

# **The Durm German Lemmatizer**

Praharshana Perera and René Witte

Universität Karlsruhe

Institut für Programmstrukturen und Datenorganisation (IPD)

Karlsruhe, Germany

witte@ipd.uka.de

May 28, 2006



# Contents

|          |                                       |           |
|----------|---------------------------------------|-----------|
| <b>1</b> | <b>The Durm Lemmatizer</b>            | <b>5</b>  |
| 1.1      | Overview                              | 5         |
| 1.2      | Setup                                 | 6         |
| <b>2</b> | <b>German Case Tagger</b>             | <b>9</b>  |
| 2.1      | Overview                              | 9         |
| 2.2      | Usage                                 | 9         |
| 2.2.1    | Initialization Paramaters             | 9         |
| 2.2.2    | Runtime parameters                    | 9         |
| 2.2.3    | Output Annotations                    | 10        |
| 2.3      | Implementation notes                  | 10        |
| 2.3.1    | Probability Files                     | 10        |
| <b>3</b> | <b>German POS-based Number Tagger</b> | <b>13</b> |
| 3.1      | Overview                              | 13        |
| 3.2      | Usage                                 | 13        |
| 3.2.1    | Runtime parameters                    | 13        |
| 3.2.2    | Output Annotations                    | 13        |
| 3.3      | Implementation notes                  | 13        |
| <b>4</b> | <b>German Morphological Analyzer</b>  | <b>15</b> |
| 4.1      | Overview                              | 15        |
| 4.2      | Usage                                 | 15        |
| 4.2.1    | Runtime parameters                    | 15        |
| 4.2.2    | Output Annotations                    | 15        |
| 4.3      | Implementation notes                  | 16        |
| <b>5</b> | <b>German Lemmatizer</b>              | <b>17</b> |
| 5.1      | Overview                              | 17        |
| 5.2      | Durm Lexicon                          | 17        |
| 5.2.1    | Evolving the Lexicon                  | 18        |
| 5.2.2    | Manual Correction of Entries          | 19        |
| 5.3      | Usage                                 | 19        |
| 5.3.1    | Runtime parameters                    | 19        |
| 5.3.2    | Output Annotations                    | 19        |
| 5.4      | Implementation notes                  | 19        |

## About this document

This document contains documentation for the *Durm* German Lemmatization system. You can get the latest version from <http://www.ipd.uka.de/~durm/tm/lemma/>.

## **Acknowledgments**

Development of the German Lemmatizer has been supported by the German research foundation (DFG) within the project "Entstehungswissen" (LO296/18-1). The TIGER Treebank (Version 2) has been used for training and evaluation of the Case Tagger.

# 1 The Durm Lemmatizer

The Durm Lemmatization System performs morphological analysis and lemmatization for German nouns.

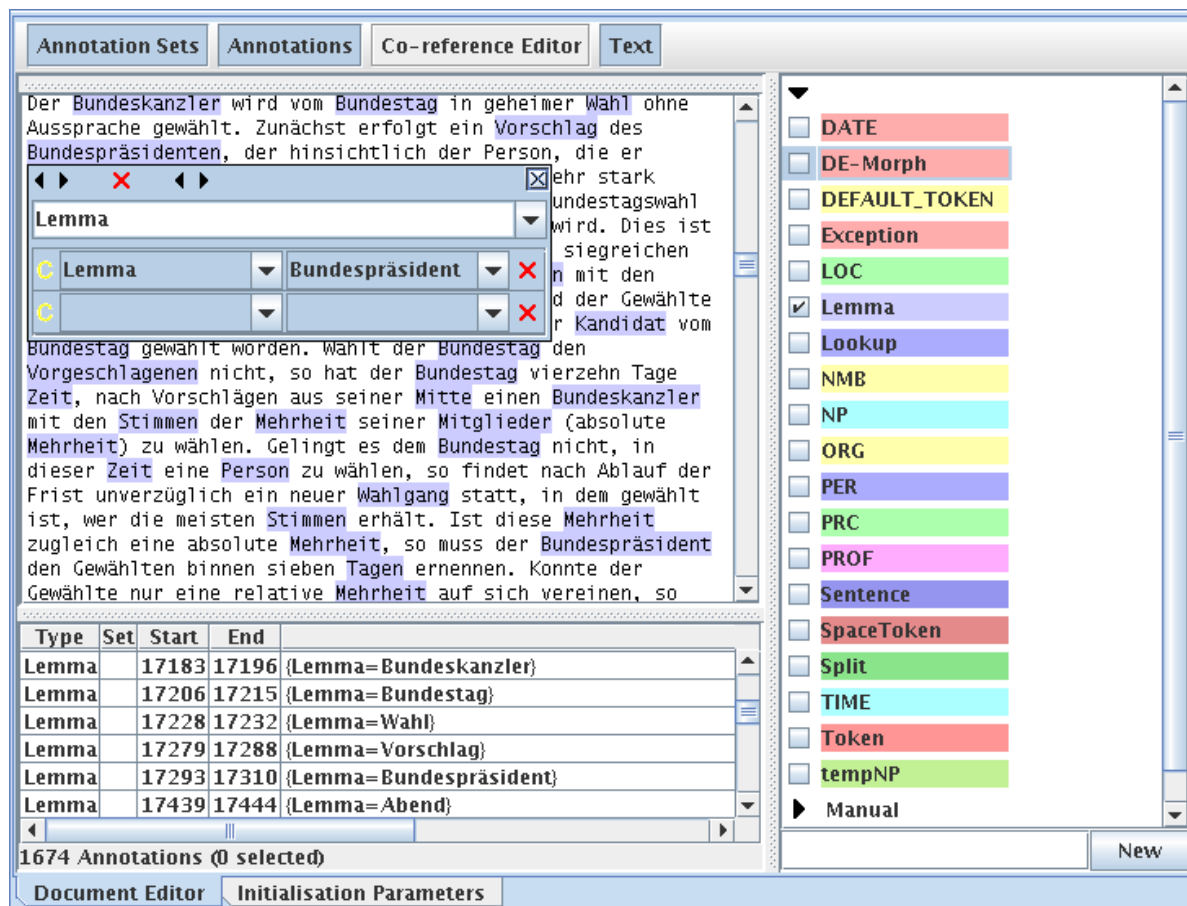


Figure 1.1: Annotations generated by the Durm Lemmatizer running in the GATE environment

## 1.1 Overview

The Durm Lemmatization system consists of a number of GATE components and resources, which have to be used within the GATE (<http://gate.ac.uk/>) architecture. It includes the following components to perform morphological analysis and lemmatization:

- The *Case Tagger*, which adds case information (Nominativ, Genitiv, Dativ, Akkusativ) to nouns;
- The *POS-based Number Tagger*, which adds number information (singular, plural);

## 1 The Durm Lemmatizer

- The *Morphological Analyser*, which classifies nouns into morphological classes;
- The *German Lemmatizer*, which annotates nouns with their lemma.

Additionally, it uses information provided by two other components:

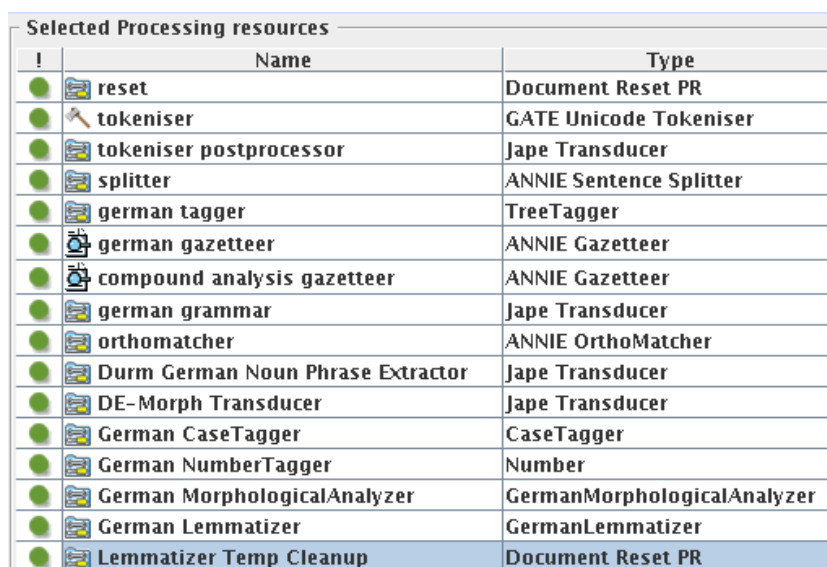
- A POS-tagger for German (currently we only support the STTS tagset as used by the TreeTagger<sup>1</sup>)
- The *MuNPEX*<sup>2</sup> noun phrase chunker for German.

The *Durm German lexicon* is a main resource in the lemmatization system. It is an automatically created and updated German lexicon containing lemma, number, and case information for nouns.

For a more detailed motivation, as well as the theoretical background, you should read our paper on German lemmatization (Perera and Witte, 2005).

## 1.2 Setup

You should have a working GATE installation including the TreeTagger. Then set up a pipeline with the following components (Figure 1.2):<sup>3</sup>



| ! | Name                              | Type                        |
|---|-----------------------------------|-----------------------------|
|   | reset                             | Document Reset PR           |
|   | tokeniser                         | GATE Unicode Tokeniser      |
|   | tokeniser postprocessor           | Jape Transducer             |
|   | splitter                          | ANNIE Sentence Splitter     |
|   | german tagger                     | TreeTagger                  |
|   | german gazetteer                  | ANNIE Gazetteer             |
|   | compound analysis gazetteer       | ANNIE Gazetteer             |
|   | german grammar                    | Jape Transducer             |
|   | orthomatcher                      | ANNIE OrthoMatcher          |
|   | Durm German Noun Phrase Extractor | Jape Transducer             |
|   | DE-Morph Transducer               | Jape Transducer             |
|   | German CaseTagger                 | CaseTagger                  |
|   | German NumberTagger               | Number                      |
|   | German MorphologicalAnalyzer      | GermanMorphologicalAnalyzer |
|   | German Lemmatizer                 | GermanLemmatizer            |
|   | Lemmatizer Temp Cleanup           | Document Reset PR           |

Figure 1.2: Sample pipeline configuration for the lemmatization system

1. Load GATE's sample application for German: `german+tagger.gapp` (you can find it in your GATE installation under `gate/plugins/german/resources/`). **Note:** this pipeline works on the annotation set "NE," so you'll either have to (a) also use "NE" as the input/output annotation set for all downstream components or (b) (perhaps simpler) remove all references to "NE" within the GATE pipeline to make it work on the default annotation set.

<sup>1</sup><http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html>

<sup>2</sup><http://www.ipd.uka.de/~durm/tm/munpex/>

<sup>3</sup>If you do not know how to add a new CREOLE repository or load new components into a pipeline, please read the GATE's user guide first at <http://www.gate.ac.uk/sale/tao/index.html>.

2. Add the MuNPEX<sup>4</sup> noun phrase chunker for German (using the main grammar file `de-np.main.jape`)
3. Add a JAPE-Transducer component with the grammar file `DeLem/de_morph.main.jape`
4. Add the *Case Tagger* component (`CaseTagger/build`). Here you'll have to set the initialization parameter to the `CaseProbs` directory containing the probability files.  
**Note:** To add this and the next three components from the GUI, use *File* → *Manage CREOLE plugins* → *Add a new CREOLE repository* and then select the indicated build directory.
5. Add the *Number Tagger* component (`Number/build`)
6. Add the *German Morphological Analyzer* component (`GermanMorphologicalAnalyzer/build`)
7. Add the main *German Lemmatizer* component (`GermanLemmatizer/build`). Here you'll have to set the initialization parameter to the file containing the German lexicon: `DE-Lexicon/delexicon.txt`.
8. Optionally: add a "Document Reset" component to remove the temp annotations `NPFNP`, `PosNumber`, `Preposition`, `PN`, `Case`, `Num`, and `Gender` (see Figure 1.3)





| Name  | Type                | Required | Value   |
|---|---------------------|----------|---|
|  annotationTypes | java.util.ArrayList |          | [NPFNP, PosNumber, Preposition, PN, Case, Num, Gender ]  |
|  setsToKeep      | java.util.ArrayList |          | []   |

Figure 1.3: You can remove temporary annotations generated by the Durm lemmatizer with a *Document Reset PR* component

Now load some German texts and run the pipeline. Enjoy the new annotations for **Lemma** and **DE-Morph**.

<sup>4</sup><http://www.ipd.uka.de/~durm/tm/munpex/>





## 2 German Case Tagger

The German case tagger assigns a grammatical case (Nom, Gen, Dat, Akk) for each noun in a document.

### 2.1 Overview

The case tagger is developed as an additional resource to support the Durm lemmatization system. It is a main component in the Durm lemmatizer since the lemmatizer requires the grammatical case for nouns in order to determine their base forms. It uses information provided by a POS-tagger<sup>1</sup>. It takes sentences tagged for part-of-speech as input and attempts to produce the best case tag for each noun or pronoun in the sentence. The underlying tagging algorithm of the case tagger is based on the stochastic tagging algorithm generally known as the Hidden Markov Model or HMM tagger. For a morphologically complex language like German, assigning the correct case for each noun is a very difficult task. For example, given the following sentence:

Sie/NOM AKK trinken Wasser/NOM AKK DAT

Automatically assigning a case tag to each noun or pronoun is not trivial, because the grammatical case for *Sie* and *Wasser* is ambiguous. That is, they have more than one possible grammatical case. The pronoun *Sie* can either be *nominative* or *accusative* and the noun *Wasser* can take 3 possible values for case, *nominative*, *accusative*, and *dative* respectively. The task of case tagging is to resolve these ambiguities, choosing the proper tag for the context.

### 2.2 Usage

#### 2.2.1 Initialization Parameters

When initializing the case tagger component, you'll have to set the following parameters:

**Name** the component's name, as it appears under PROCESSINGRESOURCE. You can leave it empty, it will then default to CASETAGGER.

**probabilityFiles** path to the directory, where the probability files, listed in section 2.3.1, are stored for the calculation of probabilities for stochastic case tagging. This parameter should be set when initializing this component.

#### 2.2.2 Runtime parameters

**inputASName** input annotation set

**outputASName** output annotation set

**extractAnnotations** these are annotations to extract in addition to tokens for the case tagger. Since this component requires NPs in addition to tokens, set this parameter to NP.

---

<sup>1</sup>Currently, only the TreeTagger with the STTS tagset is supported

### 2.2.3 Output Annotations

**Case** The case (Nom, Gen, Dat, Akk) for every noun in a document.

## 2.3 Implementation notes

The implementation of the case tagger is based on an HMM tagger. Like other stochastic taggers, it picks the most likely tag for the noun based on learned probabilities of a training corpus. The required probabilities are located as files in the directory given as a parameter when initializing the component.

HMM taggers choose the tag sequence that maximizes the following formula:

$$P(\text{word} \mid \text{tag}) * P(\text{tag} \mid \text{previous } n \text{ tags}) \quad (2.1)$$

HMM taggers generally choose a tag sequence for a whole sentence rather than for a single word. The case tagger is based on a trigram-HMM and chooses the tag  $t_i$  for word  $w_i$  that is most probable given the previous two tags  $t_{i-1}$  and  $t_{i-2}$  and the current word  $w_i$ :

$$t_i = \underset{j}{\operatorname{argmax}} P(t_j \mid t_{i-1}t_{i-2})P(w_i \mid t_i) \quad (2.2)$$

The interface to the case tagger defines one method, `String[] getCaseTags(Document gatedoc, ArrayList tokens)`, it accepts a GATE document and an ArrayList of tokens as parameters. The list of tokens contains a sentence, which contains tokens that are annotated for part-of-speech. This method returns an array of strings that contains the grammatical case for each noun in the sentence in the same order as the nouns in the sentence. The underlying implementation of the case tagger calls the methods in the *Viterbi* interface, which defines the functionality of the *Viterbi* algorithm<sup>2</sup>, which finds the best sequence of case tags for the nouns in the sentence.

### 2.3.1 Probability Files

**n1counts.txt** Unigram counts

**n2counts.txt** Bigram counts

**n3counts.txt** Trigram counts

**context2.txt** Bigram probabilities

**context3.txt** Trigram probabilities

**WordStat.txt** Lexical probabilities  $P(\text{NN} \vee \text{NE} \mid \text{tag})$  for normales Nomen or Eigennamen

**adjstat.txt** Lexical probabilities  $P(\text{ADJA} \mid \text{tag})$  for attributives Adjektiv

**apprstat.txt** Lexical probabilities  $P(\text{APPR} \mid \text{tag})$  for Präposition

**apprartstat.txt** Lexical probabilities  $P(\text{APPRARTS} \mid \text{tag})$  for Präposition mit Artikel

**artstat.txt** Lexical probabilities  $P(\text{ART} \mid \text{tag})$  for bestimmter oder unbestimmter Artikel

**pdatstat.txt** Lexical probabilities  $P(\text{PDAT} \mid \text{tag})$  for attribuierendes Demonstrativepronomen

**pidatstat.txt** Lexical probabilities  $P(\text{PIDAT} \mid \text{tag})$  for attribuierendes Indefinitepronomen mit Determiner

<sup>2</sup>[http://en.wikipedia.org/wiki/Viterbi\\_algorithm](http://en.wikipedia.org/wiki/Viterbi_algorithm)

**piatstat.txt** Lexical probabilities  $P(\text{PIAT} \mid \text{tag})$  for attribuierendes Indefinitepronomen ohne Determiner

**pposatstat.txt** Lexical probabilities  $P(\text{PPOSAT} \mid \text{tag})$  for attribuierendes Possesivepronomen

**pperstat.txt** Lexical probabilities  $P(\text{PPER} \mid \text{tag})$  for irreflexives Personalpronomen

**prfstat.txt** Lexical probabilities  $P(\text{PRF} \mid \text{tag})$  for reflexives Personalpronomen

**Visit the JavaDoc Documentation**

<../../../../Gate/CaseTagger/doc/javadoc/index.html>

## 2 German Case Tagger

## 3 German POS-based Number Tagger

The German POS-based number tagger determines the number of the subject noun of a sentence.

### 3.1 Overview

The POS-based number tagger has been developed as an additional resource to support the Durm lemmatization system. It analyzes a whole sentence using the part-of-speech information provided by a German POS-tagger<sup>1</sup> and the case information provided by the case tagger in order to determine the number of the subject noun of a sentence.

Determining the number of the subject is based on a basic grammatical rule in German that says, the number of the subject should agree with the number of the main verb in the sentence. With the help of the case tagger we find the subject (case: Nom) of the sentence and with the help of the part-of-speech tagger we find the main verb. Since we have both the main verb and the subject, we apply a small heuristic to determine the number of the main verb. This is done by checking the suffix of the main verb, since in German most of the plural verbs have the suffix *-en* or *-n*. In this way, we determine the number of the main verb and in turn the number of the subject noun.

### 3.2 Usage

In a pipeline, this component must be inserted after a POS-tagger and the Case tagger, since it uses information given by these two components.

#### 3.2.1 Runtime parameters

**inputASName** input annotation set

**outputASName** output annotation set

**extractAnnotations** these are annotations to extract in addition to tokens for the POS-based number tagger. Since this component requires NPs in addition to tokens, set this parameter to NP.

#### 3.2.2 Output Annotations

**PosNumber** The number (Sg or Pl) for the subject in a sentence.

### 3.3 Implementation notes

Initially this component determines the main verb of the sentence by looking at the POS tags. This is done by iterating through each token of a sentence and examining the POS

---

<sup>1</sup>Currently, only the TreeTagger with the STTS tagset is supported

### 3 German POS-based Number Tagger

of each token until a token with the POS *VVFIN*<sup>2</sup>, *VAFIN*<sup>3</sup>, or *VMFIN*<sup>4</sup> is found. When the program finds a token with one of these POS tags, this token is assigned as the main verb of the sentence, since in German the POS of a main verb is *VVFIN*, *VAFIN*, or *VMFIN*. After determining the main verb, the program determines the subject of the sentence by looking at the grammatical case tag for each noun. This is again done by iterating through the tokens in a sentence until a match is found for *POS = NN and Case = Nom*, i.e., a noun with the case tagged as nominative. After finding the subject, it applies the heuristic explained above, looking at the suffix of the verb, in order to determine the number of the subject.

The interface to the POS-Number tagger defines the method `public java.lang.String getNumber (java.Util.ArrayList tokens)` method, where it accepts an `ArrayList` of tokens as its arguments to the method and returns a string representing the position of the subject of the sentence and whether the subject is singular or plural. For example, the string "30" means that the 3rd noun of the sentence is singular. The last digit of the string defines the number, i.e., *0 = singular and 1 = plural*.

**Visit the JavaDoc Documentation**

<../../../../Gate/Number/doc/javadoc/index.html>

---

<sup>2</sup>finites Verb, voll

<sup>3</sup>Infinitive, aux

<sup>4</sup>finites Verb, modal

# 4 German Morphological Analyzer

The German morphological analyzer assigns number and gender for nouns in a document.

## 4.1 Overview

The morphological analyzer considers the context of nouns by analyzing NPs given by the multi-lingual NP chunker (MuNPE) in order to provide the morphological classification required for lemmatization. The Durm lemmatizer processes nouns considering their morphological features such as number, gender, and their surrounding context. Since information required for lemmatization regarding number and gender cannot be solely determined from the word form itself, the lemmatization algorithm captures the context of nouns by analyzing NP chunks.

The algorithm processes NPs with:

- determiners,
- determiners and modifiers,
- modifiers only,
- without determiners or modifiers,

in order to compute the features *number* and *gender*. This component uses rules and heuristics based on the German grammar.

## 4.2 Usage

Since the main input to this component are NP chunks, this component must be inserted to the pipeline after an NP chunker.

### 4.2.1 Runtime parameters

**inputASName** input annotation set

**outputASName** output annotation set

**extractAnnotations** these are annotations to extract in addition to tokens for the German morphological analyzer. Since this component requires NPs in addition to tokens, set this parameter to NP.

### 4.2.2 Output Annotations

**Number** The number for nouns in the document.

**Gender** The gender for nouns in the document.

### 4.3 Implementation notes

This component processes nouns in different ways with respect to their context information given by the NP chunker. To facilitate this kind of processing, the interface to the lemmatization algorithm defines one method, *public MorphologyImpl classifyMorphology (Annotation token, gate.Document doc)*, which takes a token as an argument within the currently processing document and returns an object of type *MorphologyImpl*, which holds information regarding number, gender etc. This method is then implemented in different ways in different subclasses in the inheritance hierarchy.

In order to decouple the interface from its implementation so that the two can vary independently, we have employed the *Bridge* design pattern. The class *Morphology* defines the abstraction interface, which maintains a reference to an object of type implementor and the class *GermanMorphology* extends the interface defined by the Abstraction. The class *MorphologyImpl* defines the interface for implementation classes and its subclasses implement the Implementor interface and defines their concrete implementation.

**Visit the JavaDoc Documentation**

[../../../../Gate/LemmatizationAlg/doc/javadoc/index.html](http://../../../../Gate/LemmatizationAlg/doc/javadoc/index.html)



# 5 German Lemmatizer

The German Lemmatizer lemmatizes nouns in a document based on the morphological features number, gender, and case based on the morphological classes generated by the *German Morphological Analyzer* and lookups in the Durm lexicon. Additionally, it inserts new entries into the Durm lexicon and updates existing entries, allowing the lexicon to evolve in both coverage and accuracy.

## 5.1 Overview

This component is the last component within the Durm lemmatization system, where the actual lemmatization and lexicon generation take place. It determines the lemma of nouns or possible lemma candidates for them by applying a simple algorithm, depending on their morphological classes as given by the German Morphological Analyzer. It then uses the lemma given by the lemmatizer to update the lexicon. When it updates the lexicon it also tries to correct the existing entries in the lexicon as well as the entry that is currently entered to the lexicon from the entries that are already in the lexicon.

The input to this component is a noun with its morphological features number, gender, and case. Based on these features the lemmatizer determines the lemma or lemma candidates<sup>1</sup>. Afterwards the noun with the lemma including other morphological information is used to update the lexicon.

## 5.2 Durm Lexicon

The Durm lexicon is generated automatically from nouns processed by the German Lemmatizer. It grows by updating itself, learning correct values for the lexical entries. The lexicon stores full forms of words with their base form or lemma and other morphological features such as number, gender, and case. Additionally, it also holds information on:

- The number of times that the entry has been found when generating the lexicon
- Inserted time
- Modified time
- Reference to a file, which specifies documents, where the entry has been found
- A lock to specify, whether the entry's lemma is correct or manually corrected, and therefore, needs not to be updated.

An example entry in the lexicon is shown below:

```
Kinder Pl Masc Nom.Akk.Dat Kind 102 27/7/2005  
20:14:52 21/10/2005 8:47:19 36248 locked
```

<sup>1</sup>Lemma candidates are generated for nouns with irregular morphological features, for example, nouns with umlauts. The correct lemma for these nouns can be identified, when the same noun appears again in a different context

## 5 German Lemmatizer

The file reference (36248) points to the an entry in an auxiliary file holding information about the documents containing the word:

```
36248
file:/home/user/Testdata/TestCorpus3/Test100.txt
file:/home/user/Testdata/TestCorpus3/Test102.txt
.
.
file:/home/user/Downloads/Spiegel/12.05.2005wwwww.txt
```

Currently the lexicon is available in plain text format. We are also working on making it available in XML format.

### 5.2.1 Evolving the Lexicon

The lexicon has the capability of self-correction. This feature has been implemented by lexicon update procedures. These procedures are illustrated using examples.

#### Updating Lemmas

If a new word to be inserted has more than one lemma candidate, the lexicon tries to assign the correct lemma for this new word by looking at the lemmas that are already in the lexicon:

| Current state of the lexicon (lemma only) |      |
|---|------|
| Land                                      | Land |
| Landes                                    | Land |

| New Entry |                             |
|-----------|-----------------------------|
| Länder    | Lände . Länd . Lande . Land |

| State of the lexicon after update |      |
|-----------------------------------|------|
| Land                              | Land |
| Landes                            | Land |
| Länder                            | Land |

In the same way, if a new word to be inserted has the correct lemma, the lexicon tries to update the words in the lexicon that have more than one lemma using the lemma of the new word:

| Current state of the lexicon (lemma only) |   |
|---|---|
| Länder                                    | Lände . Länd . Lande . Land                   |
| Ländern                                   | Länder . Lände . Länd . Lander . Lande . Land |

| New Entry |      |
|-----------|------|
| Landes    | Land |

| State of the lexicon after update |      |
|-----------------------------------|------|
| Landes                            | Land |
| Länder                            | Land |
| Ländern                           | Land |

#### Automatic Error Correction

The lemmatization algorithm may produce errors, for example, a plural noun wrongly tagged as singular may not be lemmatized, resulting in a wrong entry. While the lexicon evolves, such errors produced by the algorithm are corrected automatically. If a word that has a wrong entry in the lexicon is entered again with the correct lemma, the word itself and all its inflectional forms will be updated with the correct lemma:

|   |                                       |
|---|---------------------------------------|
| Current state of the lexicon (lemma only) |                                       |
| Jahr                                      | Jahr                                  |
| Jahre                                     | Jahre <b>(wrong)</b>                  |
| New Entry                                 |                                       |
| Jahren                                    | Jahre.Jahr                            |
| State of the lexicon after update         |                                       |
| Jahr                                      | Jahr                                  |
| Jahre                                     | Jahre <b>(wrong)</b>                  |
| Jahren                                    | Jahre.Jahr <b>(two possibilities)</b> |
| New Entry                                 |                                       |
| Jahre                                     | Jahr <b>(correct lemmatization)</b>   |
| State of the lexicon after update         |                                       |
| Jahr                                      | Jahr                                  |
| Jahre                                     | Jahr                                  |
| Jahren                                    | Jahr                                  |

### 5.2.2 Manual Correction of Entries

Some entries in the lexicon may need to be corrected manually or some manual inspection may be done to avoid the system from updating correct lemmas. Those entries that have been manually corrected or have been determined as correct will be locked. Locked entries are not be processed by lexicon update algorithms.

## 5.3 Usage

This component must run after case tagger, POS-based number tagger, and the morphological analyzer components.

When initializing the case tagger component, you'll have to set the following parameters:

**lexiconPath** path to the directory, where the Durm lexicon is stored. This parameter must be set when initializing this component.

### 5.3.1 Runtime parameters

**inputASName** input annotation set

**outputASName** output annotation set

### 5.3.2 Output Annotations

**Lemma** The lemma produced by the lemmatizer for nouns in a document.

**DE-Morph** An annotation containing values for *number*, *gender*, and *case* for nouns in a document (Figure 5.1).

## 5.4 Implementation notes

The lexicon is loaded when the component is initialized. The updates to the lexicon are written to the lexicon during run-time. The lexicon entries are stored in hash tables. In order to support self-correction and fast updates, three hash tables are employed. The first hash table *lexiconEntries* stores lexicon entries, pointing to all its features like number, gender, case, lemma etc., the next hash table *entriesLemma* only to the lemma, and the last

## 5 German Lemmatizer

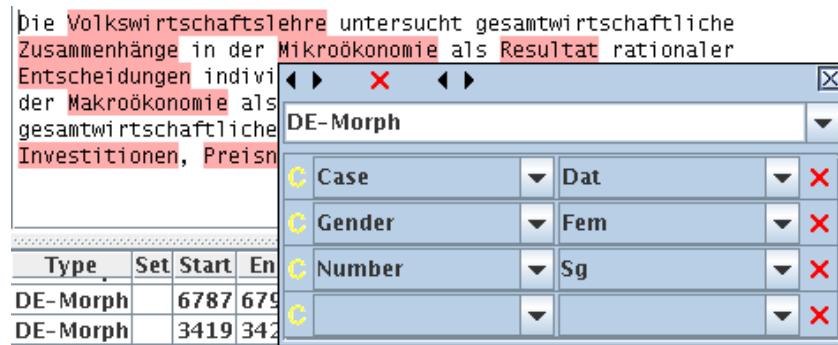


Figure 5.1: Example output annotation generated by the Durm lemmatizer

hash table *lemmaEntries* contains lemmas in the lexicon pointing to their respective entries. The lexicon update algorithms are coupled with these hash tables.

Due to its higher accuracy, the lemma produced by the lexicon has precedence over the lemma produced by the algorithm, if both are bale to determine the lemma.

**Visit the JavaDoc Documentation**

<../../../../Gate/GermanLemmatizer/doc/javadoc/index.html>

# Bibliography

Praharshana Perera and René Witte. A Self-Learning Context-Aware Lemmatizer for German. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP 2005)*, pages 636–643, Vancouver, British Columbia, Canada, October 6–8 2005. Association for Computational Linguistics. <http://www.aclweb.org/anthology/H/H05/H05-1080>.