Deliverable D3.2.1

Definition of Interfaces

Author(s): Hala Almaghout, Ergun Biçici, Stephen Doherty, Federico Gaspari, Declan Groves, Antonio Toral, Josef van Genabith (DCU), Maja Popović (DFKI), Stelios Piperidis (ATHENA RC)

Dissemination Level: Public

Date: 10.11.2014

Grant agreement no.	296347
Project acronym	QTLaunchPad
Project full title	Preparation and Launch of a Large-scale Action for Quality Translation Technology
Funding scheme	Coordination and Support Action
Coordinator	Prof. Hans Uszkoreit (DFKI)
Start date, duration	1 July 2012, 24 months
Distribution	Public
Contractual date of delivery	February 2013
Actual date of delivery	March 12, 2013 [updated November 2013, November 2014]
Deliverable number	D3.2.1
Deliverable title	Definition of Interfaces
Type	Report
Status and version	v1.2_10112014.
Number of pages	66
Contributing partners	DCU, DFKI, ILSP
WP leader	DFKI
Task leader	DCU
Authors	Hala Almaghout, Ergun Biçici, Stephen Doherty, Federico Gaspari, Declan Groves, Antonio Toral, Josef van Genabith, Maja Popović, Stelios Piperidis
EC project officer	Kimmo Rossi
The partners in QTLaunchPad are:	Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany
	Dublin City University (DCU), Ireland
	Institute for Language and Speech Processing, R.C. “Athena” (ILSP/ATHENA RC), Greece
	The University of Sheffield (USFD), United Kingdom

For copies of reports, updates on project activities and other QTLaunchPad-related information, contact:

DFKI GmbH
QTLaunchPad
Dr. Aljoscha Burchardt — aljoscha.burchardt@dfki.de
Alt-Moabit 91c — Phone: +49 (30) 23895-1838
10559 Berlin, Germany — Fax: +49 (30) 23895-1810

Copies of reports and other material can also be accessed via http://www.qt21.eu/launchpad

Table of Contents

1. Executive Summary

The aim of this report is to define the interfaces for the tools used in the MT development and evaluation scenarios as included in the QTLaunchPad (QTLP) infrastructure. Specification of the interfaces is important for the interaction and interoperability of the tools in the developed QTLP infrastructure. In addressing this aim, the report provides:

1. Descriptions of the common aspects of the tools and their standardized data formats;
2. Descriptions of the interfaces for the tools for interoperability.

where the tools are categorized into preparation, development, and evaluation categories including the human interfaces for quality assessment with multidimensional quality metrics. Interface specifications allow a modular tool infrastructure, flexibly selecting among alternative implementations, enabling realistic expectations to be made at different sections of the QTLP information flow pipeline, and supporting the QTLP infrastructure.

D3.2.1 allows the emergence of the QTLP infrastructure and helps the identification and acquisition of existing tools (D4.4.1), the integration of identified language processing tools (D3.3.1), their implementation (D3.4.1), and their testing (D3.5.1). QTLP infrastructure facilitates the organization and running of the quality translation shared task (D5.2.1) related activities. We also provide human interfaces for translation quality assessment with the multidimensional quality metrics (D1.1.1). D3.2.1 is a living document until M12, which is when the identification and acquisition of existing tools (D4.4.1) and the implementation of identified language processing tools (D3.4.1) are due.

2. Introduction

The EU project QTLaunchPad (Preparation and Launch of a Large-Scale Action for Quality Translation Technology)[1] aims to identify and cross quality barriers in MT translation output by producing better quality assessment, evaluation, and estimation techniques, which can be made possible with efficient and clear integration of the requirements for MT development and evaluation processes. We envision the existence of quality barriers separating translations of high quality from average quality and average quality from low quality (Figure 1). If we can identify the translations that lie at the lower end of these barriers, we can spend more resources to move them to the higher quality translation regions. This type of targeted post-processing towards specific translations can increase the performance of MT output significantly. Quality estimation allows for the automatic identification of these quality barriers without using reference translations and will help us expend our efforts in a targeted manner.

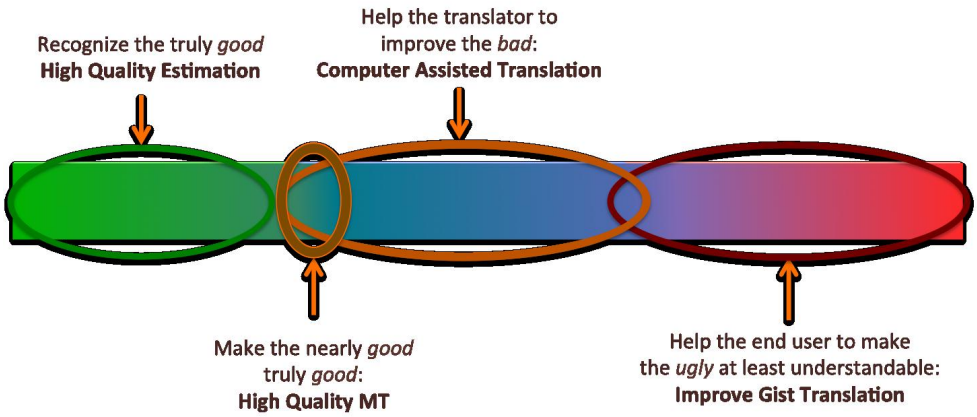


Figure 1: Quality Barriers of MT

The aim of Task 3.2 is to define and document the interfaces for the successful integration of the tools required for the QTLP infrastructure. *Interface* is the specification of the common aspects of the tools and their standardized data formats for allowing interoperability. Task 3.1 (D3.1.1) identifies the common workflows for the development and evaluation of MT systems and their desired feature specifications together with an initial set of language processing tools.

The report is structured into two parts, the first part details the tools required for the current

[1] http://www.qt21.eu

workflows and is divided into three tool categories: preparation, development, and evaluation. We detail the data format for the inputs and outputs of each tool and concentrate on the common aspects of the tools and their standardized data formats. Moving from a summary of the main findings, the second part of the report defines the interfaces necessary for the flow of information between tools and applications as described by the workflows given in D3.1.1, again categorized according to the tool categories. We note here that this deliverable provides information and descriptions for the typical and most prevalent workflows in machine translation and the related areas that are relevant to the remit of the QTLaunchPad project and future directions of QT21. Where appropriate external information is listed to provide additional information to other workflows and related descrptions that can be consulted for further reading.

2.1. Interface Representation

This section defines our representation of the tool interfaces and a set of attributes for specifying the description of tools. The attributes, if used, are either optional or required for the specification and they can have a range and a default value. A tool interface defines the inputs, the outputs, and the attributes of the tool together with the attributes of the inputs and the outputs.

A tool specification is composed of the tool name, a set of formats for the inputs and the outputs, and the tool confidence score:

[tname : tcs : <input formats> : <output formats>]

If the tool has *n* inputs and *m* outputs, input and output formats can be defined as follows, providing the specifications for all the inputs and the outputs of the tool:

<input formats> := input1F; input2F; …; input*n*F
<output formats> := output1F; output2F; …; output*m*F

Individual input and output format (input*i*F and output*i*F) are a 4-tuple:

(name, cs, type, lang)

An example for each input and output should be provided for any tool implementing the interface.

The attributes are defined as follows:

- tname: Tool name, required; string or URI.
- tcs: Tool confidence score, required; represents the performance of the tool in its expertise domain as specified in its own documentation or in a publication of its authors. The score is domain and possibly language dependent.
- cs: Confidence score, optional; specifies a real number in the interval [0, 1], which represents the quality of the input or the output. (default 1.0)
- name: Required; string or URI
- type: Required; possible types: text file, speech file, grammar file, XML file, ARPA file, ...
- lang: Language, required; the language of the resource.

Notes:

- If a text file is part of a parallel text file, then we add a link between the two texts in the interface diagram.
- The specifications given in section 3 use an identifier instead of tname for tools or name for inputs/outputs for clear presentation. In section 4, we differentiate these and use the identifiers to refer to the objects.
- An implementation of the interfaces should provide examples for all the inputs and the outputs. The specifications for the implementations should also contain information about the paths for the components together with example usage.

3. Description of Tools

We provide a set of attributes for the abstraction of the description of the tools used in QTLP workflows. One of our goals is to unearth the commonalities found in different tool categories. Related work includes the W3C Internationalization Tag Set (ITS) (ITS 2.0[2]), which provides standards for data categories and their implementation as a set of elements and their attributes to promote the creation of multilingual Web content and localization workflows based on the XML Localization Interchange File Format (XLIFF). The EU project META-NET[3] provides a META-SHARE metadata model for describing language resources including data processing tools. The EU project PANACEA[4] provides common interfaces for web service providers with a goal of creating a language resources factory wrapping each resource as an independent web service. In contrast to previous work, our goal is not to categorize and describe language resources. We also do not specify tool dependent details that cannot be easily abstracted (i.e. intricacies which may be language or system dependent such as running time) and we do not focus on graphical user interfaces.

We categorize the tools into these main categories:

- preparation tools
- development tools
- evaluation tools

We follow the interface representation (section 2.1) when describing the tools. Each tool category contains a description, the input and the output descriptions, a common tool specification, and examples.

[2] http://www.w3.org/TR/2012/WD-its20-20121206/
[3] http://www.meta-net.eu
[4] http://www.panacea-lr.eu

3.1. Preparation Tools

The tools in this category are used for preparing for the development or evaluation of MT systems. Preparation tools can be used for acquiring monolingual or bilingual corpora and they homogenize the parallel corpora and the monolingual language model corpora that are used during the development of MT systems. Corpora preparation involves corpus filtering, which focuses on homogenizing a given corpus according to a set of criteria (i.e. in terms of the lengths of the sentences it contains), tokenization, and casing mainly to reduce the number of out-of-vocabulary (OOV) items during translation. Extra linguistic information may also be used for instance by morphological analysis of the tokens, which can further reduce the OOV rate.

3.1.1. Focused Crawler

Description:

The Focused Crawler (FC) is a tool for acquiring domain-specific monolingual and bilingual corpora. The FC integrates modules for text normalization, language identification, document clean-up, text classification, links ranking, (near) de-duplication and identification of documents that are translations of each other. The required input for the FC consists of a list of seed URLs pointing to domain-relevant web pages. The FC is initialized with the seed URLs and extracts their links. Following the most promising links, the FC visits new pages and continues crawling the web until crawl time expires. Depending on the user's goal (i.e. construction of monolingual or bilingual corpora), an FC tool follows slightly different workflows (i.e. input, crawling strategy, processing steps and output are differentiated).

Input:

A text file that contains a list of seed URLs.

In the case of bilingual crawling, it is proposed that the seed URLs point to multilingual sites since it is very likely that such web sites contain high quality translations.

Output:

A text file that contains lists of URLs pointing to stored versions of domain-relevant crawled pages.

In the case of monolingual crawling, the output contains a list of corpus URLs pointing to XML files (each XML file corresponds to a downloaded web page during the crawling stage and provides the main content of the web page and its metadata).

In the case of bilingual crawling, the output contains a list of corpus URLs pointing to XML files (each XML file corresponds to a pair of parallel documents and contains links to these documents).

Common tool specification:

[focused_crawler: tcs : <(seedURLs, cs, text file, lang)> : <(corpusURLs, cs, text file, lang)>]

Examples:

1) Running a monolingual crawl with a focused crawler like the ILSP Focused Crawler (ilsp-fc) from http://nlp.ilsp.gr/redmine/projects/ilsp-fc :

```
java -jar ilsp-fc.jar \
      -u seedURLs.txt -of corpusURLs.txt \
      -cfg config.xml \
      –lang en -type m
```

2) Running a bilingual crawl with a focused crawler like the ILSP Focused Crawler (ilsp-fc) from http://nlp.ilsp.gr/redmine/projects/ilsp-fc :

```
java -jar ilsp-fc.jar \
      -u seedURLs.txt -of corpusURLs.txt \
      -cfg config.xml \
      –lang en,de -type p
```

Additionally, as in the above examples, tools can use a configuration file that can set several settings influencing a crawling session (e.g. type of crawl: (monolingual | parallel), the number of threads that will be used to fetch web pages in parallel, connection timeout, accepted mime types, etc.). Finally, the settings of the configuration file can be overridden via the command line, as in the case of the lang and type parameters in the above

examples.

3.1.2. Sentence Splitter

Description:

The sentence splitting step takes a document and splits its sentences so that each sentence occupies a separate line. The resulting split sentences are ordered according to their location in the original document. The input is a text file and the output is the text file that contains a sentence in each line.

Input:

A text file in language L1.

Output:

A text file in language L1.

Common tool specification:

[sentence_splitter : tcs : <(unsplit, cs, text file, lang)> : <(split, cs, text file, lang)>]

Examples:

1) split-sentences.perl [1] has the following usage:

```
perl split-sentences.perl [-l en|de|...] < textfile > splitfile
```

Any tool implementing the sentence splitter interface will be using input and output containing the language attribute, which can be used to specify the language parameter for the tool (i.e. -l lang).

3.1.3. Tokenizer

Description:

The tokenizer is used to separate sentences in a given language into tokens which represent basic units of information. Tokenization is language dependent. The input is a text file and the output is the tokenized text file in which each token in each sentence is

separated from adjacent tokens by a space.

Input:

A text file in language L1.

Output:

A text file in language L1.

Common tool specification:

[tokenizer : tcs : <(untokenized, cs, text file, lang)> : <(tokenized, cs, text file, lang)>]

Examples:

1) tokenizer.perl [1] has the following usage:

```
perl tokenizer.perl [-l en|de|...] < textfile > tokenizedfile
```

2) Tokenizers for English, French, Italian, Greek, German, Portuguese, and Swedish are provided by Europarl tools [1].

Any tool implementing the tokenizer interface will be using input and output containing the language attribute, which can be used to specify the language parameter for the tool (i.e. -l lang).

3.1.4. Sentence Aligner

Description:

Sentence aligner takes a parallel document as input and produces a set of aligned bichunks where a bichunk is a bilingual pair of chunks which convey the same meaning and each chunk contains at least one sentence. Chunks may contain more than one sentence depending on the sentence aligner's capabilities. Some translations may split a given source sentence during translation and some may convey the same meaning in a single sentence; therefore it is not uncommon to see chunks containing more than one sentence. The input is two text files in languages L1 and L2 representing the documents in the parallel document and the output is two text files in languages L1 and L2 representing

the aligned documents where both files have the same number of lines.

Input:

A text file in language L1.

A text file in language L2.

Output:

A text file in language L1.

A text file in language L2.

Common tool specification:

[sentence_aligner : tcs : <(doc1, cs, text file, lang); (doc2, cs, text file, lang)> : <(aligned_doc1, cs, text file, lang); (aligned_doc2, cs, text file, lang)>]

Examples:

1) Moore's aligner [2] has the following usage:

```
perl align-sents-all.pl input1 input2
```

which outputs into files named input1.aligned and input2.aligned.

3.1.5. Corpus Filterer

Description:

Corpus filtering homogenizes a parallel corpus according to some criteria such as the length of the sentences. This step filters out lines that are unlikely to belong to the domain of interest or likely to be noisy (i.e. very long sentences or bichunks with significantly differing chunk lengths) while maintaining the parallel document property. The input is two text files representing the documents in the parallel document and the output is two text files representing the filtered documents.

Input:

A text file in language L1.

A text file in language L2.

Output:

A text file in language L1.
A text file in language L2.

Common tool specification:

[corpus_filterer : tcs : <(doc1, cs, text file, lang); (doc2, cs, text file, lang)> : <(filtered_doc1, cs, text file, lang); (filtered_doc2, cs, text file, lang)>]

Examples:

1) The following corpus filterer is distributed with Moses [3] with the following usage:

```
perl clean-corpus-n.perl corpus lang1 lang2 clean-corpus min max
```

corpus is the corpus base with corpus.lang1 corresponding to input1 and corpus.lang2 corresponding to input2 in languages lang1 and lang2 correspondingly. clean-corpus is the corpus base for the cleaned corpus. min and max specify the minimum and the maximum number of tokens each chunk can contain.

3.1.6. Instance Selector

Description:

Instance selection prepares translation task specific parallel training data and LM corpora both selected from larger bilingual parallel and monolingual LM corpora for improving translation quality and reducing the computational costs for MT deployment. Parallel Feature Decay Algorithms (FDA) is an instance selection tool that provides fast deployment of accurate statistical machine translation systems [34]. Parallel FDA can be used over the collected SMT resources available like the parallel training corpora and the LM training corpus crawled by Focused Crawler. Parallel FDA raise the expectations from SMT translation outputs with highly accurate translations and lowering the bar to entry for SMT into new domains and tasks by allowing fast deployment of SMT systems in about half a day. The selected LM corpus with Parallel FDA also results in up to 86% reduction in the number of OOVs and up to 74% reduction in the perplexity and allows us to model higher order dependencies with up to 0.0023 increase in BLEU [34]. The input is a text file representing the test document in the source language and a specification file containing the target language to translate to and the paths for the monolingual and bilingual language resources and the output is the selected parallel corpus and the LM corpus for training the SMT model.

Input:

A text file in language L1.

A text file. (FDA model specification file)

Output:

A text file in language L1.

A text file in language L2.

A LM corpus text file in L2.

Common tool specification:

[instance_selector : tcs : <(doc1, cs, text file, lang); (FDAModel, cs, text file, lang)> : <(selected_doc1, cs, text file, lang); (selected_doc2, cs, text file, lang); (selected_doc3, cs, text file, lang)>]

Examples:

1) Parallel FDA [34] has the following usage for selecting the parallel training data:

```
python FDA.py test.source.corpus corpus P lang1 lang2 order
```

test.source.corpus is the test source corpus we are interested to translate, corpus is the available parallel corpus base with corpus.lang1 corresponding to the source side and corpus.lang2 corresponding to the target side. P represents the percentage of the corpus to select (typically 1 million sentences are enough so if we have 100 million sentences in the training corpus, P=1) and order specifies the n-gram order for features used to select the instances.

Parallel FDA [34] has the following usage for selecting the LM corpus:

```
python FDA.py train.target.corpus LM.corpus N lang
```

train.target.corpus is the training target corpus we are using for building the MT system, preferably selected by an earlier parallel FDA step, LM.corpus is the available LM corpus in the target language, N represents the number of sentences to select for the LM corpus (typically 8 to 10 million sentences are a good choice [34]) and lang specifies the target language.

3.1.7. Named Entity Recognizer

Description:

A named entity recognizer detects entities (i.e. names of people or places, numbers) in text. The inputs are a text file and a model file for a given language. The output is a text file with entities labeled.

Input:

A text file in language L1.

A model file in language L1.

Output:

A text file in language L1.

Common tool specification:

[NER : tcs : <(doc, cs, text file, lang); (NERmodel, cs, grammar file, lang)> : <(NER-labeled_file, cs, text file, lang)>]

Examples:

1) OpenNLP tool [35] has the following usage:

```
./opennlp TokenNameFinder en-ner-person.bin < inputFile > outputFile
```

3.1.8. POS Tagger

Description:

A POS tagger assigns each word in the sentence a tag which reflects the word's morhpo-syntactic function (e.g. verb, noun, preposition). The tag might also include additional information such as gender, number, or tense. The POS tag set can vary even for the same language. Penn Treebank tag set [9] is commonly used for English. Universal tag set [29] provides conversion tools for tags used in different treebanks and languages into 12 common POS tags. The input of the POS tagger is a text file and a model file for a given language. The output is a POS-tagged text file, which has a POS tag, *POStag,* assigned to each word *w* in the input sentence separated by space:

w|POStag

Input:

A text file in language L1.

A model file in language L1.

Output:

A text file in language L1.

Common tool specification:

[POStagger : tcs : <(set, cs, text file, lang)>; (POSModel, cs, text file, lang)> : <(POS-tagged_file, cs, text file, lang)>]

Examples:

1) Stanford POS tagger [10] for English has the following usage:

```
./stanford-postagger.sh models/left3words-wsj-0-18.tagger inputFile
> outputFile
```

3.1.9. Parser

Description:

A parser provides an analysis of the syntactic structure of a sentence according to a grammar formalism. The inputs are a text file and a grammar[5] file that contains the grammar formalism for the parser. The output is a text file representing the parser output which follow the CoNLL-X data format [8,21], which is the common format used for parser output. The parsing output for each sentence is separated by a blank line and for each token in the sentence, there is a line that contains ten fields separated by tab. Fields are not allowed to contain space or blank characters. More details and examples are given in the CoNLL-X website [8,21].

Input:

A text file in language L1.

A grammar file in language L1.

[5] While this knowledge resource is commonly considered as an internal part of the tool for other NLP tasks (e.g. PoS tagging, morphological analysers), in the case of parsers we specify it as an input parameter due to the fact that most state-of-the-art parsers take this resource as input.

Output:

A text file in language L1.

Common tool specification:

[Parser : tcs : <(doc, cs, text file, lang); (grammar, cs, grammar file, lang)> : <(parse, cs, text file, lang)>]

Examples:

1) Stanford parser [6,11] has the following usage:

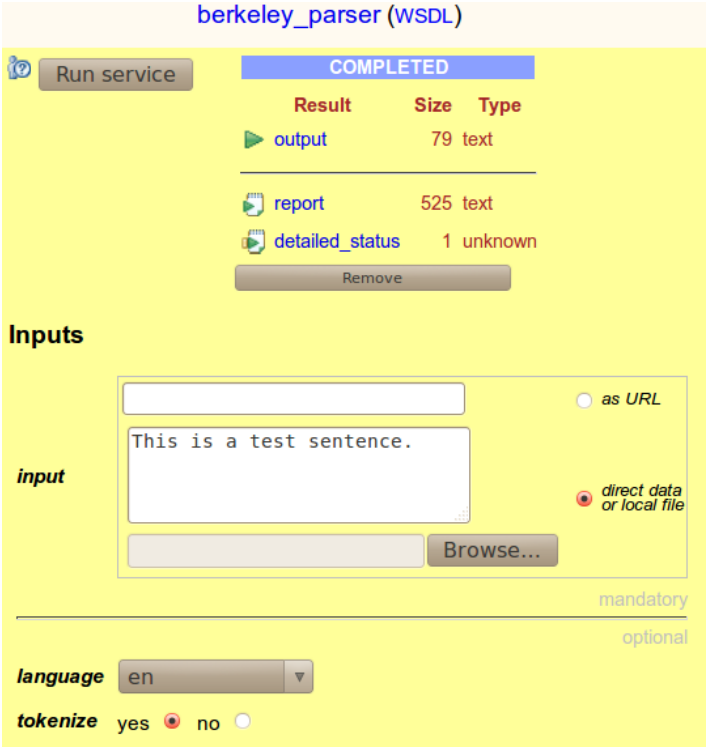
```
./lexparser.sh inputFile > outputFile
```

2) CCG parser [12] has the following usage:

```
bin/parser --parser models/parser --super models/super
--input=input.pos --output=out.ccg
```

3) Parsers (and other categories of language technology tools) included in the PANACEA web service platform[6] have a common set of mandatory parameters (input text in the case of parsers) while optionally they can offer more parameters (specific to the parser at hand). For example, for the Berkeley parser, optional parameters include the language and whether the input should be tokenised. The next Figure shows a snapshot of this web service, with the mandatory and optional input parameters.

[6] http://registry.elda.org/



The output is then given in a link ("output"), whose contents for this execution would be:

((S (NP (DT This)) (VP (VBZ is) (NP (DT a) (NN test) (NN sentence))) (. .)))

3.1.10. Morphological Analyzer

Description:

A morphological analyzer extracts linguistic information conveyed by the morphological form of the words, analyzes the words in terms of their morphemes, and performs morphological processing on the words such as stemming and lemmatization. The input is a text file and the output is the XML file containing the morphological analysis of each word in the input file. The morphological features produced along with its format depends on the tool used to analyze the text and the language. Morphological analysis can be expressed in a generalized XML format according to the TEI-P3 standard [7], which provides the ability to represent the information of the text in terms of feature structures. Feature structures are general-purpose data structures with each feature having a name and one or more values. The XML format allows to represent the morphologically analyzed text in a standard way.

Input:

A text file in language L1.

A model file in language L1.

Output:

An XML file in language L1.

Common tool specification:

[MorphologicalAnalyzer : tcs : <(doc, cs, text file, lang)> : <(morphological_analysis, cs, XML file, lang)>]

Examples:

1) Morph [14]: a morphological analyzer for Czech.

2) Stanford lemmatizer for English [15].

3.1.11. Speech Recognizer

Description:

A speech recognizer converts vocal utterances to text. The inputs are a recorded speech file and a model file for a given language, which can include an acoustic model, a language model, and a dictionary specification containing phonetic transcriptions of words. The output is text file.

Input:

A speech file uttered in language L1.

A model file in language L1.

Output:

A text file in language L1.

Common tool specification:

[SpeechRecognizer : tcs : <(speech, cs, speech file, lang); (SRModel, cs, text file, lang)> : <(text_file, cs, text file, lang)>]

Examples:

1) CMU Sphinx tool [35].

3.1.12. Speech Generator

Description:

A speech generator or a text-to-speech program converts a text to speech. The input is a text file and the output is a speech file.

Input:

A text file in language L1.

Output:

A speech file in language L1.

Common tool specification:

[SpeechGenerator : tcs : <(text, cs, text file, lang)> : <(speech_file, cs, speech file, lang)>]

Examples:

1) eSpeak [36] has the following usage (output in WAV format):

```
espeak -f inputFile -w outputFile
```

3.2. Development Tools

The tools in this category are used for the development of MT systems. The general MT development workflow, which contains the training, tuning, and decoding steps and the final output is the translation, is presented in D3.1.1. This section focuses on the tools used for developing MT systems and their specification.

3.2.1. Word Aligner

Description:

A word aligner learns mappings between the tokens in the bichunks and creates a word alignment file which specifies word translation correspondences for each sentence pair in the parallel training set. The input is a parallel training set and the output is a word alignment file, which can be used to create the phrase table using a phrase extractor. The word alignment file produced by the word aligner contains alignments between the source and target tokens of each bichunk in the parallel corpus. Each line in this file contains a list of word alignments separated by a space. Each alignment is a pair of indices corresponding to the source and target token indices in the source and the target chunk, respectively. Indices start from 0 but not all the tokens in a sentence need to be specified in the alignments as some tokens may not be aligned to any other token or alignments were not found for the bichunk by the word aligner. Hence, each alignment *a* is represented as *i-j* where $0 \leq i \leq I-1$ and $0 \leq j \leq J-1$ where I is the length of the source chunk and J is the length of the target chunk.

Input:

A text file in language L1.

A text file in language L2.

Output:

A text file.

Common tool specification:

[WordAligner : tcs : <(set1, cs, text file, lang); (set2, cs, text file, lang)> :

<(word_alignment_file, cs, text file, lang)>]

output is language independent and its language may be specified using the pair of languages used for input1 and input2.

Examples:

1) GIZA++ [8] provides a suite of tools for creating co-occurrence tables and word alignments.

3.2.2. Phrase Extractor

Description:

A phrase extractor uses the word alignment information to extract phrases consisting of sequences of words, which can be used during translation. The input is a word alignment file and the parallel training set used for obtaining the alignments and the output is the phrase translation table or the phrase table. A phrase table stores possible translations for the phrases in the source language. The phrase table format follows Moses' format [3] where each line presents a phrase table entry composed of a source phrase, a target phrase it translates to, and a set of scores for evaluating the quality of the translation all separated by "|||" and the scores separated by a space:

source phrase ||| target phrase ||| score*1* score*2* … score*n*

If the scores represent probabilities, they are normalized (i.e. $\sum_i p(t_{phrase_i} | s_{phrase}) = 1$ for t_{phrase_i} representing the *i*th target phrase and s_{phrase} representing the source phrase).

Input:

A text file in language L1 specifying word alignments.

A text file in language L1.

A text file in language L2.

Output:

A text file.

Common tool specification:

[PhraseExtractor : tcs : <(word_alignment_file, cs, text file, lang); (set1, cs, text file, lang); (set2, cs, text file, lang)> : <(phrase_table, cs, text file, lang)>]

output is language independent and its language may be specified using the pair of languages used for input1 and input2.

Examples:

1) Moses [3] training script `train-model.perl` creates the phrase table as one of the outputs.

3.2.3. Language Modeler

Description:

A language modeler builds a language model (LM), which contains the probabilities for calculating the likelihood of observing text based on the input text file. The input is a text document and the output is a LM file in ARPA format [18].

Input:

A text file in language L1.

Output:

An ARPA file.

Common tool specification:

[LanguageModeler : tcs : <(doc, cs, text file, lang)> : <(LM, cs, ARPA file, lang)>]

Examples:

1) SRILM [19] LM toolkit has the following usage:

```
ngram-count -text doc -lm LM
```

2) KenLM toolkit [20] has the following usage:

```
bin/lmplz < doc > LM
```

3.2.4. Parameter Optimizer

Description:

Parameter optimization or tuning aims at finding the set of MT model weights maximizing translation quality according to an automatic evaluation metric such as BLEU [27] on the development set. The inputs to the parameter optimizer are a parallel development set, a text file for the automatic evaluator specification, and a text file for the MT model specification including the decoder, and other resources used such as the language model, and the phrase table. The specifications contain information about the paths for the components together with an example usage. The output is a text file containing the optimized MT model weights, optimizing the MT model translation quality on the development set according to the automatic evaluator, and the score obtained with the weights.

Input:

A text file in language L1. (development set)
A text file in language L2. (development set)
A text file. (automatic evaluator specification)
A text file. (MT model specification)

Output:

A text file.

Common tool specification:

[ParameterOptimizer : tcs : <(devset1, cs, text file, lang); (devset2, cs, text file, lang); (evaluator, cs, text file, lang); (MTModel, cs, text file, lang)> : <(weights, cs, text file, lang)>]

Examples:

1) MERT [5]. A script mert-moses.pl for tuning using MERT is provided by Moses [3] and has the following usage:

```
mert-moses.pl devset_L1 devset_L2 MosesDecoderBinary moses.ini
```

3.2.5. Decoder

Description:

A decoder produces the translation with the highest score or an N-best list for an input source text. The inputs it takes are a text file in the source language and another text file which is the MT model specification, specifying the MT model specific resources including the phrase table, language model, and the model weights learned after parameter optimization and whether the highest scoring translation or the N-best list is needed. The decoder can choose to use a subset of this information. The output is a text file containing the translation or the N-best list for each line in the input text. The N-best list format follows Moses's format for N-best lists[7] where each line is composed of (i) the sentence number, (ii) the translation, (iii) unweighted individual component scores, and (iv) weighted overall score, all of which are separated by "|||".

Input:

A text file in language L1.
A text file. (MT model specification)

Output:

A text file in language L2.

Common tool specification:

[Decoder : tcs : <(source, cs, text file, lang); (MTModel, cs, text file, lang)> : <(translation, cs, text file, lang)>]

Examples:

1) Moses [3,22] provides a suite of tools including Phrase-Based and Hierarchical Phrase-Based decoders.

3.2.6. True Caser

Description:

[7] Moses manual (sect. 3.3.8, Generating n-Best Lists): http://www.statmt.org/moses/manual/manual.pdf

A true caser or recaser recases lower cased data using a model learnt from a cased monolingual training corpus. The input is a text file in lower case and a text file for the true caser model specification and the output is the cased version of the input.

Input:

A text file in language L1.

A text file. (Recaser model specification)[8]

Output:

A text file in language L1.

Common tool specification:

[Recaser : tcs : <(lowercased, cs, text file, lang); (RecasingModel, cs, text file, lang)> : <(truecased, cs, text file, lang)>]

Examples:

1) Moses [3] recaser script recase.perl has the following usage:

```
perl recase.perl --in lowercased  --model model/moses.ini  --moses MosesDecoderBinary > truecased
```

[8] The format of the truecaser model contains one word per line, and for each word it specifies the total number of occurrences, and the number of occurrences it appears as upper case and lower case. E.g. “leyes (4/5) Leyes (1)”, the word leyes appears 5 times in the training set, out of which 4 times as lower case and 5 times as upper case.

3.3. Evaluation Tools

Automatic evaluation uses a reference to evaluate the performance of the translation, its inputs being the translation, the reference, and the evaluator. The evaluator can choose to use only the textual information present in the translation and the reference sentences or incorporate extra linguistic information for the evaluation. The output evaluation score represents the quality of the translation with respect to the reference translation. Evaluation workflow is presented in D3.1.1, which details the flow of information during evaluation.

3.3.1. Automatic Evaluator

Description

The MT automatic evaluation tools provide a score of translation quality for a translated segment or a translated document in comparison with one or more reference translations. The inputs are a translation output file and a reference file, which contains one or more reference translations for each sentence in the translation output file. Multiple references for each sentence are hard to find but can result with better evaluation of the translations possibly by producing a score for paraphrases as well. The output is a score which represents the translation quality of the input document according to the reference document in terms of an MT evaluation metric (e.g. BLEU [27], TER, METEOR [17], NIST). The output can optionally contain scores for each sentence or units of information with which evaluation is performed.

Input:

A text file in language L1.

A text file in language L1.[9]

Output:

A text file.

[9] If there are multiple references, they can be specified as multiple files (instead of only one file when we are working with one reference). This is the format of the *multi-bleu.perl* script included with the Moses toolkit.

Common tool specification:

[AutomaticEvaluator : tcs : <(translation, cs, text file, lang); (reference, cs, text file, lang)> : <(scores, cs, text file, lang)>]

Examples:

1) mteval-v11b.pl [16] calculates the NIST and BLEU scores and has the following usage:

```
perl mteval-v11b.pl -r referenceFile.sgm -t MToutputFile.sgm -s sourceFile.sgm -c
```

2) METEOR [17] has the following usage:

```
java -Xmx2G -jar meteor-*.jar MToutputFile ReferenceFile -norm -writeAlignments -f EvalOutput
```

3) DELiC4MT [38] is a tool for MT evaluation over user-defined linguistic phenomena such as the nouns followed by a specific preposition and it is also available as a web service from http://registry.elda.org/services/301.

4) Asiya [30] is an automatic evaluation tool for machine translation incorporating lexical, syntactic, and semantic similarity metrics.

3.3.2. Automatic Error Analyzer

Description

Automatic error analyzer classifies the types of errors and the rate at which they are encountered in the translation output. The inputs are a translation output file and a reference file, which contains one or more reference translations for each sentence in the translation output file. The output is a report presenting the accuracy categorized according to error types. Error types include morphological errors, incorrect word order, missing words, extra words and incorrect lexical choice.

Input:

A text file in language L1.

A text file in language L1.

Output:

A text file and/or an HTML file.

Common tool specification:

[AutomaticErrorAnalyzer : tcs : <(translation, cs, text file, lang); (reference, cs, text file, lang)> : <(report, cs, text file, lang)>]

Examples:

1) hjerson [22] has a following usage:

```
hjerson.py --ref example.ref --hyp example.hyp --baseref
example.ref.base --basehyp example.hyp.base [--addref example.ref.pos]
[--addhyp example.hyp.pos] [--html example.html] [--cats example.cats]
[--sent example.sentence-error-rates] > example.total-error-rates
```

The required input files are the translation reference, the translation hypothesis, base forms of the translation reference, and base forms of the translation hypothesis. Additional information (such as POS tags) can be also included optionally. The default output are the overall (document level) scores for each error class. It is also possible to get sentence level scores, translation reference and hypothesis tagged with error classes, as well as an HTML file presenting the errors.

2) Addicter [23] consists of a monolingual aligner, an error detector and labeller, a corpus browser, and an alignment summarizer. Detailed instructions of downloading, installing and using Addicter can be found at https://wiki.ufal.ms.mff.cuni.cz/user:zeman:addicter.

3.3.3. Quality Estimator: QuEst

Description:

Quality estimation tool QuEst [28] being developed as part of WP2 estimate the quality of the translation output based solely on the source sentence and/or its target translation without depending on a reference translation. Since an SMT system might encounter modeling or search errors during decoding [31], extrinsic and independent evaluation and prediction of performance and quality is important [32,37]. Quality estimator predicts the quality of the translations without a reference translation by learning over the features

derived from the source sentence, its translation, and possibly additional external resources. In so doing, intrinsic by-products of MT model development workflows can be used including the phrase table, the training, the development, and the test sets, and the target LM. Quality estimation workflow is presented in D3.1.1, which details the flow of information during quality estimation. The output of quality estimation is a quality prediction score representing the quality of the translations as would be obtained using automatic evaluation with a reference translation. Quality estimator takes two text files which contain the source sentences and their translations in the target language as input, creates an intermediate features file after feature extraction, and outputs a text file containing the quality estimation scores for each sentence. Quality estimator interface is divided into two parts: feature extraction and prediction.

Input:
A text file in language L1.
A text file in language L2.

Intermediate Output:
A text file in language L1.

Output:
A text file.

Common tool specification:

[quality_estimator : tcs : <(sources, cs, text file, lang); (translations, cs, text file, lang)> : <(scores, cs, text file, lang)>]

Output is language independent and its language may be specified using the pair of languages used for input1 and input2.

Examples:
1) WMT'12 Quality Estimation Task baseline software [3] and QuEst, the WMT'13 Quality Estimation Task baseline software [28].

3.4. Translation Quality Assessment

Translation quality assessment (QA) is important but can be subjective according to the reviewers who mark the errors. QTLaunchPad is pioneering a new approach for QA based on the multidimensional quality metrics (MQM), which attempts to clearly define the quality expectations from translation output.

3.4.1. Multidimensional Quality Metrics (MQM)

The main goals of MQM can be summarized as:

- adding quality expectations in terms of fluency, accuracy, and adequacy,
- normalizing according to problems sourcing from the quality of the given source text,
- unifying the evaluation of machine translation outputs and the translations by humans

into the evaluation.

MQM identifies various issue types with which the quality score is obtained: monolingual *fluency* of the sentence, bilingual *accuracy* of the translations for the words, and the *verity* measuring in terms of semantic and pragmatic the fit of the text for the purpose. MQM are discussed in more detail in deliverable D1.1.2.

3.4.2. MQM Evaluation

The MQM quality score, Q, provides the overall rating of quality and is calculated as follows:

$$Q = 100 - P_A - P_F - P_V + (\Delta_F + \Delta_V)$$

where:

- P_A, P_F, and P_V are the penalties for accuracy, fluency, and verity issues.
- Δ_F and Δ_V are the difference between the source and target fluency and verity respectively.

MQM quality score rewards translators for fixing the problems found in the original source sentence.

3.4.3. MQM Evaluation Discussion

MQM and its evaluation is still under development. In this section, we discuss some of its important elements, which allows us to develop the human interfaces for quality assessment and describe its inputs, outputs, and attributes.

MQM separates source and target fluency and end-user adequacy scores to reward translators for fixing the problems found in the original source sentence. Transcreation [24] is the process by which translators create and adapt according to the audience or the domain (i.e. american football event description would be transcreated in another way to local audience in Europe).

MQM unifies the evaluation of machine translation outputs and the translations by humans into the evaluation. Additionally, MQM evaluation formula in a sense contains post-editing within it by allowing the edition and correction of source sentences. The pre-editing of the source sentences before translation can create a variety of translation tasks from a single sentence, which allows richer comparison. One of the principles of MQM is comparability, which aims that the results of one assessment should be comparable with others. Allowing edits of the source is stretching this principle by making the translations for the same source sentence less comparable with each other. But at the same time, edits allow richer content, multiple choices over alternative translations, and multiple-reference translations to become possible. Multiple references for each sentence are hard to find but can result with better evaluation of the translations possibly by producing a score for paraphrases as well. In many machine translation challenges, the presence of multiple-reference translations for the same sentence is encouraged to better evaluate the performance. MQM takes the multiple-reference approach further by also allowing and encouraging multiple close source sentence edits to be used as the source for the translation.

Additionally, the initial TQ score (100) can be made to vary according to the quality of the source sentence given. For instance, if we have tweets as the source sentence for translation, they may not be in a grammatically correct form and hence our expected quality of their translation would be low. In practice, we may be content with $A = 80$ for each translation but we may want to get some important topics to be translated perfectly. Therefore, we may prefer accurate translations for every word being translated, making a precision level of 100, but we can accept missing some of the target words. Given a

precision of 100 and a recall of 67, we reach $F_1 = 80$, which can be used to set the practical level for A. Using F_1 over the words can allow us to give thresholds for quality that better reflect the expectations for translation. F_1 correlates well with human judgments of translation quality and performs better than BLEU [27] according to the automatic evaluation metrics challenge results on some translation tasks [25,26,33].

3.4.4. Human MQM Evaluator

Description

In the human MQM evaluation, humans are provided with a source sentence or document and its translation without reference translations and are asked to fill in the translation quality assessment file, which consists of the markings of the errors according to the MQM evaluation. The inputs are a source input file, a translation output file, and a display file, which merges the source sentences and their translations for each sentence in the source input and translation output file for easier visualization. The source input file and the translation output file have the same number of lines in languages L1 and L2 representing the source sentences in language L1 and their translations in L2. The input display file presents each source sentence and its translation one after the other after tokenization together with an index for each word in each sentence. An example sentence for German-English translation is given below:

ich habe hier eine kopie der einigung vom dezember des vergangenen jahres .
0 1 2 3 4 5 6 7 8 9 10 11 12
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
i have in front of me a copy of the agreement of last december .

The first line is the source sentence (German), the second line contains the indices for the source words centered in the middle of each word, the third line contains the indices for the target words, and the fourth line contains the target sentence (English). Each of these four display lines are separated by a blank line. The format of the display file helps the translator identify the words in each sentence and link them easily. The display file can also be used for manual word alignment tasks. The output is a translation quality

assessment (TQA) file marking the errors found in each translation. A special format for TQA files may be specified later.

Input:

A text file in language L1.

A text file in language L2.

A text file.

Output:

A text file.

Common tool specification:

[HumanMQMEvaluator : tcs : <(source, cs, text file, lang); (translation, cs, text file, lang); (display, cs, text file, lang)> : <(TQA, cs, text file, lang)>]

3.4.5. MQM Evaluation Scorer

Description

The automatic MQM evaluation scorer calculates the score of the translation quality assessment according to the MQM evaluation configuration. The inputs are a TQA file and a MQM specification file, which specifies the parameters used when calculating the MQM evaluation score using the markings present in the TQA file. A sample MQM description file in XML format is given in D1.1.2[10]. The output is a score, Q, representing the translation quality assessment of the inputs according to the MQM evaluation. The output contains scores for each sentence and can contain a score for each document as well.

Input:

A text file. (TQA file)

A text file. (MQM specification)

Output:

A text file.

[10] D1.1.2: Section 10 Appendix B: Sample metrics description file.

Common tool specification:

[MQMEvaluationScorer : tcs : <(TQA, cs, text file, lang); (MQMModel, cs, text file, lang)> : <(scores, cs, text file, lang)>]

4. Description of Interfaces

In this section, we describe the common standards for the interfaces. A tool interface defines the inputs, the outputs, and the attributes of the tool together with the attributes of the inputs and the outputs. We use *acyclic graphs* to represent the interfaces where the center vertex represents the tool to where all edges are connected to and the tool uses the inputs to produce the outputs. Inputs are represented with **blue** circles, the tool vertex is represented with a **green** rectangle, the outputs with **red** circles, and all the attributes with **gray** rectangles. Directed edges represent the flow of information from the inputs to the tool or from the tool to the outputs. Undirected edges represent the attributes the tool has. We also use undirected edges colored in **orange** to link between the corpora that are part of a parallel corpora in the inputs or the outputs.

In the following subsections, the interface standards for different tools belonging to different tool categories are described accompanied by a diagram of the acyclic graph representation. The common interface description is given below:

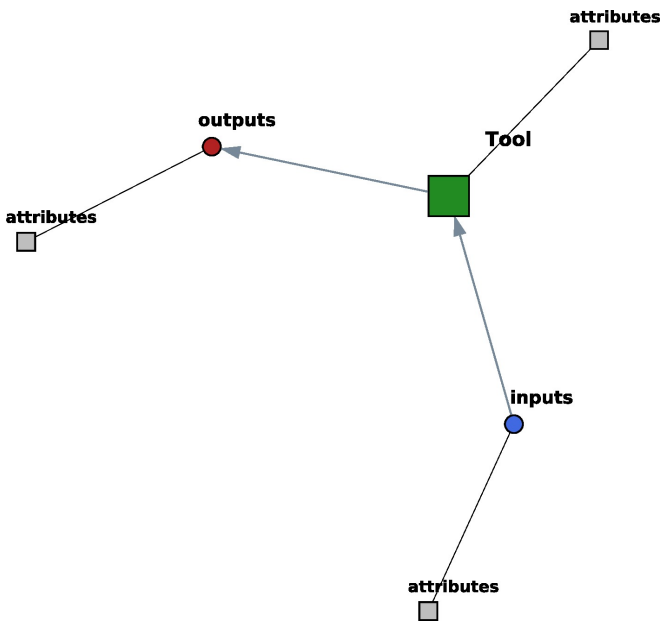


Figure 2: Common interface.

4.1. Preparation

4.1.1. Focused Crawler

Input:

A text file that contains a list of seed URLs.

Output:

A text file that contains lists of URLs pointing to stored versions of domain-relevant crawled pages.

Common tool specification:

[focused_crawler: tcs : <(seedURLs, cs, text file, lang)> : <(corpusURLs, cs, text file, lang)>]

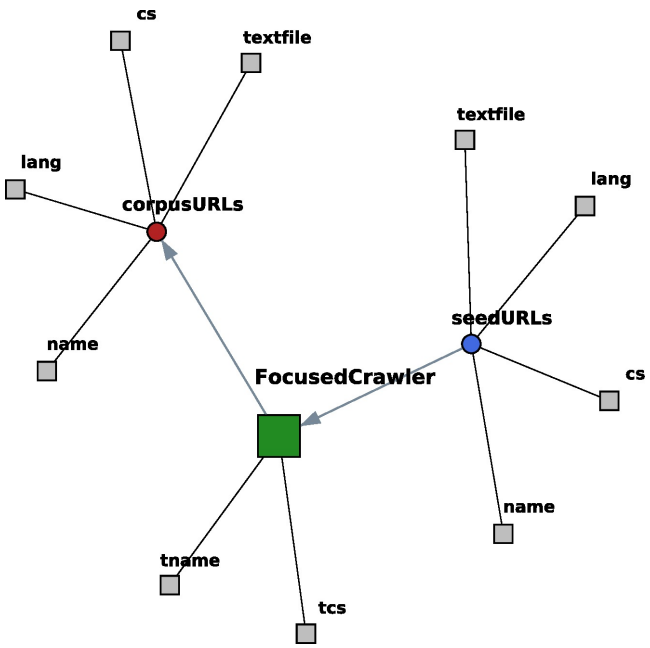


Figure 3: Interface for the focused crawler.

4.1.2. Sentence Splitter

Input:

A text file in language L1.

Output:

A text file in language L1.

Common tool specification:

[sentence_splitter : tcs : <(unsplit, cs, text file, lang)> : <(split, cs, text file, lang)>]

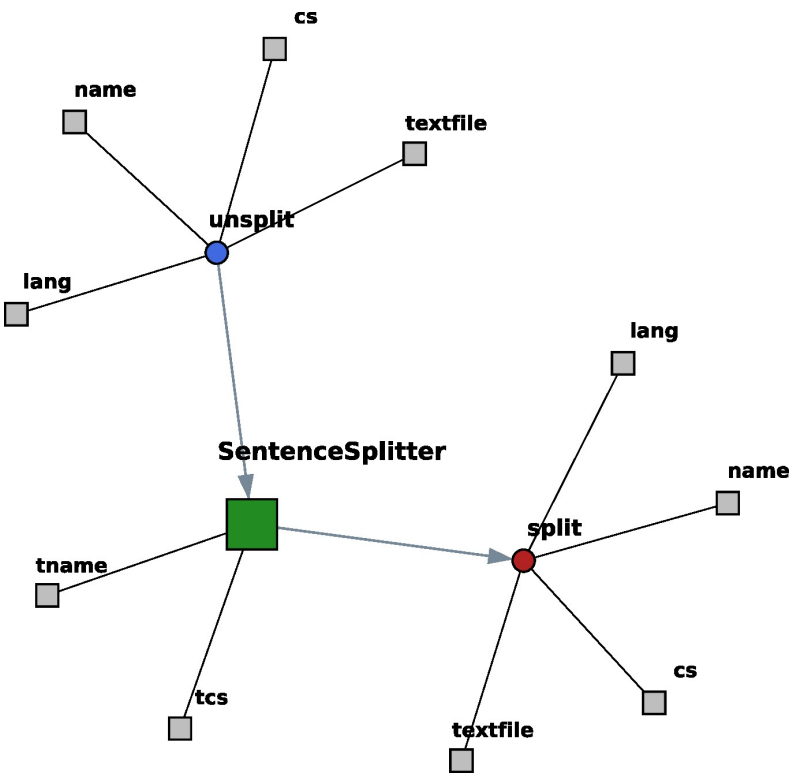


Figure 4: Interface for the sentence splitter.

4.1.3. Tokenizer

Input:

A text file in language L1.

Output:

A text file in language L1.

Common tool specification:

[tokenizer : tcs : <(untokenized, cs, text file, lang)> : <(tokenized, cs, text file, lang)>]

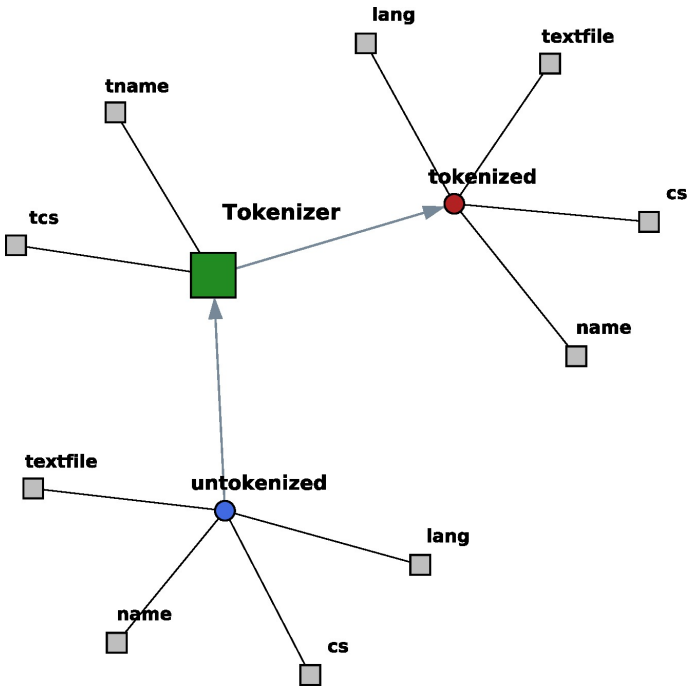


Figure 5: Interface for the tokenizer.

4.1.4. Sentence Aligner

Input:

A text file in language L1.

A text file in language L2.

Output:

A text file in language L1.

A text file in language L2.

Common tool specification:

[sentence_aligner : tcs : <(doc1, cs, text file, lang); (doc2, cs, text file, lang)> : <(aligned_doc1, cs, text file, lang); (aligned_doc2, cs, text file, lang)>]

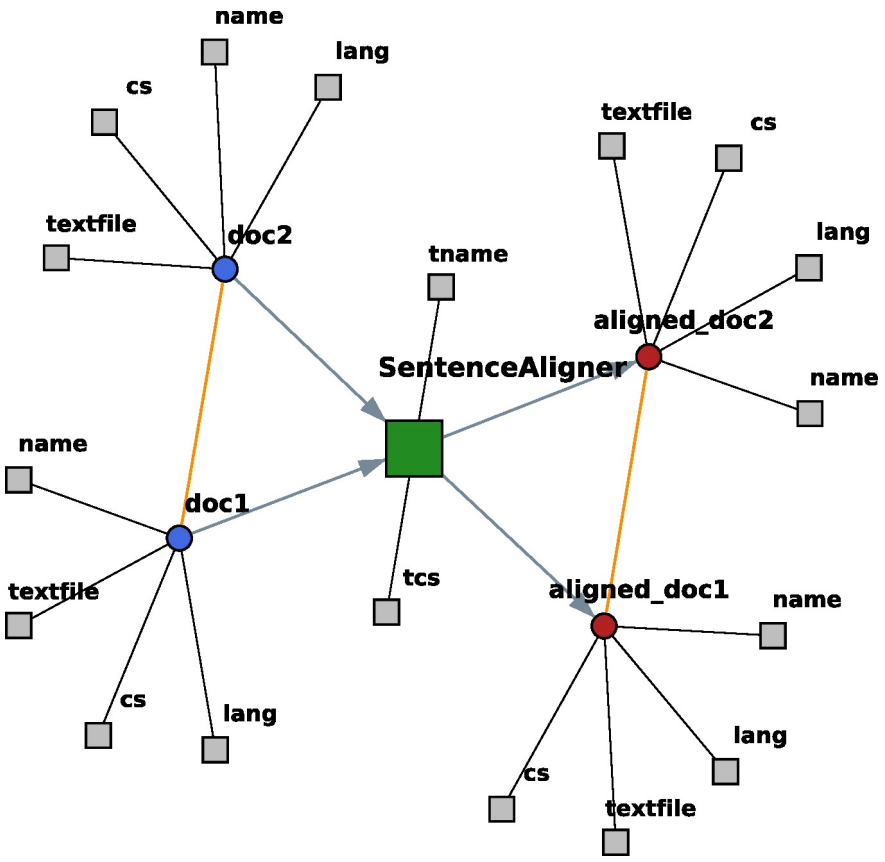


Figure 6: Interface for the sentence aligner.

4.1.5. Corpus Filterer

Input:

A text file in language L1.

A text file in language L2.

Output:

A text file in language L1.

A text file in language L2.

Common tool specification:

[corpus_filterer : tcs : <(doc1, cs, text file, lang); (doc2, cs, text file, lang)> : <(filtered_doc1, cs, text file, lang); (filtered_doc2, cs, text file, lang)>]

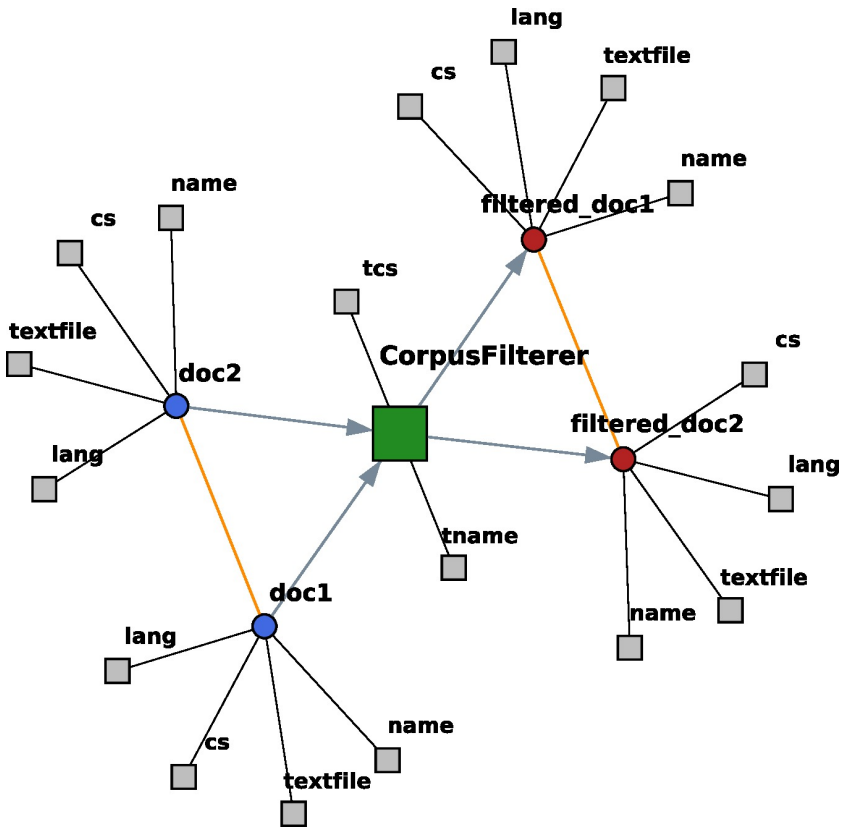


Figure 7: Interface for the corpus filterer.

4.1.6. Instance Selector

Input:

A text file in language L1.

A text file. (FDA model specification file)

Output:

A text file in language L1.

A text file in language L2.

A LM corpus text file in L2.

Common tool specification:

[instance_selector : tcs : <(doc1, cs, text file, lang); (FDAModel, cs, text file, lang)> : <(selected_doc1, cs, text file, lang); (selected_doc2, cs, text file, lang); (selected_doc3, cs, text file, lang)>]

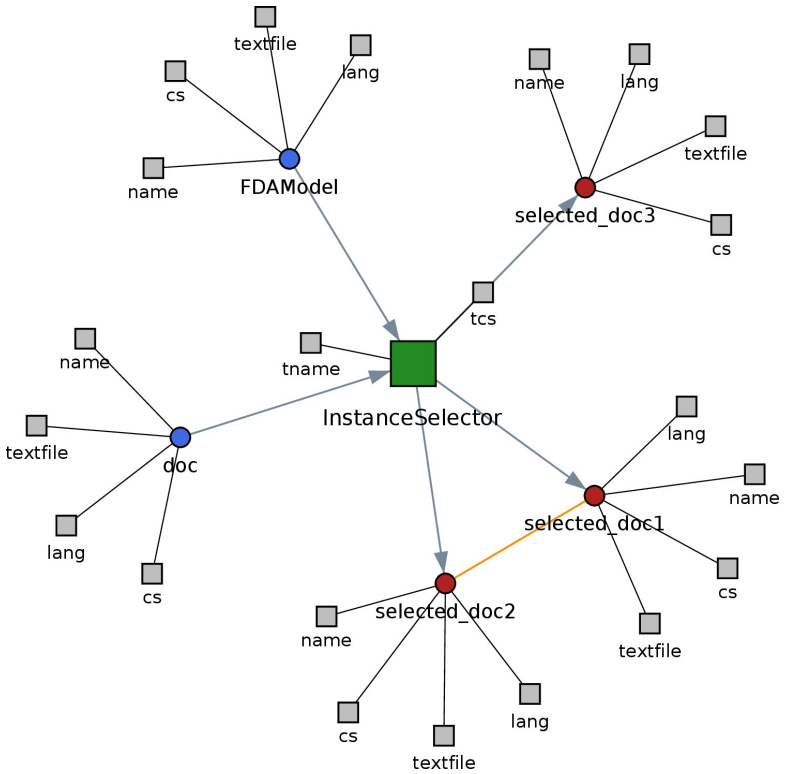


Figure 8: Interface for the instance selector.

4.1.7. Named Entity Recognizer

Input:

A text file in language L1.

A model file in language L1.

Output:

A text file in language L1.

Common tool specification:

[NER : tcs : <(doc, cs, text file, lang); (NERmodel, cs, grammar file, lang)> : <(NER-labeled_file, cs, text file, lang)>]

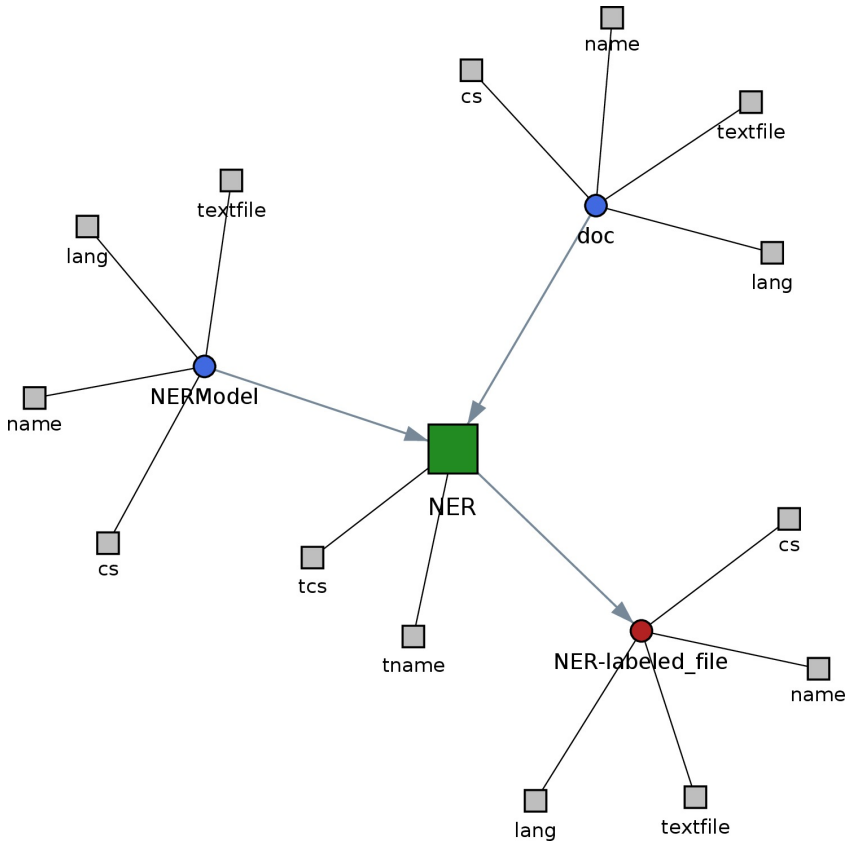


Figure 9: Interface for the NER.

4.1.8. POS Tagger

Input:

A text file in language L1.

A model file in language L1.

Output:

A text file in language L1.

Common tool specification:

[POStagger : tcs : <(set, cs, text file, lang)>; (POSModel, cs, text file, lang)> : <(POS-tagged_file, cs, text file, lang)>]

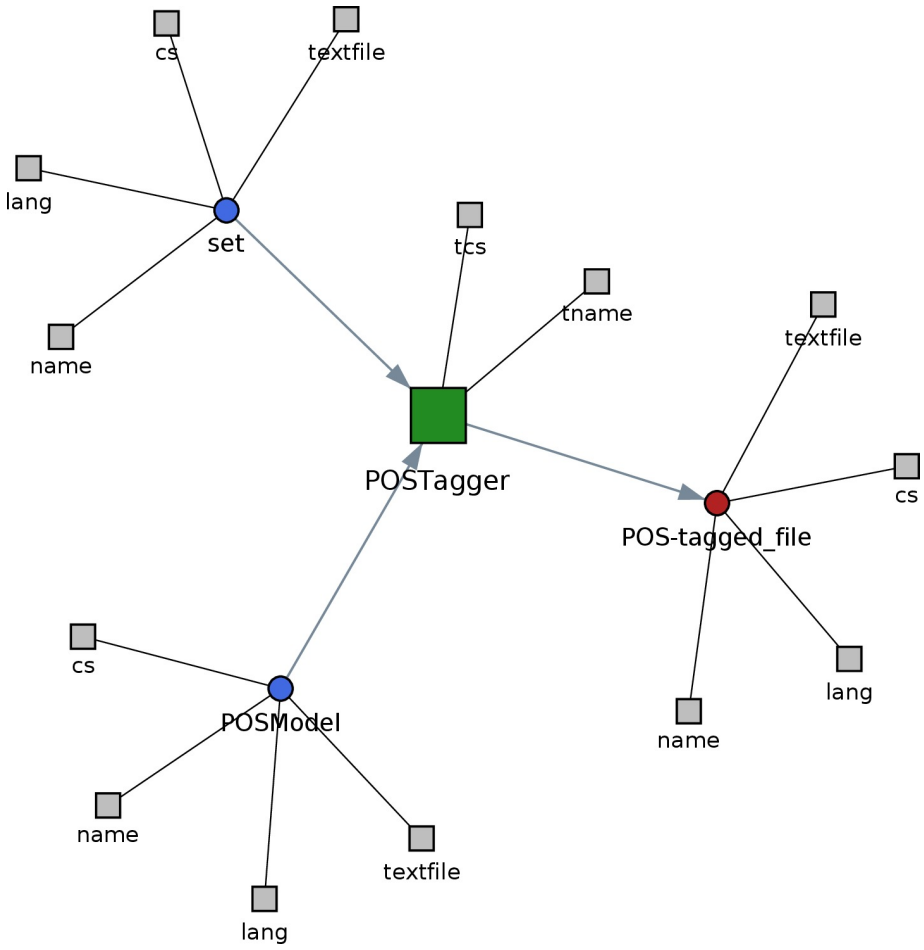


Figure 10: Interface for the POS tagger.

4.1.9. Parser

Input:

A text file in language L1.

A grammar file in language L1.

Output:

A text file in language L1.

Common tool specification:

[Parser : tcs : <(doc, cs, text file, lang); (grammar, cs, grammar file, lang)> : <(parse, cs, text file, lang)>]

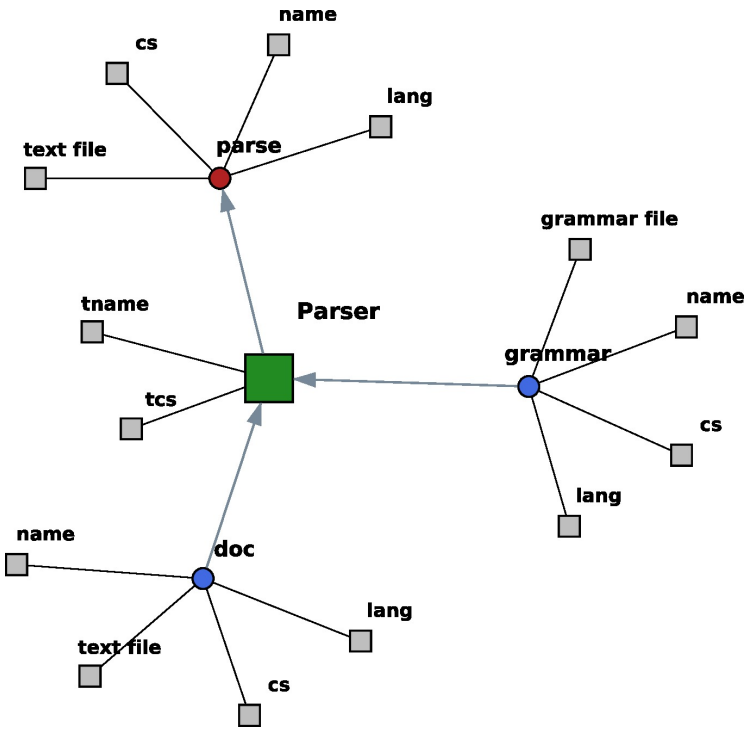


Figure 11: Interface for the parser.

4.1.10. Morphological Analyzer

Input:

A text file in language L1.

Output:

An XML file in language L1.

Common tool specification:

[MorphologicalAnalyzer : tcs : <(doc, cs, text file, lang)> : <(morphological_analysis, cs, XML file, lang)>]

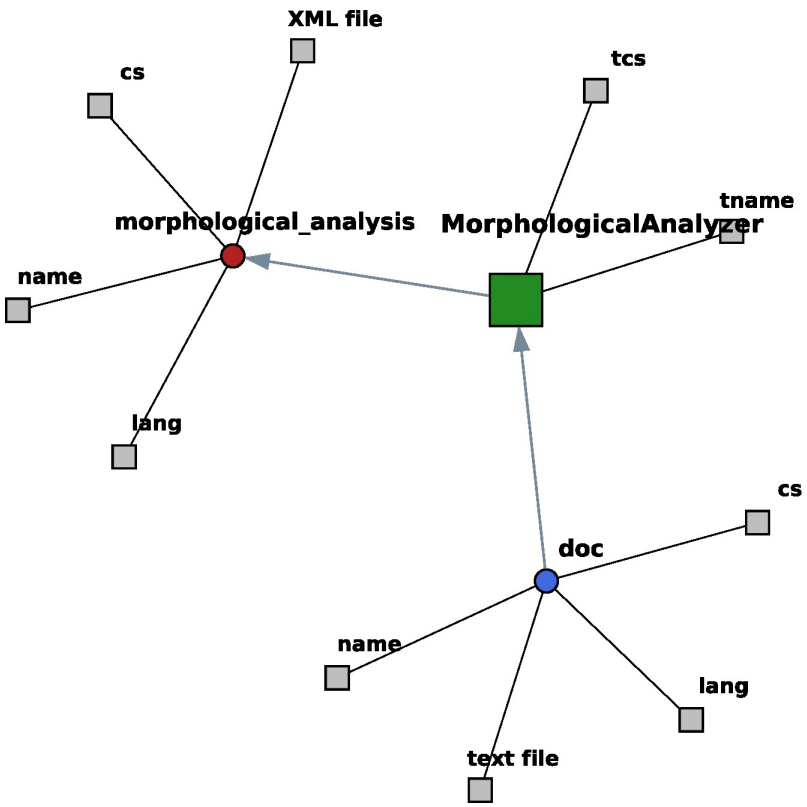


Figure 12: Interface for the morphological analyzer.

4.1.11. Speech Recognizer

Input:

A speech file uttered in language L1.

A model file in language L1.

Output:

A text file in language L1.

Common tool specification:

[SpeechRecognizer : tcs : <(speech, cs, speech file, lang); (SRModel, cs, text file, lang)> : <(text_file, cs, text file, lang)>]

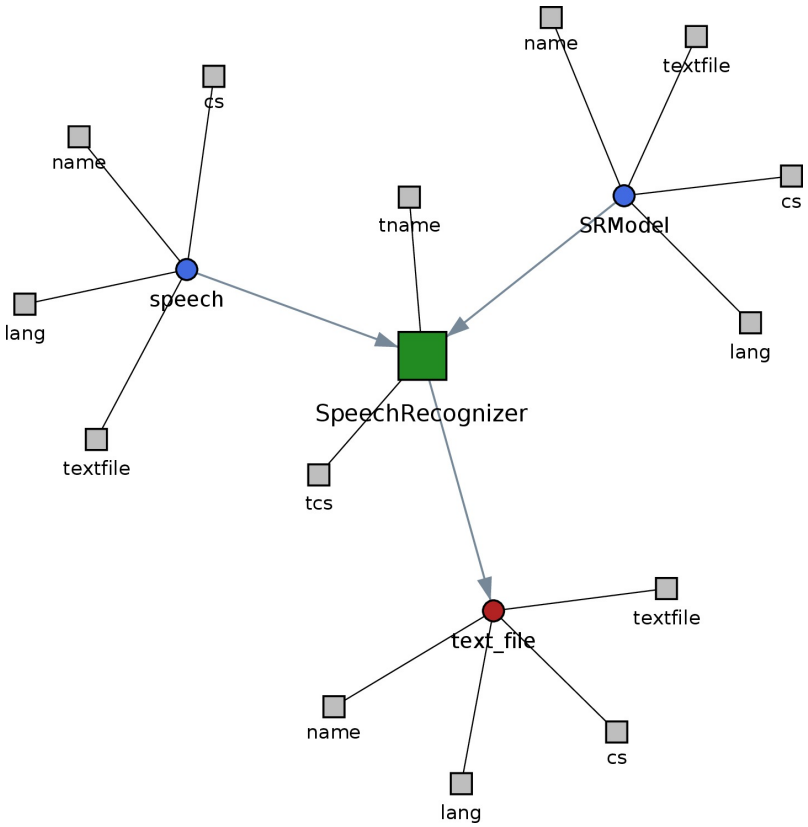


Figure 13: Interface for the speech recognizer.

4.1.12. Speech Generator

Input:

A text file in language L1.

Output:

A speech file in language L1.

Common tool specification:

[SpeechGenerator : tcs : <(text, cs, text file, lang)> : <(speech_file, cs, speech file, lang)>]

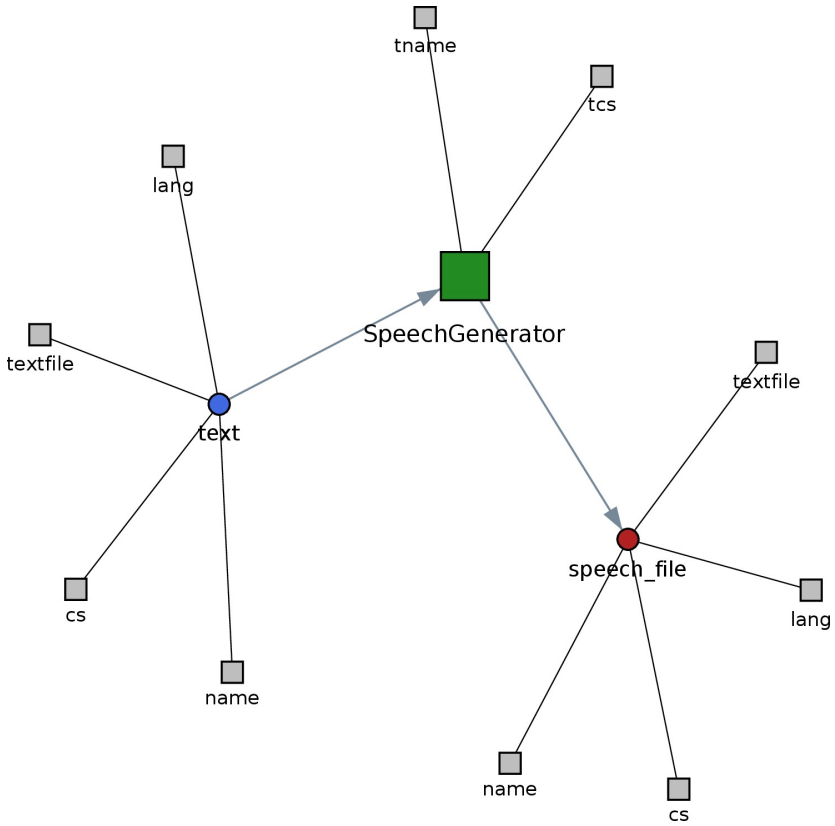


Figure 14: Interface for the morphological analyzer.

4.2. Development

4.2.1. Word Aligner

Input:

A text file in language L1.

A text file in language L2.

Output:

A text file.

Common tool specification:

[WordAligner : tcs : <(set1, cs, text file, lang); (set2, cs, text file, lang)> : <(word_alignment_file, cs, text file, lang)>]

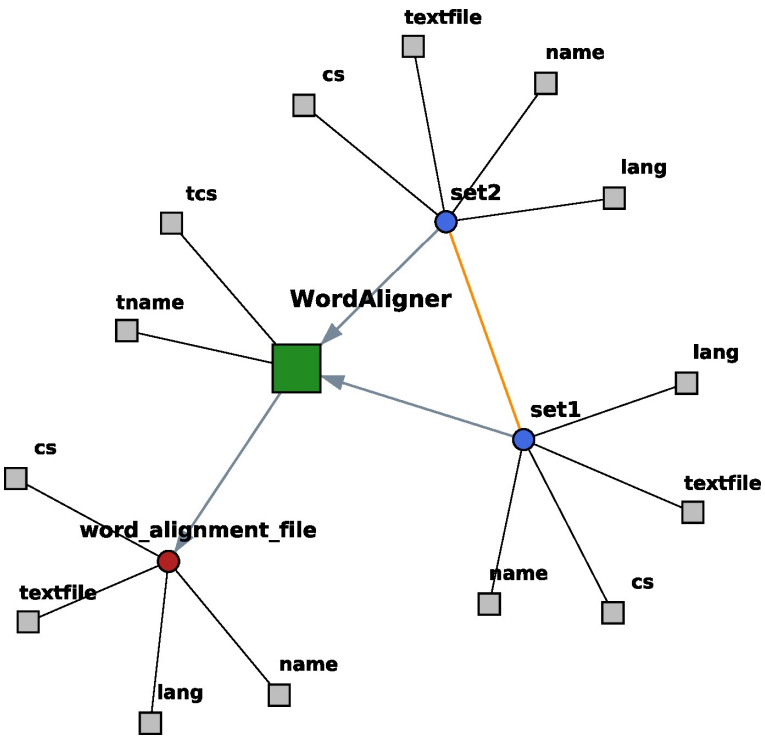


Figure 15: Interface for the word aligner.

4.2.2. Phrase Extractor

Input:

A text file in language L1 specifying word alignments.

A text file in language L1.

A text file in language L2.

Output:

A text file.

Common tool specification:

[PhraseExtractor : tcs : <(word_alignment_file, cs, text file, lang); (set1, cs, text file, lang); (set2, cs, text file, lang)> : <(phrase_table, cs, text file, lang)>]

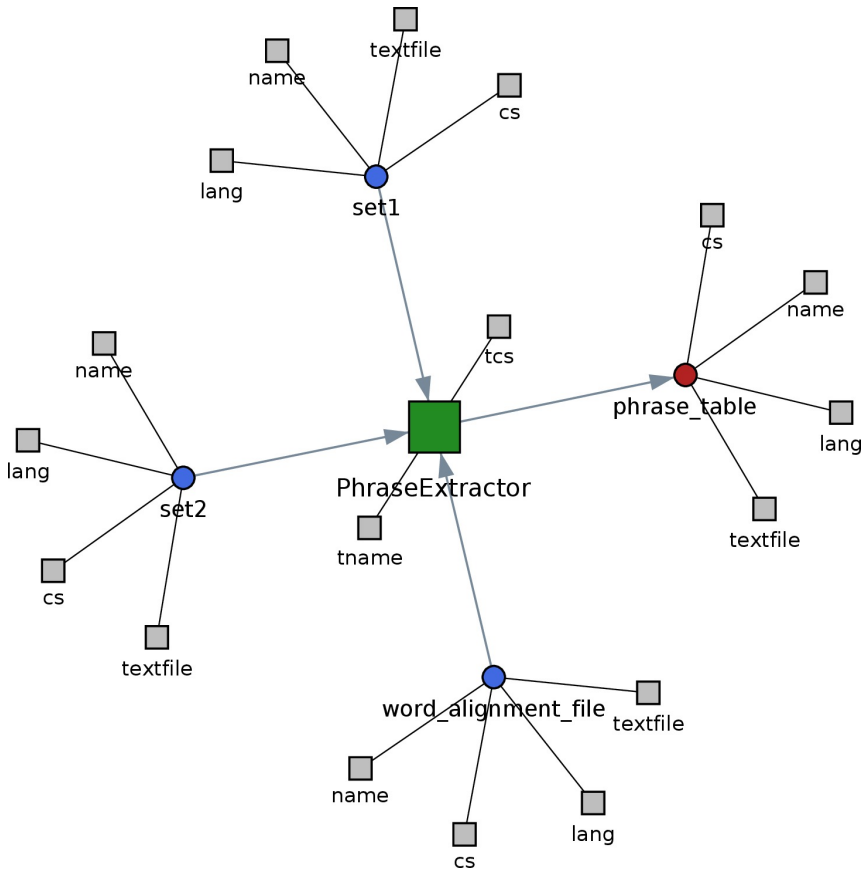


Figure 16: Interface for the phrase extractor.

4.2.3. Language Modeler

Input:

A text file in language L1

Output:

An ARPA file.

Common tool specification:

[LanguageModeler : tcs : <(doc, cs, text file, lang)> : <(LM, cs, ARPA file, lang)>]

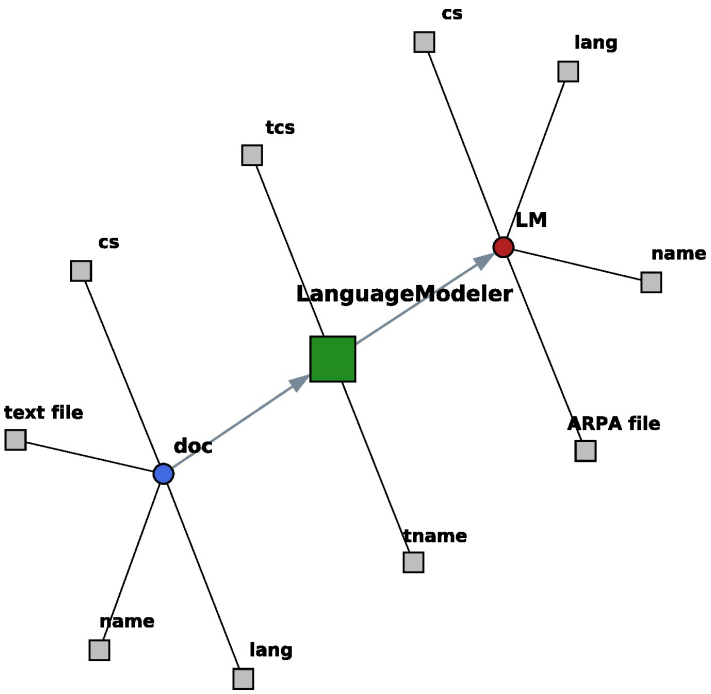


Figure 17: Interface for the language modeler.

4.2.4. Parameter Optimizer

Input:

A text file in language L1. (development set)

A text file in language L2. (development set)

A text file. (automatic evaluator specification)

A text file. (MT model specification)

Output:

A text file.

Common tool specification:

[ParameterOptimizer : tcs : <(devset1, cs, text file, lang); (devset2, cs, text file, lang); (evaluator, cs, text file, lang); (MTModel, cs, text file, lang)> : <(weights, cs, text file, lang)>]

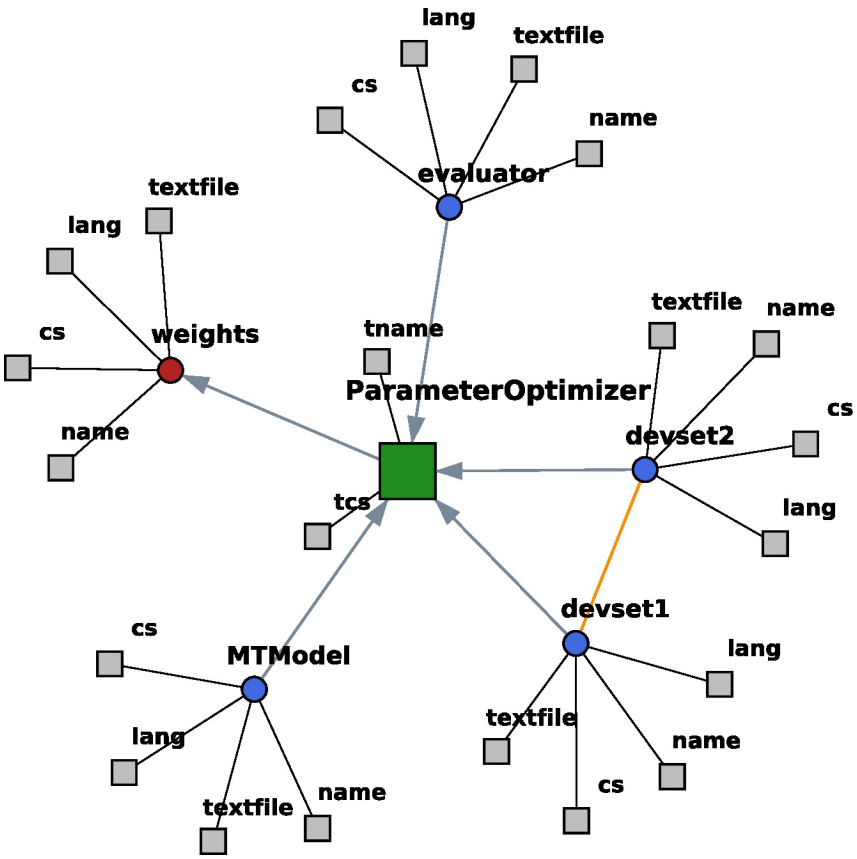


Figure 18: Interface for the parameter optimizer.

4.2.5. Decoder

Input:

A text file in language L1.

A text file. (MT model specification)

Output:

A text file in language L2.

Common tool specification:

[Decoder : tcs : <(source, cs, text file, lang); (MTModel, cs, text file, lang)> : <(translation, cs, text file, lang)>]

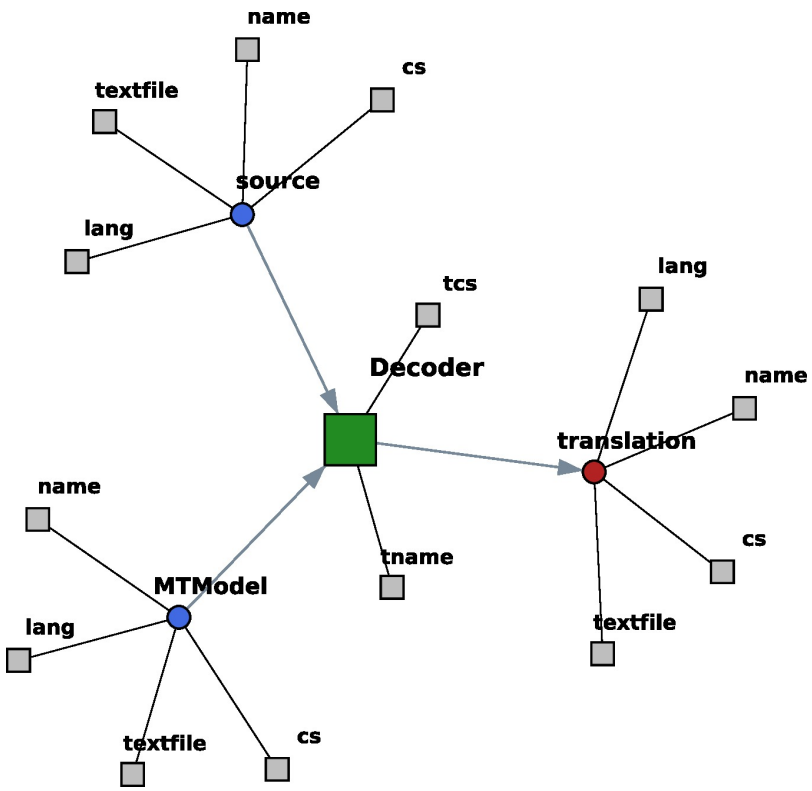


Figure 19: Interface for the decoder.

4.2.6. True Caser

Input:

A text file in language L1.

A text file. (Recaser model specification)

Output:

A text file in language L1.

Common tool specification:

[Recaser : tcs : <(lowercased, cs, text file, lang); (RecasingModel, cs, text file, lang)> : <(truecased, cs, text file, lang)>]

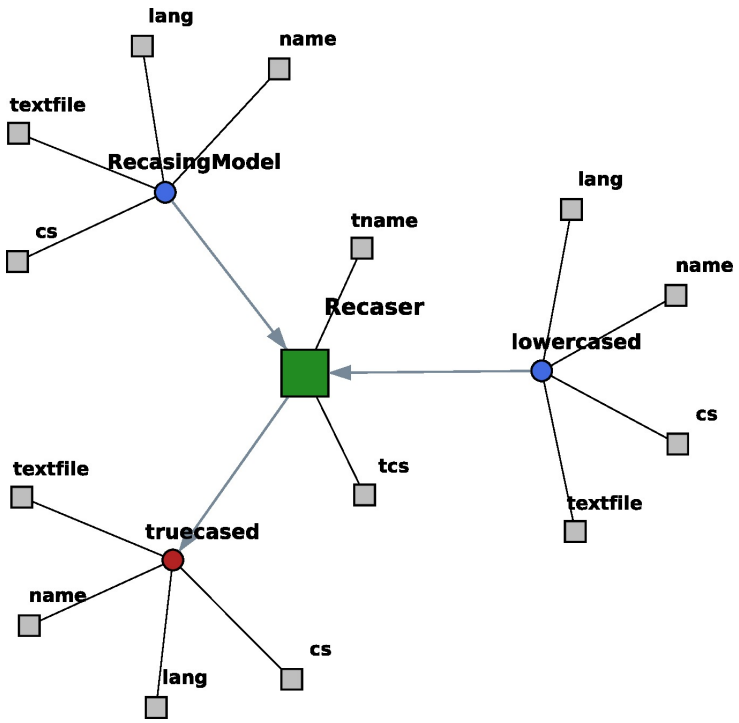


Figure 20: Interface for the true caser.

4.3. Evaluation

4.3.1. Automatic Evaluator

Input:

A text file in language L1.

A text file in language L1.

Output:

A text file.

Common tool specification:

[AutomaticEvaluator : tcs : <(translation, cs, text file, lang); (reference, cs, text file, lang)> : <(scores, cs, text file, lang)>]

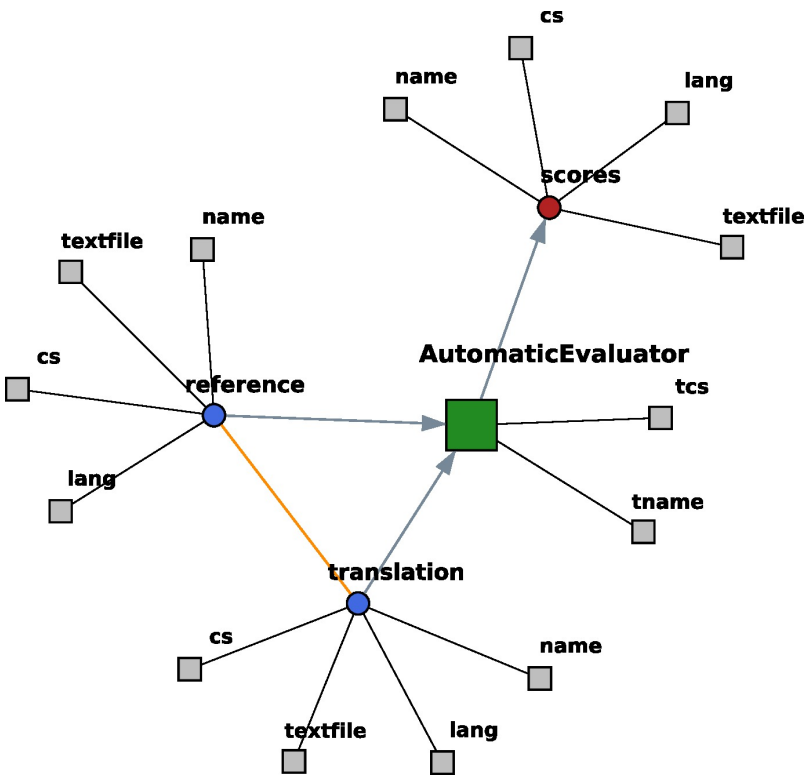


Figure 21: Interface for the automatic evaluator.

4.3.2. Automatic Error Analyzer

Input:

A text file in language L1.

A text file in language L1.

Output:

A text file and/or an HTML file.

Common tool specification:

[AutomaticErrorAnalyzer : tcs : <(translation, cs, text file, lang); (reference, cs, text file, lang)> : <(report, cs, text file, lang)>]

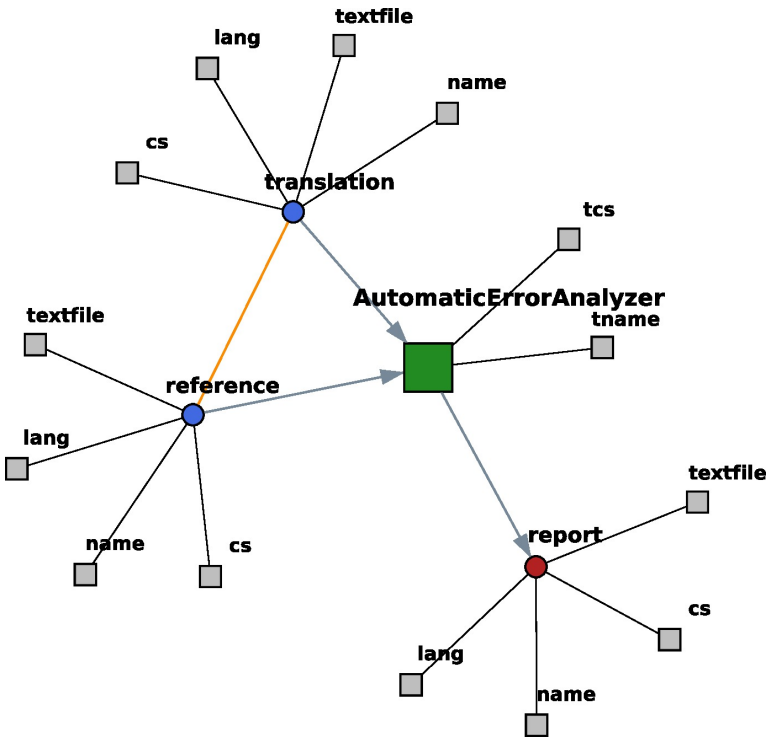


Figure 22: Interface for the automatic error analyzer.

4.3.3. Quality Estimator: QuEst

Input:

A text file in language L1.

A text file in language L2.

Intermediate Output:

A text file in language L1.

Output:

A text file.

Common tool specification:

[quality_estimator : tcs : <(sources, cs, text file, lang); (translations, cs, text file, lang)> : <(scores, cs, text file, lang)>]

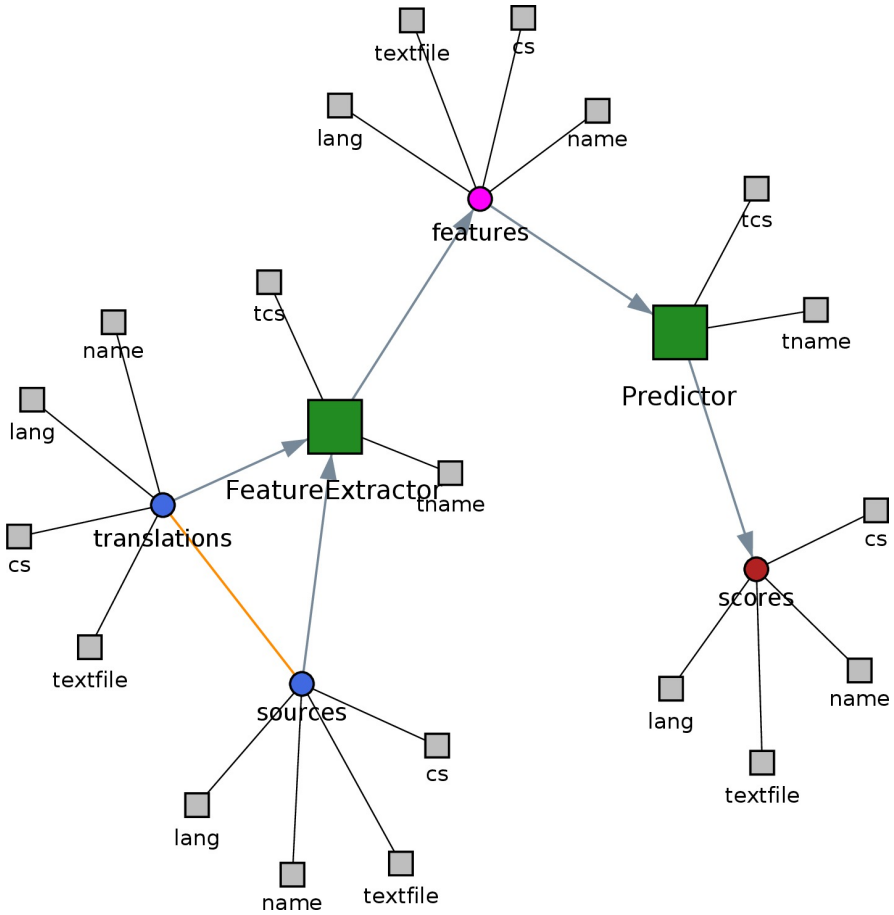


Figure 23: Interface for the quality estimator.

4.4. Translation Quality Assessment

4.4.1. Human MQM Evaluator

Input:

A text file in language L1.

A text file in language L2.

A text file.

Output:

A text file.

Common tool specification:

[HumanMQMEvaluator : tcs : <(source, cs, text file, lang); (translation, cs, text file, lang); (display, cs, text file, lang)> : <(TQA, cs, text file, lang)>]

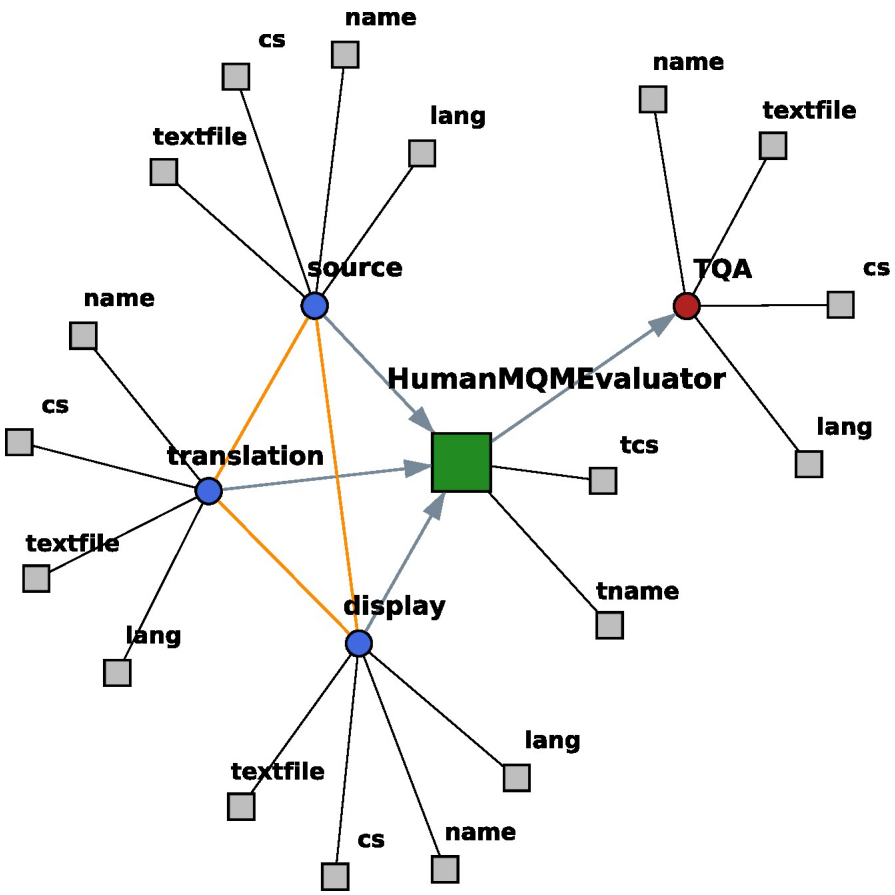


Figure 24: Interface for the human MQM evaluator.

4.4.2. MQM Evaluation Scorer

Input:

A text file. (TQA file)

A text file. (MQM specification)

Output:

A text file.

Common tool specification:

[MQMEvaluationScorer : tcs : <(TQA, cs, text file, lang); (MQMModel, cs, text file, lang)> : <(scores, cs, text file, lang)>]

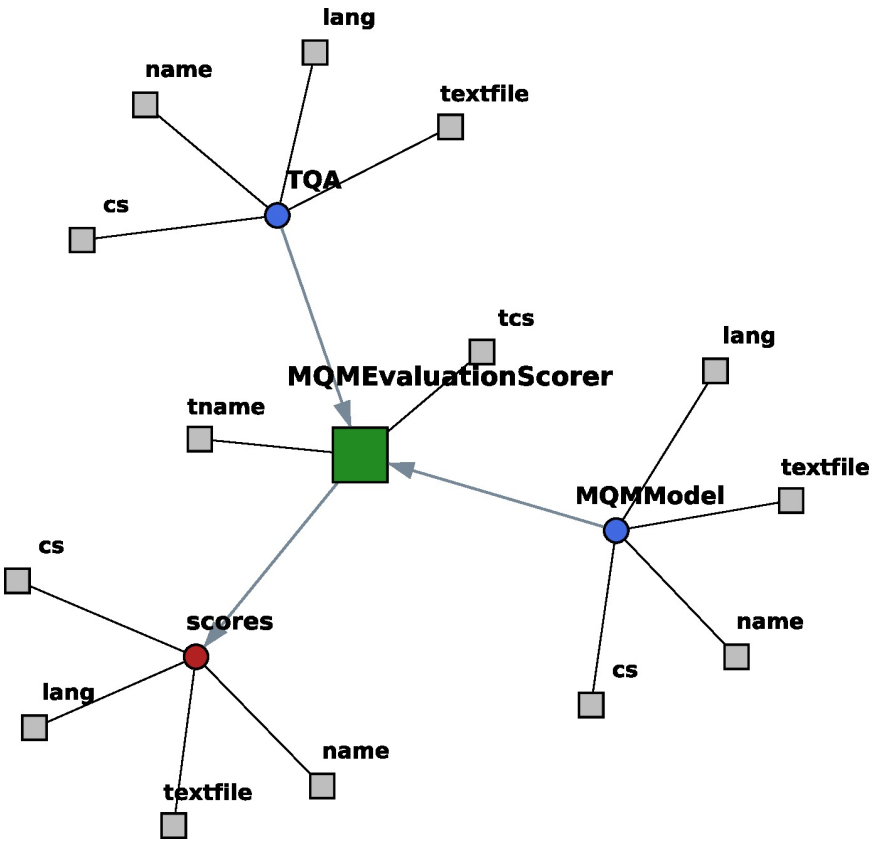


Figure 25: Interface for the MQM evaluation scorer.

5. Summary

The aim of this report is to define the interfaces for the tools involved in MT development and evaluation scenarios as included in the QTLP infrastructure. The report gives detailed specifications of the tools in the first main section and the interfaces involved in the following section, where in both sections, the tools are categorised into three categories: preparation, development, and evaluation. We also develop the human interfaces for quality assessment for the multidimensional quality metrics.

D3.2.1 allows the emergence of the QTLP infrastructure and helps the identification and acquisition of existing tools (D4.4.1), the integration of identified language processing tools (D3.3.1), their implementation (D3.4.1), and their testing (D3.5.1). QTLP infrastructure will facilitate the organization and running of the quality translation shared task (D5.2.1). We also provide human interfaces for translation quality assessment with the multidimensional quality metrics (D1.1.1).

Interface specifications allow a modular tool infrastructure, flexibly selecting among alternative implementations and enabling realistic expectations to be made at different sections of the QTLP information flow pipeline. Interface specifications support the QTLP infrastructure.

References

[1] Europarl parallel corpus, http://www.statmt.org/europarl/.

[2] Robert C. Moore. 2002. Fast and accurate sentence alignment of bilingual corpora. In AMTA 02: Proceedings of the 5th Conference of the Association for Machine Translation in the Americas on Machine Translation: From Research to Real Users, pages 135–144, London, UK. Springer-Verlag.

[3] Moses, statistical machine translation system, http://www.statmt.org/moses/.

[4] Chris Callison-Burch, Philipp Koehn, Christof Monz, Matt Post, Radu Soricut, and Lucia Specia. 2012. Findings of the 2012 workshop on statistical machine translation. In Proceedings of the Seventh Workshop on Statistical Machine Translation, pages 10–51, Montreal, Canada, June. Association for Computational Linguistics.

[5] Och, F. J. (2003). Minimum error rate training in statistical machine translation. In Proceedings of the 41st Annual Meeting on Association for Computational Linguistics Volume 1, pages 160–167, Sapporo Convention Center, Japan.

[6] Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The Stanford typed dependencies representation. In Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation, pp. 1-8.

[7] C. Michael Sperberg-McQueen, Lou Burnard, Guidelines for Electronic Text Encoding and Interchange (TEI P3), Text Encoding Initiative, Chicago, Illinois, April 1994.

[8] CoNLL-X Shared Task: Multi-lingual Dependency Parsing, http://ilk.uvt.nl/conll/.

[9] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz: Building a Large Annotated Corpus of English: The Penn Treebank, in Computational Linguistics, Volume 19, Number 2 (June 1993), pp. 313--330 (Special Issue on Using Large Corpora).

[10] 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.

[11] Dan Klein and Christopher D. Manning. 2003. Accurate Unlexicalized Parsing. Proceedings of the 41st Meeting of the Association for Computational Linguistics, pp. 423-430.

[12] Stephen Clark and James R. Curran (2007): Wide-Coverage Efficient Statistical Parsing with CCG and Log-Linear Models. Computational Linguistics, 33(4), 2007.

[14] Morfo - Czech Morphological Analyzer, http://ufal.mff.cuni.cz/morfo/#documentation

[15] Stanford lemmatizer, http://nlp.stanford.edu/software/corenlp.shtml

[16] mteval-v11b.pl automatic evaluator, http://www.statmt.org/wmt07/baseline.html

[17] METEOR, Automatic Machine Translation Evaluation System, Michael Denkowski, Alon Lavie, http://www.cs.cmu.edu/~alavie/METEOR/

[18] The ARPA-MIT LM Format, http://www.ee.ucla.edu/~weichu/htkbook/node243_ct.html

[19] SRILM - The SRI Language Modeling Toolkit, http://www.speech.sri.com/projects/srilm/

[20] KenLM Language Modeling Toolkit, http://kheafield.com/code/kenlm/

[21] CoNLL-X Shared Task: Dependency Parsing, http://nextens.uvt.nl/depparse-wiki/SharedTaskWebsite

[22] Maja Popović. 2011. “Hjerson: An Open Source Tool for Automatic Error Classification of Machine Translation Output”. The Prague Bulletin of Mathematical Linguistics No. 96. pp. 59--68, October.

[23] Daniel Zeman, Mark Fishel, Jan Berka, Ondrej Bojar. 2011. “Addicter: What is Wrong with My Translations?”. The Prague Bulletin of Mathematical Linguistics No. 96, pp. 79--88, October.

[24] GALA Webinar: QTLaunchPad presentation on Multidimensional Quality Metrics, Lucia Specia and Arle Lommel. Slides can be reached from here: http://www.gala-global.org/update-qtlaunchpad-project-ppt.

[25] Ergun Biçici and Deniz Yuret. RegMT system for machine translation, system combination, and evaluation. In Proceedings of the Sixth Workshop on Statistical Machine Translation, pages 323–329, Edinburgh, Scotland, July 2011. Association for Computational Linguistics.

[26] Chris Callison-Burch, Philipp Koehn, Christof Monz, and Omer F. Zaidan. Findings of the 2011 Workshop on Statistical Machine Translation. In Proceedings of the Sixth Workshop on Statistical Machine Translation, Edinburgh, England, July 2011. Association for Computational Linguistics.

[27] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In ACL ’02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, pages 311–318, Morristown, NJ, USA, 2001. Association for Computational Linguistics.

[28] QuEst - an open source tool for translation quality estimation, http://www.quest.dcs.shef.ac.uk/.

[29] Universal tag set conversion tool, https://code.google.com/p/universal-pos-tags/.

[30] Asiya - An Open Toolkit for Automatic Machine Translation (Meta-)Evaluation,

http://nlp.lsi.upc.edu/asiya/

[31] Philipp Koehn. 2010. Statistical Machine Translation. Cambridge University Press.

[32] Ergun Biçici, Declan Groves, and Josef van Genabith. Predicting sentence translation quality using extrinsic and language independent features. Machine Translation, 2013.

[33] Ergun Biçici. 2011. *The Regression Model of Machine Translation*. PhD Thesis, Koç University.

[34] Ergun Biçici. 2013. Feature decay algorithms for fast deployment of accurate statistical machine translation systems. In Proceedings of the Eighth Workshop on Statistical Machine Translation, Sofia, Bulgaria, August. Association for Computational Linguistics.

[35] OpenNLP tool NER: http://opennlp.apache.org/documentation/1.5.3/manual/opennlp.html#tools.namefind.recognition

[36] eSpeak text to speech: http://espeak.sourceforge.net/

[37] Ergun Biçici. 2013. Referential Translation Machines for Quality Estimation. In Proceedings of the Eighth Workshop on Statistical Machine Translation, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

[38] Antonio Toral, Sudip Kumar Naskar, Federico Gaspari, Declan Groves. 2012. "DELiC4MT: A Tool for Diagnostic MT Evaluation over User-defined Linguistic Phenomena". *The Prague Bulletin of Mathematical Linguistics* No 98, 2012, pp. 121-131., ISSN 0032-6585, DOI: 10.2478/v10108-012-0014-9 (also available as a web service from: http://registry.elda.org/services/301).