Akkadian Word Segmentation
Developers Documentation

Timo Homburg
Institute for Computer Science
Goethe University, Frankfurt, Germany
timo.homburg@gmx.de

December 26, 2018

Contents

Chapter 1

Introduction

This document is the developer documentation of a Master thesis project for the prupose of segmenting Akkadian cuneiform texts. It will give a short introduction on how to use this application to segment texts and how to interpret results and how to extend the application.

1.1 Purpose

The purpose of the application can be described by describing its components:

- CorpusImportModule: Imports Corporadata
- CorpusHandlingModule: Prepares Corpora for classification purposes
- DictionaryModule: Collection of gained corpora data
- EvaluationModule: Evaluates generated segmentations by comparing them to the original segmentations
- SegmentationModule: Implements algorithms to segment
- TransliterationModule: Assigns transliterations to an already segmented text
- TranscriptionModule: Convertes transliterations to transcriptions
- POSTaggingModule: Tries to POSTag already texts in transliteration or Unicode representation
- TranslationModule: Using the POSTaggingModule tries to translate a given text to a given target language
- GUIModule: Displays statistics and Comparion Views

This documentation will aim to explain every module and the corresponding classes and provide an idea for an developer on how to extend the program to fit the needs. Furthermore the intended naming scheme and resource files of the program shall be illustrated. It should also provide a guideline on how to use the program to classify different other languages than already implemented.

Chapter 2

Getting Started

In this chapter a brief introduction on how to start and work with the program shall be given. Before starting the program make sure a recent Java Virtual Machine Version 1.7 or higher is installed on your system.

2.1 How to use it

The program can be started using the following command:

```
java -jar wordseg.jar
```

In this case, the graphical user interface will be started.
Moreover the program offers the following options to work with it:

- Command Line interface for classifying data. This command line interface is currently not up-to-date.

  ```
  Main
  ```

- Creation of FeatureSets using FeatureSet Generators with the following class:

  ```
  ArffHandler
  ```

- Single execution of a classifier for Machine Learning to create models

  ```
  WekaMethods
  ```

2.1.1 Data structure

The program needs a certain data structure of folders to be started. The data structure is illustrated in the following figure:

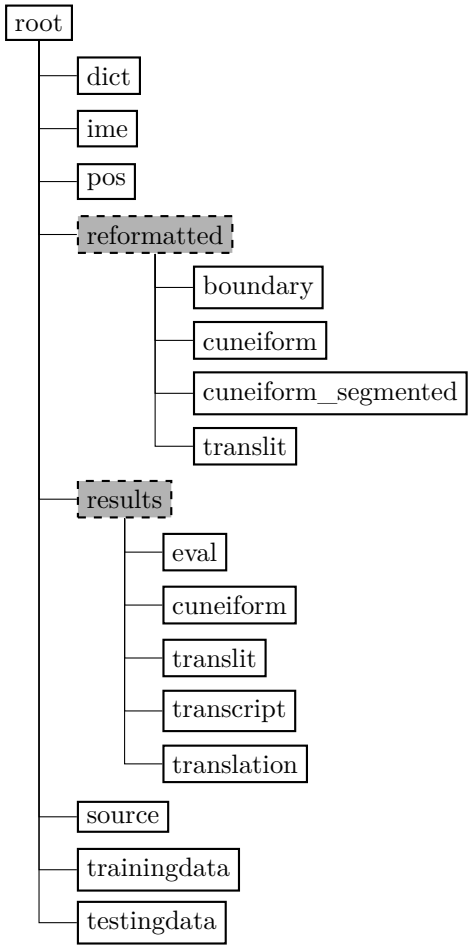


Figure 2.1: Data and folder structure

The naming conventions of the program are as follows and will be illustrated further in the corresponding chapters:

- Preformatted corpora stick to their originally given name and are moved to the folder reformatted
- Resultfiles have the following naming scheme:

  sourceFileName_TestMethod_ClassificationMethod_TransliterationMethod_result.txt
  Zum Beispiel:
  corpus_crossvalidation_minwcmatch_first_result.txt

- Statistics are named as follows:

sourceFileName_TestMethod_ClassificationMethod_TransliterationMethod_eval.txt
Zum Beispiel:
corpus_crossvalidation_minwcmatch_first_eval.txt

2.2 Navigation and Options

Options to start a classification shall be presented as follows:

Classification

A classication chain needs the following parameters to be started:

- ClassificationMethod
- TransliterationMethod
- TranslationMethod
- Trainingfile
- Testfile
- TestMethod - Crossvalidation, Percentage Split...
- CharType - Which language?
- Target language

Every parameter can be in need of other parameters. For example: A featureset is needed if the classification method is a Machine Learning classifier.

Input Method Engine

Data for input method engines can be created for the following implementations:

- Ibus[1]
- JIMF (Java Input Method Framework)[2]
- JQuery: Für die Eigenentwicklung Webime[3]
- SCIM[4]

Visualisation

The visualisation of the processing chain is described in chapter GUIModule. Every module contributing to the result of the will be presented with an own comparison and statistic view.

[1]Ibus: https://code.google.com/p/ibus/
[2]JIMF:
[3]WebIME: https://github.com/situx/webime
[4]SCIM: http://sourceforge.net/projects/scim/

Chapter 3

Structure

The structure of the program will shortly be outlined in this chapter. Modules mentioned in this package structured will later be presented in detail in the chapter Modules. The source code was designed to be able to cope with any language to be segmented. Another focus was to keep the code as extensible as possible.

3.1 Packages and Classes

The program was separated into the following packages:

- de.unifrankfurt.cs.acoli.akkad.dict
  - de.unifrankfurt.cs.acoli.akkad.dict.chars - Definition of Character classes for different languages
  - de.unifrankfurt.cs.acoli.akkad.dict.corpusimport - ImportHandler to import data from XML and ATF
  - de.unifrankfurt.cs.acoli.akkad.dict.dictHandler - DictHanler to manage words and characters extracted from corpora
  - de.unifrankfurt.cs.acoli.akkad.dict.importHandler - ImportHandler for previously exported dictionaries
  - de.unifrankfurt.cs.acoli.akkad.dict.pos - POSTagger
  - de.unifrankfurt.cs.acoli.akkad.dict.utils - Util classe for managing dictionaries
- de.unifrankfurt.cs.acoli.akkad.eval - Evaluationmetrics and statistics
- de.unifrankfurt.cs.acoli.akkad.main - Graphical user interface and command line tools
- de.unifrankfurt.cs.acoli.akkad.methods - Segmentation, transliteration, transcription and translation methods
- de.unifrankfurt.cs.acoli.akkad.dict.util - Enums to store options

3.2 Extensibility

To add another language the following steps need to be done:

- Define a character class and add its definition to the CharTypes Enums
- Define a DictHandler class
- Define a CorpusHandler class to handle imports of the corpus format

Upon creating support for a new language, all segmentation methods can be used on this new language instantly. If implemented correctly no further modifications are needed. As it concerns transliteration, transcription, POS-Tagging and translation, they have to be modified language dependently to fit the users needes.

Chapter 4

Modules

Modules provide the core functionality to the program. The presented modules work together to build a natural language processing chain illustrated in figure 4.

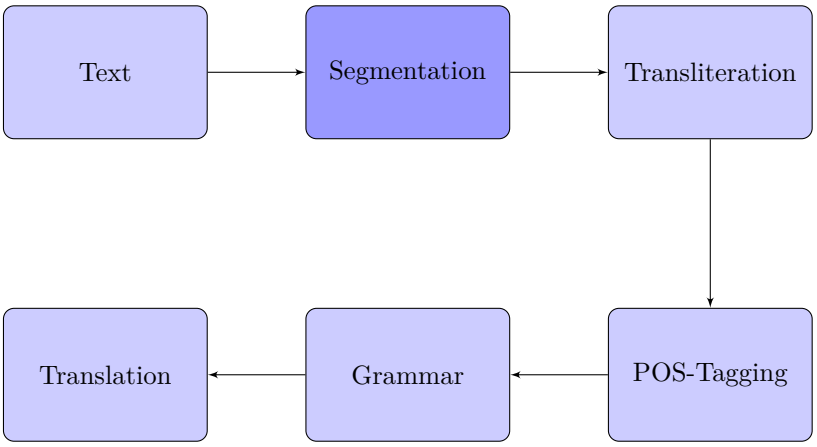


Figure 4.1: Natural Language Processing Pipeline as (partly) implemented in the program

In its current state the focus was to implement the segmentation part of the chain. The implementation of other parts has begun but very often can be considered as rudimentary at this state. More information about the current status will be provided in the corresponding subsections of the modules.

4.1 CorpusHandlingModule

The corpus handling module manages the import of a corpus, its transformation in to an annotated dictionary and a classification task performed on the corpus using a predefined classification method. In particular the following corpus modes are supported:

- Manual selection of a Trainingfile and a Testfile

- Crossvalidation on the corpus (Fold can be specified)
- Percentage Split on the corpus (Percentage can be specified) on a char or text basis
- Random Split on the corpus (Percentage and text or char based randomness can be specified)

If a mode is chosen, the corpus handling module proceeds as follows:

1. Split the corpus according to the needs of the method
2. Save splitted files for further use
3. Start a classification task

To implement a new corpus splitting method, this method needs to be registered in the corresponding Enum *TestMethod*. Next, the proposed method needs to be implemented in the class *CorpusHandlerAPI*, the class of which every CorpusHandlerModule inherits or a class inheriting *CorpusHandlerAPI*. Mostly splitting methods are not language dependent and can therefore be implemented in the class directly.

4.1.1 General Attributes

A CorpusHandlerAPI shall always convert the given texts to the reformatted version used by segmentation algorithms. This means it has to call an import handler to import text data and create a dictionary from the desired training part after the chosen splitting process and should over conversion methods to interact between various occuring formats:

- Conversion to boundary format
- Conversion from UTF8 to ASCII and vice versa
- Conversion from corpus data to intermediate format

4.1.2 CorpusImportModule

The corpus import module aims to import data from given corpora or dictionaries. To import a certain dictionary or corpus it is necessary to define an Importer class which has to inherit the abstract class *ImportHandler*. The required methods by implementing this class are as follows:

- reformatToASCIITranscription
- reformatToUnicodeTranscription

This is to provide support for transliterations using diacritic signs and ASCII transcriptions/transliterations replacing them with numbers. Results of the parsing process shall be imported into a DictHandler described in 4.2 to be further processed in the segmentation process. One should also be aware of the language to be parsed. If no language support classes have been developed in the DictionaryModule language processing cannot occur properly.

4.1.3 Special Import Needs

Some corpora are used to be exported as one corpus file, while many corpora versions are used to be shipped in files contained in several directories. To accomodate those needs the CorpusHandlingModule needs to call the CorpusImportModule many times to merge its results. It is also possible to merge a many-file corpus format into a one-file corpus format to simplify processing of corpora.

4.2 DictionaryModule

The dictionary module is the core data management module of the application. Any corpus being analyzed and transferred into the program will be internally converted into a dictionary which may or may not be exported. To define a dictionary it is necessary to define the language of the dictionary first. As until now it is only possible to include single language corpora. Any dictionary consists of a DictHandler, a managing class of the dictionary and a language or writing specific character class.

4.2.1 Character Classes

Character classes are multifunctional classes defining both characters and words as they appear in the corresponding language. They hereby consist of basic attributes described in the class *LangChar*. Those attributes are attributes any character of a language can include e.g.

- Relative/Absolute Occurance of the character in a corpus
- List of Transliterations
- List of Translations
- Preceding/Following Chars with occurances
- Encountered POSTags
- Character length

If on a character basis the character length describes the number of bytes used to represent this character in Unicode. This might as well vary in between the same language (e.g. Chinese) depending on the unicode block and has therefore be annotated on a per-char basis.
Character classes are dependent on an initial language configuration enum defined in *CharTypes*. Within this enum the following language specific parameters need to be configured upon creating a new language:

- Enum Identifier
- Language Name to display in the graphical user interface
- Locale String (Java if available, otherwise costumized String)
- Char Length in Unicode (if many apply the most common char length)
- List of StopChars before and after which a segmentation has to occur
- Regular Expression to define legal transliteration characters

Language Specific Character Classes

Language specific character classes like AkkadChar inherit LangChar. If a language uses a common writing system used by other languages it might be useful to include another layer of more generalized classes. For example the class CuneiChar works as the base class for the Cuneiform language classes AkkadianChar, SumerianChar and HittiteChar as all three languages are written in Cuneiform. Typical criteria on a language specific or writing specific level include:

- Grammar aspects such as: Determinatives and Sumerograms in Akkadian or Word Categories like Number, Name a.s.o. in Chinese
- Suffix, Prefix, Infix indicators

4.2.2 DictHandler

A DictHandler is a managing class for managing corpus data already organised in a dictionary form. It contains statistical data, the words of the dictionary and several mappings between the components. Every DictHandler class inherits the abstract class DictHandling which provides the following basic methods for a DictHandler:

- addWord/addFollowingWord/addPrecedingWord/addStopWord - Adding words with several functions to the dictionary
- calculateAvgWordLength - Calculates the average word length under consideration of the current char length
- matchWord/matchWordByTransliteration/matchWordByTranscription - Lookup methods for the dictionary
- parseDictFile - Imports an already preformatted dictionary
- toIME - Exports the dictionary to an IME representation as defined in the *ExportMethods* Enum

4.2.3 New Dictionary

To create a new dictionary by using an already existing or costumized CorpusImportModule the following implementation steps have to be done:

1. Create a character class in de.unifrankfurt.cs.acoli.akkad.dict.chars and let it inherit from LangChar
2. Implement potentially missing methods
3. Implement a new class inherting DictHandling or one of its subclasses

4.2.4 Adding External Resources

External Resources can be added to a dictionary to enrich it. Preferred resources would be translations, POSTagging information or statistics. To import translations from Websites, a WebCrawler Module has been implemented in the class *DictWebCrawler*. Importer for external resources can be found in the dictionary module. Those have to be manually started after parsing the dictionary. Examples are the classes *TranslationImportHandler* and *TranslationImportHandler2*.

4.3 EvaluationModule

The Evaluation module evaluates given segmentations by comparing it to the original segmentation. It therefore offers to implement several algorithms for this purpose. Results of an Evaluation are stored in an *EvalResult* object and can further be processed e.g. in the GUI module for graphs or tables. EvalResult objects are collected in an *EvalStatistics* object present in the *Evaluation* class used to manage the evaluation process. Algorithms will work either on a segmented transliteration or a segmented target text whereas the latter is preferred. Nevertheless the choice between the two options can be established in the initialization method of the class *Evaluation* handling evaluation methods. The configuration of which methods to use is handled using the *EvaluationMethod* Enum in which Evaluation methods are registered. To register an evaluation method the following items are required:

- Enum Identifier
- Log Identifier (for Evaluation Results in text form)
- Full Name Identifier
- The Evaluation Result Type

The Evaluation Result Type is configured in the *EvalResultType* Enum and may take one of the following values:

- FScore - A metric producing precision, recall and fscore
- OneScore - A metric producing exactly one score as a result
- TwoScore - A metric producing two results as a score

4.3.1 Export

Every evaluation process produces an export file save in the directory results/eval/. The result file includes the statistics of the given evaluation.

Listing 4.1: Evaluation File

```
==========Evaluating: results/translit/corpus_maxmatch_first_result.txt==========
Transliteration Evaluation
Matched: 19902.0
Missed: 31586.0
Exp. Correct Segmentations: 0.0
Total: 51488.0
Matches(%): 38.65366687383468
Misses(%): 61.34633312616532
Accuracy(%): 92.9017357727978
Precision(%):83.49555294512501
Recall(%): 100.0
F-Measure(%): 91.00553294617953
G-Measure(%): 91.37590105992116
TruePositive: 19902.0
FalsePositive: 3934.0
TrueNegative: 31586.0
FalseNegative: 0.0
```

4.4 SegmentationModule

The segmentation module handles the segmentation of a given text. For this purpose the module refers to a segmentation algorithm desscription in the ClassificationMethod Enum consisting of the following attributes:

- Name of the classification method
- Short description of the classification method
- Printing label of the classification method
- FeatureSet indicator
- A framework indicator defined by the Options Enum 4.11.2

4.4.1 Segmentation Method Collectors

Segmentation methods can be gathered in segmentation method collector classes which inherit a segmentation method type specific interface containing segmentation methods and the base abstract class *SegmentationMethods*. It might be useful to group segmentation methods by type in those classes. Implementation examples are given with the interface *DictMethodAPI* and the class *DictMethods*. Important methods in *SegmentationMethods* include:

- assignTransliteration - assigns transliterations by transliteration method an given dictionary
- fileToBoundaries - gets a boundary representation of the current input file for Machine Learning purposes

Apart from important methods the class contains FileReader and FileWriter objects needed to read and write the corresponding files to be classified.

4.4.2 Segmentation Method Collector Implementations

A segmentation method collector implementation consists of an init method and two methods at a time for one implementation of a segmentation method. The init method initializes the segmentation by preparing reader and writer objects and choosing desired parameters for a segmentation. It is called by one of the public segmentation methods and internally calls one of the private segmentation methods of the same name. For example: A call to public avgWordlength would call the init method using the parameter AVG as defined in the ClassificationMethod Enum to initialize the call of the private method avgWordlength for execution. Segmentation methods produce a segmentation results in Unicode characters, a transliteration result, a transcription result and a translation result. To make sure such results are produced on implementing a new segmentation method the segmentation result can be transformed to transliteration by using the method assignTransliteration of *SegmentationMethods*.

4.4.3 Segmentation Method Collectors for Machine Learning

If Machine Learning is approached in one or many of the segmentation methods of a collector the generation of the featuresets and the handling of word boundary files has to be considered in the init method. The structure of a segmentation method collector changes as follows:

- Check if a Training and Testsets have already been generated
- If not training and/or test set has been generated, use the selected FeatureSetGenerator 4.5.1 to do so

After having generated and/or checked for needed FeatureSets, Machine Learning algorithms accessed by using external frameworks are usually implemented in a framework specific class like *WekaMethods*. This class provides the execution of the Machine Learning algorithm using the specified Training and Testsets, if necessary preprocessing methods which may or may not be provided by the framework, saving of models and the interpretation of results by retransforming them back into a segmented text. For the two used Machine Learning Frameworks WEKA and MALLET, the following facts need to be considered:

- WEKA expects the so-called ARFF format in which it is recommended to use a string representation for word segmentation characteristica as ngrams are a popular feature to use
- If a preprocessing is needed, WEKA transforms FeatureSets into a StringToWordVector representation. This can not be done if a multi-instance classification is done like in a HMM case
- If preprocessed using a StringToWordVector, the Machine Learning files will be in a so-called Sparse ARFF format. It is therefore necessary to check the first column for class 1 as 0s will be ignored
- Further implementations are needed for different result formats
- MALLET provides a simple result format that won't change depending on the algorithm. A sparse format does not exist

4.5 FeatureSets

FeatureSets are needed for Machine Learning purposes and mostly need to be generated in advance to process a specific text using FeatureSetGenerators.

4.5.1 FeatureSet Generators

A FeatureSet Generator is a class in which an algorithm containing rules to build a specific FeatureSet are implemented. The algorithm receives a preformatted text and produces depending on its initial parameters a Trainingset including the class to be classified or a Testingset lacking the class to be classified.
FeatureSets are defined in the enum FeatureSets using the following parameters:

- Array of classes of the featureset

- Number of features
- Name of the featureset
- Char or Wordbased Featureset indicator

Once a Featureset is registered, a featureset generator class can be implemented to contain the actual code of creating the featureset out of a text. A FeatureSet Generator class consists of a FeatureSetManager linking to the generating methods and FeatureSet Generator Collector classes implementing the generation methods. A FeatureSetManager inherits from FeatureSetAPI providing it with the method getContext, a method which for every character in the original text according to the selected featureset produces a line for the desired FeatureSet.

4.5.2 Generating a FeatureSet

Generating a FeatureSet requires a managing class to read the source text file and to write the destined FeatureSet file. Depending on the desired file format for the chosen Machine Learning framework this might need different managing classes. For now the ArffHandler class handles the creation of .arff files used in WEKA and .mallet files used in the MALLET framework. Depending on an Options Parameter 4.11.2 the class decides which format to produce. Classes like ArffHandlers are usually used in Segmentation Collectors for Machine Learning to generate Featuresets on-the-fly if not available previous to classifying. A featureset is generated in the subfolder trainingdata using a unique name of the file to be classified. More about names provides the section Naming Conventions.

4.6 Transliteration Module

The Transliteration Module is responsible for assigning transliterations to an already presegmented text. It is therefore necessary for an algorithm to choose an appropriate transliteration among predefined criterias. Already implemented transliteration methods are:

- Choose Transliteration according to the first transliteration in the transliteration list for the current word/char as present in the dictionary (First Transliteration)
- Choose Transliteration according to the transliteration in the transliteration list for the current word/char as present in the dictionary with highest frequency in the corpus (MaxProb Transliteration)
- Choose a random transliteration out of the transliteration list (Random)

The algorithms are implemented in the package methods.transliterations, yet they are encapsulated in the corresponding DictHandler module where they are delegated to the actual Character class implementing the call. A transliteration method should therefore be registered in the corresponding transliteration enum and the aforementioned chain of classes should be implemented to register the new method. The actual implementation will be in the corresponding Character class.

4.7 Transcription Module

The transcription module aims to produce a transcription of a previously transliterated text. In Akkadian this is currently done by removing syllable separators and converting so-called sumerograms to the Akkadian writing forms. For every language trying to be processed in this program, a transcription ruleset can be generated by creating a class and/or method linked to TranscriptionMethods. In the future, a transcription method can be added in the *CharTypes* enum to get a fixed method of transliterting characters. It might also be useful to let the user pick an appropriate transcription method as many could exist in a language to choose from.

4.8 POSTagging Module

The POSTagging Module is not only a module but can be used as a standalone program on its own in a forked POSTagger project becoming available shortly. The POSTagger module aims to POSTag an already segmented and transliterated text to get information about the word types and to approach an evaluation of the given segmentation. The POSTagger is currently implemented in a simple rulebased manner. Rules can be defined using Regular Expressions and will be used by the postagger to classify texts. This is by far not an optimal solution and can not in all aspects be used in every language. Especially irregular languages would need an update of this module to a statistical approach. The rulebased POSTagging module consists of the following parts:

- POSDefinition Files
- Dictionary
- IME

4.8.1 POSDefinition File

A POSDefinition file is an XML file containing definitions of POSTags and regular expressions to detect them. To discover attributes of already recognised word types a second stage allows to process those words to filter those criterias. Listing 4.2 shows an example file of a POSDefinition:

Listing 4.2: POSDefinition File

```
<colors>
COLOR SECTION -> Colors for coloring the words according to word types are saved here
<tagcolor tag="DEFAULT" desc="default" matchorder="13" color="#C0C0C0"/>
<tagcolor tag="CARD" desc="number" matchorder="1" color="#FFC0CB"/>
<tagcolor tag="PREP" desc="preposition" matchorder="2" color="#00FFFF"/>
...
</colors>
<dependencies>
DEPENDENCIES for grammars can be defined here
<dependence dependee="PREP" depender="DET"/>
<dependence dependee="DET" depender="PRO"/>
...
</dependencies>
<groupconfigs>
```

```
GROUPS define already recognized word types to gain further information about them
<groupconfig tag="VV" desc="verb">
<!-- (la/li/le/lu for past tense or ni for plural)
(Vocal before first root)
(Vocal after first root)
(ta-,te- for Perfect) -->
<group desc="verb" tag="VV"
name="Case:  First Person Plural"
regex="^(ni|nu)-$" group="2" case="declination" value="FIRST_PLURAL"/>
<group desc="verb" tag="VV"
name="Tense: Past Tense"
regex="^(li|la|le|lu)-$" group="2"
case="tense" value="PAST" />
</groupconfig>
</groupconfigs>
<tags>
TAGS define word types and their corresponding Regular Expressions,
Transliteration and Translation values if appropriate
<tag desc="verb" name="VV"
equals="" regex ="^.*().*$" case="VERB" value="ta verb" />
<tag desc="verb" name="VV"
equals="" regex ="^.*().*$" case="VERB" value="te verb" />
<tag desc="number" name="CARD"
equals="" regex ="^()+$" cunei="disz" case="NUMBER" value="1"/>
<tag desc="pronoun" name="PRO"
cunei="a-na-ku-ma" equals   ="" regex=""
case="FIRST_SINGULAR_NOMINATIVE" value="I" />
<tag desc="nounoradj" name="NA" equals=""
regex="(.*(-sza))$" case="FIRST_GENITIVE_SINGULAR_FEMALE"/>
<tag desc="nounoradj" name="NA" equals=""
regex="(.*(-szi))$" case="FIRST_ACCUSATIVE_SINGULAR_FEMALE"/>
```

POSDefinition files are stored in the folder pos/ and are usually named after the language shortcut they are implementing. The language shortcut has to be defined in the corresponding CharTypes enum in the program.

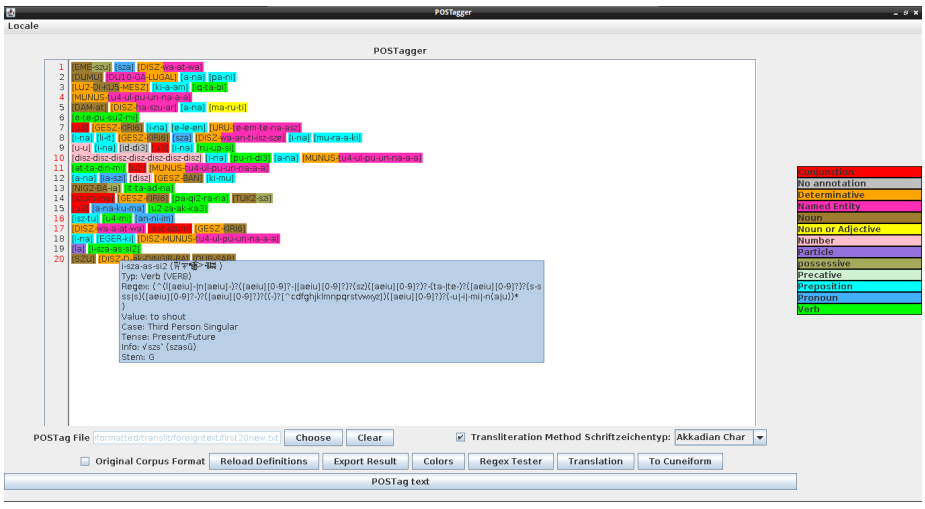


Figure 4.2: External POSTagging Example

4.8.2 Dictionary

If integrated into the main program the POSTagger can use the dictionaries the main program is using. If exported to an extra application those dictionaries have to be transferred as well.

4.8.3 IME

An unfinished approach let the POSTagging module use an Input Method Enging for Akkadian to input text that is on-the-fly classified by methods of POSTagging. This is a feature only present in the standalone version and ist not intended for the word segmentation tool. However an input method engine can be tested in one of the option selections. More about Input Method Engines is provided in the corresponding section.

4.9 Translation Module

The translation module can also be used in the standalone version of the POSTagging module and therefore has to applications. If used in the segmentation program it will try a plane 1:1 word translation if the word given is recognised. This is not yet very effective as word types and forms are not considered. In the external POSTagger program at first POSTagging is generated on the given text and then according to the given translation information through external resources the algorithm tries to produce an enhanced word-by-word translation considering word forms.

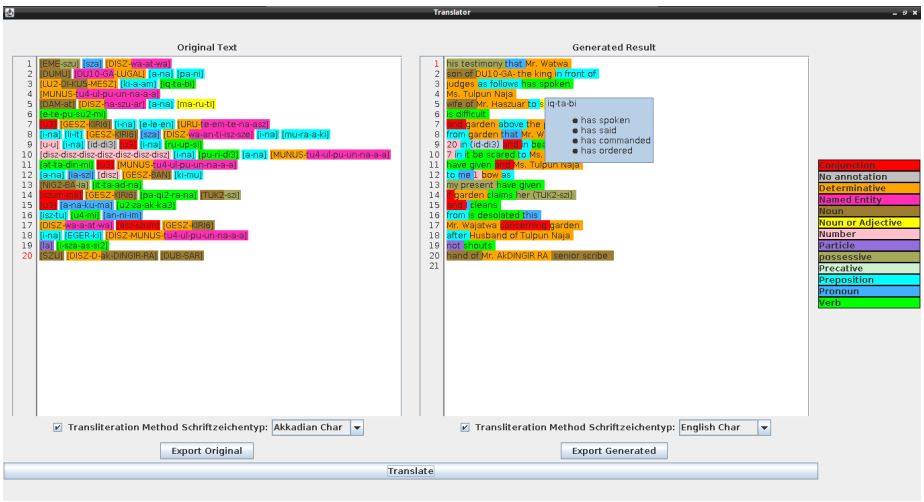


Figure 4.3: External Translation Example

4.9.1 Translator Classes

To perform a translation one has to define a source language and a target language.

4.10 GUIModule

The GUI module is defined in the package main.gui. It contains a starting screen, a statistic screen, several comparison screens and a few option screens.

4.10.1 Internationalization

The program supports Java Resource Bundles common in many Java Applications. However the class to load Resource Bundles needed to be rewritten to confirm to an UTF-8 Loading behavior of ResourceBundles. The implementation is located in the class **UTF-8 Bundle**. So far an English and German localization with English being the default have been implemented. Further support of languages is appreciated.

4.10.2 Title Screen

The title screen is where the user is able to choose the desired algorithm among other configurations (see Figure 4.4)

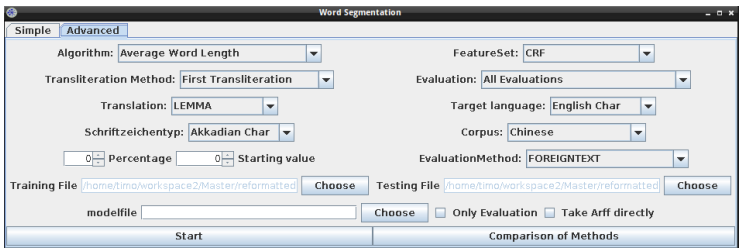


Figure 4.4: Advanced GUI Screen

The starting screen is defined in the class MainGUI whereas *MainGUI* inherits the *GUIFormat* class defining a unified layout for all windows of the program. Calls from this view will be received by the class *Main* which routes calls from the Graphical User Interface to the destined segmentation method classes described in 4.4 SegmentationModule. The implementation of the title screen has been refined to include a simple mode and an advanced mode. The simple mode was designed to offer simple algorithms which in a standard bundle of this application should work out-of-the-box and in a reasonable time. This makes the view a suitable one for presentations of the program. The advanced module was designed to offer all possibilities of the program as well as to begin Machine Learning training and similar longterm projects.

4.10.3 Comparison Screen

Comparison Screens are needed if comparisons between an original text and a segmented text should be presented to the user. The user should be able to see differences clearly and be able to find weaknesses of the implementations easily.
Comparison screens are used to

- Compare Segmentations
- Compare Transliterations

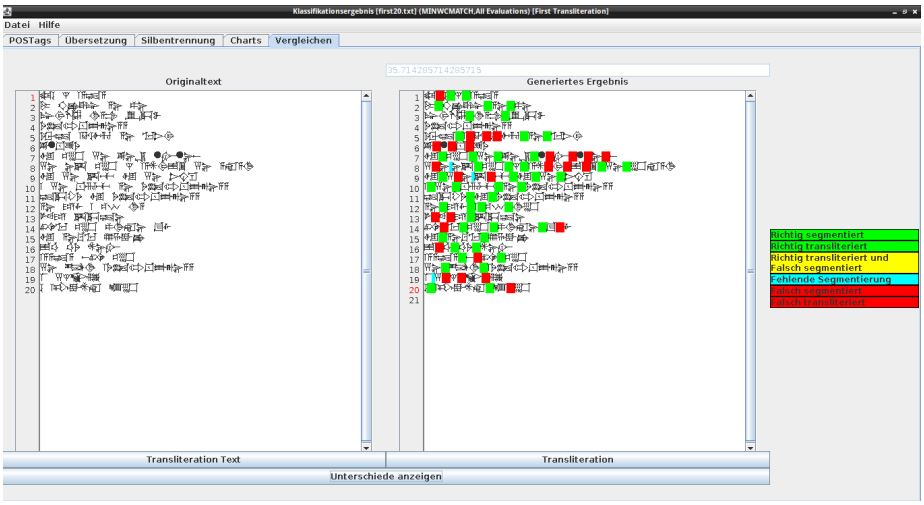


Figure 4.5: Segmentation Comparison Screen

- Compare Transliterations to Transcriptions
- Compare Original Texts to Translations

Every Comparison Screen inherits the abstract class *CompGUI*. This class implements the basic view of the window. A comparison window consists of two buttons, a legend and two JToolTipAreas, a costumized class to fit the needs of comparisons better. To highlight possible differences between segmentations JToolTipAreas will be colored using a highlighter. They may as well contain additional information depending on the screen.

Implementation Advice

The implemented highlighting algorithms expect a whitespace after every word of text. This is implementation specific and also requires a whitespace in front of every newline. If a whitespace in front of a newline is missing the marking algorithm will produce malicious results.

In some screens the legend should also be used as a tool to costumize the highlights given in the textareas. A map from whitespace to color should remember which spaces have been marked in which color to clear those spaces and redraw them if chosen in the legend respectively.

JToolTipArea

A JToolTipArea is a modified JEditorPane of Java Swing. It essentially contains the implementation of a Tooltip extension making it possible to display highlighting Tooltips over words. It also provides the option to directly copy a clicked word to the clipboard in order to conveniently copy selected texts. JToolTipAreas in itself also provide the data management for the given texts and annotations they contain and access methods respectively. JToolTipAreas also incorporate a sidebar in which the linenumbers are counted. In the case of

the POSTagging module this sidebar also functions as a sentence end indicator using colored numbers and as a method to select sentences.

4.10.4 Statistics Screen

The statistics screen gathers information provided by the classes described in 4.3. Information is usually represented in JTables and a graph is represented using the library JFreeChart.

Result Tables

Result Tables use modified TableRenderer structure which can be found in the untility pacage of the gui package. Inserted data from the evaluation classes will be considered and colored in a scale ranging from red to green. The coloring is performed on a scale from 0% (red) to 100% (green). Depending on the measure being used it might be suitable to create different TableRenderers to fit the needs to different measures.

Graph

The graph of the statistics screen was created using JFreeChart. In includes the Accuracy, Recall, Precision, F-Score and G-Scores of the different measures. If a measure does not contain the aforementioned measure it will be represented using the value 0 in all of them. It is therefore not a good idea to put single value measures in the graph as there is no room for comparison to an F-Score measure. However another JFreeChart graph could be used to incorporate single value measure in the future. To create a JFreeChart XYSeries of data have to be generated as an input format to the JFreeChart library. Result data from EvaluationResults have therefore to be processed further.

4.11 Utilities

In this chapter used libraries, utility classes and further applications of the programm shall be illustrated.

4.11.1 Libraries

The program uses the following libraries for the following purposes:

4.11.2 Options

The options enum represents options that are needed throughout the program. Options may for example be the following facts:

- Framework Names
- Training/Testingset Indicators

Library	License	Usage
Abego Treelayout [1]	BSD License	Treelayout classes to display syntactic trees
Apache Xerces[2]	Apache License 2.0	XML Parser
HMMWeka[3]	GPL	HMM Classifier as WEKA Plugin
Jericho HTML Parser[4]	LGPL	HTML Parser
JFreeChart[5]	LGPL	Graph library
LibSVM[6]	BSD License	SVM Classifier in WEKA Classifier
MALLET[7]	CPL 1.0	MALLET CRF Classifier
Pig Latin IME Java [8]	Creative Commons License	Creation of an Input Method Engine
Regular expression Tester [9]	No restriction	Tester for Regular Expressions
SimpleNLG [10]	Mozilla Public License 1.1	Verb Generator
TextLineNumber [11]	No restriction	LineNumber Panel
WEKA[12]	GPL	kNN, kMeans, Logistic, MultiLayerPerceptron, NaiveBayes Classifier

Table 4.1: Libraries used in the program

Chapter 5

Resources and Data

This chapter is about the several resources and data formats used in the program as well as about its naming conventions to identify corresponding files.

5.1 Naming Conventions

Names of corpora depend on the names given to the corpora files when first given into the program. During the processing of corpus files the following files using the following naming conventions might be created:

5.1.1 Preprocessing

Corpus files will be preprocessed in various was in the subfolder reformatted consisting of

- Transliteration (translit/)
- Boundary Representation (boundary/)
- Segmented Target Text (cuneiform_segmented/)
- Unsegmented Target Text (cuneiform/)

Depending on which corpus segmentation method of the CorpusHandling Module was chosen several subfolders will be contained in the folders mentioned above:

- (Foreigntext/) - No special treatment of the text
- (Crossvalidation/) - Corpus text is splitted according to crossvalidation rules
- (Percentage/) - Corpus has been splitted according to a percentage split rule
- (RandomSample/) - A random sample of a corpus has been chosen

Files created in the above mentioned dictionaries will have the following name conventions:

- Name of the original file in source/
- If Crossvalidation it will append a suffix _(NUMBER Of Split)
- If Percentage Split it will append a suffix _t(PERCENT)
- If Random it will append a suffix _t(PERCENT)

It is therefore feasible for the program to access files according to the corpus split method being selected.

5.1.2 FeatureSets

Using preprocessed data the program is able to create FeatureSets needed for Machine Learning. Machine Learning Data will be stored in the following folders:

- trainingdata/ for Trainingsets
- testdata/ for Testsets
- trainingsdata/model for generated model files

The naming conventions for FeatureSets when generated by the program are as follows:

- Name of the reformatted file
- _Name of the featureset as defined in the enum *FeatureSets*
- File extensions according to the Machine Learning Framework

The program can therefore access the desired files by concatenating the source file name, the preprocessing step, the featureset name and the Machine Learning framework.

Model files

Model files represent already trained data and have therefore already been processed by a Machine Learning algorithm. The naming scheme therefore has to incorporate the Machine Learning algorithm to uniquely identify the model file.

- Name of the reformatted file
- Name of the algorithm used for training
- _Name of the featureset as defined in the enum *FeatureSets*
- File extensions according to the Machine Learning Framework

5.1.3 Results

Results of the algorithm have to be saved in the folder results/. Within those folders various results can occur:

- Target language results (cuneiform/)
- Evaluation statistics (eval/)
- Transliteration results (translit/)
- Transcription results (transcript/)
- Translation results (translartion/)

A result file consists of all the elements necessary to create the result file namely:

- Name of the reformatted file
- _Segmentation Method
- Optional: _Name of the featureset as defined in the enum *FeatureSets*
- _Transliteration Method
- Suffix _result

5.2 Corpora Data

Resources consist of corpora data stored in different formats and are usually stored in the subfolder source. Names of the corpora data can be freely choosen but will become possible unique identifiers for further classifications. Examples of corpora data can be the ATF-Format for cuneiform:

Listing 5.1: ATF-Format as Corpus Format

```
&P388494 = AASOR 16, 015
#atf: lang akk
@tablet
@obverse
1. _eme_−szu sza {disz}wa−at−wa
2. _dumu du10−ga−lugal_ a−na pa−ni
3. {lu2}_di−ku5_{mesz} ki−a−am iq−ta−bi
4. {munus}tu4−ul−pu−un−na−a−a
5. _dam_−at {disz}ha−szu−ar a−na ma−ru−ti
6. e−te−pu−us−su2−mi
7. u3 {gesz}_kiri6_ i−na e−le−en {uru}te−em−<te>−na−asz
8. i−na li−it {gesz}_kiri6_ sza {disz}wa−an−ti−isz−sze i−na mu−ra−a−ki
9. 2(u) i−na szi2−id−di3 u3 i−na ru−up−szi#
10. 7(disz) i−na pu−ri−di3 a−na {munus}tu4−ul−pu−un−na−a−a
11. at−ta−din−mi u3 {munus}tu4−ul−pu−un−[na]−a−a
12. a−na ia−szi 1(disz) {gesz}_ban_ ki−mu
13. _nig2−ba_−ia it−ta−ad−na
14. szum−ma {gesz}_kiri6_ pa−qi2−ra−na _tuk2_−szi
15. u3 a−na−ku−ma u2−za−ak−ka3
16. isz−tu u4−mi an−ni−im
17. {disz}wa#−a−at−wa asz−szum {gesz}_kiri6_
18. i−[na] _eger_−ki {disz}{munus}tu4−ul−pu−un−na−a−a
19. la i−sza−as−si2
```

5.3 Intermediate Format

The intermediate format is a special format used for segmentation purposes. The transliteration of a corpus is represented in syllables separated by the minus character and words enclosed in brackets separated by whitespaces. This format is language independent as every language should be able to be transformed in a format of this kind.
Language dependent formats like

- Segmented or unsegmented text of the target language
- Boundary representation of the text

can be generated from the intermediate format.

Listing 5.2: Intermediate Format

```
&P388494 = AASOR 16, 015
#atf: lang akk
@tablet
@obverse
1. _eme_−szu sza {disz}wa−at−wa
2. _dumu du10−ga−lugal_ a−na pa−ni
3. {lu2}_di−ku5_{mesz} ki−a−am iq−ta−bi
4. {munus}tu4−ul−pu−un−na−a−a
5. _dam_−at {disz}ha−szu−ar a−na ma−ru−ti

is transformed to the intermediate format:

[eme−szu] [sza] [disz−wa−at−wa]
[dumu] [du10−ga−lugal] [a−na] [pa−ni]
[lu2−di−ku5−mesz] [ki−a−am] [iq−ta−bi]
[munus−tu4−ul−pu−un−na−a−a]
[dam−at] [disz−ha−szu−ar] [a−na] [ma−ru−ti]
```

5.4 Boundary Format

The boundary format is a representation used for Machine Learning. The given text in intermediate format is hereby transformed using the following rules:

- If the current syllable does not end a word write *0,*
- If the current syllable ends a word write *1,*
- Transfer newlines from the text into the new text

The result will be a boundary representation representing the necessary information of where a segmentation took place in the classes 0 and 1 for segmenting and not segmenting.

Listing 5.3: Boundary Representation

```
[eme−szu] [sza] [disz−wa−at−wa]
[dumu] [du10−ga−lugal] [a−na] [pa−ni]
[lu2−di−ku5−mesz] [ki−a−am] [iq−ta−bi]
[munus−tu4−ul−pu−un−na−a−a]
[dam−at] [disz−ha−szu−ar] [a−na] [ma−ru−ti]
```

[e−te−pu−us−su2−mi]

is transformed to the boundary representation:

0,1,1,0,0,0,1,
1,0,0,1,0,1,0,1,
0,0,0,1,0,0,1,0,0,1,
0,0,0,0,0,0,0,1,
0,1,0,0,0,1,0,1,0,0,1,

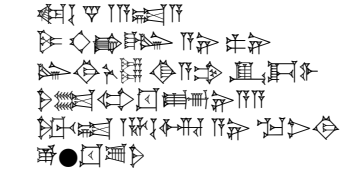
5.5 Target Language Formats

Target language formats represent text in the target language on which a segmentation should occur. Target language formats often require a manual or automated conversion process from the intermediate format using a list. Those resources can either be generated from corpora data directly or are delivered in the application root itself. For Akkadian, the file akkadian.xml represents such a table file.

Listing 5.4: Target Language Representation

[eme−szu] [sza] [disz−wa−at−wa]
[dumu] [du10−ga−lugal] [a−na] [pa−ni]
[lu2−di−ku5−mesz] [ki−a−am] [iq−ta−bi]
[munus−tu4−ul−pu−un−na−a−a]
[dam−at] [disz−ha−szu−ar] [a−na] [ma−ru−ti]
[e−te−pu−us−su2−mi]

is transferred to the target language representation:



5.6 Dictionary Resources

Dictionary resources are located in the subfolder dict/ of the application. Dictionary resources are hereby defined as dictionaries created by the application itself. Those dictionaries can be considered as an import file format for corpora not requiring any resource files to be manually processed. They might as well be an XML solution for other programs.
Dictionary resources are separated in _map files and _dict files. While map files only contain character mappings of a corpus, dict mappings can contain words and single characters if occured as words. Both files will be processed by the program according to the language enum. If the current language present is Akkadian the program will therefore search for the file dicts/Akkadian_map.xml and dicts/Akkadian_dict.xml.

5.6.1 Content

Dictionaries contain character to unicode mappings, characters statistics, transliterations, translations and POSTags as well as following and preceding words/chars with their corresponding statistics.

Listing 5.5: Dictionary Contents

```
<dictentry logogram="𓀀𓀁𓀂" determinative="false" logograph="false" phonogram="false"
relativeOccurance="1.2692656391659113E−4"
absoluteOccurance="7.0" begin="0.0" middle="0.0" end="0.0" single="false">
<transliteration begin="0.0" middle="0.0" end="0.0" single="0.0" isWord="false"
absoluteOccurance="1"
relativeOccurance="1.813236627379873E−5">a−a−ta</transliteration>
<following absoluteOccurance="2.0" absboundary="0.0">𓀃𓀄𓀅</following>
<following absoluteOccurance="1.0" absboundary="0.0">𓀆𓀇𓀈</following>
<following absoluteOccurance="1.0" absboundary="0.0">𓀉𓀊𓀋𓀌</following>
<following absoluteOccurance="1.0" absboundary="0.0">𓀍𓀎𓀏𓀐</following>
<following absoluteOccurance="1.0" absboundary="0.0">𓀑𓀒</following>
𓀓𓀔𓀕</dictentry>
```

5.7 Exports

The program features several exports from corpora data which should be highlighted in this section. The following exports opportunities exist:

- Input Method Engine Exports in folder (ime/):
  - Ibus Input Method Engine (ibus_header and ibus_footer files needed to process)
  - JQuery (jquery_header and jquery_footer files needed to process)
  - SCIM (scim_header and scim_footer files needed to process)
- Anki Export:

5.8 Results

Results are stored in the subdirectory results/ and contain results for different purposes:

5.8.1 Boundaries

Boundary results are usually not stored in the program. Instead boundary results are retransformed into a target language or transliteration representation to simplify the evaluation process. The retransformation works as follows:

- According to the Machine Learning result try to parse the classification result per char
- Take a copy of the unsegmented target language text and on-the-fly enter whitespaces where the classification signifies

- Assign a transliteration to the result and continue in the natural language processing pipeline

In the end the result of boundaries becomes obsolete and is not saved in one of the subdirectories of results/.

5.8.2 Evaluation

For every evaluation taken place an evaluation file is saved in results/eval.

5.8.3 Target Language

A target language for the classification can be chosen in the application and will determine if the classification is done with e.g. an Akkadian, Hittite or Sumerian corpus. In fact, additional target languages can be defined in the program and can be used if corpora resources are available in the source/ directory.

5.8.4 Transliteration

Transliteration files are saved in folder reformatted/translit. According to the transliteration method chosen and the configuration of the classification they will be placed in a subdirectory of transliteration according to the split of the corpus.

5.8.5 Transcription

Transcriptions are generated from the given transliterations by removing sign related information and by simplifying some transliteration aspects. They are stored in the same fashion as transliterations under the subdirectory reformatted/transcriptions.

Chapter 6

Useful Future Improvements

6.1 SegmentationModule

The segmentation module can be improved by adding more algorithms and by creating a chain of algorithms to process a certain text. Algorithms could be combined or chained to receive better results or even chained with themselves in a learning process. Optional parameters of several algorithms should be made available to the user in the user interface to guarantee more control over the actual classification process. The number of featuresets could be increased and optimized for specific languages to gain better results.

6.2 TransliterationModule

The transliteration module needs to implement transliteration methods based on current state of the art methods. Perhaps methods used in the segmentation module can be reused to assign transliterations as well. Transliteration methods should be tested on already segmented texts for performance to gain better results.

6.3 POSTagging Module

The POSTagging module for Akkadian is already on a good way. It should be researched if a rudimentary rulebased POSTagging approach can satisfy languages like Akkadian in a way that can cover most of the common word types. For remaining parts in can be useful to assign word types statistically or to rely on an machine learning approach to learn POSTag rules. The POSTagging Module guarantees an easy implementation of both new modules and extensions. Consequently a POSTagging approach can also be done on the target language to remove the transliteration part from the natural language processing pipeline. This will remove a potential source of fault and guarantee a better result in the end.

6.4 TranslationModule

The translation module highly depends on dictionary resources being available. For Akkadian this is the major obstacle to overcome. Using a dictionary and a word type extension liek the POSTagging module it is possible to identify words more easily and translate them word-by-word to different languages. Foundations of a translation module have been implemented using the given Translator classes. However, context representations have only sparsly been addressed by the translators yet. The implementation and usage of dependency grammars and other types of context analysis has to be implemented language dependent to guarantee a good translation.

6.5 Input Method Engines

The included input method engine can be improved in various ways:

- Include a fuzzy search for correcting typing mistakes
- Context-aware typing
- Switching of languages on-the-fly
- Managing of fonts depending on the language
- Usability improvements
- Integration of a drawing module for characters

Input method engines should also be developed for more platforms, the most important being Windows using the Windows Input Method Framework.

6.6 Exports

Exports can be important to transfer data to other applications that may help to improve the situation of the designated languages in general. It is therefore important to research and apply further uses of the gathered data and write corresponding export modules to help.

6.7 GUIModule

The GUIModule can be improved in various ways as follows:

- Tidy up statistics module and improve the graph representation
- Export statistics to other formats
- Usability improvements