Schema and Variation: Digitizing Printed Dictionaries

Christian Schneiker and **Dietmar Seipel**
Department of Computer Science
University of Würzburg, Germany
{schneiker, seipel}@informatik.uni-wuerzburg.de

Werner Wegstein
Department of Computational Philology
University of Würzburg, Germany
werner.wegstein@uni-wuerzburg.de

Abstract

In this paper we show how to exploit typographical and textual features of raw text for creating a fine–grain XML Schema Markup with special focus on capturing linguistic variation in dictionaries. We use declarative programming techniques and context–free grammars implemented in PROLOG.

1 Introduction

In 1996, Cambridge University Press proudly presented an outstanding milestone in electronic publishing: Samuel Johnson: A Dictionary of the English Language on CD–ROM, edited by Anne McDermott; containing the First Edition 1755 and the significantly revised Fourth Edition 1773 (McDermott, 1996). "The Dictionary is not only the first great work of English lexicography but also a literary and historical resource of immense value, and this electronic edition has been prepared to the highest standards by a team of scholars at the University of Birmingham." (Cambridge University Press Catalogue, 2009)

The announcement highlighted all the key *characteristics of electronic texts*: accessability, completeness, use of multi–media environment, searchability and highest standards applied by scholars, i.e. philological reliability and precision, wrapped in leading edge technology (Gärtner/Wisbey, 1974). Today, more than a decade and at least one electronic product life-cycle later, the CD is still on sale – as far as we could find out unchanged – and has not lost anything of its former value.

In the context of *digitizing* the cultural heritage there is even a strong and growing demand for digitizing research tools like dictionaries (cf., e.g., Gallica Digital Library Charter/Chapter: Time period covered). But, in the field of electronic text editing, requirements grow rapidly and standards develop fast. The users of electronic texts today want to search not only for words, phrases, headwords, quotations, and authors of sources. They would like to get access to and search for *variant forms, grammatical categories, usage indicators* and the structuring of the description of *word senses*, etc., not only in single dictionaries, but – perhaps using a grid environment – in fully connected dictionary networks (cf. the dictionary search, possible within the Trier Dictionary Net and as a TextGrid feature). In the context of these new user scenarios, possibly grid–based, usable for collabortive research and secured safely in longterm archive structures, we try to put fine–grain encoding ideas into practise using Joachim Heinrich Campe's dictionary of the German Language as testbed.

This is one of the reasons why TEXTGRID (2009), the first grid project in German e*Humanities*, funded by the Federal Ministry of Education and Research, chose the Campe Dictionary (1811): 6 volumes with altogether about 6.000 pages and about 140.000 entries, published between 1807 and 1813 as one testbed for their TEXTGRID Lab, a Virtual Research Library. It entails a grid–enabled workbench that will process, analyse, annotate, edit and publish text data for academic research and TEXTGRIDRep, a grid repository for long–term storage. Electronic dictionaries are used in a wide field of new research areas such as the growing community of *eHumanities*. One of the main projects for the German humanities is the community project TEXTGRID, which aims to develop a platform for the collaborative editing, annotation, analysis and publication of texts (TEXTGRID, 2009). According to the TEI Consortium (2009), large text corpora from different epochs have to be parsed and annotated for performing further analysis, such as building a meta–lemma list for the project *interdependencies* between *language* and *genomes*.

In general, there are the following important *prerequisites* for state of the art text encoding in the Humanities. The encoding should use international standards, especially XML and related standards, e.g., TEI P5 with

Proceedings of the Third Linguistic Annotation Workshop, ACL-IJCNLP 2009, pages 82–89,
Suntec, Singapore, 6-7 August 2009.

XML Schema. Also the combination of text and image is necessay. The text capture should aim at reference quality for the encoded text. A fine–grain encoding preserving lexicographical and textual variation without blurring (distorting) the content modelling of XML elements is helpful. Finally, a TEI schema (Relax NG) that is flexible enough to encode variation in lexicographical and textual structures without loosening the grip of the constraints is necessary to define clear cut element content.

In this paper, we present an annotation *workflow* using declarative programming techniques for fine–grain text markup, and we apply it for retro–digitizing a printed version of the Campe Dictionary. Our *parsing* and *annotation* toolkit is based on SWI–PROLOG and the XML query and transformation language FNQUERY (Seipel, 2002), which is implemented in SWI–PROLOG. Using PROLOG technology for parsing and annotating is common in natural language processing. It has, e.g., been used within the Jean Paul project at the Berlin–Brandenburg Academy of Sciences (Seipel et al., 2005), where XML transformations based on FNQUERY turned out to be easier to write than XSLT transformations. A frequently applied method in PROLOG programming is to find the proper level of abstraction and to write suitable macros for frequently occurring patterns in the code; PROLOG even allows to design dedicated special–purpose languages (O'Keefe, 1990). Definite clause grammars have been developed as an abstraction for parsing; this has, e.g., been very successful for parsing controlled natural languages (Fuchs et al., 1994; Fuchs et al., 1995; Schwitter, 2008).

Structure of the Paper. The rest of this paper is organized as follows: In Section 2, we sketch the worklflow for capturing text from the printed text corpus and the semi–automatic error correction to produce a source file for our parsing and annotation toolkit. Section 3 gives an overview of the structure of the different entries in the dictionary; we will explain this structure with the lemma "Der Aal", and we will examplify the variation of entries. The next section shows the annotation of the different lemma variants and the parsing of nouns and verbs. In Section 5, we describe the parsing and annotation of the sense block with citations and references, punctuation, linebreaks and hyphenation. The last section gives a conclusion of our work.

2 The Campe Workflow for Text Capture

Since the Campe Dictionary was written in the early 19th century, the text could not be captured with modern methods of optical character recognition (OCR). Thus, in a first step, the whole corpus had to be doubled–keyed in China. This could also avoid unconsciously corrected spelling variants in old German text, which is frequently done by native speakers. Figure 1 shows the typographical layout of an entry in the Campe Dictionary.

The second step in the text capture workflow was the correction of illegible characters according to the context, as well the manual correction of printing errors provided by the publishers. For providing an efficient and easy–to–undo workflow system for these corrections, we decided to use a semi–automatic process: corrections made by human experts could be repeated on the whole context by using regular expressions in a standard POSIX–Regex environment, and automatic corrections could be done by processing the workflow logfiles of other volumes of the Campe Dictionary.

One of the main concerns in this preprocessing steps was the pre–annotation of abbreviatons used by the author such as etc., s. a. and z. b. or even abbreviated author names like C. for Campe. These had to be checked manually and pre–annotated by a parser written in PROLOG, which can also recognize named entities.

After logging all these corrections in UNIX diff files, the base file for the text conversion into XML could be generated.

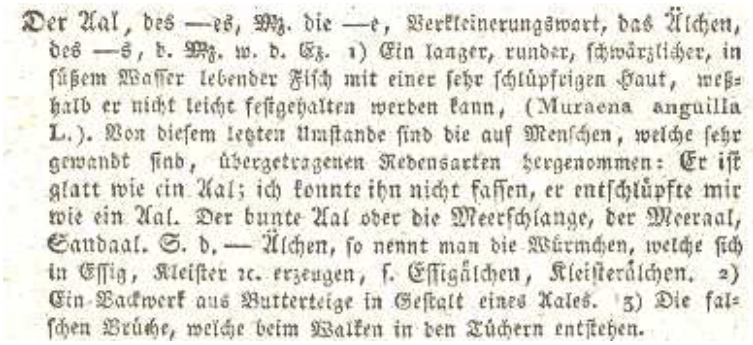
3 The Structure of Entries

Within the parsing process, the only annotations available so far for structure recognition were the declaration of the different font sizes used by Joachim Heinrich Campe, the numbering of the line and page breaks in the dictionary, and paragraphs; thus, we found a very limited XML structure in the source file which we used for the first basic transformaions.

In most available dictionaries, each entry is encapsulated in its own paragraph, and thus, it could be easily detected. In the following preannotated example, which is the result of the double key process, an entry is annotated with `paragraph` and is followed by an element `W_2`, which shows the lemma of the entry in a larger font; recognizing both elements is necessary, because there could exist other `paragraph` elements which do not represent entries. This preliminary structure, which is not yet according to TEI, is used as the starting point for the annotation process.

```
<paragraph>
   <W_2>Der Aal</W_2>,
   <W_1>
      des -- es, Mz. die -- e
   </W_1>, ...
</paragraph>
```

The following annotation process derives an encoding based on the TEI P5 Guidelines (TEI Consortium, 2009), using a Relax NG Schema. The encoding structure uses elements to markup an entry of a dictionary, which consists of 1) a form block with inflectional and morphological information and 2) a sense block handling semantic description and references, quotations, related entries, usage as well as notes. In the future,



Der Aal, des —es, Mz. die —e, Verkleinerungswort, das Älchen, des —s, d. Mz. w. d. Ez. 1) Ein langer, runder, schwärzlicher, in süßem Wasser lebender Fisch mit einer sehr schlüpfrigen Haut, weßhalb er nicht leicht festgehalten werden kann, (Muraena anguilla L.). Von diesem letzten Umstande sind die auf Menschen, welche sehr gewandt sind, übergetragenen Redensarten hergenommen: Er ist glatt wie ein Aal; ich konnte ihn nicht fassen, er entschlüpfte mir wie ein Aal. Der bunte Aal oder die Meerschlange, der Meeraal, Sandaal. S. d. — Älchen, so nennt man die Würmchen, welche sich in Essig, Kleister ꝛc. erzeugen, s. Essigälchen, Kleisterälchen. 2) Ein Backwerk aus Butterteige in Gestalt eines Aales. 3) Die falschen Brüche, welche beim Walken in den Tüchern entstehen.

Figure 1: Excerpt from the Campe Dictionary

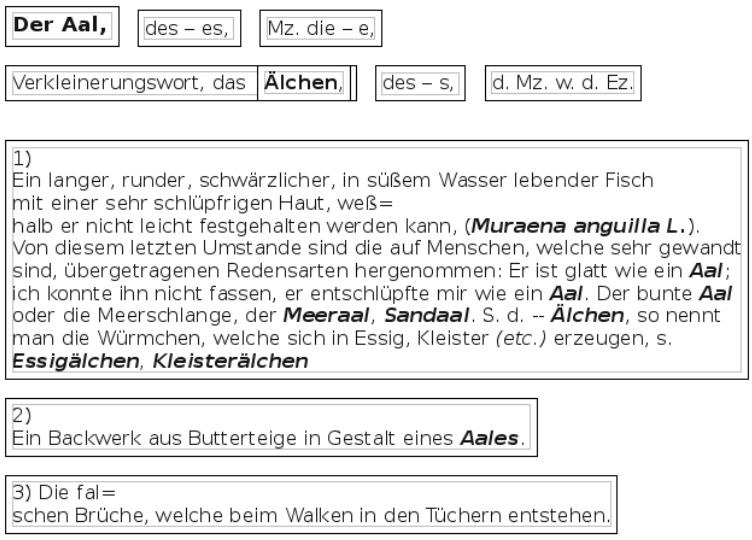


Figure 2: Rendering of an Annotated Entry

this encoding will help us to structure the digital world according to semantic criteria and thus provide an essential basis for constructing reliable ontologies. The annotation of the form block and of the sense block will be described in the Sections 4 and 5, respectively. For both, we have to be able to handle many forms of variation.

The Variation Problem. Lexicographical structures, such as in the Campe Dictionay, can have a lot of variation in entry and headword. E.g., volume 1 has 26.940 entries. The morphological structure is as follows: attributes are used to form elements for the encoding of inflectional, lexical and dialect variation [orthographical, ...], as well as variation in usage. In semantical structures, attributes to elements of the sense block are used to encode semantics.

Variation could, e.g., consist of several headwords linked by conjunctions like ”oder” and ”und”; the additional headwords are usually printed with a smaller font size than the first headword of the entry. The following example shows such a variant with more than one headword and its appropriate inflectional forms. Abbreviations like ”d. Mz. w. d. Ez.” or ”Mz. s. Ez.” are defining a plural form with the same notation as the singular. These inflections have to be recognized and annotated; in the following preannotated example, the singluar form element is repeated.

```
<paragraph>
   <W_2>Der Aalstreif</W_2>,
   <W_1>des -- es, Mz. die -- e</W_1>,
   oder <W_1>der Aalstreifen, des
   ^$0002.18 -- s</W_1>,
   d. <W_1>Mz.</W_1> w. d. Ez.
</paragraph>
```

4 Annotating the Form Block in TEI

We use declarative programming in PROLOG and FNQUERY as a solution for text conversion in general. In the following, we will illustrate and discuss this for nouns and verbs. This reflects our workflow for annotating the Campe Dictionary, but our approach can also be applied to other dictionaries.

4.1 Nouns

The lemma line ”Der Aal” is encoded as follows:

```
<form type="lemma">
   <gramGrp>
      <pos value="noun" />
      <gen value="m" />
   </gramGrp>
   <form type="determiner">
      <orth>Der</orth>
   </form>
   <form type="headword">
      <orth>Aal</orth>
   </form>
</form>
```

For the inflected lemma line "Mz. die – e" we would like to obtain the following TEI structure:

```
<form type="inflected">
   <gramGrp>
      <gram type="number">
      <abbr>Mz.</abbr> </gram>
      <case value="nominative"/>
      <number value="plural"/>
   </gramGrp>
   <form type="determiner">
      <orth>die</orth>
   </form>
   <form type="headword">
      <orth>
         <oVar>
            <oRef>-- e</oRef>
         </oVar>
      </orth>
   </form>
</form>
```

The Parsing Workflow in PROLOG. A sequence "Xs" of form elements is read using the new PROLOG predicate "sequence", which we have implemented. This is a compact way of specifying lists of tokens of the same type (in our case form).

```
campe_parse_headword(Xs) -->
   sequence('*', form, Xs).
```

In standard PROLOG, we would have to encode this in a more verbous way using recursion. In addition, the rule above uses the *definite clause grammar (*DCG*)* notation ("-->") of PROLOG (Gazdar, 1989; O'Keefe, 1990).

For handling complex specifications, a more compact grammar formalism than standard DCG's is needed (Abramson, 1989; Sperberg–McQueen, 2003). For parsing text, we have mainly used an additional grammar formalism ("==>"), which we have developed, the so–called *extended definite clause grammars* (EDCG's); the technical details of EDCG's are described in Schneiker et al. (2009). The following EDCG rules can derive an XML structure that is very close to the TEI for the inflected lemma line above. The rules almost look like the rules of a context–free grammar. A form element consists of a grammar determiner followed by a form headword.

```
form ==>
   grammar_determiner,
   form_headword.
```

A grammar determiner is either a gram element followed by a determiner, or simply a determiner. The alternative is encoded by ";", which stands for "or" in PROLOG. The cut "!" freezes the first alternative, if we detect a gram element; i.e., then a simple determiner is not allowed.

```
grammar_determiner ==>
   ( gram, !, determiner
   ; determiner ).
```

Tokens from the input strean are read using the list notation "[...]". A gram element can only be of the form "Mz.", and a determiner is a token "X", that is a campe determiner. The bracket notation "{...}" does not read from the input stream; instead, it is used for expressing test conditions on the tokens.

```
gram ==> ['Mz.'].
determiner ==> [X],
   { campe_is_determiner(X) }.
```

Finally, a form headword is an orth element, which itself must be the sequence '--' followed by any other token. The wildcard for arbitrary tokens is the anonymous variable "_" of PROLOG.

```
form_headword ==> orth.
orth ==> ['--', _].
```

The 6 EDCG rules above form an EDCG grammar, which can be applied to the stream of input tokens. Thus, we obtain the following XML structure; the tag names are generically taken from the EDCG rules. At this stage, the most important and complicated steps of the parsing have been done. In some further steps of fine tuning, the desired TEI structure can be obtained using XSLT or the FNTRANSFORM component of FNQUERY.

```
<form>
   <grammar_determiner>
      <gram>Mz.</gram>
      <determiner>die</determiner>
   </grammar_determiner>
   <form_headword>
      <orth>-- e</orth>
   </form_headword>
</form>
```

Finally, sequences of campe headwords can be parsed using the PROLOG predicate "campe_parse_headword".

Visualization of EDCG Rules. EDCG's could be easily visualized using derivation trees (Figure 3); each non–terminal is shown in an ellipse, the terminals are denoted by rectangles, representing the leaves of the

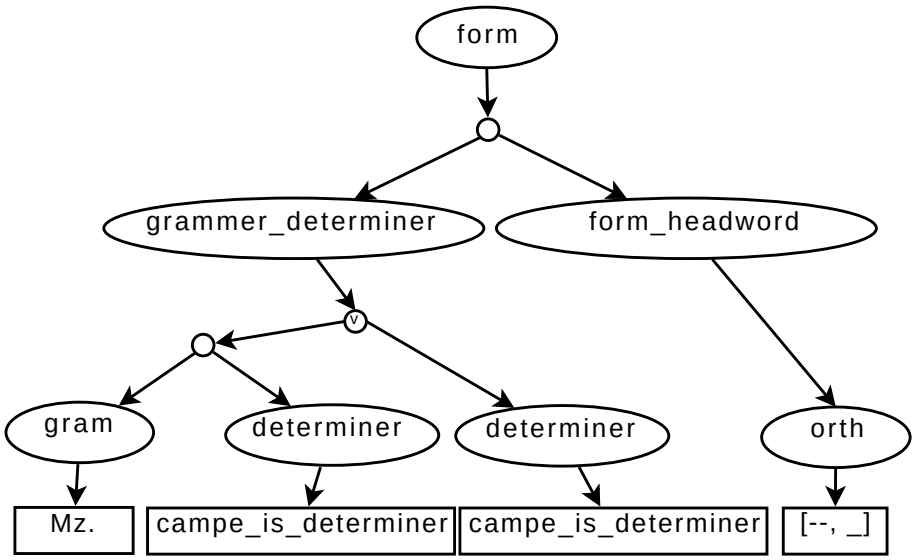


Figure 3: Visualization of the EDCG-rules for parsing forms

tree. Nodes could be either emtpy circles for conjunctions – the "," in the EDCG's – or circles with a "∨" for a logical disjunctions – denoted by ";" in PROLOG.

Handling of Variation. *Grammar* formalisms like DCG's or EDCG's can very well handle variation. The different alternatives can be represented using multiple rules for an XML element or by the alternative construct ";" within a single rule. Moreover, since our grammar formalisms are very compact and thus easily readable, it is possible for the implementer to understand even larger sets of rules and to keep track of the complex structures.

Finally, when several ways of parsing a sequence of tokens are possible, the *backtracking* mechanism of PROLOG explores all alternatives, and it can return all possible results. If later inconsistencies make previous choices of parsing rules impossible, then PROLOG goes back to the last choice point and explores the next alternative. In other programming languages, backtracking has to be implemented explicitely, whereas it is implicit in PROLOG. This overhead makes backtracking more difficult to handle in other programming languages.

4.2 Verbs

Each verb could have additional information about its corresponding part of speech. This information is highlighted with a roman font type in the Campe Dictionary and 8 groups could be isolated:

v. ↦ "verb",
imp. ↦ "impersonal",
intr. ↦ "intransitive",
ntr. ↦ "neuter",
rec. ↦ "reciprocal",
regelm. ↦ "regular",
trs. ↦ 'transitive",
unregelm. ↦ "irregular"

In our base file, we find two different variants of pre–annotated `pos` elements depending on the current processing stage:

```
<A>v.</A>
<A>trs.</A>
<A>unregel.</A>
```

or

```
<hi _>v.</hi>
<hi _>trs.</hi>
<hi _>unregel.</hi>
```

where "_" stands for the attribute/value pair "rend="roman"", which would be annotated as follows:

```
<gramGrp>
   <pos value="verb">
      <abbr>
         <hi rend="roman">v. </hi>
      </abbr>
   </pos>
   <subc value="transitive">
      <abbr>trs.</abbr>
   </subc>
   <subc value="irregular">
      <abbr>unregelm.</abbr>
   </subc>
</gramGrp>
```

Inflected forms are possible for verbs, too.

5 Annotating the Sense Block in TEI

Lists in the sense block can have many different forms and a complex nesting structure, like different sense blocks, citations, hyphenations, references and different font types.

For annotating these sequences and variation, we frequently use the predicate `sequence` of the DDK. Moreover, for parsing lists, we make extensive use of PROLOG's backtracking feature.

5.1 Structuring the Sense Block

The sense block could have a complex nesting structure for defining different meanings of a lemma. In a printed dictionary, often arabic or roman numbers a used for creating a fine–grained structure.

```
<W_2>Abängsten und Abängstigen</W_2>,
<abbr><A>v.</A></abbr>
I) <abbr><A>trs.</A></abbr>
1) Sehr ängsten oder ...
2) Durch Angstmachen zu etwas ...
II) <abbr><A>rec.</A></abbr> ...
<W_1>Das Abängsten,
<lb n="0003.91" /> Abängstigen.
Die Abängstung, Abängstigung</W_1>.
```

Each sense could be part of another subsense, or a new sense could be created. Using PROLOG's backtracking feature, we can find a suitable interpretation of such a structure and annotate it in XML:

```
<sense n="I">
   <lbl type="ordering">I)</lbl> ...
   <sense n="1">
      <lbl type="ordering">1)</lbl> ...
   </sense>
   <sense n="2">
      <lbl type="ordering">2)</lbl> ...
   </sense>
</sense>
```

PROLOG is very well–suited for parsing such nested structures. In general, roman or arabic numbering could be used for a listing at any depth. E.g., the text

```
1)...2)...1)...2)...3)
```

could be structured as a list "`1,2(1,2,3)`" with two elements, where the second element has three subelements, or as a list "`1,2(1,2),3`" with three elements, where the second element has two subelements. Both alternatives can be generated by backtracking. But, if we extend the text by "`...3)`", then our PROLOG approach correctly structures the above prefix in the first way; otherwise there would be two consecutive top–list elements with the same numbering.

5.2 Citations and References

Citations and cross references to other entries are used all over the text corpus.

Citations. Often, citations could be recognized by bible citations and names of authers like *Lessing* or *Richter*, which are often pre–annotated in a larger font size.

```
Um Geld zu fischen, Geld! Um Geld,
^$0004.71 Geld einem Juden
<W_1>abzubangen</W_1>, Geld!
<W_1>Lessing</W_1>.
```

These citations are annotated with a `cit` tag containing the citation as a `quote` tag and the corresponding author in `bibl` and `author`.

```
<cit type="quote">
   <quote> ... </quote>
   <bibl>
      <author n="#Lessing">
         <hi rend="spaced">Lessing</hi>
      </author>
   </bibl>
</cit>
```

References. Cross references to other entries of the Campe Dictionary are usally marked with "S.", "Siehe da" or "s.a." in the sense block.

```
<W_1>Die Abberufung</W_1>. S. d.

<W_1>
   s. Essigälchen, Kleisterälchen
</W_1>
```

These references are annoated with `xr` containing an `lbl` tag with an attribute for marking it as a reference. The target of this references is annotated with `ref` and the corresponding entries as the `target` attribute.

```
<xr>
   <lbl type="reference">s.</lbl>
   <ref target=
   "Essigälchen, Kleisterälchen">
      Essigälchen
      <c type="$,">,</c>
      Kleisterälchen
   </ref>
</xr>
```

5.3 Punctuation

For annotating punctuation in a lemma, which can appear between single headwords, the DCG predicate `campe_punctuation` is used: for each token we check if it is a punctuation mark, and – if so – annotate it with a `c` tag. The meta–predicate `sequence` used in the DCG predicate `campe_punctuations` parses such a list of elements.

```
campe_punctuations(Xs) -->
   sequence('*',
      campe_punctuation, Xs).
```

```
campe_punctuation(X) -->
   ( [A],
     { is_punctuation(A), X = c:[A] }
   ; [X] ).
```

5.4 Linebreaks and Hyphenations

Linear structures like linebreaks and hyphenations are parsed using a combination of FNQUERY and DCG's. A linebreak is annotated as an `lb` element; e.g., `^$0001.24` becomes `<lb n="0001.24" />`. In the base file, each hyphenation is labeled with an equals sign as a separator followed by a line break element.

```
auf Muenzen das Zei=
<lb n="0001.24" />
chen der ersten Stadt
```

The hyphenation itself should not be visual later in the rendered representation of the XML document, so we have removed the delimiter symbol and defined this syllable division as an attribute `rend` of the surrounding `<w>` element. [1]

```
auf Muenzen das
<w rend="Zei-chen">
   Zei=
   <lb n="0001.24"/>
   chen
</w>
der ersten Stadt
```

This sequence could be parsed easily with standard DCG rules in PROLOG. The predicate `create_hyphenation_element` creates the hyphenation XML element with the required attribute and content.

```
campe_hyphenations(Xs) -->
   sequence('*',
     campe_hyphenation, Xs).

campe_hyphenation(X) -->
   ( campe_hyphenation_element(X)
   ; [X] ).
```

The difference between standard DCG's (operator "`-->`") and the new EDCG formalism (operator "`==>`") proposed by us is that EDCG's are more compact and more readable, since they hide the output arguments for the derived syntax tree and produce a generic XML structure instead.

5.5 Font Types

The different font types in the Campe Dictionary, like the roman font family or larger fonts sizes for headwords and inflected forms, are pre–annotated in the capturing process.

[1] We would like to remark that for a better text processing an additional attribute is required. This attribute has to represent the correct spelling of the hyphenated word without any delimiter symbol

For transforming these annotations according to our TEI schema, we used our transforming technology FNTRANSFORM which is implemented in PROLOG. These transformations could also be processed using XSLT stylesheets.

6 Conclusions and Future Work

For retro–digitizing old printed German dictionaries, we have presented a workflow for parsing and annotating these text corpora according to the *Text Encoding Initiative*. With declarative programming techniques like EDCG's and FNQUERY, a fast and reliable parser could be implemented. Combined with transformation languages like FNTRANSFORM and XSLT, we are able to handle different types of *variation*, such as different types of entries, inflected forms, lemma variants, and flexible XML schemas. To exemplify these annotations, we have processed the Campe Dictionary with 6 volumes and over 140.000 different entries. The techniques, which we have applied to the German Campe Dictionary, could be used for handling other types of text copora as well, and of course also other languages like English or French.

In a current project, a web interface for a free community access is implemented for our toolkit as a *streaming editor*. With this editor, an easy to use graphical user interface gives access to a huge platform for parsing and annotating text corpora for the *eHumanities*, with the ability to reuse the already implemented parser for handling other text corpora. The declarative toolkit DDK, which includes all of the described frameworks, is available on the web.

A subject of future work will be the implementation of an XSLT preprocessor in PROLOG to provide a native interface for handling EDCG's within XSLT; the path language XPATH is already implemented in our XML toolkit FNQUERY.

References

ABRAMSON, H.; DAHL, V.: *Logic Grammars.* Springer, 1989

BLÜMM, M. (ed.): *Die textsortenspezifische Kernkodierung für Dokumente in TEI P5, TextGrid 2007, 2009.* http://www.textgrid.de/fileadmin/TextGrid/reports/Textsortenspezifische_Kernkodierung_2009.pdf, accessed 30.04.2009

CAMBRIDGE UNIVERSITY PRESS CATALOGUE, *A Dictionary of the English Language on CD–ROM.* http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521557658, accessed 30.04.2009

CAMPE, Joachim Heinrich: *Wörterbuch der deutschen Sprache.* 5 Volumes, Braunschweig 1807–1811

COVINGTON, M.A.: *GULP 3.1: An Extension of Prolog for Unification–Based Grammar.* Research

Report AI–1994–06, Artificial Intelligence Center, University of Georgia, 1994

DEREKO: *The German Reference Corpus Project.* http://www.ids-mannheim.de/kl/projekte/korpora/, 2009

FUCHS, N.E.; FROMHERZ, M.P.J.: *Transformational Development of Logic Programs from Executable Specifications – Schema Based Visual and Textual Composition of Logic Programs.* Beckstein, C.; Geske, U. (eds.), Entwicklung, Test und Wartung deklarativer KI–Programme, GMD Studien Nr. 238, Gesellschaft für Informatik und Datenverarbeitung, 1994

FUCHS, N.E.; SCHWITTER, R.: *Specifying Logic Programs in Controlled Natural Language.* Proc. Workshop on Computational Logic for Natural Language Processing (CLNP), 1995

GÄRTNER, K.; WISBEY, R.: *Die Bedeutung des Computers für die Edition altdeutscher Texte.* Kritische Bewahrung. Beiträge zur deutschen Philologie. Festschrift für Werner Schröder zum 60. Geburtstag. Hg. von Ernst–Joachim Schmidt, Berlin, 1974

GAZDAR, G.; MELLISH, C. *Natural Language Processing in Prolog. An Introduction to Computational Linguistics.* Addison–Wesley, 1989

HAUSMANN, F.J.; REICHMANN, O.; WIEGAND, H.E.; ZGUSTA, L. (eds.): *Wörterbücher / Dictionaries / Dictionnaires – Ein internationales Handbuch zur Lexikographie / An International Encyclopedia of Lexicography / Encyclopédie internationale de lexicographie.* Vol. 1 1989; Vol. 2 1990; Vol. 3 1991; Berlin et. al.

HIRAKAWA, H.; ONO, K.; YOSHIMURA, Y.: *Automatic Refinement of a POS Tagger Using a Reliable Parser and Plain Text Corpora.* Proc. 18th International Conference on Computational Linguistics (COLING), 2000

LANDAU, S.: *Dictionaries. The Art and Craft of Lexicography.* 2nd Edition, Cambridge, 2001

LLOYD, J.: *Practical Advantages of Declarative Programming.* CSLI Lecture Notes, Number 10, 1987

MCDERMOTT, A. (ed.): *Samuel Johnson. A Dictionary of the English Language on CD–ROM.* The First and Fourth Edition. Introduction and CD, Cambridge, 1996

O'KEEFE, R.A.: *The Craft of Prolog.* MIT Press, 1990

PEREIRA, F.C.N.; SHIEBER, S.M: *Prolog and Natural Language Analysis.* Lecture Notes, CSLI, Number 10, 1987

SCHNEIKER, C.; SEIPEL, D.; WEGSTEIN, W.; PRÄTOR, K.: *Declarative Parsing and Annotation of Electronic Dictionaries.* Proc. 6th International Workshop on Natural Language Processing and Cognitive Science (NLPCS), 2009

SCHWITTER, R.: *Working for Two: a Bidirectional Grammer for a Controlled Natural Language.* Proc. 28th International Conference on Artificial Intelligence (AI), 2008

SEIPEL, D.: *Processing* XML*–Documents in Prolog.* Proc. 17th Workshop on Logic Programmierung (WLP), 2002

SEIPEL, D.; PRÄTOR, K.: XML *Transformations Based on Logic Programming*. Proc. 18th Workshop on Logic Programming (WLP), 2005

SPERBERG–MCQUEEN, C. M.: *Logic Grammars and XML Schema*. Proc. Conference on Extreme Markup Languages, 2003.

TEI CONSORTIUM (eds.): TEI *P5: Guidelines for Electronic Text Encoding and Interchange.* 2 Vols. Oxford/Providence/Charlotteville/Nancy, 2008 http://www.tei-c.org/release/doc/tei-p5-doc/en/html/index.html, accessed 30.04.2009

TEXTGRID: *A Modular Platform for Collaborative Textual Editing – a Community Grid for the Humanities.* http://www.textgrid.de, 2009