by hand or low level tools.) In addition, many problems can be solved significantly faster for trees or sequences than for general graphs.
- PML is supported with a rich set of tools (TrEd and other tools described in this paper). We were not able to find a similar set of tools for LAF.

Plain text There are many advantages of a structured format over a plain-text vertical format (e.g. popular CoNLL Shared Task format). The main drawbacks of the simpler plain-text format is that it does not support standard encoding of meta information, and that complex structures (e.g. lists of lists) and relations in multi-layered annotation are encoded in an ad-hoc fashion which is prone to errors. For details, see (Straňák and Štěpánek, 2010).

EXMARaLDA We have also used PML to encode the Czesl learner corpus. As the corpus uses layered annotation, the only established alternative was the tabular format used by EXMARaLDA (Schmidt, 2009). However, the format has several disadvantages (see, e.g. Hana et al., 2010; Hana et al., 2012). Most importantly, the correspondences between the original word form and its corrected equivalents or annotations at other levels may be lost, especially

[12] http://ufal.mff.cuni.cz/styx/

[13] http://www.tei-c.org/Guidelines/P5/

for errors in discontinuous phrases. The feat editor supports import from and export to several formats, including EXMARaLDA.

5 Future Work

In the current specification, PML instances use a dedicated namespace. A better solution would be to let the user specify his or her own namespace in a PML schema. Support for handling additional namespaces would also be desirable (one might use it e.g. to add documentation or comments to schemas and data), however, this feature need much more work: if some PML elements are moved or deleted by an application, should it also move or delete the foreign namespace content?

List members and alternative members are always represented by `<LM>`, resp. `<AM>` XML elements. Several users requested a possibility to define a different name in a PML schema. This change would make the data more readable for human eyes, but it might complicate the internal data representation.

We would also like to extend support of PML in Java and add support for additional languages.

Finally, we plan to perform a detailed comparison with the LAF-based formats and create conversion tools between PML and LAF.

Acknowledgements

This work has been using language resources developed and/or stored and/or distributed by the LINDAT-Clarin project of the Ministry of Education of the Czech Republic (project LM2010013). This paper and the development of the framework were also supported by the Grant Agency of the Czech Republic (grants 406/10/P193 and 406/10/P328).

References

Sašo Džeroski, 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.

Jan Hajič, Jarmila Panevová, Eva Hajičová, Petr Sgall, Petr Pajas, Jan Štěpánek, Jiří Havelka, Marie Mikulová, Zdeněk Žabokrtský, and Magda Ševčíková-Razímová. 2006. *Prague Dependency Treebank 2.0.* Linguistic Data Consortium, Philadelphia. CD-ROM, CAT: LDC2001T10.

Jirka Hana and Barbora Hladká. 2012. Getting more data – Schoolkids as annotators. In *Proceedings of the Eighth Conference on Language Resources and Evaluation (LREC 2012)*, Istanbul. accepted.

Jirka Hana, Alexandr Rosen, Svatava Škodová, and Barbora Štindlová. 2010. Error-tagged Learner Corpus of Czech. In *Proceedings of The Fourth Linguistic Annotation Workshop (LAW IV)*, Uppsala.

Jirka Hana, Alexandr Rosen, Barbora Štindlová, and Petr Jäger. 2012. Building a learner corpus. In *Proceedings of the Eighth Conference on Language Resources and Evaluation (LREC 2012)*, Istanbul. accepted.

Nancy Ide and Laurent Romary. 2004. International standard for a linguistic annotation framework. *Nat. Lang. Eng.*, 10(3-4):211–225, September.

Nancy Ide and Keith Suderman. 2007. Graf: a graph-based format for linguistic annotations. In *Proceedings of the Linguistic Annotation Workshop*, LAW '07, pages 1–8.

Petr Pajas and Jan Štěpánek. 2006. XML-based representation of multi-layered annotation in the PDT 2.0. In Richard Erhard Hinrichs, Nancy Ide, Martha Palmer, and James Pustejovsky, editors, *Proceedings of the LREC Workshop on Merging and Layering Linguistic Information (LREC 2006)*, pages 40–47.

Petr Pajas and Jan Štěpánek. 2008. Recent advances in a feature-rich framework for treebank annotation. In Donia Scott and Hans Uszkoreit, editors, *The 22nd International Conference on Computational Linguistics – Proceedings of the Conference*, volume 2, pages 673–680, Manchester.

Petr Pajas and Jan Štěpánek. 2009. System for querying syntactically annotated corpora. In Gary Lee and Sabine Schulte im Walde, editors, *Proceedings of the ACL-IJCNLP 2009 Software Demonstrations*, pages 33–36, Singapore. Association for Computational Linguistics.

Martin Popel and Zdeněk Žabokrtský. 2010. TectoMT: Modular NLP framework. In *Proceedings of IceTAL, 7th International Conference on Natural Language Processing*, pages 293–304, Reykjavik.

Prokopis Prokopidis, Elina Desypri, Maria Koutsombogera, Haris Papageorgiou, and Stelios 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.

Thomas Schmidt. 2009. Creating and working with spoken language corpora in EXMARaLDA. In *LULCL II: Lesser Used Languages & Computer Linguistics II*, pages 151–164.

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

Jan Štěpánek and Petr Pajas. 2010. Querying diverse treebanks in a uniform way. In *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC 2010)*, pages 1828–1835. European Language Resources Association.

Pavel Straňák and Jan Štěpánek. 2010. Representing layered and structured data in the CoNLL-ST format. In Alex Fang, Nancy Ide, and Jonathan Webster, editors, *Proceedings of the Second International Conference on Global Interoperability for Language Resources*, pages 143–152, Hong Kong.

Keith Suderman and Nancy Ide. 2006. Layering and merging linguistic annotations. In *Proceedings of the 5th Workshop on NLP and XML: Multi-Dimensional Markup in Natural Language Processing*, NLPXML '06, pages 89–92.

Marko Tadić. 2007. Building the Croatian Dependency Treebank: the initial stages. In *Contemporary Linguistics*, volume 63, pages 85–92.

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

Leonoor van der Beek, Gosse Bouma, Robert Malouf, and Gertjan van Noord. 2002. The Alpino Dependency Treebank. In *Computational Linguistics in the Netherlands CLIN 2001*, Amsterdam.

Lonneke van der Plas, Tanja Samardzic, and Paola Merlo. 2010. Cross-lingual validity of PropBank in the manual annotation of French. In *Proceedings of The Fourth Linguistic Annotation Workshop (LAW IV)*, Uppsala.