Preliminary Specification, Design and Proof–of–Concept Implementation of a Portable Audio Concordance (PAC)

RFC 1.0

Thorsten Trippel, Nils Jahn, Dafydd Gibbon (U Bielefeld)
Soma Ouattara (U Cocody, Abidjan)

DOBES Technical Report *n'* (Ega)
(Status: RFC draft, January 2001 — printed January 8, 2001)

Contents

List of Figures

List of Tables

1 Overview

The Portable Audio Concordance (PAC) is a basic tool required for spoken language lexicography. The present tools is tailored to the needs of training local corpus analysists and lexicographers concerned with the documentation of endangered languages in their own countries using low–end hardware.

The tool is specified as a deliverable in the DOBES project *Ega: a documentation model for an endangered Ivorian language.*

It is anticipated that this tool will be used in a hybrid lingware development environment together with tools such as Praat or Transcriber and Shoebox. Compatibility with other tools is ensured by providing ASCII interfaces, in particular transcriptions in X–SAMPA and with XML markup, and by training local computer science personnel in automatic text processing techniques in order to interface tools in a hybrid environment.

The educational and local involvement aim has priority: the tool is not intended to be an industry standard implementation, but may of course be used a specification and proof–of–concept by other implementers.

We present a requirements specification, a system design specification, a project design specification, an implementation description, and an outline of the planned evaluation, distribution and maintenance policies. Object specifications in a standard format, and source code of a proof–of–concept implementation in Perl are included in the Appendices.

Perl is not an ideal language for software development but was selected because of built–in efficient regular expression processing capability, seamless integration with CGI and other interactive interfaces, and the portability of the code from UNIX/Linux to Windows and Macintosh environments, enabling rapid prototyping and efficient cyclical development.

This is a Request for Comments document. The specification is incomplete and we are aware that a number of errors and inconsistencies remain at the time of in–consortium distribution. Comments in the form of critique, corrections and suggestions for improvement and extension are welcome.

2 Portable Audio Concordance (PAC): Requirements

An important workhorse tool for language documentation, in particular for lexicography, is the *concordance.* There are many varieties of concordance, from the traditional ‘keyword in context’ concordance on paper to electronic hyperconcordances which are statically pre–compiled or dynamically compiled on the fly. To be useful with unwritten languages (and indeed all forms of spoken language), a concordance needs to include audio indexing. For these reasons, the design and proof–of–concept implementation of an audio concordance was included in our proposal *Ega: a document model for an endangered Ivorian language.*

The overall goal of the present activity is to present in relatively informal outline form

1. a minimal specification of requirements
2. a system design and a project design
3. a proof of concept implementation

for a portable audio concordance for use in lexicography within the framework of the documentation of endangered languages. This does not exclude utility in other contexts, but the present minimal specification is specifically formulated to ensure efficient initial language documentation. By ‘Portable Audio Concordance’ (PAC) we mean concordance shell software which can be used on as many platforms as possible, in particular in an offline laptop environment in the field or with low-end or older hardware.

The specific goals are

1. to provide a minimal, semi–formal specification and operationalisation as a proof–of–context implementation
2. with descriptions to enable other implementers to adopt and productise the system if wished
3. to provide sufficient information to make it easy to local computational personnel to maintain and modify the system
4. to be usable in local linguistic and computational linguistic training for the documentation of endangered languages

We proceed by specifying requirements in terms of first user groups, second application areas, third user needs.

2.1 Potential users, applications, needs

The potential groups for a audio concordance include:

- Language documentation activities:

  - Linguistic and anthropological fieldwork
  - Lexicography
  - Archiving
  - Database maintenance
- Language documentation training:
  - Students of phonetics and lexicography
  - Teachers of phonetics (used in teaching: example data)
- Access to documentation:
  - Researchers (audio examples and corpus)
  - Corpus linguists (looking for corpora)
  - Students of linguistics (studying different languages)
  - Engineers from language technology (speech recogniser, synthesizer, e.g. for applications for the blind)
  - Other interested persons and institutions

The main intended application of PAC is in the lexicography of endangered languages. Consequently, the main considerations are *ergonomic use by the language documenter* rather than other users:

- portability, e.g. by using common hypertext implementation techniques,
- simplicity of use,
- minimal necessary function (in this case, plain word access rather than access via morphosyntactic or semantic classes),
- ease of maintenance and extension by any computational linguists, in particular computational lexicographers,
- compatibility with other datasets and tools by the use of ASCII-based interfaces, especially for import and export, including normalised raw text formats and XML.

2.2 Technical Requirements

Our implementation perspective leans strongly towards practice in computational linguistics, and involves

- published design specifications for architecture, modules, interfaces, data structures, algorithms
- open source, possibly under GNU Public Licence (GPL)

- ISO (SGML) or *de facto* (XML, SAMPA, EAGLES) standards oriented, respect for best laboratory practice
- modular, library-oriented code rather than a single system

The present tool is straightforward and does not involve essentially new concepts; it takes up ideas and experience from the VerbMobil and EAGLES projects and applies them to the specific area of the documentation of endangered languages.

The further technical requirements for PAC are summarised informally as follows:

1. platform independence as far as possible
2. browser–neutral web–accessibility (including text browsers)
3. offline operational capability
4. non–proprietary ASCII storage and interchange formats for textual data in order to ensure ease of re–usability without extensive reverse engineering
5. *de facto* standard formats for non-textual data, e.g. WAV audio
6. full published documentation for design, evaluation, maintenance, extension and use
7. use of easily accessible high–level programming constructs (e.g. regular expressions)
8. compatibility with older low–level hardware found in many parts of the world.

2.3 Lingware requirements for lexicography

A specific lingware requirement is that PAC should be interfaced with

1. the microstructure of a computational lexicon or encyclopaedia with onomasiological, semasiological or other macrostructure
2. a corpus database management system.

Interfaces to standard ASCII based formats for text components of corpora and lexica need to be specified (e.g. widely used UNIX databases, archives with ASCII markup such as HTML, XML, Shoebox files).

By ‘corpus’ we mean a set of related primary language data sources, following EAGLES recommendations (cf. [Gibbon, Moore & Winski 1997]), including the following minimum

- audio, video and other signals recording communicative events
- transcriptions of recordings
- time–aligned annotations of recordings (including transcriptions and linguistic and other markup)

- corpus metadata including corpus wordlists (minimal lexica).

The following options may also be included:

- markup tables (interlinear gloss tables)
- corpus lexica and lexicon metadata
- corpus characterisations.

Finally, the audio-concordance should be as language independent as possible. This means in particular that it should take into acount more than one language and has to be extensible to other languages as long as the data are available in some standard way. This requirement applies to the *proof of concept* implementation.

3 Design specification

In this design specification we outline both system design and project design.

3.1 Concordance system design

3.1.1 Declarative considerations

We define a *Concordance System* from a declarative point of view as a pair of functions

$$ConcordanceSystem =< f_{acquisition}, f_{access} > \tag{1}$$

The acquisition function maps a corpus into a concordance consisting of a set of pairs of keyword and keyword–in–context set:

$$f_{aquisition} : Corpus \to Concordance \tag{2}$$

where

$$Corpus =< Signals, Annotations > \tag{3}$$

$$Concordance =< Key, Contextpairs > \tag{4}$$

$$Contextpair =< Text, Audio > \tag{5}$$

The aquisition function creates a list of *keys* from a given — possibly marked up — text. This list of *keys* are to be used as access criteria to the contexts of these keys, i.e. the *Key Words In Context* (KWIC).

The consultation function maps a pair of keys (often just one) and a corpus into a keyword–in–context concordance:

$$f_{consultation} : Keys \times Corpus \to KWIC \tag{6}$$

where

$$Keys = List \mid Regexp \mid LexicalLemma \tag{7}$$

The main dependencies involving both functions are illustrated in condensed form in Figure 1.

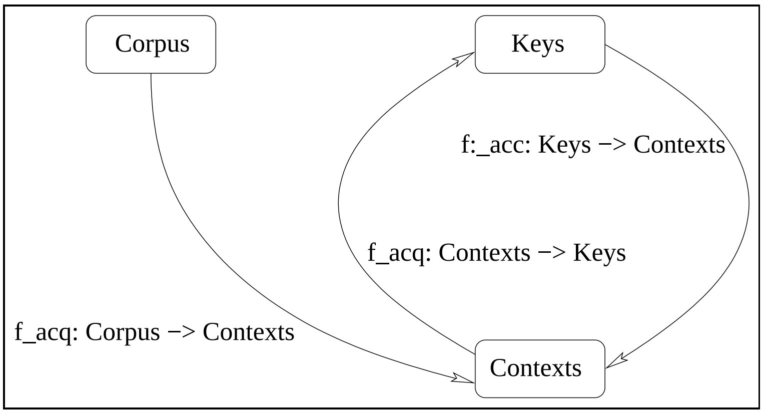


Figure 1: Main concordance dependencies.

3.1.2 Procedural considerations

Two different types of electronic concordance were taken into consideration from a procedural point of view:

1. The static concordance in which all links are predefined:
   - The main advantage of this solution is that it can be stored easily as an archive in a standard ASCII format and accessed with standard hypertext browser (HTML, XML) software. A static concordance can also be implemented manually if the amount of data is relatively small.
   - The main disadvantage of static concordances is that the set of keywords, context and keyword–context pairs is fixed during compilation.
2. The dynamic concordance, in which a keyword (or combination of keywords or phrases) from a wordlist, or a generalisation (such as a regular expression) is formulated and related on the fly to possible contexts in the corpus:
   - The main advantage of this solution is that the corpus can be extended rather simply, and that arbitrary queries can be formulated.
   - The main disadvantage of this solution is that a non–standardised program environment is needed in order to search the corpus database for keywords.

We note that there is a logical dependency between static and dynamic concordances: a static concordance is a subset of the output of a dynamic concordance. Consequently, PAC design starts with the dyamic concordance. The modules of the dynamic concordance are shown in Figure 2 and the architecture of a system designed to realise these functions is shown in Figure 3.

The hyperconcordance[1] which is the output of the software should be a browser accessible format. A user interface event should result on a list of occurrences of the keyword in context and an audio rendering of the context, including the keyword.

---

[1] Cf. the static hyperconcordance for T. S. Eliot's *Old Possum's Book of Practical Cats*:
`http://coral.lili.uni-bielefeld.de/Classes/Summer97/SemGS/WebLex/OldPossum/oldpossumlex/`
and the dynamic VM–HyprLex concordance:
`http://coral.lili.uni-bielefeld.de/VM-HyprLex`

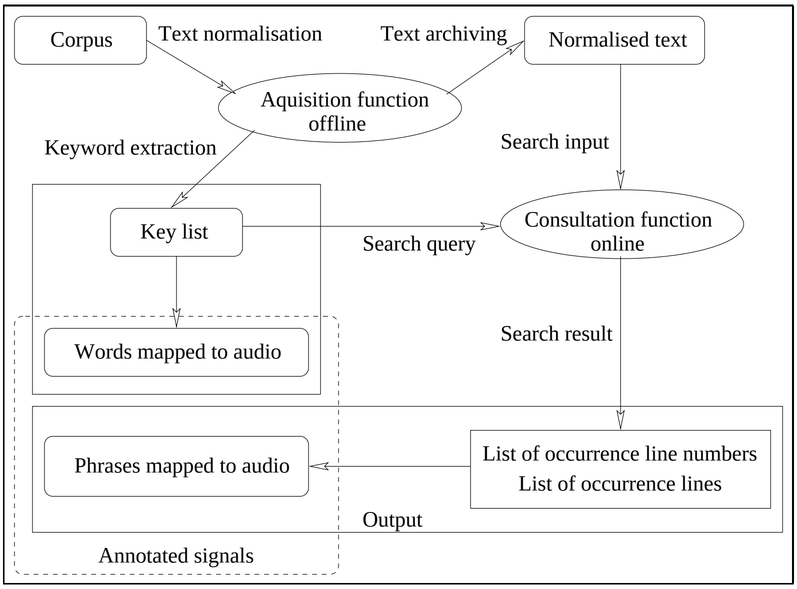


Figure 2: System modules and interface types.

3.1.3 Relation to lexicon microstructure

It is intended to use the concordance as a source of contextual lexical information within a lexicon, as lexicon as described by [Adouakou & Schulte 2000]. Further information on a microstructure for a suitable lexicon can be found in [Gibbon 2001].

The lexicon microstructure required for Ega, a tone language putatively with vowel harmony, consists of the following items:

- Phonology
  - Phonetic transcription
  - Consonant-Vowel pattern
  - Tone pattern
  - Syllable sequence condition
  - Consonant mutation
  - Elision
  - Vowel harmony: [± ATR] (Advanced Tongue Root)
- Morphology
  - Part of Speech
  - Inflexion
  - Pronoun
  - Nouns
  - Verbs
  - Word formation

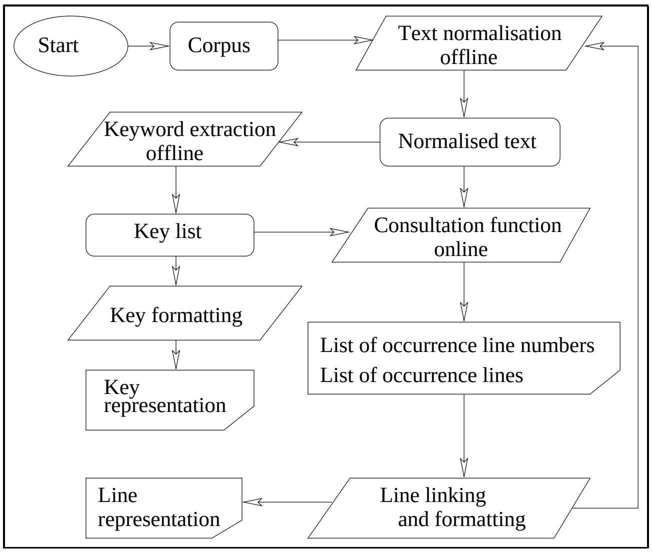

Start
Corpus
Text normalisation offline
Keyword extraction offline
Normalised text
Key list
Consultation function online
Key formatting
List of occurrence line numbers
List of occurrence lines
Key representation
Line representation
Line linking and formatting

Figure 3: Text handling dataflow.

  - Derivation
  - Compound
- Other
  - Sound
  - Picture
  - Domain
  - Definition
  - Concordance (i.e. contextualised examples)
  - Entry metadata

For the concordance a simpler microstructure subset is used:

- a head with administrative information on the text such as: title, author, date, date of changes and language.
- a body with annotated text:
  - the top level container element is for the text,
  - the next level is the sentence tag, which contains either the words or sentence end punctuation such as period, question marks, exclamation marks.

The tagging hierarchy for use with the concordance is shown in Figure 4.

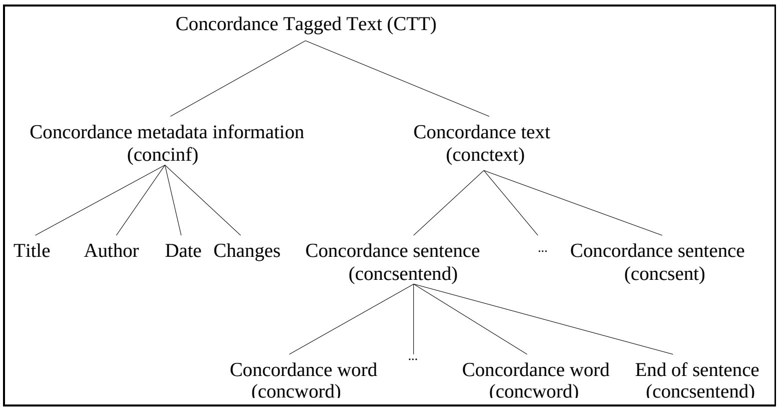

Concordance Tagged Text (CTT)
Concordance metadata information (concinf)
Concordance text (conctext)
Title
Author
Date
Changes
Concordance sentence (concsentend)
...
Concordance sentence (concsent)
Concordance word (concword)
...
Concordance word (concword)
End of sentence (concsentend)

Figure 4: Tree hierarchy for concordance tree tagging

3.2 Summary of modules and data sources

The system should require the following files:

- Audio files for all words, either one file per word and/or per sentence. For investigations on tonality in context the sentence might be preferred, for a standard canonized pronunciation one file per word might be included. At a later stage this could be replaced by a speech synthesis system based on phonetic transcriptions.
- Text file, ASCII formated as the standard input file.
- Marked up text file generated from the text file by a script but not on the fly (due to possible lack of computing power) but static. If this is done automatically in a later stage this could happen on the fly as well.
- Word list generated from the text file. For the word-list the same applies as for the marked up text, it should be generated statically with appropriate markup for hypertext presentation. This file should be the standard user entry to the system.

Required scripts/modules are:

- ASCII-text-to-marked-up-text converter
- Search function (input: word from the wordlist possibly by get or post operations of http), resulting in an output of sentence numbers and sentences of each occurance of a word searched for (output with appropriate markup for hypertextual presentation and link to the original text and audio files)

Generated, static files include:

- Marked up text
- Wordlist

Generated, not static files include:

- list of all occurances of a searched for word with sentence number and sentence

Three user interfaces are being included:

1. Command line (exists)
2. Online (CGI server, standard browser client)
3. Offline (widget supporting standalone environment)

Further interface design specifications will follow in a later version of this document.

3.2.1 Required resources

In order to test the functionality, the PAC system was subjected to initial informal evaluation using two different Ivorian languages: Koulango (Gur/Senoufo), Anyi (Kwa/Tano).

Currently the first Ega data is arriving from Abidjan and is being incorporated into the evaluation.

Further corpus data specifications will follow in a later version of this document.

3.3 Concordance subproject structure

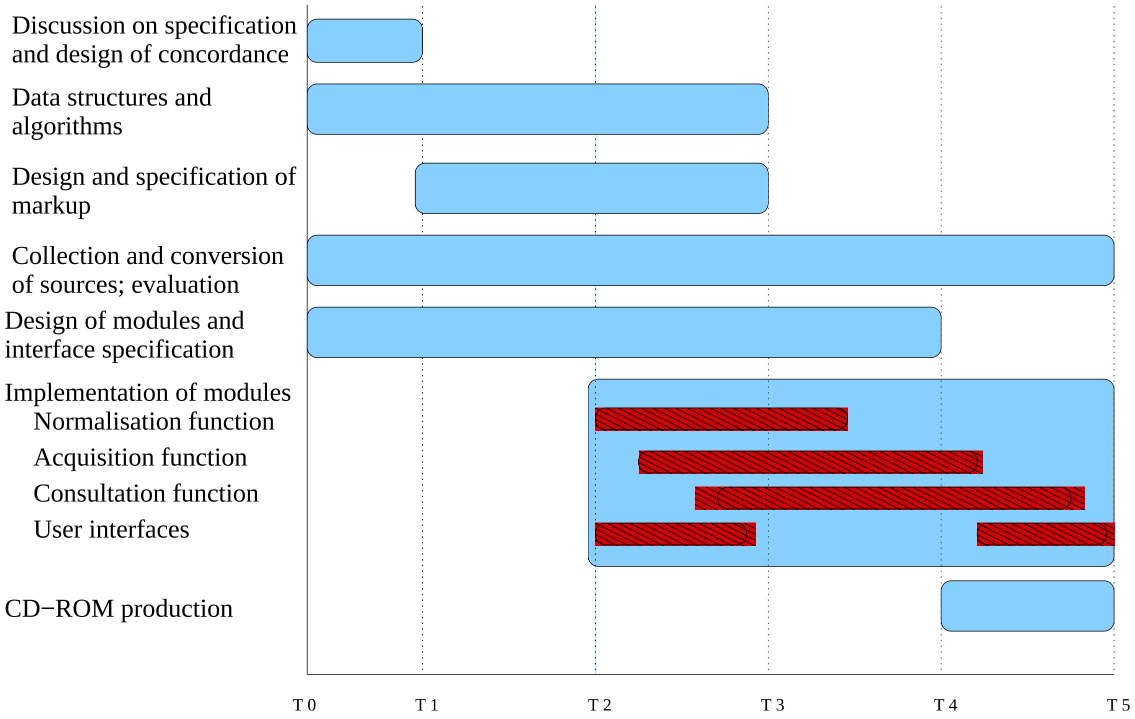


Figure 5: Time management bar chart.

Table 1: Task assignment table

Task	Who
Specification and design for an audio-concordance	Trippel, Ouattara, Jahn
Specification and design for an audio-concordance	Trippel, Ouattara, Jahn
Design and definition of markup	Trippel, Ouattara, Schulte
Coordination, collation	Trippel, Schulte, Adouakou
Evaluation	Trippel, Adouaou
Module definition	Trippel, Ouattara, Jahn
ASCII to Markup converter	Trippel, Ouattara, Jahn
Search function	Ouattara, Jahn
User interface design	Trippel, Jahn
CD-Rom production	Trippel, Adouakou

3.3.1 Task definition

The following main tasks have been defined:

1. General issues in specification and design for an audio concordance
2. Specification and design documentation
3. User interface design
4. Coordination with related projects on lexicon microstructure
5. Coordination with related projects on resources: text files, audio files
6. Module definition
7. Basic concordance markup XML DTD
8. Specification of data structures
9. Specification of algorithms
10. Implementation of modules
11. Evaluation with Anyi, Koulango, Ega
12. CD-Rom production

3.3.2 Time and personnel management

The tasks are coordinated closely (and some shared) with the DAAD project *Encyclopaedia Design for Ivory Coast Languages* until the end of that project (December 2000).

4 Implementation

The basic conditions for implementation are as follows:

1. to accomplish platform independence and reusability the main (but not exclusive) interchange format for the texts and lexicon is XML; an XML DTD has been defined for the concordance
2. for convenience the audio formats is initially the wide-spread WAV which is usable on most platforms and which can easily be converted into other formats
3. graphics are in standard formats such as JPEG, GIF, TIFF, EPS
4. documentation is in LaTeX, with automatic conversion into HTML and PDF
5. implementation is in Perl because of the availability of efficiently implemented regular expression based search
6. pending the availability of an appropriate XML processor, markup is pre–processed into single–character–based ASCII conventions for efficient treatment

4.1 DTD for Concordance Tagged Text (CTT)

The design of the 'container tree' of elements and tag types was specified above (cf. Figure 4): a text element is the container element for sentences, which are in turn container elements for words and sentence-end punctuation such as periods, question marks, exclamation marks. These are included because they have a semantic function for orthographic texts. Tone–language–specific prosodic markup will be included at a later stage.

The DTD is deliberately minimal and subject to revision with respect to the distinction between elements and attribute–value pairs in consultation with other teams.

```
<!-- DTD for the concordance markup Concordance Tagged Text (CTT) -->
<!-- Developed 2000 by Thorsten Trippel, Soma Outtara, Nils Jahn
at the University of Bielefeld, Germany -->

<!-- root element is ctt -->
<!ELEMENT ctt - - (concinf, conctext)>

<!-- head element with general information -->
<!ELEMENT concinf - - (title, author, date, changes*)>
<!ELEMENT title - - (#PCDATA)>
<!ELEMENT author - - (#PCDATA)>
<!ELEMENT date - - (#PCDATA)>
<!ELEMENT language - - (#PCDATA)>
<!ELEMENT changes - - (#PCDATA)>

<!-- body element with marked up text -->
<!ELEMENT  conctext  - - (concsentence)*  >

<!-- Element to tag single sentences with id attributes -->
```

```
<!ELEMENT  concsentence - - ((concword+),concsentend) >
<!ATTLIST concsentence  sentencenumber ID #REQUIRED>

<!-- Element to tag single words with id attributes -->
<!ELEMENT concword - - (#PCDATA)>
<!ATTLIST concword wordnumber ID #REQUIRED>

<!-- Element to tag sentence end punctuation such as . ! ? -->
<!ELEMENT concsentend - - (#PCDATA)>
```

The following sample text illustrates the CTT format:

```
<?xml version="1.0" standalone="no"?>

<!DOCTYPE ctt public "-//UBI//DTD CONCORDANCE 0.1a//EN" >

<ctt>

<concinf>
<title>Testtext</title>
<author>Trippel</author>
<date>08 Oct 2000</date>
<language>English</language>
<changes>08 Oct 2000</changes>
</concinf>

<conctext>
<concsentence sentencenumber="sentence1">
<concword wordnumber="word1">This</concword>
<concword wordnumber="word2">is </concword>
<concword wordnumber="word3">sentence</concword>
<concword wordnumber="word4">1</concword>
<concsentend>.</concsentend>
</concsentence>
</conctext>

</ctt>
```

4.2 Normalisation function

The normalisation function converts a SAMPA text into a marked-up text. Every sentence receives a unique identification number. Within the sentences each word receives an identification which is composed of the number of the current sentence and the number of its position in the sentence. The SAMPA text does not contain any punctuation and the end of a sentence is marked up with the line-feed symbol. The normalisation function will be invoked once per text.

The normalisation function (cf. Table 2) expects two arguments which are the name of the input file and the name of the output file. The read line of the input file is stored in a string variable. The line of the file will be splitted into an array, and two integer variables are used as index variables. The first index will be the number of the current sentence and the second the number of the current word of the sentence.

Table 2: Pseudocode for normalisation function.

```
normalisation(input, output)
openfile(input)
openfile(output)
print output, header-markup
word-index ←1
sentence-number ←1
while input-line not end-of-file do
    line-array ← split input-line at blank
        if length(line-array) > 0
            then print output sentence-markup + sentence-number
            foreach word in line-array do
                    print output word-markup + word-index + sentence-number
                    increase word-index
            end foreach
            print output sentence-markup
        end if
        word-index ← 1
        if length(line-array) > 0
            then increase sentence-number
        end if
end while
print output bottom-markup
closefile(input)
closefile(output)
```

The following algorithm first opens the input and the output file. The first thing which will be written into the output file is the markup header, i.e. root element and information about title, author, and date of the text. The two index variables are initialised and the input file is read line by line. Each line is split into an array and the the sentence and the words of the sentence are marked up with tags and also receive identification numbers which are provided by the index variables sentence-number and word-index. When the end of the input file is reached the end tags are written into the output file and the files are closed.

For the provisional proof–of–concept code see Appendix B.

4.3 Acquisition function

keywords which occur in the input texts. The list is in alphabetical order and does not contain any double occurences. The list is in ASCII and each word is seperated by a line feed symbol.

The acquisition module searches the data directory and processes each file in this directory. The module first stores the files of the mentioned directory in an array. Then a file is opened, its contents are read line by line and the extracted words are stored in an array. After that the file is closed. This procedure is repeated until all files in the directory have been processed.

The array which contains the words of all files is sorted alphabetically. After that the first element of the array is copied into another array. The module checks if the next element is equal to the one just copied. If it is equal it checks the next one, if it is not equal it is copied

Table 3: Pseudocode for keyword extraction function.

```
normalisation(input, output)
array ← empty
sorted-array ← empty
unique-array ← empty
directory ← read-current-directory()
index ←0
current ←0
for file in directory do
    openfile(file)
    while(file not eof) do
        line ←readline(file)
        word ←extractword(line)
        append(word, array)
    end while
    closefile(file)
end for
sorted-array ←sort(array)
while(index <= length(sorted-array) do
    append(sorted-array[index],unique-array)
    increase(index)
    while(unique-array[current] not equal sorted-array[index]
        increase(index)
    end while
    increase(current)
end while
openfile(output)
list ←convert sorted-array into string
write(output,list)
closefile(output)
```

to the next array. This is repeated until each element of the first array has been checked. The second array consists of unique words. The array is split into a string variable and each word is separated by a newline and stored into the target file.

For the provisional proof–of–concept code see Appendix D.

4.4 Consultation function

The consultation module searches a given text for a specific keyword. If the keyword is found all sentences are shown in which the desired keyword occures. In case there are no matches it returns the message that no matches were found. It is also possible to search more than one text at the same time, but the results are sorted according the source texts.

The consultation module expects as arguments the keyword to be searched, the output file where the results will be stored and a list of input files which could be as long as necessary.

The module first checks whether the number of arguments is less than 3. In that case it terminates and returns an error message to the user and prints out how which arguments are expected. The first input and the output file is opened. The whole input file is stored into one variable.

The whole variable is searched for a specific pattern and the line number and the sentence are stored in variables.

Then the variable which contains the sentence is searched whether it contains the keyword and if it contains it the whole sentence with the line number is written to the output file. This is repeated until the whole input file is checked, then the input file is closed. Then the next file from the input list is processed. This is repeated until the whole list is processed. At the end the module checks whether there were any matches, if not a message is returned to the screen. The output file is closed.

Table 4: Pseudocode for consultation function.

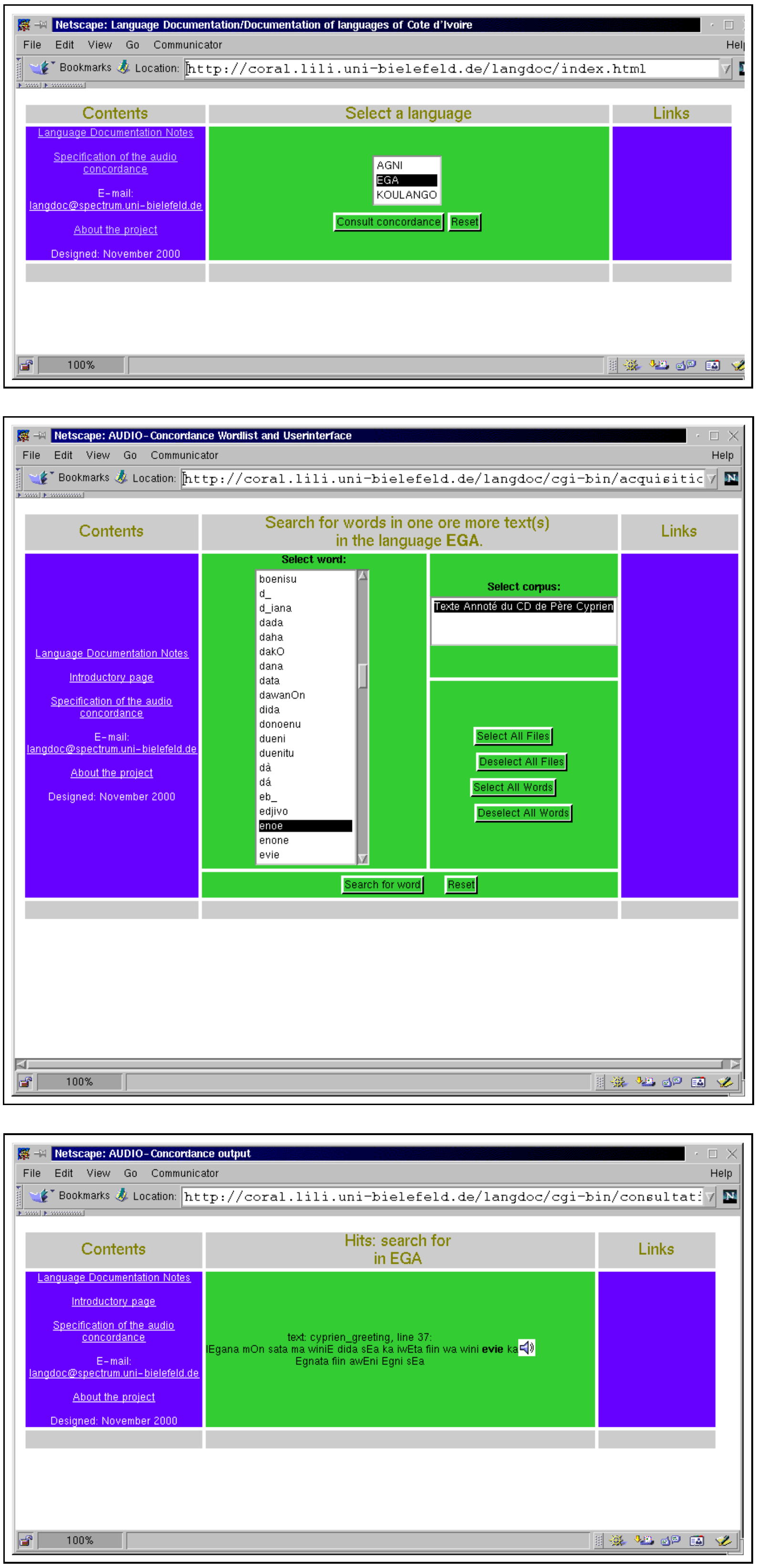
```
consultation(keyword,outputfile,list-of-inputfiles)
openfile(outputfile)
found ←false
for inputfile in list-of-inputfiles do
    openfile(inputfile)
    text ←readfile(inputfile)
    for sentence in text do
        if keyword is in sentence do
            print output sentence
            found ←true
        end ifend for  closefile(inputfile) end for if found not equals true do
    print "no matches found!"
end if
closefile(outputfile)
```

The consultation function is frontended with CGI-interaction for user access. After selecting a language from a pick-list on the introductory page (see figure 6), two pick lists enable users to select a word and a corpus where a context could come from that is language specific. It is also possible to select all corpora at the same time.

After the consultation request a list of occurences with accompanying line numbers and contexts are given. All of this is generated *on the fly.*

Three user interfaces are being incorporated; design and implementation (except for command line access) is still in process. The current implementation of the graphical user-interface forms is shown in figure 6.

For provisional proof–of–concept code see Appendix F.



Netscape: Language Documentation/Documentation of languages of Cote d'Ivoire

`http://coral.lili.uni-bielefeld.de/langdoc/index.html`

Contents

Language Documentation Notes

Specification of the audio concordance

E-mail:
langdoc@spectrum.uni-bielefeld.de

About the project

Designed: November 2000

Select a language

AGNI
EGA
KOULANGO

Links

Netscape: AUDIO-Concordance Wordlist and Userinterface

`http://coral.lili.uni-bielefeld.de/langdoc/cgi-bin/acquisitio`

Contents

Language Documentation Notes

Introductory page

Specification of the audio concordance

E-mail:
langdoc@spectrum.uni-bielefeld.de

About the project

Designed: November 2000

Search for words in one ore more text(s)
in the language **EGA**.

Select word:

boenisu
d_
d_iana
dada
daha
dakO
dana
data
dawanOn
dida
donoenu
dueni
duenitu
dà
dá
eb_
edjivo
enoe
enone
evie

Select corpus:

Texte Annoté du CD de Père Cyprien

Links

Netscape: AUDIO-Concordance output

`http://coral.lili.uni-bielefeld.de/langdoc/cgi-bin/consultat:`

Contents

Language Documentation Notes

Introductory page

Specification of the audio concordance

E-mail:
langdoc@spectrum.uni-bielefeld.de

About the project

Designed: November 2000

Hits: search for
in EGA

text: cyprien_greeting, line 37:
lEgana mOn sata ma winiE dida sEa ka iwEta fiin wa wini **evie** ka
Egnata fiin awEni Egni sEa

Links

Figure 6: Basic user interface forms.

5 Evaluation, distribution, maintenance

The testing programme follows EAGLES recommendations for language and speech technology ([Gibbon, Mertins & Moore 2000]) and involves

1. structural testing (glass box Testing, 'diagnostic testing')
2. functional testing (black box testing, 'evaluation')
3. field testing (black box testing, 'assessment') with different languages (Anyi, Koulango, Ega); tests are being performed in Bielefeld and Abidjan on
   - client platforms: Linux, Solaris, various Win32 versions
   - client browsers: Opera (preferred), NN, MS-IE (but under WinNT MS-IE failed to render multiply embedded tables)
   - server: Linux, Solaris; not yet with Win32

The software and documentation is distributed continuously within the Ega project between Bielefeld and Abidjan, and the present document makes it available in preliminary form to the partners in the DOBES project.

The documentation will simultaneously be made available on the Ega project website.

References

[Adouakou & Schulte 2000] Adouakou, Sandrine & Michaela Schulte (2000). Sprachen der Elfenbeinküste. Universität Bielefeld.

[Gibbon, Moore & Winski 1997] Gibbon, Dafydd, Inge Mertins & Roger Moore, eds. (1997). *Handbook of Standards and Resources for Spoken Language Systems.* Berlin: Mouton de Gruyter.

[Gibbon, Mertins & Moore 2000] Gibbon, Dafydd, Inge Mertins & Roger Moore, eds. (2000). *Handbook of Multimodal and Spoken Dialogue Systems: Terminology, and Product Evaluation.* Dordrecht/New York: Kluwer Academic Publishers (Chapter 4).

[Gibbon 2001] Gibbon, Dafydd (2001). On lexical objects and their properties. A contribution to the 'MetaLex' requirements specification for spoken language lexicon documentation. Universität Bielefeld: DOBES Technical Report *n* (Ega)

Appendices:

Object specification tables

Proof–of–concept PERL code

A Object specifications

Initial object specifications follow for

1. PAC - Portable Audio Concordance
2. Text normalisation module
3. Keyword extraction and formatting module
4. Consultation query and response module
5. User interface module

Object name

PAC - Portable Audio Concordance

Services description

PAC is designed to aid corpus lexicographers with low–technology equipment in the documentation of endangered languages.
The present documentation covers an initial specification and proof–of–concept implementation.
PAC has the following functionality:

1. Mapping of SAMPA annotated audio corpus to keywords + text/audio contexts (acquisition)
2. Mapping of keywords into text/audio contexts (access)
3. Corpus transcription normalisation in XML and ASCII DB import formats
4. Audio indexing from words and sentences into signal files
5. Keyword extraction from corpus and pick list formatting
6. KWIC output formatting in XML/HTML
7. Online and offline operation with standard browsers

Service calls

1. Text normalisation module
2. Keyword extraction and formatting module
3. Consultation query module and response module
4. User interface module

Parameter data formats

Input:	phrase–chunked speech files, (multi-tier) SAMPA annotations
Intermediate:	keywords, normalised ASCII SAMPA text with XML markup
Output:	Hypertext (XML/HTML) formatted text/audio query results

Object name

Text normalisation module

Services description

The module maps

1. an X–SAMPA transcription
2. into a normalised archivable text
3. with XML markup

The marked–up text serves as the input to the KWIC (KeyWord in Context) search function.

Service calls

1. XML DTD for Concordance Tagged Text (CTT)
2. SAMPA specification
3. corpus access file handling

Parameter data formats

Input:	time–stamped phrase–chunked X–SAMPA transcription
Intermediate:	—
Output:	XML marked up normalised X–SAMPA transcription

Object name

Keyword extraction and formatting module

Services description

The keyword extraction module maps a normalised text into a set of keywords for mapping to a lexicon and for consultation queries to the KWIC consultation function.

1. keyword set
2. markup for GUI picklist widget
3. marked–up GUI picklist widget

The keywords are also intended to be generalised to regular expressions for generalised search.

Service calls

1. Tree–traversal function for XML Concordance Tagged Text
2. XML DTD for picklist

Parameter data formats

Input:	XML Concordance Tagged Text
Intermediate:	Line separated ASCII SAMPA formatted keyword set
Output:	XML marked up picklist widget

Object name

Consultation query and response module

Services description

The consultation query module maps a keyword (set) and a normalised text with Computer Tagged Text XML markup into a set of transcription and audio context pairs in which the keyword occurs.

1. set of transcription/audio pairs
2. with GUI oriented XML/HTML formatting of transcription
3. audio file link
4. concordance statistics about keyword

Service calls

1. keyword set access
2. normalised XML formatted text archive access
3. search optimisation of XML format
4. context search
5. audio file linking

Parameter data formats

Input:	XML formatted keyword list and XML formatted text
Intermediate:	search optimised text format
Output:	GUI formatted KWIC keyword–textcontext–audiocontext triples

Object name

User interface module

Services description

Four user interfaces are provided:

1. Dynamic concordancance with access via command line (for development and for basic DOS and VT100 type terminal interaction)
2. Dynamic concordance for standard HTML (or XML) browser client interacting online via HTTP with CGI and server–side application
3. Static (pre–compiled) concordance for standalone HTML (or XML) browser client
4. Dynamic concordance for standalone application with standard GUI interface (in the proof–of–concept version: Perl/Tk)

Service calls

All (G)UI functions have service calls to all other modules.

Parameter data formats

Input:	Outputs of all other modules
Intermediate:	—
Output:	Events to trigger all other modules; (G)UI forms

B Normalisation function

```
#!/vol/bin/perl -w

# normalisation.pl
# version: 0.9b
# N. Jahn, S. Ouattara, T. Trippel
# November 2000, University of Bielefeld, Germany
# [jahn,soma,ttrippel]@spectrum.uni-bielefeld.de

# Functionality: for a given line it ennumerates the line,
# breaks it into words, gives every word a unique identifier.

# Syntax: normalisation.pl <INFILE> <OUTFILE>

# Additional information: the user will be prompted for
# title, author, date of and changes to the document

($input, $output) = @ARGV ;  #store the arguments in $input and $output

if ($#ARGV < 1) { #if there are less than 2 arguments, tell the user and exit program
    print "usage: normalisation.pl <input> <output>\n" ;
    exit ;
}

open (IN, "< $input") #open $input for reading access
or
        die "\n Input file couldn't be opened!!\n" ;

open (OUT, "> $output") #open $output for writing access
or
        die "\n Output file couldn't be created!!\n" ;

print "Please give the title: \n";
chomp($title = <STDIN>);
print "Please give the authors name: \n" ;
chomp($author =<STDIN>);
print "Please give the date when the text was created:\n";
chomp($date = <STDIN>);
print "Please give the language of the text:\n";
chomp($language = <STDIN>);
print "Please give the changes to the text:\n";
chomp($changes = <STDIN>);

print OUT qq%<?xml version="1.0" standalone="no"?>\n% ;  #print document markups into $output
print OUT "<!DOCTYPE ctt public \"-//UBI//DTD CONCORDANCE 0.1a//EN\" >\n\n";
print OUT "<ctt>\n\n" ;

print OUT "<concinf>\n";
print OUT "<title>$title</title>\n";
print OUT "<author>$author</author>\n";
print OUT "<date>$date</date>\n";
print OUT "<language>$language</language>\n";
print OUT "<changes>$changes</changes>\n";
print OUT "</concinf>\n\n";

print OUT "<conctext>\n";
$count = 1 ;
```

```
$line = 1 ;
$word = 0 ;

while(<IN>) {
     chop ; # deletes the last character of the line
     @array = split(" ", $_) ; # splits the words into an array
     if ($#array > 0) { # checks for non-empty lines
 print OUT "<concsentence sentencenumber=\"$line\">\n" ;
 foreach $word (@array) { # for each word of the current line do
     print OUT "<concword wordnumber=\"word$count.$line\">$word<\/concword>\n";
     $count++ ;
 }
 print OUT "<\/concsentence>\n" ;
     }
     $count= 1;
     $line++ if ($#array > 0) ; # increments the line number by one if the line is non-empty
}

print OUT "<\/ctt>\n" ;

close(IN) ;
close(OUT);
```

C Normalisation function with basic TK GUI

```
#!/vol/bin/perl

use Tk ;

sub openError {
    $nf = MainWindow->new() ;
    $nf->Frame(-label => "\nNo input or output file specified!\n")->pack() ;
    $nf->Button(-text => "ok", -command =>sub {$nf->withdraw()})->pack() ;
}

sub norm {
    $mw = shift ;
    $input = $e5->get() ;
    $output = $e6->get() ;
    $title = $e1->get() ;
    $author = $e2->get() ;
    $date = $e3->get() ;
    $changes = $e4->get() ;

    if ($input eq undef) {
openError ;
    }

    open (IN, "< $input")
or
    die "Input file couldn't be opened!\n" ;

    open (OUT, "> $output")
or
    die "Output file couldn't be opened!\n" ;

    print OUT qq%<?xml version="1.0" standalone="no"?>\n% ;
    print OUT "<!DOCTYPE ctt public \"-//UBI//DTD CONCORDANCE 0.1a//EN\" >\n\n";
    print OUT "<ctt>\n\n" ;

    print OUT "<concinf>\n";
    print OUT "<title>$title</title>\n";
    print OUT "<author>$author</author>\n";
    print OUT "<date>$date</date>\n";
    print OUT "<changes>$changes</changes>\n";
    print OUT "</concinf>\n\n";

    print OUT "<conctext>\n";
    $count = 1 ;
    $line = 1 ;
    $word = 0 ;

    while(<IN>) {
chop ; # deletes the last character of the line
@array = split(" ", $_) ; # splits the words into an array
if ($#array > 0) { # checks for non-empty lines
    print OUT "<concsentence sentencenumber=\"$line\">\n" ;
    foreach $word (@array) { # for each word of the current line do
print OUT "<concword wordnumber=\"word$count.$line\">$word<\/concword>\n";
$count++ ;
    }
    print OUT "<\/concsentence>\n" ;
}
$count= 1;
$line++ if ($#array > 0) ;
# increments the line number by one if the line is non-empty
    }
    print OUT "<\/ctt>\n" ;
    close(IN) ;
    close(OUT);
    exit ;
}

$mw = MainWindow->new() ;
$mw->Label(-text => "Title")->pack() ;
$e1 = $mw->Entry()->pack() ;
$mw->Label(-text => "Author")->pack() ;
$e2 = $mw->Entry()->pack() ;
$mw->Label(-text => "Date")->pack() ;
$e3 = $mw->Entry()->pack() ;
$mw->Label(-text => "Last changes")->pack() ;
$e4 = $mw->Entry()->pack() ;
$mw->Label(-text => "")->pack() ;
$mw->Label(-text => "Input File")->pack() ;
$e5 = $mw->Entry()->pack() ;
$mw->Label(-text => "Output File")->pack() ;
$e6 = $mw->Entry()->pack() ;
$mw->Label(-text => "")->pack() ;
$mw->Button(-text => "Normalise Text", -command => \&norm)->pack() ;
$mw->Button(-text => "Quit", -command => sub {exit})->pack() ;
MainLoop;
```

D Word list extraction

```
#!/vol/bin/perl

#authors Jahn & Ouattara
#Program : acquire
#gets as input a <ctt> text, then creates a key wordlist and sorts it automatically

sub getDir {
opendir(ETC, "/project/langdoc/SOFTWARE/CONCORDANCE/DATA/")
   or
die "Cannot open it!" ;

while ($toc = readdir(ETC))  {
if ($toc =~ m/\S+?\.ctt/g) {
push(@inh, $toc) ;
}
}
        closedir(ETC);
return @inh ;
}

@array = () ;
@narray = () ;
@dir = getDir() ;
print @dir ;
foreach $datei (@dir) {

    open(DATEI, "< /project/langdoc/SOFTWARE/CONCORDANCE/DATA/$datei") ;

    while (<DATEI>) {
if (m/<concword wordnumber=\"word\d+\.\d+\">(\S+?)<\/concword>/g) {
push(@array, $1) ;
        } #extracts a word and pushes it onto an array
    }

    close(DATEI) ;
}

@narray = sort(@array) ; #sorts the array

$index = 0 ;
$current = 0 ;
@rarray = () ;


while($index <= $#narray){ #compares index to the length of the array
push(@rarray, $narray[$index]) ; #pushes word onto the array
        $index++ ;
        while ($rarray[$current] eq $narray[$index]){
      $index++ ; #skips equal words
        }
$current++;
}

open(OUT, "> /project/langdoc/SOFTWARE/CONCORDANCE/DATA/wortliste.wl") ;
print OUT join("\n", @rarray) ; # converts the array into string
close(OUT) ;
```

E CGI interaction for wordlist extraction

```
#!/vol/bin/perl -w

use CGI qw(:standard);

my $language = param("language"); #the language to be investigated
#my $language = "AGNI";
$defaultpath= "../html-data/DATA/"."$language"."/";

# $_=$DATEI;
#s/\/project\/langdoc\/SOFTWARE\/CONCORDANCE\/DATA\///;
#s/\.ctt//;
#$filename=$_;

%titlefile =();
@array = () ;
@narray = () ;
@dir = getDir() ;
# print @dir ;
foreach $datei (@dir) {

    open(DATEI, "< /project/langdoc/SOFTWARE/CONCORDANCE/DATA/$language/$datei") ;

    while (<DATEI>) {
        if (m/<concword wordnumber=\"word\d+\.\d+\">(\S+?)<\/concword>/g) {
                        push(@array, $1) ;
        } #extracts a word and pushes it onto an array
elsif (/<title>\w(.*)<\/title>/){
s/<title>//;
s/<\/title>//;
$title=$_;
$titlefile{"$datei"}=$title;
}

}
close(DATEI);
}

@narray = sort(@array) ; #sorts the array

$index = "0" ;
$current = "0" ;
@rarray = () ;

while($index <= $#narray){ #compares index to the length of the array
        push(@rarray, $narray[$index]) ; #pushes word onto the array
        $index++ ;
        while ($rarray[$current] eq $narray[$index]){
              $index++ ; #skips equal words
        }
        $current++;

}

print  header();

print  <<END_of_HEAD;
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html>
<head>
<title>AUDIO-Concordance Wordlist and Userinterface</title>
<link rev="MADE" href="mailto:ttrippel\@spectrum.uni-bielefeld.de" />
<base href="http://coral.lili.uni-bielefeld.de/langdoc/cgi-bin/acquisition.pl" />
<meta name="copyright" content="University of Bielefeld, Computational Linguistics and Spoken Language" />
<meta name="author" content="Thorsten Trippel" />
<meta name="description" content="Wordlist for the audio concordance and Userinterface" />
<meta name="date" content="23 Nov 2000" />
<link rel="stylesheet" href="../langdoc.css" />
<script language="JavaScript">

<!--

function doList() {
counter=0;

for(var i=0;i<document.forms["consultationstart"].infile.options.length;i++) {
  if(document.forms["consultationstart"].infile.options[i].selected) {
    counter++;
  }
}

    if(counter==0){
       alert("Don't want to proceed?\\n There are no files selected!");
return;
                  }
     else {document.forms["consultationstart"].submit();}

function select_all(formList) {
  for (var i = 0; i < formList.options.length; i++) {
    formList.options[i].selected =true;
  }
}
function deselect_all(formList) {
  for (var i = 0; i < formList.options.length; i++) {
    formList.options[i].selected =false;
  }
}

// -->
</script>

</head>
<body link="#ffffff" vlink="#fafafa" alink="ff0000">
END_of_HEAD

@alltitles= values(%titlefile);

print  <<HEAD_of_TABLE;
<form name="consultationstart"
  action="http://coral.lili.uni-bielefeld.de/langdoc/cgi-bin/consultation.pl"
  method="post">
<input type="hidden" name="language" value="$language" />
<table class="intern" >
```

```
<tr><!-- 1.Reihe  leer nur leere Bilder -->
<td class="background" width="137">
<img src="../IMAGES/1pix.gif" width="1" height="1" alt="" hspace="68" vspace="1" /></td>
<td class="background" width="300%" colspan="3"><img src="../IMAGES/1pix.gif"
    width="1" height="1" alt="" hspace="137" vspace="1" /></td>
<td class="background" width="137"><img src="../IMAGES/1pix.gif"
    width="1" height="1" alt="" hspace="68" vspace="1" /></td>
</tr>

<tr><!-- 2. Reihe Tabellenueberschriften -->
<td class="tablehead">Contents</td>
<td class="tablehead" colspan="3">Search for words in one ore more text(s)
<br />
in the language <b>$language</b>.</td>
<td class="tablehead">Links</td>
</tr>
<tr> <!-- 3. Reihe, das ist die erste Reihe des Tabelleninhalts -->
<td class="content" rowspan="3">
<p><a href="../LangDoc/index.html">Language Documentation Notes</a></p>
<p><a href="../index.html">Introductory page</a></p>
<!-- <p><a href="../acquisition.pl">Search the concordance</a></p> -->
<p><a href="../SPECIFICATION/">Specification of the audio concordance</a></p>
<p>E-mail: <a href="mailto:langdoc\@spectrum.uni-bielefeld.de">langdoc\@spectrum.uni-bielefeld.de</a></p>
<p><a href="../about.html">About the project</a></p>
<p>Designed: November 2000</p>
<!-- <img src="../IMAGES/1pix.gif"
    width="1" height="1" alt="" hspace="1" vspace="100" /> -->

</td>
<td class="body" rowspan="2">

<!-- <table class="intern" border="0"  cellspacing="0" cellpadding="0"> -->

HEAD_of_TABLE

print  <<SELECT_END;

<!-- <tr>
<td class="body" rowspan="2" > -->
<strong>Select word:</strong><br />

<select name="word" size="20" multiple="multiple">

SELECT_END

$word="0";
for ($word=0;$word<=$#rarray;$word++){
print  "<option value=\"$rarray[$word]\">$rarray[$word]</option>\n"
}

print  <<CONTENT_START;
</select>
</td>
<td align="center" class="body" colspan="2" rowspan="1">
<strong>Select corpus:</strong><br />

<select name="infile" size="3" multiple="multiple">

CONTENT_START
```

```
while (($file,$title) = each(%titlefile)){

print "<option value=\"$defaultpath$file\">$title</option>\n";

}
print  <<CONTENT_END;
</select>
</td>
<td class="linklist" rowspan="3">

&nbsp;
</td>
  </tr>

<tr> <!-- 4. Reihe, Uebersicht und linkliste sind verbraucht,
2. Spalte auch  bleibt noch Spalte 3 und 4 -->

<td align="center" class="body" colspan="2">
<input type="button" value="Select All Files" onclick="select_all(form.infile)" />
&nbsp;&nbsp;&nbsp;&nbsp;
<!-- </td>

<td align="center" class="body" > -->
<input type="button" value="Deselect All Files" onclick="deselect_all(form.infile)" />
<br /><input type="button" value="Select All Words" onclick="select_all(form.word)" />
&nbsp;&nbsp;&nbsp;&nbsp;
<!-- </td>

<td align="center" class="body" > -->
<input type="button" value="Deselect All Words" onclick="deselect_all(form.word)" /></td>
</tr>

<tr><!-- 5. Reihe, Uebersicht und linkliste sind verbraucht, Rest noch nicht -->
<td align="center" class="body" colspan="3">
<input type="button" value="Search for word" onclick="doList(form)" />
<!-- <input type="submit" value="Search for word"  / >
</td>
<td align="center" class="body" >-->&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;<input type="reset" value="Reset" /></td>
</tr>

CONTENT_END

print  <<END_of_TABLE;
<!-- </table>

</td>

</tr> -->
<tr>
<td class="tablehead">&nbsp;</td>
<td class="tablehead" colspan="3">&nbsp;</td>
<td class="tablehead">&nbsp;</td>
</tr>

</table>
</form>
</body>
</html>
```

```
END_of_TABLE

sub getDir {
opendir(ETC, "/project/langdoc/SOFTWARE/CONCORDANCE/DATA/$language/")
   or
die "Cannot open the DATA directory!" ;

while ($toc = readdir(ETC))  {
if ($toc =~ m/\S+?\.ctt/g) {
push(@inh, $toc) ;
}
}
        closedir(ETC);
return @inh ;
}
```

F Consultation function

```
#!/vol/bin/perl -w

#authors Jahn & Ouattara

#Program : search.pl
#looks for a given word in a given text and prints out the results in multiple matching
#a result is composed of the line number and the the contents of that line

undef $/ ;

if ($#ARGV < 2) {
print "Usage: consultation.pl search outputfile inputfile(1) ... inputfile(n)\n" ;
        exit[0] ;
}

print "@ARGV\n" ;
$word = $ARGV[0] ;              #the word to be searched
@input = @ARGV[2..$#ARGV] ;     #the input file to look through
$output = $ARGV[1] ;            #the output file
$found = 0 ;                    #boolean variable which is 0 if there aren't any matches

open (OUT, "> $output")
or
        die "\n Output file couldn't be created!!\n" ;

foreach $dat (@input) {

open(IN, "< $dat")
or
die "$dat couldn't be opened!!\n" ;
$text = <IN> ;

while ($text =~ m/<concsentence sentencenumber=\"(\d+)\">(.+?)<\/concsentence>/gs) { #matches the
#sentence number and its contents in standard variables
$zeile = $1 ;
$inhalt = $2 ;
print $inhalt ;
if ($inhalt  =~ m/>$word</g) { #matches the word with the contents
$found = 1 ;
$inhalt =~ s/concword wordnumber/a name/g ;
$inhalt =~ s/concword>/a>/g ;
print OUT "line $zeile\n" ; #prints the sentence number into a file
        print OUT "$inhalt\n" ;     #prints the contents of the sentence into a file
         }
}
close(IN) ;
}

if ($found == 0) {
print "No matches found !!\n" ;
}

close(OUT);
```

G CGI interaction for consultation

```
#!/vol/bin/perl -w
use CGI qw(:standard);


my @word = param("word"); #the word to be searched
my @input= param("infile"); #the input file to look through
# my $output= param("outfile"); #the output file
my $language = param("language"); #the language to be investigated

undef $/ ;

print header();

print <<END_of_HEAD;

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html>
<head>
<title>AUDIO-Concordance output</title>
<link rev="MADE" href="mailto:ttrippel\@spectrum.uni-bielefeld.de" />
<base href="http://coral.lili.uni-bielefeld.de/langdoc/cgi-bin/test.pl" />
<meta name="copyright"
  content="University of Bielefeld, Computational Linguistics and Spoken Language" />
<meta name="author" content="Thorsten Trippel" />
<meta name="description" content="Results from the audio concordance query" />
<meta name="date" content="23 Nov 2000" />
<link rel="stylesheet" href="../langdoc.css" />

</head>
<body link="#ffffff" vlink="#fafafa" alink="#fa1340">
END_of_HEAD

print <<HEAD_of_TABLE;

<table class="intern" >
<tr>
<td class="background" width="137">
<img src="../IMAGES/1pix.gif" width="1" height="1" alt="" hspace="68" vspace="1" /></td>
<td class="background" width="300%"><img src="../IMAGES/1pix.gif"
    width="1" height="1" alt="" hspace="137" vspace="1" /></td>
<td class="background" width="137"><img src="../IMAGES/1pix.gif"
    width="1" height="1" alt="" hspace="68" vspace="1" /></td>
</tr>

<tr>
<td class="tablehead" bgcolor="#CCCCCC" >Contents</td>
<td class="tablehead">Hits: search for <b>$word</b> <br />in $language
<!-- text $filename as a corpus. -->
</td>
<td class="tablehead">Links</td>
</tr>
<tr>
<td class="content">
<p><a href="/LangDoc/index.html">Language Documentation Notes</a></p>
```

```
<p><a href="../index.html">Introductory page</a></p>
<!-- <p><a href="acquisition.pl">Search the concordance</a></p> -->
<p><a href="../SPECIFICATION/">Specification of the audio concordance</a></p>

<p>E-mail: <a href="mailto:langdoc\@spectrum.uni-bielefeld.de">langdoc\@spectrum.uni-bielefeld.de</a></p>
<p><a href="about.html">About the project</a></p>
<p>Designed: November 2000</p>

</td>
<td>

<table class="intern" border="0"  cellspacing="0" cellpadding="0">

HEAD_of_TABLE

foreach $file (@input) {

open (IN, "< $file")
or
        die "\n Input file couldn’t be opened!!\n" ;

$text = <IN> ;

$_=$file;
s/\.\.\/html-data\/DATA\/$language\///;
s/\.ctt//;
$filename=$_;

while ($text =~ m/<concsentence sentencenumber=\"(\d+)\">(.+?)<\/concsentence>/gs) { #matches the
#sentence number and its contents in standard variables
$zeile = $1 ;
$inhalt = $2 ;
foreach $word (@word){
if ($inhalt  =~ m/>$word</g) { #matches the word with the contents

#
# $found = 1 ;
$inhalt =~ s/concword wordnumber/a name/g ;
$inhalt =~ s/concword>/a>/g ;
$inhalt =~ s/>$word</><b>$word<\/b></g ;


print #<<CONTENT_END;
("<tr><td class=\"body\">text: $filename, line $zeile:
<br />
$inhalt\n</td><td class=\"body\"><a href=\"../CORPUS/AUDIO/$filename"."$zeile.wav\">
<img src=\"../IMAGES/speaker.gif\" alt=\"Link to audio\" /></a>
</td>

</tr> ")

# CONTENT_END

# print p("text: $file <br /> line $zeile: <br /> $inhalt\n") ;
#prints the sentence number into a file
#prints the contents of the sentence into a file
        }
}
}
}
print <<END_of_TABLE;
</table>
<!-- -->
</td>

<td class="linklist">

&nbsp;
</td>
</tr>
<tr>
<td class="tablehead">&nbsp;</td>
<td class="tablehead">&nbsp;</td>
<td class="tablehead">&nbsp;</td>
</tr>

</table>

</body>
</html>

END_of_TABLE

close(IN) ;
#print end_html();
```