ACCURAT

Analysis and Evaluation of Comparable Corpora
for Under Resourced Areas of Machine Translation

www.accurat-project.eu

Project no. 248347

Deliverable D2.6

Toolkit for multi-level alignment and information extraction from comparable corpora

Version No. 2.1

31/03/2012

Document Information

Deliverable number:	D2.6
Deliverable title:	Toolkit for multi-level alignment and information extraction from comparable corpora
Due date of deliverable:	31/08/2011
Actual submission date of deliverable:	31/03/2012
Main Author(s):	Radu Ion, Mārcis Pinnis, Dan Ştefănescu, Ahmet Aker, Monica Paramita, Fangzhong Su, Elena Irimia, Xiaojun Zhang, Nikola Ljubešić
Participants:	Radu Ion, Mārcis Pinnis, Dan Ştefănescu, Ahmet Aker, Monica Paramita, Fangzhong Su, Elena Irimia, Xiaojun Zhang, Nikola Ljubešić
Internal reviewer:	Tilde
Workpackage:	WP2
Workpackage title:	Multi-level alignment methods and information extraction from comparable corpora
Workpackage leader:	RACAI
Dissemination Level:	**PU**
Version:	V2.1
Keywords:	Toolkit, document alignment, phrase alignment, comparable corpora, named entity recognition and alignment, terminology extraction and alignment

History of Versions

Version	Date	Status	Name of the Author (Partner)	Contributions	Description/ Approval Level
V0.1	01/08/2011	Draft	RACAI	All partners	Initial compilation
V0.2	02/08/2011	Draft	RACAI	RACAI	Added RACAI contributions
V0.3	03/08/2011	Draft	RACAI	RACAI	Added some more RACAI contributions
V0.4	04/08/2011	Draft	RACAI	RACAI, USFD	Added some modifications requested by USFD and even some more RACAI tools
V0.5	05/08/2011	Draft	RACAI	RACAI	Added Intro and a section about pipelines

Version	Date	Status	Name of the Author (Partner)	Contributions	Description/ Approval Level
V0.6	08/08/2011	Draft	TILDE	TILDE	Spell-checking and re-formatting of sections 1-5.
V0.7	10/08/2011	Draft	TILDE	TILDE, DFKI, USFD	Integrated and spell-checked DFKI toolkit for multi-level parallel phrase extraction. Updated AlignerUSFD section
V0.8	16/08/2011	Draft	TILDE	TILDE	Updated Introduction, improved General Use Case Section and TildeNER section.
V0.9	23/08/2011	Draft	TILDE	TILDE, USFD	Added KEATEWrapper and OpenNLPWrapper section, updated MapperUSFD, TildeNER and Tilde's wrapper system's for CollTerm section.
V0.10	23/08/2011	Draft	RACAI	RACAI	Modified PEXACC documentation
V0.11	23/08/2011	Draft	TILDE	TILDE, USFD, FFZG	Updated MapperUSFD section and the Tilde's wrapper system's for CollTerm section.
V0.12	24/08/2011	Draft	TILDE	TILDE	Changed document structure
V0.13	24/08/2011	Draft	RACAI	RACAI, DFKI, USFD	Changed EMACC documentation. Added modifications to all other documentations in Section 2 of this document. Added an executive summary.
V0.14	25/08/2011	Draft	RACAI	CTS, RACAI	Added documentation from CTS and documentation for RACAI NERA1 and TE.
V0.15	26/08/2011	Draft	RACAI	RACAI, CTS	Added information about parallel data extraction workflow and changed ComMetric I/O section.
V0.16	29/08/2011	Draft	RACAI	RACAI	Added documentation about NERA2 and RACAI Terminology Aligner applications. Also added conclusions.

Version	Date	Status	Name of the Author (Partner)	Contributions	Description/ Approval Level
V0.17	29/08/2011	Draft	RACAI	RACAI, FFZG	Modified the documentation for parallel data extraction workflow. Updated documentation from FFZG (CollEX).
V0.18	30/08/2011	Draft	RACAI	TILDE	Added the documentation of the NE/Term mapping workflow from Tilde and included the Document Aligner documentation..
V0.19	31/08/2011	Final	RACAI	RACAI	Re-written the Introduction at Tilde's suggestion and added a tool table summary. Final checks.
V1.0	31/08/2011	Final	TILDE	TILDE	Submitted to EC
V1.01	10/10/2011		TILDE	TILDE	NERTEWF, TildeNER and Tilde's Wrapper System's for CollEx updates for the second release of the Toolkit.
V1.02	15/12/2011		TILDE	TILDE, FFZG	Replaced CollEx with CollTerm from FFZG and updated Tilde's Wrapper System for CollEx to Tilde's Wrapper System for CollTerm
V1.03	16/12/2011		TILDE	TILDE, DFKI	Replaced DFKI's ME parallel data extractor.
V1.04	21/12/2011		TILDE	TILDE, LT	Added a new tool from LT.
V1.05	21/12/2011		RACAI	RACAI	Updated EMACC/PEXACC and validated all other RACAI documentation.
V1.06	21/12/2011		TILDE	TILDE	Updated version of the NE/term mapping workflow.
V1.07	22/12/2011		RACAI	RACAI	Added "Changes from previous version" to all RACAI tools. Checked the description of the parallel data workflow in light of new developments.

Version	Date	Status	Name of the Author (Partner)	Contributions	Description/ Approval Level
V1.08	28/12/2011		TILDE	TILDE, CTS	Updated ComMetric and DicMetric sections.
V2.0	30/12/2011		TILDE	TILDE	Final formatting for the second version of the deliverable.
V2.1	15/03/2012		RACAI	RACAI	Added a diagram showing how the tools are used (reviewer comment no. 1 for D2.2 wrongly put there)

EXECUTIVE SUMMARY

This document contains technical documentation of all important tools that have been currently developed within the ACCURAT project. By using them, the users may expect to obtain parallel texts, parallel terminology, general translation lexicons and translated named entities, all of which are useful as training data/resources for either SMT or Example-bases/Rule-based MT. The documentation will help the interested user to install and run the applications individually or in the provided workflows: "parallel data mining from comparable corpora" and "named entities/terminology extraction and mapping from comparable corpora". Considerable efforts have been put into making this documentation accessible to the user with average computer skills and in implementing the tools' interfaces so that easy integration with future tools is ensured, together with facile tool manipulation.

The ACCURAT Toolkit is stored at the ACCURAT repository and is freely available after completing the registration form (http://www.accurat-project.eu/index.php?p=toolkit).

Table of Contents

Abbreviations

Table 1 Abbreviations used throughout this document.

Abbreviation	Term/definition
API	Application Programming Interface
MENER	Maximum Entropy Named Entity Recognizer
MSD	Morpho-Syntactic Descriptor
MT	Machine Translation
NE	Named Entity/Named entity Extraction
NER	Named Entity Recognition
NERC	Named Entity Recognition and Classification
NFS	Network File System
NLP	Natural Language Processing
NP	Noun Phrase
POS	Part of Speech
TE	Terminology Extraction
TF/IDF	Term Frequency/Inverse Document Frequency
SMT	Statistical Machine Translation
SSH	Secure Shell
WP	Work Package

Summary of tools and workflows in the ACCURAT Toolkit

Table 2 Summary of tools and workflows in the ACCURAT Toolkit.

Tool Name	Operation Type	Developed by	Contact
Parallel data mining	Workflow	RACAI	radu@racai.ro
NE/TE recognition and mapping	Workflow	Tilde	marcis.pinnis@tilde.lv
ComMetric	Document aligner	CTS	f.su@leeds.ac.uk
Features extractor and document pair classifier	Document aligner	USFD	m.paramita@shef.ac.uk
EMACC	Document aligner	RACAI	radu@racai.ro
PEXACC	Parallel textual unit extractor	RACAI	radu@racai.ro
ME Parallel Sentence Extractor	Parallel textual unit extractor	DFKI	xiaojun.zhang@dfki.de
TildeNER	NE recognizer	Tilde	marcis.pinnis@tilde.lv
OpenNLP wrapper	NE recognizer	USFD	a.aker@dcs.shef.ac.uk
NERA1	NE recognizer	RACAI	danstef@racai.ro
Tilde's wrapper for CollTerm	Terminology extractor	Tilde	marcis.pinnis@tilde.lv
KEA wrapper	Terminology extractor	USFD	a.aker@dcs.shef.ac.uk
CollTerm	Terminology extractor	FFZG	nljubesi@gmail.com
TE for English and Romanian	Terminology extractor	RACAI	danstef@racai.ro
Multi-lingual named entity and terminology mapper	NE mapper, Term mapper	USFD	a.aker@dcs.shef.ac.uk
NERA2	NE mapper	RACAI	danstef@racai.ro
Language independent terminology aligner	Term mapper	RACAI	danstef@racai.ro
P2G: A tool to extract term candidates from aligned phrases	Term mapper	LT	g.thurmair@linguatec.de
Google and Bing Translation Interface	MT system	CTS	f.su@leeds.ac.uk
DEACC	Dictionary extractor	RACAI	elena@racai.ro

Summary of updated tools and workflows in the ACCURAT Toolkit from the previous version

Table 3 Updated tools from the previous version of the ACCURAT Toolkit

Tool Name	Operation Type	Developed by	Contact
NE/TE recognition and mapping	Workflow	Tilde	marcis.pinnis@tilde.lv
EMACC	Document aligner	RACAI	radu@racai.ro
PEXACC	Parallel textual unit extractor	RACAI	radu@racai.ro
ME Parallel Sentence Extractor	Parallel textual unit extractor	DFKI	xiaojun.zhang@dfki.de
TildeNER	NE recognizer	Tilde	marcis.pinnis@tilde.lv
OpenNLP wrapper	NE recognizer for English, sentence splitting	USFD	a.aker@dcs.shef.ac.uk
NERA1	NE recognizer	RACAI	danstef@racai.ro
Tilde’s wrapper for CollTerm	Terminology extractor	Tilde	marcis.pinnis@tilde.lv
CollTerm	Terminology extractor	FFZG	nljubesi@gmail.com
TE for English and Romanian	Terminology extractor	RACAI	danstef@racai.ro
Multi-lingual named entity and terminology mapper	NE mapper, Term mapper	USFD	a.aker@dcs.shef.ac.uk
Language independent terminology aligner	Term mapper	RACAI	danstef@racai.ro
P2G: A tool to extract term candidates from aligned phrases	Term mapper	LT	g.thurmair@linguatec.de

Introduction

Lack of sufficient linguistic resources for many languages and domains currently is one of the major obstacles in further advancement of automated translation. The main goal of the ACCURAT project is to research methods and techniques to overcome this obstacle by finding, analyzing and evaluating novel methods that exploit comparable corpora to generate training data/resources for either SMT or Rule/Example-based MT.

This document describes a collection of software tools, developed within the ACCURAT project according to the ACCURAT project's methodology that will contribute towards achieving the goals of the ACCURAT project. These tools (which will be collectively referred to as the "ACCURAT Toolkit") produce different types of data extracted from comparable corpora that are useful to Statistical and Rule/Example-based Machine Translation. We plan to improve/adapt the current versions of these tools as the project progresses.

The types of MT-useful data that the ACCURAT Toolkit produces can be classified along these lines (the Toolkit's tools that implement the respective operations are :

- **translation dictionaries** extracted from comparable corpora; these dictionaries are expected to supplement existing translation lexicons which are useful to both statistical and rule/example-based MT. The tool that implements this operation is *DEACC* (see section 6.2 of this document);
- **translated terminology** extracted (mapped) from comparable corpora; this type of data is presented in a dictionary-like format and is expected to improve domain-dependent translation (please refer to the ACCURAT Deliverable D2.3 "Report on information extraction from comparable corpora" for more information on methods). Tools that implement this operation are: *the multi-lingual named entity and terminology mapper* (section 5.1) and *the language independent terminology aligner* (section 5.3),
- **translated named entities** extracted (mapped) from comparable corpora; also presented in a dictionary-like format, these lexicons are expected to improve the parallel phrase extraction algorithms from comparable corpora and be useful by themselves when actually used in translation (the problem of named entity mapping is not trivial to solve since named entities may be transliterated and/or actually translated either word by word or as idioms; please refer to the ACCURAT Deliverable D2.3 "Report on information extraction from comparable corpora" for more information on methods). Tools that implement this operation are: *the multi-lingual named entity and terminology mapper* (section 5.1) and *NERA2* (section 5.2);
- **comparable document (and other textual unit types) alignment** that will facilitate the task of parallel phrase extraction by massively reducing the search space of such algorithms (please refer to the ACCURAT Deliverable D2.2 for more information on methods). Tools available for completing this operation are: *EMACC* (section 2.4), *ComMetric* (section 2.1) and *the feature-based document pair classifier* (section 2.3);

- **parallel sentence/phrase mapping** from comparable corpora which aims at supplying clean parallel data useful for statistical translation model learning (please refer to the ACCURAT Deliverable D2.2 “Report on multi-level alignment of comparable corpora” for more information on methods). Existing tools for this operation are: *PEXACC* (section 2.5) and *the ME parallel sentence extractor* (section 2.6).

In order to map terms and named entities bilingually, the ACCURAT Toolkit also provides tools for detecting and annotating these types of expressions in a monolingual fashion. Thus, the toolkit also contains:

- three NER applications: *NERA1* (section 3.3), *the OpenNLP NER Wrapper* (section 3.2) and the *TildeNER* tool (section 3.1);
- three terminology extraction applications: *the KEA TE Wrapper* (section 4.2), *the Tilde’s wrapper system for CollTerm* (see section 4.1) and *TE for English and Romanian* (section 4.4).

The purpose of this document is twofold:

1. to accurately describe the running environment, the system requirements, the external dependencies and the setup of each individual tool so that a computer knowledgeable user is able to install and run the tool;
2. to specify general use case scenarios (pipeline schemes) with which, the computer knowledgeable user is able to obtain data of one of the previously mentioned types.

It is important to note that this document is intended to be useful to the rather advanced computer user who generally knows how to install different applications in both Windows and Linux environments and how to operate with command line tools. While the Windows installations are generally automated, Linux installations sometimes require knowledge of C/C++ compilation and tweaking. For tools requiring cluster operations, cluster elements need to be installed by hand (e.g. adding user accounts, installing NFS servers and clients, mounting NFS drives, installing password-less SSH connections, etc.).

The document at first defines the general use case scenarios (see section 1) and tools required to run each of the workflows and then (sections 2.1 and further) describes each separate tool in more detailed levels. In order for the user to successfully execute the general use case scenarios, it is important to follow installation instructions of each separate tool and update the workflow property files according to the installation paths of the user’s local system.

The tools described in this deliverable depend also on third party intellectual properties (tools developed by other parties). All dependencies and usage restrictions of third party tools are defined in the ACCURAT Deliverable D6.7 (Report on IPR of the project results). Every user must acknowledge the restrictions and make sure that no third party IPR are violated.

1 General Use Case Scenarios

The toolkit provides in total two General Use Case Scenarios, which means that the toolkit is compliant with its provided tools within the two pre-defined workflows. If the user requires the toolkit to operate in other ways than the pre-defined use cases, he/she must be proficient enough to work with each individual tool separately and, if necessary, also create integration scripts between separate tools if the user's desired use case is not covered by the supported workflows.

1.1 The "Parallel sentence/phrase mapping" workflow

1.1.1 Overview and purpose of the workflow

This workflow aims at providing parallel textual unit mining (sentences and/or phrases) from comparable corpora. The assumption that we have worked with is that, given two collections of source and target documents, these documents need to first be aligned as to their probability of containing parallel textual units so that the parallel textual unit extractors (CPU intensive algorithms) do not have to search in each possible document pair. We think that by doing the parallel data mining this way, we minimize the execution time and we also do not miss many parallel pairs that could be found in pairs of documents not in our alignment list.

After the document alignment has been found, a generic parallel textual unit extractor can search for parallel pairs only in the offered document pairs.

This toolkit contains three applications that implement the "generic document aligner/document pair classifier" operation:

- *EMACC* (section 2.4) which outputs a list of document pairs, each with its alignment (logarithmic) probability;
- *ComMetric* (section 2.1) which also outputs a list of document pairs along with comparability scores;
- *Feature-based document pair classifier* (section 2.3) which outputs a list of document pairs, each with its detected comparability level: "parallel", "strongly comparable", "weakly comparable" and "not related".

and two applications that take over the role of a "parallel textual unit extractor" operation:

- *PEXACC* (section 2.5) which takes the output of any of the previous applications and outputs a list of parallel sentences or phrases (depending on the configuration);
- *MaxEnt Extract* (section 2.6) which also takes the output of any of the previous applications and outputs a list of parallel sentences.

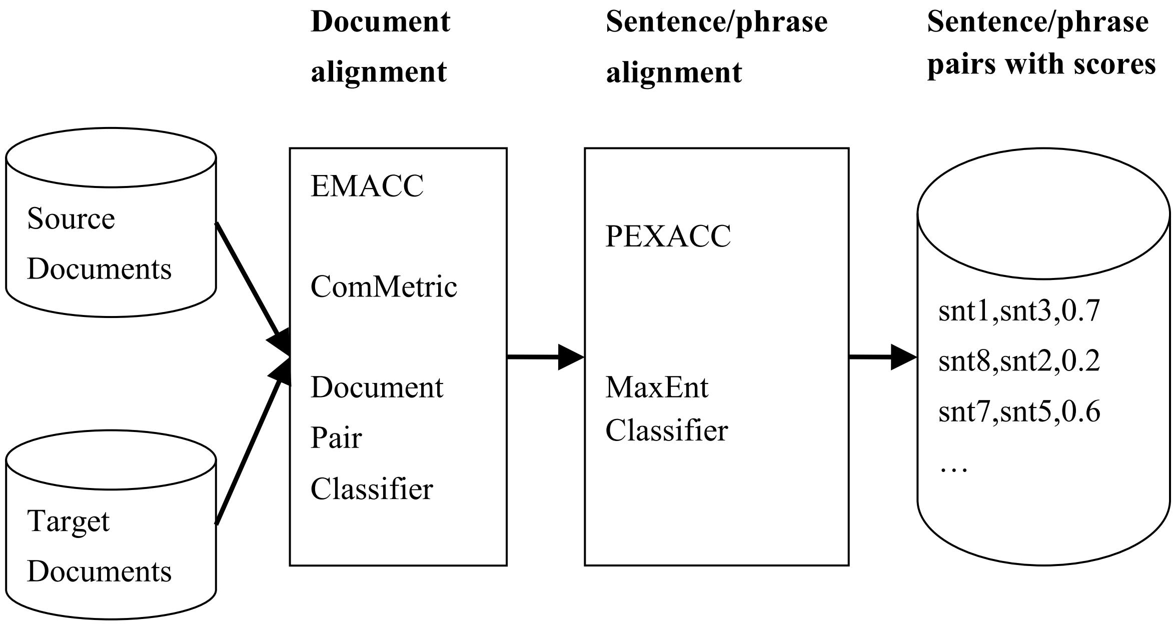


Figure 1: Graphical overview of the "Parallel sentence/phrase mapping" workflow

1.1.2 Changes from the previous version

There are no changes from the previous version.

1.1.3 Software dependencies and system requirements

The parallel data mining workflow is provided as a self-contained kit prepared for running on *Windows* 32-bit platforms (it will not run on *Linux* or on 64-bit *Windows* platforms due to several required *C++* applications/DLLs that have been compiled on a *Windows XP Professional* machine under *Cygwin* and that belong to this kit).

In order to be able to run the main application of the workflow, "*ParallelDataMining.pl*", one must be sure that *Java*[1] and *Perl*[2] are installed and that the paths of the executables are present in the system's environment variable "*PATH*" (thus, "*echo %PATH%*" should contain the directories in which "*perl*" and "*java*" executables are to be found).

1.1.4 Installation

No other installation is necessary other than *Perl* and *Java* as mentioned in the previous section.

[1] Download it from http://www.oracle.com/technetwork/java/javase/downloads/index.html

[2] Download it from http://www.activestate.com/activeperl/downloads

1.1.5 Execution instructions

The application that implements this workflow is called “*ParallelDataMining.pl*” and it has the following usage:

```
Usage:perl ParallelDataMining.pl
        --source <language> --target <language>
        --param CONFIG=path\to\config.prop
        --param DOCALIGN=<commetric|emacc|featclass>
        --param PHRMAP=<pexacc|meextract>
        --input <path to source documents file>
        --input <path to target documents file>
        --output <path to the extracted textual units file>
```

where the command line switches have the following meanings:

- *“--source”* and “*--target”* specify the language of the source documents and the language of the target documents respectively (may be given in full name or as 2 or 3 letter codes);
- *“--param CONFIG”* specifies the name of the custom property file. If it is not given, the program will read the file “*ParallelDataMining.prop*” from the same directory. This file contains configuration specific options for the tools involved in the workflow (see the respective sections for details). If the user desires to customize the workflow, this is the file to be modified;
- “*--param DOCALIGN*” specifies the application that will perform document alignment. The user may choose between *EMACC* (“*emacc*”), *ComMetric* (“*commetric*”) or the *Feature-based document pair classifier* (“*featclass*”);
- “*--param PHRMAP*” specifies the application which will handle the parallel textual unit extraction. The user may choose between *PEXACC* (“*pexacc*”) or *MaxEnt Extract* (“*meextract*”);
- “*--input*” (both of them) specify the source and target document lists. The format of the input files is the format of the input files *EMACC*, *ComMetric* or the *Feature-based Document Pair Classifier* accept (see the respective sections);
- “*--output*” specifies the name of the output file. The format of the output file is the same as the format used by *EMACC*, *ComMetric* or the *Feature-based Document Pair Classifier* (see the respective sections).

Suppose that one would like to mine for parallel phrases from a comparable English-Romanian corpus. The English (source) documents are listed in the “*doclist-en.txt*” file and the Romanian (target) documents are listed in the “*doclist-ro.txt*”. Furthermore, suppose that the user wants to use *ComMetric* for the document alignment task and *PEXACC* for the parallel phrase extraction task. Then, using “*ParallelDataMining.pl*”, the required command is:

```
perl ParallelDataMining.pl \
  --source en --target ro \
  --param DOCALIGN=commetric --param PHRMAP=pexacc \
  --input doclist-en.txt --input doclist-ro.txt \
  --output result.txt
```

1.1.6 Input/Output data formats

The workflow will take as input lists of documents (in source and target languages) and will produce at output a file containing the parallel pairs of textual units (sentences or phrases) that have been extracted from the comparable corpus.

All the applications that are included in this workflow follow certain I/O data conventions:

- the I/O data for the "generic document aligner" (*EMACC*, *ComMetric* and the *Feature-based Document Pair Classifier* all implement the "generic document aligner") may be sampled by observing the I/O data of *EMACC* (see section 2.4.6);
- the I/O data for the "generic parallel textual unit extractor" (*PEXACC* and the *MaxEnt Extract* implement the "generic parallel textual unit extractor") may be sampled by observing the I/O data of *PEXACC* (see section 2.5.6).

Consequently, the input data for the workflow is the input data for the generic document aligner and the output data of the workflow is the output data of the generic parallel textual unit extractor.

1.1.7 Integration with external tools

Other tools may be added to this workflow if they implement, in a compatible manner, the "document alignment" or the "parallel textual unit extraction" operations. As long as the new additions respect the format of I/O data presented in the previous section, they can be incorporated into the workflow by the Perl-programmer user (that will need to edit to file "*ParallelDataMining.pl*").

1.2 The "Named entity and term mapping" workflow

1.2.1 Overview and purpose of the workflow

The *Named entity and term mapping workflow* (*NERTEWF*) provides means for multi-lingual named entity or term mapping as well as named entity recognition and term extraction using tools developed within the ACCURAT project as well as tools integrated within the workflow, but not developed within the ACCURAT project (for instance, *OpenNLP* named entity recognition for English).

The workflow provides three different processing methods:

- Named entity extraction and/or named entity mapping using bilingual comparable corpora (method "NE");
- Terminology extraction and/or term mapping using bilingual comparable corpora (method "T");
- Term mapping using parallel data (as produced by *PEXACC*; see section 2.5; method "PT").

In the first two methods the workflow is built on the assumption that the user has collected multi-lingual comparable corpora and has executed the document alignment operation by using the *Document Alignment wrapper* (see section 1.2.5), that is, a requirement of these methods is the comparable document pair list file that specifies comparability between two document pairs.

In the third method the workflow is built on the assumption that the user has executed the Parallel sentence/phrase mapping workflow (see section 1.1), that is, a requirement of this workflow is the parallel data file (see section 2.5.6 for a format description).

The workflow has the following named entity recognition tools integrated:

- *TildeNER* (see section3.1);
- *OpenNLP wrapper* (see section 3.2);
- *NERA1: Named Entity Recognition for English and Romanian* (see section 3.3).

The workflow has the following terminology extraction tools integrated:

- *Tilde's wrapper system for CollTerm* (see section 4.1);
- *KEA wrapper* (see section 4.2);
- *Terminology Extraction for English and Romanian* (see section 4.4).

The workflow has the following mapping tools integrated:

- *Multi-lingual named entity and terminology mapper* (see section 5.1);
- *NERA2: Language Independent Named Entity Mapping* (see section 5.2);
- *A language independent terminology aligner* (see section 5.3);
- *P2G: A tool to extract term candidates from aligned phrases* (see section 5.4).

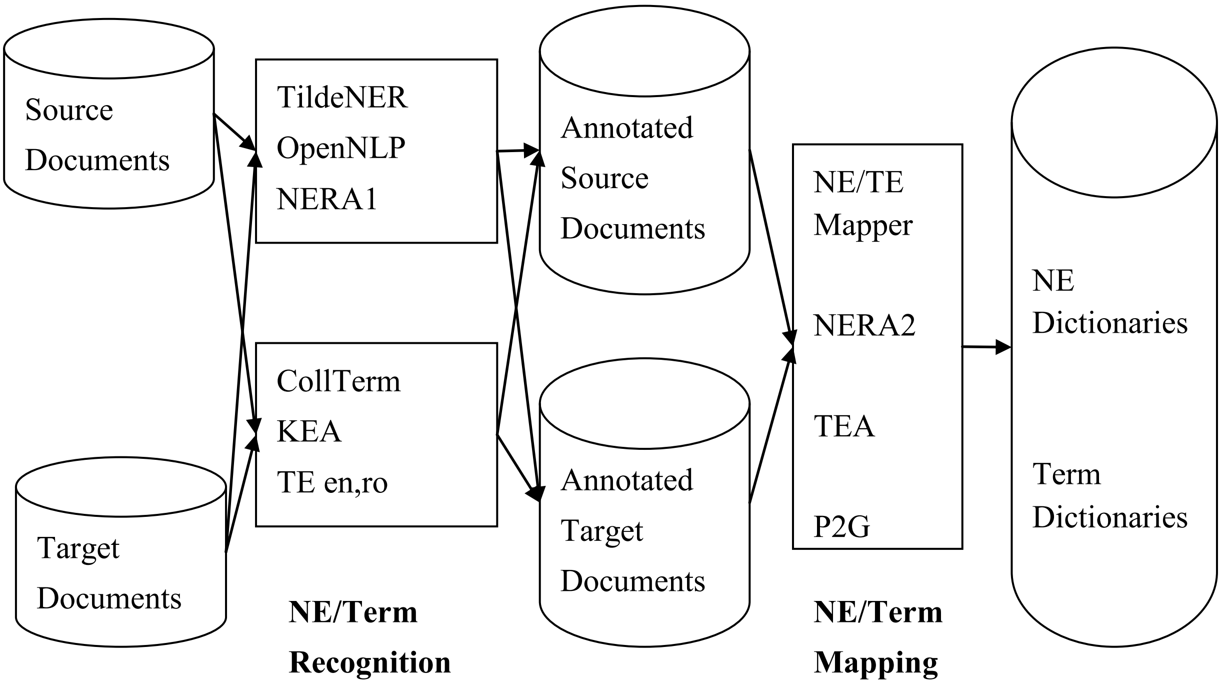


Figure 2: Graphical overview of the "Named entity and term mapping" workflow

1.2.2 Changes from previous version

The second version of the Named entity and term mapping workflow includes updated versions of all tools that have been updated. The workflow now includes also a third method – term mapping using parallel data extracted by *PEXACC*, which is done by the tool *P2G* (see section 5.4 for more details). The updated version also contains "*easy to use*" testing scripts ("*RUN*" scripts), which allow testing whether all parts of the workflow work on the user's local system.

1.2.3 Software dependencies and system requirements

The workflow contains internal tool dependencies (for instance, "*Tagger.exe*" for Latvian and Lithuanian POS-tagging in *TildeNER* and *Tilde's wrapper system for CollTerm*), therefore, the only software dependencies are runtime environments:

- *Java Runtime Environment* (version 1.6.0);
- *Perl* (Windows - *Strawberry Perl* v5.12.1; *Linux – Perl* v5.10.1);
- *.Net Framework 4.0 (Windows), Mono 2.10* (*Linux*);
- *Python* (*Windows – Python* v2.7.1; *Linux – Python* v2.6.5).

The user must be sure that *Java*, *Perl*, *Python* and the *.NET Framework* (or *Mono* on *Linux*) are installed and that the executable paths are present in the system's environment variable "*PATH*" (thus, the string returned by "*echo %PATH%*" should contain the directories in which "*perl*", "*java*" and "*python*" executables are to be found).

The workflow is built as a tagging and extraction workflow and does not involve system training; therefore the system requirements may be lower than specified for separate tools, which also provide training options:

- A *Linux* or *Windows* (*XP* or newer) operating system;
- 1GB or more RAM (for the third method, the Java heap size should be larger than 512MB);
- Intel® Pentium® 4 CPU 3.00GHz, 2992 Mhz, 1 Core(s), 2 Logical Processors or faster.

1.2.4 Installation

The workflow requires no installation. Simply extract the "*D2_6_Section_1_2_NERTEWF.zip*" contents in a directory where the user has read/write/execute permissions and run the workflow as specified further.

The workflow currently supports only Latvian, Lithuanian, English and Romanian language named entity recognition and term extraction in the first two methods and German and English term mapping in the third method. To add support for other language named entity recognition, the user has to integrate his tool in the workflow following the guidelines in section 1.2.7.

The *Multi-lingual named entity and terminology mapper* supports all language pair NE/Term mapping, but the *NERA2* and the *Language independent terminology aligner* tools currently support only "*EN-RO*" pair named entity mapping. For additional support the user requires

GIZA++ translation lexicons in the form "*[SRCL]_[TRGL]*" placed in the "*RACAI_NERA2*" and also in the "*RACAI_TA*" directories. "*[SRCL]*" is the source document language code defined with two lowercase characters (for instance, "*lv*", "*lt*", "*en*", "*ro*", etc.) and the "*[TRGL]*" is the target document language code defined with two lowercase characters.

1.2.5 Execution instructions

As mentioned in the overview, the workflow requires a document pair list file. To obtain the document pairing, the user may employ the *DocumentAligner wrapper* that is offered together with this workflow. A second alternative would be for the user to generate (by some other means) his/her own document pair list file.

The *DocumentAligner wrapper* is a *Perl* application that will standardize the calling interface of all document aligner applications present in this toolkit into the interface of a "generic document aligner" tool. The applications that are included in this wrapper are: *EMACC* (section 2.4), *Feature-based Document Pair Classifier* (section 2.3) and *ComMetric* (section 2.1).

The *DocumentAligner wrapper* is implemented by the Perl script "*DocumentAligner.pl*".

The usage of this script is as follows:

```
Usage: DocumentAlignment.pl
    --source <language> --target <language>
    --param CONFIG=path\to\config.prop
    --param DOCALIGN=<commetric|emacc|featclass>
    --input <path to source documents file>
    --input <path to target documents file>
    --output <path to the aligned documents file>
```

where

- "*--source*" and "*--target*" specify the language of the source documents and the language of the target documents respectively (may be given in full name or as 2 or 3 letter codes);
- "*--param CONFIG*" specifies the name of the custom property file. If it is not given, the program will read the file "*DocumentAligner.prop*" from the same directory. This file contains configuration specific options for the tools involved in the wrapper (see the respective sections for details);
- „*--param DOCALIGN*" specifies the application that will perform document alignment. The user may choose between *EMACC* ("*emacc*"), *ComMetric* ("*commetric*") or the Feature-based document pair classifier ("*featclass*");
- "*--input*" (both of them) specify the source and target document lists;
- "*--output*" specifies the name of the output file.

The format of the Input/Output data of the wrapper is consistent with the format of the Input/Output data for all the applications that the wrapper encapsulates. For an example, please take a look at section 2.4.6 of the *EMACC* document aligner.

To execute named entity or terminology mapping on all document pairs from a given aligned document pair list the user has to execute the following command line:

```
perl EntityMappingWorkflow.pl --source [Source Language] --target [Target
Language] --param "propFile=[Property File Path]" --param method=[Mapping
and Tagging Method NE|T|PT] --param parsedSource=[Source Parsed? 0|1] --
param parsedTarget=[Target Parsed? 0|1] --param skipMapping=[Skip Mapping?
0|1] --input [Document Pair List File Path] --output [Mapped NE/Term File
Path]
```

The script requires the following parameters in any order (the elements in brackets “[…]”):

- “*--source [Source Language]*” – the source (first corpus) document language (for instance, “LV”, “LT”, “EN”, etc.). This parameter is **mandatory**.
- “*--target [Target Language]*” – the target (second corpus) document language (for instance, “LV”, “LT”, “EN”, etc.). This parameter is **mandatory**.
- “*--param "propFile=[Property File Path]"*” – the path to the workflow property file. The format is described in section 1.2.6.1. This parameter is **optional** and if not given the default property file („*NE-TermWorkflowProperties.prop*”) will be used.
- “*--param method=[Mapping and Tagging Method NE|T|PT]*” – the tagging and mapping method. For NE tagging and mapping use the value “*NE*”. For term tagging and mapping use the value “*T*”. For term mapping using a parallel data file use the value “*PT*”. This parameter is **mandatory**.
- “*--param parsedSource=[Source Parsed? 0|1]*” – specifies whether the source language documents are already tagged for named entities or terms according to the method (“*0*” – are not tagged; “*1*” – are tagged). This parameter is **optional** and if not given a default value “*0*” will be used. The parameter is not used if the method “PT” is used.
- “*--param parsedTarget=[Target Parsed? 0|1]*” – specifies whether the target language documents are already tagged for named entities or terms according to the method (“*0*” – are not tagged; “*1*” – are tagged). This parameter is **optional** and if not given a default value “*0*” will be used. The parameter is not used if the method “*PT*” is used.
- “*--param skipMapping=[Skip Mapping? 0|1]*” – specifies whether mapping should be skipped after tagging (“*0*” – should be executed; “*1*” – should be skipped). This parameter is **optional** and if not given a default value “*0*” will be used. The parameter is not used if the method “*PT*” is used.
- “*--input [Document Pair List File Path]*” – the path to the aligned document pair list file (if the method is either „*NE*” or „*T*”). The format is described in section 1.2.6.2. If the method “*PT*” is used, the input file should be the parallel sentence/phrase file (see section 2.5.6 for a format description). This parameter is **mandatory**.

- "*--output [Mapped NE/Term File Path]*" – the file path to the file where mapping results should be saved (if mapping will be executed). The format is described in section 1.2.6.4. This parameter is **mandatory**.

An example execution call sequence is as follows:

```
perl "C:\RuntimeTempDir\NERTEWF\EntityMappingWorkflow.pl" --source EN --
target LV --param "propFile=C:\RuntimeTempDir\NERTEWF\NE-
TermWorkflowProperties.prop" --param method=NE --param parsedSource=0 --
param parsedTarget=0 --input "C:\RuntimeTempDir\NERTEWF_TEMP\EN_LV.txt" --
output "C:\RuntimeTempDir\NERTEWF_TEMP\EN_LV_USFD_NE_OUT.txt"
```

An example of a call with the least arguments is as follows:

```
perl "C:\RuntimeTempDir\NERTEWF\EntityMappingWorkflow.pl" --source EN --
target LV --param method=NE --input
"C:\RuntimeTempDir\NERTEWF_TEMP\EN_LV.txt" --output
"C:\RuntimeTempDir\NERTEWF_TEMP\EN_LV_USFD_NE_OUT.txt"
```

In order to provide assistance in execution of the scripts the NE/Term mapping workflow package contains predefined *Bash* ("*sh*"; for *Linux*) and *Batch* ("*bat*"; for *Windows*) scripts in the form "*RUN_###.bat*" or "*RUN_###.sh*". As the possible execution scenarios depend on the user's requirements, the scripts provide functionality for only a limited number of use cases. Input data is taken from the "*TEST*" subdirectory and individual tool resources (models, property files, etc.) are taken from the corresponding tool subdirectories in the *NERTEWF* package. All output data is saved to the "*TEST*" directory.

The provided scripts are as follows:

- **NE-tagging and NE mapping of plaintext documents:**
  - For the English – Lithuanian document pairs:
    - *RUN_EN-LT_Plaintext_NE_Mapping.bat (Windows)*
    - *RUN_EN-LT_Plaintext_NE_Mapping.sh (Linux)*
    - The input document pair list file is taken from „*./TEST/en_lt_plain_pairs_in.txt*" and the mapped NE pairs are saved in „*./TEST/en_lt_NE_pairs_out.txt*".
  - For the English – Latvian document pairs:
    - *RUN_EN-LV_Plaintext_NE_Mapping.bat (Windows)*
    - *RUN_EN-LV_Plaintext_NE_Mapping.sh (Linux)*
    - The input document pair list file is taken from „*./TEST/en_lv_plain_pairs_in.txt*" and the mapped NE pairs are saved in „*./TEST/en_lv_NE_pairs_out.txt*".
  - For the English – Romanian document pairs (mapping is done with *MapperUSFD*):
    - *RUN_EN-RO_Plaintext_NE_Mapping.bat (Windows)*
    - *RUN_EN-RO_Plaintext_NE_Mapping.sh (Linux)*
    - The input document pair list file is taken from „*./TEST/en_ro_plain_pairs_in.txt*" and the mapped NE pairs are saved in „*./TEST/en_ro_NE_pairs_out.txt*".
- **Term-tagging and term mapping of plaintext documents:**
  - For the English – Lithuanian document pairs:
    - *RUN_EN-LT_Plaintext_T_Mapping.bat (Windows)*
    - *RUN_EN-LT_Plaintext_T_Mapping.sh (Linux)*

    - The input document pair list file is taken from „*./TEST/en_lt_plain_pairs_in.txt*” and the mapped term pairs are saved in „*./TEST/en_lt_T_pairs_out.txt*”.
  - For the English – Latvian document pairs:
    - *RUN_EN-LV_Plaintext_T_Mapping.bat (Windows)*
    - *RUN_EN-LV_Plaintext_T_Mapping.sh (Linux)*
    - The input document pair list file is taken from „*./TEST/en_lv_plain_pairs_in.txt*” and the mapped term pairs are saved in „*./TEST/en_lv_T_pairs_out.txt*”.
  - For the English – Romanian document pairs (mapping is done with *MapperUSFD*):
    - *RUN_EN-RO_Plaintext_T_Mapping.bat (Windows)*
    - *RUN_EN-RO_Plaintext_T_Mapping.sh (Linux)*
    - The input document pair list file is taken from „*./TEST/en_ro_plain_pairs_in.txt*” and the mapped term pairs are saved in „*./TEST/en_ro_T_pairs_out.txt*”.
- **NE mapping of MUC-7 annotated documents (in this scenario NE tagging is skipped as the *NERTEWF* is called on pre-tagged documents):**
  - For the English – Lithuanian document pairs:
    - *RUN_EN-LT_MUC7-tagged_NE_Mapping.bat (Windows)*
    - *RUN_EN-LT_MUC7-tagged_NE_Mapping.sh (Linux)*
    - The input document pair list file is taken from „*./TEST/en_lt_muc7_pairs_in.txt*” and the mapped NE pairs are saved in „*./TEST/en_lt_muc7_NE_pairs_out.txt*”.
  - For the English – Latvian document pairs:
    - *RUN_EN-LV_MUC7-tagged_NE_Mapping.bat (Windows)*
    - *RUN_EN-LV_MUC7-tagged_NE_Mapping.sh (Linux)*
    - The input document pair list file is taken from „*./TEST/en_lv_muc7_pairs_in.txt*” and the mapped NE pairs are saved in „*./TEST/en_lv_muc7_NE_pairs_out.txt*”.
  - For the English – Romanian document pairs:
    - Mapping NEs with the *NERA2* NE mapping tool:
      - *RUN_EN-RO_MUC7-tagged_RACAI_NE_Mapping.bat (Windows)*
      - *RUN_EN-RO_MUC7-tagged_RACAI_NE_Mapping.sh (Linux)*
      - The input document pair list file is taken from „*./TEST/en_ro_muc7_pairs_in.txt*” and the mapped NE pairs are saved in „*./TEST/en_ro_muc7_RACAI_NE_pairs_out.txt*”.
    - Mapping NEs with the *MapperUSFD* NE mapping tool:
      - *RUN_EN-RO_MUC7-tagged_USFD_NE_Mapping.bat (Windows)*
      - *RUN_EN-RO_MUC7-tagged_USFD_NE_Mapping.sh (Linux)*
      - The input document pair list file is taken from „*./TEST/en_ro_muc7_pairs_in.txt*” and the mapped NE pairs are saved in „*./TEST/en_ro_muc7_USFD_NE_pairs_out.txt*”.
- **Term mapping of term-tagged documents (in this scenario term tagging is skipped as the *NERTEWF* is called on pre-tagged documents):**
  - For the English – Lithuanian document pairs:
    - *RUN_EN-LT_term-tagged_T_Mapping.bat (Windows)*
    - *RUN_EN-LT_term-tagged_T_Mapping.sh (Linux)*

        - The input document pair list file is taken from „*./TEST/en_lt_term_pairs_in.txt*” and the mapped term pairs are saved in „*./TEST/en_lt_term_T_pairs_out.txt*”.
    - For the English – Latvian document pairs:
        - *RUN_EN-LV_term-tagged_T_Mapping.bat (Windows)*
        - *RUN_EN-LV_term-tagged_T_Mapping.sh (Linux)*
        - The input document pair list file is taken from „*./TEST/en_lt_term_pairs_in.txt*” and the mapped term pairs are saved in „*./TEST/en_lt_term_NE_pairs_out.txt*”.
    - For the English – Romanian document pairs:
        - Mapping terms with the RACAI TA term mapping tool:
            - *RUN_EN-RO_term-tagged_RACAI_T_Mapping.bat (Windows)*
            - *RUN_EN-RO_term-tagged_RACAI_T_Mapping.sh (Linux)*
            - The input document pair list file is taken from „*./TEST/en_ro_term_pairs_in.txt*” and the mapped term pairs are saved in „*./TEST/en_ro_term_RACAI_T_pairs_out.txt*”.
        - Mapping terms with the *MapperUSFD* term mapping tool:
            - *RUN_EN-RO_term-tagged_USFD_T_Mapping.bat (Windows)*
            - *RUN_EN-RO_term-tagged_USFD_T_Mapping.sh (Linux)*
            - The input document pair list file is taken from „*./TEST/en_ro_term_pairs_in.txt*” and the mapped term pairs are saved in „*./TEST/en_ro_term_USFD_T_pairs_out.txt*”.
- **Term mapping from parallel sentences/phrases:**
    - For the English – German language:
        - *RUN_EN-DE_PEXACC_RES_T_Mapping.bat (Windows)*
        - *RUN_EN-DE_PEXACC_RES_T_Mapping.sh (Linux)*
        - The input parallel data document is taken from „*./TEST/en_de_phrtest_pexacc_in.txt*” and the mapped term pairs are saved in „*./TEST/en_de_phrtest_pexacc_tabsep_out.txt*”.

1.2.6 Input/Output data formats

1.2.6.1 The NE/Term mapping property file format

The workflow makes use of a property file to configure the tools included in the workflow. A property file contains one parameter per line (comments are allowed only at the beginning of each line starting with the symbol “#”; also empty lines are allowed). Each property starts with an identifier, which is followed by an equation symbol “=”. The value of the property is everything (trimming both end whitespaces) that is after the equation symbol.

All supported properties are:

- “***MapperToUse***” – specifies, which mapping tool to use. Possible values are: “*USFD*” for *Multi-lingual named entity and terminology mapper* and “*RACAI*” for *NERA2: Language Independent Named Entity Mapping* and the *Language independent terminology aligner*. The default value is “*USFD*”.

- "***DefaultEnNER***" – specifies, which English NER tool to use. Possible values are: "*USFD*" for *OpenNLP wrapper* and "*RACAI*" for *NERA1: Named Entity Recognition for English and Romanian*. The default value is "*USFD*".
- "***DefaultEnTE***" – specifies, which English TE tool to use. Possible values are: "*USFD*" for *KEA wrapper* and "*RACAI*" for *Terminology Extraction for English and Romanian*. The default value is "*USFD*".
- "***LV_RefDefString***" – specifies the refinement order definition string used in *TildeNER* for Latvian named entity recognition (for further information see section 3.1.5.4.3). The default value is "*L N S R_0.7 C T_0.90 A*" (achieves higher precision with minimal recall loss).
- "***LT_RefDefString***" - specifies the refinement order definition string used in *TildeNER* for Lithuanian named entity recognition (for further information see section 3.1.5.4.3). The default value is "*L N S R_0.7 C T_0.90 A*" (achieves higher precision with minimal recall loss).
- "***RACAINERA2_MoreAnnot***" – specifies, whether the input documents to the *NERA2* tool contain any *XML* tags other than the valid term tags ("*<TENAME>*"). If "*TRUE*", the data within the tags will be ignored. If "*FALSE*", the tags will be treated as text. The default value is "*FALSE*".
- "***RACAITermAligner_MoreAnnot***" – specifies, whether the input documents to the NERA2 tool contain any *XML* tags other than the valid term tags ("*<TENAME>*"). If "*TRUE*", the data within the tags will be ignored. If "*FALSE*", the tags will be treated as text. The default value is "*FALSE*".
- "***PhrT2Glo_Thr***" – specifies the threshold of parallel data entries that are to be considered for term mapping if the third method ("*PT*") is used.

The default property "*NE-TermWorkflowProperties.prop*" is given in the "*NERTEWF*" directory of the "*NERTEWF.zip*" file. Do not delete this file as it is the default property file and is used by the workflow if no other property file is given.

1.2.6.2 The document pair list file format

For the input document pair list file format refer to the output data of *ComMetric: a toolkit for measuring comparability of comparable documents* described in section 2.1.6.

1.2.6.3 The parallel data file format

For the input parallel data file format refer to the output of *PEXACC* described in section 2.5.6.

1.2.6.4 Mapped named entity or term file format.

For the output format of each of the mapping tools refer to:

- Section 5.1.6 of *Multi-lingual named entity and terminology mapper*. A sample output (for a language pair "*EN_LV*") is as follows:

```
Apple Inc.=Apple Inc:::0.9056779744930279
```

```
Ross Bell=Ross Bells:::0.9056779744930279
Florida=Floridas:::0.8989061958123105
Roma=Romas:::0.8415685232306558
Guam=Guama:::0.8415685232306558
```

- Section 5.2.6 of *NERA2: Language Independent Named Entity Mapping*. A sample output (for a language pair "*EN_RO*") is as follows:

```
saturn            saturn            1
venus             venus             1
cristian mungiu   cristian mungiu   1
romania           românia           0,890909
romania           româniei          0,8
transylvania      transilvania      0,711538461538462
hungary           ungaria           0,895397
```

- Section 5.3.6 of the *Language independent terminology aligner*. A sample output (for a language pair "*EN_RO*") is as follows:

```
Confederacy       confederaţie      0,696969696969697
Confederacy       Confederation     0,742424242424242
Franco-American   Spaniol-American  0,64375
prairie           prerie            0,75
colonization      Colonizatorii     0,602564102564103
colonization      colonizatori      0,721153846153846
colonization      colonizatorii     0,682692307692308
federalist        federaţie         0,633333333333333
independents      Independenţă      0,721153846153846
```

1.2.7 Integration with external tools

If the user wishes to add additional Languages to the supported language list, the user has to:

- Be able to write *Perl* scripts;
- Be in a possession of a named entity recognition and (or) terminology extraction tool that accepts a tab-separated document pair list where each line contains two entries – the input document that has to be tagged and the output file where the results should be saved. A sample format is given below:

```
[Plaintext input file 1]      [Tagged output file 1]
…
[Plaintext input file N]      [Tagged output file N]
```

The output of the user's named entity recognizer has to be compliant with the *MUC-7* annotation format described in section 3.1.6.2. The output of the user's terminology extraction tool has to be compliant with the format also described in section 3.1.6.2.

If the user's system does not support such I/O data formats, the user will have to write a wrapper system that:

- pre-processes the input data so that the user's system can understand it

- post-processes the output data so that the workflow’s mapping tools can understand it.

Then the user has to add a new “*elsif*” script section for his language in methods “*TagNamedEntities*” for NER and “*TagTerms*” for TE.

The format of the section is as follows:

```
elsif ($language eq "[LANGUAGE_CODE]")
{
 $execCommand = "cd \"[TOOL_DIRECTORY]\" && [COMMAND_LINE_WITH_$fileList]
\"".$fileList."\"";
}
```

The user has to define:

- A two lowercase character language code (“*[LANGUAGE_CODE]*”). If the method “*GetTwoCharCode*” does not contain the required language mapping to a two lowercase character code, the user must add a new mapping.
- The path to the directory where the user’s NER or TE tool is located (“*[TOOL_DIRECTORY]*”).
- The command line (“*[COMMAND_LINE_WITH_$fileList]*”) to execute the user’s tool as a Perl string. The command line has to contain the “*$fileList*” variable (the I/O file list for tagging) as an argument.

2 Tools to identify comparable documents and to extract parallel sentences and/or phrases from them

This section covers the tools that classify or rank document pairs according to their comparability levels and tools which, given a list of comparable document pairs, will attempt to extract parallel textual units (sentences or phrases) from each comparable document pair.

The tools included in this section of the ACCURAT toolkit that deal with document pairing are:

- *ComMetric: a toolkit for measuring comparability of comparable documents* (developed by CTS; see section 2.1).
- *Features extractor and document pair classifier* (developed by USFD; see section 2.3).
- *EMACC: a textual unit aligner for comparable corpora using Expectation-Maximization* (developed by RACAI; see section 2.4).

The tools that deal with parallel sentence/phrase extraction included in this section are:

- *PEXACC: a parallel phrase extractor from comparable corpora* (developed by RACAI; see section 2.5).
- *A toolkit for Multi-level Parallel Data Extraction* (developed by DFKI; see section 2.6)

2.1 ComMetric: a toolkit for measuring comparability of comparable documents

2.1.1 Overview and purpose of the tool

ComMetric is designed to measure the comparability levels of document pairs via a cosine measure. The toolkit can compute comparability scores for both monolingual document pairs and bi-lingual document pairs (via using our translation toolkit). Also, given the fact that for some under-resourced languages it is usually difficult to obtain satisfactory language processing resources or tools (e.g., POS taggers, machine-readable lexicons, stop word lists, word stemmers and lemmatizers), we first translate everything into English and then we can measure the comparability levels by utilizing the English language resources.

ComMetric contains two modules: text translation and the cosine-based comparability computation.

2.1.1.1 Text translation

The translation toolkit allows users to translate text collections from a source language to a target language by using the available *Google* translation java API or *Microsoft Bing* translation java API. Currently *Google translation API* supports 63 languages and *Bing Translation API* supports 36 languages. The supported languages are listed as below.

Supported languages by *Google Translation API*:

```
AUTO_DETECT AFRIKAANS ALBANIAN AMHARIC ARABIC ARMENIAN AZERBAIJANI BASQUE
BELARUSIAN  BENGALI  BIHARI  BULGARIAN  BURMESE  CATALAN  CHEROKEE  CHINESE
CHINESE_SIMPLIFIED CHINESE_TRADITIONAL CROATIAN CZECH DANISH DHIVEHI DUTCH
ENGLISH ESPERANTO ESTONIAN FILIPINO FINNISH FRENCH GALICIAN GEORGIAN GERMAN
GREEK  GUARANI  GUJARATI  HEBREW  HINDI  HUNGARIAN  ICELANDIC  INDONESIAN
INUKTITUT IRISH ITALIAN JAPANESE KANNADA KAZAKH KHMER KOREAN KURDISH KYRGYZ
LAOTHIAN  LATVIAN  LITHUANIAN  MACEDONIAN  MALAY  MALAYALAM  MALTESE  MARATHI
MONGOLIAN NEPALI NORWEGIAN ORIYA PASHTO PERSIAN POLISH PORTUGUESE PUNJABI
ROMANIAN RUSSIAN SANSKRIT SERBIAN SINDHI SINHALESE SLOVAK SLOVENIAN SPANISH
SWAHILI SWEDISH TAJIK TAMIL TAGALOG TELUGU THAI TIBETAN TURKISH UKRANIAN
URDU UZBEK UIGHUR VIETNAMESE WELSH YIDDISH
```

Supported languages by *Bing Translation API*:

```
AUTO_DETECT  ARABIC  BULGARIAN  CHINESE_SIMPLIFIED  CHINESE_TRADITIONAL  CZECH
DANISH  DUTCH  ENGLISH  ESTONIAN  FINNISH  FRENCH  GERMAN  GREEK  HATIAN_CREOLE
HEBREW  HUNGARIAN  INDONESIAN  ITALIAN  JAPANESE  KOREAN  LATVIAN  LITHUANIAN
NORWEGIAN  POLISH  PORTUGUESE  ROMANIAN  RUSSIAN  SLOVAK  SLOVENIAN  SPANISH
SWEDISH THAI TURKISH UKRANIAN VIETNAMESE
```

The translation toolkits support two different manners of translation. For each translation call, you can send either a text string, or a string array for translation. Technically, in the following format:

```
Manner  1:  String  result=Translate.execute(String  text,  SourceLanguage,
TargetLanguage)
Manner 2: String[] result=Translate.execute(String[] text, SourceLanguage,
TargetLanguage)
```

By default, the toolkit will call Manner 1 unless the user specifies using string array translation (Manner 2).

Also, the toolkit supports two different inputs of source documents which will be translated. (1) The uses can put all the documents to be translated in a directory, and the toolkit will read all the documents from that directory for translation. (2) Sometimes the documents to be translated are from different directories, in this case the user can provide a file which lists all the documents to be translated with full path, and the toolkit will read the documents using this file, and precede the translation. Finally, apart from outputting the translated documents, a file which lists the full path of each translated document will be generated as well.

2.1.1.2 Comparability computation

The toolkit at first calls the *Standford POS-tagger* (available at http://nlp.stanford.edu/software/tagger.shtml) for POS-tagging and word tokenization. Then *JWI* (*MIT Java WordNet Interface*) is called for *WordNet*-based stemming. After word stemming, the stemmed text are converted into index vectors. Finally, the toolkit computes the comparability score of document pairs by applying cosine similarity measure on the index vectors. Document pairs with a cosine score >=threshold (a predefined value, between 0-1) are returned as output.

2.1.2 Changes from previous version

Apart from additional information in the documentation, there are no changes from the previous version to the *ComMetric* tool.

2.1.3 Software dependencies and system requirements

(1) WordNet: the toolkit uses a *WordNet*-based word stemmer. *WordNet* is available at http://wordnet.princeton.edu/wordnet/download/current-version/, the latest versions are: *WordNet 2.1* for *Windows*, and *WordNet 3.0* for *Unix*-like systems.

(2) *JWI*: the toolkit uses *MIT Java Wordnet* Interface (*JWI*, available at http://projects.csail.mit.edu/jwi/) to access the *WordNet*-based word stemming. Not like the traditional word stemmer which return the stem form of a word (the stems are usually not words), the *WordNet*-based stemmer will check if possible stems are in the *WordNet*. If so, it will only return these *WordNet*-based stems; and if not it will return the traditional stem form. Since most of the stems are words, the *WordNet*-based stemming is like a simple word lemmatization tool (which returns lemma of a given word).

(3) Stanford POS-tagger: the toolkit uses the *Stanford POS-tagger* (available at http://nlp.stanford.edu/software/tagger.shtml) for POS-tagging and word tokenization.

(4) System platform: platform independent (*Windows*, *Linux* or *Mac*)

(5) JRE: *JRE 1.6.0* (lower version should also work)

(6) English stop word list: a stop word list "*en-stopwords.txt*" is already included in the toolkit

(7) Internet: The system uses *Google* and *Bing* translation APIs for the text translation. Given that both *Google* and *Bing* translation API need to send request to remote servers for translation, the user should ensure that Internet is stably connected.

2.1.4 Installation

(1) WordNet installation: download the latest *WordNet* version (*Windows* or *Unix*-like system) and install it. Record the path to the root of the *WordNet* installation directory (for example, "*/usr/local/WordNet-3.0*" for Linux, and "*C:\WordNet-2.1*" for *Windows*) and the dictionary data directory "*dict*" (the toolkit mainly uses the *WordNet* data in the "*dict*" directory) must be appended to this path (for example, "*/usr/local/WordNet-3.0/dict*", and "*C:\WordNet-2.1\dict*"). This might be different on your system, depending on where the *WordNet* files are located.

(2) JRE installation: download and install *Windows*-based or *Linux*-based *JREs*, depending what system you use.

2.1.5 Execution instructions

2.1.5.1 Usage

```
java -jar ComMetric.jar --source [SourceLanguage] --target [TargetLanguage]
--WN [Path2WordNet] --threshold [value] --translationAPI [google|bing] --
input [path2SourceFileList] --input [path2TargetFileList] --output
[path2result] --tempDir [path2TemporaryDirectory]
```

2.1.5.2 Parameter description

"*--source [SourceLanguage]*" – non-English language.

"*--target [TargetLanguage]*" – any supported language by translation API

"*--WN [path2WordNet]*" – the full path to the *WordNet* installation directory

"*--threshold [value]*" – output the document pairs with a comparability score >= threshold (between 0-1)

"*--translationAPI [google|bing]*" – use either *Google* or *Bing* translation API

"*--input [path2SourceFileList]*" – path to the file that lists the full path to the documents in source language

"*--input [path2TargetFileList]*" – path to the file that lists the full path to the documents in target language

"*--output [path2result]*" – path to the file that store comparable document pairs with comparability scores

"*--tempDir [path2TemporaryDirectory]*" – path to a temporary directory (must exist) for storing intermediate outputs

2.1.5.3 Examples

Linux:

```
java -jar ComMetric.jar --source LATVIAN --target ENGLISH --WN
/home/fzsu/WordNet-3.0 --threshold 0.4 --translationAPI google --input
/home/fzsu/ComMetric/sample/lv.txt --input
/home/fzsu/ComMetric/sample/en.txt --output
/home/fzsu/ComMetric/sample/result.txt --tempDir
/home/fzsu/ComMetric/sample/temp
```

Windows:

```
java -jar ComMetric.jar --source LATVIAN --target ENGLISH --WN
C:\WordNet\2.1 --threshold 0.4 --translationAPI google --input
C:\ComMetric\sample\lv.txt --input C:\ComMetric\sample\en.txt --output
C:\ComMetric\sample\result.txt --tempDir C:\ComMetric\sample\temp
```

The above command example first translates Latvian documents listed in "*lv.txt*" in English and creates a folder called "*LATVIAN-translation*" in the directory "*temp*" to store the translated documents. A file called "*LATVIAN-translation.txt*" which lists full path to all the translated documents is also generated in the directory "*temp*". In addition, the stemmed data by word stemming process, and word and index vectors from text-to-vector process are stored in the directory "*temp*" as well. Finally, the toolkit computes comparability, and a document called “*result.txt*” which listed document pairs with comparability score >=threshold is generated in the specified path “*/home/fzsu/ComMetric/sample/result.txt*”.

2.1.6 Input/Output data formats

2.1.6.1 Input

For the corpus, all the documents should be *UTF-8* encoded, and in plain text. *ComMetric* takes two files containing source documents and target documents listings. In these two files, each line stores the full path to a document.

For example, in *Linux* a document listing file is as below:

```
/home/fzsu/ComMetric/sample/LV/agriculture_lv.txt
/home/fzsu/ComMetric/sample/LV/alcohol_lv.txt
/home/fzsu/ComMetric/sample/LV/cystitis_lv.txt
/home/fzsu/ComMetric/sample/LV/hockey3_lv.txt
/home/fzsu/ComMetric/sample/LV/instruction7_lv.txt
...
```

and in *Windows*, as:

```
C:\ComMetric\ComMetric\sample\LV\agriculture_lv.txt
C:\ComMetric\ComMetric\sample\LV\alcohol_lv.txt
C:\ComMetric\ComMetric\sample\LV\cystitis_lv.txt
C:\ComMetric\ComMetric\sample\LV\hockey3_lv.txt
C:\ComMetric\ComMetric\sample\LV\instruction7_lv.txt
...
```

2.1.6.2 Output

The final output file which lists document pairs with comparability scores is specified by the “*--output*” parameter. In this file, each line stores a pair of documents (full path to the documents) and the corresponding comparability score, separated by “*<TAB>*”.

Linux example of the output file:

```
/home/fzsu/ComMetric/sample/LV/instruction7_lv.txt<tab>/home/fzsu/ComMetric
/sample/EN/instruction7_en.txt<tab>0.2331
/home/fzsu/ComMetric/sample/LV/alcohol_lv.txt<tab>/home/fzsu/ComMetric/samp
le/EN/alcohol_en.txt<tab>0.8334
...
```

Windows example of *ComMetric* output:

```
C:\ComMetric\sample\LV\agriculture_lv.txt<tab>C:\ComMetric\sample\EN\agricu
lture_en.txt<tab>0.5258
C:\ComMetric\sample\LV\plant_lv.txt<tab>C:\ComMetric\sample\EN\plant_en.txt
<tab>0.7555
...
```

2.1.7 Integration with external tools

Assuming *WordNet* has been installed, the external tools in this toolkit include *Stanford POS-tagger*, *JWI* (*Java WordNet Interface*), both of them being *Java* programs and packaged as "*.jar*" files. They have been included in this toolkit and no installation is required.

2.1.8 Licence

The toolkit uses five external resources: *WordNet*, *JWI*, and *Standford POS tagger*, *Bing translation API* and *Google Translation API*. *WordNet* and *JWI* are free for both research and commercial purposes, as long as proper acknowledgement is made; *Standford POS tagger* is free for research purpose but not commercial use. *Bing* and *Google Translation APIs* are also for research purpose only. Therefore, the licence of this toolkit is "**Free to use/modify for research purposes**".

2.1.9 Contact

For further information and technical support installing and/or running this tool, please email to Fangzhong Su: F.Su@leeds.ac.uk.

2.1.10 Useful references

(1) Christiane Fellbaum. WordNet: An Electronic Lexical Database. MIT Press, 1998.

(2) Kristina Toutanova and Christopher D. Manning. 2000. Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger. In Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000), pp. 63-70.

(3) Kristina Toutanova, Dan Klein, Christopher Manning, and Yoram Singer. 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In Proceedings of HLT-NAACL 2003, pp. 252-259.

(4) JWI: http://projects.csail.mit.edu/jwi/

2.2 DicMetric: a toolkit for measuring comparability of comparable documents

2.2.1 Overview and purpose of this toolkit

This toolkit (*DicMetric*) is designed to measure the comparability levels of document pairs via cosine measure. The toolkit can compute comparability scores for both monolingual document pairs and bi-lingual document pairs. Overall, the toolkit contains two modules: text translation by lexical mapping and cosine-based comparability computation.

2.2.1.1 Text translation

The toolkit supports two types of text translation. First, for non-English and English language pairs (e.g., RO-EN), we translated the non-English texts (RO) into English by using lexical mapping from the available *GIZA++* based bilingual dictionaries. Second, for non-English language pairs (e.g., both the source and target languages are not English, i.e., EL-RO or RO-

DE), the toolkit can either translate source language (i.e., Greek or Romanian) texts into target language (i.e., Romanian or German) using "*el_ro.txt*" (or "*ro_de.txt*") dictionary; or it can also use English as the pivot language and translate both source and target language texts into English. For example, for language pair EL-RO, both the Greek and Romanian texts are translated into English using "*el_en.txt*" and "*ro_en.txt*" dictionaries and the subsequent comparability measure is thus based on English.

2.2.1.2 Comparability computation

The toolkit at first calls the *Standford POS-tagger* (available at http://nlp.stanford.edu/software/tagger.shtml) for POS-tagging and word tokenization. Then *JWI* (*MIT Java Wordnet Interface*) is called for *WordNet*-based English word stemming. After word stemming for English language, the stemmed texts are converted into index vectors. If the translated texts are not in English, then word stemming step will be skipped and directly go into feature vector conversion. Finally, the toolkit computes the comparability score of document pairs by applying cosine similarity measure on the index vectors. Document pairs with a cosine score >=threshold (a predefined value, between 0-1) are returned as output.

2.2.2 Changes from the previous version

This is a new tool added to the second version of D2.6.

2.2.3 Software dependencies and system requirements

(1) WordNet: the toolkit uses *WordNet* in a *WordNet*-based English word stemmer. *WordNet* is available at http://wordnet.princeton.edu/wordnet/download/current-version/, the latest versions are *WordNet 2.1* for *Windows*, and *WordNet 3.0* for *Unix*-like system.

(2) JWI: the toolkit uses *MIT Java Wordnet Interface* (*JWI*, available at http://projects.csail.mit.edu/jwi/) to access *WordNet* for a *WordNet*-based word stemming. Not like the traditional word stemmer which return the stem form of a word (the stems are usually not words), the *WordNet*-based stemmer will check if possible stems are in the *WordNet*. If so, it will only return these *WordNet*-based stems; and if not it will return the traditional stem form. Since most of the stems are words, the *WordNet*-based stemming is like a simple word lemmatization tool (which returns lemma of a given word).

(3) Stanford POS-tagger: the toolkit use *Standford POS-tagger* (available at http://nlp.stanford.edu/software/tagger.shtml) for POS-tagging and word tokenization.

(4) System platform: platform independent (*Windows*, *Linux* or *Mac*)

(5) JRE: JRE 1.6.0 (lower version should also work)

(6) Stopword list: stopword lists for ACCURAT languages, which are already included in the toolkit

(7) Bilingual dictionary: *GIZA++* based bilingual dictionaries for ACCURAT language pairs, which are included in the toolkit.

2.2.4 Installation

(1) WordNet installation: download the latest *WordNet* version (*Windows* or *Unix*-like system) and install it. Record the path to the root of the *WordNet* installation directory (For example, "*/usr/local/WordNet-3.0*" for *Linux*, and "*C:\WordNet-2.1*" for *Windows*) and the dictionary data directory "*dict*" (the toolkit mainly uses the *WordNet* data in the "*dict*" directory) must be appended to this path (for example, "*/usr/local/WordNet-3.0/dict*", and "*C:\WordNet-2.1\dict*"). This might be different on your system, depending on where the *WordNet* files are located.

(2) JRE installation: download and install *Windows*-based or *Linux*-based *JREs*, depending what system you use.

2.2.5 Execution instructions

2.2.5.1 Usage

```
java -jar DicMetric.jar --source [SourceLanguage] --target [TargetLanguage]
--WN  [Path2WordNet]  --threshold  [value]  --input  [path2SourceFileList]  --
input     [path2TargetFileList]     --output     [path2result]     --tempDir
[path2TemporaryDirectory] --option [0|1]
```

2.2.5.2 Parameter description

"*--source [SourceLanguage]*" – Non-English language

"*--target [TargetLanguage]*" – any supported language by translation API

"*--WN [path2WordNet]*" – the full path to the *WordNet* installation directory

"*--threshold [value]*" – output the document pairs with a comparability score >= threshold (between 0-1)

"*--input [path2SourceFileList]*" – path to the file that lists the full path to the documents in source language

"*--input [path2TargetFileList]*" – path to the file that lists the full path to the documents in target language

"*--output [path2result]*" – path to the file that store comparable document pairs with comparability scores

"*--tempDir [path2TemporaryDirectory]*" – path to a temporary directory (must exist) for storing intermediate outputs

"*--option [0|1]*" – translate the non-English text into English (option=1) or not (option=0), applied to non-English language pairs

2.2.5.3 Examples

Linux:

(1) non-English and English language pairs

```
java   -jar   DicMetric.jar   --source   latvian   --target   english   --WN
/home/fzsu/WordNet-3.0          --threshold          0.1           --input
/home/fzsu/DicMetric/sample/lv.txt                                  --input
```

```
/home/fzsu/DicMetric/sample/en.txt --output
/home/fzsu/DicMetric/sample/result.txt --tempDir
/home/fzsu/DicMetric/sample/temp --option 0
```

(2) non-English language pairs

```
java -jar DicMetric.jar --source greek --target romanian --WN
/home/fzsu/WordNet-3.0 --threshold 0.1 --input /home/fzsu/DicMetric/el-
ro/el.txt --input /home/fzsu/DicMetric/el-ro/ro.txt --output
/home/fzsu/DicMetric/el-ro/result.txt --tempDir /home/fzsu/DicMetric/el-
ro/temp --option 0
```

(3) non-English language pairs

```
java -jar DicMetric.jar --source greek --target romanian --WN
/home/fzsu/WordNet-3.0 --threshold 0.1 --input /home/fzsu/DicMetric/el-
ro/el.txt --input /home/fzsu/DicMetric/el-ro/ro.txt --output
/home/fzsu/DicMetric/el-ro/result.txt --tempDir /home/fzsu/DicMetric/el-
ro/temp --option 1
```

Windows:

```
java -jar DicMetric.jar --source latvian --target english --WN
C:\WordNet\2.1 --threshold 0.1 --input C:\DicMetric\sample\lv.txt --input
C:\DicMetric\sample\en.txt --output C:\DicMetric\sample\result.txt --
tempDir C:\DicMetric\sample\temp --option 0
```

The above command example (1) first translates Latvian documents listed in "*lv.txt*" in English and creates a folder called "*LATVIAN-translation*" in the directory "*temp*" to store the translated documents. A file called "*LATVIAN-translation.txt*" which lists full path to all the translated documents is also generated in the directory "temp". In addition, the stemmed data by word stemming process, and word and index vectors from text-to-vector process are stored in the directory "*temp*" as well. Finally, the toolkit computes comparability, and a document called "*result.txt*" which listed document pairs with comparability score >=threshold is generated in the specified path "*/home/fzsu/DicMetric/sample/result.txt*".

For non-English language pairs, example (2) will translate texts in source language (Greek) into target language (Romanian, non-English), later only stop-word filtering will be applied on the translated text and target language texts (both in Romanian now) as word lemmatization is not publically available for non-English languages. Example (3) will translate both source and target language texts into English, so both stop-word filtering and word lemmatization will be further applied on the translated texts.

2.2.5.4 DicMetric API

Apart from running this toolkit from command line, we also release a toolkit API, which allows users to call *DicMetric* directly from command line or in their own java programs. The guidelines of using this API are below:

(1) Add "*DicMetric.jar*" in the program; also put "*edu.mit.jwi_2.1.5_jdk.jar*" and "*stanford-postagger-2011-05-18.jar*" in the same directory as "*DicMetric.jar*" because they are external *JAR* files called by "*DicMetric.jar*".

(2) Import "*import leeds.cts.nlp.metric*" in the program, as "*metric*" is the only class in the *JAR* file.

(3) Put the folder "*dict*" and "*stopwords*" in the current directory of your program, as they are assumed to be in the current directory.

(4) For language pair contains English, you can call the function "*metric.ENTrack()*", for language pairs does not involve English, call "*metric.NonENTrack()*". The parameters for these two functions are given within the source code.

For example:

```
import leeds.cts.nlp.metric;
metric a=new metric();
//a.ENTrack("latvian",         "english",         "/home/fzsu/WordNet-3.0",
"/home/fzsu/DicMetric/sample/lv.txt", "/home/fzsu/DicMetric/sample/en.txt",
"/home/fzsu/DicMetric/sample/temp",
"/home/fzsu/DicMetric/sample/output.txt", 0.1);
a.NonENTrack("greek",    "romanian",    "/home/fzsu/WordNet-3.0",    "1",
"/home/fzsu/DicMetric/el-ro/el.txt",   "/home/fzsu/DicMetric/el-ro/ro.txt",
"/home/fzsu/DicMetric/el-ro/temp", "/home/fzsu/DicMetric/el-ro/output.txt",
0.1);
```

2.2.6 Input/Output data formats

2.2.6.1 Input

For the corpus, all the documents should be *UTF-8* encoded, and in plain text.

Also, two files (such as "*lv.txt*" and "*en.txt*" in the above example) which lists full path to documents in source language and target document should be available. In these two files, each line stores the full path to a document.

For example, the format of "*lv.txt*" in Linux is as below:

```
/home/fzsu/DicMetric/sample/LV/agriculture_lv.txt
/home/fzsu/DicMetric/sample/LV/alcohol_lv.txt
/home/fzsu/DicMetric/sample/LV/cystitis_lv.txt
/home/fzsu/DicMetric/sample/LV/hockey3_lv.txt
/home/fzsu/DicMetric/sample/LV/instruction7_lv.txt
```

In *Windows* its format is as below:

```
C:\DicMetric\sample\LV\agriculture_lv.txt
C:\DicMetric\sample\LV\alcohol_lv.txt
C:\DicMetric\sample\LV\cystitis_lv.txt
C:\DicMetric\sample\LV\hockey3_lv.txt
C:\DicMetric\sample\LV\instruction7_lv.txt
```

2.2.6.2 Output

The final output file which lists document pairs with comparability scores is specified by the "*--output*" parameter. So in the above example, the result will be store in the file "*result.txt*". In this file, each line stores a pair of documents (full path to the documents) and the corresponding comparability score, separated by "<*TAB*>".

For example, the format of “*result.txt*” is as below:

```
/home/fzsu/DicMetric/sample/LV/instruction7_lv.txt<tab>/home/fzsu/DicMetric/sample/EN/instruction7_en.txt<tab>0.2352
/home/fzsu/DicMetric/sample/LV/agriculture_lv.txt<tab>/home/fzsu/DicMetric/sample/EN/agriculture_en.txt<tab>0.4298
/home/fzsu/DicMetric/sample/LV/alcohol_lv.txt<tab>/home/fzsu/DicMetric/sample/EN/alcohol_en.txt<tab>0.5065
```

In *Windows*, its format is as below.

```
C:\DicMetric\sample\LV\agriculture_lv.txt<tab>C:\DicMetric\sample\EN\agriculture_en.txt<tab>0.4298
C:\DicMetric\sample\LV\alcohol_lv.txt<tab>C:\DicMetric\sample\EN\alcohol_en.txt<tab>0.5065
C:\DicMetric\sample\LV\plant_lv.txt<tab>C:\DicMetric\sample\EN\plant_en.txt<tab>0.575
```

2.2.7 Integration with external tools

Assuming *WordNet* has been installed, the external tools in this toolkit include *Stanford POS-tagger*, *JWI* (*Java WordNet Interface*), both of them are *java* programs and packaged as “*.jar*” files, thus they have been included in this toolkit and no installation is required.

2.2.8 Licence

The toolkit uses three external resources: *WordNet*, *JWI*, and *Standford POS tagger*. *WordNet* and *JWI* are free for both research and commercial purposes, as long as proper acknowledgement is made; *Standford POS tagger* is free for research purpose but not commercial use. Therefore, the licence of this toolkit is “**Free to use/modify for research purposes**”.

2.2.9 Contact

For further information and technical support installing and/or running this tool, please email to Fangzhong Su: F.Su@leeds.ac.uk.

2.2.10 Useful references

(1) Christiane Fellbaum. WordNet: An Electronic Lexical Database. MIT Press, 1998.

(2) Kristina Toutanova and Christopher D. Manning. 2000. Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger. In Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000), pp. 63-70.

(3) Kristina Toutanova, Dan Klein, Christopher Manning, and Yoram Singer. 2003. Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In Proceedings of HLT-NAACL 2003, pp. 252-259.

(4) JWI: http://projects.csail.mit.edu/jwi/

(5) Franz Josef Och, Hermann Ney. A Systematic Comparison of Various Statistical Alignment Models. Computational Linguistics, volume 29, number 1, pp. 19-51 March 2003.

2.3 Features extractor and document pair classifier

2.3.1 Overview and purpose of the tool

Methods developed in *WP3* will retrieve comparable documents from different sources. Some methods will have these documents paired, while some will retrieve sets of documents which are about the same topic without pairing them. A tool is, therefore, needed to select and pair documents, which are judged to be comparable from this set. Given a list of source documents and target documents, this tool will use all possible pairs of documents and extract numerous features from them. These features will then be used by the classifier to predict the comparability class of all the given pairs, enabling a subset of document pairs to be chosen as comparable documents.

This tool contains two processes: features extractor and classifier, which is wrapped using "*Classifier.pl*". The main workflow is described in Figure 3. A more detailed description on the purpose of each file is also described below.

The feature extractor tool will extract language dependent features and language independent features from the pairs. To enable all features to be extracted correctly, the tool will require the English translation of documents (to calculate language dependent feature) and *HTML* documents (to calculate language independent feature). These features will be extracted using "*CalculateDependentFeatures.pl*" and "*CalculateIndependentFeatures.pl*", and later summarized using "*FeaturesSummariser.pl*". The output of the Features Extractor tool will contain the score of all extracted features for all the document pairs. This output is then passed into the Classifier.

This classifier is made of two major components: (1) a binary classifier, which used Thorsten Joachim's *SVM*^{light} to implement the method, and (2) an error correction schema. At the moment, the classifier has already been trained using the Initial Comparable Corpora. However, users may use different training data for the classifier by running "*TrainDocuments.pl*". All the previously extracted features will be passed to "*ClassifyDocuments.pl*" together with the classifier model resulted from the training process, and the final output of this tool consists of selected document pairs and their predicted comparability levels.

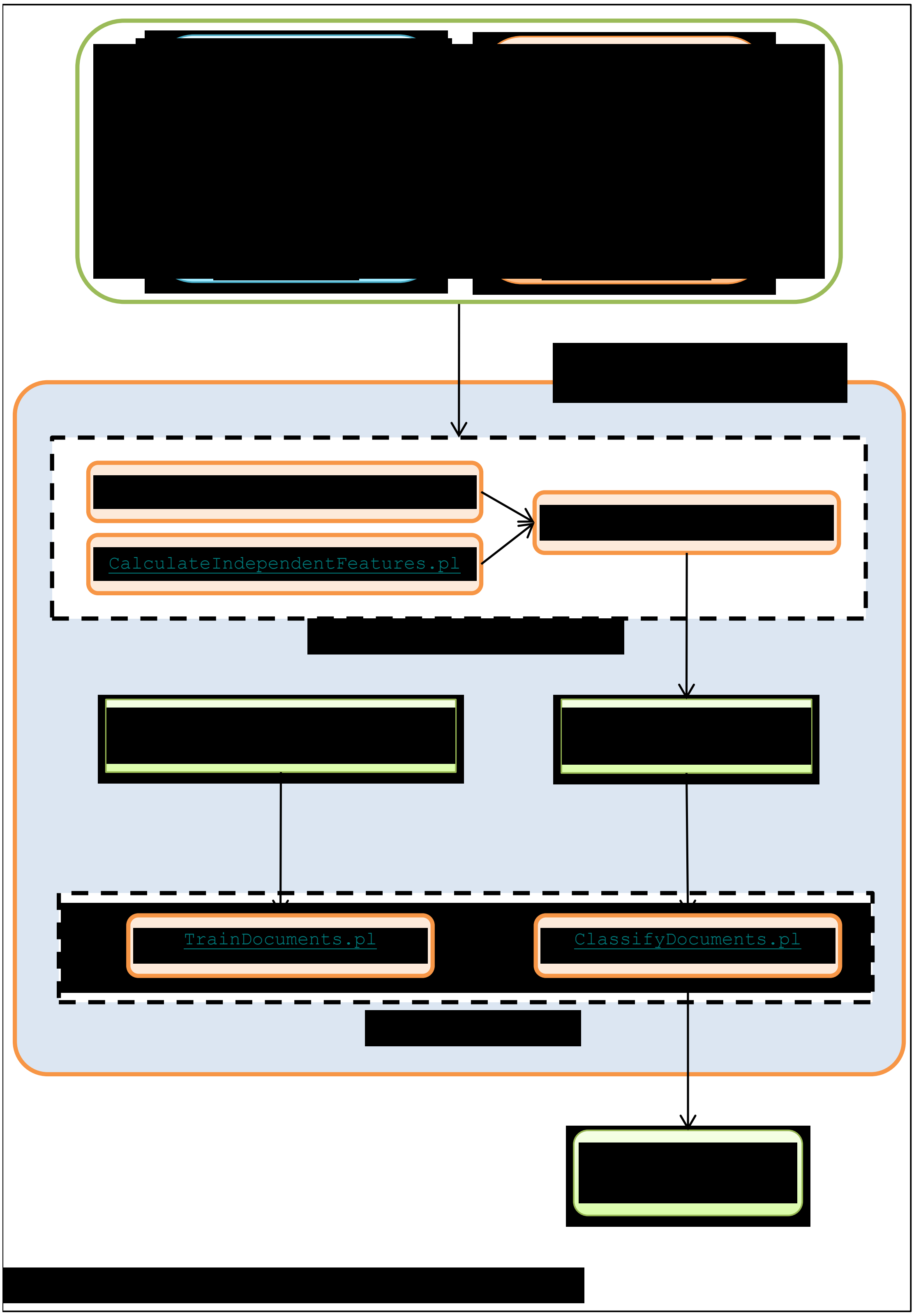


Figure 3: Workflow of Features Extractor and Classifier

2.3.2 Changes from the previous version

There are no changes from the previous version.

2.3.3 Software dependencies and system requirements

Both the features extractor and the classifier are implemented in the programming language *Perl* and can be run in *Windows* platforms. The following software is required to run this tool:

1. Perl v5.10 or above
2. 1+ GB RAM; and
3. SVM^{light}, which is available from: http://svmlight.joachims.org/[3]

2.3.4 Installation

These tools do not require any installation. Simply copy and extract "*D2_6_Section_2_3_FeaturesExtractor-Classifier.zip*". This installation also contains a folder called "*TestingData*", which contains an example scenario on how to run the program. Please have a look at "*Readme.txt*" to get further information. The detail of execution instructions is described in Section 2.3.5.

2.3.5 Execution instructions

To execute the tool, users need to run this command:

```
perl  Classifier.pl  --source  [sourceLang]  --target  [targetLang]  --input
[listOfSourceDocs]  --input  [listOfTargetDocs]  --output  [outputFile]  --
sourcehtml  [listOfHTMLSourceDocs]  --targethtml  [listOfHTMLTargetDocs]  --
sourcetranslation     [listOfTranslatedSourceDocs]     --targettranslation
[listOfTranslatedTargetDocs]  --param  threshold=[minComparabilityLevel]  -
param model=[modelFolder] -param mapping=[class mapping]
```

This script requires several parameters:

1. *[sourceLang]* represents the source language of the documents, such as Croatian, Latvian, Lithuanian, etc.
2. *[targetLang]* represents the target language of the documents, such as English.
3. *[listOfSourceDocs]* and *[listOfTargetDocs]* represent lists containing all source and target documents which need to be extracted. The format of this file is described in detail in Section 2.3.6.
4. *[listOfHTMLSourceDocs]* and *[listOfHTMLTargetDocs]* represent lists containing the corresponding HTML documents of the previous input files.
5. *[listOfTranslatedSourceDocs]* and *[listOfTranslatedTargetDocs]* represent lists containing all translated documents of the previous input files. Translations are needed only for non English documents. When translation files are not available, users may skip the "*--sourcetranslation*" and "*--targettranslation*", and the tool will automatically call the *Google Translation API* described in Section 6, which will translate the required documents. The features extractor will continue once the translation process has finished.

[3] Since testing was conducted in Windows, the version of SVM^{light} included in this package is the Windows version: http://download.joachims.org/svm_light/current/svm_light_windows.zip. Different version should be downloaded if different operating system is used to run this tool.

6. *[minComparabilityLevel]* is an optional parameter, and is used to specify the minimum threshold of the selected documents. The number represents the comparability class: *1=not parallel*, *2=weakly comparable*, *3=strongly comparable*, *4=parallel*. For example, *threshold=3* will result in only strongly comparable and parallel document pairs to be written in the output document.
7. *[model folder]* is an optional parameter, and is used to specify different models for classification. When this value is not set, the default model will be used, which is the models trained on the Initial Comparable Corpora.
8. *[class mapping]* is an optional parameter, and is also used when different data were used in the classification. This value is described in more detail in the "*TrainDocuments.pl*" description further below.

Users will not be required to run any other file, since "*Classifier.pl*" calls all the necessary classes to extract the features, classify the pairs and produce an output of the selected documents. However, each phase of this tool can also be run separately, which is explained in the following description.

First, to extract the dependent features only from the documents, users need to run this command:

```
perl    CalculateDependentFeatures.pl    --source    [sourceLang]    --target
[targetLang] --metadata [metadataFile] --outputFolder [outputFolder]
```

This script requires four parameters:

1. *[sourceLang]* represents the abbreviations of source language of the documents, such as HR, LV, LT, etc.
2. *[targetLang]* represents the abbreviations of target language of the documents, such as EN.
3. *[metadataFile]* represents a file containing all document pairs which need to be extracted. The format of this file is described in detail in Section 2.3.6.
4. *[outputFolder]* represents an output folder which will be used to store the extracted features.

The file "*CalculateDependentFeatures.pl*" contains these following modules:

1. *IndexAllFilesSub.pm*:
   This file indexes all files in the corpora, enabling the *TF* (*Term Frequency*) and *IDF* (*Inverse Document Frequency*) to be calculated.
2. *BiGramFreqOverlapStemmedCosineSimilaritySub.pm*:
   This file calculates the cosine similarity of words bi-gram frequency overlap of a document pair. The content of these documents are stemmed[4] beforehand.
3. *DocLengthWithTranslationSub.pm*:
   This file calculates the word length difference of a source document (which is previously translated to English) and a target document.
4. *TermFreqOverlapCosineSimilaritySub.pm*:

[4] All stemming processes use Porter Stemming Algorithm which is available in http://tartarus.org/~martin/PorterStemmer/

This file calculates the cosine similarity of term frequency overlap between the two documents.

5. *TermFreqOverlapStemmedCosineSimilaritySub.pm*:
   This file calculates the cosine similarity of term frequency overlap between the two documents. Both documents are previously stemmed.
6. *TFIDFOverlapStemmedCosineSimilaritySub.pm*:
   This file calculates the cosine similarity of *TF*IDF* score between the two documents. Both documents are previously stemmed.
7. *TriGramFreqOverlapStemmedCosineSimilaritySub.pm*:
   This file calculates the cosine similarity of words tri-gram frequency overlap of a document pair. The content of these documents are stemmed beforehand.
8. *WordOverlapSub.pm:*
   This file calculates the word overlap between both documents.
9. *WordOverlapCosineSimilaritySub.pm*:
   This file calculates the cosine similarity of word overlap between both documents.
10. *WordOverlapStemmedSub.pm*:
    This file calculates the cosine similarity of word overlap between both documents. Both documents are previously stemmed.
11. *WordOverlapStemmedCosineSimilaritySub.pm*:
    This file calculates the cosine similarity of word overlap between both documents. Both documents are previously stemmed.

Other features which do not require translations are extracted using "*CalculateIndependentFeatures.pl*". This tool requires the exact same parameters as the previous script and can be run using this command:

```
perl   CalculateIndependentFeatures.pl   --source   [sourceLang]   --target
[targetLang] --metadata [metadataFile] --outputFolder [outputFolder]
```

This file calls the following modules:

1. *AllInterLinksOverlapSub.pm*:
   This file calculates the overlap of inter links between the two documents.
2. *AllOutLinksOverlapSub.pm*:
   This file calculates the overlap of outlinks between the two documents.
3. *DocLengthWithoutTranslationSub.pm*:
   This file calculates the difference between document lengths of the original documents (both documents are not translated).
4. *ImageLinksFilenameOverlapSub.pm*:
   This file calculates the character overlap of image filenames in both documents.
5. *ImageLinksOverlapSub.pm*:
   This file calculates the character overlap of the entire image links in both documents.
6. *URLLevelAndCharacterOverlapSub.pm*:
   This file calculates the *URL* level overlap and *URL* character overlap of both documents.

Each of these subs will produce output in the tool. To summarise the extracted features from all the document pairs, users need to run this command:

```
perl FeaturesSummariser.pl --source [sourceLang] --target [targetLang] --
outputFolder [outputFolder] --comparabilityFile [comparabilityFile]
```

The last parameter "*--comparabilityFile [comparabilityFile]*" is optional and only needs to be used when the comparability levels between the document pairs are already known. This parameter will need to be included when *Features Extractor* tool is used to extract features scores from a training set (such as the Initial Comparable Corpora), but is not needed to extract evaluation documents. The format of comparability file is described in more detail in Section 2.3.6.

Given that the features are extracted correctly, this system will give notification output:

```
Finished: Your summary can be found in [path of the output file].
```

This output file will later be needed as an input for the classifier: *[featuresFile]*.

Before classifying these document pairs into the different comparability levels, users need to make sure that the appropriate models for the classifiers exist in the tool. At the moment, available models from the ICC are included in the folder "*Model*". However, the classifier can be trained using different corpus using the format as shown below:

```
perl trainDocuments.pl --source [sourceLang] --target [targetLang] --input
[featuresFile]    --model    [outputModelFolder]    --param    "mapping=[class
mapping]"
```

An example of this script is:

```
perl  trainDocuments.pl  --source  HR  --target  EN  --input  C:\ACCURAT\HR-EN-
summary.txt --model C:\ACCURAT\Model --param "mapping=1 0 0 0 2 3 4"
```

The script above requires five arguments to be passed:

- *[sourceLang]* represents the source language (abbreviations) of the documents, such as "*HR*" (for Croatian), "*LV*" (for Latvian), etc.
- *[targetLang]* represents the target language (abbreviations) of the documents, such as "*EN*" (for English), etc.
- *[featuresFile]* represents the extracted features of the training data which shows the comparability classes and features values.
- *[outputModelFolder]* represents an output folder which is used to stores the classifier's model.
- *[class mapping]* represents a space-separated code which is used to map the comparability levels of the features file into different classes. This is specifically used when the training data and evaluation have different classes of comparability. For example, the training corpora might contain 7 comparability classes:
  1. Not comparable (documents of different domain and genre)
  2. Not comparable (documents of different domain but the same genre)
  3. Not comparable (documents of same domain but different genre)
  4. Maybe weakly comparable (documents which are automatically aligned from a set of unaligned weakly comparable sets)

5. Weakly comparable (documents which are manually judged to be weakly comparable)
6. Strongly comparable
7. Parallel

This mapping value is used to select which classes are used in the training and testing data (the mapping value should be the same for both processes). A few examples of the mapped value which can be used are listed in the table below. The "√" symbol represents the comparability level class being used in the process. The unticked column represents the unused comparability class.

Table 4. Example of Different Mapping Values

Comparability Classes							Mapping	Description
[1]	[2]	[3]	[4]	[5]	[6]	[7]		
√	√	√	√	√	√	√	1 2 3 4 5 6 7	All classes are used in training and in the evaluation process; documents will be classified into 7 classes.
√				√	√	√	1 0 0 0 2 3 4	Four different classes are used in training and evaluation, while the data from the other 3 classes are discarded.
√	√	√		√	√	√	1 1 1 0 2 3 4	In this example, 6 different classes are used in the training, however, in the evaluation, documents will be classified into 4 classes only (all three classes of 'non comparable' documents: [1], [2] and [3], are regarded as non-comparable for evaluation).

After training is finished, users may classify the documents by running the command below:

```
perl ClassifyDocuments.pl --source [sourceLang] --target [targetLang] --input [featuresFile] --model [modelFolder] --output [outputFile] --param "mapping=[class mapping]"
```

The script above requires six essential arguments to be passed:

9. *[sourceLang]* represents the source language (abbreviations) of the documents, such as "*HR*" (for Croatian), "*LV*" (for Latvian), etc.
10. *[targetLang]* represents the target language (abbreviations) of the documents, such as "*EN*" (for English), etc.
11. *[featuresFile]* represents the extracted features of the documents to be classified
12. *[modelFolder]* represents the folder containing models to be used for classification. This is an output from "*TrainDocuments.pl*".
13. *[outputFile]* represents the output file which will be used to store the predicted comparability classes for the document pairs
14. *[class mapping]* represents the mapping value used in the training (as explained in Table 4)

The result from this script is a file containing user's specified document pairs and their predicted comparability levels.

2.3.6 Input/output data formats

This section will describe the format of input and output data for this tool. As described in Section 2.3.5, users need to run "*Classifier.pl*" and specifying inputs of list of documents: *[listOfSourceDocs]*, *[listOfTargetDocs]*, *[listOfHTMLSourceDocs]*, *[listOfHTMLTargetDocs]*, and if available, *[listOfTranslatedSourceDocs]* and *[listOfTranslatedTargetDocs]*. These files must contain the absolute path of the files, with each file written in each row:

```
C:\ACCURAT\HR\1000.txt
C:\ACCURAT\HR\1002.txt
C:\ACCURAT\HR\1005.txt
...
```

The output file will be a tab separated document, specifying the selected document pairs and their predicted comparability classes: 4 represents parallel documents, 3 represents strongly comparable documents, 2 represents weakly comparable documents, while 1 represents non comparable documents. Only documents which score higher than the threshold will be written in the output file. If threshold is not set, all document pairs will be written in the results file with their corresponding scores. An example of output file is shown below:

```
C:\ACCURAT\HR\1000.txt  C:\ACCURAT\EN\1000.txt  4
C:\ACCURAT\HR\1002.txt  C:\ACCURAT\EN\1000.txt  1
C:\ACCURAT\HR\1005.txt  C:\ACCURAT\EN\1010.txt  3
...
```

The output above represents the finished result of the features extractor and classifier. If only specific tasks are needed, different files should be run independently instead. The input and output for each file are described in more detail in the following section.

2.3.6.1 Features Extractor

In this phase, the tool will extract features which are language dependent and language independent. To extract the former one, documents in source language (and target language if necessary) need to be translated to English. Language independent features will not require translation; however, they will require information regarding links or images in the documents, which are only available in *HTML* files. Therefore, to enable all features to be extracted correctly, for each document pair, users will need to prepare the plain text of both documents, the *HTML* version of both documents, and the translated (to English) plain text. Both "*CalculateDependentFeatures.pl*" and *CalculateIndependentFeatures.pl*" will require a "*[metadataFile]*" which contains this format:

```
[SourceDoc] [SourceTranslatedDoc]   [SourceHTMLDoc]   [SourceURL]
      [TargetDoc] [TargetTranslatedDoc]   [TargetHTMLDoc]   [TargetURL]
```

As mentioned before, translated document is only needed for non-English document. Otherwise, the value should be set to be the same as the original document as the example of translated target documents below:

```
C:\ACCURAT\HR\1000.txt  C:\ACCURAT\HR\1000_en.txt
      C:\ACCURAT\HR\1000.html www.1000.com      C:\ACCURAT\EN\1000.txt
      C:\ACCURAT\EN\1000.txt  C:\ACCURAT\EN\1000.html www.1000.co.uk
```

All the fields must contain the *absolute path* of the files.

When comparability level is known, e.g. when extracting features of training data, a *[comparabilityFile]* needs to be included when using “*FeaturesSummariser.pl*”. The format of this file is as shown below:

```
[Comparability Level (numeric)]     [DocName_1]       [DocName_2]
```

The *[outputFile]* of the features extractor contains the following format:

```
[sourceFile]      [targetFile]      [comparabilityLevel]     [f1 score]   [f2
score]  [f3 score] …
```

An example of this output is shown below:

```
SourceFile  TargetFile  Comparability Level     AllInterLinksOverlap
      AllOutLinksOverlap      ImageLinksFilenameOverlap     ...
C:\ACCURAT\HR\1000.txt  C:\ACCURAT\EN\1000.txt  null  0.234 0.302 0.51 ...
C:\ACCURAT\HR\1002.txt  C:\ACCURAT\EN\1000.txt  null  0.544 0.244 0.48 ...
...
```

2.3.6.2 Classifier

As an input, the classifier will require the output from features extractor as described above, which lists all the document pairs and the values for each feature. The output from the Classifier will be a file listing all document pairs and their predicted comparability levels. The format of the output is tab separated and contains the information of the document pair and the predicted comparability level as described in the introduction of Section 2.3.6.

2.3.7 Integration with external tools

The tools described above require two external tools:

1. *Porter Stemmer*, which can be downloaded from http://tartarus.org/~martin/PorterStemmer/. At this moment, this stemmer has been included in the Features Extractor tool so no further download is necessary.
2. *SVM^{light}* algorithm by Joachims, which can be downloaded from http://svmlight.joachims.org/. At this moment, this tool has been included in the Features Extractor tool and therefore no further download is necessary. However, different *SVM^{light}* toolkit may be needed when this tool is used in different platform.

2.3.8 Contact

For further information and technical support installing and/or running this tool, please email to Monica Paramita: m.paramita@shef.ac.uk.

2.4 EMACC: a textual unit aligner for comparable corpora using Expectation-Maximization

2.4.1 Overview and purpose of the tool

EMACC is designed to align (translation-wise) different types of textual units such as documents, paragraphs or sentences in order to reduce the search space for subsequent alignment tasks. For instance, suppose that we want to word-align a bilingual comparable corpus consisting of M documents per language, each with k words, using the IBM-1 word alignment algorithm (Brown et al., 1993). This algorithm searches for each source word, the target words that have a maximum translation probability with the source word. Aligning all the words in our corpus with no regard to document boundaries, would yield a time complexity of k^2M^2 operations. The alternative would be in finding a 1:p (with p a small positive integer, usually 1, 2 or 3) document assignment (a set of aligned document pairs) that would enforce the “no search outside the document boundary” condition when doing word alignment with the advantage of reducing the time complexity to k^2Mp operations. When M is large, the reduction may actually be vital to getting a result in a reasonable amount of time. The downside of this simplification is the loss of information: two documents may not be correctly aligned thus depriving the word-alignment algorithm of the part of the search space that would have contained the right alignments.

The principle behind *EMACC's* functionality is that, translation equivalents (both correct and, surprisingly, incorrect) play a key role in document alignment. We have experimentally found that there is a certain balance between *the degree of correctness of translation equivalents* and *their ability to pin-point correct document alignments*. In other words, the paradox resides in the fact that if a certain pair of translation equivalents is not correct but the respective words appear only in documents which correctly align to one another, that pair is very important to the alignment process. Conversely, if a pair of translation equivalents has a very high probability score (thus being correct) but appears in almost every possible pair of documents, that pair is not informative to the alignment process and must be excluded. We see now that the *EMACC* aims at finding the set of translation equivalents that is maximally informative with respect to the set of document alignments.

The basic workflow of *EMACC* is as follows:

1. Pre-compute the initial document alignment distribution according to the *D2* distribution (Ion, 2011);
2. Iteratively (greedily) find the best document alignment set (called an assignment) by computing a (translation equivalents based) similarity measure between each pair of source and target documents;
3. Re-estimate the probabilities of translation equivalents from the best assignment and resume from step 2 for a given number of steps.

2.4.2 Changes from previous version

- *EMACC* is now able to align multiple target documents to one source document and by doing so, it is possible that many source documents align to a target document;
- Fixed a bug where *EMACC* behaved randomly with the same set of parameters;
- Heavy memory optimization (hard disk matrices) which resolve "*Out of memory*" issues of *EMACC* when running on large document sets;
- Added a "simple" run mode for *EMACC* such that, for very large document collections, no EM is performed (it takes a very long time to complete);
- Added portable Perl code for running in both Windows and Linux environments.

2.4.3 Software dependencies and system requirements

EMACC is entirely written in *Perl* and it works on a fairly recent version of *Perl* (5.8 or 5.10). The pre-computing of the initial document alignments distribution can be done in a parallel fashion (on a *Linux* cluster). If one wants to run in this mode, additional requirements must be met:

- *SSH* secure remote shell must be installed on the master node of the cluster;
- User "*rion*" must be allowed passwordless *SSH* access[5] from the master node to each of the cluster nodes (the user name is not configurable);
- A *NFS* mount point[6] must be read and write accessible to all the nodes of the cluster.

The test-bed is a 4-node *Linux* cluster running *Ubuntu Linux* (versions beginning with 9.04 and newer). Nodes have one or two *Intel Xeon* processors, each with 4 cores. The RAM on each node varies between 6 and 8 GB and the *NFS* drive has around 8TB of storage available (although for smooth performance, depending on the size of the document collection, at least 10-20 GB of storage should be available).

2.4.4 Installation

If "*emacc2.pl*" is going to be run on a cluster and assuming that *Perl* is already installed on all cluster nodes, one should go through each of the following steps to obtain a working setup of EMACC:

- Install a NFS server onto the master node of the cluster and NFS clients onto all nodes of the cluster. Mount the same NFS point (e.g. onto *"/mnt/nfs*") in read and write modes on each node of the cluster;
- Edit the configuration file "*emaccconf.pm*" and setup all desired values (more on this in the next section);

[5] To set up a SSH login without passwords authentication, see here: http://linuxproblem.org/art_9.html

[6] To set up NFS on Ubuntu Linux, see here: https://help.ubuntu.com/community/SettingUpNFSHowTo

- Copy the script "*precompworker.pl*" onto each node of the cluster in the home directory of the user "*rion*". Make sure that "*precompworker.pl*" has executable rights by "*rion*".

2.4.5 Execution instructions

In order to run EMACC one must go through the following steps (in order):

1. Edit and configure the file "*emaccconf.pm*". This is the global configuration file from which all values for all parameters are read (the important running parameters may also be specified in the command line);
2. Run "*emacc2.pl*" taking care of giving it (if desired) important running parameters (see below) and (mandatory) the document lists files: the source document list file and the target document list file.

In order to run "*emacc.pl*", another file has to be edited: "*cluster.info*". This file contains a description of processor cores that are available for running (on the local machine or on remote machines). The format of the file is given below:

```
#hostname<TAB>IP<TAB>CPUID
## is a comment
nefertiti   172.16.39.117     cpu0
nefertiti   172.16.39.117     cpu1
nefertiti   172.16.39.117     cpu2
nefertiti   172.16.39.117     cpu3
nefertiti   172.16.39.117     cpu4
nefertiti   172.16.39.117     cpu5
nefertiti   172.16.39.117     cpu6
nefertiti   172.16.39.117     cpu7
akhenaten   172.16.39.118     cpu0
akhenaten   172.16.39.118     cpu1
akhenaten   172.16.39.118     cpu2
akhenaten   172.16.39.118     cpu3
```

Empty lines or lines beginning with "*#*" are ignored (as comments). A useful line states the hostname of the machine (say "*Nefertiti*"), its IP address ("*172.16.39.117*") and a processor core ID. For instance, let's say that the cluster has only 2 nodes: "*nefertiti*" and "*akhenaten*". "*nefertiti*" has 2 processors each with 4 cores and "*akhenaten*" has only 1 processor with 4 cores. Thus, the cores of "*nefertiti*" may be named "*cpu0*" through "*cpu7*" and the cores of "*Akhenaten*", "*cpu0*" through "*cpu3*".

The number of useful lines from "*cluster.info*" will say how many processes will be run in parallel by "*emacc2.pl*". If one does not have a cluster, this file will contain the local host name along with its IP and IDs of its CPU cores. The local host name must be the same as the name reported by the "*hostname*" command (both *Linux* and *Windows*). There is a command line option (see below) that instructs *EMACC* to auto-generate this file for local use (not for cluster use) such that the user need not bother with the configuration.

The *EMACC* configuration file is a *Perl* module called “*emaccconf.pm*”. Configuration is done by simply editing this file and fill in the appropriate values in the marked sections. The configuration file is self-explanatory. Just read the comments above each parameter.

The command line of “*emacc.pl*” is as follows (if the script is in the current directory):

```
Usage : emacc.pl \
        [--source en] [--target ro] \
        [--param EMLOOPS=5] \
        [--param EMACCMODE=emacc-full] \
        [--param MAXTARGETALIGNMENTS=3] \
        [--param INIDISTRIB=D2] \
        [--param TEQPUPDATETHR=0.4] \
        [--param LEXALSCORE=0.4] \
        [--param PROBTYPE=giza] \
        [--param CLUSTERFILE=generate] \
        --input <source language document list file> \
        --input <target language document list file> \
        [--output <output file name>]
```

where a file containing a document listing (specified with a “*--input*” option) has the following format:

```
/path/to/the/document1.txt
/path/to/the/document2.txt
/path/to/the/document3.txt
…
```

Thus a line contains the path to a document in the source/target collection. Both target and source documents must be on the NFS drive if “*emacc2.pl*” is run on the cluster. When this command is invoked, a number of “*precompworker.pl*” processes is started (determined from reading the “*cluster.info*” file). The STDERR of the main process provides logging on what’s going on. The output of *EMACC* is a file (optionally specified with “*--output*” switch) containing the listing of document pairs (alignments) along with alignment scores.

Command line switches that may be specified and that influence the behaviour of *EMACC* are as follows:

- “*--source*” specifies the language of the source document collection. May be given by full name or as a 2 or 3 letter code;
- “*--target*” specifies the language of the target document collection. May be given by full name or as a 2 or 3 letter code;
- “*--param EMLOOPS*”, which specifies for how many loops the EM algorithm is looping in order to find the most probable document alignment (usually set to 3 or 5). This applies only when *EMACCMODE* is set to “*emacc-full*”;
- “*--param EMACCMODE*”, which specifies the default operation of the aligner. When “*emacc-full*” is given, the full Expectation-Maximization re-estimation of

the alignments is done. “*emacc-simple*” will perform a simpler, greedy alignment based on the pre-computed D2 alignments. The second option is suggested when dealing with large document collections (more than 5000 documents per language);

- “*--param MAXTARGETALIGNMENTS*” enables one-to-many document alignments. The integer value of this parameter instructs *EMACC* to find at most that many target documents that align to a single source document;
- “*--param INIDISTRIB*” is the initial document alignment distribution from which EM begins estimating new alignments. “*D2*” provides better results. “*D1*” may also be specified and means a uniform distribution;
- “*--param TEQPUPDATETHR*” is the threshold over which translation equivalents probabilities are re-estimated in the course of the EM procedure. Values between 0.1 and 0.5 provided the best results in our tests;
- “*--param LEXALSCORE*” is the threshold over which translation equivalents from the *GIZA++* dictionary are considered in document alignment. Do not increase this value too much or there will not be enough translation information to properly align the documents;
- “*--param PROBTYPE*” is the type of probability of translation equivalents. “*giza*” is the probability extracted from the dictionary and “*comp*” is a modified version.
- “*--param CLUSTERFILE*” specifies the “*cluster.info*” file that instructs *EMACC* how many processes to spawn. If the value of this parameter is “*generate*”, *EMACC* will auto-generate this file for local use (not cluster use! -- for cluster use, modify it by hand!) so that the user does not need to configure it (or even create the file for that matter);
- “*--input*” (the first one) specifies the source documents list file;
- “*--input*” (the second one) specifies the target documents list file;
- “*--output*” specifies the output file that will contain the final document alignments. If not specified, a default file will be used (check “*emaccconf.pm*” for the name and location of that file).

All parameters (“*--param*” options) may also be specified directly into the “*emaccconf.pm*” file (at least those that are not frequently modified). The only mandatory arguments are the “*--input*” options (2 of them).

2.4.6 Input/Output data formats

EMACC requires the lists of source and target documents in two separate files (specified with the “*--input*” command line switch). The format of a list file is repeated here for convenience:

```
/path/to/the/document1.txt
/path/to/the/document2.txt
/path/to/the/document3.txt
…
```

"*emacc2.pl*" produces a document alignment of the form:

```
/path/to/source/document1.txt<TAB>/path/to/target/document15.txt<TAB>-1.1
/path/to/source/document1.txt<TAB>/path/to/target/document10.txt<TAB>-2.2
/path/to/source/document2.txt<TAB>/path/to/target/document2.txt<TAB>-3
…
```

where, on each line of the file, we can find the path to the source document of the pair, the path of the target document and the alignment score (presented as probabilities in natural logarithm), all separated by "*\t*" (the *TAB* character). This is the standard input for our phrase extractor tool called *PEXACC* (see the previous section).

EMACC makes use of several linguistic resources which, in case of running on the cluster, must be installed on the *NFS* drive and visible by all cluster nodes. In what follows, we will make a summary of these resources, each with its own format:

- *stop word lists*: lists of word forms of functional words (one word per line, *UTF-8* encoded)
- *inflectional endings lists*: lists of inflectional endings that are used to stem words in order to gain statistical significance (one ending per line, *UTF-8* encoded)
- *GIZA++* dictionaries in the form:

```
source word <TAB> target word <TAB> probability <NEWLINE>...
```

2.4.7 Integration with external tools

There are no other tools that need to be installed and/or used in conjunction with *EMACC*.

2.4.8 Contact

For further information and technical support installing and/or running this tool, please email to Radu Ion: radu@racai.ro.

2.4.9 Useful references

Brown, P. F., Lai, J. C., and Mercer, R. L. 1991. *Aligning sentences in parallel corpora*. In Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics, pp. 169–176, June 8-21, 1991, University of California, Berkeley, California, USA.

Ion, R., Ceauşu, A., and Irimia, E. 2011. *An Expectation Maximization Algorithm for Textual Unit Alignment*. In Proceedings of the 4th Workshop on Building and Using Comparable Corpora (BUCC 2011) held at the 49th Annual Meeting of the Association for Computational Linguistics, pp. 128—135, Portland, Oregon, USA, June 24th, 2011. (C) 2011 Association for Computational Linguistics. ISBN: 978-1-937284-01-5.

2.5 PEXACC: a parallel phrase extractor from comparable corpora

2.5.1 Overview and purpose of the tool

Comparable corpora are inherently different from parallel corpora:

- The order of translation is not preserved; thus, the significant search space optimization from which all parallel alignment algorithms beneficiate (the "translation window" out of which no translations are possible) is null in this case;
- The translations that one finds in comparable corpora are (most of them) accidental; thus, the match between pieces of text is more difficult due to the fact that the meaning of the source phrase may only be approximately reproduced in one target candidate phrase.

Given these characteristics of comparable corpora, *PEXACC* will try to alleviate the effect of these problems by:

- Trying (and scoring) all possible combinations of pairs of pieces of text (or textual units) so that each pair will receive a "translation probability score" that the source textual unit is translated by the respective target textual unit;
- Using relevance feedback loops which is a mechanism by which *PEXACC* learns new translations from the already mapped data so that, new information may be found and added to the parallel data already found.

The general purpose of *PEXACC* is to extract parallel data from comparable corpora for use in SMT training of translation models. The granularity level of the textual units that can be mapped is customizable. Thus, *PEXACC* can align sentences and/or sub-sentential parts of text to which we will refer as "chunks". We have imposed this restriction in order to deal with weakly comparable corpora which, generally, do not contain sentential translations.

The general processing flow of *PEXACC* is as follows:

1. For a list of document pairs found by *EMACC* (see the next section) and for each pair of documents from that list,
2. Split the source and the target documents at sentence/chunk level (depending on a configuration option),
3. Find all pairs of sentences/chunks that score above a certain threshold at "translation probability",
4. Apply *GIZA++* on all the pairs found at step 3 and add the resulting dictionary to the base dictionary that *PEXACC* uses,
5. Go to step 3 and rescore all the pairs of sentences/chunks. Repeat this loop for a number of steps (experimentally, set to 5)

2.5.2 Changes from previous version

- Added portable *Perl* code for running in both *Windows* and *Linux* environments;
- No dependency upon the *String::Similarity* package from *CPAN.org;*

- Changed the parallel computation paradigm and removed the *NFS* file reading/writing for improving speed when running on a network cluster;
- Changed the parallelism similarity metric to be symmetrical in order to have evidence from both translation directions;
- Fixed a sentence splitting bug where empty sentences were generated;
- Added a filtering step such that punctuation/number phrases are filtered out;
- Added chunk clustering when aligning chunks of text (consecutive aligned chunks of text form larger pieces of aligned text);
- Added a better tokenization routine of the sentences to be paired;
- Fixed several bugs found in the similarity measure routine.

2.5.3 Software dependencies and system requirements

PEXACC is entirely written in *Perl* and it works on a fairly recent version of *Perl* (5.8 or 5.10). The software that needs to be installed so that *PEXACC* can run is:

- *Perl* (5.10 preferred); optionally, the String::Similarity package installed from CPAN.org;
- *GIZA++* 1.0.5 from *Google Code*: http://code.google.com/p/giza-pp/downloads/detail?name=giza-pp-v1.0.5.tar.gz

Note: when compiling *giza-pp 1.0.5* on *Ubuntu 9.04*, one must remove the "*gcc*" optimization flags (-On where n=2, 3) from the *GIZA++ Makefile* because the resulting executable crashes randomly.

PEXACC has support for distributed computing. If one wants to run in this mode, additional requirements must be met:

- *SSH* secure remote shell must be installed on the master node of the cluster and the "*scp*" remote copy utility must be accessible in the PATH;
- *GIZA++* must be installed on the master node of the cluster;
- User "*rion*" must be allowed passwordless *SSH* access[7] from the master node to each of the cluster nodes (the user name is not configurable) and from each cluster node to the master;
- All the resources that *PEXACC* uses (dictionaries, inflection lists, stop words lists) must be copied onto each node of the cluster **with the same absolute paths** as on the master.

The test-bed is a 4-node *Linux* cluster running *Ubuntu Linux* (versions beginning with 9.04 and newer). Nodes have one or two *Intel Xeon* processors, each with 4 cores. The RAM on each node varies between 6 and 8 GB and the *NFS* drive has around 8TB of storage available (although for smooth performance, depending on the size of the document collection, at least 10-20 GB of storage should be available).

[7] To set up a SSH login without passwords authentication, see here: http://linuxproblem.org/art_9.html

2.5.4 Installation

Assuming that *Perl* is already installed, one should go through each of the following steps to obtain a working setup of *PEXACC*:

- Use the "*perl –MCPAN –e shell*" command to install the *Perl* package *String::Similarity* onto each node of the cluster; if this package is not installed, *PEXACC* will use its own internal implementation of the string similarity measure;
- Download and compile *giza-pp-1.0.5*. Install it on the computer running *PEXACC* in a location that you will remember (e.g. "*/usr/local/giza++-1.0.5/bin*") because the location of the *GIZA++* executable and other utilities is required to configure *PEXACC*;
- Edit the configuration file "*pexacc2conf.pm*" and setup all desired values (more on this in the next section);
- Copy the script "*pdataworker.pl*" and the package "*strsim.pm*" onto each node of the cluster in the home directory of the user "*rion*". Make sure that "*pdataworker.pl*" has executable rights by "*rion*";
- Copy the "*res/*" and "*dict/*" directories from the distribution kit onto each node of the cluster making sure that the absolute path to these directories from the master node is the same on each cluster node. This path can be set by editing the "*pexacc2conf.pm*" file and configuring the "*PEXACCWORKINGDIR*" entry.

2.5.5 Execution instructions

In order to run *PEXACC* one must follow some configuration steps first.

First file that has to be edited is "*cluster.info*". This file contains a description of processor cores that are available for running (on the local machine or on remote machines). The format of the file is given below:

```
#hostname<TAB>IP<TAB>CPUID
## is a comment
nefertiti   172.16.39.117     cpu0
nefertiti   172.16.39.117     cpu1
nefertiti   172.16.39.117     cpu2
nefertiti   172.16.39.117     cpu3
nefertiti   172.16.39.117     cpu4
nefertiti   172.16.39.117     cpu5
nefertiti   172.16.39.117     cpu6
nefertiti   172.16.39.117     cpu7
#akhenaten  172.16.39.118     cpu0
#akhenaten  172.16.39.118     cpu1
#akhenaten  172.16.39.118     cpu2
#akhenaten  172.16.39.118     cpu3
```

Empty lines or lines beginning with "*#*" are ignored (as comments). A useful line states the hostname of the machine (say "*nefertiti*"), its IP address ("*172.16.39.117*") and a processor core ID. For instance, let's say that the cluster has only 2 nodes: "*Nefertiti*" and "*akhenaten*". "*Nefertiti*" has 2 processors each with 4 cores and "*akhenaten*" has only 1 processor with 4 cores. Thus, the cores of "*Nefertiti*" may be named "*cpu0*" through "*cpu7*" and the cores of "*akhenaten*", "*cpu0*" through "*cpu3*".

The number of useful lines from "*cluster.info*" will say how many processes will be run in parallel by *PEXACC*. If one does not have a cluster, this file will contain the local host name along with its IP and IDs of its CPU cores. The local host name must be the same as the name reported by the "*hostname*" command (both Linux and Windows). There is a command line switch (see below) that will force *PEXACC* to auto-generate this file in order for it to run locally[8] (no clustering involved).

The *PEXACC* configuration file is a Perl module called "*pexacc2conf.pm*". Configuration is done by simply editing this file and fill in the appropriate values in the marked and commented sections or by supplying values for a selected collection of parameters directly into the command line of *PEXACC*. The configuration file is self-explanatory. Just read the comments above each parameter.

The command line run of *PEXACC* is as follows (if the script is in the current directory):

```
Usage: ./pdataextract-p.pl \
        [--source en] [--target ro] \
        [--param GIZAPPEXE=/usr/local/giza++-1.0.5/bin/GIZA++] \
        [--param PLAIN2SNTEXE=/usr/local/giza++-1.0.5/bin/plain2snt.out] \
        [--param CLUSTERFILE=generate] \
        [--param SENTRATIO=1.5] \
        [--param SPLITMODE=chunk] \
        [--param OUTPUTTHR=0.1] \
        [--param GIZAPPITERATIONS=3] \
        --input <--output file from emacc.pl or equivalent> \
        [--output <output file>]
```

where the command line switches have the following meanings (indicated are the defaults):

- "*--source*" and "*--target*" specify the source language and the target language respectively (for all ACCURAT languages with English on one side, preferably the source). These languages may be specified by their full name or by 2 or 3 letter codes. They are converted internally to a 2 letter code;
- "*--param GIZAPPEXE*" specifies the location of the *GIZA++* executable (it can be configured only once in "*pdataextractconf.pm*");
- "*--param PLAIN2SNTEXE*" specifies the location of the "*plain2snt.out*" executable (it can be configured only once in "*pexacc2conf.pm*");

[8] If run on *Linux*/*Unix*, *PEXACC* is able to detect the number of processors/cores that the system recognizes. If run on *Windows*, only one processor/one core is assumed (specific C++ routines are required to correctly determine this information). So, it is strongly recommended to edit the "*cluster.info*" file if running in an *Windows* environment.

- "*--param CLUSTERFILE*" specifies the "*cluster.info*" file. If the value of this parameter is "*generate*", an auto-generated file will be created that will ensure that *PEXACC* can be run locally without the user needing to configure this file;
- "*--param SENTRATIO*" is the maximum ratio allowed between the larger text fragment (count in words) over the smaller text fragment in a candidate pair. Any pairs exceeding this value will be discarded;
- "*--param SPLITMODE*" may be one of "*chunk*" or "*sent*". If "*chunk*", then *PEXACC* will split the text into sentences and then, into smaller text fragments (use "*chunk*" with weakly comparable corpora). "*sent*" instructs *PEXACC* to perform only sentence splitting;
- "*--param OUTPUTTHR*" is the parallelism probability threshold over which parallel sentences/phrases are actually written in the output file;
- "*--param GIZAPPITERATIONS*" is the number of extract-train-loop iterations that *PEXACC* is going to execute. A couple of iterations (3 to 5) experimentally guarantees that more (and better) parallel sentences/phrases are extracted;
- "*--input*" requires the document alignment file from *EMACC* or similar;
- "*--output*" specifies the name of the output file. This file will contain the final results.

All these options (with the exception of the file from the "*--input*" switch) and some more may be configured by also directly editing the configuration file "*pexacc2conf.pm*".

The file containing the document alignments (specified with the "*--input*" switch) has the following format:

```
/path/to/source/document1.txt<TAB>/path/to/target/document15.txt<TAB>-0.5
/path/to/source/document1.txt<TAB>/path/to/target/document10.txt<TAB>-1
/path/to/source/document2.txt<TAB>/path/to/target/document2.txt<TAB>-2
…
```

Thus a line contains a pair of documents with an alignment score (probabilities in natural logarithm). The source and target documents are separated by "*\t*" (the *TAB* character) and the alignment score is also separated by "*\t*" from the pair. This file is typically produced a document aligner application such as *EMACC* (see the next section).

When that command is invoked, a number of "*pdataworker.pl*" processes is started (determined from reading the "*cluster.info*" file) and each of the collection of aligned phrases/sentences is written in an iteration-dependent file (see the "*pexacc2conf.pm*" file for the location and naming convention of the results). The *STDERR* of the main process provides logging on what's going on.

PEXACC must be run on the master node of the cluster (the one containing the "*pexacc2.pl*" script).

2.5.6 Input/Output data formats

PEXACC requires as input a single document alignment file produced by *EMACC* for instance. The format of that file has been presented already but is also given below:

```
/path/to/source/document1.txt<TAB>/path/to/target/document15.txt<TAB>-0.5
/path/to/source/document1.txt<TAB>/path/to/target/document10.txt<TAB>-1
/path/to/source/document2.txt<TAB>/path/to/target/document2.txt<TAB>-2
…
```

Thus a line contains a pair of documents with an alignment score (probabilities in natural logarithm). The source and target documents are separated by "\t" (the *TAB* character) and the alignment score is also separated by "\t" from the pair. The source and target documents themselves must be *UTF-8* encoded with not byte order markings at the beginning. Both target and source documents must be on the *NFS* drive if *PEXACC* is run on the cluster.

PEXACC uses the following types of resources (see the *PEXACC* distribution for examples) **which must be copied onto each cluster node if run in cluster mode**:

- *GIZA++* dictionaries with the following format:

```
source word form<TAB>target word form<TAB>probability<NEW LINE>
```

- *markers files*: lists of word forms of functional words (one word per line, *UTF-8* encoded) that usually mark the end of clauses or other types of syntactic phrases (e.g. verbal or prepositional phrases)
- *inflectional endings lists* of strings (*UTF-8* encoded, one string per line) that usually appear in words' suffixes to indicate gender, definiteness, etc.
- *stop word lists*: lists of word forms of functional words (one word per line, UTF-8 encoded)

The output of *PEXACC* is a collection of files, one for each iteration. These files contain the set of aligned phrases/sentences that *PEXACC* found at iteration *i*. The file corresponding to the last iteration should contain the largest set of aligned phrases. The format of such a file is:

```
source phrase 1
target phrase 1
probability 1

source phrase 2
target phase 2
probability 2


…
```

If the configuration file "*pexacc2conf.pm*" is not changed with respect of output file naming, then, for a 3 iteration run of *PEXACC*, for English ("*en*") to Romanian ("*ro*") alignment of phrases from "*Wikipedia*" documents, the last output file would be named "*en-ro-Wikipedia-pdataextract-p-3.txt*". If the "*--output*" command line switch is specified, the last output file will also be copied into the value of the "*--output*" switch.

2.5.7 Integration with external tools

GIZA++ is the only external tool that needs to be available to *PEXACC* (see the "System requirements" section). *GIZA++* is configured by a file called "*pdataextract-gizapp.gizacfg*" which is distributed along with PEXACC. It is a standard *GIZA++* configuration file minus the dynamic information ("*.snt*" and "*.vcb*" files) supplied by *PEXACC* at run time. It needs not be erased between runs.

2.5.8 Contact

For further information and technical support installing and/or running this tool, please email to Radu Ion: radu@racai.ro.

2.5.9 Useful references

Official *Ubuntu* Documentation (if installation is done on *Ubuntu*):

https://help.ubuntu.com/

Comprehensive *Perl* Archive Network:

http://www.cpan.org/

Open SSL:

http://www.openssl.org/

D2.2 ACCURAT Deliverable: "Report on multi-level alignment of comparable corpora" which documents the *PEXACC* algorithm.

GIZA++ documentation:

Franz Josef Och, Hermann Ney. "Improved Statistical Alignment Models". *Proc. of the 38th Annual Meeting of the Association for Computational Linguistics*, pp. 440-447, Hong Kong, China, October 2000.

2.6 A ME parallel sentence extractor tool

2.6.1 Overview and purpose of the tool

This tool is built to extract parallel sentences or phrases from comparable corpus. A maximal entropy method (Munteanu and Marcu 2005) was applied to classify each potential aligned sentence/phrase pair as parallel or non-parallel. User could extract their own parallel data with any language pair. In addition, users are able to train their own maximal entropy (ME) model by providing numbers of samples.

2.6.2 Changes from previous version

The updated version of the ME parallel sentence extractor allows users to train their own models and contains an updated feature set. As the updated version contains new use cases, the execution sequences from the last version won't work in the new version. Users must take this change into account when updating to the new version.

2.6.3 Dependencies and system requirements

1) This tool has been tested on the following environment:

Table 5 Tested environments

Machine Type	OS	Gcc version
X86_64	Ubuntu 4.4 version 2.6.31	4.4.1/ 4.2.4
i686	SUSE Linux version 2.6.11.4	3.3.5

2) A word translation probability table should be provided, with a numeric value which present its probability, one space is between them for splitting, as:

```
Target_word_1          source_word_1     number1
Target_word_2          source_word_2     number2
...
```

i.e. we use Giza's lexcial table after word alignment on europarl-v6.news-commentary corpus, German as the source language, English as the target language:

```
functions Fluglizenzen 0.1428571
regulated Fluglizenzen 0.0714286
for Fluglizenzen 0.1428571
...
```

2.6.4 Installation

We provide a default binary executable "*extract*", which is runnable under *x86_64* environment. If you want to re-build it, please input the following command under *Linux* bash:

1) ***make clean***: clean all executive and output files.
2) ***make all***: compile, link and generate all output file and executable.

2.6.5 Execution instructions

After installation, you will get one executable files: ***extract***.

It's a build for extracting parallel corpus from comparable data.

2.6.5.1 Instructions of extraction

(1) Usage:

```
chmod +x extract
./extract --source […] --target […] --param LEX=[…] --input […] --output
[…]
```

(2) Parameter description:

- ***--source [LANG]*** - Source Language

- ***--target [LANG]*** - Target Language
- ***--param [[…]=[…]]*** – parameters:
    - ***LEX=[…]*** – File path of lexical translation table
    - ***TRAIN=[0/1]*** – Switch between extract/train mode
    - ***TRAIN_SIZE=[…]*** – Training sample number (Train mode only)
    - ***TEST_SIZE=[…]*** – Test sample number (Train mode only)
- ***--input***
    - ***[PATH TO A FILE]*** – The document pair list file from the comparability metrics. (extract mode)
    - ***[PATH TO A Directory]*** – The directory which contains training and test sample files with specific name format. (train mode)
- ***--output[PATH TO A FILE]*** – Output file path . (extract mode only)

(3) Command line examples

An extraction example is in „*./sample/extract.sh*”,

```
./extract  --source  de  --target  en  --param  LEX=Script/lex.6.f2e  TRAIN=0
TRAIN_SIZE=0 TEST_SIZE=0 --input ./sample/input --output ./sample/output
```

An training example is in „*./sample/train.sh*”,

```
./extract  --source  de  --target  en  --param  LEX=Script/lex.6.f2e  TRAIN=1
TRAIN_SIZE=10000    TEST_SIZE=1000    --input    ./sample/data    --output
./sample/output
```

2.6.6 Data formats and constraints

EXTRACTION:

(1) Input

All the documents, lexical table should be *UTF-8* encoded and in plain text. The sentences in comparable corpus should be split into lines.

(2) Output

In the output file, the list of extracted phrases is written in the form:

```
source sentence1
target sentence1
score1

source sentence2
target sentence2
score2
```

TRAINING:

(1)Input

Sample file name are described by 3 choices: *[train/test].[pos/neg].[source/target]*

8 combinations should be provided as *UTF-8* plain text for both training and test purpose. For each identical file pair (i.e. *train.pos.de* & *train.pos.en*), they should contain the same number lines of sentences/phrases, which indicate their parallelism/non-parallelism.

(2)Output

Two numeric values are printed as precision and recall.

One derived model is saved as "*Script/model*".

2.6.7 Integration with external tools

This tool is based on *maxent-2.1.1*: a simple *C++* library for maximum entropy classifiers, which was developed by *Tsujii laboratory* from *University of Tokyo*. Any further release/publication should include its LICENSE.

2.6.8 Contact

Further information and technical support, please email to Xiaojun Zhang: Xiaojun.Zhang@dfki.de.

2.6.9 Useful references

A simple *C++* library for maximum entropy classification, University of Tokyo, Tsuji Laboratory: http://www-tsujii.is.s.u-tokyo.ac.jp/~tsuruoka/maxent/.

D. S. Munteanu, D. Marcu, Improving Machine Translation Performance by Exploiting Non-Parallel Corpora, Computational Linguistics, volume 31, number 4, pp. 477-504, December 2005.

3 Tools for named entity recognition

This section covers the tools that perform named entity recognition and tools that are created to integrate out of ACCURAT project developed tools within the toolkit's general use case workflows.

The tools included in this section of the ACCURAT toolkit are:

- *TildeNER* (Latvian and Lithuanian named entity recognition developed by Tilde; see section 3.1).
- *OpenNLPWrapper* (*OpenNLP* English named entity recognition system's wrapper system developed by USFD; see section 3.2).
- *NERA1: Named Entity Recognition for English and Romanian (*developed by RACAI; see section 3.3*).*

3.1 TildeNER

3.1.1 Overview and purpose of the tool

TildeNER is a named entity recognition and classification system. The system contains workflows that allow not only named entity (NE) tagging of single files, but also pre-processing and post-processing of plaintext documents and even whole directories. The system also contains a heavily configurable bootstrapping module, which allows training, improvement and evaluation of new NE models if necessary. The system is originally designed and developed for Latvian and Lithuanian named entity recognition, but is not limited to the design languages; therefore new languages can be easily added with the included bootstrapping module. Some of the usage scenarios are described further. The system's core functionality – classification is done by the *Stanford NER* conditional random field (*CRF*) classifier (some minor changes have been made to the classifier in order to support additional feature functions and the *TildeNER* input and output data standards).

The tool allows tagging of plaintext or pre-processed tab-separated (tokenized, POS-tagged, lemmatized) documents and it allows the results to be saved in a *MUC-7* compliant plaintext mark-up or as tab-separated (tokenized, POS-tagged, lemmatized and NE-tagged) documents.

Named entity recognizers are widely used to improve search functionalities (for example, person, organization, location search), to extract keywords from text, to find non-dividable phrases in texts, etc. The latter is useful in machine translation and allows finding segments, which should be translated using specific methods, or that should not be translated at all or only transliterated (for example, person names, often locations).

The architecture of the *TildeNER* bootstrapping system is shown in Figure 4. The system requires *MUC-7* compliant annotated corpus (seed list, development data and test data) and an unlabelled data corpus in order to train a NER model (an annotation tool is included in the toolkit). Also gazetteer data can be provided (but is not mandatory) in order to train the system.

The system iteratively trains new NER models on the training data of the particular iteration. In the first iteration the training data is the seed data, but in the further iterations new data is acquired by tagging the unlabelled data corpus and selecting new candidate training sentences based on uniqueness constraints and sentence ranking. The sentences are ranked according to the classifier assigned NE token average probabilities. A threshold is used to control that low likelihood NE-tagged tokens do not get selected as new candidate training data.

In each of the iterations the system is evaluated on development and test data. We use the development data in order to fine tune the system. An option allows usage of only positive iteration candidate training data for further iterations (it has proven to give the best results).

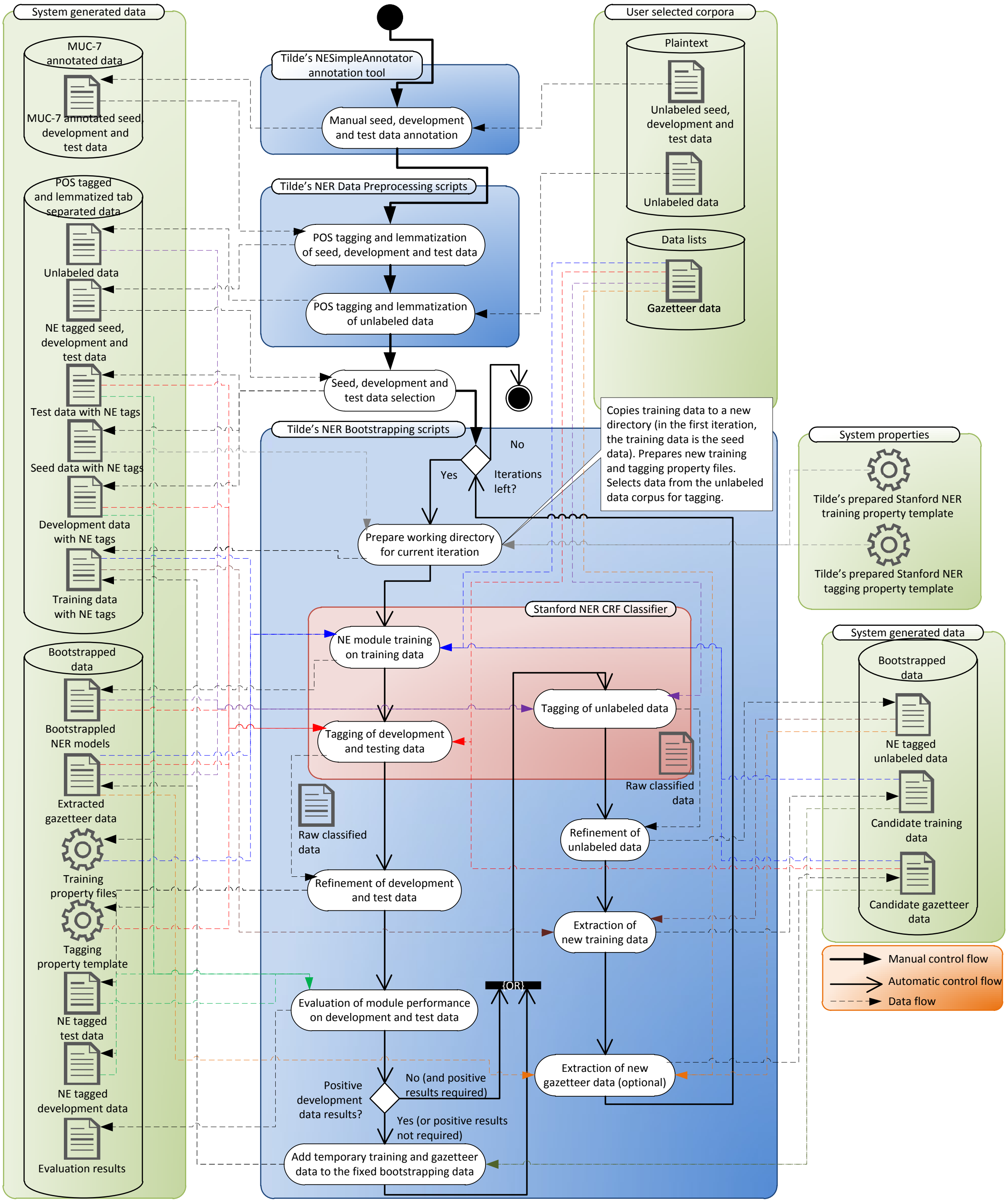

System generated data
MUC-7 annotated data
MUC-7 annotated seed, development and test data
POS tagged and lemmatized tab separated data
Unlabeled data
NE tagged seed, development and test data
Test data with NE tags
Seed data with NE tags
Development data with NE tags
Training data with NE tags
Bootstrapped data
Bootstrappled NER models
Extracted gazetteer data
Training property files
Tagging property template
NE tagged test data
NE tagged development data
Evaluation results
Tilde's NESimpleAnnotator annotation tool
Manual seed, development and test data annotation
Tilde's NER Data Preprocessing scripts
POS tagging and lemmatization of seed, development and test data
POS tagging and lemmatization of unlabeled data
Seed, development and test data selection
Tilde's NER Bootstrapping scripts
Iterations left?
Yes
No
Prepare working directory for current iteration
Copies training data to a new directory (in the first iteration, the training data is the seed data). Prepares new training and tagging property files. Selects data from the unlabeled data corpus for tagging.
Stanford NER CRF Classifier
NE module training on training data
Tagging of unlabeled data
Tagging of development and testing data
Raw classified data
Raw classified data
Refinement of unlabeled data
Refinement of development and test data
Extraction of new training data
OR
Evaluation of module performance on development and test data
Positive development data results?
No (and positive results required)
Yes (or positive results not required)
Extraction of new gazetteer data (optional)
Add temporary training and gazetteer data to the fixed bootstrapping data
User selected corpora
Plaintext
Unlabeled seed, development and test data
Unlabeled data
Data lists
Gazetteer data
System properties
Tilde's prepared Stanford NER training property template
Tilde's prepared Stanford NER tagging property template
System generated data
Bootstrapped data
NE tagged unlabeled data
Candidate training data
Candidate gazetteer data
Manual control flow
Automatic control flow
Data flow

Figure 4 TildeNER bootstrapping architecture.

The system allows also execution of several refinements (refer to section 3.1.5.4.3), which allow fine-tuning of the system towards increasing recall (increases also the F-measure) or precision of the system. Some of the refinements also try to correct corrupt NE tagging (missing quotation marks, web addresses incorrectly tagged as entities, etc.). The Latvian and Lithuanian models have been bootstrapped using fine-tuning for precision and using only positive iterations.

The system also allows automatic extraction of gazetteer data, which is then used in order to train new NER models in further iterations. In training of the Latvian NER model person name, common organization and location gazetteers have been used.

As a sample the Figure 5 shows the tagging workflow of a plaintext document. The results are saved in the *MUC-7* annotated data format.

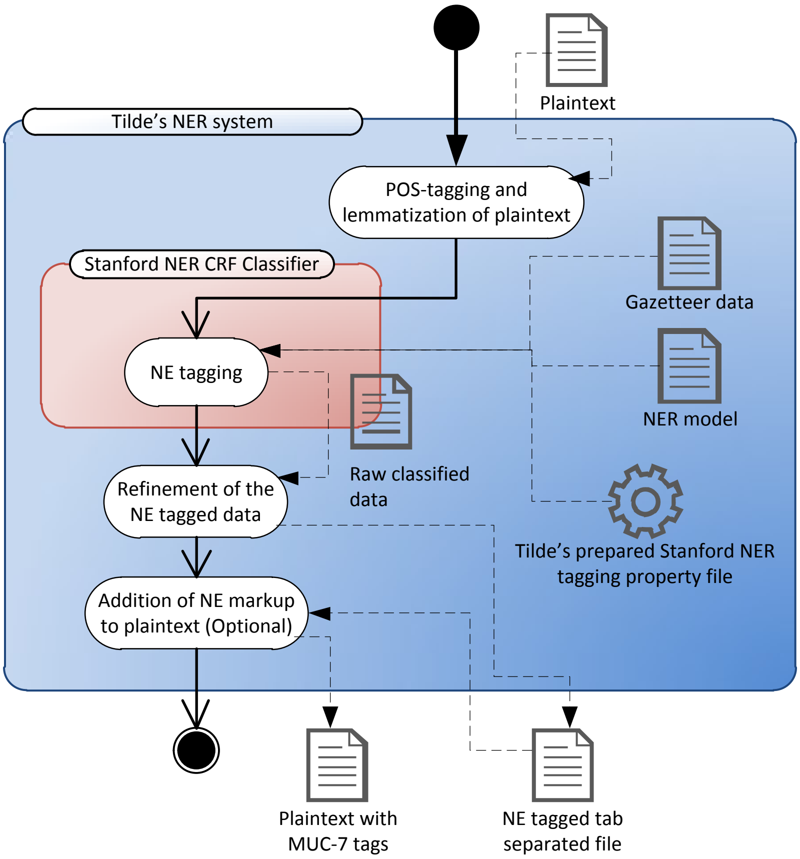

Plaintext
Tilde's NER system
POS-tagging and lemmatization of plaintext
Stanford NER CRF Classifier
Gazetteer data
NE tagging
NER model
Raw classified data
Refinement of the NE tagged data
Tilde's prepared Stanford NER tagging property file
Addition of NE markup to plaintext (Optional)
Plaintext with MUC-7 tags
NE tagged tab separated file

Figure 5 Sample workflow of plaintext to MUC-7 annotated data tagging.

3.1.2 Changes from previous version

Changes include minor bug fixes and usage samples ("*RUN*" scripts for faster testing and smaller learning curve for users). The system speed and data quality have not been affected by the changes.

3.1.3 Software dependencies and system requirements

TildeNER software dependencies are as follows:

- A modified version of *Stanford NER* (included in the toolkit)[9]
- *TreeTagger* (if the user wishes to train a non-Baltic language NER model)[10]
- *Tagger.exe* – the Tilde's Baltic language POS-tagging web service interface on Windows.
- *tagger.sh* – the Tilde's Baltic language POS-tagging web service interface on Linux.
- *Java Runtime Environment* (version 1.6.0)
- *Perl* (Windows - *Strawberry Perl* v5.12.1; *Linux – Perl* v5.10.1).

TildeNER system requirements are as follows:

- For training:
  - A *Linux* operating system (for instance, *Ubuntu 10.04.2*);
  - More than 4 GB RAM;
  - *Intel® Core™2* CPU (1.8 GHz for a single core) or faster.
- For tagging:
  - A *Linux* or *Windows* (*XP* or newer) operating system;
  - 2 or more (1.5 GB should be accessible) GB RAM;
  - *Intel® Pentium® 4* CPU 3.00GHz, 2992 Mhz, 1 Core(s), 2 Logical Processors or faster.

The system requirements shown are based on the *Linux* training system and a *Windows* based tagging system used for Latvian and Lithuanian model training and evaluation. Faster performance can be achieved using a faster system and for larger annotated corpora more RAM can be necessary (for the training stage).

The fast NE annotation tool included in the toolkit (*NESimpleAnnotator* – runs only on Windows) depends on:

- *Microsoft .NET Framework 4.0 Redistributable*

The system requirements for *NESimpleAnnotator* are as follows:

- *Windows* (*XP SP2* or newer) operating system;
- 2 or more GB RAM;

Intel® Pentium® 4 CPU 3.00GHz, 2992 Mhz, 1 Core(s), 2 Logical Processors or faster.

3.1.4 Installation

The *TildeNER* system does not require installation. Simply copy the whole "*TildeNER*" directory to a directory from where you would like to run the named entity recognizer and

[9] For commercial licensing please refer to http://otlportal.stanford.edu/techfinder/technology/ID=24628

[10] TreeTagger is available only for research, evaluation and teaching purposes as defined in the license http://www.ims.uni-stuttgart.de/~schmid/Tagger-Licence; for commercial application, the user will have to use a different POS-tagger.

execute the *Perl* workflow scripts whenever it is necessary using a *Perl* interpreter (for example, *Strawberry Perl* on *Windows*) from the command line (*Command Prompt* or *PowerShell* on *Windows* or any programming language that supports shell executions).

The user will have to create a property file in order to execute the training or tagging scripts. Sample property files are located within the "*Sample_Data*" subdirectory of the "*TildeNER*" directory. The user will also have to alter the "*gazette*" parameter in the property files so that the system is able to find the gazetteer data files in the user's system (the sample addresses are relative and will work only if the system's working directory will be the *TildeNER* root directory). The gazetteer files used for training of the Latvian and Lithuanian NER models are located in the "*LV_Gazetteer*" and "*LT_Gazetteer*" subdirectories of the "*Sample_Data*" directory. Please use the "/" directory separation character in gazetteer file addresses also on *Windows*. The *Stanford NER* classifier does not process the *Windows* directory separation character "\" correctly; therefore, the *Linux* variant should be used instead.

The Latvian and Lithuanian NER models and gazetteer data is located within the "*Sample_Data*" directory. More details on the provided sample data can be found in section 3.1.5.6.

Dependency installation on a *Linux* OS:

- For installation of *Perl* refer to http://www.perl.org/get.html.
- For installation of *Java* refer to http://openjdk.java.net/install/.

Dependency installation on *Windows* OS:

- For installation of *Perl* refer to http://strawberryperl.com/.
- For installation of *Java* refer to http://www.java.com/en/download/help/windows_manual_download.xml.

For installation of *.NET Framework 4.0 Redistributable* refer to http://www.microsoft.com/download/en/details.aspx?id=17718

3.1.5 Execution instructions

The *TildeNER* system consists of multiple workflows (external execution scripts), which create the general use case scenarios of the *TildeNER* system. Each of the workflows makes use of internal execution scripts (*see 3.1.5.3),* which are developed to offer partial workflow functionality and modules (*see 3.1.5.4*), which contain utility and functionality methods used by the scripts. All Perl modules and scripts have well documented code; therefore, if any additional questions arise, the user should refer to the comments within the code.

The system has two general use cases - the bootstrapping of a new NER model (see Figure 6) and tagging of a plaintext document (see Figure 7).

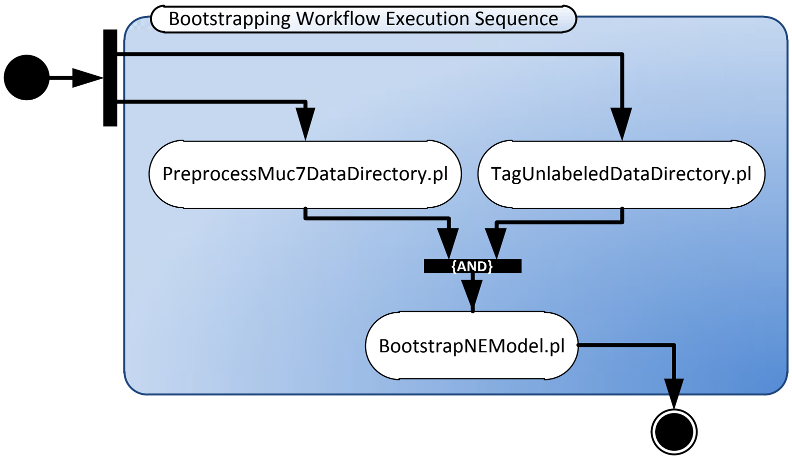

Bootstrapping Workflow Execution Sequence
PreprocessMuc7DataDirectory.pl
TagUnlabeledDataDirectory.pl
{AND}
BootstrapNEModel.pl

Figure 6 NER model bootstrapping execution sequence.

It is necessary for the user to prepare and pre-process the seed, development, test and unlabelled data corpora, therefore, the pre-processing scripts have to be executed prior to the bootstrapping script. If the user uses his/her own pre-processing tools, which produce compliant input data (refer to section 3.1.6.3for more details), the pre-processing scripts may also be skipped.

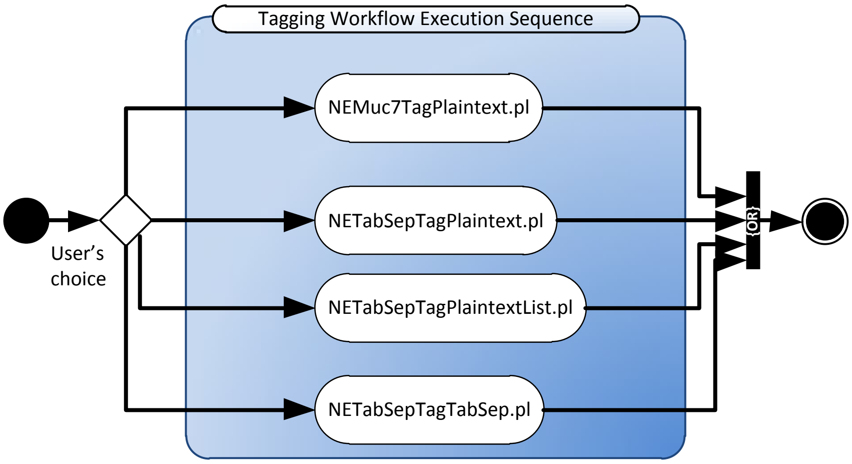


Figure 7 Single document tagging execution sequence.

The *TildeNER* system offers three different tagging workflows; therefore, the user has to choose the appropriate tagging script according to the provided input data and the required output data.

3.1.5.1 NESimpleAnnotator

As the *TildeNER* system requires annotated seed, development and test data in order to train a NER model and tune the system for best performance, an annotation tool was developed in order to allow fast annotation of plaintext documents (refer to section 3.1.6.1 for a format description). The tool saves the annotated documents in the *MUC-7* annotated data format described in section 3.1.6.2.

For user manual and named entity annotation guidelines refer to the document "***NE Markup Guidelines.docx***" that can be found in the "*NESimpleAnnotator*" subdirectory of the

"*TildeNER*" directory. The annotation tool ("*NESimpleAnnotator.exe*") can be found in the same directory.

3.1.5.2 External execution scripts

The external execution scripts provide the main functionality of the *TildeNER* system. The scripts are also part of the general use case scenarios. In order to provide assistance in execution of the scripts the *TildeNER* package contains predefined *Bash* ("*sh*"; for *Linux*) and *Batch* ("*bat*"; for *Windows*) scripts in the form "*RUN-###.bat*" or "*RUN-###.sh*" (where "*###*" is the name of the external execution script, which command is executed by the script, for instance, "*RUN-PreprocessMuc7DataDirectory.bat*"). The scripts make use of sample property files, models and gazetteer files in the "*Sample_Data*" directory and the input data (and also output data after execution) in the "*TEST*" directory. The scripts operate on data in Latvian (the user has to make modifications to the scripts and provide additional data for other language support).

3.1.5.2.1 Training, development and testing data pre-processing

To train a new model for an existing (Latvian, Lithuanian) or a new language, it is necessary to provide training, testing and development data. It is up to the user to decide, how to divide a named entity annotated corpus, but once the annotated data is created (and the format is compliant to the *MUC-7* annotated data specified in 3.1.6.2), the user can use the script "***PreprocessMuc7DataDirectory.pl***" to perform all required data pre-processing.

The script performs data pre-processing on a single directory (subdirectories are not processed) that contains *MUC-7* annotated documents. For each file it separates the NE annotation from the plaintext, tokenizes, POS-tags and lemmatizes the plaintext and combines the tab-separated outcome of the plaintext with the separated NE annotation in a tab-separated data file (see 3.1.6.3 for the data format description).

The command line to call the pre-processing for a single directory is as follows:

```
perl ./PreprocessMuc7DataDirectory.pl [1: Input directory] [2: Output directory] [3: Input file extension] [4: Output file extension] [5: Language] [6: POS-tagger]
```

The script requires in total six arguments passed to the script in a fixed order:

1. The source (input) data directory path.
2. The target (output) data directory path.
3. The input file extension (suggested is "*txt*" for *MUC-7* annotated plaintext).
4. The output file extension (suggested is "*gold*" for human annotated data).
5. The language of the input documents. The language has to be supported by the POS-tagger.
6. The POS-tagger to use for pre-processing.

Available POS-tagger and language pairs are defined in 3.1.5.5. For information on how to add other POS-taggers refer to section 3.1.7.

The script depends on “***ProcessDirectory.pl***” and “***PrepareNEData.pl***” scripts and in a general use case has to be executed only once – to prepare annotated data.

For testing purposes and to provide execution examples “*RUN-PreprocessMuc7DataDirectory.bat*” (*Windows*) and “*RUN-PreprocessMuc7DataDirectory.sh*” (*Linux*) scripts are provided. The scripts are preconfigured to execute “*PreprocessMuc7DataDirectory.pl*” on *MUC-7* annotated documents (with “*txt*” extensions) located in directory “*./TEST/gold_muc7_plaintext_in*” using the POS-tagger “*Tagger*” for Latvian “*lv*”. Results will be saved in “*./TEST/gold_tab_sep_out*”.

3.1.5.2.2 Unlabeled data pre-processing

If the user wants to train a named entity recognition model using bootstrapping, it is required to pre-process all unlabelled data documents. The script “***TagUnlabeledDataDirectory.pl***” performs data pre-processing on a single directory (subdirectories are not processed) that contains plaintext documents. For each file it executes tokenization, lemmatization and POS-tagging and saves the results in tab-separated output file (see 3.1.6.4 for the data format description).

The command line to call the pre-processing for a single directory is as follows:

```
perl ./TagUnlabeledDataDirectory.pl [1: Language] [2: POS-tagger] [3: Input
directory] [4: Output directory] [5: Input file extension] [6: Output file
extension] [7: Delete (or not) temporary files]
```

The script requires in total seven arguments passed to the script in a fixed order:

1. The language of the plaintext documents. The language has to be supported by the POS-tagger.
2. The POS-tagger to use for pre-processing.
3. The source (input) data directory path.
4. The target (output) data directory path.
5. The input file extension (suggested is “*txt*” for plaintext).
6. The output file extension (suggested is “*pos*” for unlabeled data).
7. Indicator, whether to delete temporary files (deletes temporary files if "1", keeps if "0").

Available POS-tagger and language pairs are defined in 3.1.5.4.5. For information how to add other POS-taggers refer to section 3.1.7.

The script depends on the “***Tag.pm***” module and in a general use case has to be executed only once – to prepare unlabeled data.

For testing purposes and to provide execution examples “*RUN-TagUnlabeledDataDirectory.bat*” (*Windows*) and “*RUN-TagUnlabeledDataDirectory.sh*” (*Linux*) scripts are provided. The scripts are preconfigured to execute “*TagUnlabeledDataDirectory.pl*” on unlabeled plaintext documents (with “*txt*” extensions) located in directory “*./TEST/plaintext_in*” using the POS-tagger “*Tagger*” for Latvian “*lv*”. Results will be saved in “*./TEST/unannotated_tab_sep_out*”.

3.1.5.2.3 Training a Named Entity Recognition Model Using Bootstrapping

Once the user has pre-processed training (initially seed data), development, test and unlabelled data, the bootstrapping of a new NER model can be started. The script "***BootstrapNEModel.pl***" iteratively trains NER models on training data, in each iteration evaluates current model's performance on development and test data, tags the unlabelled data (a selected subsection in each iteration) and extracts a maximum number of new top ranked unique training samples for the next iteration.

The uniqueness constraint used is the uniqueness of sentence morpho-syntactic tag sequences. Refer to section 3.1.6.3 for an example of Tilde's morpho-syntactic tag and what to do if the user's POS-tagger does not assign morho-syntactic tags to tokens.

The bootstrapping script also offers gazetteer automatic extraction from the unannotated corpora and usage of only positive (increase the results on development data) iterations in new training data extraction.

The command line to start the bootstrapping is as follows:

```
perl ./BootstrapNEModel.pl [1: Seed list directory] [2: Seed file
extension] [3: Development list directory] [4: Development file extension]
[5: Test list directory] [6: Test file extension] [7: Unlabelled data
directory] [8: Unlabelled file extension] [9: Training property template]
[10: Tagging property template] [11: Working directory] [12: Number of
iterations] [13: Unlabelled documents to tag] [14: Sentences to select per
NE tag] [15: Refinement order definition string] [16: Bootstrapped
gazetteer file] [17: Use only positive iterations] [18: Positive iterator
condition]
```

The script requires in total eighteen arguments passed to the script in a fixed order (the last three are optional):

1. The seed list directory path.
2. The seed list data file extension (suffix before the point).
3. The development data directory path.
4. The development list data file extension (suffix before the point).
5. The test data directory path.
6. The test list data file extension (suffix before the point).
7. The unlabelled data directory path.
8. The unlabelled list data file extension (suffix before the point).
9. The path of the training property template (*Stanford NER*). The template defines, which feature functions to use in training, and contains a list of *Stanford NER* system properties. It should not contain entries of seed list data files as the template will be changed by the script in all iterations. A sample training property template ("*LV_Training_prop_template.prop*") can be found in the "*Sample_Data*" subdirectory of the "*TildeNER*" directory.
10. The path of the tagging property template (*Stanford NER*). The template defines, which feature functions to use when tagging documents, and contains a list of *Stanford NER* system properties. A sample tagging property template

("*LV_Tagging_prop_template.prop*") can be found in the "*Sample_Data*" subdirectory of the "*TildeNER*" directory.

11. The working directory where all results of all iterations will be stored. This should be an empty directory as all existing files will be overwritten and none non-conflicting files will be deleted (this could cause the system to work incorrectly if wrong training or gazetteer data would be present).
12. Bootstrapping iteration amount (for Latvian and Lithuanian *200* were used as a maximum).
13. The number of unlabelled documents to be tagged and processed in a single bootstrapping iteration. For Latvian and Lithuanian "500" was used.
14. The number of sentences to extract as new training data for the next iteration for each NE tag. For Latvian and Lithuanian "30" was used.
15. The refinement order definition string – defines which and in which order refinements are executed on NE tagged data. For more information on the refinement order definition string refer to section 3.1.5.4.3. For Latvian "*L N S R_0.7 C T_0.90 A*" was used to run bootstrapping in order to improve precision and "*L N S R_0.4 T_0.70 A*" was used for F-measure.
16. The path of the gazetteer file for extracted named entity samples. If the user does not want automatic gazetteers to be extracted an empty value (*""*) may be passed instead.
17. The indicator, whether to use only positive bootstrapping iterations. "1" should be passed if only positive iterations should be used.
18. The positive iteration condition. The value is used only if the previous parameter is set to "1". Allowed values are:
    a. "*P*" for precision ("*P*" means that only those iterations will be considered positive, in which precision will increase over the last best precision).
    b. "R" for recall.
    c. "F" for F-measure.
    d. "A" for accuracy.
    e. Everything else means that all values will be required to increase for a positive iteration.

The script will store each NE model in its own iteration directory and also produce a combined result file for all iterations. It is up to the user to decide, which model, from which iteration to use further after the bootstrapping will be finished.

The script depends on the "***NETrainAndEvaluate.pl***" and "***NETagDirectory.pl***" scripts and "***BootstrapTools.pm***" and "***NEUtilities.pm***" modules. The script will have to be run multiple times if the user wishes to test various configuration options.

For testing purposes and to provide execution examples the "*RUN-BootstrapNEModel.sh*" (*Linux*) script is provided. A *Windows* version is not included as training requires a *Linux* operating system. The script is preconfigured to execute "*BootstrapNEModel.pl*" so that seed data is taken from "*./TEST/seed_in*" (file extension – "*gold*"), development data is taken from "*./TEST/dev_in*" (file extension – "*gold*"), test data is taken from "*./TEST/gold_tab_sep_in*" (file extension – "*gold*") and the unannotated data is taken from "*./TEST/unannotated_tab_sep_in*" (file extension – "*pos*"). The script makes use of the sample property templates located in "*./Sample_Data/LV_Training_prop_template.prop*" and "*./Sample_Data/LV_Tagging_prop_template.prop*". The working directory is set to "*./TEST/bootstrap_out*". For system testing purposes the bootstrapping iteration amount is limited to *5*, unlabeled document amount is limited to *5* and only one sentence will be executed in each bootstrapping run for every NE category. The refinement order definition string is set to "*L N S R_0.7 C T_0.90 A*", which will raise precision. The extracted gazetteer data will be saved in "*./TEST/bootstrap_out/bootstrapped_gazetteer.txt*". The example script does not make use of the positive iteration functionality.

3.1.5.2.4 Plaintext to MUC-7 document tagging

Once the user has acquired a trained named entity recognition model for the required language, the tagging scripts may be executed. The first of the tagging scripts is the "***NEMuc7TagPlaintext.pl***" script, which tags a plaintext document (for the format refer to section 3.1.6.1) for named entities and saves the result as a plaintext document with named entities marked with *MUC-7* tags (for the format refer to section 3.1.6.2).

The command line to call the NE-tagging for a single plaintext file is as follows:

```
perl ./NEMuc7TagPlaintext.pl [1: NER model path] [2: Plaintext input file]
[3: MUC-7 output file] [4: Tagging property file] [5: Language] [6: POS-
tagger] [7: Keep temporary files] [8: Refinement order definition string]
```

The script requires in total eight arguments passed to the script in a fixed order:

1. The path to the NER model. Latvian and Lithuanian trained models are available in the "*Sample_Data*" subdirectory of the "*TildeNER*" directory – "*LV_Model_P.ser.gz*" (Latvian bootstrapped for increased precision), "*LV_Model_F.ser.gz*" (increased F-measure), "*LT_BASELINE_Model.ser.gz*" (The baseline Lithuanian NER model) and "*LT_Model_F.ser.gz*" (Lithuanian bootstrapped for increased F-measure). The Lithuanian baseline model already shows high precision (with applied refinement methods) and the bootstrapping did not result in increased results (therefore, the baseline model is included).
2. The path of the plaintext file, which has to be tagged.
3. The path of the *MUC-7* annotated output file (an existing file will be overwritten).
4. The Stanford NER tagging property file. Sample tagging property file ("*LV_P_Tagging_prop_sample.prop*" (for the precision increasing model) or "*LV_F_Tagging_prop_sample.prop*" (for the F-measure increasing model)) can be found in the "*Sample_Data*" subdirectory of the "*TildeNER*" directory (note that this is not the same as the template property file as used in bootstrapping!).
5. The language of the plaintext document. The language has to be supported by the POS-tagger.
6. The POS-tagger to use for pre-processing of the plaintext document.

7. Indicator, whether to keep temporary files. If "*1*", temporary files will be kept. Any other value means that temporary files will be deleted.
8. The refinement order definition string – defines which and the order in which refinements are executed on NE tagged data. For more information on the refinement order definition string refer to section 3.1.5.4.3. For Latvian "*L N S R_0.7 C T_0.90 A*" achieves the highest precision and "*L N S R_0.4 T_0.70 A*" achieves the highest F-measure (the respective (see point 1) NER models and property files (see point 4) have to be used to achieve the best required results).

Available POS-tagger and language pairs are defined in section 3.1.5.4.5. For information how to add other POS-taggers refer to section 3.1.7.

The script depends on the "***NEPreprocess.pm***", "***Tag.pm***" and "***NERefinements.pm***" Perl modules and the *Stanford NER* module "***stanford-ner.jar***". The script will have to be run once for each plaintext document.

For testing purposes and to provide execution examples the "*RUN-NEMuc7TagPlaintext.bat*" (*Windows*) and "*RUN-NEMuc7TagPlaintext.sh*" (*Linux*) scripts are provided. The scripts are preconfigured to execute "*NEMuc7TagPlaintext.pl*" so that input data is taken from the file "*./TEST/plaintext_in.txt*", the "*./Sample_Data/LV_Model_P.ser.gz*" NER model and the "*./Sample_Data/LV_P_Tagging_prop_sample.prop*" property file are used in tagging, the POS-tagger "*Tagger*" for Latvian ("*lv*") is used and the refinement order definition string is set to "*L N S R_0.7 C T_0.90 A*", which will raise precision. The results will be saved in "*./TEST/muc-7_plaintext_out.txt*".

3.1.5.2.5 Plaintext to MUC-7 document list tagging

The ACCURAT workflow for NE/Term mapping allows NE tagging of lists of files. Therefore, the script "***NEMuc7TagPlaintextList.pl***" was created. The script tags each plaintext document (for the format refer to section 3.1.6.1) specified in an I/O document pair list (for the format refer to section 3.1.6.7) and saves each plaintext document with named entities marked with *MUC-7* tags (for the format refer to section 3.1.6.2) in files also specified by the d document pair list.

The command line to call the NE-tagging using a document pair list file is as follows:

```
perl ./NEMuc7TagPlaintextList.pl [1: NER model path] [2: Document pair list file] [3: Tagging property file] [4: Language] [5: POS-tagger] [6: Refinement order definition string]
```

The script requires in total six arguments passed to the script in a fixed order:

1. The path to the NER model. Latvian and Lithuanian trained models are available in the "*Sample_Data*" subdirectory of the "*TildeNER*" directory – "*LV_Model_P.ser.gz*" (Latvian bootstrapped for increased precision), "*LV_Model_F.ser.gz*" (increased F-measure), "*LT_BASELINE_Model.ser.gz*" (The baseline Lithuanian NER model) and "*LT_Model_F.ser.gz*" (Lithuanian bootstrapped for increased F-measure). The Lithuanian baseline model already shows high precision (with applied refinement methods) and the bootstrapping did not result in increased results (therefore, the baseline model is included).

2. The path of the I/O document pair list file (for the format refer to section 3.1.6.7). Each line of the document contains two tab-separated (“*\t*” character) entries – the plaintext input file (see section 3.1.6.1) and the *MUC-7* annotated output file (see section 3.1.6.2).
3. The Stanford NER tagging property file. Sample tagging property file (“*LV_P_Tagging_prop_sample.prop*” (for the precision increasing model) or “*LV_F_Tagging_prop_sample.prop*” (for the F-measure increasing model)) can be found in the “*Sample_Data*” subdirectory of the “*TildeNER*” directory (note that this is not the same as the template property file as used in bootstrapping!).
4. The language of the plaintext document. The language has to be supported by the POS-tagger.
5. The POS-tagger to use for pre-processing of the plaintext document.
6. The refinement order definition string – defines which and the order in which refinements are executed on NE tagged data. For more information on the refinement order definition string refer to section 3.1.5.4.3. For Latvian “*L N S R_0.7 C T_0.90 A*” achieves the highest precision and “*L N S R_0.4 T_0.70 A*” achieves the highest F-measure (the respective (see point 1) NER models and property files (see point 4) have to be used to achieve the best required results).

Available POS-tagger and language pairs are defined in section 3.1.5.4.5. For information how to add other POS-taggers refer to section 3.1.7.

The script depends on the “***NEMuc7TagPlaintext.pl***” script.

For testing purposes and to provide execution examples the “*RUN-NEMuc7TagPlaintextList.bat*” (*Windows*) and “*RUN-NEMuc7TagPlaintextList.sh*” (*Linux*) scripts are provided. The scripts are preconfigured to execute “*NEMuc7TagPlaintextList.pl*” so that input data is taken from the file “*./TEST/plaintextList_in.txt*”, the “*./Sample_Data/LV_Model_P.ser.gz*” NER model and the “*./Sample_Data/LV_P_Tagging_prop_sample.prop*” property file are used in tagging, the POS-tagger “*Tagger*” for Latvian (“*lv*”) is used and the refinement order definition string is set to “*L N S R_0.7 C T_0.90 A*”, which will raise precision.

3.1.5.2.6 Plaintext to tab-separated document tagging

The second of the tagging scripts is the “***NETabSepTagPlaintext.pl***” script, which tags a plaintext document (for the format refer to section 3.1.6.1) for named entities and saves the result as a tab-separated, tokenized, POS-tagged, lemmatized and NE-tagged document (for the format refer to section 3.1.6.5).

The command line to call the NE-tagging for a single plaintext file is as follows:

```
perl ./NETabSepTagPlaintext.pl [1: NER model path] [2: Plaintext input
file] [3: Tab-separated output file] [4: Tagging property file] [5:
Language] [6: POS-tagger] [7: Keep temporary files] [8: Refinement order
definition string]
```

The script requires in total eight arguments passed to the script in a fixed order:

1. The path to the NER model. Latvian and Lithuanian trained models are available in the "*Sample_Data*" subdirectory of the "*TildeNER*" directory – "*LV_Model_P.ser.gz*" (Latvian bootstrapped for increased precision), "*LV_Model_F.ser.gz*" (increased F-measure), "*LT_BASELINE_Model.ser.gz*" (The baseline Lithuanian NER model) and "*LT_Model_F.ser.gz*" (Lithuanian bootstrapped for increased F-measure). The Lithuanian baseline model already shows high precision (with applied refinement methods) and the bootstrapping did not result in increased results (therefore, the baseline model is included).
2. The path of the plaintext file, which has to be tagged.
3. The path of the tab-separated output file (an existing file will be overwritten).
4. The Stanford NER tagging property file. A sample tagging property file ("*LV_P_Tagging_prop_sample.prop*" (for the precision increasing model) or "*LV_F_Tagging_prop_sample.prop*" (for the F-measure increasing model)) can be found in the "*Sample_Data*" subdirectory of the "*TildeNER*" directory.
5. The language of the plaintext document. The language has to be supported by the POS-tagger.
6. The POS-tagger to use for pre-processing of the plaintext document.
7. Indicator, whether to keep temporary files. If "1", temporary files will be kept. Any other value means that temporary files will be deleted.
8. The refinement order definition string – defines which and the order in which refinements are executed on NE tagged data. For more information on the refinement order definition string refer to section 3.1.5.4.3. For Latvian "*L N S R_0.7 C T_0.90 A*" achieves the highest precision and "*L N S R_0.4 T_0.70 A*" achieves the highest F-measure (the respective (see point 1) NER models and property files (see point 4) have to be used to achieve the best required results).

Available POS-tagger and language pairs are defined in 3.1.5.4.5. For information how to add other POS-taggers refer to section 3.1.7.

The script depends on the "***Tag.pm***" and "***NERefinements.pm***" Perl modules and the Stanford NER module "***stanford-ner.jar***". The script will have to be run once for each plaintext document.

For testing purposes and to provide execution examples the "*RUN-NETabSepTagPlaintext.bat*" (*Windows*) and "*RUN-NETabSepTagPlaintext.sh*" (*Linux*) scripts are provided. The scripts are preconfigured to execute "*NETabSepTagPlaintext.pl*" so that input data is taken from the file "*./TEST/plaintext_in.txt*", the "*./Sample_Data/LV_Model_P.ser.gz*" NER model and the "*./Sample_Data/LV_P_Tagging_prop_sample.prop*" property file are used in tagging, the POS-tagger "*Tagger*" for Latvian ("*lv*") is used and the refinement order definition string is set to "*L N S R_0.7 C T_0.90 A*", which will raise precision. The results will be saved in "*./TEST/tab_sep_out.txt*".

3.1.5.2.7 *Tab-separated document to tab-separated document* tagging

The third (and last one) of the tagging scripts is the "***NETabSepTagTabSep.pl***" script, which tags named entities in an already pre-processed (for instance, with the script "***TagUnlabeledDataDirectory.pl***") document (for the format refer to section 3.1.6.4) and saves the result as a tab-separated, tokenized, POS-tagged, lemmatized and NE-tagged document (for the format refer to section 3.1.6.5).

The command line to call the NE-tagging for a single plaintext file is as follows:

```
perl NETabSepTagTabSep.pl [1: NER model path] [2: Tab-separated input file]
[3: Tab-separated output file] [4: Tagging property file] [5: Keep
temporary files] [6: Refinement order definition string]
```

The script requires in total six arguments passed to the script in a fixed order:

1. The path to the NER model. Latvian and Lithuanian trained models are available in the "*Sample_Data*" subdirectory of the "*TildeNER*" directory – "*LV_Model_P.ser.gz*" (Latvian bootstrapped for increased precision), "*LV_Model_F.ser.gz*" (increased F-measure), "*LT_BASELINE_Model.ser.gz*" (The baseline Lithuanian NER model) and "*LT_Model_F.ser.gz*" (Lithuanian bootstrapped for increased F-measure). The Lithuanian baseline model already shows high precision (with applied refinement methods) and the bootstrapping did not result in increased results (therefore, the baseline model is included).
2. The path of the tab-separated input file, which has to be tagged.
3. The path of the tab-separated output file (an existing file will be overwritten).
4. The Stanford NER tagging property file. A sample tagging property file ("*LV_P_Tagging_prop_sample.prop*" (for the precision increasing model) or "*LV_F_Tagging_prop_sample.prop*" (for the F-measure increasing model)) can be found in the "*Sample_Data*" subdirectory of the "*TildeNER*" directory.
5. Indicator, whether to keep temporary files. If "1", temporary files will be kept. Any other value means that temporary files will be deleted.
6. The refinement order definition string – defines which and the order in which refinements are executed on NE tagged data. For more information on the refinement order definition string refer to section 3.1.5.4.3. For Latvian "*L N S R_0.7 C T_0.90 A*" achieves the highest precision and "*L N S R_0.4 T_0.70 A*" achieves the highest F-measure (the respective (see point 1) NER models and property files (see point 4) have to be used to achieve the best required results).

The script depends on the "***NEPreprocess.pm***" and "***NERefinements.pm***" Perl modules and the Stanford NER module "***stanford-ner.jar***". The script will have to be run once for each tab-separated document.

For testing purposes and to provide execution examples the "*RUN-NETabSepTagTabSep.bat*" (*Windows*) and "*RUN-NETabSepTagTabSep.sh*" (*Linux*) scripts are provided. The scripts are preconfigured to execute "*NETabSepTagTabSep.pl*" so that input data is taken from the file "*./TEST/tab_sep_in.pos*", the "*./Sample_Data/LV_Model_P.ser.gz*" NER model and the

“*./Sample_Data/LV_P_Tagging_prop_sample.prop*” property file are used in tagging and the refinement order definition string is set to “*L N S R_0.7 C T_0.90 A*”, which will raise precision. The results will be saved in “*./TEST/tab_sep_out.pos*”.

3.1.5.3 Internal execution scripts

The internal execution scripts define a set of scripts that do not require to be manually executed by the user, but in some cases may be helpful to the user (if an alternative use case is required). Although, the external execution scripts provided execution examples with “*bat*” and “*sh*” scripts, the internal scripts do not provide such examples and are meant to be executed by a more advanced user with more knowledge on *Perl*. All of the following internal execution scripts are integrated within the external execution scripts.

3.1.5.3.1 Executing a process on a directory

Many processes within the general use case scenarios require a single process to be executed on all files within a directory. Therefore, the script “***ProcessDirectory.pl***” is provided, which can execute any process, which requires one input file and one output file and additional parameters on a whole directory.

The command line to execute a process on a directory is as follows:

```
perl ProcessDirectory.pl [1: Input directory] [2: Output directory] [3:
Input file extension] [4: Output file extension] [5: Process to execute]
[6: Middle parameters] [7: End parameters]
```

The script requires in total seven arguments passed to the script in a fixed order (the last two are optional):

1. The input directory from which to read files.
2. The output directory to which the process will write files.
3. The input file extension (suffix before the point). Only the files with the correct extension will be processed.
4. The output file extension (suffix before the point).
5. The process to run (with before input file parameters).
6. The parameters between input and output files (optional).
7. The parameters after the output file (optional).

For each file in the input directory Perl executes the following command:

```
`[Process to execute] "[Input file]" [Middle parameters] "[Output file]"
[End parameters]`
```

This means that only those processes are supported, which require parameters to be passed in the specified order.

3.1.5.3.2 Pre-processing a single MUC-7 annotated document

In order to pre-process *MUC-7* annotated data for training, each document has to be processed with the script “***PrepareNEData.pl***” (integrated within the general use case in section 3.1.5.2.1). For each *MUC-7* annotated document (the format is specified in 3.1.6.2) the script separates the NE annotation from the plaintext, tokenizes, POS-tags and lemmatizes the plaintext and combines the tab-separated outcome of the plaintext with the separated NE annotation in a tab-separated data file (see 3.1.6.3 for the data format description).

The command line to call the pre-processing for a single file is as follows:

```
perl PrepareNEData.pl [1: Language] [2: POS-tagger] [3: Input file] [4: Output file] [5: Delete temp files]
```

The script requires in total five arguments passed to the script in a fixed order (the last one is optional):

1. The language of the input document. The language has to be supported by the POS-tagger.
2. The POS-tagger to use for pre-processing.
3. The input file path.
4. The output file path.
5. Indicator, whether to delete temporary files. "*-D*" means that temporary files will be deleted.

The script depends on "***Tag.pm***" and "***NEPreprocess.pm***" modules.

3.1.5.3.3 Training and evaluating a single NER model

Training and evaluation of a single NER module within a single bootstrapping iteration is done by the script "***NETrainAndEvaluate.pl***". It can also be used to individually train NER models without bootstrapping. Training, development and test data formats are defined in section 3.1.6.3. The tagging result data formats are defined in section 3.1.6.5. The gazetteer data format (if the user requires gazetteers to be used in training and tagging within the *Stanford NER* property template) is defined in section 3.1.6.6.

The command line to call training and evaluation for a single NER model is as follows:

```
perl ./NETrainAndEvaluate.pl [1: Training list directory] [2: Test list directory] [3: Development list directory] [4: Training file extension] [5: Test file extension] [6: Development file extension] [7: Working directory] [8: Training property template] [9: Tagging property template] [10: Refinement order definition string]
```

The script requires in total ten arguments passed to the script in a fixed order:

1. The training data directory path.
2. The test data directory path.
3. The development data directory path.
4. The training list data file extension (suffix before the point).
5. The test list data file extension (suffix before the point).
6. The development list data file extension (suffix before the point).
7. The working directory where all results will be stored. This should be an empty directory as all existing files will be overwritten and none non-conflicting files will be deleted (this could cause the system to work incorrectly if wrong training or gazetteer data would be present).

8. The path of the training property template (*Stanford NER*). The template defines, which feature functions to use in training, and contains a list of *Stanford NER* system properties. It should not contain entries of training data files as the template will be changed by the script. A sample training property template ("*LV_Training_prop_template.prop*") can be found in the "*Sample_Data*" subdirectory of the "*TildeNER*" directory.
9. The path of the tagging property template (*Stanford NER*). The template defines, which feature functions to use when tagging documents, and contains a list of *Stanford NER* system properties. A sample tagging property template ("*LV_Tagging_prop_template.prop*") can be found in the "*Sample_Data*" subdirectory of the "*TildeNER*" directory.
10. The refinement order definition string – defines which and in which order refinements are executed on NE tagged data. For more information on the refinement order definition string refer to section 3.1.5.4.3. For Latvian "*L N S R_0.7 C T_0.90 A*" achieves the best precision and "*L N S R_0.4 T_0.70 A*" achieves the best F-measure.

The script depends on the "***NEUtilities.pm***" module, "***NETagDirectory.pl***" script and the *Stanford NER* module "***stanford-ner.jar***".

3.1.5.3.4 Evaluating a NER system

It is important to evaluate a system when training a new NER model in order to evaluate its performance in comparison with different systems/NER models. Therefore, the script "***NEEvaluation_v2.pl***" has been developed. The script evaluates the precision, recall, accuracy and F-measure ($F = 2 \times \frac{P \times R}{P + R}$) of all named entity token categories (*B-ORG*, *I-ORG*, etc.), all full named entities (*LOCATION*, *ORGANIZATION*, etc.) and the total (average system performance) for single tokens (*TOTAL_TOKEN*) and full named entities (*TOTAL_NE*) by providing two directories – a gold data directory (for data formats refer to section 3.1.6.3) and a test result data directory (for data formats refer to section 3.1.6.5). The script requires for the directories to have equal file names (extensions/suffixes before the dot in file names may differ). A file is produced, which contains evaluation results. A sample file contents is as follows:

```
TOTAL_NE     39.81   73.95   -       51.76
MONEY        27.03   56.34   -       36.53
LOCATION     53.11   72.32   -       61.24
PERSON       48.00   90.00   -       62.61
ORGANIZATION 23.83   78.23   -       36.53
DATE         41.52   74.47   -       53.31
TIME         7.77    66.67   -       13.92
PRODUCT      26.77   54.84   -       35.98
TOTAL_TOKEN  45.65   84.01   89.54   59.16
B-MON        43.24   90.14   99.82   58.44
I-DATE       71.63   93.15   98.75   80.98
```

```
B-LOC         59.89   81.54   98.45   69.06
I-PERS        65.43   89.83   99.66   75.71
I-LOC         13.60   54.20   98.98   21.74
B-PERS        48.50   90.94   99.33   63.26
I-ORG         23.60   71.35   98.19   35.47
I-TIME        26.14   86.79   99.73   40.18
B-ORG         27.36   89.80   98.54   41.94
B-DATE        46.26   82.98   98.94   59.40
B-TIME        10.68   91.67   99.81   19.13
I-MON         42.27   87.58   99.60   57.02
I-PROD        26.00   57.96   99.35   35.90
B-PROD        29.40   60.22   99.32   39.51
```

The columns in the tab separated result file represent the following in the exact sequence: result category, recall, precision, accuracy and F-measure. For full named entities accuracy results will not be given (accuracy can be estimated on single token performance only and not on multiple token sequences as the interpretation of non-entities and their possible sequences is ambiguous).

The command line to call the evaluation script is as follows:

```
perl ./NEEvaluation_v2.pl [1: Gold data directory] [2: Test result
directory] [3: Output file]
```

The script requires in total three arguments passed to the script in a fixed order:

1. The path of the directory containing the human annotated/gold documents.
2. The path of the directory containing the test result documents.
3. The path to the evaluation result output file.

The script depends on the "***NEUtilities.pm***" Perl module.

3.1.5.3.5 Tagging and evaluating files in a directory

As the bootstrapping and NE training scripts require tagging of multiple full directories of files (development, test and unlabelled data), the script "***NETagDirectory.pl***" is provided. The script executes *Stanford NER* NE classification, NE refinements and also evaluation (optional) on all files in a directory. The files have to be pre-processed (for input data formats refer to section 3.1.6.3 and 3.1.6.4). The script creates tab-separated NE-tagged files (for the output data format refer to section 3.1.6.5).

The command line to call the NE-tagging for a single directory is as follows:

```
perl ./NETagDirectory.pl [1: NER model path] [2: Input directory] [3:
Output directory] [4: Input file extension] [5: Output file extension] [6:
Tagging property file] [7: Evaluation result file] [8: Refinement order
definition string]
```

The script requires in total eight arguments passed to the script in a fixed order (the last two are optional):

1. The path to the NER model.
2. The directory from which to read the tab-separated pre-processed files.
3. The directory to which the NE-tagged tab-separated files will be written.
4. The extension (suffix before the dot) of the input files.
5. The extension (suffix before the dot) of the output files.
6. The path to the NE tagging properties file.
7. The evaluation file path (optional and only if test/development data is passed in the input data! May be empty if the last parameter is required).
8. The refinement order definition string - defines the order in which refinements are executed on NE tagged data.

The script depends on the "***NERefinements.pm***" *Perl* module and the *Stanford NER* module "***stanford-ner.jar***".

3.1.5.4 Internal modules

Internal modules are not supposed to be called externally (manually) by the user, however the scripts contain many useful functions, which could be useful to the user if he/she would want to extend the system or create his/her own NER system.

3.1.5.4.1 Bootstrapping module

The *Perl* module "***BootstrapTools.pm***" provides a set of functions used in the bootstrapping workflow. A list of functions used in bootstrapping is as follows:

1. ***GetTopSentencesFromDirectory*** - analyses all files within a directory and returns an array, which contains at most N top ranked sentences for each NE token. Only sentences with unique morphological tags are selected (analysing also existing training data). If the POS tagger does not support morpho-syntactic tags, the uniqueness constraint is not used.
2. ***ExtractNewGazetteerData*** - Extracts new gazetteer data from a directory of NE-tagged files into a target file. Only those named entities are extracted, which are considered the most likely using a threshold and are unique and non-existing in the existing gazetteer data files defined in the provided *Stanford NER* property file. Extracted named entity length is limited to less than or equal to ten tokens.
3. ***PrintSent*** – Prints sentences from an array to a new training data file. The method is used after sentence extraction in each bootstrapping iteration to save the newly extracted training data.

The module depends on the "***NEUtilities.pm***" and "***NERefinements.pm***" *Perl* modules.

3.1.5.4.2 Data pre-processing module

The *Perl* module "***NEPreprocess.pm***" provides a set of functions used in document pre-processing before NE-tagging and training. It also provides a method to mark plaintext with NE tags. A list of functions used in the workflows is as follows:

1. ***RemoveEmptyLines*** - Removes empty lines from a tab-separated document. The method allows correction of tokenizer errors (wrong sentence borders) and at the same time allows tokens from 2 lines in input data to be separated within two different sentences. This process makes it possible to prohibit cross-line NE-tagging. According to a selected option all empty lines are kept ("1"), all empty lines, where 2 or more empty lines are one after another are kept ("2"), all lines are removed (all other values).
2. ***Detagger*** – Splits NE tags and plaintext from a *MUC-7* annotated document. NE tags and the plaintext are saved in separate documents. After calling this method, the plaintext can be POS-tagged.
3. ***AddNewTags*** – After POS-tagging of a plaintext document, this method combines the tab-separated tokenized, POS-tagged and lemmatized document with the NE tags, which were split from the plaintext using the ***Detagger*** method.
4. ***FindTokenPos*** – If the POS-tagger used in POS-tagging of a plaintext document does not produce positional token information that would allow NE-markup to be applied to the plaintext (for instance, TreeTagger does not produce any – line from, column from, line to, column to), the method analyses the POS-tagged document and the plaintext and assigns positional information for each token.
5. ***AddMarkupToPlaintext*** – Adds NE markup from a tab-separated NE tagged document to the plaintext document. The plaintext document should contain the exact number of tokens (and the same tokens) as in the tab-separated document. The produced output file is a *MUC-7* annotated plaintext.

The module depends on the "***NEUtilities.pm***" *Perl* module.

3.1.5.4.3 NE-tagged data refinement module

The *Perl* module "***NERefinements.pm***" provides a set of functions used in NE classification refinement. As the *Stanford NER* may produce inconsistencies in the tagged data (for instance, when only one quotation mark is tagged or the classifier does not obey the one sense per discourse rule), refinements can improve the overall NE-tagging results. A list of available refinement and refinement utility functions (except testing functions) is as follows:

1. ***LoadTabSepFile*** – Reads all tokens form a tab separated document into an array. The refinement methods operate only with the token array and do not read tab separated documents. As the whole document is read into memory, it is strongly advised to not use large files that could fill the system's random access memory.
2. ***SaveTabSepDoc*** – Saves the token array into a tab separated document. This method is called after all refinements are applied. The method ***AddMissingLineBreaks*** can be called only after the tab-separated document is saved.

3. ***CalculateProbibility*** – Calculates the average value of an array's elements. The method is used to calculate full named entity probabilities from single tokens (as a NE consists of a sequence of tokens).
4. ***GetFullNETagsFromTokens*** – The method is used in most of the refinement methods to find named entity positions (for example, NE "*X*" of type "*LOCATION*" starting at token "*Y*" and ending at token "*Z*") within an array of NE tagged tokens.
5. ***WriteNEtagsInTokens*** – Each of the refinement methods makes changes in the original token array. This method applies the changes to the existing token array's named entities.
6. ***CombinedRefsOnFile*** – Executes refinements in a predefined order or in a required order if the user provides a "***Refinement order definition string***". Refinements may be executed on a single file multiple times if such are defined in the refinement order definition string. If a Refinement order definition string is not given, the default order "*L N S R_0.7 C T_0.90*" is used. This configuration proved to achieve better precision on Latvian development data (at the same time considering recall).
7. ***ConsolidateEqualEntities*** – Finds named entities (token sequences) with equal lemmas, which are classified to different NE categories and consolidates them (assigns only one NE category) according to the highest likely NE category (the average probability takes named entity total count and individual category counts into account). If an ambiguous situation is found where it is not possible to distinguish between a most likely category no changes are applied. This method tries to apply the "*One sense per discourse*" rule on named entities.
8. ***RemoveLowProbNETags*** – Removes named entities (the NE category is replaced with the non-entity category "*O*") with the average probability lower than a given threshold.
9. ***TagEqualLemmas*** – Tags missing lemma (tokens classified as non-entities) sequences if the same lemma sequences in different positions have been tagged as named entities. A threshold is applied to the existing named entities in order to find missing ones.
10. ***CleanBracketsAndQuotations*** – Finds named entities with brackets or quotation marks in tokens and tries to re-tag the named entities if it contains any unclosed quotation marks or brackets. If the bracket or quotation mark is in the middle of the named entity, the method tries to tag all tokens till a nearly located (up to a threshold in length) closing bracket or quotation mark (in the appropriate direction – left or right, depending on the missing symbol) token as part of the named entity. If the bracket or quotation mark is at the end or beginning of the entity, removes it from the named entity.
11. ***RemoveCorruptStringTokensFromNETags*** – Re-tags named entities containing a pre-defined list of strings as non-entities if the tokens containing the strings are in the middle of named entities or removes the tokens from the named entity if

they are in the beginning or at the end of the named entity. For instance, internet addresses containing protocol indicative strings (“*://*”) are removed.

12. ***RemoveCorruptStringNETags*** – Re-tags the whole named entity as a non-entity if the named entity of a predefined category (for instance, “*ORG*”) contains more than allowed number of predefined strings (each string is counted independently of other strings). For instance, a person or an organization is not allowed to contain more than one “/” symbol in the name.

The ***Refinement order definition string*** may consist of any number and any order of space separated refinement identifiers. Each identifier represents a single refinement function. Some functions (for instance, “*R*” and “*T*”) require thresholds to be passed in the form “*[ID]_0.#*” together with the identifier (“*[ID]*” represents the identifier; “*#*” represents decimal digits after 0). Available identifiers are:

1. “*A*” for the method ***AddMissingLineBreaks*** – this method has to be called after all other refinements as all other refinements operate with a token array, but this method operates with a tab-separated document. Also, in order to control that the method is called as the last one, all refinements after this method’s call are ignored. The method is described in section 3.1.5.4.4.
2. “*C*” for the method ***ConsolidateEqualEntities***
3. “*L*” for the method ***CleanBracketsAndQuotations***
4. “*N*” for the method ***RemoveCorruptStringNETags***
5. “*S*” for the method ***RemoveCorruptStringTokensFromNETags***
6. “*R_0.#*” for the method ***RemoveLowProbNETags***
7. “*T_0.#*” for the method ***TagEqualLemmas***

The module depends on the “***NEUtilities.pm***” Perl module.

3.1.5.4.4 Utility functions for named entity recognition

The Perl module “***NEUtilities.pm***” provides a set of useful utility functions used in the whole named entity recognition system. A list of functions is as follows:

1. ***IsValidGazetteerType*** – Returns “*1*” if the short NE tag (for instance, “*LOC*”, “*ORG*”, “*PERS*”, etc.) is valid for gazetteer extraction; if not the method returns “*0*”. The method is used in bootstrapping, when extracting new gazetteer data. If the user wishes to change, which named entities are extracted for gazetteer data, the appropriate value (“1” or “0”) has to be changed in this method. The method is used when pre-processing *MUC-7* annotated data. This method (and the next three methods) is created to minimize changes, which would have to be made if the user would want to change the number of NE categories that the system supports.
2. ***GetShortTagType*** - Returns a short NE tag type from a *MUC-7* NE tag type. For instance, passing “*LOCATION*”, the method would return “*LOC*”.

3. ***GetNEtagType*** - Returns a *MUC-7* NE tag type from a short NE tag type. For instance, passing "*LOC*", the method would return "*LOCATION*". The method is used when applying NE markup to plaintext.
4. ***GetMucTagName*** - Returns a *MUC-7* tag name from a short NE tag type. For instance, passing "*LOC*", the method would return "*ENAMEX*". The method is used when applying NE markup to plaintext.
5. ***AddMissingLineBreaks*** – Post-processes POS-tagged and NE tagged files and creates a result file, which contains NE tagged data from the NE tagged file including empty lines from the POS tagged document. In cases where named entities span over multiple lines, the NE is either split in two entities or the trailing tokens of the second entity are re-tagged as non-entity tokens. The selection (re-tagging or removal) is controlled by a threshold of the first token's (of the second line) classification probability.
6. ***CreateDirectoryFileList*** – Creates a comma separated list of file addresses in a directory and returns the result as a string. The method is used to create the training file list for a bootstrapping iteration.
7. ***AddPropertyToFile*** – Appends a new property at the end of the *Stanford NER* property file as a new line. The method does not check, whether the property is already existing.
8. ***ReadPropertyFromFile*** – Reads a property's value from a *Stanford NER* property file. If the file contains multiple equal properties, the first property's value is returned.
9. ***ChangePropertyInFile*** – Changes a property's value in a *Stanford NER* property file. If the file contains multiple equal properties, all found property values are changed.
10. ***AppendAFileToAFile*** – Appends a file's contents to another file.
11. ***ReadExistingGazetteerData*** – Reads all lines of the tab-separated gazetteer files and returns a hash table containing unique lines.
12. ***CopyFilesFromDirectory*** – Copies files with a specified extension from one directory to another, thereby changing the file extension to a new extension (if required).
13. ***MoveFilesFromDirectory*** – Moves files with a specified extension from one directory to another, thereby changing the file extension to a new extension (if required).
14. ***CopyFilesFromArray*** – Copies files specified in an array to a target directory without changing file extensions.
15. ***MoveFilesFromArray*** – Moves files specified in an array to a target directory without changing file extensions.
16. ***GetRandomFiles*** – Returns a number of random file addresses (in an array) containing a specified extension from a directory.

17. ***GetTokenTotalResultLine*** – Finds and returns the line containing token total evaluation results in a given evaluation file (which is created by the "***NEEvaluation_v2.pl***" script).
18. ***GetTokenResultEntry*** – Finds and returns a specific token total result entry in a given evaluation file (which is created by the "***NEEvaluation_v2.pl***" script). The result entry is specified by a column number. The column numbers are: "1" - recall, "2" - precision, "3" - accuracy, "4" - F-measure.
19. ***GetNETotalResultLine*** – Finds and returns the line containing full named entity total evaluation results in a given evaluation file (which is created by the "***NEEvaluation_v2.pl***" script).
20. ***GetTime*** – For logging purposes returns the current system time.

The module does not have additional dependencies.

3.1.5.4.5 Tokenization, lemmatization and POS-tagging module

The *Perl* module "***Tag.pm***" provides POS-tagging functionality for data pre-processing. The module is extendable if it is required by the user to use a different POS-tagger (the POS-tagger has to produce compliant output data). The possibility to extend the module is described in section 3.1.7.

The only method that the module contains is "***TagText***", which POS-tags a single plaintext document using a specified POS-tagger and language. The results are saved in a tab-separated data file. The method creates a set of temporary files, some of which are required by the tagging and data pre-processing workflows. The method, however, allows also removal of temporary files, but the option is not used in the current workflows.

The currently supported POS-tagger and language combinations are as follows:

1. "***Tagger***" – "***et***" (Estonian), "***lv***" (Latvian) and "***lt***" (Lithuanian). "*Tagger*" represents the Tilde's POS-tagging web service for the Baltic languages and is the suggested choice for these languages.
2. "***Tree***" – "***bg***" (Bulgarian), "***de***" (German), "***el***" (Greek), "***en***" (English), "***es***" (Spanish), "***et***" (Estonian), "***fr***" (French) and "***it***" (Italian). "*Tree"* represents the language independent part-of-speech tagger *TreeTagger*.

Note that *TildeNER* currently supports only Latvian and Lithuanian named entity recognition. Other languages require *Stanford NER* models to be either acquired or trained.

The module depends on the "***NEPreprocess.pm***" *Perl* module.

3.1.5.4.6 Stanford NER

The main classification of named entities and training is done by the *Stanford NER* conditional random field classifier; therefore, the *TildeNER* system depends on the "*stanford-ner.jar*" module to be available. The standard downloadable version[11], however, won't be

[11] Stanford NER non-modified version: http://nlp.stanford.edu/software/CRF-NER.shtml.

compliant with the workflows as it does not support the required input and output data standards. Therefore, a modified version has been included in the toolkit under the "*TildeNER*" directory.

3.1.5.5 POS-taggers included in the toolkit

The *TildeNER* system has integration with two POS-taggers – Tilde's Baltic language POS-tagging web service (*Tagger.exe* and *tagger.sh*) and the *TreeTagger* (should be included in the "*Treetagger*" subdirectory in the "*TildeNER*" directory). Both taggers may be used for research purposes and non-commercial usage. This toolkit does not provide commercial licensing of these POS-taggers.

It is advised to use Tilde's Baltic language POS-tagging web service for Latvian, Lithuaian and Estonian. For all other languages the user will have to acquire *TreeTagger* and make modifications as described below.

TreeTagger can be accessed and freely downloaded from http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/. The user will need to download:

- The user platform's tagger package (where the source code to the "*tree-tagger*" or "*tree-tagger.exe*" application is located).
- The tagging scripts' package (where the "*tokenize.pl*" script and the language dependent abbreviation file is located).
- The language dependent parameter file for *TreeTagger*.

In order to integrate TreeTagger within TildeNER, the user will have to:

- Copy the "*tree-tagger*" executables, the "*tokenize.pl*" script, the abbreviation file and the parameter file to the "*Treetagger*" subdirectory of "*TildeNER*".
- Modify "*tokenize.pl*" so that it would accept the following execution command from within the "*Treetagger*" directory.

```
perl tokenize.pl [Input File] [Output File] [Clitic Identifier]
[Abbreviation Usage] [Abbreviation File]
```

The script requires in total five parameters:

- The path of the file that has to be tokenized.
- The path of the output file where the results will be written.
- An optional parameter, which identifies sequences that will be cut off before and after words. May be "*-e*" for English, "*-f*" for French and "*-i*" for Italian. For other languages an empty string will be passed if abbreviation usage will be specified or nothing if no abbreviations will be used.
- An optional parameter that identifies whether abbreviations will be used to tokenize the input file. Allowed values are "*-a*" for abbreviation usage and nothing for no abbreviations.
- An optional parameter that identifies the path to the abbreviation file located within the "*Treetagger*" subdirectory of the "*TildeNER*" directory.

For supported languages, parameter files and abbreviation files please refer to the “*Tag.pm*” script.

3.1.5.6 NER models and data samples included in the toolkit

As the *TildeNER* system has been developed for Latvian and Lithuanian named entity recognition, the system also provides data required for named entity tagging of documents in both languages.

The “***Sample_Data***” subdirectory of the “*TildeNER*” directory, therefore, provides:

1. NER models for both languages:
   - 1.1. “***LV_Model_P.ser.gz***” – Latvian bootstrapped model for increased precision (use the “*L N S R_0.7 C T_0.90 A*” refinement order definition string and the “***LV_P_Tagging_prop_sample.prop***” property file to achieve the best results);
   - 1.2. “***LV_Model_F.ser.gz***” – Latvian bootstrapped model for increased F-measure (use the “*L N S R_0.4 T_0.70 A*” refinement order definition string and the “***LV_F_Tagging_prop_sample.prop***” property file to achieve the best results);
   - 1.3. “***LT_Model_F.ser.gz***” – Lithuanian bootstrapped model for increased F-measure (use the “*L N S R_0.4 T_0.70 A*” refinement order definition string and the “***LT_F_Tagging_prop_sample.prop***” property file to achieve the best results);
   - 1.4. “***LT_BASELINE_Model.ser.gz***” – Lithuanian baseline model that achieves the best precision (use the “*L N S R_0.7 C T_0.90 A*” refinement order definition string and the “***LT_B_Tagging_prop_sample.prop***” property file to achieve the best results);
2. Tagging property files required by Stanford NER (“***LV_F_Tagging_prop_sample.prop***”, “***LV_P_Tagging_prop_sample.prop***”, “***LT_F_Tagging_prop_sample.prop***” and “***LT_B_Tagging_prop_sample.prop***”);
3. Gazetteer data used in training and required when tagging documents. The data is located in the “***LV_Gazetteer***” and “***LT_Gazetteer***” subdirectories (separately for Latvian and Lithuanian). The gazetteer data contains:
   - 3.1. Latvian location gazetteer “***LV_LOC_GAZETTEER.txt***”;
   - 3.2. Latvian organization gazetteer “***LV_ORG_GAZETTEER.txt***”;
   - 3.3. Latvian organization type gazetteer “***LV_ORG_INIT_GAZETTEER.txt***”;
   - 3.4. Latvian person name gazetteer “***LV_PERS_GAZETTEER.txt***”;
   - 3.5. Latvian precision bootstrapped gazetteer “***LV_PRECISION_BOOTSTRAPPED_GAZETTEER.txt***” (contains person names, locations and organizations extracted in bootstrapping);
   - 3.6. Latvian F-measure bootstrapped gazetteer “***LV_FMEASURE_BOOTSTRAPPED_GAZETTEER.txt***” (contains person names, locations and organizations extracted in bootstrapping);Lithuanian location gazetteer “***LT_LOC_GAZETTEER.txt***”;

3.7. Lithuanian organization gazetteer “***LT_ORG_GAZETTEER.txt***”;

3.8. Lithuanian person name gazetteer “***LT_PERS_GAZETTEER.txt***”;

3.9. Lithuanian F-measure bootstrapped gazetteer “***LT_FMEASURE_BOOTSTRAPPED_GAZETTEER.txt***” (contains person names, locations and organizations extracted in bootstrapping);

The sample data directory contains also training property template files (“***LV_Training_prop_template.prop***” and “***LT_Training_prop_template.prop***”) and tagging property template files (“***LV_Tagging_prop_template.prop***” and “***LT_Tagging_prop_template.prop***”) that were used to train the provided Latvian and Lithuanian NER models.

Note that all property files and templates have to be updated with the user system’s local paths so that the system can access gazetteer data! If not updated, the system will unexpectedly crash if the working directory will not be set to the *TildeNER* root directory. The user has to update the “*gazette*” property in all property files and templates in use. Also on *Windows* the *Linux* directory separation character “/” should be used instead of the *Windows* character “\”.

3.1.6 Input/Output data formats

All documents used in the *TildeNER* system should be encoded using *UTF-8* encoding. Other encodings are not supported. The *TildeNER* system is *BOM* insensitive; however, it is advised for the user to strip the BOM characters before processing data as some POS-taggers may operate incorrectly. It is also advised because of the same reason to remove all control characters except *LF* (“*\n*”), *CR* (“*\r*”) and “*TAB*” (“*\t*”) from the input data.

All input and output data files have to contain file extensions (for instance, “*.txt*” for plaintext documents, “*.pos*” for POS-tagged documents, etc.); otherwise, the system may perform unexpectedly.

3.1.6.1 Plaintext Format

The first and the most simple data format for named entity tagging is plaintext. A plaintext document is not allowed to contain mark-up within the text. All mark-up will be considered as part of the plaintext and processed together with the text.

3.1.6.2 *MUC-7 Annotated Data Format*

The manual annotation tool *NESimpleAnnotator* saves documents in the *MUC-7* annotated data format. The format allows MUC-7 named entity tags, as shown below (for each named entity type), to be embedded within the plaintext.

```
<ENAMEX TYPE="ORGANIZATION">LVRTC</ENAMEX>
<ENAMEX TYPE="PERSON">Krišjānis Peters</ENAMEX>
<ENAMEX TYPE="LOCATION">Latvijā</ENAMEX>
<ENAMEX TYPE="PRODUCT">Windows 7</ENAMEX>
<NUMEX TYPE="MONEY">Ls 7011 mēnesī</NUMEX>
<TIMEX TYPE="TIME">ap 21–22</TIMEX>
<TIMEX TYPE="DATE">1994.gadā</TIMEX>
```

A sample annotated document (shown is only one sentence) is as follows:

```
Kompānijai <ENAMEX TYPE="ORGANIZATION">Lattelecom</ENAMEX>, savukārt pieder
23% <ENAMEX TYPE="LOCATION">Latvijas</ENAMEX> mobilā operatora <ENAMEX
TYPE="ORGANIZATION">Latvijas Mobilais Telefons</ENAMEX> kapitāldaļu.
```

If other annotation tools (not the *NESimpleAnnotator*) are to be used, the user has to make sure that named entities do not start or end with a whitespace because in this case the pre-processing workflow won't be able to align the annotation borders with the token borders within the POS-tagged document. If border mismatches will be found, the named entity mark-up will be removed.

3.1.6.3 *Tab-separated Training/Development/Testing Data Format*

The annotated data pre-processing workflow (***PreprocessMuc7DataDirectory.pl***) produces data in a tab-separated, POS-tagged, tokenized, lemmatized and NE-tagged format. The training, evaluation and bootstrapping scripts require training (also seed), development and test data to be prepared in this format.

The tab-separated format contains (in a fixed and non-changeable sequence):

1. The original word form
2. Part of speech
3. Lemma
4. Morpho-syntactic tag (may be also non-positional, but as a sample the Tilde's positional 28 category morpho-syntactic tag is given)
5. Line in which the token starts in the original plaintext document
6. Column in which the token starts in the original plaintext document
7. Line in which the token ends in the original plaintext document
8. Column in which the token ends in the original plaintext document
9. Named entity category

A sample pre-processed test data sentence is as follows:

```
Pēc     S    pēc     S----------------pdp------f-  24  100  24  102  B-DATE
divām   M    divi    M-fpd---c-----------------l-  24  104  24  108  I-DATE
```

```
dienām    N    diena      N-fpd---------n-----------l-  24  110  24  115  I-DATE
,         T    ,          T--------------------------,  24  116  24  116  O
14        D    14         D--pg-----------------------  24  118  24  119  B-DATE
.         T    .          T--------------------------.  24  120  24  120  I-DATE
novembrī  N    novembris  N-msl---------n-----------l-  24  122  24  129  I-DATE
pilsētā   N    pilsēta    N-fsl---------n-----------l-  24  131  24  137  O
iebrauca  V    iebraukt   Vs----3--i----------------l-  24  139  24  146  O
Ziemeļu   N    Ziemele    N-fpg---------n-----------f-  24  148  24  154  B-ORG
alianses  N    alianse    N-fsg---------n-----------l-  24  156  24  163  I-ORG
tanki     N    tanks      N-mpn---------n-----------l-  24  165  24  169  O
.         SENT .          T--------------------------.  24  170  24  170  O
```

The data suggests that the sentence is form the 24th line starting from the 100th character in the line. The sentence ends in the 24th line and the last character is in the 170th position of the line. The sentence contains three named entities – two *DATE* and one *ORGANIZATION* named entity.

The first three columns are the standard TreeTagger output format. As TreeTagger does not produce the morpho-syntactic tag as well as the positional indicators, the workflow's pre-processing script is able to align tokens with the plaintext (therefore, support for all POS-taggers, which produce data in the standard TreeTagger format, can be added – as long as these can be executed using a command line, read and write data using input and output data files and operate using the UTF-8 encoding).

The Tilde's morpho-syntactic tag for the Baltic languages consists of 28 positions – part of speech (*N, V, T, etc.*), tense (*present, past, future, etc.*), gender (*masculine, feminine, neutral or common*), number (*singular, plural, dual*), case (*nominative, genitive, dative, etc.*), degree of comparison (*positive, comparative, diminutive, superlative*), person (*first, second, third*), adjective definiteness marker (*indefinite, definite*), numeral type (*cardinal, ordinal, collective or numeral*), mode (*indicative, imperative, conditional, etc.*), noun type (*place name, surname, proper name, etc. – reserved, but not implemented*), voice (*active, passive*), semantic subclass of pronouns (*personal, reflexive, possessive, etc.*), subtype of participles (*indeclinable, partly declinable, progressive, etc.*), diminutive marker for nouns (*diminutive, not diminutive, short*), reflexivity of verbs (*non-reflexive and reflexive*), negative prefix marker (*negative, affirmative*), number required for agreement with prepositions (*singular, plural*), case required for agreement with prepositions (*genitive, dative, accusative or instrumental*), place of preposition (*preposition or postposition*), verb group (*1 to 9 and perfective, imperfective or two-aspect*), semantic type of adverb (*gradual, existential, interrogative, etc.*), relation type of conjunction (*coordinating, subordinating, subject clause, etc.*), Wh marker (*reserved – not in use*), transitivity (*transitive, intransitive – reserved for Russian verbs*), animation (*animate, inanimate – reserved for Russian nouns*), usage of capital letters (*lowercase, starts with a capital, uppercase*), punctuation mark (*".", "?", "!", etc.*).

The morpho-syntactic tag is used by the Stanford NER classifier as a whole string and is not analysed as a positional morpho-syntactic tag (character by character); therefore, the system also supports other POS-tagger morpho-syntactic tags. If a POS-tagger does not assign a morpho-syntactic tag to a token, the new training data uniqueness constraint in the bootstrapping algorithm is not used and new training data is extracted based on the named entity ranking within sentences. Also the training and tagging property files should be altered so that Stanford NER does not use Morpho-syntactic tag feature functions (the property "***useMorphoTags***" has to be set to "***false***").

The format specifies that each token has to have a named entity category. Non-entities receive the category "*O*". The first token in a named entity receives a category that starts with "*B-*"; all other tokens within a named entity receive a category that starts with an "*I-*". It is not allowed for a named entity to start with an "*I-*" token. All possible (currently supported) categories are:

```
"B-ORG" and "I-ORG" for "ORGANIZATION" tokens;
"B-LOC" and "I-LOC" for "LOCATION" tokens;
"B-PERS" and "I-PERS" for "PERSON" tokens;
"B-PROD" and "I-PROD" for "PRODUCT" tokens;
"B-DATE" and "I-DATE" for "DATE" tokens;
"B-TIME" and "I-TIME" for "TIME" tokens;
"B-MON" and "I-MON" for "MONEY" tokens;
"O" for non-entities.
```

The document format also requires 2 empty lines to be present for newline characters in the plaintext. Sentences may be separated using one empty line, but that is not mandatory if the *TreeTagger* "*SENT*" category is used to mark sentence ending characters.

3.1.6.4 Tab-separated Pre-processed Unannotated Data Format

The unannotated data pre-processing workflow (***TagUnlabeledDataDirectory.pl***) produces data in a tab-separated, POS-tagged, tokenized and lemmatized format. The only difference from the previous format is that it does not contain named entity categories. A sample sentence of the format is as follows:

```
Sobrīd    -     Sobrīd   ----------------------------  0  271  0   276
Latvijā   N     Latvija  N-fsl---------n-----------f-  0  278  0   284
ir        V     būt      Vp----3--i----------7-----l-  0  286  0   287
77        D     77       D--p------------------------  0  289  0   290
pilsētas  N     pilsēta  N-fpa---------n-----------l-  0  292  0   299
.         SENT  .        T--------------------------.  0  300  0   300
```

For descriptions of the columns refer to the previous section 3.1.6.3.

3.1.6.5 Tab-separated NE-Tagged Data Format

The tagging scripts (within the bootstrapping and single document tagging workflows) produce named entity tagged data in a tab-separated, POS-tagged, tokenized, lemmatized and NE-tagged format similar to the pre-processed annotated data described in section 3.1.6.3. The only difference is that each token contains (as the last column) the probability with which the particular NE category has been assigned to the token by the *Stanford NER CRF classifier*.

A sample sentence of the format is as follows:

```
Skolotāja  N   skolotājs   N-msg---------n-----------f-  11  4123  11  4131
           O   0.9607617498899327
un         C   un          C---------------------c---l-  11  4133  11  4134
           O   0.9983287984939869
policijas  N   policija    N-fsg---------n-----------l-  11  4136  11  4144
           O   0.9966184704168756
virsnieka  N   virsnieks   N-msg---------n-----------l-  11  4146  11  4154
           O   0.9972953832284743
mēnešalga  N   mēnešalga   N-fsn---------n-----------l-  11  4156  11  4164
           O   0.99467356585981
svārstās   V   svārstīties Vp----3--i-----y----------l-  11  4166  11  4173
           O   0.9997877203504323
ap         S   ap          S----------------pdp------l-  11  4175  11  4176
           O   0.8406855682177047
240        D   240         D--pg-----------------------  11  4178  11  4180
           B-MON           0.7120957375851391
latiem     N   lats        N-mpd---------n-----------l-  11  4182  11  4187
           I-MON           0.8652995897915107
.          SENT .          T--------------------------.  11  4188  11  4188
           O   0.9997774220023308
```

3.1.6.6 Gazetteer Data Format

The last data format used in the *TildeNER* system is the gazetteer data. The gazetteer data has to be in a tab-separated format. The first column has to be the entry category (may be freely defined by the user as each category will form a new feature function within the NER system). The second column has to contain the named entities. If the named entity consists of multiple tokens, these have to be passed using a space as a separator symbol.

A sample gazetteer document is as follows.

```
LOC        Āraiši
LOC        Āraišu ezers
PERS       Adalberts
PERS       Adela
PERS       Adelaida
ORG_INIT   SIA
ORG_INIT   Ltd .
ORG_INIT   AS
ORG        Lattelecom
ORG        Microsoft
ORG        Ford
ORG        Delta Air Lines
```

The user may use multiple gazetteer documents and divide them as he/she requires. All gazetteer files have to be defined in both tagging and training property files. The system may produce unexpected results if different gazetteer lists/categories will be used in training and tagging. The sample property files contain a gazetteer entry:

```
gazette                                                                 =
/home/NER/CORPUS/GAZETTEERS/LV_PERS_GAZETTEER.txt,/home/NER/CORPUS/GAZETTEE
RS/LV_LOC_GAZETTEER.txt,/home/NER/CORPUS/GAZETTEERS/LV_ORG_INIT_GAZETTEER.t
xt,/home/NER/CORPUS/GAZETTEERS/LV_ORG_GAZETTEER.txt
```

The paths to the gazetteer files have to be manually set by the user on each system before tagging or training! If the user does not want gazetteers to be used in training and later tagging, the property has to be removed from the property files. However, the user will not be able to use the Latvian and Lithuanian NER models or the results may be unexpected if some gazetteer files will be missing when tagging documents.

3.1.6.7 Input/Output Document Pair List File Format

The I/O document pair list file format is used in the ACCURAT toolkit's NE/term mapping workflow in order to allow NE/term tagging of multiple files. The document pair list is a tab-separated text file where each line contains two elements – the input file path and the output file path. The file paths should be absolute when using the ACCURAT toolkit to avoid unexpected behaviour. The format sample is as follows:

```
[Input File Path 1]     [Output File Path 1]
…
[Input File Path N]     [Output File Path N]
```

A sample file with real values is as follows:

```
C:\NE-plain-EN\Apple.txt      C:\NE-plain-EN\Apple.txt.NE_tagged
C:\NE-plain-EN\FightClub.txt  C:\NE-plain-EN\FightClub.txt.NE_tagged
C:\NE-plain-EN\Latvia.txt     C:\NE-plain-EN\Latvia.txt.NE_tagged
C:\NE-plain-EN\Microsoft.txt  C:\NE-plain-EN\Microsoft.txt.NE_tagged
C:\NE-plain-EN\USA.txt        C:\NE-plain-EN\USA.txt.NE_tagged
```

3.1.7 Integration with external tools

To integrate the *TildeNER* system within another system that requires NE-tagging, the target system has to be able to execute command line commands. The most of the programming languages contain libraries to execute command line commands. All standard *TildeNER* execution commands are described in section 3.1.5.

To add another POS-tagger to the data pre-processing workflows, the user has to modify the script in the module ***Tag.pm***.

First of all the POS-tagger has to produce output at least in the *TreeTagger* format (or the full unlabelled data format described in section 3.1.6.4). If the POS-tagger does not produce the output in the required format, the user has to provide a wrapper system that calls the POS-tagger and converts the data to the required format.

If the POS-tagger produces the required output, the user has to create a new "*elsif*" block in the "*TagText*" function after the "*elsif ($_[1] eq "Tagger")*" block and before the "*else*" block according to the following structure:

```
elsif ($_[1] eq "[TAGGER_CODE]") {
 if($_[0] eq '[LANGUAGE_CODE]') {
  #Call the POS tagger to tokenize and tag plaintext.
  @agrs=("$_[2]","$outputDir$filename.Tree");  #Set  additional  arguments
  here!
  system "[PATH_TO_YOUR_POS_TAGGER]",@agrs;
  #Add token positions to the POS-tagger file using the POS-tagged file and
  the plaintext file.
  NEPreprocess::FindTokenPos("$_[2]"
  ,"$outputDir$filename.Tree","$outputDir$filename.temp");
 }
 else  {  print  STDERR  "[Tag::TagText]  ERROR:  no  such  tagger-language
combination: \"$_[1]\"-\"$_[0]\""; die; }
}
```

If the POS-tagger produces output data, also assigning positional token information (as defined in section 3.1.6.4), the user may also skip the execution of the "*FindTokenPos*" method. In this case the "*elsif*" statement has to be as follows:

```
elsif ($_[1] eq "[TAGGER_CODE]") {
 if($_[0] eq '[LANGUAGE_CODE]') {
  #Call the POS tagger to tokenize and tag plaintext.
  @agrs=("$_[2]","$outputDir$filename.temp");  #Set  additional  arguments
  here!
  system "[PATH_TO_YOUR_POS_TAGGER]",@agrs;
 }
 else {
  print STDERR "[Tag::TagText] ERROR: no such tagger-language combination:
  \"$_[1]\"-\"$_[0]\""; die; }
}
```

The user has to define:

1. A code for the POS-tagger ("*[TAGGER_CODE]*")
2. The language that the POS-tagger supports ("*[LANGUAGE_CODE]*")
3. Additional arguments (if any required) or rearrange existing arguments (if required) in the array "*@agrs*" ("*$_[2]*" represents the input file and "*$outputDir$filename.temp*" and "*$outputDir$filename.Tagger*" represent the output files – these parameters are mandatory and should not be changed)
4. Specify the POS-tagger's path in the user's system ("*[PATH_TO_YOUR_POS_TAGGER]*");

The user may also replace the “***Tag.pm***” module if the existing module prohibits some sort of integration, but the input/output parameters (including the temporary file “**.temp*”) should be the same; otherwise the system may crash.

3.1.8 Contact

For further information and technical support installing and/or running this tool, please email to Mārcis Pinnis: marcis.pinnis@tilde.lv.

3.1.9 Useful references

The *TildeNER* system is inspired and the ideas are based on the following papers:

1. Andrew Carlson, Sue Ann Hong, Kevin Killourhy and Sophie Wang, Active Learning for Information Extraction via Bootstrapping, 2009.
2. Dan Wu, Wee Sun Lee, Nan Ye and Hai Leong Chieu, Domain adaptive bootstrapping for named entity recognition, EMNLP '09 Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3 - Volume 3, Association for Computational Linguistics Stroudsburg, PA, USA, 2009
3. David Nadeau, Semi-Supervised Named Entity Recognition: Learning to Recognize 100 Entity Types with Little Supervision, PhD Thesis, Ottawa, Canada, 2007
4. Erik F. Tjong Kim Sang and Fien De Meulder, Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition, CONLL '03 Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 - Volume 4, Association for Computational Linguistics, Stroudsburg, PA, USA, 2003.
5. Fien De Meulder and Walter Daelemans, Memory-based named entity recognition using unannotated data, CONLL '03 Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 - Volume 4, Association for Computational Linguistics Stroudsburg, PA, USA, 2003.
6. Jing Jiang, Chengxiang Zhai, Instance weighting for domain adaptation in NLP, Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, Association for Computational Linguistics, Prague, Czech Republic, 2007
7. Jon Patrick, Casey Whitelaw and Robert Munro, SLINERC: the Sydney Language-Independent Named Entity Recogniser and Classifier, COLING-02 proceedings of the 6th conference on Natural language learning - Volume 20, Association for Computational Linguistics Stroudsburg, PA, USA, 2002.
8. Zornitsa Kozareva, Bootstrapping named entity recognition with automatically generated gazetteer lists, EACL '06: Proceedings of the Eleventh Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop, Association for Computational Linguistics Stroudsburg, PA, USA, 2006.

The NE annotated data standard is based on the MUC-7 NE annotation guidelines: http://www-nlpir.nist.gov/related_projects/muc/proceedings/ne_task.html.

The POS-tagged data standard is an extended version of the TreeTagger format: http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/.

The Stanford NER CRF classifier is used as the core named entity classification system: http://nlp.stanford.edu/software/CRF-NER.shtml.

3.2 OpenNLP wrapper

3.2.1 Overview and purpose of the tool

In the multi-lingual NE and term mapper (see section 5.1) we make use of *OpenNLP* to tag named entities for the English documents. *OpenNLP* is an existing tool and is not implemented within the ACCURAT project. The output of this system is, therefore, different from the input format of the NE mapper. The wrapper:

- enables that the output of *OpenNLP* is of the same format as the input files to the mapper
- provides a scenario to users where the mapper can be run on existing annotated data
- enables the users to use other NER systems to prepare the input to the mapper.

3.2.2 Changes from previous version

The following bugs have been resolved from the previous version:

- possible overlapping of NE markup when converting from the internal to the *MUC-7* compliant output format;
- wrong output encoding – changed to *UTF-8*.

3.2.3 Software dependencies and system requirements

The wrapper is implemented in the programming language *Java*. It requires the following settings to run:

- *JRE* (*Java Runtime Environment*) 1.6
- 1+ GB RAM

3.2.4 Installation

The wrapper does not require any installation.

3.2.5 Execution instructions

The *OpenNLP* wrapper can be run using the following command:

```
java –jar OpenNLPWrapper.jar [fileList]
```

fileList: a tab separated list of files. On each line the file contains the file name (with the full path) to be annotated by the wrapper, a tab for separation and the file name (with the full path) where the results of the annotation should be saved. The output file will be automatically generated by the wrapper. For a sample of the format of the file list refer to section 3.1.6.7 of the *TildeNER* system.

Please also make sure that you run the wrapper from the folder where all the required resources are saved. These resources are the entire folders (docs, dataand testdocs) and are provided with the wrapper.

3.2.6 Input/Output data formats

Input to the wrapper is text that is encoded in *UTF-8*.

Output of the *OpenNLP* wrapper is text with NEs tagged. NEs are tagged according to *MUC-7* style (for a more detailed format description refer to section 3.1.6.2 of the *TildeNER* system). For more details see the description for the NE and term mapper.

3.2.7 Contact

For further information and technical support installing and/or running this tool, please email to Ahmet Aker: a.aker@dcs.shef.ac.uk.

3.3 NERA1: Named Entity Recognition for English and Romanian

3.3.1 Overview and purpose of the tool

NERA1 tool is designed to identify and label Named Entities in raw or already preprocessed texts. It is designed to work for English and Romanian and to identify 6 types of Named Entities: *PERSON*, *ORGANIZATION*, *LOCATION*, *PRODUCT*, *DATE*, *TIME* and *MONEY*. The current version focuses mainly on the first 3 types and works without any use of gazetteers. First, it identifies named entities boundaries using regular expressions, and then, it labels the entities according to a Maximum Entropy classifier trained on contextual features. *NERA1* needs the input files to be preprocessed and in order to do this, it calls the *TTL* web service (hosted at RACAI). However, as Romanian is a language with diacritics and many Romanian texts are missing these diacritics, when dealing with it, *NERA1* is able call the *diacritics insertion* web service (also hosted at RACAI), if requested.

Important facts:

• *NERA1* can receive as input raw text files with no preprocessing: in this case *TTL* web service is called for preprocessing and internet connection is needed;

• *NERA1* can work on existing annotated data if the already existing annotation is compliant with RACAI's *XML* resource format.

3.3.2 Changes from the previous version

Aside from bug fixing, there are no functional modifications and/or changes to the user's interface of this tool.

3.3.3 Software dependencies and system requirements

NERA1 is implemented in *C#* using *.Net Framework 4.0*. For machines using *Windows*, the users should install *.Net Framework 4.0*. For machines using *Linux*, the users should use

Mono 2.10 (http://www.mono-project.com/Main_Page). The machine should have at least 1GB of RAM.

3.3.4 Installation

NERA1 does not require any special installation other that ensuring *.NET Framework* is installed.

3.3.5 Execution instructions

The command line for *NERA1* is (the language of the text being processed is automatically recognized):

```
NERA1.exe --input [DATA_FILE] [--source [LANG]]
[--param [ap]=[TRUE]/[FALSE]] [--param [di]=[TRUE]/[FALSE]]
[--param [k]=[TRUE]/[FALSE]]
```

where:

"*DATA_FILE*" – Each line in the *DATA_FILE* should contain the path of an input file and the path of an output file, tab separated;

"*LANG*" – The language of the texts; Default: "*ro"*;

"*-ap*" – optional argument usable when the input files are already pre-processed and the annotation is compliant with RACAI's *XML* resource format; Default: *FALSE*;

"*-di*" – optional argument for calling the diacritics web service for Romanian; Default: *FALSE*;

"*-k*" – optional argument for keeping *TTL*'s (original) annotation in the output file.

3.3.6 Input/Output data formats

The input files are either raw *UTF-8* texts or pre-processed texts (RACAI's *XML* resource format) (see the "*ap*" option of the *NERA1* executable in the previous section).

The output files are texts with NEs tagged, according to *MUC-7* style (for a more detailed format description, refer to section 3.1.6.2 of the *TildeNER* system).

3.3.7 Contact

For further information and technical support installing and/or running this tool, please email to Dan Ştefănescu: danstef@racai.ro.

4 Tools for terminology extraction

This section covers the tools that perform terminology extraction and tools that are created to integrate out of ACCURAT project developed tools within the toolkit's general use case workflows.

The tools included in this section of the ACCURAT toolkit are:

- *Tilde's wrapper system for CollTerm* (developed by Tilde; see section 4.1);
- *KEA wrapper* (a wrapper system for the external tool KEA; developed by USFD; see section 4.2);
- *CollTerm, a tool for term extraction* (develped by FFZG, see section 4.3);
- *Terminology Extraction for English and Romanian* (developed by RACAI, see section 4.4).

4.1 Tilde's wrapper system for CollTerm

4.1.1 Overview and purpose of the tool

The *CollTerm* system (developed by FFZG as part of the ACCURAT project) is a tool for extracting collocations of length two to four words. It is based on POS/MSD phrase pattern and stop-word filters and association measures that determine how strongly two or more words co-occur. It can also extract unigrams based on a *TF*IDF* keyword weighing algorithm. The input for the system is a set of documents which are tokenized (verticalized), POS/MSD-tagged and lemmatized. The output are candidates of a specific number of words ordered by their collocation strength.

The scoring of the n-grams (starting from bigrams) that pass the POS/MSD filters and stop-word filters is performed by five different association measures. Association measures, loosely speaking, measure how much words in a sequence of words co-occur more than by chance.

The five association measures implemented in this tool are the following:

- *Dice coefficient*

$$DICE(w_1...w_n) = \frac{nf(w_1...w_n)}{\sum_{i=1}^{n} f(w_i)}$$

where $f(.)$ is the frequency of a specific n-gram.

- *Modified pointwise mutual information*

$$I'(w_1...w_n) = \log_2 \frac{f(w_1...w_n)P(w_1...w_n)}{\prod_{i=1}^{n} P(w_i)}$$

where $f(.)$ is the frequency of a specific n-gram and $P(.)$ is the probability of a n-gram calculated as a maximum likelihood estimate.

- *Chi-square statistic*

  $$\chi^2 = \sum_{i,j} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

  where O_{ij} and E_{ij} are observed and expected frequencies in a contingency table of two dimensions for bigrams (contingency tables for n-grams have n dimensions).
- *Log-likelihood ratio*

  $$G^2 = 2\sum_{i,j} O_{ij} \log \frac{O_{ij}}{E_{ij}}$$

  where observed and expected frequencies are calculated as in the chi-square statistic.
- *T-score statistic*

  $$tscore = \frac{O_{11} - E_{11}}{\sqrt{E_{11}}}$$

  where observed and expected frequencies are calculated as in the chi-square statistic and the log-likelihood ratio.

These association measures have been selected from an exhaustive list of existing association measures since previous research for bigrams[12][13], and n-grams[14] has shown that these measures show the most consistent results on different datasets and languages. Additionally, only these association measures are implemented since other measures do not show consistent and statistically significant improvements over each other.

For unigrams, the system uses a *TF*IDF* (term frequency in a document times inverse document frequency on a reference corpus) based keyword ranking measure.

Tilde's wrapper system for CollTerm provides functionality for term extraction in plaintext documents, pre-processing of term-annotated documents and also evaluation of *CollTerm* results for a given test corpus. As *CollTerm* requires pre-processed data (see section 3.1.6.4 for a format description), the wrapper provides all required pre-processing scripts.

The wrapper system has been created also in order to support varied length term extraction using *CollTerm*. As *CollTerm* supports only fixed length (from one to four tokens) n-gram extraction, the wrapper system executes *CollTerm* multiple times and combines the results in one output data file for each input document.

[12] Stefan Evert: The statistics of word cooccurrences: Word pairs and collocations. PhD thesis, Universität Stuttgart, Institut für Maschinelle Sprachverarbeitung, 2005.

[13] Pavel Pecina: Lexical association measures: Collocation Extraction. Studies in Computational and Theoretical Linguistics. Institute of Formal and Applied Linguistics, Prague, Czech Republic, 2009.

[14] Saša Petrović et al: Extending lexical association measures for collocation extraction. Computer Speech and Language 24 (2), 383–394, 2010.

4.1.2 Changes from previous version

Tilde's wrapper system for CollTerm has been updated to support the *CollTerm* version 0.5. *CollTerm* now supports also unigram term tagging; therefore, the wrapper system is able to tag unigram terms as well. Further improvements include adjusted tagging samples for Latvian, Lithuanian as well as support for English term tagging. However, the user will have to acquire *TreeTagger* in order to tag documents in English (see *TildeNER* section 3.1.5.5 on how to adapt *TreeTagger* for the wrapper system). In order to help users get acquainted with the system, "*RUN*" scripts have introduced. The "*RUN*" scripts allow the user to test various scripts easily and during set-up of the tool can help figuring out whether some dependencies are missing.

The new version, mainly because of introduction of unigram term tagging, performs better than the previous system. The F-measure for Latvian term tagging has been increased from 46.15 to 53.01 for full terms (border detection included) and from 54.3 to 60.86 in the token level. Although precision has dropped from 50.21% to 48.66% for full terms, recall has been significantly improved from 42.7% to 58.21%.

4.1.3 Software dependencies and system requirements

Tilde's wrapper system's for CollTerm software dependencies are as follows:

- *TreeTagger* (if the user wishes to tag a non-Baltic language document)[15]
- *Tagger.exe* – the Tilde's Baltic language POS-tagging web service interface on Windows.
- *tagger.sh* – the Tilde's Baltic language POS-tagging web service interface on Linux.
- *Perl* (Windows - *Strawberry Perl* v5.12.1; *Linux – Perl* v5.10.1).
- *Python (Windows – Python v2.7.1; Linux – Python v2.6.5)*

Tilde's wrapper system's for CollTerm system requirements are as follows:

- For tagging:
  - A *Linux* or *Windows* (*XP* or newer) operating system;
  - 1 or more GB RAM (the accessible RAM depends on the input data file size and may be larger if large (more than 100MB) documents will be processed!);
  - *Intel® Pentium® 4* CPU 3.00GHz, 2992 Mhz, 1 Core(s), 2 Logical Processors or faster.

The system requirements shown are based on a *Windows* based testing system used for Latvian and Lithuanian *CollTerm* evaluation. Faster performance can be achieved using a faster system and for larger annotated corpora more RAM can be necessary.

[15] TreeTagger is available only for research, evaluation and teaching purposes as defined in the license http://www.ims.uni-stuttgart.de/~schmid/Tagger-Licence; for commercial application, the user will have to use a different POS-tagger.

The fast term annotation tool included in the toolkit (*TESimpleAnnotator* – runs only on *Windows*) depends on:

- *Microsoft .NET Framework 4.0 Redistributable*

The system requirements for *TESimpleAnnotator* are as follows:

- *Windows* (*XP SP2* or newer) operating system;
- 2 or more GB RAM;
- *Intel® Pentium® 4* CPU 3.00GHz, 2992 Mhz, 1 Core(s), 2 Logical Processors or faster.

4.1.4 Installation

The *Tilde's wrapper system for CollTerm* does not require installation. Simply copy the whole "*TildeCollTermWrapper*" directory to a directory from where you would like to run the term extraction and execute the *Perl* workflow scripts whenever it is necessary using a *Perl* interpreter (for example, *Strawberry Perl* on *Windows*) from the command line (Command Prompt or PowerShell on Windows or any programming language that supports shell executions).

The user will have to create a property file in order to execute term tagging. Sample property files are located within the "*Sample_Data*" subdirectory of the "Tilde*CollTermWrapper*" directory – "*##_exec_plain.prop*" for plaintext tagging and "*##_exec_tabSep.prop*" for testing and tab-separated document tagging ("*##*" stands for the respective language code – "*lv*" - Latvian, "*lt*" – Lithuanian and "*en*" - English). The user has to change all "*phraseN*" and "*stopN*" property values according to the correct user system's local paths to the corresponding files located in the "*Sample_Data*" directory (the default values refer to relative addresses and are valid only if the working directory is the *Tilde's Wrapper system's for CollTerm* root directory).

Dependency installation on a *Linux* OS:

- For installation of *Perl* refer to http://www.perl.org/get.html.

Dependency installation on *Windows* OS:

- For installation of *Perl* refer to http://strawberryperl.com/.

For installation of *.NET Framework 4.0 Redistributable* refer to http://www.microsoft.com/download/en/details.aspx?id=17718

4.1.5 Execution instructions

The *Tilde's wrapper system for CollTerm* similarly to the *TildeNER* system contains external and internal execution scripts (however, an advanced user may also execute the internal execution scripts). The toolkit also provides a term annotation tool (*TESimpleAnnotator*) that can be used to acquire test data.

The general use case scenario to test *CollTerm* performance is as follows:

1. Annotate test data with the ***TESimpleAnnotator*** tool;
2. Pre-process the annotated documents with the script "***PreprocessAnnotatedDataDirectory.pl***" (running it on a directory);

3. Tag the pre-processed documents and evaluate the results with the script "***TermTagDirectory.pl***" (running it on a directory). Note that the user will need to provide a property file (see 4.1.6.9) that contains absolute paths to a phrase table a stop-word list and an *IDF* (*Inverse Document Frequency*) list; therefore the sample data property files (see 4.1.5.6) will have to be adjusted manually once before the execution.

The general use case scenario to Term-tag a single document is as follows: tag the document with the script "***ExecuteCollTermOnFile.pl***".

4.1.5.1 TESimpleAnnotator

In order to evaluate *CollTerm* performance on the Baltic languages an annotation tool was developed in order to allow fast annotation of plaintext documents (refer to section 4.1.6.1 for a format description). The tool saves the annotated documents in an annotated data format described in section 4.1.6.2.

For a user manual and term mark-up guidelines refer to the document "***Term Markup Guidelines.docx***" that can be found in the "*TESimpleAnnotator*" subdirectory of the "*TildeCollTermWrapper*" directory. The annotation tool ("TESimpleAnnotator.exe") can be found in the same directory.

4.1.5.2 External execution scripts

The external execution scripts provide the main functionality of *Tilde's Wrapper System for CollTerm*. The scripts are also part of the general use case scenarios. In order to provide assistance in execution of the scripts the *Tilde's Wrapper System for CollTerm* package contains predefined *Bash* ("*sh*"; for *Linux*) and *Batch* ("*bat*"; for *Windows*) scripts in the form "*RUN-###.bat*" or "*RUN-###.sh*" (where "*###*" is the name of the external execution script, which command is executed by the script, for instance, "*RUN-PreprocessMuc7DataDirectory.bat*"). The scripts make use of sample property files, phrase tables, IDF lists and stop-word files in the "*Sample_Data*" directory and the input data (and also output data after execution) in the "*TEST*" directory. The scripts operate on data in Latvian (the user has to make modifications to the scripts and provide additional data for other language support).

4.1.5.2.1 Test data pre-processing

Once the term-annotated test data is created (and the format is compliant to the annotated data specified in 4.1.6.2), the user can use the script "***PreprocessAnnotatedDataDirectory.pl***" to perform all required data pre-processing.

The script performs data pre-processing on a single directory (subdirectories are not processed) that contains annotated documents. For each file it separates the term annotation from the plaintext, tokenizes, POS-tags and lemmatizes the plaintext and combines the tab-separated outcome of the plaintext with the separated term annotation in a tab-separated data file (see 4.1.6.3 for the data format description).

The command line to call the pre-processing for a single directory is as follows:

```
perl ./PreprocessAnnotatedDataDirectory.pl [1: Input directory] [2: Output directory] [3: Input file extension] [4: Output file extension] [5: Language] [6: POS-tagger]
```

The script requires in total six arguments passed to the script in a fixed order:

1. The source (input) data directory path.
2. The target (output) data directory path.
3. The input file extension (suggested is "txt" for annotated plaintext).
4. The output file extension (suggested is "gold" for human annotated data).
5. The language of the input documents. The language has to be supported by the POS-tagger.
6. The POS-tagger to use for pre-processing.

Available POS-tagger and language pairs are defined in section 3.1.5.5 of the *TildeNER* POS-tagging module. For information on how to add other POS-taggers refer to the section 3.1.7.

The script depends on "***ProcessDirectory.pl***" and "***PrepareTEData.pl***" scripts and in a general use case has to be executed only once – to prepare annotated data.

For testing purposes and to provide execution examples "*RUN-PreprocessAnnotatedDataDirectory.bat*" (*Windows*) and "*RUN-PreprocessAnnotatedDataDirectory.sh*" (*Linux*) scripts are provided. The scripts are preconfigured to execute "*PreprocessAnnotatedDataDirectory.pl*" on term-annotated documents (with "*txt*" extensions) located in directory "*./TEST/gold_plaintext_in*" using the POS-tagger "*Tagger*" for Latvian "*lv*". Results will be saved in "*./TEST/gold_tabsep_out*".

4.1.5.2.2 Unlabeled data pre-processing

The toolkit also provides unlabeled data directory preprocessing, which is done using the script "***TagUnlabeledDataDirectory.pl***". The script is identical to the *TildeNER* unlabeled data pre-processing script defined in section 3.1.5.2.2. This script, however, is not essential as the next two scripts do not explicitly require data in a pre-processed format (but the data may be also provided in a pre-processed format nonetheless).

For testing purposes and to provide execution examples "*RUN-TagUnlabeledDataDirectory.bat*" (*Windows*) and "*RUN-TagUnlabeledDataDirectory.sh*" (*Linux*) scripts are provided. The scripts are preconfigured to execute "*TagUnlabeledDataDirectory.pl*" on unlabeled plaintext documents (with "*txt*" extensions) located in directory "*./TEST/unlabeled_plaintext_in*" using the POS-tagger "*Tagger*" for Latvian "*lv*". Results will be saved in "*./TEST/unlabeled_tabsep_out*".

4.1.5.2.3 Tagging of terms in a single document

In order to execute term extraction on a single plaintext or pre-processed document, the script "***ExecuteCollTermOnFile.pl***" has to be used. The script allows multiple executions of *CollTerm* for up to four times. The multiple executions are required as *CollTerm* extracts only fixed length n-grams, but terms may be of different lengths. The *CollTerm* execution of particular length n-grams is controlled by the "*execN*" property in the property file. If the property is set to "true", the terms of the n-gram length "*N*" will be extracted. Every execution will cause a term list file to be created. A threshold will be applied to the term list file to filter unlikely n-grams. After all executions all term list files will be combined with a tab-separated pre-processed document and if required also a term annotated result file will be created.

The command line to call the term extraction for a single plaintext file is as follows:

```
perl ./ExecuteCollTermOnFile.pl [1: Input file] [2: Output file] [3:
Property file] [4: IDF list file] [5: Keep temporary files]
```

The script requires in total four arguments passed to the script in a fixed order (the last one is optional):

1. The path to the input file. If the property file defines that POS-tagging should be used ("*execPosTagger = true*"), the input file has to be in the plaintext format (see section 4.1.6.1); otherwise the input file has to be in one of the pre-processed data formats (4.1.6.3, 4.1.6.4 or 4.1.6.5).
2. The path to the output file where results will be written. If the property file defines that POS-tagging should be used ("*execPosTagger = true*"), the result file will be created in the annotated plaintext data format (see section 4.1.6.2); otherwise the result file will be created in the tab-separated term-tagged format (see section 4.1.6.5).
3. The wrapper system's property file that defines all required data pre-processing and term extraction properties (see section 4.1.6.9). Note that the sample property files defined in the section 4.1.5.6 will have to be updated to reflect the user system's local paths.
4. The path to an IDF list file that has been compiled using a general domain corpus. The tool is bundled with three language IDF files precompiled (Latvian, Lithuanian and English). For a format description see section 4.1.6.8.
5. The indicator, whether to keep temporary files. If "1" temporary files will be kept.

The script depends on the "***TEUtilities.pm***", "***Tag.pm***" and "***TEPostprocess.pm***" Perl modules and the ***CollTerm*** system. The script will have to be run once for each document.

For testing purposes and to provide execution examples:

- For plaintext to term-annotated plaintext tagging the "*RUN-ExecuteCollTermOnFile-plaintext.bat*" (*Windows*) and "*RUN-ExecuteCollTermOnFile-plaintext.sh*" (*Linux*) scripts are provided. The scripts are preconfigured to execute "*ExecuteCollTermOnFile.pl*" so that input data is taken from the file "*./TEST/plaintext_in.txt*", the "*./Sample_Data/lv_exec_plain.prop*" property file is used in tagging; the POS-tagger "*Tagger*" for Latvian ("*lv*") is used. The results will be saved in "*./TEST/muc-7_plaintext_out.txt*".

- For tab-separated (POS-tagged and lemmatized) to term-annotated plaintext tagging the "*RUN-ExecuteCollTermOnFile-tabsep.bat*" (*Windows*) and "*RUN-ExecuteCollTermOnFile-tabsep.sh*" (*Linux*) scripts are provided. The scripts are preconfigured to execute "*ExecuteCollTermOnFile.pl*" so that input data is taken from the file "*./TEST/tabsep_in.pos*" and the "*./Sample_Data/lv_exec_tabsep.prop*" property file is used in tagging. The results will be saved in "*./TEST/annotated_tabsep_out.pos*".

4.1.5.2.4 Tagging of terms in all files of a directory

The script "***TermTagDirectory.pl***" allows execution of *CollTerm* on all files in a single directory. The script also allows optionally evaluating the results if pre-processed test data is passed to the script. As the script calls the "***ExecuteCollTermOnFile.pl***" script, for input/output data formats refer to the section 4.1.5.2.3.

The command line to call the term-tagging for a single directory is as follows:

```
perl ./TermTagDirectory.pl [1: Input directory] [2: Output directory] [3: Input file extension] [4: Output file extension] [5: Property file] [6: IDF list file] [7: Evaluation result file]
```

The script requires in total six arguments passed to the script in a fixed order (the last two are optional):

1. The directory from which to read the input files.
2. The directory to which the term-tagged files will be written.
3. The extension (suffix before the dot) of the input files.
4. The extension (suffix before the dot) of the output files.
5. The path to the property file (see section 4.1.6.9). Note that the sample property files defined in the section 4.1.5.6 will have to be updated to reflect the user system's local paths.
6. The path to an IDF list file that has been compiled using a general domain corpus. The tool is bundled with three language IDF files precompiled (Latvian, Lithuanian and English). For a format description see section 4.1.6.8.
7. The evaluation file path (optional and only if test data is passed as the input data! May be empty if the last parameter is required). This file (if defined) will be created by the *Perl* script.

The script depends on the "***ExecuteCollTermOnFile.pl***" script. This is the most important script after the annotated data pre-processing script as both of these scripts in a combination allow evaluation of the *CollTerm* tool's performance if annotated test data is provided.

For testing purposes and to provide execution examples:

- For plaintext to term-annotated plaintext tagging the "*RUN-TermTagDirectory-plaintext.bat*" (*Windows*) and "*RUN-TermTagDirectory-plaintext.sh*" (*Linux*) scripts are provided. The scripts are preconfigured to execute "*TermTagDirectory.pl*" so that input data is taken from the directory "*./TEST/unlabeled_plaintext_in*" ("*txt*" files), the

“*./Sample_Data/lv_exec_plain.prop*” property file is used in tagging; the POS-tagger “*Tagger*” for Latvian (“*lv*”) is used. The results will be saved in the directory “*./TEST/annotated_plaintext_out*”.

- For tab-separated (POS-tagged and lemmatized) to term-annotated plaintext tagging the “*RUN-TermTagDirectory-tabsep.bat*” (*Windows*) and “*RUN-TermTagDirectory-tabsep.sh*” (*Linux*) scripts are provided. The scripts are preconfigured to execute “*TermTagDirectory.pl*” so that input data is taken from the directory “*./TEST/unlabeled_tabsep_in*” (“*pos*” files) and the “*./Sample_Data/lv_exec_tabsep.prop*” property file is used in tagging. The results will be saved in the directory “*./TEST/annotated_tabsep_out*”.
- For gold-annotated tab-separated (POS-tagged and lemmatized) to term-annotated plaintext tagging with evaluation the “*RUN-TermTagDirectory-tabsep+gold.bat*” (*Windows*) and “*RUN-TermTagDirectory-tabsep+gold.sh*” (*Linux*) scripts are provided. The scripts are preconfigured to execute “*TermTagDirectory.pl*” so that input data is taken from the directory “*./TEST/gold_tabsep_in*” (“*gold*” files) and the *./Sample_Data/lv_exec_tabsep.prop*” property file is used in tagging. The results will be saved in the directory “*./TEST/gold_annotated_tabsep_out*” and the evaluation results will be saved in the file “*./TEST/eval.txt*”.

4.1.5.2.5 Plaintext to term-annotated document list tagging

The ACCURAT workflow for NE/Term mapping allows term tagging of lists of files. Therefore, the script “***ExecuteCollTermOnFileList.pl***” was created. The script tags each plaintext document (for the format refer to section 4.1.6.1) specified in an I/O document pair list (for the format refer to section 3.1.6.7 of the *TildeNER* system) and saves each plaintext document with terms marked with “<*TENAME*>” tags (for the format refer to section 4.1.6.2) in files also specified by the document pair list.

The command line to call the term extraction for a document pair list is as follows:

```
perl ./ExecuteCollTermOnFileList.pl [1: Input file list] [2: Property file]
[3: IDF list file]
```

The script requires in total two arguments passed to the script in a fixed order:

- The path of the I/O document pair list file (for the format refer to section 3.1.6.7 of the *TildeNER* system). Each line of the document contains two tab-separated (“\t” character) entries – the plaintext input file (see section 5.5.1) and the term-annotated output file (see section 5.5.2).
- The wrapper system’s property file that defines all required data pre-processing and term extraction properties (see section 4.1.6.9). Note that the sample property files defined in the section 4.1.5.6 will have to be updated to reflect the user system’s local paths.
- The path to an IDF list file that has been compiled using a general domain corpus. The tool is bundled with three language IDF files precompiled (Latvian, Lithuanian and English). For a format description see section 4.1.6.8.

The script depends on the “***ExecuteCollTermOnFile.pl***” script.

For testing purposes and to provide execution examples:

- For plaintext to term-annotated plaintext tagging the “*RUN-ExecuteCollTermOnFileList-plaintext.bat*” (*Windows*) and “*RUN-ExecuteCollTermOnFileList-plaintext.sh*” (*Linux*) scripts are provided. The scripts are preconfigured to execute “*ExecuteCollTermOnFileList.pl*” so that input I/O file list is taken from the file “*./TEST/plaintext_fileList.txt*” and the “*./Sample_Data/lv_exec_plain.prop*” property file is used in tagging (the POS-tagger “*Tagger*” for Latvian (“*lv*”) is used).
- For tab-separated (POS-tagged and lemmatized) to term-annotated plaintext tagging the “*RUN-ExecuteCollTermOnFileList-tabsep.bat*” (*Windows*) and “*RUN-ExecuteCollTermOnFileList-tabsep.sh*” (*Linux*) scripts are provided. The scripts are preconfigured to execute “*ExecuteCollTermOnFileList.pl*” so that input I/O file list is taken from the file “*./TEST/tabsep_fileList.txt*” and the “*./Sample_Data/lv_exec_tabsep.prop*” property file is used in tagging.

4.1.5.3 Internal execution scripts

4.1.5.3.1 Pre-processing a single term annotated document

In order to pre-process term annotated data for testing the script “***PrepareTEData.pl***” is provided. For a single term annotated document (the format is specified in 4.1.6.2) the script separates the term annotation from the plaintext, tokenizes, POS-tags and lemmatizes the plaintext and combines the tab-separated outcome of the plaintext with the separated term annotation in a tab-separated data file (see 4.1.6.3 for the data format description).

The command line to call the pre-processing for a single file is as follows:

```
perl PrepareTEData.pl [1: Language] [2: POS-tagger] [3: Input file] [4:
Output file] [5: Delete temp files]
```

The script requires in total five arguments passed to the script in a fixed order (the last one is optional):

1. The language of the input document. The language has to be supported by the POS-tagger.
2. The POS-tagger to use for pre-processing.
3. The input file path.
4. The output file path.
5. Indicator, whether to delete temporary files. “-D” means that temporary files will be deleted.

The script depends on “***Tag.pm***” and “***TEPreprocess.pm***” modules.

4.1.5.3.2 Evaluating terminology extraction

The script “***TEEvaluation.pl***” allows the user to evaluate term-tagged tab-separated documents with gold standard term-tagged tab-separated documents. The script evaluates the precision, recall, accuracy and F-measure ($F = 2 \times \frac{P \times R}{P+R}$) of the two token categories ("*B-TERM*" and "*I-TERM*"), the full terms and the total (average system performance) for single tokens (*TERM_TOKENS*) by providing two directories – a gold data directory (for data formats refer to section 4.1.6.3) and a test result data directory (for data formats refer to section 4.1.6.5). The script requires for the directories to have equal file names (extensions/suffixes before the dot in file names may differ). A file is produced, which contains evaluation results. A sample file contents is as follows:

```
FULL_TERMS    98.43     98.04     -         98.23
TERM_TOKENS   99.41     97.67     99.75     98.53
I-TERM        100.00    94.38     99.84     97.11
B-TERM        99.21     98.82     99.84     99.01
```

The columns in the tab separated result file represent the following in the exact sequence: result category, recall, precision, accuracy and F-measure. For full terms accuracy results will not be given (accuracy can be estimated on single token performance only and not on multiple token sequences as the interpretation of non-term entities and their possible sequences is ambiguous).

The command line to call the evaluation script is as follows:

```
perl ./TEEvaluation.pl [1: Gold data directory] [2: Test result directory]
[3: Output file]
```

The script requires in total three arguments passed to the script in a fixed order:

1. The path of the directory containing the human annotated/gold documents.
2. The path of the directory containing the test result documents.
3. The path to the evaluation result output file.

The script does not depend on any other system module or script.

4.1.5.3.3 Executing a process on a directory

Similarly to *TildeNER*, the term tagging and data pre-processing workflows require all files in a directory to be processed. Therefore, the same "***ProcessDirectory.pl***" script as in *TildeNER* is used. For more information refer to the section 3.1.5.3.1.

4.1.5.4 Internal modules

Internal modules are not supposed to be called externally (manually) by the user, however the scripts contain many useful functions, which could be useful to the user if he/she would want to extend the system.

4.1.5.4.1 Data pre-processing module

The *Perl* module "***TEPreprocess.pm***" provides a set of functions used in document pre-processing before term extraction. A list of functions used in the workflows is as follows:

1. ***RemoveEmptyLines*** - Removes empty lines from a tab-separated document. According to a selected option all empty lines are kept ("1"), all empty lines, where 2 or more empty lines are one after another are kept ("2"), all lines are

removed (all other values). The term pre-processing workflows allow all empty lines to be kept.

2. ***Detagger*** – Splits term tags and plaintext from a term annotated document. Term tags and the plaintext are saved in separate documents. After calling this method, the plaintext can be POS-tagged.
3. ***AddNewTags*** – After POS-tagging of a plaintext document, this method combines the tab-separated tokenized, POS-tagged and lemmatized document with the term tags, which were split from the plaintext using the ***Detagger*** method.
4. ***FindTokenPos*** – If the POS-tagger used in POS-tagging of a plaintext document does not produce positional token information that would allow term markup to be applied to the plaintext (for instance, TreeTagger does not produce any – line from, column from, line to, column to), the method analyzes the POS-tagged document and the plaintext and assigns positional information for each token.

The module does not depend on any other system module.

4.1.5.4.2 Data post-processing module

The *Perl* module "***TEPostprocess.pm***" provides a set of functions used in term-tagged document post-processing. A list of functions used in the workflows is as follows:

1. ***TagTermsFromMultipleFiles*** – the method creates a term-tagged tab-separated data file from a tab-separated data file (all of the formats described in sections 4.1.6.3, 4.1.6.4 and 4.1.6.5 are supported) and an array of term lists (as extracted by *CollTerm*). The array has to be sorted by the n-gram length of the terms in the term list files in a descending order. The method tags all terms from the term list files in the tab-separated document and saves the new tab-separated document as a result.
2. ***TaggedTokensToTaggedPlaintext*** – the method applies term markup from a tab-separated data file (see section 4.1.6.5) to a plaintext data file (see section 4.1.6.1) and saves the result as a term annotated data file (see section 4.1.6.2).

The module does not depend on any other system module or script.

4.1.5.4.3 Tokenization, lemmatization and POS-tagging module

Identically to the *TildeNER* system the *Tilde's wrapper system for CollTerm* uses the *Perl* module "***Tag.pm***", which provides POS-tagging functionality for data pre-processing. For a description of the module refer to the section 3.1.5.4.5 of the *TildeNER* system's description.

4.1.5.4.4 Utility functions for term extraction

The Perl module "***TEUtilities.pm***" provides a set of useful utility functions. The list of functions is as follows:

1. ***ReadPropertyFile*** – the method reads a property file (refer to section 4.1.6.9) and returns the property keys and values in a hash table.

2. ***ApplyTermThreshold*** – as *CollTerm* execution will cause all valid n-grams to be extracted, the method allows applying a threshold to a term list file. A new file is created as a result.

The module does not depend on any other system module.

4.1.5.4.5 CollTerm module

The most important module is the *CollTerm* module as the wrapper system only provides functionality to easily execute the *CollTerm* system and evaluate the results if necessary. As the workflows already contain *CollTerm* execution sequences, the user does not require additional knowledge about this module.

4.1.5.5 POS-taggers included in the toolkit

For a description of POS-taggers supported by the wrapper system, refer to the *TildeNER* POS-tagger section 3.1.5.5.

4.1.5.6 Data samples included in the toolkit

Within the ACCURAT Toolkit we provide also sample data for the execution of CollTerm and the Tilde's wrapper system for CollTerm. The sample data is included in the "***Sample_Data***" subdirectory of the "*TildeCollTermWrapper*" directory. The provided data is as follows:

1. Stop-word list files for Latvian, Lithuanian and English ("*STOP_LV.txt*","*STOP_LT.txt*" and "*STOP_EN.txt*");
2. Phrase tables for Latvian, Lithuanian and English ("*LV_TERM_POS.txt*","*LT_TERM_POS.txt*" and "*EN_TERM_POS.txt*");
3. Sample property files for the execution of *Tilde's wrapper system for CollTerm* on plaintext documents (valid also for the *NE/Term Mapping Workflow* of the ACCURAT Toolkit) for Latvian, Lithuanian and English ("*lv_exec_plain.prop*", "*lt_exec_plain.prop*" and "*en_exec_plain.prop*");
4. Sample property files for the execution of Tilde's wrapper system for CollTerm on tab-separated pre-processed documents (see section 3.1.6.4 for a format description) (not valid for the NE/Term Mapping Workflow) for Latvian,Lithuanian and English ("*lv_exec_tabsep.prop*", "*lt_exec_tabsep.prop*" and "*en_exec_tabsep.prop*");
5. IDF list files for Latvian, Lithuanian and English ("*LV_IDF.txt*", "*LT_IDF.txt*" and "*EN_IDF.txt*").

4.1.6 Input/output data formats

All documents used in the Tilde's wrapper system and the *CollTerm* system should be encoded using UTF-8 encoding. Other encodings are not supported. The systems are BOM insensitive; however, it is advised for the user to strip the BOM characters before processing data as some POS-taggers may operate incorrectly. It is also advised because of the same reason to remove all control characters except *LF* ("*\n*"), *CR* ("*\r*") and "*TAB*" ("*\t*") from the input data.

All input and output data files have to contain file extensions (for instance, “*.*txt*” for plaintext documents, “*.*pos*” for POS-tagged documents, etc.); otherwise, the system may perform unexpectedly.

4.1.6.1 Plaintext format

The first and the most simple data format for named entity tagging is plaintext. A plaintext document is not allowed to contain mark-up within the text. All mark-up will be considered as part of the plaintext and processed together with the text.

4.1.6.2 Term annotated data format

The manual annotation tool *TESimpleAnnotator* and the tagging workflow script “***ExecuteCollTermOnFile.pl***” generates documents in a format where each term is tagged using “<*TENAME*>” tags. All other tags are considered as a part of the plaintext document. A sample annotated document (shown is only one sentence) is as follows:

```
Loga   augšējā   labajā   stūrī   obligāti   noklikšķiniet   uz   <TENAME>Vadības
paneļa</TENAME>  sākumlogs  ,  lai  <TENAME>Klasiskais  skats</TENAME>  nebūtu
aktīvs.
```

4.1.6.3 Tab-separated testing data format

The annotated data pre-processing workflow (***PreprocessAnnotatedDataDirectory.pl***) produces data in a tab-separated, POS-tagged, tokenized, lemmatized and term-tagged format.

The tab-separated format contains (in a fixed and non-changeable sequence):

1. The original word form
2. Part of speech
3. Lemma
4. Morpho-syntactic tag (may be also non-positional, but as a sample the Tilde’s positional 28 category morpho-syntactic tag is given)
5. Line in which the token starts in the original plaintext document
6. Column in which the token starts in the original plaintext document
7. Line in which the token ends in the original plaintext document
8. Column in which the token ends in the original plaintext document
9. Term category

A sample pre-processed test data sentence is as follows:

```
Kompaktdiska      N     kompaktdisks      N-msg---------n-----------f-   0
      0     0     11    B-TERM
vai   C     vai   C---------------------c---l-   0     13    0     15    O
DVD   -     DVD   ----------------------------   0     17    0     19    B-
TERM
diska N     disks N-msg---------n-----------l-   0     21    0     25    I-
TERM
```

```
ierakstīšana      N     ierakstīšana      N-fsn---------n-----------l-  0
      27    0     38    O
programmā   N     programma   N-fsl---------n-----------l-  0     40    0
      48    B-TERM
Windows     N     Windows     N-fsn---------n-----------f-  0     50    0
      56    O
Media -     Media ----------------------------  0     58    0     62    O
Player      -     Player      ----------------------------  0     64    0
      69    O
```

A wider description of the format's first seven columns is given in section 3.1.6.3.

The format specifies that each token has to have a term category (column eight). Non-terms receive the category "*O*". The first token in a term receives a category that starts with "*B-*"; all other tokens within a term receive a category that starts with an "*I-*". It is not allowed for a term to start with an "*I-*" token. All possible (currently supported) categories are:

1. "*B-TERM*" – the first token of a term;
2. "*I-TERM*" – the other (not first) tokens of a term;
3. "*O*" – not a term.

The document format also requires 2 empty lines to be present for newline characters in the plaintext. Sentences may be separated using one empty line, but that is not mandatory if the TreeTagger "*SENT*" category is used to mark sentence ending characters.

4.1.6.4 Tab-separated pre-processed unannotated data format

The data format is identical to the pre-processed unannotated data format used in named entity recognition (described in section 3.1.6.4).

4.1.6.5 Tab-separated term-tagged data format

The tagging workflow script "***ExecuteCollTermOnFile.pl***" generates also tagged documents in a tab-separated format (before applying markup to plaintext). The format is similar to the pre-processed data format described in section 3.1.6.3. The difference is that this format has an additional column – the ranking of the extracted terms (assigned by *CollTerm*). A sample sentence of the format is as follows:

```
Diska        N   disks        N-msg---------n-----------f- 0  0   0  11  B-TERM 0.9
vai          C   vai          C---------------------c---l- 0  13  0  15  O      0
DVD          -   DVD          ---------------------------- 0  17  0  19  B-TERM 0.7
diska        N   disks        N-msg---------n-----------l- 0  21  0  25  I-TERM 0.7
rakstīšana   N   rakstīšana   N-fsn---------n-----------l- 0  27  0  38  O      0
programmā    N   programma    N-fsl---------n-----------l- 0  40  0  48  B-TERM 0.8
Windows      N   Windows      N-fsn---------n-----------f- 0  50  0  56  O      0
Media        -   Media        ---------------------------- 0  58  0  62  O      0
Player       -   Player       ---------------------------- 0  64  0  69  O      0
```

All tokens of a single term will contain the same rank as the rank is assigned to full terms and not individual tokens.

4.1.6.6 Stop-word list format

The stop-word list file required by *CollTerm* has to contain one stop-word per line. A sample of the stop-word list is as follows:

```
in
on
at
before
```

4.1.6.7 Phrase table format

As various POS taggers have different tagsets the *CollTerm* tool supports an external phrase table of allowed POS tag sequences, which is required for valid term extraction. The configuration file hast to be a tab separated file, which contains in each line a single POS tag sequence. Each POS tag may be also represented with a regular expression, in order to support also positional tagsets and allow flexible phrase definition.

A sample of the phrase configuration file is as follows:

```
N        N          N
N
^n…g.*   ^n.*
```

The last example shows usage of regular expressions. The first two examples require for the whole POS tag to match the given value.

4.1.6.8 IDF list file format

The IDF list file is a tab-separated text file that contains one entry per line. Each line contains one token and its IDF score. It is advised to calculate the score on a relatively large general domain corpus (one million running words should be sufficient). All entries are sorted in an ascending order.

A sample extract from the Latvian IDF sample file is given below:

```
.               0.0025
,               0.0124
būt             0.0735
un              0.1149
no              0.2999
```

4.1.6.9 CollTerm execution property file

In order to execute *CollTerm* in the “***ExecuteCollTermOnFile.pl***” and “***TermTagDirectory.pl***” scripts, a property file is required, which specifies how many times and with which parameters *CollTerm* should be executed. The properties also specify, whether the input data has to be pre-processed before *CollTerm* execution.

A property file contains one parameter per line (comments are allowed only at the beginning of each line starting with the symbol “#”; also empty lines are allowed). Each property starts with an identifier, which is followed by an equation symbol “=”. The value of the property is everything (trimming both end whitespaces) that is after the equation symbol.

All supported properties are:

1. ***execPosTagger*** – specifies, whether POS-tagging has to be called for the input data. If "*true*", results will be saved in the annotated plaintext format. If "*false*", results will be saved in the tab-separated format.
2. ***POSTagger*** – specifies, which tagger to use in POS-tagging. Supported are all "Tag.pm" supported values.
3. ***Language*** – specifies, which language to use in POS-tagging. Supported are all "*Tag.pm*" supported values.
4. ***execN*** – if "*true*", n-grams of length "*N*" will be extracted (supported are n-grams up to the length of four tokens).
5. ***methodN*** – the *CollTerm* method to use for n-gram of length "*N*" extraction. The available values are:
   a. ***"dice"*** for the Dice coefficient
   b. ***"mi"*** for the modified mutual information
   c. ***"chisq"*** for the chi-square statistic
   d. ***"ll"*** for the log-likelihood ratio and
   e. ***"tscore"*** for the t-score statistic
6. ***lenN*** – the length of the extracted n-grams (the length has to be equal to "*N*").
7. ***minFreqN*** – the minimum frequency of an n-gram to be considered as a possible term.
8. ***thresholdN*** – the decimal threshold of extracted n-grams of length "*N*". As *CollTerm* will extract all n-grams, only the ones ranked higher than the threshold will be used in tagging.
9. ***phraseN*** – the phrase table to use for n-gram of length "*N*" extraction. The paths to an existing local phrase table file. See section 4.1.5.6 for sample data. Note that the sample property files have to be adjusted to reflect the user's local system's paths!
10. ***stopN*** – the stop-word list to use for n-gram of length "*N*" extraction. The paths to an existing local stop-word list file. See section 4.1.5.6 for sample data. Note that the sample property files have to be adjusted to reflect the user's local system's paths!
11. ***posN*** – the positions of word forms, POS-tags and lemmas in the tab-separated pre-processed documents passed to *CollTerm* for n-gram of length "*N*" extraction.

See section 4.1.5.6 for sample property files.

4.1.7 Integration with external tools

To integrate the *Tilde's wrapper system for CollTerm* into another system that requires term extraction, the target system has to be able to execute command line commands. All standard execution commands are described in section 4.1.5.

For information on how to add another POS-tagger refer to the section 3.1.7 of the *TildeNER* system (both systems share the same POS-tagger integration solution).

4.1.8 Contact

For further information and technical support installing and/or running this tool, please email to Mārcis Pinnis (marcis.pinnis@tilde.lv; for questions regarding the Perl wrapper system) and Marko Tadic (marko.tadic@ffzg.hr; for questions regarding the CollTerm system).

4.1.9 Useful references

The POS-tagged data standard is an extended version of the TreeTagger format:

http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/.

4.2 KEA wrapper

4.2.1 Overview and purpose of the tool

In the multi-lingual NE and term mapper (see section 5.1) we make use of *KEA* to tag terms for the English documents. The system is an existing tool and is not implemented within the ACCURAT project. The output of this system is, therefore, different from the input format of the NE and term mapper. The wrapper:

- enable that the output of *KEA* is of the same format as the input files to the mapper
- provides a scenario to users where the mapper can be run on existing annotated data
- enables the users to use other TE systems to prepare the input to the mapper.

4.2.2 Changes from the previous version

There are no changes from the previous version.

4.2.3 Software dependencies and system requirements

The wrapper is implemented in the programming language Java. It requires the following settings to run:

- *JRE* (*Java Runtime Environment*) 1.6
- 1+ GB RAM

4.2.4 Installation

The wrapper does not require any installation.

4.2.5 Execution instructions

The *KEA wrapper* can be run using the following command:

```
java -jar KEATEWrapper.jar [fileList]
```

fileList: a tab separated list of files. On each line the file contains the file name (with the full path) to be annotated by the wrapper, a tab for separation and the file name (with the full path) where the results of the annotation should be saved. The output file will be automatically generated by the wrapper. For a sample of the format of the file list refer to section 3.1.6.7 of the *TildeNER* system.

Please also make sure that you run the wrapper from the folder where all the required resources are saved. These resources are the entire folders ("*docs*", "*data*", "*workingFolderForTE*" and "*testdocs*") and are provided with the wrapper.

4.2.6 Input/Output data formats

Input to the wrappers is text that is encoded in *UTF-8*.

Output of the *KEA* wrapper is text with terms tagged. Terms are tagged with *<TENAME>term</TENAME>* (for a more detailed format description refer to section 4.1.6.2 of the *Tilde's wrapper system for CollTerm*). For more details see the description for the NE and term mapper.

4.2.7 Contact

For further information and technical support installing and/or running this tool, please email to Ahmet Aker: a.aker@dcs.shef.ac.uk.

4.3 CollTerm – a tool for collocation extraction

4.3.1 Overview and purpose of the tool

CollTerm is a tool for collocation and term extraction, i.e. extracting word sequences that co-occur more than by chance or that occur significantly more frequently in a domain corpus than in a reference corpus. This tool extracts collocation and term candidates by applying POS/MSD phrase filters, stop-word filters and computing different statistical association measures between sequences of words. If an *IDF* list file is present, the tool takes into account the significance of the term frequency regarding a reference corpus. The output of the tool is a list of collocation and term candidates ranked by their strength.

4.3.2 Changes from previous version

There are several differences between versions 0.3 and 0.7. One of them is the name of the tool. Since in the newest version of the tool extracts not only collocations of length 2-4, but terms of length 1-4 by taking into account a reference corpus, the name of the tool was changed to *CollTerm*.

Specific changes from version 0.3 are these:

- term extraction
  - The *IDF* value for each lemma is computed from a reference corpus with an additional script ("calculate_idf.py")
  - The *TF*IDF* ranking method is added. Thereby the possibility of extracting terms of length 1 is given.
  - If the "*idf*" argument (and thereby the *IDF* file) is given and any other but the *TF*IDF* ranking method is used, a linear combination of the ranking method (a collocation extraction method) and the average *IDF* of the n-gram is computed. In that way the possibility of combining clues about co-

occurrence on one side and specificity regarding a reference corpus on the other side is given.

- output control
  - Collocation and term candidates can be given as lemma sequences, most frequent token sequences and all token sequences.
  - The ranking method result can be normalized to a specified range.
  - The list of collocation and term candidates can be controlled also by a threshold (minimum value) of the ranking method.
  - Collocation and term candidates can be given without the ranking method results (appropriate for further processing)

4.3.3 Software dependencies and system requirements

The tool is platform independent. It can be run on python2.6 or python2.7 on any platform that provides the environment for this programming language. There are no specific hardware requirements.

4.3.4 Installation

The system requires a working Python interpreter. No installation of the tool is required.

4.3.5 Execution instructions

The tool has to be executed in command line by calling the script with the corresponding arguments. The possible arguments are the following:

-i tab separated input file from which to extract terms (mandatory). The argument requires one value – the path to the file.

-s stop-word file (optional). If none specified, the system assumes that no stop-word list for the file is given. The argument requires one value – the path to the file.

-p phrase configuration file (mandatory). The argument requires one value – the path to the file.

-n the maximum number of top ranked terms to be extracted from the input file (optional). The argument requires one value – an unsigned integer. If the argument is not given, all valid n-grams will be extracted.

-t the threshold (minimum score) for term weight (optional). The argument requires one value – a floating point or integer value. If the argument is not given, all valid n-grams will be extracted.

-m the n-gram ranking method which is applied to rank n-grams from the input file (mandatory). The argument requires one value – a string enumerator of the ranking method. Possible string enumerators are:

- "*dice*" for the Dice coefficient
- "*mi*" for the modified mutual information
- "*chisq*" for the chi-square statistic
- "*ll*" for the log-likelihood ratio and

- “*tscore*” for the t-score statistic
- “*tfidf*” for the *TF*IDF* score (accompanied with a mandatory *IDF* file)

-l the length of the n-grams to be extracted (mandatory). The argument requires one value – a natural number (1–4) that specifies the n-gram length. For the n-gram length of 1 only the “*tfidf*” ranking method is applicable.

-o the extracted term list output file (optional). The argument requires one value – the path of the output file. If the argument is not specified, the results are printed to the standard output stream (console).

-pos the positions (column indices) of the tokens (actual word), POS/MSD tags and lemmas in the input file (optional). The argument requires three values – unsigned integers in the format “*[token position] [POS/MSD tag position] [lemma position]*”, for instance, *“-pos 0 2 1*” defines that the token is defined in the first column, the POS/MSD tag is defined in the third column and lemma is defined in the second column. If the argument is not given, a default position (token – 0, POS/MSD tag – 1 and lemma – 2) is assumed.

-min the minimum frequency of a n-gram to be considered as a candidate (optional). The argument requires one value – an unsigned integer. If the argument is not given, the minimum frequency of 5 is assumed.

-prop the property file (optional). The argument requires one value – the path to the file. All properties (except “*-prop*”) can be also passed to the tool using a property file (in this way the user can avoid writing huge commands).

-idf a file with *IDF* weights (optional, but mandatory for the *TF*IDF* ranking method). If the *IDF* weights are given and any other ranking method than *TF*IDF* is used, a linear combination of the average *IDF* score and the ranking method is computed.

-seq type of n-gram output (optional). If the argument is not specified, “0” is assumed.

“0” where each n-gram is represented as a sequence of lemmas

“1” where each n-gram is represented as the most frequent token sequence

“2” where each n-gram is represented with all recorded token sequences with their frequencies

-norm normalizes the output to the [0,x] range (optional). If argument “0” is given, or the argument is not specified, normalization is not performed.

-terms type of term output (optional). If the argument is not specified, “0” is assumed.

“0” – output terms and their corresponding weights (as well as frequency if “*-seq 2*” is defined)

“1” – output terms only (regardless of the “*seq*” argument value)

An example of calling the script is this:

```
python CollTerm.py -i input_text.txt -s stopwords.txt -p phrases.txt -pos 0
5 3 -prop properties.txt -t 0.5 -min 3 -l 2 -m dice - seq 1 -terms 1
```

4.3.6 Input/Output data formats

The input text file has to be a UTF-8 encoded tab-separated file with columns with token, POS/MSD and lemma information. The location (index of the column) in which this data is located can be given by the “*-pos*” argument. An example of the input file (which requires the “*-pos 0 2 1*” argument since token is on the first, POS/MSD on the third and lemma on the second position) is this:

```
Najpoznatija        poznat        Npmsn            0    1
ravnateljica        ravnateljica  Ncfsn            1    1
nekog               nekoji        Pi-msg--n-a--    0    0
```

The stop-word file has to be a UTF-8 encoded file with one stop-word per row. An example of the stop-word file is this:

```
i
je
od
za
```

The phrase configuration file enables two sorts of filters: POS/MSD filters and stop-word filters.

The POS/MSD filters are tab-separated Python regular expressions. More filters can be defined for n-grams of specific length and every n-gram has to satisfy at least one filter.

Some examples for POS/MSD filters are these:

```
N..n.*      N..g.*
A..n.*      Nc.n.*
A.*         A.*         N.*
```

The stop-word filters are tab-separated special symbols: “*STOP*” for a stop word, “*!STOP*” for a non-stop word and “*” for any stop word. Only one stop-word-filter can be defined for n-grams of a specific length and all n-grams of that length have to satisfy this filter. Some examples for stop-word filters are these:

```
STOP        !STOP
*           STOP        *
```

Some or all of the input parameters for the tool can be defined in a property file as well. Each line in property file consists of the argument identifier without the dash “-“ (“*i*”, “*pos*”) followed by the equal sign and the value or values of the argument. Values of arguments can be divided by spaces or defined inside double quotes “"” (if spaces are parts of the argument values). Some examples of the property file lines are these:

```
i = c:\input\in.txt
s= "c:\input directory with spaces\stopwords.txt"
n=500
m=PMI
l=3
o=c:\OutputFile.txt
pos = 3 7 2
```

The output of the tool is written on standard output or in a file (depending on the "*-o*" parameter) with *UTF-8* encoding, without *BOM*, with "*\n*" for newline. Each collocational candidate is represented as a sequence of space-delimited lemmas with its collocational strength delimited by a tab. The data is sorted by collocational strength in descending order. An example of the output is this:

```
uljani repica     0.97
voden kozica      0.93
carski rez        0.90
limfni čvor       0.87
```

The IDF file can be constructed with a script distributed with the *CollTerm* system "*calculate_idf.py*". An example of calling that script is this:

```
python calculate_idf.py reference_corpus.txt idf.txt
```

The reference corpus file should consist of a sequence of lemmata, each in its' row with documents separated by empty rows. An example of the reference corpus file is this:

```
…
velik
dio
zemlja
odron
jadranski
magistrala

pobijediti
sam
iskoristiti
šansa
drugi
…
```

It is recommended to leave only content words in the reference corpus file since the main objective of the *IDF* measure is to differentiate between the specificity of content words in a domain corpus. The control over non-content words can better be obtained by defining stop-words and phrases.

The output of the "*calculate_idf.py*" script is each lemma found in the reference corpus with its' *IDF* weight separated by a tab. An example of the *IDF* file is this:

```
biti                            0.0627
htjeti                          1.0737
godina                          1.5452
sav                             1.5555
moći                            1.5973
imati                           1.796
velik                           1.8268
```

```
još                         1.91
reći                        2.0627
prvi                        2.0848
```

The output of the *CollTerm* tool is written on standard output or in a file (depending on the "*o*" argument) with *UTF-8* encoding, without *BOM*, with "*\n*" for newline. Each term candidate is by default represented as a sequence of space-delimited lemmas with its strength following after a tab character. The data is sorted by strength in descending order. An example of the default output is this:

```
generalan zastupnik         1.0
fiatov logo                 0.83
završan obrada              0.82
zračan jastuk               0.79
štetan plin                 0.68
ugljičan vlakno             0.67
stupanj prijenos            0.67
ugljičan dioksid            0.63
parkirni senzor             0.63
```

By intervening in the default value of the "*seq*" argument ("0"), different n-gram representations can be obtained. By setting the argument value to "1" ("*-seq 1*") each n-gram is represented as the most frequent token sequence of that lemma n-gram:

```
generalnom zastupniku       1.0
fiatovim logom              0.83
završna obrada              0.82
zračnih jastuka             0.79
štetnih plinova             0.68
ugljičnih vlakana           0.67
stupnjeva prijenosa         0.67
ugljičnog dioksida          0.63
parkirnih senzora           0.63
```

While the first output ("*-seq 0*") is more adjusted for machines and further processing (such as annotating the corpus), the second output ("*-seq 1*") is more adjusted for human reading.

If there is a need for all token sequences representing a lemma n-gram, the value "2" for the "*seq*" argument can be used. In this output between each token sequence and the score the frequency of that token sequence is given. An example of such output is this:

```
generalnog zastupnika       5             1.0
generalnom zastupniku       12            1.0
fiatovim logom              5             0.83
završnom obradom            2             0.82
završne obrade              3             0.82
završnoj obradi             4             0.82
```

```
završnu obradu              1              0.82
završna obrada              10             0.82
zračne jastuke              5              0.79
zračnih jastuka             18             0.79
zračni jastuk               7              0.79
zračna jastuka              7              0.79
zračni jastuci              3              0.79
zračnim jastukom            1              0.79
…
```

If only terms are required for further processing, without the need for their weights, the "terms" argument can be used with the value "1". The previous output, in that case (argument combination "*-seq 2 -terms 1*"), is as follows:

```
generalnog zastupnika
generalnom zastupniku
fiatovim logom
završnom obradom
završne obrade
završnoj obradi
završnu obradu
završna obrada
zračne jastuke
zračnih jastuka
zračni jastuk
zračna jastuka
zračni jastuci
zračnim jastukom
…
```

If only lemma n-grams without corresponding weights are needed (for corpus annotation or similar), the argument combination "*-seq 0 -terms 1*" (or just "*-terms 1*" since "0" is the default value for the "*seq*" argument) produces the following output:

```
generalan zastupnik
fiatov logo
završan obrada
zračan jastuk
štetan plin
ugljičan vlakno
stupanj prijenos
ugljičan dioksid
parkirni senzor
```

Normalization of the ranking method results can be obtained with the "*norm*" parameter. The value of the parameter represents the upper bound while the lower bound is always 0. If the value of the parameter is “0”, normalization is not performed. An example of the results normalized to the [0,100] range (“*-norm 100*”) is:

```
generalnom zastupniku        100.0
fiatovim logom               83.29
završna obrada               81.58
zračnih jastuka              78.79
štetnih plinova              68.21
ugljičnih vlakana            66.58
…
```

The *IDF* file is mandatory when using the *TF*IDF* ranking method (the only ranking method capable of extracting unigrams). If an *IDF* file is provided and any other ranking method but *TF*IDF* is used, then a linear combination of the average *IDF* value and the normalized ranking method is computed.

4.3.7 Integration with external tools

Since the tool does not use any tools outside *python2.7*, no integration with external tools is necessary.

4.3.8 Useful references

The implemented association measures were chosen by examining primarily the following literature:

- Stefan Evert: The statistics of word cooccurrences: Word pairs and collocations. PhD thesis, Universität Stuttgart, Institut für Maschinelle Sprachverarbeitung, 2005.
- Pavel Pecina: Lexical association measures: Collocation Extraction. Studies in Computational and Theoretical Linguistics. Institute of Formal and Applied Linguistics, Prague, Czech Republic, 2009.
- Saša Petrović et al: Extending lexical association measures for collocation extraction. Computer Speech and Language 24 (2), 383–394, 2010.

4.4 Terminology Extraction for English and Romanian

4.4.1 Overview and purpose of the tool

The *Terminology Extraction* (TE) tool is designed to identify mono and multi word terminological terms in raw texts. It is designed to work for English and Romanian. In order for it to work, the application needs the input files to be pre-processed. To do so, it calls for the *TTL* web service (hosted at RACAI, *WSDL* file at http://ws.racai.ro/ttlws.wsdl). For more information about the technology it implements, please consult ACCURAT Report D2.3. In order to properly work, the application needs that all the files given in the input file be part of the same domain, as it take them all into account for having enough statistical relevance, in computing the probabilities of various words and expressions of being terms.

4.4.2 Changes from previous version

Aside from bug fixing, there are no functional modifications and/or changes to the user's interface of this tool.

4.4.3 Software dependencies and system requirements

Terminology Extraction is implemented in *C#* using *.Net Framework 4.0*. For machines using *Windows*, the users should install *.Net Framework 4.0*. For machines using *Linux*, the users should use *Mono 2.10* (available from http://www.mono-project.com/Main_Page). The machine should have at least 1GB of RAM.

4.4.4 Installation

TE does not require any special installation apart from *.NET Framework*.

4.4.5 Execution instructions

The command line for *Terminology Extraction* is:

```
TerminologyExtraction.exe --input [DATA_FILE] [--source [LANG]]
[--param [ap]=[TRUE]/[FALSE]]
[--param [kif]=[TRUE]/[FALSE]]
```

where:

"*DATA_FILE*" – Each line in the *DATA_FILE* should contain the path of an input file and the path of an output file, tab separated;

"*LANG*" – The language of the texts; Default: ro;

"*-ap*" – optional argument usable when the input files are already pre-processed and the annotation is compliant with RACAI's *XML* resource format; Default: *FALSE*;

"*-kif*" – optional argument allowing the user to keep the intermediary files; Default: *FALSE*.

4.4.6 Input/Output data formats

The input files are either raw *UTF-8* texts or preprocessed texts (RACAI's *XML* resource format) (see the "*ap*" option of the *TerminologyExtraction* executable in the previous section).

The output is the input text file with terminological terms tagged, according to *MUC-7* style.

4.4.7 Contact

For further information and technical support installing and/or running this tool, please email to Dan Ştefănescu: danstef@racai.ro.

5 Tools for named entity and terminology mapping

This section covers the tools that perform multi-lingual named entity and terminology mapping and are created within the ACCURAT project.
The tools included in this section of the ACCURAT toolkit are:

- *Multi-lingual named entity and terminology mapper* (developed by USFD; see section 5.1);
- *NERA2 language independent named entities mapper* (developed by RACAI; see section 5.2);
- *TerminologyAligner language independent terminology mapper* (developed by RACAI; see section 5.3).
- *P2G: A tool to extract term candidates from aligned phrases* (developed by LT; see section 5.4).

5.1 Multi-lingual named entity and terminology mapper

5.1.1 Overview and purpose of the tool

Many events such as news events are reported in several languages. Each such report will vary in detail. The content is also likely to be tailored to a specific reader group or is entirely influenced by its writer on how he/she sees the entire event. However, what will be common in all the different reports are the use of some named entities and/or some technical terms. Named entities can be person, location or organization names but also other named entity types such as dates, day names or currencies are considered as named entities. As technical term one could regard specific names used, for instance, only in medicine or automotive domain.

Reports in different languages about the same event can be regarded as comparable because they are likely to share some textual units which are translation of each other. Named entities and technical terms can play an important role in finding such textual units. For instance, if two sentences in different languages contain the same named entity or the same technical term it is likely that these sentences contain some translation units. Sentences in different languages which do not share named entities or technical terms are less likely to have such units. However, before such translation units are identified one has to first identify entries in these multi lingual reports which refer to the same named entity or technical term.

We implemented a multi-lingual language independent application which aims to map such entries in reports written in different languages to each other.

5.1.1.1 NE mapping

For Named Entity (NE) mapping we implemented two scenarios. In the first scenario the *NE mapper* takes as input two comparable documents in text format and outputs pair of NEs with scores indicating their level of mapping. On both sides we use *OpenNLP* (http://incubator.apache.org/opennlp/) to identify sentence boundaries. Next, on the English text the mapper applies *OpenNLP NER* to extract English NEs. On the foreign text it uses case information to identify candidates as foreign NEs. It treats all capitalized words as NEs and uses for comparison with the English NEs. Consecutive capitalized words are treated as a single NE. For each word in the beginning of each sentence we compare its lowercase variant with a list of lowercase words. If the lowercase variant is found in the list then it is not treated as NE. After having collected NEs in English and so called NEs in the foreign language we compare each English NE with all the other foreign NEs. The comparison is computed using cognate based methods described in the ACCURAT Deliverable D2.3 "Report on information extraction from comparable corpora".

In the second scenario the mapper uses proper NE identification on both sides. On English side it continues using the *OpenNLP NER*. On the foreign text side it assumes that the NEs are identified using the NER systems described in D2.3. Having both lists of NEs with their types (*PERSON*, *LOCATION*, *ORGANIZATION*) it uses cognate based methods to align them. However, instead of comparing every English NE with every foreign NE it compares every English NE with type *X* with every foreign NE of the same type. For the comparison we use cognate methods described in D2.3.

5.1.1.2 Terminology mapping

USFD applies the same cognate based approach as in NE mapping to align terminologies. On English side an English terminology extractor is used. On the target one the ACCURAT specific tools are used. Extracted terminologies from both sides are aligned using cognate based methods.

For English term extraction the *KEA TE* extractor can be used (http://www.nzdl.org/Kea/). On the foreign sites we use the TE tools described in D2.3.

5.1.2 Changes from previous version

The updated version contains a bug-fix that resolves a system crash on specific input data. We also replaced *KEA TE* with *Tilde's wrapper system for CollTerm* developed by Tilde and FFZG for automatic collocation extraction to extract English terms. Details of the adaptation are described in the ACCURAT project's Deliverable D2.3.

5.1.3 Software dependencies and system requirements

The mapper is implemented in the programming language *Java*. It requires the following settings to run:

1. *JRE* (*Java Runtime Environment*) 1.6
2. 1+ GB RAM
3. *OpenNLP* tools (can be downloaded from http://incubator.apache.org/opennlp/)
4. *KEA TE tool* (is packaged with the tool)

5.1.4 Installation

The mapper does not require any installation.

5.1.5 Execution instructions

The application can be run using the following command:

```
java -jar MapperUSFD.jar [method] [mappingFile] [outputFile] [taggingInfo]
[foreignLang]
```

method:

"*NE*" – select this when you want NE mapping, please specify also

taggingInfo: *NE0*, *NE1*, *NE2*

(NE0 means both input files are not *NE* tagged, *NE1* means the foreign files are NE tagged according to *MUC-7* style, *NE2* means both input files are NE tagged according to *MUC-7* style).

"*T*" – select this when you want term mapping. Both input files must be tagged with terms, please also specify

taggingInfo: *T0*, *T1* (*T0* means the foreign files are tagged with terms, *T1* means both input files are tagged with terms. Style of tags: *<TENAME>value</TENAME>*)

mappingFile: Path to the files (full paths) where the mapping information between the input files is given. Structure "*enFile***t***foreignFile***n**".

outputfile: path to the output file where results will be written.

foreignLang: language code of the foreign language e.g. el=greek

Please also make sure that when you run the mapper with the settings *NE0*, *NE1* and *T0* that you run the command from the folder where all the required resources are saved. These resources are the entire folders (*docs, data, workingFolderForTE and testdocs*) and are provided with the tool.

5.1.6 Input/Output data formats

Input to the mapper is text that is encoded in *UTF-8*.

For NE mapping the following input formats are needed:

1. English text (*UTF-8*) without any mark-up language tags such as *HTML* tags. The text has to be clean. It is not required to have formatting, such as, single sentence per line, etc.
2. Foreign text (*UTF-8*):
   a. Scenario one input: clean text with the settings as in English text.
   b. Scenario two input: clean text with the settings as in English text and NEs have to be marked-up according to the *MUC-7* style (e.g. *<ENAMEX TYPE="PERSON">Barack Obama</ENAMEX>*)

For TE mapping the following input formats are needed:

1. English text (UTF-8) without any mark-up language tags such as *HTML* tags. The text has to be clean. It is not required to have a formatting, such as, a single sentence per line, etc.

2. Foreign text (*UTF-8*): clean text with the settings as in English text and terms have to be marked-up according to the required style (e.g. *<TENAME>aspirin</TENAME>)*

Output of both the NE and term mapper is a list of NE/term pairs with scores. The scores indicate how strong the mapping is. A score of 1 means strong map and a score of 0 means no match.

5.1.7 Integration with external tools

As an external tool the *NE/term mapper* requires *OpenNLP*. *OpenNLP* can be downloaded from http://incubator.apache.org/opennlp/. For NE mapping (in case of scenario two) it requires a NER (Named Entity Recognizer) for the foreign language input text (ACCURAT related tools are described in D2.3). The term mapper requires the *KEA* tool to perform term identification for English text. For the foreign text the term extraction can be performed by the tools described in D2.3.

5.1.8 Contact

For further information and technical support installing and/or running this tool, please email to Ahmet Aker: a.aker@dcs.shef.ac.uk.

5.2 NERA2: Language Independent Named Entity Mapping

5.2.1 Overview and purpose of the tool

Named Entity Aligner (*NERA2*) tool is designed to map the named entities extracted from comparable or parallel documents. The algorithm is language independent and the application is intended to work for any pair of languages as long as a translation equivalents table exists for that pair of languages for occurrence forms. As input, the application needs the corresponding documents (comparable or parallel) with named entities marked according to *MUC-7* style (for a more detailed format description, refer to section 3.1.6.2 of the *TildeNER* system). *NERA2's* input is perfectly compatible with *NERA1* output (see section 3.3).

5.2.2 Changes from the previous version

Aside from bug fixing, there are no functional modifications and/or changes to the user's interface of this tool.

5.2.3 Software dependencies and system requirements

NERA2 is implemented in *C#* using *.NET Framework 4.0*. For machines using *Windows*, the users should install .Net Framework 4.0. For machines using *Linux*, the users should use *Mono 2.10* (http://www.mono-project.com/Main_Page). The machine should have at least 1GB RAM. In order to work for a pair of languages the application requires a translation equivalence table (e.g. a *GIZA++* translation lexicon) at the wordform level. A line in that table should be:

```
srcLang word <tab> trgLang word <tab> probability
```

and the name of the table should be: *srcLang_trgLang* (ex.: *en_ro*)

5.2.4 Installation

NERA2 does not require any installation other than the *.NET Framework*.

5.2.5 Execution instructions

```
NERA2.exe  --input  [FILE]  --output  [FILE]  [--source  [LANG]]  [--target
[LANG]] [--param aa=TRUE|FALSE]
```

where the command line switches have the following meanings:

"*--input FILE*" – Each line in the *input file* should contain the paths of two corresponding files (comparable or parallel) tab separated, having the Named Entities annotated in *MUC-7* style;

"*--output FILE*" – The output file contains named entities translation equivalents and their assigned probability scores, extracted from the input files.

"*--source [LANG]*" – source language. By default is: "*en*"

"*--target [LANG]*" – target language. By default is: "*ro*"

"*--param aa=TRUE|FALSE*" – optional parameter signalling the existence of additional *XML*-like mark-up in the input files.

5.2.6 Input/Output data formats

The input should be *UTF-8* text containing NE markups in the *MUC-7* style.

The output contains named entities translation equivalents and their assigned probability scores, extracted from the input files. Each line in the output file is of the form:

```
[source NE]<tab>[target NE]<tab>[SCORE]
```

5.2.7 Contact

For further information and technical support installing and/or running this tool, please email to Dan Ştefănescu: danstef@racai.ro.

5.3 A language independent terminology aligner

5.3.1 Overview and purpose of the tool

The *TerminologyAligner* tool is designed to map the terminological terms extracted from comparable or parallel documents. The algorithm is language independent and the application is intended to work for any pair of languages as long as a translation equivalents table exists for that pair of languages for occurrence forms. As input, the application needs the corresponding documents (comparable or parallel) with terminology marked according to *MUC-7* style. *Terminology Aligner's* input is perfectly compatible with *Terminology Extraction* output (see section 4.4).

5.3.2 Changes from the previous version

Aside from bug fixing, there are no functional modifications and/or changes to the user's interface of this tool.

5.3.3 Software dependencies and system requirements

TerminologyAligner is implemented in *C#* using *.Net Framework 4.0*. For machines using Windows, the users should install *.Net Framework 4.0*. For machines using Linux, the users should use *Mono 2.10*. The machine should have at least 1GB RAM. In order to work for a pair of languages the application requires a translation equivalence table (e.g. *GIZA++* translation lexicons) at the wordform level. A line in that table should be:

```
srcLang word <tab> trgLang word <tab> probability
```

and the name of the table should be: *srcLang_trgLang* (ex.: *en_ro*).

5.3.4 Installation

TerminologyAligner does not require any special installation steps other than those required by *.NET Framework*.

5.3.5 Execution instructions

The command line for TerminologyAligner is:

```
TerminologyAligner.exe --input [FILE] --output [FILE] [--source [LANG]] [--
target [LANG]] [--param aa=TRUE|FALSE]
```

where:

"*--input FILE*" – Each line in the *input file* should contain the paths of two corresponding files (comparable or parallel) tab separated, having the terminology annotated in *MUC-7* style;

"*--output FILE*" – The output file contains terminology translation equivalents and their assigned probability scores, extracted from the input files.

"*--source [LANG]*" – source language. By default is: "*en*"

"*--target [LANG]*" – target language. By default is: "*ro*"

"*--param aa=TRUE|FALSE*" – optional parameter signalling the existence of additional *XML*-like mark-up in the input files.

5.3.6 Input/Output data formats

The input should be *UTF-8* text containing terminology mark-ups in the *MUC-7* style.

The output contains terminological translation equivalents and their assigned probability scores, extracted from the input files. Each line in the output file is of the form:

```
[source term] <tab> [target term] <tab> [SCORE]
```

5.3.7 Contact

For further information and technical support installing and/or running this tool, please email to Dan Ştefănescu: danstef@racai.ro.

5.4 P2G: A tool to extract term candidates from aligned phrases

5.4.1 Overview and purpose of the tool

P2G (*PhraseTable2Glossary*) is a tool which extracts well-formed term candidates from phrase-aligned data, be it phrase tables or other outputs of phrase alignment (like *AnymAlign*, *PEXACC* etc.).

The principal approach is to apply a series of filters to the input candidate phrases, to output only the ones which can really be terms. Term candidates are brought into the right shape (lemmatisation, true-casing, gender and number agreement (in case of multi-words) etc.

- First, the tool creates a lattice of <*lemma, POS*> pairs for each word of the input candidate, using a lemmatiser (and decomposer for German).
- This lattice is then compared to a filter of (single and multiword) structures which lets only pass sequences having a “legal” term structure.
- This is done both for source and target candidate.
- In case of success, a proper term entry is created, by lemmatising the head of the term into singular form, by true-casing all its parts (capitalising nouns, uppercasing acronyms etc.), and by creating proper agreements between head nouns and modifying adjectives (using a noun gender defaulting mechanism).
- Finally, a filter can be applied to filter out term candidates which are already known (e.g. from a general-purpose lexicon, or stop words etc.), and only the rest is output [not in the current version].

Tests have shown that in the best case (using *MOSES*-aligned data) the overall error rate of the *P2G* tool is about 5% (2-3% each coming from errors in German or English pattern extraction or term creation); additional 6% errors result from incorrect phrase alignments by *MOSES*, so the overall error rate is about 11%. This is considered sufficient for human post-editing. Speed is about 100K phrase table entries per second, with about every 500th phrase table entry containing a well-formed term (in the automotive test, *P2G* created about 15.7 K terms from a 6.9 million phrase table, in 65 seconds).

5.4.2 Changes from the previous version

This is a new tool added to the second version of D2.6.

5.4.3 Software dependencies and system requirements

The system requires a preinstalled Java runtime environment. In order to perform term translation candidate extraction the user’s system has to be configured to use a heap size for java from 512MB up to at least 1024MB (for instance, execute the java command line with “*java -Xms512m -Xmx1024m*”).

5.4.4 Installation

The system comes in a zip file, which must be extracted. It contains two items: the "*jar*" file and a directory containing all linguistic resources required by the program. Both items can be placed where the users want; the only relevant information for later use is the "*datapath*", which indicates where the data directory is; this information must be given to the tool as a parameter at runtime later on.

5.4.5 Execution instructions

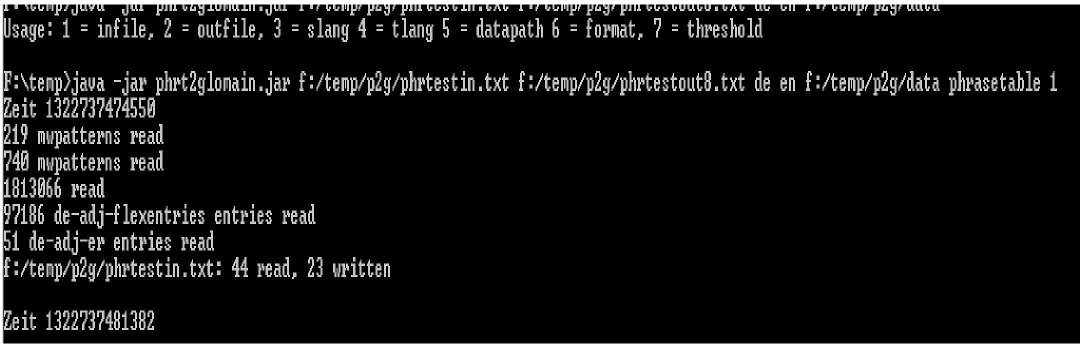
It is a command line call:

```
java -jar phrt2glomain.jar
```

The following parameters have to be passed to the command line in a strictly defined order:

- "***infile***" – the file containing the data to be processed, e.g. a phrase table. Infile should be encoded in *UTF-8* without *BOM*.
- "***outfile***" – the file containing the term candidates. Outfile will be encoded in *UTF-8* without *BOM*.
- "***source language***" – the ISO abbreviation of the source language. Legal values are: "*de*", "*en*".
- "***target language***" – the ISO abbreviation of the target language. Legal values are: "*de*", "*en*".
- "***datapath***" - the absolute pathname where the language resources are located.
- "***input format***" – the format of the input file. Legal values are: "*phrasetable*", "*anymalign*", „*pexacc*".
- "***threshold***" – the threshold is a frequency information in case of "*phrasetable*" or "*anymalign*" input, and a probability (like "*0.4*") in case of a "*pexacc*" input.

An example is given below:



```
F:\temp>java -jar phrt2glomain.jar f:/temp/p2g/phrtestin.txt f:/temp/p2g/phrtestout8.txt de en f:/temp/p2g/data
Usage: 1 = infile, 2 = outfile, 3 = slang 4 = tlang 5 = datapath 6 = format, 7 = threshold

F:\temp>java -jar phrt2glomain.jar f:/temp/p2g/phrtestin.txt f:/temp/p2g/phrtestout8.txt de en f:/temp/p2g/data phrasetable 1
Zeit 1322737474550
219 mwpatterns read
740 mwpatterns read
1813066 read
97186 de-adj-flexentries entries read
51 de-adj-er entries read
f:/temp/p2g/phrtestin.txt: 44 read, 23 written

Zeit 1322737481382
```

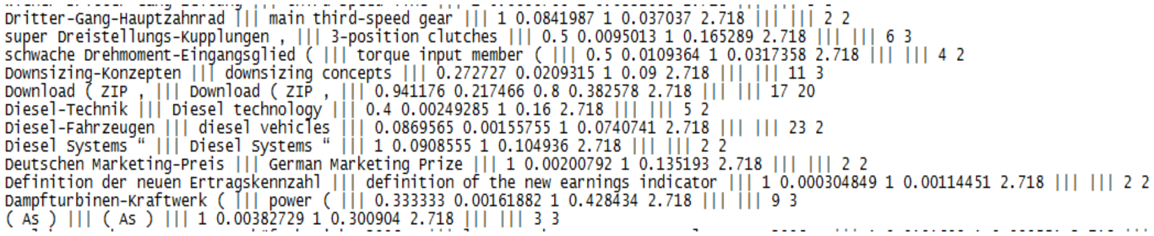
5.4.6 Input / output data formats

5.4.6.1 Input formats

Three formats are supported: "*phrasetable*", "*anymalign*", and "*pexacc*".

5.4.6.1.1 PhraseTable Format

This format is produced by standard SMT using the *MOSES* toolkit. It looks as follows:

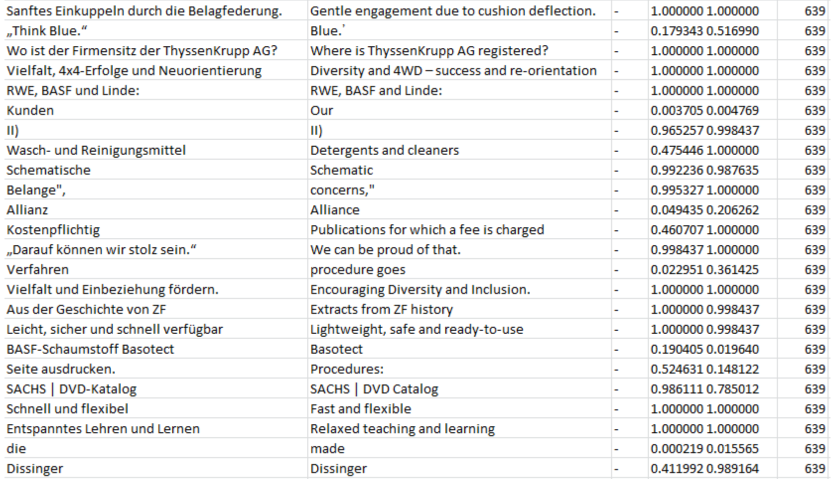


```
Dritter-Gang-Hauptzahnrad ||| main third-speed gear ||| 1 0.0841987 1 0.037037 2.718 ||| ||| 2 2
super Dreistellungs-Kupplungen , ||| 3-position clutches ||| 0.5 0.0095013 1 0.165289 2.718 ||| ||| 6 3
schwache Drehmoment-Eingangsglied ( ||| torque input member ( ||| 0.5 0.0109364 1 0.0317358 2.718 ||| ||| 4 2
Downsizing-Konzepten ||| downsizing concepts ||| 0.272727 0.0209315 1 0.09 2.718 ||| ||| 11 3
Download ( ZIP , ||| Download ( ZIP , ||| 0.941176 0.217466 0.8 0.382578 2.718 ||| ||| 17 20
Diesel-Technik ||| Diesel technology ||| 0.4 0.00249285 1 0.16 2.718 ||| ||| 5 2
Diesel-Fahrzeugen ||| diesel vehicles ||| 0.0869565 0.00155755 1 0.0740741 2.718 ||| ||| 23 2
Diesel Systems “ ||| Diesel Systems “ ||| 1 0.0908555 1 0.104936 2.718 ||| ||| 2 2
Deutschen Marketing-Preis ||| German Marketing Prize ||| 1 0.00200792 1 0.135193 2.718 ||| ||| 2 2
Definition der neuen Ertragskennzahl ||| definition of the new earnings indicator ||| 1 0.000304849 1 0.00114451 2.718 ||| ||| 2 2
Dampfturbinen-Kraftwerk ( ||| power ( ||| 0.333333 0.00161882 1 0.428434 2.718 ||| ||| 9 3
( As ) ||| ( As ) ||| 1 0.00382729 1 0.300904 2.718 ||| ||| 3 3
```

Each record consists of 5 parts, separated by three pipes (|||); relevant sections are source and target candidate (col. 1 and 2), translation probability (3) and frequency (5).

5.4.6.1.2 AnymAlign Format

This format consists of records; each record has 5 fields, separated by a *TAB*:

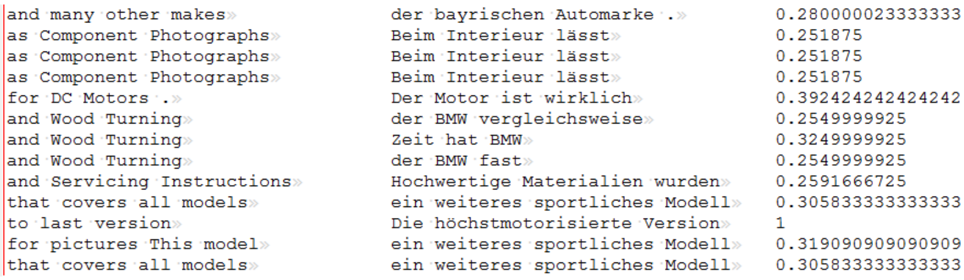


Sanftes Einkuppeln durch die Belagfederung.	Gentle engagement due to cushion deflection.	-	1.000000 1.000000	639
„Think Blue.“	Blue.’	-	0.179343 0.516990	639
Wo ist der Firmensitz der ThyssenKrupp AG?	Where is ThyssenKrupp AG registered?	-	1.000000 1.000000	639
Vielfalt, 4x4-Erfolge und Neuorientierung	Diversity and 4WD – success and re-orientation	-	1.000000 1.000000	639
RWE, BASF und Linde:	RWE, BASF and Linde:	-	1.000000 1.000000	639
Kunden	Our	-	0.003705 0.004769	639
II)	II)	-	0.965257 0.998437	639
Wasch- und Reinigungsmittel	Detergents and cleaners	-	0.475446 1.000000	639
Schematische	Schematic	-	0.992236 0.987635	639
Belange",	concerns,"	-	0.995327 1.000000	639
Allianz	Alliance	-	0.049435 0.206262	639
Kostenpflichtig	Publications for which a fee is charged	-	0.460707 1.000000	639
„Darauf können wir stolz sein.“	We can be proud of that.	-	0.998437 1.000000	639
Verfahren	procedure goes	-	0.022951 0.361425	639
Vielfalt und Einbeziehung fördern.	Encouraging Diversity and Inclusion.	-	1.000000 1.000000	639
Aus der Geschichte von ZF	Extracts from ZF history	-	1.000000 0.998437	639
Leicht, sicher und schnell verfügbar	Lightweight, safe and ready-to-use	-	1.000000 0.998437	639
BASF-Schaumstoff Basotect	Basotect	-	0.190405 0.019640	639
Seite ausdrucken.	Procedures:	-	0.524631 0.148122	639
SACHS | DVD-Katalog	SACHS | DVD Catalog	-	0.986111 0.785012	639
Schnell und flexibel	Fast and flexible	-	1.000000 1.000000	639
Entspanntes Lehren und Lernen	Relaxed teaching and learning	-	1.000000 1.000000	639
die	made	-	0.000219 0.015565	639
Dissinger	Dissinger	-	0.411992 0.989164	639

Relevant fields are source and target candidate (col. 1 and 2) and frequency (col. 5).

5.4.6.1.3 PEXACC format

This format consists of records containing three elements: source, target, and probability. While the original *PEXACC* output provides this information in separate lines (cf. figure), the *P2G* tool expects it in one single line; so a little converter must be used to bring it into the format below:

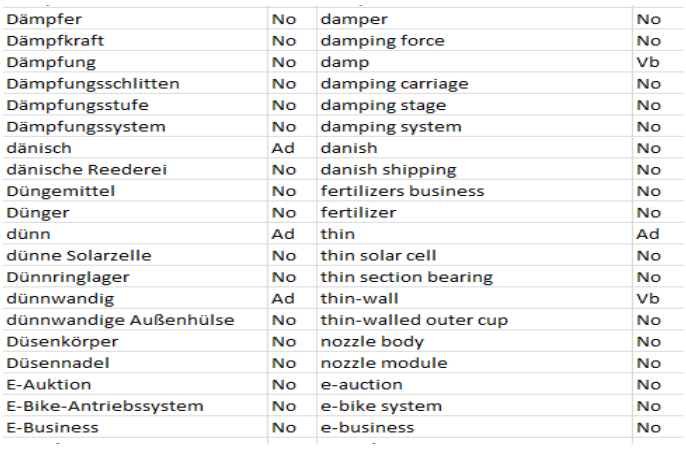


```
and many other makes»             der bayrischen Automarke .»        0.280000023333333
as Component Photographs»         Beim Interieur lässt»              0.251875
as Component Photographs»         Beim Interieur lässt»              0.251875
as Component Photographs»         Beim Interieur lässt»              0.251875
for DC Motors .»                  Der Motor ist wirklich»            0.392424242424242
and Wood Turning»                 der BMW vergleichsweise»           0.2549999925
and Wood Turning»                 Zeit hat BMW»                      0.3249999925
and Wood Turning»                 der BMW fast»                      0.2549999925
and Servicing Instructions»       Hochwertige Materialien wurden»    0.2591666725
that covers all models»           ein weiteres sportliches Modell»   0.305833333333333
to last version»                  Die höchstmotorisierte Version»    1
for pictures This model»          ein weiteres sportliches Modell»   0.319090909090909
that covers all models»           ein weiteres sportliches Modell»   0.305833333333333
```

5.4.6.2 Output format

The output contains records with term candidates. Each record contains four fields, separated by *TABs*:

- Source term and its part of speech (*No* - noun, *Vb* - verb, *Ad* - adjective)
- Target term and its part of speech



Dämpfer	No	damper	No
Dämpfkraft	No	damping force	No
Dämpfung	No	damp	Vb
Dämpfungsschlitten	No	damping carriage	No
Dämpfungsstufe	No	damping stage	No
Dämpfungssystem	No	damping system	No
dänisch	Ad	danish	No
dänische Reederei	No	danish shipping	No
Düngemittel	No	fertilizers business	No
Dünger	No	fertilizer	No
dünn	Ad	thin	Ad
dünne Solarzelle	No	thin solar cell	No
Dünnringlager	No	thin section bearing	No
dünnwandig	Ad	thin-wall	Vb
dünnwandige Außenhülse	No	thin-walled outer cup	No
Düsenkörper	No	nozzle body	No
Düsennadel	No	nozzle module	No
E-Auktion	No	e-auction	No
E-Bike-Antriebssystem	No	e-bike system	No
E-Business	No	e-business	No

5.4.7 Integration with external tools

P2G is a stand-alone tool, with no integration attempted or required.

5.4.8 Contact

For further information and technical support installing and/or running this tool, please email to Gr. Thurmair, Linguatec (g.thurmair@linguatec.de).

5.4.9 Useful references

There is a short description on how the tool works in the ACCURAT deliverable D2.5.

6 Other useful tools

This section covers the tools that are useful for additional tasks, such as, document translation, dictionary creation and have been developed within the ACCURAT project in order to support the tools described in previous sections.

The tools included in this section of the ACCURAT toolkit are:

- *A toolkit for text translation using Google translation API or Microsoft translation API* (developed by CTS; see section 6.1).
- *DEACC: lexical dictionary extractor from comparable corpora* (developed by RACAI; see section 6.2).

6.1 A toolkit for text translation using Google translation API or Microsoft translation API

6.1.1 Overview and purpose of the tool

This *Java* toolkit allows users to translate text collections from a source language to a target language by using the available *Google translation Java API* or *Microsoft Bing translation Java API*. Currently, *Google translation API* supports 63 languages and *Bing Translation API* supports 36 languages.

6.1.2 Changes from the previous version

There are no changes from the previous version.

6.1.3 Software dependencies and system requirements

(1) **System:** platform independent (*Java* program).

(2) **Internet:** Given that both *Google* and *Bing translation API* require requests to be sent to remote servers for translation, the system should ensure that internet connectivity is available.

(3) **JRE:** *1.6.0* (not specified, lower versions should also work OK).

6.1.4 Installation

No installation other than the *JRE* engine is required. For the latter, just follow the instructions of the wizard.

6.1.5 Execution instructions

Line length limits: The length of each line in a document should not exceed 5000 characters for *Google translation API*, and 5000 Bytes for *Microsoft translation API*. So in the text collection, if there is line with length over this limit, the user should split into several shorter lines.

Google translation API and *Bing translation API* have different length limits for each translation request. For *Google translation API*, the text String length limit of each call is 5000 character for all the supported languages.

For *Bing translation API*, the length limit is 10,000 Bytes for most of the Western languages (if in that language encoding, 1 character takes 1 byte). However, for Greek, Chinese and Russian (and maybe some other languages), the length limit is around 5000 bytes. This is because in Greek and Russian a character takes 2 bytes while in Chinese a character takes 3 bytes. This conversion is the same under the UTF-8 encoding.

Therefore, in our toolkit we set the length limit of a translation request at 5000 characters for Google translation API and 5000 bytes for Microsoft translation API.

Also, as we prepare the text by line, we can send each line as text string for translation. However, for a large collection of documents, this will require too many calls and can quickly reach the translation access limit (as both *Google* and *Microsoft* APIs will report errors or exceptions if there are too many translation requests within a relatively short time from the same IP address). Therefore, in order to reduce the number of translation requests, it is better to send longer strings for translation (i.e. around 5000 characters). This is because, as long as input string length does not exceed the length limit, the translation access limit of both *Google* and *Microsoft* APIs are based on the number of translation requests, but not the length of overall input string length.

The toolkit supports two different manners of translation. For each translation call, you can send either a text string, or a string array for translation. Technically, the calls are:

Manner 1:

```
String result =
Translate.execute(String text, SourceLanguage, TargetLanguage)
```

Manner 2:

```
String[] result =
Translate.execute(String[] text, SourceLanguage, TargetLanguage)
```

Command line usage:

```
java   -jar   Translation.jar   option=1|2   SourceLanguage   TargetLanguage
SourcePath TargetPath
```

Translation.jar: *GoogleTranslate.jar* or *BingTranslate.jar*

Options:

option=1: merge several lines into a long string for translation (Manner 1);

option=2: store lines as a string array for translation (Manner 2);

SourceLanguage: one of the languages supported by the specific engine (see below);

TargetLanguage: one of the languages supported by the specific engine (see below);

SourcePath: the path to the text collection to be translated (absolute path to the input data directory);

TargetPath: the destination directory where to store translations (directory path – the directory has to exist, otherwise the API will return an exception).

Note that the parameters are separated by space, and both the source language and the target language parameters should be uppercased. The supported language list is as follows:

Google translation API:

```
AUTO_DETECT AFRIKAANS ALBANIAN AMHARIC ARABIC ARMENIAN AZERBAIJANI BASQUE
BELARUSIAN BENGALI BIHARI BULGARIAN BURMESE CATALAN CHEROKEE CHINESE
CHINESE_SIMPLIFIED CHINESE_TRADITIONAL CROATIAN CZECH DANISH DHIVEHI DUTCH
ENGLISH ESPERANTO ESTONIAN FILIPINO FINNISH FRENCH GALICIAN GEORGIAN GERMAN
GREEK GUARANI GUJARATI HEBREW HINDI HUNGARIAN ICELANDIC INDONESIAN
INUKTITUT IRISH ITALIAN JAPANESE KANNADA KAZAKH KHMER KOREAN KURDISH KYRGYZ
LAOTHIAN LATVIAN LITHUANIAN MACEDONIAN MALAY MALAYALAM MALTESE MARATHI
MONGOLIAN NEPALI NORWEGIAN ORIYA PASHTO PERSIAN POLISH PORTUGUESE PUNJABI
ROMANIAN RUSSIAN SANSKRIT SERBIAN SINDHI SINHALESE SLOVAK SLOVENIAN SPANISH
SWAHILI SWEDISH TAJIK TAMIL TAGALOG TELUGU THAI TIBETAN TURKISH UKRANIAN
URDU UZBEK UIGHUR VIETNAMESE WELSH YIDDISH
```

Microsoft translation API:

```
AUTO_DETECT ARABIC BULGARIAN CHINESE_SIMPLIFIED CHINESE_TRADITIONAL CZECH
DANISH DUTCH ENGLISH ESTONIAN FINNISH FRENCH GERMAN GREEK HATIAN_CREOLE
HEBREW HUNGARIAN INDONESIAN ITALIAN JAPANESE KOREAN LATVIAN LITHUANIAN
NORWEGIAN POLISH PORTUGUESE ROMANIAN RUSSIAN SLOVAK SLOVENIAN SPANISH
SWEDISH THAI TURKISH UKRANIAN VIETNAMESE
```

Run examples:

Linux command:

```
java -jar GoogleTranslate.jar option=1 LATVIAN ENGLISH
/home/fzsu/TranslationToolkit/sample/LV /home/fzsu/TranslationToolkit/LV-
translation
```

Windows command:

```
java -jar GoogleTranslate.jar option=1 LATVIAN ENGLISH
C:\TranslationToolkit\sample\LV C:\TranslationToolkit\LV-translation
```

The above command will translate the *Latvian* documents in the source path "*/home/fzsu/TranslationToolkit/sample/LV*" (*Linux* platform) or "*C:\TranslationToolkit\sample\LV*" (*Windows* platform) into English, and then save them in the target path "*/home/fzsu/TranslationToolkit/LV-translation*" or (*C:\TranslationToolkit\LV-translation*). A folder called "*translation*" in Directory "*LV-translation*" will store all the translated documents.

6.1.6 Input/Output data formats

Both input files and output files are plain *UTF-8* text files.

6.1.7 Integration with external tools

None noted other than the *JRE* engine.

6.1.8 Contact

For further information and technical support installing and/or running this tool, please email to Fangzhong Su: smlfs@leeds.ac.uk.

6.1.9 Useful references

Google Translation API: http://code.google.com/apis/language/translate/overview.html

Microsoft Translation API: http://www.microsofttranslator.com/tools/

6.2 DEACC: lexical dictionary extractor from comparable corpora

6.2.1 Overview and purpose of the tool

The purpose of the tool is to extract bilingual lexical dictionaries (word-to-word) from comparable corpora. The corpus does not have to be aligned at any level (document, paragraph, etc.)

The method implemented in this tool is introduced by (Rapp, 1999). The application basically counts word co-occurrences between unknown words in the comparable corpora and known words from a *Moses* extracted general domain translation table (which from now on will be referred to as the base lexicon). We adapted the algorithm to work with polysemous entries in the translation table (very frequent situation which is not treated in the standard approach).

As the purpose of this tool (and of all the other tools in the project) is to extract from comparable corpora data that would enrich the information already available from parallel corpora, it seems reasonable to focus on the open class (versus closed class) words. Obviously, this approach reduces the space and time necessities. Moreover, the closed class words we decided to ignore (pronouns, prepositions, conjunctions, articles, auxiliary verbs) do not behave according to any semantic pattern (they are too vague); therefore, they are not useful in an approach that is based on the tendency of some words to occur in the same semantic context as other words. Because in many languages, the auxiliary verbs can also be main verbs, frequently basic concepts in the language (see "*be*" or "*have*" in English), and most often the POS-taggers don't discriminate correctly between the two roles, we decided to eliminate their main verb occurrences also. For this purpose, the user is asked to provide a list of all these types with all their forms in the language of interest other than English.

6.2.2 Changes from the previous version

There are no changes from the previous version.

6.2.2.1 A short description of the algorithm

Firstly on the corpus of the source language and secondly on the corpus of the target language, a co-occurrence matrix is computed, whose rows are all word types occurring in the corpus and whose columns are words in that corpus appearing in the base lexicon. Initially, the co-occurrence matrix contains the co-occurrence frequencies.

The next step is to replace all these frequencies with the log-likelihood scores. In the end, a similarity computation is done between all the vectors in the source matrix and all the vectors in the target matrix.

For a specific source vector, the first ten target vectors with the highest similarities are considered to be the possible translations of the corresponding source-language word. The similarity score can be used in a Moses type decoder to select the most probable translation of the word in a specific context.

The measure used to compute the similarity score is *DiceMin*. Please refer to (Gamallo, 2008) for a discussion about the efficiency of several similarity metrics combined with two weighting schemes: simple occurrences and log likelihood.

6.2.3 Software dependencies and system requirements

The aligner is implemented in the programming language *C#*, under the *.NET Framework 2.0*. It requires the following settings to run:

1. *.NET Framework 2.0*.
2. 2+ GB RAM (4 GB preferred)
3. On a multi-processing system, the computing of the similarity score can be divided on different processors at the user's request, by a parameter in the configuration file of the application.

6.2.4 Installation

The application does not require any installation aside that of *.NET Framework* which is publicly available.

6.2.5 Execution instructions

Given that the user machine has *.NET Framework 2.0* installed; the application can be run as an executable file both under *Windows* and *Linux* platforms.

The "*.exe*" file must be placed in a working folder, containing two subfolders: "*source corpus*" and "*target corpus*" and two other files: the "*base lexicon*" and a configuration file named: "*cooc.cfg*"

The "*source corpus*" and "*target corpus*" folders will contain one or more documents, named after the rule: "**_corpus.txt*". The text in the documents should be in the format:

```
word_form1|lemma1|POS1 word_form2|lemma2|POS2 …
```

The base lexicon is in the format:

```
source_word_form|target_word_form
```

The "*cooc.cfg*" file, reproduced below, is self-explanatory:

```
//1. if the user's machine has multiple processors, the application
//   can apply a function that splits the time consuming problem of
//   computing the vector similarities and runs it in parallel.
*multithreading:yes|no (default=no)
```

```
//2. to avoid overloading the memory, the application gives the user
//   the opportunity to decide how many of the source/target vectors
//   are loaded in the memory at a specific moment; it avoids
//   overloading the memory but can bring an important time delay;
//   this parameter is activated only for "multithreading:yes"
//   default value: 0; if the parameter's value is bigger than the
//   number of vectors in the matrix, its use becomes obsolete.
*loading:int (default=0)

//3. the minimal frequency in the corpus of the words the user wants
//  to find  translation equivalents for: being based on word
//  counts, the method is sensitive to the frequency of the words.
//  The bigger frequency, the better performance. This parameter
//  should be at least bigger than 3 and should take into account
//  the corpus size.
*frequency:int (default=3)

//4. the user can specify the length of the text window in
//   which co-occurrences are counted.
*window:int (default=5)

//5. asking for the log-likelihood of a co-occurrence to be bigger than
//   a certain threshold, the user can reduce the space and time costs
*ll:int (default=3)

//6. the user is asked to introduce a list of all the auxiliary/modal
//   verbs for the source language, with all their
//   morphological variants, separated by white space.
*sourceamverblist:string (default=is are be will shall may can)

//7. the user is asked to introduce a list of all the auxiliary/modal
//   verbs for the target language, with all their morphological variants,
//   separated by white space.
//   Default value is set for Romanian.
*targetamverblist:string  (default=este  sunt  suntem  sunteţi  fi  poate  pot
putem puteţi)

//8. the user can decide if he/she allows to the application to cross
//   the boundaries between the parts of speech (i.e. to translate a
//   noun as a verb).
*crossPOS:yes|no (default:no)

//9. the user has to provide a list of all the open class POS labels
```

```
//   (i.e. labels for common nouns, proper nouns, adjective, adverbs
//   and main verbs) of the source language.
//   Default value uses MSDs first two letters.
*sPOSlist:string (default=nc np a r vm)

//10. the user has to provide a list of all the open class POS labels
//   (i.e. labels for common nouns, proper nouns, adjective, adverbs
//   and main verbs) of the target language.
//   Default value uses MSDs first two letters.
*tPOSlist:string (default=nc np a r vm)

//11. the user can decide if a cognate score (Levenshtein distance) will
//    be taken into account in computing the vector similarities for
//    proper nouns matching.
*LD:yes|no (default=yes)

//   Working settings for all parameters above for EN-RO processing:
multithreading:yes
loading:5000
frequency:10
window:5
ll:3
sourceamverblist:am is are was were been beeing had has have be will would
shall should may might must can could need
targetamverblist:este sunt suntem va voi vor vom fi pot putea puteam
puteaţi
crossPOS:no
sPOSlist:nc np a r vm
tPOSlist:nc np a r vm
LD:yes
```

6.2.6 Input/Output data formats

6.2.6.1 *Input data formats*

The "*source corpus*" and "*target corpus*" folders will contain one or more *UTF-8* documents, named after the rule: "**_corpus.txt*". The text in the documents should be in the format:

```
"word_form1|lemma1|POS1 word_form2|lemma2|POS2 word_form3|lemma3|POS3 …"
…
```

The base lexicon is in the format:

```
source_word_form|target_word_form<new line>
…
```

The base lexicon and configuration file must also be *UTF-8* encoded.

6.2.6.2 *Output data format*

The program outputs a *UTF-8* dictionary in the format:

```
<source_word^POS>|<target_candidate1^POS><score>#<target_candidate2^POS><sc
ore>…#<target_candidate10^POS> <score><new line>
…
```

6.2.7 Integration with external tools

The application does not need any external tool.

6.2.8 Contact

For further information and technical support installing and/or running this tool, please email to Elena Irimia: elena@racai.ro.

6.2.9 Useful references

Gamallo, P. 2008 *Evaluating two different methods for the task of extracting bilingual lexicons from comparable corpora*. In Proceedings of LREC 2008 Workshop on Comparable Corpora, Marrakech, Morocco, pp. 19-26. ISBN: 2-9517408-4-0.

Rapp, R. 1999. *Automatic Identification of Word Translations from Unrelated English and German Corpora*. In Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL'99), pages 519-526, college Park, Maryland, USA.

7 Conclusions

This document contains the technical descriptions of the parallel data extraction and alignment tools that have been developed within the ACCURAT project at the time of writing (for the second half of tools that deal with comparable corpora acquisition from web refer to the Deliverable D3.5 of the ACCURAT project). Most of them are included in predefined workflows that are ready for immediate use:

- parallel textual unit (sentences and/or phrases) extraction from comparable corpora (see section 1.1)
- NE/Term mapping from comparable corpora (see section 1.2).

The toolkit also contains an application that extracts translation lexicons from comparable corpora (*DEACC*, section 6.2) and also tools that translate text using *Google* and/or *Microsoft* provided APIs.

The documentation is intended to guide the (computer knowledgeable) user in installing and running the tools. By using them, the users may expect to obtain parallel texts, parallel terminology, general translation lexicons and translated named entities, all of which are useful as training data/resources for either SMT or Example-bases/Rule-based MT.

As some new methods were proposed by partners recently (for instance, for parallel data extraction and terminology mapping), we could release a new version of the toolkit by the end of the project. We will gradually go from the individual tool evaluation to evaluation of MT improvement when using resources generated by these tools. Thus, we aim at measuring the performance of any of these tools by the degree of improvement (or degradation) of MT accuracy.