MATEMATICKO-FYZIKÁLNÍ FAKULTA

PRAHA

TECTOMT, DEVELOPER'S GUIDE

Zdeněk Žabokrtský, Ondřej Bojar

úfal/ckl technical report

TR-2008-38

UNIVERSITAS CAROLINA PRAGENSIS

TectoMT,

Developer's Guide

Zdeněk Žabokrtský
Ondřej Bojar

ÚFAL/CKL Technical Report TR-2008-39
ISSN 1214-5521

December 2008

Table of Contents

1. Introduction

1.1. What is TectoMT?

TectoMT is a highly modular NLP (Natural Language Processing) software system implemented in Perl programming language under Linux. It is primarily aimed at Machine Translation, from English to Czech in the first phase, making use of the ideas and technology created during the Prague Dependency Treebank project. At the same time, it is also hoped to significantly facilitate and accelerate 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.

This document describes TectoMT from the technical viewpoint. The theoretical background related to the translation itself (the question of lexical transfer etc.) are not discussed here.

1.2. What is not TectoMT?

TectoMT -- as a whole -- is not an end-user application, and will never be. It is an experimental development environment, too large and too complex to become a widely used robust product; also, authors rights and specific licenses associated with some of the integrated components must be respected. However, building and public releasing of real end-user applications (consisting of selected TectoMT components) is possible and supported by the current TectoMT architecture.

1.3. What was the motivation for creating TectoMT?

When we started developing the pilot version of TectoMT in autumn 2005, our motivation for building the system was twofold.

First, we believe that the abstraction power offered by the tectogrammatical layer of language representation (as introduced by Petr Sgall in 1960's and recently implemented within the Prague Dependency Treebank project, t-layer for short) can contribute to the state-of-the-art in Machine Translation. Not only that the system based on "tecto" should not loose its linguistic interpretability in any phase and thus it should allow for simple debugging and monotonous improvements. Compared to the popular n-gram translation models, there are advantages also from the statistical viewpoint. Namely, abstracting from the repertoires of language means (such as inflection, agglutination, word order, functional words, intonation), which are used to varying extent in different languages for expressing non-lexical meanings, should make the training data contained in available parallel corpora much less sparse (data sparseness is a notorious problem in MT), and thus better machine-learnable.

Second, even if the first assumption might be wrong, we are sure it would be helpful for me and our colleagues at the institute to be able to integrate existing NLP tools (be they ours or external) into a common software framework. Thus we could ultimately get rid of the endless format conversions and frustrating ah-hoc tweaking of other people's source codes whenever one wants to perform any single operation on any single piece of linguistic data.

1.4. Who can use TectoMT?

'Can' relates to two meanings here: (a) 'to be able' and (b) 'to be allowed.'

Ad (a): As mentioned above, TectoMT is rather a development software framework, far from the end-user application shape. It can be effectively used only by programmers with at least a basic experience in Linux/bash, including e.g. writing/understanding simple Makefiles, and with advanced knowledge of Perl, including OO-programming. Experience in working with (and customizing of) the tree editor TrEd might be also very useful, as well as knowledge of PML (xml-based Prague Markup Language) and especially knowledge of the layered annotation scenario of the Prague Dependency Treebank.

Ad (b): As for licensing, most TectoMT source codes are available under GNU General Public License, version 2.0, which is always explicitly noted in the files. However, the license status of the system as a whole is not formally clear at this moment, as there will always be some components in TectoMT which we are allowed to use but not to freely distribute or re-license under GNU GPL. So all TectoMT developers are asked not to distribute TectoMT as a whole or its parts outside UFAL, unless they carefully checked all the licence issues.

1.5. Related documents

- Official TectoMT website: http://ufal.mff.cuni.cz/tectomt/
- TectoMT tutorial: https://wiki.ufal.ms.mff.cuni.cz/external:tectomt:tutorial
- Online version of this document: http://ufal.mff.cuni.cz/tectomt/guide/guidelines.html

1.6. How to cite TectoMT?

The users of TectoMT are kindly asked to refer to the first published work about TectoMT:

Zdeněk Žabokrtský, Jan Ptáček, and Petr Pajas. 2008. TectoMT: Highly modular MT system with tectogrammatics used as transfer layer. In Proceedings of the Third Workshop on Statistical Machine Translation, pages 167–170, Columbus, Ohio, June. Association for Computational Linguistics.

1.7. Acknowledgments

The work on this project was partially supported by 1ET101120503, MSM 0021620838 and FP6-IST-5-034291-STP (Euromatrix).

2. TectoMT Architecture Overview

2.1. Main design decisions

The implementation of TectoMT is based on the following design decisions:

- *Modularity* is emphasized in TectoMT. Any non-trivial NLP task should be decomposed into a sequence of subsequent steps, implemented as so called *blocks*. The sequences of blocks (strictly linear, without branches) are called *scenarios*.
- Each block should have a well-documented, meaningful, and — if possible — also linguistically interpretable functionality, so that it can be easily substituted with an alternative solution (another block), which attempts at solving the same subtask using a different method/approach. Since granularity of the task decomposition is not given in advance, one block can have the same functionality as an alternative solution composed of several blocks (e.g., some taggers perform also lemmatization, whereas another taggers have to be followed by separate lemmatizers). As a rule of thumb, the size of a block should not exceed several hundred lines of code.
- Each *block is a Perl module* (more specifically, a Perl class with an inherited interface). However, this does not mean that the solution of the task itself has to be implemented in Perl too: the module itself can be only a wrapper for a binary application or a Java application, or a client of a web service running on a remote machine, etc.
- In order to allow a fully automatic, repeated and parallelized execution of block sequences, blocks can rely on *no user interaction*. They can communicate exclusively via the prescribed API. Of course, this does not exclude the possibility of using them later in an interactive application.
- TectoMT is implemented in *Linux*. Full portability of the whole TectoMT to other operating systems is not realistic in the near future. But again, this does not exclude the possibility of releasing platform independent applications made of selected components. So, naturally, platform independent solutions should be searched whenever possible. Needless to say that hardware-architecture independent solutions should be preferred too.
- Processing of any type of linguistic data in TectoMT can be viewed as a path through the Vauquois triangle (with vertical axis corresponding to the level/layer of language abstractions and horizontal axis possibly corresponding to different languages). It should be always clear with which layers a given block works. By default, TectoMT mirrors the system of layers as developed in the PDT (morphological layer, analytical layer for surface dependency syntax, tectogrammatical layer for deep syntax), but other layers might be added too. By default, sentence representation at any level is supposed to form a tree (even if it is a flat tree on the morphological level and even if co-reference links might be seen as non-tree edges on the tectogrammatical layer).
- TectoMT is *neutral with respect to the methodology* employed in the individual blocks: fully stochastic, hybrid, or fully symbolic (rule-based) approaches can be

used. The only preference is following: the solution which reaches the best evaluation result for the given subtask (according to some measurable criteria) is the best.

- Any block in TectoMT should be capable of *massive data processing*. It makes no sense to develop a block which needs in average more than a few hundred milliseconds per processed sentence (rule of thumb: the complete translation block sequence should not need more than a couple of seconds per sentence). Also, memory requirements of any block should not exceed reasonable limits, so that individual developers can run the blocks using their "home computers".
- *TectoMT is composed of two parts*. The first part (the development part), which contains especially the processing blocks and other in-house tools and Perl libraries, is stored in an SVN repository so that it can be developed in parallel by more developers (and also outside the UFAL Linux network). The second part (the shared part), which contains downloaded libraries, downloaded software tools, independently existing linguistic data resources, generated data, etc., is shared without versioning because (a) it is supposed to be changed more or less only additively, (b) it is huge, as it contains large data resources, and (c) it should be automatically reconstructable (simply by redownloading, regeneration or reinstallation of its parts) if needed.
- Typically, TectoMT processing of linguistic data is composed of three steps: (1) convert your data (e.g. a plain text to be translated) into the tmt data format (PML-based format developed for TectoMT purposes), (2) apply the sequence of processing blocks, using the TectoMT object-oriented interface to the data, (3) convert the resulting structures to the desired output format (e.g., HTML containing the resulting translation).
- The main difference between the tmt data format and the PML applications used in PDT 2.0 is the following: in tmt, all representations of a textual document at the individual layers of language description are stored in one single file. As the number of linguistic layers in TectoMT might multiplied by the number of processed languages (two or more in the case of parallel corpora) and by direction of their processing (source vs. target during translation), manipulation with a growing number of files corresponding to a single textual document would become too cumbersome.

2.2. Directory structure

As already said, TectoMT system is composed of (1) the small versioned development part, and (2) the large unversioned part called `share`.

The development (versioned) part is structured as follows (of course, the subdirectory listings are not complete):

```
tectomt/      # you can name this root directory as you like, but $TMT_ROOT system variab
le must point here
|
+--libs/      # "in-house" Perl modules (developed for TectoMT purposes)
|   +--core/  # core classes for general processing units and processed ling. structures
|   |   +--TectoMT/
|   |   |   +--Block.pm, Scenario.pm    # processing blocks and their sequences
|   |   |   +--Document.pm, Bundle.pm   # representation of documents and sentence bundles
|   |   |   +--Node.pm                  # general node
|   |   |   +--Node/                    # specific types of nodes
|   |   |       +--T.pm                 # general t-layer nodes
|   |   |       +--SEnglishT.pm         # t-layer node on English (source) side
|   |   +--Report.pm                    # module for printing error, warning and debug mes
sages
|   +--blocks/                          # processing blocks, derived from TectoMT::Blocks
|   |   +--SEnglishA_to_SEnglishT/      # blocks from converting English a-layer to t-
layer
|   |   +--SEnglishT_to_TCzechT/        # English (source) t-layer to Czech (target) t-
layer
|   |   +--Tutorial/
|   |   +--BlockTemplate.pm
|   +--other/
|
+--config/
|   +--init_devel_environ.sh
|   +--TectoMT_TredMacros.mak
|   +--tred_stylesheets/
|
+--tools/
|   +--general/
|   |   +--brunblocks
|   +--format_convertors/
|   |   +--plaintext_to_tmt/plaintext_to_tmt.pl
|   |   +--tmt_to_pedtpml/tmt_to_pedtpml.pl
|   +--format_validators/
|
+--pml_schemas/                         # specifications of PML schemas used in TectoMT
|   +--tmt_schema.xml                   # PML schema of the tmt format
|
+--applications/
|   +--analysis/
|   |   +--cs/
|   |   +--en/
|   +--demo/
|   |   +--alignment.scen
|   |   +--Makefile
|   +--tutorial/
|
+--evaluation/
|   +--compare_czech_taggers/
|
+--personal/                            # space for experiments of the individual users
|   +--klimes/
|   +--ptacek/
|   +--zabokrtsky/
|
+--tests/
|
+--release_building/  # packaging of "TectoMT-independent" applications for users outside 
UFAL
+--tmp/               # mount point for temporary data directory
+--share/             # mount point for unversioned part -see next figure
```

The shared (unversioned) part of TectoMT is structured as follows:

```
tectomt/
+--share/                   # either a regular directory or a symlink
   +--data/
   |  +--models/
   |     +--morpho_analysis/
   |     |  +--cs/
   |     |  +--en/
   |     +--tecto_transfer/
   |        +--cs2en/
   |        +--en2cs/
   +--external_libs/          # Perl modules implemented elsewhere (esp. CPAN)
   +--external_tools/         # software tools implemented elsewhere
   +--releases/               # archive of releases of applications for users outside UFAL
   +--tred/                 #
```

3. Data Structure Units in TectoMT

3.1. Documents, Bundles, Trees, Nodes, Attributes

In TectoMT, linguistic representations of running texts are organized in the following hierarchy:

- One physical file corresponds to one document.
- A document consists of a sequence of bundles, mirroring a sequence of natural language sentences (typically, but not necessarily, originating from the same text). Attributes (attribute-value pairs) can to attached to a document as a whole.
- A bundle one sentence in its various forms/representations (esp. its representations on various levels of language description, but also possibly including its counterpart sentence from a parallel corpus, or its automatically created translation, and their linguistic representations, be they created by analysis / transfer / synthesis). Attributes can be attached to a bundle as a whole.
- All sentence representations are tree-shaped structures - the term bundle stands for 'a bundle of trees'.
- In each bundle, its trees are "named" by the names of layers, such as SEnglishM (see the next section). In other words, there is at most one tree for a given layer in each bundle.
- Trees are formed by nodes and edges. Attributes can be attached only to nodes. Edge's attributes must be equivalently stored as the lower node's attributes. Tree's attributes must be stored as attributes of the root node.
- Attributes can bear atomic values or can be further structured (besides atomic values also lists, structures etc.), as allowed by PML.

For those, who are acquainted with the structures used in PDT 2.0, the most important difference lies in bundles: the level added between documents and trees, which comprises all layers of representation of a given sentence. As one document is stored as one physical file, all layers of language representations can be stored in one file in TectoMT (unlike in PDT 2.0).

3.2. 'Layers' of Linguistic Structures

The notion of 'layer' has a combinatorial nature in TectoMT. It corresponds not only the layer of language description as used e.g. in the Prague Dependency Treebank, but it is also specific for a given language (e.g., possible values of morphological tags are typically different for different languages) and even for how the data on the given layer were created (whether by analysis from the lower layer or by synthesis/transfer).

Thus, the set of TectoMT layers is Cartesian product {S,T} x {English,Czech} x {W,M,P,A,T}, in which:

- {S,T} distinguishes whether the data was created by analysis or transfer/synthesis (mnemonics: S and T correspond to (S)ource and (T)arget in MT perspective).

- {English,Czech...} represents the language in question
- {W,M,P,A,T...} represents the layer of description in terms of PDT 2.0 (W - word layer, M - morphological layer, A - analytical layer, T - tectogrammatical layer) or extensions (P - phrase-structure layer).

TectoMT layers are denoted by stringifying the three coordinates: for example, analytical representation of an English sentence acquired by sentence analysis is denoted as SEnlishA. This naming convention is used on many places in TectoMT: for naming trees in a bundle (and corresponding xml elements), for naming blocks, for node identifier generating, etc.

Unlike layers in PDT 2.0, the set of TectoMT layers should not be understood as totally ordered. Of course, there is a strong intuition based the abstraction axis of languages description (SEnglishT requires more abstraction than SEnglishM), but the intuiting might not be sufficient in some cases (SEnglishP and SEnglishA represent roughly the same level of abstraction).

3.3. TectoMT API to linguistic structures

The linguistic structures in TectoMT are represented using the following object-oriented interface/types:

- document - `TectoMT::Document`
- bundle - `TectoMT::Bundle`
- node - `TectoMT::Node`
- document's, bundle's, and node's attributes - Perl scalars in case the PML schema prescribes an atomic type, or an appropriate class from `Fslib` correspondingly to the type specified in the PML schema.

Classes TectoMT::{Document,Bundle,Node} have their own documentation, here we list only the basic methods for navigating through a TectoMT document (Perl variables such as `$document` are used only for illustration purposes, but there are no global/predefined variables like this in TectoMT). "Contained" objects encapsulated in "container" objects can be accessed as follows:

- `my @bundles = $document->get_bundles` - an array of bundles contained in the document
- `my $root_node = $bundle->get_tree($layer_name);` - the root node of the tree of the given type in the given bundle

There are also methods for accessing the container objects from the contained objects:

- `my $document = $bundle->get_document;` - the document in which the given bundle is contained
- `my $bundle = $node->get_bundle;` - the bundle in which the given node is contained
- `my $document = $node->get_document;` - composition of the two above

There are several methods for traversing tree topology, such as

- `my @children = $node->get_children;` - array of the node's children
- `my @descendants = $node->get_descendants;` - array of the node's children and their children and children of their children ...
- `my $parent = $node->get_parent;` - parent node of the given node, or undef for root
- `my $root_node = $node->get_root;` - the root node of the tree into which the node belongs

Attributes of documents, bundles or nodes can be accessed by attribute getters and setters:

- `$document->get_attr($attr_name); $document->set_attr($attr_name, $attr_value);`
- `$bundle->get_attr($attr_name); $bundle->set_attr($attr_name, $attr_value);`
- `$node->get_attr($attr_name); $node->set_attr($attr_name, $attr_value);`

$attr_name is always a string (following the Fslib conventions in the case of structured attributes, e.g. using slash in structured attributed, e.g. 'gram/gender').

New classes, with functionality specific only for some layers, can be derived from TectoMT::Node. For example, methods for accessing effective children/parents should be defined for nodes of dependency trees. Thus, there are for example classes `TectoMT::Node::SEnglishA` or `TectoMT::Node::SCzechA` offering methods get_eff_parents and get_eff_children, which are inherited from a general analytical 'abstract class' `TectoMT::Node::A` (which itself is derived from `TectoMT::Node`). Please note that the names of the 'terminal' classes are the same as the layer names. If there is no specific class defined for some layer, `TectoMT::Node` is used as a default for nodes on this layer.

All these classes are stored in `devel/libs/core`. Obviously, they are crucial for functioning of most other components of TectoMT, so their functionality should be carefully checked after any changes.

3.4. Fslib representation behind the API

Technically, the data structures are not stored directly in `TectoMT::{Document,Bundle,Node}` representation, but there is an underlying representation using Petr Pajas's Fslib library. Practically the only data stored in TectoMT objects (besides some indexing) are references to Fslib objects. Combination of a new OO API (TectoMT) with the previously existing library (Fslib) used for the underlying memory representation was chosen because of the following reasons:

- In Fslib, it would not be possible to make the objects fully encapsulated, to introduce node-class hierarchy, and it would be very difficult to redesign the existing Fslib API (classes, functions, methods, data structures), as there is a heap of existing code dependent on Fslib. So developing a new API seemed to be necessary.
- On the other hand, there are two important advantages of using the Fslib

representation. First, we can use Prague Markup Language as the main file format, since serialization into PML (and reading PML) is fully implemented in Fslib. Second, since we use one of Fslib-compatible file format, we can use also the tree editor TrEd for visualizing the structures and btred/ntred for comfortable batch processing of our data files.

Outside the core libraries, there is almost no need to access the underlying Fslib representation -- the data should be accessed exclusively via the TectoMT interface (unless some very special Fslib functionality is needed). However, the underlying Fslib representation can be accessed from the TectoMT instances as follows:

- `$document->get_tied_fsfile()` returns the underlying FSFile instance
- `$bundle->get_tied_fsroot()` returns the underlying FSNode instance
- `$node->get_tied_fsnode()` returns the underlying FSNode instance

3.5. TMT File Format

The main file format used in TectoMT is TMT (.tmt ending). TMT format is an application of PML. Thus, TMT files are PML instances of a PML schema. The schema is stored in `${TMT_ROOT}/pml/tmt_schema.xml`. This schema merges and changes (more or less additively) the PML schemata from PDT 2.0.

The PML schema directly renders the logical structure of data: there can be one document in one tmt-file, the document has its attributes and contains a sequence of bundles, each bundle has its attributes and contains a set of trees (named by layer names), each tree consists of nodes, which again contain attributes.

Files in the TMT format are readable by naked eye, but this is in fact useful only when writing and debugging format convertors from TMT to other formats or back. Otherwise, it is much more comfortable to view the data in TrEd.

In TectoMT, one should never write components accessing directly the TMT files (of course, with the only exception of convertors from other formats to TMT or back). Instead, the data should be accessed by the components exclusively via the above mentioned object-oriented Perl API.

4. Processing units in TectoMT

4.1. Hierarchy of Processing Units

In TectoMT, there is the following hierarchy of processing units (software components that process data):

- The basic units are *blocks*. They serve for some very limited, well defined, and often linguistically interpretable tasks (e.g., tokenization, tagging, parsing). Blocks are not parametrizable. Technically, blocks are Perl classes inherited from `TectoMT::Block`.
- To solve a more complex task, selected blocks can be chained into a *block sequence*, called also a *scenario*. Technically, scenarios are instances of `TectoMT::Scenario` class, but in some situations (e.g. on the command line) it is sufficient to specify the scenario simply by listing block names separated with spaces.
- The highest unit is called application. Applications correspond to end-to-end tasks, be they real end-user applications (such as machine translation), or 'only' NLP-related experiments. Technically, applications are often implemented as Makefiles, which only glue the components existing in TectoMT.

4.2. Blocks

Technically, blocks are Perl classes derived from `TectoMT::Block`. In order to make them easily readable for other TectoMT developers, please use the following conventional structure when writing new blocks:

1. block (package) name on the first line,
2. use of pragmas and libraries
3. possibly some initialization (e.g. loading external data)
4. declaration of the `process_document` method
5. short POD documentation
6. author's copyright notice

Example of a simple block, which causes that negation particles in English will be considered as a part of verb forms during the transition from the SEnglishA layer to the SEnglishT layer:

```
package SEnglishA_to_SEnglishT::Mark_negator_as_aux;

use 5.008;
use strict;
use warnings;
use base qw(TectoMT::Block);

sub process_document {
  my ($self,$document) = @_;

  foreach my $bundle ($document->get_bundles()) {
    my $a_root = $bundle->get_tree('SEnglishA');

    foreach my $a_node ($a_root->get_descendants) {
      my ($eff_parent) = $a_node->get_eff_parents;
      if ($a_node->get_attr('m/lemma')=~/^(not|n\'t)$/
          and $eff_parent->get_attr('m/tag')=~/^V/ ) {
        $a_node->set_attr('is_aux_to_parent',1);
      }
    }
  }
}

1;
=over

=item SEnglishA_to_SEnglishT::Mark_negator_as_aux
'not' is marked as aux_to_parent (which is used in the translation scenarios,
but not in preparing data for annotators)

=back

=cut

# Copyright 2008 Zdenek Zabokrtsky
```

Blocks are stored in subdirectories of the `libs/blocks/` directory. Most blocks are distributed among the directories according to their position along the virtual path through the Vauquois triangle. More specifically, they are part of a transition from layer L1 to layer L2. Such blocks are stored in the <L1>_to_<L2> directory, e.g. in SEnglishA_to_SEnglishT. But there are also blocks for other purposes, e.g. evaluation blocks (`libs/blocks/Eval/`) or data extraction blocks (`libs/blocks/Print/`).

4.3. Scenarios

Scenarios have a strictly linear nature: the blocks are applied on tmt documents one after another, there can be no branches or cycles.

In Perl, a scenario instance can be created and applied on a TectoMT document instance as follows:

```
my $scenario =  TectoMT::Scenario->new({'blocks' => [ qw(
        SCzechW_to_SCzechM::Tokenize
        SCzechW_to_SCzechM::Simple_tagger
        SCzechW_to_SCzechM::Simple_lemmatizer
) ]});
```

```
$scenario->apply_on_tmt_documents($document);
```

In Bash, applying a sequence of blocks on a TMT file looks e.g. as follows (brunblocks alias will be described later):

```
$  brunblocks -o  SCzechW_to_SCzechM::Tokenize.pm \
        SCzechW_to_SCzechM::Simple_tagger \
        SCzechW_to_SCzechM::Simple_lemmatizer \
        -- demo.tmt
```

4.4. Applications

Typically, an application consists of three steps (not counting the 0th step of initialization the common development environment, as will be described later): (1) conversion of the input data into TMT, (2) applying a scenario on the TMT files, (3) conversion from TMT into the desired output format.

4.5. Common Development Environment

By Common Development Environment we understand several system variables' and aliases' settings. Such settings are recommended to be always performed in the current shell before starting work with TectoMT. There are two reasons for such initialization:

- TectoMT consists of two parts, versioned and unversioned. There must be a way how, for instance, a running code (typically from the versioned part) finds where some data file (possibly from the shared part) is stored. Also, paths to Perl libraries (contained both in the versioned and unversioned part, and also in the directory tree in which Tred is installed) have to be set. Obviously, if TectoMT should be usable outside the UFAL network, then it cannot rely on any absolute paths, but the location of the two main parts should be specified and all other paths should be derived from them.
- Second, working with TectoMT can be made more comfortable if one can share various aliases (accumulated in one place rather than in `.bashrc` of the individual developers), for instance for customizing TrEd to work with the TMT format.

Initialization of the Common Development Environment is performed in Bash by sourcing `config/init_devel_environ.sh`, which manifests as follows:

- Newly introduced system variables
    - `TMT_ROOT` - path to your working copy of the versioned part of TectoMT
    - `TMT_SHARED` - path to the unversioned part of TectoMT
    - `TMT_TEMP` - path to the directory for temporary files
    - `TRED_DIR` - path to the directory where TrEd is installed
- Modified system variables
    - `PERLLIB`, `PERL5LIB` - path to your working copy of the versioned part of TectoMT
    - `PATH` - paths to tools (inside root/shared)
- as `$TMT_ROOT/tools/general/` is added to `PATH`, the following commands become available:
    - `tmttred` - TrEd customized for TectoMT using several command line options (path to resources, stylesheet, etc.)
    - `tmtbtred` - btred customized for TectoMT
    - `tmtntred` - ntred customized for TectoMT
    - `brunblocks` - alias for applying a sequence of blocks on tmt-files (usage: brunblocks -o <blocks> -- <tmt_files>
    - `nrunblocks` - alias for applying a sequence of blocks on tmt-files currently loaded in ntred (usage: nrunblocks <blocks>

5. Debugging Tips

Debugging complex TectoMT applications on bigger data can be quite painful. Here are some tips we found useful.

5.1. Auto-Diagnose (Automatically Create Minimal Testcase)

If you have a `.tmt` file and a sequence of blocks that crashes somewhere, *minimize it* to speed up the loop of bug fixing or to attach it to a bug report for someone.

`devel/tools/tests/auto_diagnose.pl` will automatically create a minimal testcase for you: the first problematic sentence will be extracted from the `.tmt` file and analyzed just before the first crashing block. Finally the command line (`brunblocks`) to run the minimized test case is provided as a tiny shell script.

6. Processing Large Data, Grid Computing

TectoMT allows processing of large to huge data sets under the following conditions:

1. All files are relatively small (e.g. 50 to 200 sentences per file).
2. All directories contain relatively few files (e.g. not more than 1000 files per directory, including backup copies).
3. You use a cluster of CPUs administered by Sun Grid Engine (SGE).

6.1. qrunblocks: Parallel Processing on a Grid

In a grid environment of Sun Grid Engine (where commands like `qsub` work), you can use `qrunblocks` to apply a scenario on a set of files in parallel.

`qrunblocks` is available in `$TMT_ROOT/devel/tools/cluster_utils/qrunblocks`.

6.1.1. Basic Usage

The basic usage is:

```
qrunblocks filelist blocks
```

The set of files is splitted into `--jobs|-j` jobs. All the jobs are submitted to the grid to process the files using the scenario.

The set of files can be specified either using the filelist file or using a wildcarded expression in a `--glob|-g` option, e.g.: `--glob 'mydata/*.tmt.gz'`. The quotation marks are necessary to avoid wildcard expansion already in your shell.

The scenario, i.e. the sequence of blocks, can be specified either simply by listing the sequence in the second argument (`qrunblocks filelist 'Block1 Block2'`) or by loading the sequence from a file using `--blocksfile|-b=file`

Note that `qrunblocks` has to init the TectoMT environment in all slave processes. If the environment variable `$TMT_ROOT` is set, `qrunblocks` will use the given TectoMT root in all the slaves. Otherwise, you need to specify the path to your TectoMT root using the parameter `--tmt-root=PATH`.

It is a common mistake to forget to save the processed files. To preserve the computation time, `qrunblocks` assumes that the default is to *save* files and forces saving in all slave processes. If you don't want to save the files (e.g. because you were only collecting standard output), use `--no-save`.

Note that `qrunblocks` is very different from `ntred`-based processing where each of the servers loads its portion of files to memory. `qrunblocks` is also not based on `jtred`, the grid alternative of `btred`.

6.1.2. Grid Parameters, e.g. Priority

`qrunblocks` recognizes and passes the following parameters to Sun Grid Engine:

- `--jobname|-N=NAME` specifies the name of the job and also the base file name for all the log files. The default is `qrunblocks`.
- `--priority|-p=-100` specifies the priority of the jobs *before* submission.
- `--mem|-m=10G` should specify the memory requirements of each of the jobs. Due to weird issues in the SGE configuration at ÚFAL, this option does not really work.

6.1.3. Passing Parameters to qrunblocks Jobs

Some TectoMT blocks can be influenced by parameters specified as environment variables `$TMT_PARAM_something`. To pass these parameters to individual jobs, you have to explicitly ask for it:

- `--export=VARNAME` or `-e VARNAME` will pass the environment variable `$VARNAME` to all the jobs. You can use this for any variable, not just `$TMT_PARAM_something`. The option can be repeated.
- `--export-all-tmt-params` or `-E` will export all `$TMT_PARAM_something` variables from the current environment.

6.1.4. Collecting Output or Checking Status

The default behaviour of `qrunblocks` is to submit all the jobs and immediately exit. It is your responsibility to examine the log files and check exit status of all the jobs (see `Status:` at the end of the log if it says `FAILED`).

Launching `qrunblocks` with `--sync` causes `qrunblocks` to block until all the jobs have ended or exited. The exit status of the jobs is *not* reflected in the exit status of `qrunblocks`.

The safest way of launching `qrunblocks` is to use the flag `--join`. With this option, `qrunblocks` will wait for all the jobs to finish and if all succeed, their standard outputs will be concatenated and printed to `qrunblocks`' standard output. If any of the jobs fails, `qrunblocks` exits with non-zero exit status as well.

6.1.5. Retrying Failed Files and Automatic Retry

There are situations where the block sequence may fail due to a rather random coincidence, for example if several jobs compete for RAM. In such cases, the easiest solution is simply to re-run the jobs.

`qrunblocks` supports automatic restarts of failing jobs, just specify `--attempts|-a=number_of_attempts` on the command line.

When re-running the scenario, some of the files may have been successfully analyzed before the failure happened. To avoid re-analyzing of finished files, `qrunblocks` allows you to specify a keyword that identifies finished files. For example, if you are analyzing up to English t-layer, you may want to use `--finished-contains '<SEnglishT'` (note the opening angle bracket) to remove all files containing the XML tag `SEnglishT` the file

list, because they are quite likely already analyzed. (If a job happens to need a restart, further files will be removed the file list.)

The default behaviour of `qrunblocks` is to split the input file list evenly and let all the jobs do their filtering based on `--finished-contains`. If many files are already finished, this may lead to a disbalance in workload of individual jobs. Adding the flag `--filter-ahead` to `qrunblocks` solves the issue by first checking all the files and evenly splitting only the list of unfinished files, at the expense of non-parallel startup filtering.

6.2. Suggested Dataflow for Huge Datasets

We successfully parsed nearly a gigaword of Czech texts (51 million sentences) and 6 million of Czech-English parallel sentences up to the t-layers in TectoMT using the following dataflow:

1. Convert plaintext to a directory tree of small files on a shared network file system. (We keep the files comparable in size, e.g. 50 sentences per file.)

   Prefer to keep the files in a compressed form, i.e. `.tmt.gz` or `.pls.gz`, because it reduces the load on the NFS server.

   For an inspiration on the conversion see e.g. `tools/format_convertors/plaintext_to_tmt/plaintext_using_textseg_to_tmt.pl` or `tools/format_convertors/czeng07_to_tmt/czeng07_to_tmt.pl`.

2. Create filelist of all the files to be processed:

   ```
   find dataset-directory -name '*.tmt.gz' > dataset.list
   ```

3. Process all the files using a grid of computers:

   ```
   qrunblocks dataset.list --blocksfile scenario \
     --jobs 40 \
     --jobname MY_JOB
   ```

   Check the logfiles `MY_JOB.o[0-9]*` (default jobname is `qrunblocks`) for the final "`Status: succeeded|FAILED`".

   Beware: if the scenario is too quick (too little processing), running too many jobs at once can ruin your shared NFS server as all the jobs will write a lot of data.

4. Export analyzed sentences back to some low-level plaintext-like format, e.g.:

   ```
       export TMT_PARAM_PRINT_FACTORED="SEnglishT SCzechT
   SEnglishCzechAlignT"
       qrunblocks dataset.list "Miscel::SuicideIfDiskFull Print::Factored" \
         --jobs 40 \
         -E --no-save --join \
         --jobname MY_JOB.export \
       | gzip \
       > dataset.exported.gz
   ```

7. Quality Assurance

Stability of frequently used components of TectoMT is important. To ensure this, the whole TectoMT is checked-out and all pre-defined tests are launched every day.

If you want to rely on a component, make sure it is covered by one of the daily test. You can also add you own test.

7.1. Viewing Results of Daily Tests

Results on daily tests on various platforms are available here:
http://ufallab.ms.mff.cuni.cz/~bojar/cruise_control_tmt/

7.2. E-mail Notifications

If you wish to receive a notification about new problems, add your e-mail address to the variable RCPT in `devel/tests/Makefile`.

The notification is sent only in case a test (on a particular platform) passed yesterday but fails today.

7.3. Fixing a Bug!

Yes, indeed. The main purpose of the test suite is to let everyone fix bugs.

If a test you need or created fails, try changing relevant files (blocks/libraries/Makefiles/...). Then run the test on command-line (see below), and if you succeed, commit!

7.4. Running Tests on Command-Line

To run a test yourself, do the following:

```
cd devel/tests
make nice_file_names
# or
make try_test.test_tag_tnt
# or
make try_application.demo_translation_en2cs
# or any other test
```

7.5. Adding a New Test

There are several ways to add a new test. Choose the method according to you additional wishes.

- The very core method is to add a new goal to `devel/tests/Makefile`. This gives you full control of the test but no support (e.g. you have to init TectoMT environment yourself).
- The most visible method is to add a new application to `devel/applications/`. Running `make` (the default target) in your application directory should do the test.

To launch the application as a test from `devel/tests`, run `make try_application.YOUR_APPLICATION_NAME`.

- Somewhat intermediate method is to add a new subdirectory to `devel/tests`, see e.g. `test_mxpost`, again with a `Makefile` and the default target doing the job. This way, the test case is not so visible to all users of TectoMT but you can still easily have it launched.

7.6. Adding a Test to the Daily Suite

To have a test launched every day on all tested platforms, simply add it to the variable TESTS_TO_RUN in `devel/tests/Makefile`.

8. General Instructions for TectoMT Developers

Developers contributing to TectoMT are kindly asked to

1. read Damian Conway's Perl Best Practices,
2. make appropriate tests before committing,
3. prefer Perl/bash when writing new TectoMT components,
4. always derive paths to accessed files/directories from variables `$TMT_ROOT`, `$TMT_SHARED` etc., and never use absolute paths, paths to your home directories etc.
5. write POD in all their Perl programs/modules,
6. use Makefiles for organizing bigger tasks / experiments / applications,
7. add copyright notice (# Copyright <year> <name>) to all their source code files,
8. report detected bugs to author(s) of the respective piece of code, with CC to zabokrtsky@ufal.mff.cuni.cz.
9. respect naming conventions introduced in TectoMT (e.g. naming of layers),
10. write sufficiently descriptive and understandable comments on commits to the svn repository.
11. use Report::fatal, Report::warn or Report::info in blocks, instead of die/warn/print STDERR.
12. try to avoid situations in which your committed changes could break functionality of other people's code. For example, if you decide to rename methods in your library interface, find (e.g. grep) all spots in which these methods have been used and fix them too.
13. commit your work to the repository, even if you think it is not useful for anybody else. Otherwise your local copy may become incompatible with the rest of the TectoMT machinery after some time (see the above item).

9. TODOs

We are aware of the following issues which are not solved satisfactorily in TectoMT at this moment:

1. Adding new languages into TectoMT seems to require an inadequate amount of changes in the PML schema; adding new languages should be facilitated by allowing language parametrization, or by back-off (by default, some general scheme could be used for languages for which there is no specific scheme).
2. There is no mechanism for sending parameters to blocks. The question is how often it is really necessary/desirable, but definitely some solution must be found for blocks with more or less language-independent functionality, so that no new cut'n'paste blocks is necessary (generic blocks, which can be used for this task now, are problematic).
3. Location of Perl libraries: in the case of non-pure-Perl libraries, obviously there should be one (versioned) place where an installation package is developed and another (unversioned) place where the library is 'installed', but the second place should not be a part of tectomt_shared, otherwise functionality fall-outs for other UFAL users might appear.
4. At this moment, there is only one PML schema for all applications in TectoMT. Supporting separated application-specific schemas into TectoMT would not be trivial.
5. Similarly to the previous point, there is only one shared visualization style for all TMT files in TrEd. At this moment, there is no support for application-specific customization of bundles' appearance, which would allow for example easy relocation of the individual layers on the screen.

THE ÚFAL/CKL TECHNICAL REPORT SERIES

ÚFAL

ÚFAL (Ústav formální a aplikované lingvistiky; http://ufal.mff.cuni.cz) is the Institute of Formal and Applied linguistics, at the Faculty of Mathematics and Physics of Charles University, Prague, Czech Republic. The Institute was established in 1990 after the political changes as a continuation of the research work and teaching carried out by the former Laboratory of Algebraic Linguistics since the early 60s at the Faculty of Philosophy and later the Faculty of Mathematics and Physics. Together with the "sister" Institute of Theoretical and Computational Linguistics (Faculty of Arts) we aim at the development of teaching programs and research in the domain of theoretical and computational linguistics at the respective Faculties, collaborating closely with other departments such as the Institute of the Czech National Corpus at the Faculty of Philosophy and the Department of Computer Science at the Faculty of Mathematics and Physics.

CKL

As of 1 June 2000 the Center for Computational Linguistics (Centrum komputační lingvistiky; http://ckl.mff.cuni.cz) was established as one of the centers of excellence within the governmental program for support of research in the Czech Republic. The center is attached to the Faculty of Mathematics and Physics of Charles University in Prague.

TECHNICAL REPORTS

The ÚFAL/CKL technical report series has been established with the aim of disseminate topical results of research currently pursued by members, cooperators, or visitors of the Institute. The technical reports published in this Series are results of the research carried out in the research projects supported by the Grant Agency of the Czech Republic, GAČR 405/96/K214 ("Komplexní program"), GAČR 405/96/0198 (Treebank project), grant of the Ministry of Education of the Czech Republic VS 96151, and project of the Ministry of Education of the Czech Republic LN00A063 (Center for Computational Linguistics). Since November 1996, the following reports have been published.

ÚFAL TR-1996-01	Eva Hajičová, *The Past and Present of Computational Linguistics at Charles University* Jan Hajič and Barbora Hladká, *Probabilistic and Rule-Based Tagging of an Inflective Language – A Comparison*
ÚFAL TR-1997-02	Vladislav Kuboň, Tomáš Holan and Martin Plátek, *A Grammar-Checker for Czech*
ÚFAL TR-1997-03	Alla Bémová at al., Anotace na analytické rovině, *Návod pro anotátory (in Czech)*
ÚFAL TR-1997-04	Jan Hajič and Barbora Hladká, *Tagging Inflective Languages: Prediction of Morphological Categories for a Rich, Structural Tagset*
ÚFAL TR-1998-05	Geert-Jan M. Kruijff, *Basic Dependency-Based Logical Grammar*
ÚFAL TR-1999-06	Vladislav Kuboň, *A Robust Parser for Czech*
ÚFAL TR-1999-07	Eva Hajičová, Jarmila Panevová and Petr Sgall, *Manuál pro tektogramatické značkování (in Czech)*
ÚFAL TR-2000-08	Tomáš Holan, Vladislav Kuboň, Karel Oliva, Martin Plátek, *On Complexity of Word Order*
ÚFAL/CKL TR-2000-09	Eva Hajičová, Jarmila Panevová and Petr Sgall, *A Manual for Tectogrammatical Tagging of the Prague Dependency Treebank*
ÚFAL/CKL TR-2001-10	Zdeněk Žabokrtský, *Automatic Functor Assignment in the Prague Dependency Treebank*
ÚFAL/CKL TR-2001-11	Markéta Straňáková, *Homonymie předložkových skupin v češtině a možnost jejich automatického zpracování*

ÚFAL/CKL TR-2001-12 Eva Hajičová, Jarmila Panevová and Petr Sgall, *Manuál pro tektogramatické značkování (III. verze)*

ÚFAL/CKL TR-2002-13 Pavel Pecina and Martin Holub, *Sémanticky signifikantní kolokace*

ÚFAL/CKL TR-2002-14 Jiří Hana, Hana Hanová, *Manual for Morphological Annotation*

ÚFAL/CKL TR-2002-15 Markéta Lopatková, Zdeněk Žabokrtský, Karolína Skwarská and Vendula Benešová, *Tektogramaticky anotovaný valenční slovník českých sloves*

ÚFAL/CKL TR-2002-16 Radu Gramatovici and Martin Plátek, *D-trivial Dependency Grammars with Global Word-Order Restrictions*

ÚFAL/CKL TR-2003-17 Pavel Květoň, *Language for Grammatical Rules*

ÚFAL/CKL TR-2003-18 Markéta Lopatková, Zdeněk Žabokrtský, Karolina Skwarska, Václava Benešová, *Valency Lexicon of Czech Verbs VALLEX 1.0*

ÚFAL/CKL TR-2003-19 Lucie Kučová, Veronika Kolářová, Zdeněk Žabokrtský, Petr Pajas, Oliver Čulo, *Anotování koreference v Pražském závislostním korpusu*

ÚFAL/CKL TR-2003-20 Kateřina Veselá, Jiří Havelka, *Anotování aktuálního členění věty v Pražském závislostním korpusu*

ÚFAL/CKL TR-2004-21 Silvie Cinková, *Manuál pro tektogramatickou anotaci angličtiny*

ÚFAL/CKL TR-2004-22 Daniel Zeman, *Neprojektivity v Pražském závislostním korpusu (PDT)*

ÚFAL/CKL TR-2004-23 Jan Hajič a kol., *Anotace na analytické rovině, návod pro anotátory*

ÚFAL/CKL TR-2004-24 Jan Hajič, Zdeňka Urešová, Alevtina Bémová, Marie Kaplanová, *Anotace na tektogramatické rovině (úroveň 3)*

ÚFAL/CKL TR-2004-25 Jan Hajič, Zdeňka Urešová, Alevtina Bémová, Marie Kaplanová, *The Prague Dependency Treebank, Annotation on tectogrammatical level*

ÚFAL/CKL TR-2004-26 Martin Holub, Jiří Diviš, Jan Pávek, Pavel Pecina, Jiří Semecký, *Topics of Texts. Annotation, Automatic Searching and Indexing*

ÚFAL/CKL TR-2005-27 Jiří Hana, Daniel Zeman, *Manual for Morphological Annotation (Revision for PDT 2.0)*

ÚFAL/CKL TR-2005-28 Marie Mikulová a kol., *Pražský závislostní korpus (The Prague Dependency Treebank) Anotace na tektogramatické rovině (úroveň 3)*

ÚFAL/CKL TR-2005-29 Petr Pajas, Jan Štěpánek, *A Generic XML-Based Format for Structured Linguistic Annotation and Its application to the Prague Dependency Treebank 2.0*

ÚFAL/CKL TR-2006-30 Marie Mikulová, Alevtina Bémová, Jan Hajič, Eva Hajičová, Jiří Havelka, Veronika Kolařová, Lucie Kučová, Markéta Lopatková, Petr Pajas, Jarmila Panevová, Magda Razímová, Petr Sgall, Jan Štěpánek, Zdeňka Urešová, Kateřina Veselá, Zdeněk Žabokrtský, *Annotation on the tectogrammatical level in the Prague Dependency Treebank (Annotation manual)*

ÚFAL/CKL TR-2006-31 Marie Mikulová, Alevtina Bémová, Jan Hajič, Eva Hajičová, Jiří Havelka, Veronika Kolařová, Lucie Kučová, Markéta Lopatková, Petr Pajas, Jarmila Panevová, Petr Sgall, Magda Ševčíková, Jan Štěpánek, Zdeňka Urešová, Kateřina Veselá, Zdeněk Žabokrtský, *Anotace na tektogramatické rovině Pražského závislostního korpusu (Referenční příručka)*

ÚFAL/CKL TR-2006-32 Marie Mikulová, Alevtina Bémová, Jan Hajič, Eva Hajičová, Jiří Havelka, Veronika Kolařová, Lucie Kučová, Markéta Lopatková, Petr Pajas, Jarmila Panevová, Petr Sgall,Magda Ševčíková, Jan Štěpánek, Zdeňka Urešová, Kateřina Veselá, Zdeněk Žabokrtský, *Annotation on the tectogrammatical level in the Prague Dependency Treebank (Reference book)*

ÚFAL/CKL TR-2006-33 Jan Hajič, Marie Mikulová, Martina Otradovcová, Petr Pajas, Petr Podveský, Zdeňka Urešová, *Pražský závislostní korpus mluvené češtiny. Rekonstrukce standardizovaného textu z mluvené řeči*

ÚFAL/CKL TR-2006-34 Markéta Lopatková, Zdeněk Žabokrtský, Václava Benešová (in cooperation with Karolína Skwarska, Klára Hrstková, Michaela Nová, Eduard Bejček, Miroslav Tichý) *Valency Lexicon of Czech Verbs. VALLEX 2.0*

ÚFAL/CKL TR-2006-35 Silvie Cinková, Jan Hajič, Marie Mikulová, Lucie Mladová, Anja Nedolužko, Petr Pajas, Jarmila Panevová, Jiří Semecký, Jana Šindlerová, Josef Toman, Zdeňka Urešová, Zdeněk Žabokrtský, *Annotation of English on the tectogrammatical level*