Master Thesis

Development of a User Interface for Interactive Rule Generation

Ara Awakian

January 2010 – July 2010

Tutor:
Professor Dr.-Ing. Norbert Fuhr
Dipl.-Inform. Andrea Ernst-Gerlach

University of Duisburg-Essen
Faculty of Engineering
Department of Computer Science and Applied Cognitive Science
Institute of Information Engineering
47048 Duisburg
Contents

1 Introduction 1
  1.1 Overview ............................................. 1
  1.2 Objectives of the work ................................. 2
  1.3 Structure of the thesis ................................. 3

2 Related Work 5
  2.1 VARD 2 ........................................... 5
  2.2 LeXtractor .......................................... 9
  2.3 Evidencer 2.0 ...................................... 12

3 Usability 15
  3.1 Basics ............................................. 15
  3.2 Usability Evaluation ................................. 17
    3.2.1 Heuristic Evaluation .......................... 18
    3.2.2 Cognitive Walkthroughs ....................... 19
    3.2.3 Eye-Tracking ................................. 19

4 Requirement Analysis 21
  4.1 Main Goals ......................................... 21
  4.2 Stakeholders ...................................... 22
  4.3 Task Analysis ..................................... 22
  4.4 Non Functional Requirements ....................... 22
  4.5 Functional Requirements ............................ 24
    4.5.1 Loading historical texts ..................... 24
    4.5.2 Building Evidences .......................... 24
    4.5.3 Viewing and editing collected evidences ... 25
    4.5.4 Generating and editing transformation Rules 25
    4.5.5 Import and Export ............................ 26

5 Design Concepts 27
  5.1 Hierarchical Task Analysis ......................... 27
  5.2 The GUI conceptual Model .......................... 29
    5.2.1 GUI Components ................................ 29
## Contents

5.2.2 Prototyping ......................................................... 31
5.3 The functionality conceptual model ................................. 36

6 Implementation ......................................................... 41
6.1 RuleGenerator Main Frame .......................................... 41
6.2 Load Historical Text ................................................. 45
6.3 Auto Evidences .......................................................... 47
  6.3.1 Build ................................................................. 47
  6.3.2 Display ............................................................... 47
6.4 Evidences ............................................................... 49
6.5 Historic Text ........................................................... 52

7 Evaluation ................................................................. 55
7.1 Main Goals ............................................................... 55
7.2 Methodology ............................................................ 56
7.3 Experiment ............................................................... 57
7.4 Results, Analysis and Interpretation ................................. 58
  7.4.1 Observation and Interview Notes .................................. 59
  7.4.2 Questionnaire ...................................................... 60
  7.4.3 Eye-Tracking ....................................................... 62
  7.4.4 Bugs ................................................................. 64

8 Conclusion ................................................................. 65
8.1 Summary ................................................................. 65
8.2 Future Work ........................................................... 66

A Appendix ................................................................. iii
  A.1 Evaluation Documents ............................................... iii
    A.1.1 Introduction ................................................... iii
    A.1.2 Guidelines ...................................................... vii
    A.1.3 Tasks ............................................................. ix
    A.1.4 Questionnaires ............................................... xi
    A.1.5 Interview Guidelines .......................................... xiv

List of Figures .............................................................. xvii
List of Tables .............................................................. xix
Bibliography ............................................................... xxi
Chapter 1

Introduction

1.1 Overview

Today there are several large-scale digitisation projects aimed at digitising books and developing digital libraries, such as the 'World Digital Library', 'Google Book Search' and 'The Million Book Project'. These projects use advanced scanning and text recognition technologies. Although the optical character recognition (OCR) technology is extensively developed, even the best OCR programs do not achieve an exact rendering, due to errors in character recognition.

Digital libraries are often used by historians and linguists for full-text search in historical text collections. Since historical documents are full of spelling variations, one cannot assume that the search engine would return the desired results [GNR+09]. Most languages differ in spelling across the centuries. For example English was not standardised until the 19th century and this was done for German at the beginning of the 20th century, before which there was no normalisation of orthography. The spelling within historical text collections varies according to period and region. Hence, a search in text collections with non-standard spelling delivers low search result numbers and has low recall and incomplete partial information retrieval.

In order to enhance searches in historical documents and expand the search results, a search engine is being developed within the scope of the project 'Rule-Based Search in Text Databases with Non-standard Spelling' (RSNSR) at the department of Computer Science and Applied Cognitive Science of University Duisburg-Essen\(^1\). When the user enters a search term, each and every inflected

\(^1\) Institute of Computer Science and Interactive Systems, University of Duisburg-Essen
http://www.is.informatik.uni-duisburg.de/projects/rsnsr/index.html
and derived form of the given search term is determined first of all, using the modern dictionary. Afterwards, ‘transformation rules’ such as (t→th) are applied to each derived form of the modern search term and a corresponding historical term is determined [EGF07]. The resulting historical terms are included in the search query of the search engine, resulting in a larger number of search results being returned.

The transformation rules are generated from training data using ‘evidences’. An evidence is a pair of a modern suggestion and its corresponding historical variant. Figure 1.1 shows examples of evidences and the corresponding transformation rules from the 17th century.

1.2 Objectives of the work

To improve the search results from a historical search engine, a unique set of rules shall be generated for each historical corpus. The objective of this thesis is to design and develop an interactive Graphical User Interface (GUI), which allows a user with limited technical knowledge to build ‘evidences’ and generate
'transformation rules'. For every evidence created, transformation rules shall be generated automatically and visualized in an interactive user interface. The tool should offer separate views for evidences as well as for rules, so that the user can build or edit evidences and rules, independently of each other, himself.

In this thesis an interactive graphical user interface will be designed and developed. This will be followed by a prototypical implementation in Java. Afterwards an evaluation of the implemented user interface will be conducted. The results of the evaluation will be analyzed and interpreted to resolve the usability problems detected during the evaluation.

1.3 Structure of the thesis

This thesis comprises eight chapters. Following this introduction, chapter 2 introduces several approaches to the challenges of information retrieval in historical text collections and discusses the user interface features and functionality characteristics of three linguistic tools. In Chapter 3, usability theories and human-computer interaction concepts that are essential for the development of graphical user interfaces are presented.

In chapter 4, a requirement analysis is conducted, discussing the functional and non-functional requirements as well as the usability requirements of the prototype GUI to be developed. In chapter 5, the design and functionality concepts, derived from these requirements are described. These concepts were developed by a hierarchical task analysis, with the design of mockups and the creation of work-flow diagrams. Chapter 6 presents the implementation of the graphical user interface prototype and gives a detailed explanation of the graphical components that have been developed. In chapter 7, the methodology, concepts and results of the evaluation conducted on the GUI prototype developed here are presented.
Chapter 2

Related Work

Several linguistic research tools, that aid information retrieval in historical text collections, are currently available. For the development of the research specific tool within this thesis, a close look was taken at three interesting linguistic tools: VARD2, LeXtractor and Evidencer 2.0.

2.1 VARD 2

VARD 2, an interactive tool produced in Java, was developed at Lancaster University and is designed to assist users of historical corpora in dealing with spelling variation [BR08]. It extends the original VARD tool, which was developed for the purpose of pre-processing historical corpora for other corpus linguistic tools, such as Wmatrix [Ray07] and WordSmith Tools [Sco07]. In VARD, techniques from modern spell checkers are used to detect historical variant forms and equivalent replacement candidates. For each historical spelling variant found, the tool shows a list of modern equivalents ranked by "confidence". The list is established using three methods:

1. A manually created list of historical variant to modern equivalent mappings.
2. A phonetic matching technique, using phonetic code to link spelling variants.

Moreover, the user has the possibility of defining a threshold of "confidence" for automatic replacement of the variant forms.

VARD 2 has two user interfaces, a single file version and a batch process-
Chapter 2 Related Work

The first version, shown in Figure 2.1, allows the user to load historical text from a single file and manually replace a historical variant form with a modern variant form. The replaced variants are stored for referencing purposes and the change can be retracted if the user wishes.

![Figure 2.1: Single file processing user interface](image.png)

As can be seen in the figure above, historical text is loaded onto the left-hand side. The words highlighted in yellow are the historical variant forms detected by the system. On the right-hand side, the user can choose between four different views:

- Detected variant forms\(^1\).
- Replaced variant forms.

\(^1\) Words not in the tool’s modern lexicon.
• Modern variant forms presented by the system\(^2\).

• Uncommon words.

With the aid of a threshold meter, the user can set the threshold of replacement, which means that only those words with a "confidence" higher than the threshold will be replaced.

The tool offers two ways of processing the historical variants found:

• **Automatically:** By using the "Process All Variants" button, which replaces all variant forms with their highest ranked candidates.

• **Manually:** By right-clicking on each variant and choosing one of the candidates. (see figure 2.2)

![Figure 2.2: Manual processing of variants](image)

The batch processing version, shown in figure 2.3, allows the user to process multiple historical text files simultaneously. As with the single file version, the user can set a threshold for replacement. After processing all the files, the system will save the results in separate files.

\(^2\) Words in the tool's modern lexicon.
Chapter 2 Related Work

Figure 2.3: Batch processing user interface

VARD 2 is used for pre-processing historical texts with spelling variations and preparing them for a more accurate parsing by corpus linguistic tools. It is still being developed and improved. The accuracy\(^3\) of the tool and its effect on part-of-speech tagging have already been evaluated.

\(^3\) Accuracy of matching procedures
2.2 LeXtractor

LeXtractor is a web-based tool that enables a collaborative construction of historical lexica [GNR+09]. It is used for improving information retrieval on historical text collections by helping the user associate a modern word-form used in search queries with a corresponding historical word-form. LeXtractor was developed at the institute Centrum für Informations- und Sprachverarbeitung (CIS) of the Ludwig-Maximilians-University (LMU) Munich.

Within the scope of this project, two methods for improving information retrieval were developed:

- **Matching procedures**: This method traces historical spelling variants back to a corresponding modern word-form by applying patterns such as "t -> th". A pattern represents the difference between the historical and modern spelling variant forms.

- **Historical lexica**: A historical lexicon is a collection of historical word-forms determined by proofreading historical text collections. A corresponding modern word-form is manually assigned to each historical word-form.

The two methods introduced above were tested by means of experiments on historical text collections from four centuries (16th to 19th). The experimental results show that matching procedures are not precise enough and do not yield satisfactory results for historical texts from the 16th and 17th centuries. A combined approach of developing matching procedures and the construction of lexica is adopted in LeXtractor.

LeXtractor is built on four main modules:

- **The analyser module**: analyses a given historical corpus. Using matching procedures, it suggests one or more full modern word-forms for each historical word-form.

- **The database module**: stores the confirmed entries for a historical lexicon in a back-end database.

- **The managing module**: handles the communication between the different modules, the visualization of different views and user administration. It also
allows different users to edit the same corpus at the same time.

- **The graphical user interface module:** allows the user to add entries to the database module and view the database entries, parts of the historical corpus and the list of historical texts in the corpus.

LeXtractor is used to create lexicon entries for a given historical corpus. This can be done in two modes, a corpus and a document mode.

In the corpus mode, two lists of word-forms are presented: a list of pattern based matches on the left and a list of unknown\(^4\) words on the right (see figure 2.4).

---

\(^4\) Words with no relevant match
2.2 LeXtractor

When the user selects an entry from the list on the left, one or more pattern-based match suggestions are displayed by the system in a new window (see figure 2.5). The suggestions are derived by applying patterns to the selected entry. For each pattern, the system shows the position of change. The user can confirm or reject each suggestion. If no reasonable equivalent is suggested, the user can add the selected entry to a special list, for example "Historic word without modern equivalent", "Historic abbreviation", "Patter matcher failed", etc.

![Interpretations for string "thelie":](image)

Add token to special list

- Historic word without modern equivalent
- Historic abbreviation
- Patter matcher failed
- Named Entity
- Missing in modern lexicon

Figure 2.5: Confirm derivations dialogue

In the document mode, LeXtractor allows the user to work on a specific text within the historical corpus. The historical words detected by the system are marked in gray and probable historic words marked in gray and italic font (see figure 2.6). Mouse-over events provide the user with more information, such as the modern equivalent or a description of the word. Similarly to the corpus mode, the user can select a historical word-form and confirm the modern word form suggested by the system.

LeXtractor is used to build historical lexica for the purpose of improving recall for search engines when searching in historical corpora that contain spelling variations. New entries can be created either in the corpus mode, from a list of historical words detected by the system, or in the document mode which shows the historical text together with visual effects and mouse events. In both cases, for a given historical word form the user needs to confirm the modern word forms suggested by the system. The entries created are saved in a back-end database.

---

5 Verb or noun
2.3 Evidencer 2.0

Evidencer 2.0 [PL08] is a Java-based application that allows the filtering of historical word forms in historical texts and the creation of evidences. An evidence is a pair of a historical word-form and its corresponding modern word-form. Evidencer 2.0 was developed at the Institute of Computer Science and Interactive Systems at the University of Duisburg-Essen. The filter used in the application finds the historical spellings with the help of a Bayesian classifier and procedures to calculate the N-Grams [Win09] of each word in the historical text. The application has three main parts as shown in figure 2.7.
2.3 Evidencer 2.0

Figure 2.7: Evidencer 2.0

The historical text is displayed in the upper left of the interface and the historical word-forms that have been found are highlighted in red. The upper right section presents a list of the historical word-forms. Below, the user can select a historical word-form and give a modern word-form. It is also possible to choose one of the modern word-forms suggested by the system from a drop-down list. The evidences are saved in a back-end database.
Chapter 2 Related Work

Summary

Separate analyses of each tool lead to the following conclusions:

VARD 2 provides two ways to replace historical words in historic texts with modern words: single file processing and multiple file processing. In the single file version, a historic text is loaded and the words detected by the system as historic are highlighted in yellow. The user replaces the historic words using a right-click context menu. The multiple-file-processing version replaces historic words automatically and does not show the texts. The replacement is done by manually created replacement rules and historic modern mappings. Hence, in order to achieve reliable results the tool must first be trained.

LeXtractor allows the construction of historical lexica that improve information retrieval in historic text collections. In LeXtractor, the detected historic words are represented either in a list view or highlighted in the document view. An interesting feature of the tool is the support of mouse-events that show the user relevant details for each historical word detected.

Evidencer 2.0 allows the construction of pairs of historic and modern word-forms called evidences. The tool shows the loaded text and a list of the historic words detected. The well-designed layout of the graphical user interface is very convenient for the process of collecting evidences.

An analysis of the three different approaches described above leads to the conclusion that none of the approaches supports automatic construction of evidences and offers an interactive generation of transformation rules. The main focus of this thesis will be the design and implementation of a reliable and interactive Graphical User Interface (GUI) for the construction of "evidences" and the generation of "transformation rules" both automatically and manually. The tool will also support the visualisation and manipulation of transformation rules.
Chapter 3

Usability

This chapter illustrates the theoretical concepts that are essential in developing a Graphical User Interface (GUI). A graphical user interface enables interaction between humans and computers, by presenting program features by means of graphical components such as icons, colors and menus. Human Computer Interaction (HCI), the foundation of user interface design, is the study of the interaction between the human and the computer. Thus, it has three parts: the human, the computer and the interaction between them. The goal of HCI is to design user friendly interactive systems, so that the user can accomplish a task satisfactorily in a given amount of time [DFAB03]. HCI refers to the manner in which an interactive system should behave to meet the user’s expectations [Dah06]. The measure of user friendliness of an interactive system is termed 'usability'.

3.1 Basics

Usability is a qualitative attribute of an interactive system encompassing ease of use, ease of learning and user satisfaction.

The ISO 9241-11:1998 Standard Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten (Ergonomic requirements for office work with visual display terminals) defines usability as:

'Das Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen.'

Usability is the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context.
of use. It is not a single attribute of an interactive system, but can be broken down into further, more specific, attributes. There are three important usability attributes that can be considered in designing user interfaces: **Effectiveness**, **efficiency** and **satisfaction**. Effectiveness is the accuracy and completeness with which the users can achieve specific goals in a particular environment, meaning the degree to which the interface helps the user to accomplish a given task. Efficiency refers to the resources expended in relation to the accuracy and completeness of the achieved goals, and can be measured by the number of steps the user needs to make in order to accomplish a given task successfully. According to Nielsen, the system should be efficient to a degree that, when the user has learned the different features of the user interface, a high level of productivity is achieved [Nie93]. Satisfaction is the comfort and acceptability of the user interface and the extent to which the user finds the interface enjoyable and satisfactory to work with. In addition, **learnability**, **memorability** and **rate of errors** are considered essential attributes of usability. Learnability is the ability of the user interface to be learned quickly and effectively. It can be measured by the minimum time the user needs to perform a specific set of tasks successfully. According to Nielsen, a user interface should be sufficiently easy to learn that the user can start to work with it in a short time [Nie93]. Schneiderman defines learnability as the time needed ‘for a typical member of the user community to carry out the actions required for a set of tasks’ [SP09]. Memorability is the measure of how easily the user can become proficient and remember the interface after a period of not using it. Rate of errors is the number of errors made by the user when working on a given set of tasks. The user interface should be designed in such a way that it helps the user avoid making errors.

Usability is often related to ergonomics. Ergonomics, also known as 'human factors', studies the design of interactive systems, hardware and software, in particular looking at the work environments, efficiency, and safety needs of human beings. In ergonomics, it is essential to understand the cognitive abilities of the user, especially in information processing tasks.

The general ergonomic principles that are related to the design of dialogues between humans and interactive systems are discussed in part 110 of the ISO 9241:2006 Standard. These principles will be adopted for the development of the graphical user interface in this thesis.

- **Suitability**: A dialogue supports suitability for a task if it helps the user to complete a given task effectively and efficiently by presenting only the components that are necessary to fulfill the task. For example in an email program, the composing dialogue can contain font and color options but
should not contain the spam filters.

- **Self-descriptiveness**: A dialogue is self-descriptive if each step of it is contiguously comprehensible, in that immediate feedback is provided to the user and understandable icons and labels are deployed. Direct manipulation is a technique that enhances self-descriptiveness of a dialogue.

- **Controllability**: A dialogue is controllable if the user is able to control the speed and direction of his work until the task is fulfilled. Draggable toolbars and dockable windows are good examples of controllability of a user interface.

- **Conformity**: A dialogue supports conformity with user expectations if it adapts to the user’s experience, knowledge and education. Similar dialogues for similar tasks, as well as system status transparency, increase the conformity of a system.

- **Error tolerance**: A system can be considered to have error tolerance if, despite incorrect input from the user, it is possible to proceed with the task with minimal corrections. The errors should be explained so that user can correct them. Avoiding errors by disabling unnecessary dialogue components increases the error tolerance of a system.

- **Suitability for individualisation**: A dialogue is suitable for individualisation if it can be modified to accommodate to user-specific needs and preferences.

- **Suitability for learning**: A dialogue is learnable if it helps the user to learn how to use the system in a minimum amount of time.

### 3.2 Usability Evaluation

Within the scope of human-computer interaction studies, diverse criteria are used in classifying usability evaluation methods. Very often these methods are classified by user-based or expert-based evaluation. Another classification is analytic or empirical evaluation. While both the ‘user-based’ and ‘empirical’ evaluation methods are conducted by test participants, the ‘expert-based’ and ‘analytical’ evaluations are conducted by specialists.
3.2.1 Heuristic Evaluation

Heuristic evaluation, an expert-based evaluation methodology, is conducted in the early stages of a user interface development. Employing a set of usability principles called 'heuristics', experts evaluate whether the user interface components satisfy these principles and identify any weaknesses of the interface. This method was developed by Nielsen and Molich, who conducted several experiments [NM90] with experts and, by means of a detailed usability analysis, established a list of ten heuristics which are known as 'Nielsen’s Usability Heuristics'. Along with Shneiderman’s eight golden rules of interface design [SP09], the relevant heuristics for this thesis will be discussed next.

Feedback

Feedback refers to those reactions by the user interface to the user’s action, which keep the user informed about the current status of the interface. Pop-up messages, mouse-hover functions and tooltips are examples of this. Without a relevant response, the user would wonder whether the actions being performed were understood correctly by the user interface. According to Nielsen, 'the system should always keep users informed about what is going on, through appropriate feedback, within reasonable time.' 'Reasonable time', means that the time taken by the user interface to react to a user’s input is sufficiently short that the feedback is seen by the user before he begins a new action. For example, when the user hovers the cursor over a button and the tooltip does not appear quickly enough, the user might miss the information needed to understand the function of the button. In his 'Eight Golden Rules', Shneiderman advises: 'offer informative feedback for every action of the user'. According to Shneiderman, the reaction of the interface depends on the type of action carried out by the user. A simple, short response is sufficient in the case of frequent actions, whereas the response can more be substantial for major actions.

Consistency

A further, important aspect found in various instances of heuristics is 'Consistency'. For example, buttons which perform the same actions should always be represented by the same graphics and labeling. This is also required by the ISO standard. It is much simpler to learn a system if it is consistent. Shneiderman goes further and deems consistency to be the most important rule of all. He points out
that consistency is important for many aspects, such as terminology, i.e. meaning, colour coding and standardised menus and help files, all of which should be included in this regard. Furthermore, he says that user’s expectations should be met in all situations. This requirement is also found in the ISO standard.

**Error Handling**

Error-tolerance pertains not only to the way that incorrect user input is handled, but also to the manner in which the user can be broadly protected from making such errors. The ISO standard states that the user should be protected from the risk of erroneous input either by a warning message or restricted input options. Nevertheless, should an entry be made which the system cannot process, the user must be given constructive support to correct the mistake. Nielsen emphasises that error messages must be clear and easy to understand. Shneiderman goes on to say that inactive menus should be ‘greyed-out’ and in number fields it should not be possible to enter letters. He cites, as an example of how mistakes should be handled, that when a user is asked to fill out a form and types in the postal code wrongly, the user should not need to re-enter the entire address.

### 3.2.2 Cognitive Walkthroughs

Cognitive evaluation is a usability inspection approach, during which usability experts go through the user’s task from higher priority to lower priority. A deep understanding of the user’s goals and cognitive abilities, such as short and long-term memory, learning, decision making, reasoning and problem solving, is essential for performing cognitive walkthroughs. During each step of the walkthrough, the expert evaluates the user’s ability to identify and operate the user interface for the given task or subtask. The cognitive evaluation, conducted in the early stages as well as throughout the entire life-cycle of the process, is a way of finding usability weaknesses that cannot easily be discovered by other evaluation methods.

### 3.2.3 Eye-Tracking

Eye-tracking is a user-based evaluation method which is commonly used when a working prototype is available. Empirical methods such as click-analysis, questionnaires and interviews are generally limited to the conscious process, whereas Eye-tracking lays bare the cognitive process underlying the conscious experience.
Cognitive experience often occurs rapidly without the subject having much awareness of it. This leads to the user having difficulty in articulating the experience. By recording eye-movements, those elements which have drawn the user’s attention can be identified, as can those which have failed to do so. Eye movements reveal also which areas are easy to understand and also those which are not. Eye movements are described in terms of ‘fixation’ and ‘saccades’. ‘Saccades’ refer to eye movement from one point to another, whereas fixation denotes when the eye remains gazing at a fixed point.

**Summary**

In this chapter, the usability and human-computer interaction basics were introduced. The dialogue principles of ergonomics of human-system interaction, essential in developing graphical user interfaces, were presented and described. Following this, the usability evaluation methods; heuristic evaluation, cognitive walkthroughs and Eye-Tracking were introduced and explained.
Chapter 4

Requirement Analysis

This chapter discusses the requirements for the linguistic tool to be developed in this thesis. Using different methods of Requirement Engineering such as stakeholder analysis, requirements elicitation and specification [AW05], several aspects of the requirements of the tool were analyzed, documented and managed not only at the outset but throughout the life-cycle of the development process.

Throughout the whole chapter, within the description of the requirements a special wording form was adopted. For all absolute requirements of the specification the term SHALL will be used, meaning that the requirement must be included in the implementation of the linguistic tool. Whereas, for all optional requirements the word MAY will be used, meaning that the requirement can be included in the implementation based on the needs of the design.

4.1 Main Goals

The high level goal of the linguistic tool is to enhance information retrieval in historical text collections by offering to historians and linguists an interactive tool which helps them to build a unique set of transformation rules for historical corpora from a certain period and region. The tool shall allow the user to build evidences manually or by accepting automatically generated evidences. It should be possible to view and edit evidences as well as transformation rules.
4.2 Stakeholders

Stakeholder analysis identifies the potential users and stakeholders who may influence or be impacted by the system [HJD04]. For each stakeholder group, the responsibilities and tasks in relation to the system are determined. For this linguistic tool two stakeholder groups are to be considered:

- Historians and linguists, the direct users of the tool, who have limited technical knowledge. Hence the tool shall be operable by users with limited programming and technical knowledge.

- Developers and designers of other linguistic research and development areas. Hence, the code behind the tool shall be as accessible as possible and easy to implement interfaces to.

4.3 Task Analysis

The tool shall be designed to support building evidences and generating transformation rules in an interactive way. For each evidence that the user creates the transformation rules shall be displayed. It shall be possible to view and edit the transformation rules for a single evidence or for the complete list of evidences. The tool shall allow the user to work on evidences independently by loading historical texts or importing a list of evidences. Similarly, the user shall be able to work on transformation rules independently.

4.4 Non Functional Requirements

In software development, non-functional requirements define the qualities that resulting software must possess and are not related to the functions offered by the software. These requirements place constraints related to the characteristics of the software being developed such as reliability, accessibility, stability, on the process of development such as delivery time and costs [Som06]. Non-functional requirements also specify legal, economic or ethical constraints that the software must meet.
According to Ian Sommerville:

"Non-functional requirements, are requirements that are not directly concerned with the specific functions delivered by the system."

For the linguistic tool to be developed in this thesis the following non-functional requirements from the "IEEE-Std 830 - 1993 Standard" are relevant:

**Compatibility Requirements**

Compatibility is the capability of the software product to properly interact with other systems in its context. Within the scope of this thesis, functions that have been developed in other parts of the RSNSR project, shall be integrated in the linguistic tool to be developed. These functions were implemented in the programming language Java.

**Portability Requirements**

Portability is the ability of the software to function in more than one operating environment [Nie93]. The tool shall be platform independent, which means it should be capable to operate on the latest known Desktop operating systems such as Microsoft Windows, Linux and Mac OS X. Java was adopted for the implementation of the other classes of the RSNSR project. For this reason, the tool shall be implemented in the platform independent programming language Java.

**Usability Requirements**

Usability is the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions. The linguistic tool shall be easily and effectively operable, learnable and understandable by users with limited technical knowledge. The tool shall have a consistent user interface. Labels and messages shall have similar wording. Buttons may include icons elaborating the functionality of the button. Consistent color scheme may be used throughout.
Chapter 4 Requirement Analysis

4.5 Functional Requirements

After determining the stakeholders’ and analyzing the non-functional requirements of the linguistic tool, its functional requirements can be discussed. This section illustrates the behavior of the tool and its reaction to the users input considering the quality, performance and usability requirements discussed in the previous sections [Som06] [Poh07]. Moreover, functional requirements determine, functions that shall be offered by the tool, that derived from the main goals discussed in section 4.1 and the task analysis in section 4.3.

"Functional Requirements are statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.”

Ian Sommerville

4.5.1 Loading historical texts

Historical text collections are digitalized in different file formats. The tool shall support loading and displaying of historical texts from files with plain text format (TXT), Extensible Markup Language (XML) format and HyperText Markup Language (HTML) format. XML and HTML tags shall not be displayed with the historical text. Historical text collections are period and region dependent. When loading historical text files, the tool shall allow the user to input the language, period and the region of the historical text. It should be possible to load of multiple files simultaneously. Loading a complete folder with historical text collections shall be supported.

4.5.2 Building Evidences

Within the scope of the RSNSR project, procedures for building evidences automatically were developed. The tool shall implement these procedures on the loaded historical texts and allow the user to accept or ignore the resulting automatic evidences. The tool shall also allow building evidences manually, meaning the user shall have the possibility to input a historical word and a corresponding modern word and save it as a new evidence.

The spell-checker allows the detection of potential historical spelling variations in historical text collections. The tool shall implement these procedures on the
loaded historical texts and display the unknown words detected by the spell-checker. For each word the linguistic tool shall display a list of possible modern suggestions of the spell-checker which the user can choose to build one or more evidences. The linguistic tool shall offer the possibility to input a modern word manually in case none of the suggested modern words matches the historical word. In some cases, it can happen that the names, abbreviations, fail words or historical spelling variants for which no modern spelling variant exists, appear in the list of historical words. No evidence can be built from these words, hence the system shall allow the user to delete this words from the list of historical words. The evidences built or accepted by the user shall be saved externally, in a local file or external database.

In some cases, a historical word can have more than one meaning and the user might need contextual help. For this reason, sections from the historical text shall be displayed, along with further information such as the name of the file, as well as the period and region of the corresponding text.

4.5.3 Viewing and editing collected evidences

The linguistic tool shall allow viewing and editing of the collected evidences. The user shall be able to add a new evidence to the list of collected evidences and shall be able to delete an evidence from the list. For each evidence in the list of evidences, sections of the historical text shall be displayed. Sorting and filtering functionalities on the list of evidences shall be available. The linguistic tool shall also allow the user to edit an existing evidence by changing the modern word of the evidence.

4.5.4 Generating and editing transformation Rules

The linguistic tool shall allow the generation of transformation rules in several ways. For each evidence that the user builds, transformation rules shall be generated. It shall also be possible to generate transformation rules manually, without any evidence. In addition to rules generation, the tool shall allow to view and edit the collected transformation rules. It shall be possible to have different views of the transformation rules. For instance, view the transformation rules for a specific evidence or import and view a certain set of transformation rules. The user shall also be able to delete transformation rules and edit them by changing the "search for" or "replace with" part. It shall be possible to edit the transfor-
mation rules for a specific set of evidences. The transformation rules and all the related tasks shall be visualized so that they can be performed by users with limited programming skills and technical knowledge. The details on the modification and visualization of the transformation rules will be discussed in the concurrent thesis [Kor10].

4.5.5 Import and Export

When working on huge historical text collections, the work might need to be split between more than one user. For example one user collects the evidences and another user generates and edits the transformation rules. For this purpose the tool shall support the import and export of evidences and transformation rules thus allowing the generation of transformation rules from evidences collected by another user.

Summary

In this chapter, a requirement analysis for the GUI prototype to be implemented in this thesis, was conducted. The analysis included the specification of the usability requirements, as well as the functional and non-functional requirements of the user interface.
Chapter 5

Design Concepts

In the previous chapter, a detailed analysis of the requirements of the GUI prototype was carried out. For the tool to be developed in this thesis the name ‘Rule Generator’ was chosen. This chapter illustrates the conceptual design of the prototype, that shows how these requirements are transformed into a conceptual model. The conceptual model of ‘Rule Generator’ is divided into a GUI model and a functionality model. Based on the requirements in section 4.3, a detailed task analysis was carried out that is discussed in the next section.

5.1 Hierarchical Task Analysis

Hierarchical task analysis (HTA) provides an understanding of the tasks the user needs to perform to achieve a certain goal. Breaking down the primary tasks into subtasks makes it easier to understand the interactions between the user and the user interface [SRP07]. Having an overview of the tasks and understanding the relationship between them leads to an effective and efficient conceptual design. While designing the user interface, grouping the related and similar task together would increase the learnability of the tool. The number of steps in a task can be reduced, in order to allow the user to complete the task more easily, which would make the tool more efficient. For this purpose, a hierarchical task analysis was carried out; this is presented with the diagram in figure 5.1.

The linguistic tool shall be used to create a set of unique transformation rules for a given historical corpus. As seen in the HTA, the main task in using the linguistic tool is to generate transformation rules (plan 0). This can be done in two ways, either manually where the user needs to determine the ‘search’ and the ‘replace’ parts of the transformation rule (plan 1) or automatically by building an evidence and applying the automatic rule generation functions (plan 2). The
Chapter 5 Design Concepts

1. Add manually

2. Add from Evidence

2.2 Add from historical file

1.2 Enter "replace" part

1.1 Enter "search" part

2.1.1 Enter Historical

2.1.2 Enter Modern

2.2.1 Load historical text

2.2.2 Map Evidence

2.2.3 Accept Autoevidence

2.2.2.1 Enter historic

2.2.2.2 Enter modern

2.2.1.3 Enter language

2.2.1.2 Enter period and region

2.2.1.1 Enter file path

Plan 0: Plan 0: do 1 or 2.

Plan 1: Plan 1: do 1.1 then 1.2.

Plan 2: Plan 2: do 2.1 or 2.2.

Plan 2.1: Plan 2.1: do 2.1.1 then 2.1.2.

Plan 2.2: Plan 2.2: do 2.2.1 then 2.2.2.

If auto-evidences available do 2.2.3

Plan 2.2.1: Plan 2.2.1: do 2.2.1.1 – 2.2.1.2 – 2.2.1.3

Tasks Diagram

Figure 5.1: HTA Diagram
user can build an evidence either manually by determining a historical term and its corresponding modern term (plan 2.1) or from historical terms in historical text collections (plan 2.2). For the latter, which is the standard way of building evidences, the user needs to load historical text files in which the possible historical terms will be detected by applying a spell-checker. During the process of loading a file (plan 2.2.1) the user needs to specify the path of the file as well as the period, region and language of the historical text. The user can map historical and modern terms (2.2.2) or accept auto-evidences suggested by the tool (2.2.3).

The task analysis which has been performed will be used to develop the GUI conceptual design as well as the functional conceptual design of the linguistic tool.

### 5.2 The GUI conceptual Model

The conceptual model of the graphical user interface deals with the interaction and the presentation characteristics of the linguistic tool. The interaction design of the user interface is concerned with the fulfillment of the non-functional, especially the usability, requirements discussed in section 4.4. Whereas the presentation design of the user interface deals with the layout (components) and visual properties (font, color, etc.) of the tool.

#### 5.2.1 GUI Components

Before getting into the details of the layout and the components of the GUI, the high level components were determined by considering the requirements. The diagram in figure 5.2 shows the components needed to fulfill the requirements in section 4.3.

The 'Main Window' is the main frame of the tool, which holds all the components of the GUI. The 'Menu Bar' includes all the functions of the tool grouped in menus. A menu can include submenus and items. Each submenu can include further submenus. An item holds a single function and consists of an icon and text related to the function it presents. Similar functions will be held close together and grouped in a common menu or submenu (Consistency in section 3.2.1). The frequently used functions are made available in the component 'Toolbar'. Each function is reached through a button, which includes an icon and a text label. The 'Evidences View' can be an internal frame, or a panel within the main frame of
Figure 5.2: High Level Components
the tool. It includes all the GUI components needed to perform the tasks related to building and editing evidences, thus covering the tasks 2.1, 2.2.2 and 2.2.3 (see figure 5.2). It also includes the components needed to show sections as well as the region and period of the historic text, thus covering the task 2.2.1 and its subtasks. The 'Rules View' includes the components needed to view, edit and generate transformation rules. The visualization of the transformation rules is displayed in part of the user interface.

5.2.2 Prototyping

The creation of the prototype GUI was an iterative process. In this process the main components were incorporated into a mockup. By critical examination of this mockup it was possible to assess whether it successfully fulfills the tasks analysis requirements. This section deals with the optimized prototype and discusses how it fulfills these requirements. The final mockup or prototype was drawn using the sketching tool 'Pencil'¹ (see figure 5.3). The components on the GUI screen are analysed from top to bottom.

Layout Overview

The mockup in figure 5.3 shows an overview of the graphical user interface as well as the arrangement of the main components discussed in section 5.2.1.

The title-bar, placed at the top, includes an icon, the name of the linguistic tool and the standard window control buttons: 'minimize', 'maximize' and 'close'.

As seen in the figure above, the main window of the tool is divided into four parts, from top to bottom: the 'menu-bar', the 'tool-bar', the 'evidences view' and the 'rules view'. As in most applications, the menu-bar is placed at the top of the main window and below it the toolbar is separated from the menu-bar by a horizontal line. The standard positioning of the menu-bar and the tool-bar increases the learnability of the user interface. The remaining area of the screen is divided into four sections: The upper left part of the screen is the most viewed section of the screen. This part was provided for building, viewing and editing evidences. For this reason section ➀ (upper-left area) is made available for these tasks. The tab-pane allows the building of evidences in the first tab and viewing as well as editing evidences in the second tab. In section ➁, portions of the historical text

¹http://pencil.evolus.vn/
Figure 5.3: Mock-Up presenting the layout of the main components
are shown as contextual help for building or editing an evidence. In this case the user needs context of the text more often than when viewing or editing the transformation rule; thus placing this component on the same horizontal line to the right of the section A is a suitable area. Another frequent task performed by the user is viewing, selecting and editing transformation rules for specific evidences. Thus this component is placed right below the evidences in section C. The generated transformation rules in section C are visualized in section D. This layout ensures the shortest sight-path whether building evidences in A or generating transformation rules in C and reduces diagonal paths that might be awkward for the user.

This setup uses the standard computer screen in landscape mode, because this aspect ratio matches the content of the components best. This efficient layout was essential for working on evidences and transformation rules simultaneously, in order to have all of the pertinent information on one screen.

The MenuBar

The menu-bar includes the standard menus that can be found in most applications 'File', 'Edit', and 'Help'. Each menu contains submenus and items, each having an icon and a name. Submenus have an arrow at the right as seen in figure 5.4 which shows an example of the menu 'File'. Items with similar functionalities are grouped together and groups are separated using horizontal separators. Selected submenus and items are highlighted in grey.

![Figure 5.4: 'File' Menu prototype](image-url)
Chapter 5 Design Concepts

The Main Toolbar

The toolbar contains the frequently used functions from the menu bar. Figure 5.5 shows the design and layout of the buttons on the toolbar.

![Main toolbar prototype](image)

**Figure 5.5:** Main toolbar prototype

The buttons on the toolbar have the same icon and text as in the menu bar items (Consistency in 3.2.1). Note that the icons on the toolbar are larger than the icons in the menu bar. Similar functionalities are grouped together and have similar icons like the 'import' and 'export' buttons. This design supports the usability principles of learnability and consistency (in sections 3.1 and 3.2.1). The buttons of the tool-bar are separated by vertical separators. In case a button has more than one functionality, for example import evidences or import rules, drop down menu are used when the button is clicked.

Evidences View

The 'Evidences view' is part of the user interface where the user can build, view and edit evidences as well as read sections of the text as context. It consists of section A and B grouped together by a thin grey border with the title 'Evidences' (see figure 5.6).

![Evidences view prototype](image)

**Figure 5.6:** Evidences view prototype
5.2 The GUI conceptual Model

In section A, the tab ‘Build’ allows the user to build evidences. The potential historical spelling variants, detected by the spell-checker, are shown in the list on the left. For each selected word, modern suggestions are displayed on the right. The user can choose one or more modern words as well as enter a modern word manually, then with the save button build an evidence. The small toolbar above the list of historical words provides functions related to the potential historical words.

Viewing and editing of evidences can be done under the tab ‘View/Edit’ (see figure 5.7). The built evidences are shown in a table, which includes frequency and status (confirmed or not). The small toolbar above the table provides the functions required to edit evidences.

![Figure 5.7: View / Edit an evidence prototype](image)

The portions of the historical text in Section C show the selected historical word highlighted in yellow, as well as the file names of the historical texts where the selected historical word was found (see figure 5.3).

**Rules View**

In the ‘Rules View’ the generated transformation rules are displayed in a list on the left (section C in figure 5.3). In this view, the user can edit or delete the transformation rules (see figure 5.8). A similar toolbar as in the “Evidences View”
provides functions related to the rules. The filter button allow the user to view the transformation rules of the selected evidence in ‘Matched’ tab, or view the transformation rules generated from the complete list of evidences. When the user selects a rule in the list the rules visualisation is illustrated on the right (section D in figure 5.3). The ‘Rules View’ is also surrounded by a border called ‘Rules’.

![Figure 5.8: Rules View](image)

### 5.3 The functionality conceptual model

This section discusses the functions that have to supported by the linguistic tool and is concerned with the fulfillment of the functional requirements discussed in 4.4. In other words, the functional conceptual model discusses the ability of the tool to allow the user complete the tasks described in section 5.1 and determine the functional components needed to develop the graphical user interface. These functions are presented as in the work-flow diagram in figure 5.9.

Once the user starts the tool, the user can import evidences and transformation rules or load historical text from a single or several files. Before the files can be processed, the user needs to assign the period, region and the language of the historical text. At this point, assigning the language is important in order to apply the correct spell-checker. Processing the historical text includes applying the ‘spell-checker’ and the ‘auto-evidencer’ procedures implemented in the RSNSR project. The ‘auto-evidencer’ is a set of functions developed in the RSNSR project that create automatic evidences. The spell-checking process delivers a list of
Figure 5.9: Work-flow diagram
potential historical spelling variants, and for each word in the list a number of modern spelling variant suggestions. The list of the historical terms is displayed together with the modern terms. From these lists the user can build evidences by mapping one or more proper modern suggestions for each historical variant in the list. Another way of building an evidence is through the list of auto-evidences provided by the auto-evidencer functions. Not all the evidences in this list are necessarily correct. Hence the user needs to accept the correct evidences and reject the wrong ones. The third and last way of building an evidence is the manual variant, which is independent of the historical texts. For this purpose the user needs to determine both the historical and modern spelling variants.

The evidences created by the user are displayed in the component 'View Evidences' and saved in parallel to a local XML file. In the component 'View Evidences', the user has the possibility to view, delete and edit evidences. In both cases, the XML file is updated with the changes made. For all three ways of building an evidence, first the evidence is added to a list of evidences and second the rules, specific to each evidence, are generated using the 'rule generator' functions of the RSNSR project. These functions determine the 'search' and 'replace' parts and build a transformation rule by applying string comparison techniques to the historical and modern parts of the evidences.

The resulting transformation rules are added to the 'Rule View' component, where the user can view, delete and edit the transformation rules. The user can either view the complete set of collected transformation rules, or limit the view to the transformation rules specific for each evidence. For the latter, when the user selects an evidence, only the transformation rules specific to that evidence are displayed. Adding a rule can also be done manually, where the user can input the 'search' and 'replace' parts. When the user selects a specific transformation rule, the 'Rule Visualization' component shows the rules core and all its child rules in a two dimensional graphical form.

Summary

The concepts on the design of the 'Rule Generator' were developed through prototyping the creating mockups in an iterative process until the final design of the user interface was achieved (see figure 5.3). From the requirement analysis in chapter 4 and the cognitive walkthrough of the user’s tasks in 5.1, it can be concluded that the following GUI components are needed to fulfill the functionalities presented in the work-flow diagram in figure 5.9.
5.3 The functionality conceptual model

- **'Load Historical Text'**: that allows the user to load historical texts and input the period as well as the region.

- **'AutoEvidences'**: to display the automatically generated evidences that can be confirmed or rejected by the user.

- **'Build Evidences'**: which shows the list of potential historical words detected by the spell-checker and allows the user to build evidences from a list of modern suggestions.

- **'View and Edit Evidences'**: This component allows the user to view and edit not only the confirmed automatic evidences but also the evidences created in the 'Build Evidences' component.

- **'Historical Text'**: to provide the user with context of the historical text when building evidences.

- **'View and Edit Transformation Rules'**: to view and edit transformation rules.

- **'Visualize Transformation Rules'**: to display a graphical presentation of the transformation rules.
Chapter 6

Implementation

This chapter presents how the design concepts in the previous chapter are developed into a GUI prototype considering the requirements in chapter 4. Based on the portability requirements in section 4.4 and the implementation of the RSNSR project classes in Java.

The GUI prototype was implemented in the Application Programming Interfaces (API) Swing and Abstract Windows Toolkit (AWT) of the open source Java Development Kit (JDK) from Sun Microsystems\textsuperscript{1}. The classes of the implementation can be found in the package ’Smart evidencer.gui’ in the RSNSR project. Additionally the chapter discusses the implementation of the mockups in section 5.2 as well as the components presented in section 5.3. It also presents the interfaces to the RSNSR classes and the integration of the components ’View Transformation Rules’ and ’Visualize and edit Transformation Rules’ that were implemented in the parallel thesis [Kor10].

6.1 RuleGenerator Main Frame

The main frame of ’Rule Generator’ is the top-level window of the tool. It holds the graphical user interface components discussed in the previous chapter and defines their layout and distribution on the screen. The main window is implemented in the class ’MainFrame.java’ that extends the Swing class JFrame, which offers useful methods for window related operations. Some of the important methods used are setDefaultCloseOperation(Frame.EXIT_ON_CLOSE) that terminates the application when the windows is closed, setJMenuBar() and setJToolBar() that set the menu-bar and tool-bar of the main frame respectively.

\textsuperscript{1}http://sun.java.com
Chapter 6 Implementation

The MenuBar

The menu-bar was implemented in the class 'MenuBar.java' that extends the Swing class JMenuBar. It contains the menus 'File', 'Edit', 'Tools' and 'Help'.

As an example, Figure 6.1 shows the menu 'Tools' and including its sub-menus and menu-items.

![Figure 6.1](image)

Figure 6.1: Example of sub-menus and menu-items in the menu-bar.

Each item in the menu has a unique name and depending on its functionality it is characterised by an icon, a checkbox, or a radio button. Tooltips on the menu-items provide more information on their functionality. The submenus are characterised by an arrow, showing the user that they are expandable. In order to help the user avoid making errors (from 'Error Handling' in 3.2.1), the irrelevant menu items are greyed out. For example in the figure above, the export button is greyed out since in the initial state of the tool, when the user has not yet done any work, nothing can be exported. The menus are mnemonics-enabled which allows the user to navigate through them using the keyboard, increasing the accessibility of the tool.

The Toolbar

Figure 6.2 shows a screenshot of the tool-bar taken in the initial state of GUI prototype.

![Figure 6.2](image)

Figure 6.2: The main toolbar.

The toolbar was implemented in the class 'MainToolBar.java' that extends the Swing class JToolBar. It holds the most frequently used functions with buttons implemented in the class 'ToolBarButton.java' that extends JButton. Each
button has a textual description `setText()` and a unique icon `setIcon()` yielding/expressing its functionality. For buttons with similar functionalities, similar icons were chosen, for example the icons for ‘import’ and ‘export’ menus in figure 6.1. The tool-bar icons were manipulated to provide a glowing effect using the opensource image manipulation program ‘GIMP’\(^2\). With the help of the methods `mouseEntered()` and `mouseExited()` of the class ‘ToolBarButton.java’ a mouseover glow effect was implemented on the tool-bar buttons increasing the usability of the tool. The ‘load’ buttons opens a dialogue for loading historical text (see 6.2). The ‘clear’ button brings the tool to its initial state, in case a new session needs to be started. The ‘Auto Evidence’ buttons opens the automatic evidences window (discussed in 6.3). The import and export buttons allow the import of ‘transformation rules’ and ‘evidences’. The import and export of ‘transformation rules’ occurs through the interface in the ‘MainFrame.java’ class to the methods `importRules()` and `exportRules()` of the class ‘Rules.java’ implemented in [Kor10].

InfoNode

The InfoNode Docking Windows (IDW) is an open source Java Swing based docking framework. The windows are Swing components that can be integrated into any graphical user interface implemented in Java. They can be arranged in various layouts using split windows, tab windows and floating windows. The user can minimise, maximise, close, undock, dock and move around the windows and create a personalised window layout. All windows are placed in one container called the ‘Root Window’. For each window a ‘View’ has to be defined which contains light-weight Swing components for example a ‘JPanel’. InfoNode Docking Framework was used to position the components ‘Build Evidences’, ‘View and Edit Evidences’, ‘Historical Text’, ‘View Transformation Rules’, and ‘Visualise and Edit Transformation Rules’ on the main frame of the ‘Rule Generator’. In the ‘MainFrame.java’ four views were defined for these components that fulfill the layout concepts in section 5.2.2 see figure 5.3. The components ‘Build Evidences’ and ‘View and Edit Evidences’ share the first view called ‘Evidences’. The second view holds the component ‘Historical Text’, the third view named ‘Rule Selector’ holds the component ‘View Transformation Rules’ and finally the fourth holds the ‘Visualise and Edit Transformation Rules’ (see figure 6.3).

\(^2\)http://www.gimp.org/
Figure 6.3: Layout Overview of Rule Generator
6.2 Load Historical Text

The component 'Load Historical Text' is responsible for loading the contents of the historical texts into the tool. It was implemented in the class 'OpenDialogue.java' that extends the Swing class JDialog. While loading historical texts the user needs to determine the period, region and language of the text. This can be done in a new dialogue that is initiated when the user clicks on the open button of the main tool-bar or from the 'File' menu in the menu-bar. Figure 6.4 shows a screenshot of the load dialogue.

![Load Historical Text Dialogue](image)

**Figure 6.4: Load historical texts dialogue**

The graphical components of the dialogue are organised using the Swing layout manager ' GroupLayout '. The functions createSequentialGroup() and createParallelGroup() offer an effective way of grouping components together. As seen in the figure above, the buttons for adding and removing files are grouped together on the upper right area of the dialogue. The 'Add File(s) ' button opens a file chooser dialogue of type JFileChooser which allows the user to choose multiple files or folders. The chosen files and folders are processed by the methods addFiles() and addFolders(). The latter provides a recursive mechanism on the subfolders. Since the paths names may be long, the list is given the largest area on the dialogue so that the user does not have to scroll horizontally when browsing through the list. The 'Remove' and 'Remove All' buttons allow the user to delete
file paths from the list. Selection of multiple items in the list is possible by using the keyboard keys 'ctrl' and 'shift'. In the lower left area of the dialogue the user can input the language, period and region of the files to be loaded. The function `dictionaries()` detects available dictionaries in the 'dict' folder of the RSNSR project, which are added to the Combo-box 'Languages' dynamically. In the lower right area with the title 'Automatic Evidences', the user has the possibility to input the essential parameters of the 'auto-evidencer' function. By default 4 is set for 'Minimum length of word' and 2 for the parameters 'Maximum rules used' and 'Minimum rule occurrence'. With the checkbox 'Enable automatic evidences' the user can choose whether the automatic evidences should be displayed right after the loading of the files is finished. In either case, the automatic evidences are still generated but not shown. The 'Auto Evidences' dialogue can be activated with the 'Auto Evidences' button in the tool-bar. The 'Cancel' and 'Finish' buttons terminate the loading process. 'Cancel' takes the user back to the previous session. 'Finish' clears the previous session and passes the user's input to the 'LoadFilesTask.java' class that extends the Swing class `SwingWorker<T,V>`. If the list of files to be loaded is empty, the user is informed with a popup message. The 'Region' and 'Period' fields are mandatory, hence the loading process cannot be terminated if these fields are empty.

When implementing a GUI in Swing, time-consuming tasks should not run on the Event Dispatch Thread (EDT), otherwise the GUI becomes unresponsive. Tasks with time intensive computing have to be run at least on two threads: a thread to perform the lengthy task and the EDT for all GUI-related activities. This can be easily managed by the `SwingWorker<T,V>` that allows to perform long interacting tasks in a thread. It runs the time consuming tasks in a background thread in the method `doInBackground()` and provide updates to the GUI either while processing the tasks in the method `process()` or when done in the method `done()`. Depending on the size and number of chosen files to be processed, the spell-checking, auto-evidences, and generation of rules can be time-consuming. The methods\(^3\) for performing these tasks are initiated on a new thread in the `doInBackground()` method of the 'LoadFilesTask.java' class. During this process, the user must be given feedback. Instead of the process-bar as shown in the mockup 5.3, a small popup dialogue is initiated on the EDT informing the user that the files are being loaded. The advantage of the popup dialogue is that, it saves vertical space on the screen, attracts the attention of the user more effectively and gives textual information (see figure 6.5).

\(^3\)Implemented in 'TextCollection.java’ class in RSNSR project
6.3 Auto Evidences

6.3.1 Build

'Auto Evidences' are automatically built evidences using the methods from the classes 'TextCollection' and 'TermWrap' developed within the RSNSR project. The construction of these evidences occurs in the `doInBackground()` method of the class 'LoadFilesTask'. During the loading process of historical text discussed in section 6.2, the user chooses the paths and the language of the historical files to be loaded and sets the parameters for constructing auto evidences. Based on the language input of the user the right dictionary is used in the method `loadUnmatchedWrongSpelledTerms()` of the spellchecker to detect the unknown words in the historical text. For each wrong spelled word, firstly the method `loadSuggestions` is applied on the historical text to get the suggestions of the spellchecker, and secondly the method `acceptEvi(wordLength, maxRulesApp, minRuleOcc)` to find the modern match based on the auto evidences parameters entered by the user. When these tasks are finished, with the method `getMatchedWrongSpelledTerms()` the wrong spelled words that have a successful modern match are collected. These are the potential historical words of the new auto evidences. For each of these words the modern match is picked by the method `getPickedByRule()`. With the method `getRules()` the rules that were applied to build the auto-evidence are retrieved. The above mentioned methods are implemented in the RSNSR class 'TextCollection'.

6.3.2 Display

The constructed evidences are displayed in a separate window with the title 'Automatic Evidences'. It was implemented in the class 'AutoPairs.java' that extends the Swing class `JDialog`. Similar to the 'Evidences' view in the 'Matched' tab, a four column table of type `JTable` was implemented to hold the evidences (see Figure 6.5: Loading files feedback)
figure 6.6). The first column of the table includes a checkbox that allows the user to select an evidence. It is also possible to select all evidences with a single checkbox and confirm them. Tooltips along a complete row, provide quick overview on the context of the historical text for the historical word of the evidence in that specific row.

![Auto Evidence Window](image)

**Figure 6.6: Auto Evidence Window**

If the user wants to build evidences and generate transformation rules quickly,
this can be done by saving the successful matches with the 'Accept' button or temporarily accepted with the 'Later' button to be checked again later if the user is not completely sure. In both cases, the evidence is deleted from the 'auto evidences' table and inserted in the 'view evidences' table in the 'Matched' tab with a ✔️ icon for the 'accepted' evidences and a ✉️ icon for the 'to be checked later' evidences (see figure 6.7).

![Figure 6.7: Accepting auto evidences](image)

While working on automatic evidences the user can see the processed evidences being inserted in the 'Matched' tab. This is an important feedback feature of the tool, discussed in section 3.2.1. For this reason, whenever the 'Auto Evidences' window is activated, whether manually through the 'Auto Evidences' button in the main tool-bar or automatically after the termination of loading of files, the focus in the main frame is automatically set to the 'Matched' tab.

### 6.4 Evidences

The GUI component for building, viewing and editing evidences are implemented in the class 'PairsPanel.java'. These components are placed on a panel of type `JPanel` which is integrated in the first view of the InfoNode Framework called 'Evidences' (see figure 6.3). The area of the 'Evidences' view on the screen is not enough to hold all the components for building as well as viewing and editing evidences. This problem was solved by placing the components in two separate
Chapter 6 Implementation

tabs. The first tab, called 'Unmatched', holds the components needed to build evidences (see figure 6.8).

Figure 6.8: Build Evidences View

The potential historical words detected by the spell-checker are displayed in a list on the left. Abbreviations, names and words without a modern equivalent can be deleted with the delete button \(\times\) in the small tool-bar. The 'Add to dictionary' button \(\square\) deletes a word from the list and saves it in an XML file. The words that are added to the dictionary will be excluded from the list of potential historical words for the next sessions. This feature can be disabled by deactivating the checkbox 'Dictionary' in the 'Tools' menu. Tooltips were implemented on all the buttons of the toolbar. For each potential historic word selected in the list, several modern suggestions are displayed on the right. Since each historical word might have several equivalent modern words, the suggestion are displayed in a list of checkboxes. For each suggestion, the resulting rules are displayed in a different color (grey). The user can choose the correct modern words and build evidences by pressing the save button \(\square\). Additionally, the user has the possibility to input a modern word which is not in the list of suggestions using the input-field 'Manual variant' (see figure 6.8). The user can build a manual evidence by inserting a historical word manually with the add button \(\oplus\) and entering an equivalent modern word in the 'Manual variant'. When the user creates an evidence, the historical word disappears from the list and the next historical word is selected automatically. The list of modern suggestions is updated with the suggestions of the selected historical word. The created evidence is inserted in a
table under the 'Matched' tab (see figure 6.9) and saved in a local XML file (see section XML).

Figure 6.9: View and edit evidences

In the 'Matched' tab the user can view and edit the created evidences that are listed in a table. The 'Modern' column is set editable which means the user can edit the modern word of an evidence by double clicking the specific cell in the table. The evidences with the icon ✓ are the ones that the user is sure of and that are saved in the XML file. The evidences with the icon ? still need to be checked by the user. When the user examines an evidence and saves it with the 'Save' button its icon is changed to ✓.

The evidences can be filtered using the button ☑ in the toolbar (see figure 6.10).

Figure 6.10: Filter evidences

The user can build a new evidences manually using the + button. Figure 6.11 shows the dialogue for adding a new evidence.
In the 'Add Evidence' dialogue the four input fields are mandatory. The fields 'Period' and 'Region' are pre-filled with the input of the user during the beginning of the session (when loading files). In case the user wishes to build several manual evidences, it is not necessary to enter the period and region every time. The 'Historic' field is an editable combo-box, which allows the user to either enter the historic word manually or choose one from a list of the historical words of the evidences in the table. This feature is useful for the cases when the user wishes to build a new evidence with the historical word of an existing evidence.

6.5 Historic Text

The 'Historical Text' shows context to the user when building or editing evidences. It was implemented in the class 'TextPanel.java' and integrated in the second view of the InfoNode Framework called 'Historical Text' (see figure 6.3). Figure 6.12 shows a screenshot of the 'Historical Text' view. The context is updated everytime the user selects a new item in the list of potential historical words as well as in the table of existing evidences.
In the picture above, historical text was loaded from two different files ‘1626 Fragmente der Berliner Wochenzeitung Teil 1.txt’ and ‘1626 Fragmente der Berliner Wochenzeitung Teil 3.txt’. The word ‘theil’ highlighted in yellow was selected in the ‘Historic’ list under the ‘Unmatched’ tab. Figure 6.12 shows three blocks of context for the word ‘theil’. Each block consists of 40 word section of the text in which the word ‘theil’ appears. Additionally, the region, period and the full path of the loaded historical text is displayed. If more context is needed, the user can open the full text by clicking on the file path.

Summary

This chapter talks about the implementation of each of the components which had been selected in the design concept stage. It also describes the interfaces which integrate functionalities such as the spell-checker and auto-evidencer implemented within the RSNSR project. The number of graphical components needed for the user-interface was more than could be laid out using standard Java layout-managers. To deal with this, the InfoNode Docking Windows Framework was used instead.

This chapter also shows the integration of the rule visualisation and manipulation components implemented in [Kor10]. Consistency techniques, like similar icons for import and export and similar tool-bars in ‘Matched’ and ‘Unmatched’ tabs, were
Chapter 6 Implementation

used to make the tool easier to learn and error-handling techniques, like greyed out menu items to help the user avoid doing mistakes.
Chapter 7

Evaluation

This chapter discusses a user-based evaluation of the prototype GUI. This evaluation was conducted together with the evaluation of the two components; the "Rule Selector" and the "Rule Visualization" that were implemented in the concurrent thesis [Kor10]. At this point of the development process, the name 'SmartEvidencer' was used for the tool. The complete documentation of the evaluation can be found in appendix A.1. A concept for the evaluation was developed in which the main goals and the methods of evaluation are defined [SRP07]. This concept also discusses the practical and ethical issues concerning the participants. After the experiment is conducted, the results are analysed and interpreted.

7.1 Main Goals

This evaluation aims to test the effectiveness of the concepts and design of the graphical user interface discussed in chapter 5, and the usability of the implemented prototype that was presented in chapter 6. The evaluation considered the following points:

- Identify the usefulness, understandability of the graphical user interface for:
  - Building evidences from historic words in historical text collections
  - Building evidences manually
  - Generating transformation rules from evidences
  - Generating transformation rules manually
- Interaction between evidences and transformation rules
Chapter 7 Evaluation

- Identify potential usability problems

7.2 Methodology

In this evaluation a task-oriented evaluation methodology was adopted and methods aiming to the purposes of the evaluation (section 7.1) were developed. The following methods were chosen for this evaluation:

- **Tasks:** In order to test the work-flows of using the prototype described in chapter 5 a list of tasks was prepared (see A.1.3 in the appendix). The tasks were divided into two exercises that focused on the main goals of 'Rule Generator' (in section 1.2), building evidences and generating rules either automatically or manually by the user. The first exercise with ten tasks involved the interaction between building evidences and generating rules. The second exercise concerned the auto-evidences component of 'Rule Generator’. A closer look at the tasks will be taken in section 7.3.

- **Questionnaire:** Questionnaires help to get the opinion of the participant on specific aspects of usability. The questionnaire for this evaluation contained 28 questions (see A.1.4 in the appendix). The questions were divided into three groups. The first group of questions were related to the graphical user interface in general. The second group to the evidences section, and third group to the rules part. Participants’ answers were divided into five possible responses on a Likert scale as follows:

  - Strongly disagree
  - Disagree
  - Neutral
  - Agree
  - Strongly agree

- **Interview:** Interviews help to get the participant’s opinion of the tool. An interview of 5-10 minutes per participant was planned. The interview questions can be found in appendix A.1.5.

- **Observation:** In addition to direct observation of the participant, an eye-
tracker device produced by "SensoMotoric Instruments (SMI)"\textsuperscript{1} was used to monitor the eye movements as well as the motion of the cursor on the screen. An eyetracker is a device that allows measuring and recording of the motion, regression, fixation and the point of gaze of the eyes. It also allows the analysis of the recordings and the presentation of the results in different formats, for example charts, heat maps etc. During our observations notes were taken irrespective of whether the task was performed successfully and on those occasions when the participants’ behavior was unusual.

7.3 Experiment

Since the 'Rule Generator' was developed for German users, the evaluation was held in German. It was necessary that all participants of the evaluation were German native speakers. Ten students aged between 25 and 30 participated in the evaluation. The equipment for the experiment was provided by the department of Computer Science and Applied Cognitive Science of University of Duisburg-Essen. The application ran on a PC with the following specifications:

<table>
<thead>
<tr>
<th>Operating system</th>
<th>Windows XP Professional (SP3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Core 2 Quad 3Ghz</td>
</tr>
<tr>
<td>RAM</td>
<td>3GB</td>
</tr>
<tr>
<td>Java Runtime Environment</td>
<td>1.6.0 (20-b02)</td>
</tr>
</tbody>
</table>

Table 7.1: Evaluation equipment specifications

For the entire experiment a total period of 90 minutes per participant was planned. At first, a short introduction about the RSNSR project and the objectives of the linguistic tool was given to the participant. Afterwards the participant was handed an explanatory document of one page on the concept of the rules and evidences. A good understanding of what an evidence or a rule is, was necessary for the evaluation. Further explanation on these concepts was given to the participant if necessary. Instead of having a familiarisation period, the participant was handed two screenshots of the linguistic tool, one for the "Evidences" view and one for the "Rules" view. The screenshots included notations with short descriptions on

\textsuperscript{1}http://www.smivision.com/
the different sections of 'Rule Generator' (see Appendix A.1.1). Afterwards the participant was asked to read the consent form and sign it.

Then the tasks document was handed to the participant and he was asked to start working on the two exercises. With 1.a we wanted to investigate the text load dialog of the tool with the auto-evidences feature deactivated. In Task 1.b we wanted to investigate the construction of evidences from historical text documents and the usage of the text viewer. In 1.c, 1.d and 1.e we wanted to investigate the building of evidences manually without any historical text. With 1.f, 1.g, 1.h we wanted to investigate the interaction between the evidences and the rules views. in 1.i, and 1.j we wanted to investigate navigation of the rules within the "Rule selector" component of the tool and also creating rules manually using the "Rule visualization" component. With 2.a we wanted to investigate the text load dialog of the tool having the auto-evidences feature activated. In 2.b, 2.c and 2.d we wanted to investigate saving evidences from the evidences automatically created and suggested by the tool. In 1.f and 2.f the import of rules from an XML file was investigated. And finally in 2.k the editing of evidences in the "matched evidences table" was investigated. Participants took between 35 to 45 minutes to complete the tasks. During this, the participant was observed and for each task, notes were taken of his reactions and any unusual situations. With the help of the eyetracker it was possible to observe the participant’s eye movement and work flow on the screen and accordingly take notes when the participant had difficulties.

After the tasks were finished, the participant was asked to fill out the questionnaire. Finally, an interview of 5 to 10 minutes was held, during which all participants were asked the same questions: about likes and dislikes, what could be improved and for any feedback. During the interview the participants were asked about particular situations (about the unusual situations during working on the tasks).

7.4 Results, Analysis and Interpretation

In this section the results of the evaluation will be presented, analysed and interpreted to show to which degree the goals of the evaluation in section 7.1 were fulfilled. The eyetracker recordings will be presented using functions within the eyetracker software "BeGaze".
7.4 Results, Analysis and Interpretation

7.4.1 Observation and Interview Notes

The notes taken during the observations and the interviews were reviewed in detail. The statements made by several participants during the interview and the notes taken are shown in figure 7.1.

![Observation and Interview](image)

**Figure 7.1:** Observation and interview results

The chart in figure 7.1 illustrates the opinion of the ten participants, expressed during the interview after the experiment. Seven participants mentioned that the tool was easy to use when they were asked what they liked about the tool. Six participants mentioned the layout was well arranged. The missing ‘edit’ button for editing evidences in the 'Matched'-tab was confirmed by six participants. The ‘Add to dictionary’ button in the toolbar under the ‘Unmatched’-tab, was not clear enough for five participants. Four participants mentioned that the text segments in the 'Historic Text’ view were helpful and three participants found the segments too short. Three participants mentioned that the yellow highlighted words in the 'Historic Text’ view were helpful.
Chapter 7 Evaluation

7.4.2 Questionnaire

The questionnaire had four main sections. Eight questions in section 1 related to the general usability of the tool, ten questions in section 2 related to the building, collecting and editing of evidences. The questions in section 3 and 4 were related to the editing and visualization of the transformation rules. For this thesis, the results of the questions in section 1 and 2 are relevant. Sections 3 and 4 are discussed in the concurrent thesis [Kor10]. Figure 7.2 shows the average responses of the ten participants to the general usability questions. (1=Strongly disagree, .. , 5=Strongly agree).

![Figure 7.2: Participants’ average responses to questions about general usability](image)

From the chart above it can be seen that the question *The layout of the tool is well-arranged* (Q2.1) has an average rating of 4.3\(^2\) and *The tool was easy to use* (Q2.2) an average rating of 3.9. In addition, during the interview, participants were asked what they found good about the tool and seven of them mentioned that it was easy to use. Six participants said that the tool has a well-arranged

\(^2\)(1=Strongly disagree, 5=Strongly agree)
layout (see figure 7.1). These results demonstrated that the usability of the tool was satisfactory.

The average rating of 3.9 for the question *The tool was easy to learn* (Q2.3) shows that the learnability of the tool was acceptable. The average of less than 4.0 can be explained by the fact that the concept behind the tool was new to the participants.

The average rating of 4.0 for the question *Loading Dialog was well-arranged* (Q2.5) and 4.3 for Q2.6, give the loading dialog of the tool a satisfactory result of 4.15.

All participants found the font size of the tool satisfactory; the question *The font size of the tool was big enough* (Q2.8) was answered with an average rating of 4.9.

Figure 7.3 shows the average responses of the ten participants to the questions in block 2. (1=Strongly disagree, .. , 5=Strongly agree).

![Evidences View](image)

**Figure 7.3:** Participants’ average responses to questions about evidences

The purpose of the questions Q2.9, Q2.10, Q2.11 was to test the ability of the tool to build evidences from historical text collections. This was done using the
Chapter 7 Evaluation

Unmatched-tab of the tool (see figure 4.11). Building evidences from historical text documents achieved highly satisfactory results with an overall rating of 4.2.

Adding an evidence manually to the list of evidences had the highest average rating of 4.4 with Q2.13 (Adding an evidence manually was easy), which indicates high satisfaction from all participants.

In tasks 1.f and 1.g the participant was asked to edit a rule for a specific evidence. It was observed that all participants accomplished the task without difficulty; choosing the mentioned evidence in the list of evidences, searching for the mentioned rule in the rule selector and editing it in the rule visualization frame. These successful actions demonstrated that the required interaction between evidences and rules had been achieved and hence that the evaluation achieved its objective.

During our observations it was noted that most of the participants had some difficulty editing an evidence as asked in task 2.k (Change the modern word in the first evidence of the historical word "Befestiget" to "Befestigung"). Moreover, question Q2.15 (Editing an evidence was easy) had an average rating of 3.4 and 6 participants mentioned during the interview that an edit button was missing in the toolbar of the matched-tab. This was shown by a poor satisfaction rating by the participants for the task of editing an evidence.

7.4.3 Eye-Tracking

From the recordings of the eyetracker during task 2.k, the eye movement of the ten participants were analysed using "BeGaze". All ten participants showed similar behavior which is presented in the screenshot in figure 7.4.

The screenshot in figure 7.4 illustrates this behavior. After reading the task, as seen at point A the participant looks for the historical word "Befestiget". After finding the historical word, he looks for the modern word to be edited (see point B). Then the participant chooses the right evidence and searches for an edit button in the toolbar (see point C), instead of double clicking on the modern word in the table. The reason why the participant was searching for the edit button is that in task 2.i, when he was asked to edit a specific rule, he learned to edit it with a button in the rule visualization toolbar. Hence, the statement
of the 6 participants of a missing edit button in the toolbar is justified. For this reason, an edit button was implemented in the toolbar of ”Matched-tab”

In task 2.c *(Add the historic word ”Fixsternenhimmels” to the dictionary)* the participant was asked to insert a historic word into the dictionary. During the interview sessions five of the participants mentioned that adding the historic word to the dictionary was confusing. While observing participants we noticed that they could not find the ‘add to dictionary’ button in the toolbar of unmatched-tab. We assume the reason is that the icon of the button does not look like a dictionary. Hence the icon shall replaced by a more intuitive/self-explanatory icon.

During the interview sessions three participants mentioned that the text segments in the ”Historic Text” frame were not long enough. These participants did not know that clicking the pathnames above each text segment would show them the whole text. This can be seen as a usability problem that can be resolved by making the path look like a hyperlink, i.e. in blue and underlined.
Chapter 7 Evaluation

7.4.4 Bugs

During the observation, two bugs were discovered:

Bug 1: While working with the tool, the participant tried to load new files. Before showing the "loading dialog" the system asks the participant if he really wants to quit the current session. When the user clicks on the close button of the "JOptionPane" dialog, the loading dialog opens, instead of going back to the current session.

Bug 2: When adding a word from the list of "historical words" to the "dictionary", the word disappears from the list but remains in the "auto-evidences" list. When adding a word to the dictionary, the word should be deleted both from the list of historical words and the list of auto-evidences.

Summary

This chapter illustrated the user-based evaluation of the prototype 'Rule Generator' with ten participants. Each evaluation session had a duration of 90 minutes. First the RSNSR project and the prototype GUI were introduced with the help of screenshots. Then each participant had about 60 minutes to perform a list of tasks with the tool, during which the participants were observed and notes were taken on their behaviours. Afterwards the participants filled out a questionnaire, which was followed by an interview of 5-10 minutes.

The results of the evaluation and the subjective opinion of the participants yield that the tool was easy to use and the tasks were performed without a big effort. The participants were also satisfied with the layout of the tool and mentioned that it was easy to find the required graphical components to perform the tasks. The tasks of building evidences manually or from automatic evidences as well as the tasks to generate and manipulate transformation rules were performed successfully by all participants. The edit button in 'Matched'-tab was implemented after the evaluation.
Chapter 8

Conclusion

8.1 Summary

This thesis describes the successful development of a software application and associated interactive graphical user interface, which can be used by users with no technical and programming knowledge to build evidences and generate transformation rules.

A requirement analysis was carried out during which both the functional and the non-functional requirements of the user interface were determined. Then the user interface was designed using a detailed hierarchical task analysis and by creating and evaluating mockups. The results of the evaluation were incorporated into an improved design in stages. This was followed by a design of the functionality model of the user interface, which shows the functions to be supported by the tool and the required interfaces to the RSNSR project classes and to the GUI components developed in [Kor10].

‘Rule Generator’ offers three different ways of building evidences. Firstly, if the user wants to build evidences rapidly from a large historical text collection, this can be accomplished by means of the component ‘Auto Evidences’ (section 6.3). A list of evidences and the corresponding rule cores are displayed to the user. He can choose one or more evidences or, through a single click, the complete list of evidences and accept them. Secondly, if the user needs more control over the construction of evidences, this can be effected by means of the component ‘Evidences’ (section 6.4), in which the user can map potential historical terms with one or several modern ones suggested by the tool or his own variant. Thirdly, the user has the possibility of creating manual evidences by entering both historical and modern words manually (figure 6.11). In this case, the user needs also to specify the period and region to which the evidence belongs.
While building evidences, the user might need contextual help. This is provided by the component ‘Historical Text’ (section 6.5), in which sections of the historical texts are displayed with the detected potential historical words highlighted in yellow. With a single click, the user can also view the complete historical text if he so wishes.

For every evidence created, the transformation rules are displayed in the component ‘Rule Selector’ and visualized in the component ‘Rule Visualization’. These components were integrated in two separate InfoNode views in the class ‘MainFrame.java’ (section 6.1). The manual generation of new rules and the manipulation of existing rules can be carried out in ‘Rule Visualization’ component.

‘Rule Generator’ allows for both the importing and exporting of evidences and transformation rules. These functions have been grouped under a single menu in the menu-bar and share the same button in the tool-bar to make the user interface consistent (section 3.2.1). Nevertheless, the import and export functionalities of transformation rules were implemented in [Kor10], with the consequence that evidence and transformation rules are exported to separate XML files.

After the components developed in [Kor10] were integrated into the GUI prototype, a user-based evaluation was conducted. The participants were able to load historical texts, build evidences and edit the generated transformation rules without any difficulties. The results of the evaluation in section 7.4 show that the participants found the usability of the tool to be satisfactory.

## 8.2 Future Work

This section presents possible improvements in the usability of the GUI prototype developed in this thesis.

Portions of the historical text are displayed in the component ‘Text Viewer’. The size of these portions is defined in the implemented method to 40 words. More contextual information is displayed in a new window when the user clicks on the path name of the historical text above each portion. Such a process can be time consuming if there is a frequent need for such contextual information. This can be solved by making the size of the portion variable and making it possible for a user to change the number of the words through a small toolbar in the ‘Historic Text’ view. While reading portions of the historical text, the user might want to create an evidence from a specific word. A possible functionality is to allow the user to right-click on that word and create an evidence.
Appendix
A Appendix

A.1 Evaluation Documents

A.1.1 Introduction
Einführung


Morgenröthe → Morgenröte

Mit Hilfe des im Projekt RSNSR entwickelten Verfahrens können nun aus diesen Belegen Regeln generiert werden. Regeln bestehen immer aus einem find- und replace-Teil. Bei eingegebenen Wörtern wird dann die moderne Wortform nach dem find-Teil durchsucht, und sofern gefunden, durch den replace-Teil ersetzt, um so die historische Wortform zu erzeugen. Regeln für den obigen Beleg wären:

_ → h  t → th  te → the

Im Rahmen des Projektes RSNSR wurde das SmartEvidencer Tool entwickelt um diesen Prozess zu unterstützen.
A) **Evidences**: Der Evidences-Bereich dient zur Erstellung (im Tab „Unmatched“) bzw. Bearbeitung (im Tab „Matched“) von Belegen.

B) **Historic**: Hier werden die vom Tool gefundenen möglichen historischen Wortformen angezeigt.

C) **Modern**: Hier werden die vom Spellchecker generierten Vorschläge angezeigt

D) **Historic Text**: Hier werden, anhand der ausgewählten historischen Wortform, Abschnitte des geladenen Textes angezeigt.
1) **Rule Selector:** Der Rule Selector dient der Auswahl einer Regel und ermöglicht das Browsen der Regelbasis. Die selektierte Regel wird in der Rule Visualization angezeigt.

2) **Rule Visualization:** Die Rule Visualization zeigt die für die selektierte Regel relevanten Ober- und Unterregeln an. Sie unterstützt Standardoperationen wie Panning und Zooming.

3) **Regelmodifikation:** Modifikationen in der Regelbasis werden über die Toolbar durchgeführt.
A.1 Evaluation Documents

A.1.2 Guidelines
Leitfaden Einführung Evaluation

- Allgemeine Erklärung über Beleg- und Regelteil
  o Laden von Textdateien
  o Dictionary
  o Auto Evidences
- Belege
  o Evidences Frame
    - Unmatched
      - Historische Liste
      - Suggestions unter modern
      - Daraus Belege erzeugen
      - Add to dictionary
    - Matched
      - Tabelle (im "unmatched"-Tab erzeugte Belege, sehen, filtern, löschen oder editieren)
      - Neue Belege hinzufügen
  o Text Frame
    - Abschnitte / Gesamten Text sehen (Anhang ausgewählte historische Form)
- Regeln
  o Erklärung zum Rule Selector
    - Auswählen einer Regel -> Visualisierung in Rule Visualization
    - Unterschied: All / by selected Evidence
    - Filter
  o Erklärung zu Rule Visualization
    - Kurze Erklärung Focus+Kontext
    - Erklärung Toolbar
    - Eingabemöglichkeiten: Panning / Zooming
A.1 Evaluation Documents

A.1.3 Tasks
Aufgabe 1

a) Laden Sie die Texte „1758_berlin_1.txt“ und „1758_berlin_2.txt“, die sich in dem Ordner „c:\Historische_Texte\“ befinden (Region und Period finden Sie im Dateinamen) und deaktivieren Sie dabei die Funktion der automatischen Beleggenerierung.

b) Wählen Sie für alle Belege im „unmatched“-Tab die korrekte moderne Wortform aus und speichern Sie diese. Falls historische Wortformen ohne korrekte Vorschläge für die moderne Wortform vorhanden sind, geben Sie bitte einen eigenen Vorschlag ein.

c) In der „matched“ Tabelle fügen Sie die historische Wortform „Vortheil“ ein.

d) Erstellen Sie einen neuen Beleg für die historische Wortform „Fehlet“ mit der modernen Wortform „Fehlend“.

e) Löschen Sie den Beleg der historischen Wortform „Erkläret“ und der modernen Wortform „erklärt“.

f) Importieren Sie die Regeldatei „Berlin_Regeln_1.xml“ aus dem Ordner „c:\Historische_Texte\Regeln\“.

g) Für den Beleg der historischen Wortform „Dienet“ und der modernen Wortform „dient“, löschen Sie die Regel „n → ne“.

h) Für den Beleg der historischen Wortform „Hiebel“ und der modernen Wortform „hierbei“, löschen Sie die Regel „er → e“ und alle ihre Unterregeln.

i) Selektieren Sie im Rule Selector die Regel „e → “, und finden Sie in der Rule Visualization die Regel „dene → dne“. Löschen Sie diese.

j) Erstellen Sie eine neue Regel „te → the“.

Aufgabe 2

a) Löschen Sie die aktuelle Session und laden Sie die Texte „1758_berlin_3.txt“ und „1758_berlin_4.txt“, die sich in dem Ordner „c:\Historische_Texte\“ befinden und aktivieren Sie dabei die Funktion der automatischen Beleggenerierung.

b) Speichern Sie drei korrekte automatisch generierte Belege.

c) Fügen Sie die historische Wortform „Fixsternenhimmels“ dem Wörterbuch hinzu.

d) Aktivieren Sie die automatische Beleggenerierung und speichern Sie drei neuen Belege.

e) Wählen Sie für alle Belege im „unmatched“-Tab die korrekte moderne Wortform aus und speichern Sie diese. Falls historische Wortformen ohne korrekte Vorschläge für die moderne Wortform vorhanden sind, geben Sie bitte einen eigenen Vorschlag ein.

f) Importieren Sie die Regeldatei „Berlin_Regeln_2.xml“ aus dem Ordner „c:\Historische_Texte\Regeln\“.

g) Lassen Sie sich im Rule Selector alle Regeln anzeigen.

h) Finden Sie die Regel „kon → con“ und löschen Sie diese und alle ihre Unterregeln.

i) Finden Sie die Regel „er → e“ und modifizieren Sie diese zur Regel „ar → a“.

j) Finden Sie die Regel „uss → uß“ und erstellen Sie eine neue Regel „kuss → kuß“ auf deren Basis.

k) Ändern Sie die erste moderne Wortform der historischen Wortform „Befestiget“ in „Befestigung“. 
A.1 Evaluation Documents

A.1.4 Questionnaires
Fragebogen Versuchsperson

Versuchsperson:

F 1.1 Alter: ___________________________________
F 1.2 Geschlecht: _______________________________
F 1.3 Muttersprache: _____________________________
F 1.4 Studiengang: ______________________________

F 1.5 Seit wie vielen Jahren benutzen Sie bereits Computer? ______ Jahre

F 1.6 Wie häufig verwenden Sie den Computer?

<table>
<thead>
<tr>
<th>Täglich</th>
<th>Mehrmals pro Woche</th>
<th>Einmal pro Woche</th>
<th>Mehrmals pro Monat</th>
<th>Einmal pro Monat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Universität Duisburg-Essen Informationssysteme
Abschlussfragebogen

<table>
<thead>
<tr>
<th>Allgemeine Fragen:</th>
<th>Gar nicht</th>
<th>etwas</th>
<th>sehr</th>
</tr>
</thead>
<tbody>
<tr>
<td>F 2.1 Das Layout des Tools war übersichtlich.</td>
<td>° ° ° ° °</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2.2 Die Bedienung des Tools war einfach.</td>
<td>° ° ° ° °</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2.3 Die Erlernung des Tools war einfach.</td>
<td>° ° ° ° °</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2.4 Die Bedeutung der Icons im Tool war eindeutig.</td>
<td>° ° ° ° °</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2.5 Der Load Dialog war übersichtlich.</td>
<td>° ° ° ° °</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2.6 Es war einfach Texte zu laden.</td>
<td>° ° ° ° °</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2.7 Die Bedienung des Tools war flüssig.</td>
<td>° ° ° ° °</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2.8 Die Schriftgröße des Tools war groß genug.</td>
<td>° ° ° ° °</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fragen zu Belege (Evidences)</th>
<th>Gar nicht</th>
<th>etwas</th>
<th>sehr</th>
</tr>
</thead>
<tbody>
<tr>
<td>F 2.9 Der <strong>unmatched-tab</strong> war übersichtlich.</td>
<td>° ° ° ° °</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2.10 Die Erzeugung der Belege war einfach.</td>
<td>° ° ° ° °</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2.11 Die Anzeige der modernen Vorschläge war übersichtlich.</td>
<td>° ° ° ° °</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2.12 Der <strong>matched-tab</strong> war übersichtlich.</td>
<td>° ° ° ° °</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2.13 Das manuelle hinzufügen eines Belegs war einfach.</td>
<td>° ° ° ° °</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2.14 Das Filtern der Belege war hilfreich.</td>
<td>° ° ° ° °</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2.15 Das Editieren eines Beleges war einfach.</td>
<td>° ° ° ° °</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2.16 Die Tabelle der Belege war übersichtlich.</td>
<td>° ° ° ° °</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2.17 Der <strong>Automatic Evidences</strong> Dialog war übersichtlich.</td>
<td>° ° ° ° °</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F 2.18 Die Text Abschnitte im <strong>Text Viewer</strong> waren lang genug.</td>
<td>° ° ° ° °</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Universität Duisburg-Essen Informationssysteme
A.1.5 Interview Guidelines
Interview Leitfaden

1. Was fanden Sie gut?
2. Was fanden Sie nicht gut?
3. Was könnte man verbessern?
List of Figures

1.1 Examples of evidences and transformation rules .................. 2
2.1 Single file processing user interface ............................... 6
2.2 Manual processing of variants .......................... 7
2.3 Batch processing user interface ................................. 8
2.4 LeXtractor in corpus mode ................................ 10
2.5 Confirm derivations dialogue ................................ 11
2.6 LeXtractor in document mode ................................ 12
2.7 Evidencer 2.0 ............................................. 13
5.1 HTA Diagram ............................................. 28
5.2 High Level Components ................................ 30
5.3 Mock-Up presenting the layout of the main components ....... 32
5.4 'File' Menu prototype ................................ 33
5.5 Main toolbar prototype ................................ 34
5.6 Evidences view prototype ................................ 34
5.7 View / Edit an evidence prototype ............................ 35
5.8 Rules View ............................................. 36
5.9 Work-flow diagram .................................... 37
6.1 Example of sub-menus and menu-items in the menu-bar. ....... 42
6.2 The main toolbar ...................................... 42
6.3 Layout Overview of 'Rule Generator' ..................... 44
6.4 Load historical texts dialogue ............................... 45
6.5 Loading files feedback ................................ 47
6.6 Auto Evidence Window ................................ 48
6.7 Accepting auto evidences ................................ 49
6.8 Build Evidences View ................................ 50
6.9 View and edit evidences ................................ 51
6.10 Filter evidences ...................................... 51
6.11 Add evidence manually dialogue .......................... 52
6.12 Historical Text View .................................. 53
List of Figures

7.1 Observation and interview results ........................................ 59
7.2 Participants’ average responses to questions about general usability 60
7.3 Participants’ average responses to questions about evidences ....... 61
7.4 Eye-movements of participant 3 during task 2.k ....................... 63
List of Tables

7.1 Evaluation equipment specifications . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
Bibliography


