Design and Implementation of a fault tolerant form processing application using machine learning

Master's Thesis in Computer Science

submitted
by

Christoph Neubauer

born 23.06.1991 in Homburg (Saar)

Written at

Lehrstuhl für Mustererkennung (Informatik 5)
Department Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg.

in Cooperation with

Universidade Federal do Parana

Curitiba

Advisor: PD Dr.-Ing. habil. Peter Wilke

Second Advisor: Prof. Luiz Eduardo S. Oliveira

Started: 12.09.2016

Finished: 14.03.2017

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Richtlinien des Lehrstuhls für Studien- und Diplomarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Erlangen, den 2. März 2017

Übersicht

Abstract

Contents

Chapter 1

Introduction

Optical Character Recognition (OCR) has been the topic of research for many years, even decades. Several workshops, papers and journals have been published, conferences hold on issues in this field. While there are still many open problems (for instance the accurate recognition of arabic texts, symbols, mathematic formulas or handwriting), the knowledge in this area has already led to the development of several highly accurate systems (especially based on English text).

While already a lot of companies use those systems, there is still a majority of others that do not. But, as these systems grow more accurate each year, it is very likely that the need for ocr systems will grow.

As companies are getting connected and globalized, more and more data have to be handled. Modern keywords such as ’Big Data’ or ’Data Mining’ show, that there is currently a need for solutions to handle those data.

One of those problems is the management of invoices. Companies all over the world have to manage not only the invoices they generate, but also the ones they retrieve, e.g. from their suppliers. While there are already ERP-systems such as SAP ERP 6.0 that are capable of the generation of invoices, especially invoices of other companies are not that easy to process due to the differences between those documents. In addition to that, especially small but also medium-sized companies are mostly not able to afford such systems.

In order to facilitate and accelerate the process of invoice recognition electronic invoice formats have been introduced. If invoices are sent in such a format, a company is recognize the required fields and handle this invoice. Again, this is often the case for big companies, that have defined contracts with their suppliers or customers and have therefore been able to define an electronic invoicing format to automatically process an invoice.

For every other company, there are still problems: Not every invoice is sent in an electronic format, some are still sent as a normal pdf document or even per post.

1.1 Motivation

Although still invoices exist, that do not meet any electronic invoicing standard, it is to be expected that electronic invoice formats will be a future standard for all invoices to be sent. But even though electronic invoice formats have already been introduced, there are still a variety of formats and no real standard defined. Therefore, depending on the country or area, different formats are used. To address this issue inside the European Union, the Forum fÃ¼r elektronische Rechnung Deutschland (FeRD) developed a new format which will be very likely a new de-facto standard for companies inside the European Union. Small and medium sized companies can make use of this format as well as bigger companies. Chapter 2 will explain the different levels of the format more deeply.

While hardware parts and devices improve over time, we are now on a point were even household computers (also known as personal computers) are able to process complex calculations in lesser time. With this in mind, it would be a profit to automate invoice processing, even for small companies. Invoices that are processed manually does not only need employees, but also much more time and is (due to the human factor) error-prone.

With the use of machine learning techniques it should be possible to develop an application that works on a personal computer, can handle lots of invoices and transforms them into an electronic invoice format.

1.2 Task

The task of this master thesis is to develop an application which can handle various invoices that are present as a pdf file, extract necessary invoice information and transform and store those invoices enriched by the electronic invoice format. The advantages and disadvantages of this format should be evaluated first. During the processing of the invoice, optical character recognition should be used to extract the information from the file. During this process, machine learning should be used where it enables the most benefit for the application. The occurrence of errors during the scan process should also be handled by the application itself. The stored invoices should be retrievable again, enhanced and conform with the electronic invoice format, so that it is possible to process them further.

1.3 Related Work

Other relevant academic work and how it differs from this work, for example "textual" citation, as shown in "parenthesis" citation

1.4 Results

What has been achieved in this work?

1.5 Outline

This document is structured in the following way: In the beginning, several electronic invoice formats are presented, explained and compared against each other. Important criteria for the selection of a format are defined and based on those criteria a decision is being made.

After that, we will explain how we want to process a file to a document in the selected electronic invoice format. Chapter 3 will deal with OCR, the available systems at this time as well as a comparison between them and the selection of one of them (including the explanation why this selection has been made). As the application should learn and improve results over time, we will also deal with machine learning techniques and choose an appropriate one. Chapter 4 will focus on this issue.

From this point on, we have a good understanding about what we want to achieve, with which technologies and methods as well as necessary tools or frameworks for that. Chapter 5 will now discuss several use-cases of the application and show the architectural concept of the application.

The following sections will deal with each module and explain it in-depth. The last section will discuss problems that occurred during the implementation.

-¿ data tests?

In the end, chapter 6 will conclude about this thesis. The resulting application will be explained briefly again. Issues that are still open as well as ideas that could improve the application are listed.

1.6 Acknowledgments

During the implementation of the application and the creation of this document, several people helped me to achieve this presented work. I would like to thank some people in particular:

To Dr. Peter Wilke, who not only supervised my work, but also put thoughts to things that i have not considered before but were crucial for the application.

To Prof. Dr. Oliveira, who supervised my work during my stay in brasil and who gave me good input especially in the field of OCR.

To Prof. Daniel Weingaertner, who managed my stay in brazil, enrolled me in the university and organized all the necessary documents.

To Daniel Stemler whose engagement enabled me to gain access to over thousand invoice documents in order to get a reasonable amount of data to test on.

And to several other friends that helped me or supported me with advices or discussions about technologies or to clarify my understanding regarding a specific approach.

Chapter 2

Electronic invoice formats - A comparison

During the technologization of companies over the world, electronic invoices (also known as e-invoice) have become more and more important. E-Invoicing offers companies the possibility to improve their business processes, making invoicing faster and more efficient and enables a direct connection to other tools like ERP-Software.

To enable companies these benefits and in order to make the communication between companies even possible a comprehensive standard has to be defined. With an invoice standard at hand, companies can use invoices from their business partners and read them into their systems (in case of B2B).

There are several invoice standards in action at the moment. The next section will deal with the most important ones and describes them as well as pointing out the benefits and drawbacks of the format. After that, the next section defines criteria that are relevant for the application and how to measure them. In the last section, these criteria are applied on the formats defined in section **??** and compared against each other. Eventually a decision regarding the usage of one of these formats is made.

2.1 Description of leading formats

Each of the following subsections will present an electronic invoice format. The history of the format, as well as the current version and, if found, the future promise will be explained. As there exist many different formats, it is out of the scope of this thesis to describe them all. Instead, we will pick a few that we think are either important, promising or especially related to the region (Germany and the European Union).

2.1.1 UN/EDIFACT

EDIFACT is a well-established [?] subset of standards from CEFACT regarding the electronic interchange of structured data. The word 'EDIFACT' is an acronym from 'EDI' which stands for 'Electronic Data Interchange' in combination with 'FACT' (for Administration, Commerce and Transport). It is developed and maintained by the United Nations Economic Comission for Europe (UNECE) [?].

The European Commission states that the UN/EDIFACT INVOIC message has been a cornerstone in electronic invoicing over the past years [?, ?].

There are several subsets of EDIFACT that have been developed for different industries. For instance, the chemical industry uses CEFIC/ESCom[1] as their standard, while automotive industry is in charge with ODETTE/FTP2[2].

EDIFACT has different message types such as ORDCHG for a request to change an order or PAYORD which contains a payment order. In the context of this thesis, the message type INVOIC (containing an invoice) is the most interesting one.

2.1.2 XCBL

The XML Common Business Library is an extension of the CBL which originally has been devloped by Veo Systems Inc. [?]. The company has been bought by Commerce One Inc. in 1999 [?], page 29.

xCBL currently exists in version 4.0 (since 2003)[3]. Since the company has gone bancrupt in 2004 **??** it is not very likely that this format gains more interest in the future.

[1] see also: https://www.cefic.org/Industry-support/Implementing-reach/escom/

[2] see also: https://www.odette.org/services/oftp2

[3] see also: https://www.xcbl.org

2.1.3 OASIS/UBL

UBL stands for Universal Business Language and is being developed by OASIS. The current version is 2.1 and is normed by the international standardization organization[4].

Several countries developed their own subset of this format. Especially interesting in this case is a project called PEPPOL (Pan-European Public Procurement Online project) that aims at developing a format for public sectors in the whole European Union[5].

Also interesting in the context of invoice interchange is the UBL-based project called *simpler-invoicing* that aims at connecting ERP systems with accounting and e-invoicing software by providing an own invoicing standard[6].

2.1.4 ZugFerd

This invoice format has been published initially in 2014 [**?**]. The name is a german acronym, containing the name of the corresponding forum (FeRD). It can be translated to ”Central User Guide of the Forum for electronic Invoicing in Germany”.

Although this invoice format is rather young it tries to fulfill the directive 2014/55/EU of the european parliament [**?**] while still being flexible and simple. This directive states that the use of electronic invoice formats should be adopted by all member states of the european union until the 27. of November 2018 [**?**, **?**].

The approach of the ZugFerd-format enables not only big companies to work with that format, but also smaller and medium companies (SME's) that are in need of such a format but are normally not able to implement a complex electronic invoice standard. Furthermore three levels of conformance are defined: Basic, Comfort and Extended. Each of those levels have a different amount of required information fields, that have to be set in order to be a valid ZugFerd-format. Nevertheless, in all of the three formats, it is possible to define more information in free text fields.

This enables extensibility of the format and the possible business areas in which this standard can be used.

[4] see ISO/IEC 19845:2015

[5] see also: https://www.peppol.eu/about_peppol/about-openpeppol-1

[6] see also: www.simplerinvoicing.org/en/

The German Forum for electronic invoice (FeRD) states that this format has been accepted as a core standard in Germany to be used in the future such that every company, that wants to start business relations with a german company, has to use this standard **??**. Furthermore, the possibility to extend this standard to all European Countries is in sight, as stated in **??**.

2.2 Definition of decision criteria

While the standards defined in the section before focus on specific areas or try to combine multiple fields, this section defines the criteria that are most relevant for the application that is developed.

2.2.1 Future Potential

One of the major criteria for a suitable invoice standard should be its future potential. The application should use an electronic invoice standard that is still used in 5-10 years. Therefore, any standard that is going to be replaced should not be considered useful. The likelihood of a standard to be still used in 10 years will be defined by this criterion.

2.2.2 Relevance in Germany (and Europe)

As this thesis is being written at a German university, the chosen standard should be relevant in Germany. The more countries make use of this standard (especially European countries) the higher the relevance of this standard. On the other Hand, standards that are not of interest for Europe should be excluded.

2.2.3 Extendability to more countries

The possibilities of a standard to be used in other countries will also affect its importance over the next decades. Standards that only suits the requirements of one country are not important enough. The focus lies on standards with a wide (possible) range of countries to be affected, instead.

2.2.4 Extendability of the standard itself

Last but not least, the extendability of the standard itself is an important criterion. The world is changing and new requirements are coming while older ones are getting broken up. A valuable standard should be able to deal with these changes and should be extensible towards new requirements, or special requirements in specific business areas.

2.2.5 Complexity

The complexity of the standard is important for this thesis too. Not only is the development of the application limited by time, but also makes a complex standard it hard to understand it and less error-prone.

2.3 Comparison and decision finding

2.3.1 Application of the criteria

2.3.2 Decision and explanation

Chapter 3

OCR - State of the art, possibilities and drawbacks

During the processing of a form, the image of it is not only scanned. To retrieve additional information and work it we will use Optical Character Recognition (OCR).

The process of retrieving data, for instance in form of characters or numbers, requires several steps. In the beginning, the image to be processed is converted to a gray-scale image. Then preprocessing takes place. In this process several algorithms such as noise-reduction and the canny-edge-algorithm are applied on the image to reduce irritating and / or unnecessary information in the image and to enhance contrast.

After that, features are extracted. Those features are single characters whose vectors are defined afterwards. With the information about the feature vector it is possible to classify each character (and to detect which character of the alphabet is most likely to be represented by the feature). When all characters have been classified, post-processing takes place. Here possible failures can be corrected for example by comparing words with a predefined dictionary. These steps are also shown in figure 3.1:

The whole process of OCR is part of research since decades and several papers, dissertations and books have been published on the matter. Developing our own OCR algorithm would not only exceed the size of this master thesis, but also most likely retrieve less successful results than already developed and improved algorithms. Instead, this chapter will introduce several available OCR algorithms and compare open source solutions to find the best fit for our need.

3.1 Currently available OCR algorithms

OCR has been of interest for companies over many years. Hence we expect several possible solutions we could choose. As the amount of solutions can easily be very high we want to reduce the presented solutions to a maximum amount of 10.

Each description of a solution will contain information about the license used, the supported operating systems, programming languages used as well as if there is a software development kit. Also general information about the company will be given, the currently released version of the solution and the release date and, if existent, the number of languages supported.

3.1.1 ABBYY Fine Reader

ABBYY Fine Reader is a proprietary solution from the identically named company ABBYY founded in 1989. It is usable for all three operating systems, whereby linux-distributions are only supported as a command-line-interface. ABBYY supports a SDK for all three operating systems. Although it is written in C/C++ there exists a wrapper for java development for Linux and Windows(TODO: Link fÃ¼r Abbyy broschÃ¼re version 11).

The SDK is currently in Version 11, it supports 185 languages, the latest update was on 03.10.2016.

3.1.2 Anyline SDK

Anyline is an Austrian company founded in 2013 who aim at OCR solutions for mobile systems. They offer their SDK as a free license for non-commercial use. However, as they are focused on mobile systems, they do not explicitly support Windows-Desktop, Linux or MacOS.

It can be developed with Java and Objective-C as well as Swift, C# and Javascript. The current version of the SDK is 3.8.1 and has been released on 13.01.2017.

Currently supported are two languages: English and German.

3.1.3 Asprise OCR

Asprise has been founded in 1998 in Singapure. The company offers OCR SDKs in various programming languages (Java, C#, VB.NET, Python, C/C++ and Delphi Pascal). The SDKs are under loyalty-free license and therefore proprietary. More than 20 languages are supported, English and German are included. The SDK supports Windows, MacOS and Linux, whereby support of multiple operating systems at once increases the price.

3.1.4 GOCR

GOCR is an OCR application started by Joerg Schulenburg in 2000. The program is developed under the GNU Public License.

The latest version is 0.5 and has been released in March 2013. It is working under Windows, Linux and MacOS. The code is written in C, but is not known if there is a SDK which enables usage of the application inside another application. The number of supported languages is also unkown.

3.1.5 LEADTOOLS

Leadtools is an American company founded in 1990 and offers various products in the range of document and image processing.

The Leadtools OCR Engine can be used as an SDK and integrated in another application. Development with the SDK is possible with C# and VB as well as C/C++ and Java (and some others). The engine supports more than 40 languages, containing German and can be used on Windows, Linux and MacOS.

The current version of the SDK is 19 and has been released in December 2014.

3.1.6 MathOCR

MathOCR is a document recognition system written in Java with focus on formulas. The MathOCR project started in March 2014 and is based on the GNU General Public Liense.

The current version of the application is 0.0.3, which was released in May 2015 and is therefore still in a pre-alpha status. It can be used on Windows, Linux and MacOS. The amount of supported languages is not stated on the project page.

3.1.7 OCROpus

The OCRopus Open Source OCR System is an open source system developed and maintained by the German Research Laboratory for Artificial Intelligence under guidance by Thomas M. Breuel. It is licensed under the Apache 2.0 License.

The command-line application is written in Python and C++ and only supports Linux as Operating System. It currently uses the Tesseract as a text line recognizer but will be replace it in the future.

The current stable version is 1.0 and has been released in November 2014. The amount of supported languages is unkown, whereby it is able to work with latin-based languages.

3.1.8 OCRad

OCRad is a free OCR application under the GNU Public License and part of the GNU project. Antonio Diaz Diaz developed the application since 2003.

The current version is 0.25 and has been released in April 2015. It comes as a stand-alone console application but can also be used in the background by other applications.

3.1.9 OmniPage Capture SDK

The Nuance Communication Inc. offers an OCR tool called OmniPage Capture SDK, which enables document processing on Windows, Linux and MacOS. Depending on the underlying operating system it supports C/C++, Objective-C or C# and VB.NET.

The current version is 20 and has been released in 2016. It supports over 120 languages (German included).

3.1.10 Tesseract

The Tesseract OCR Engine historically was an early project developed by Hewlett Packard between 1984 and 1994. In 2005, it was put on an open source license. It is currently maintained by Google under the Apache 2.0 license.

Tesseract is originally written in C/C++ and can be used on Windows, MacOS and Linux. There also exists a wrapper which allows development in Java, the open source project Tess4J.

The current stable version of the Tesseract is 3.04.01 and has been released in February 2016. It supports over 100 languages (including German).

3.2 Comparison between open source algorithms

While several companies exist that offer good OCR libraries and SDKs, we have to stick to free software, since we are not able to afford a proprietary license. In a later state of the application, it could be possible to switch to a proprietary solution in order to increase our OCR efficiency. Until then, we will decide for the best fitting open source algorithm library and improve our efficiency by preprocessing the forms ourselves. This is explained in Chapter 4, Module 1.

Upon the 10 presented solutions, only 5 are free for use. These are: GOCR, Tesseract, OCROpus, MathOCR and OCRad.

The following table shows the named solutions and shows their differences regarding their version, the latest release date, the supported programming languages and operating systems as well as the license they are put on:

MathOCR is a relatively young application and therefore in a pre-alpha state. OCRad and GOCR are one step closer to the first major release. OCROpus has reached that state on November 2014. Tesseract is already on Version 3.04 and has recently released an alpha version of 4.0.

While Tesseract, OCROpus and OCRad are supporting C++, GOCR is only working with C whereas MathOCR is only Java. Tesseract is supporting C and Java as well (while using Tess4J as a wrapper). OCROpus instead is working with Python, too.

All solutions support Windows, Linux and MacOS except OCROpus, which is only working on Linux.

While GOCR, MathOCR and OCRad are licensed under the GNU Public License, Tesseract and OCROpus are licensed under the Apache 2.0 License. The difference between those two is mainly the following: Applications that are developed under usage of another program under the GNU Public License have to be licensed under the GNU Public License as well. The Apache 2.0 License allows usage of other application and enables free choice of licensing, but requires the mentioning of the underlying use of an Apache 2.0 licensed application.

3.3 Decision finding and explanation

In the beginning of this thesis, it was defined that the application should work on a Linux based operating system and be written in Java. While all presented solutions support Linux as an Operating System, not all of them work with Java as a programming language. In addition to that, OCROpus only supports Linux, which is fine for the current focus of the application, but could be of an issue later on if it should be ported to another operating system.

MathOCR is in a pre-alpha state which makes it difficult to use due to several missing functionalities and persistent bugs in the code. As it is mostly focused on mathematical equations and formulas, we will not consider MathOCR any longer, even though it supports Java as a programming language.

As explained in section 3.2, the GNU Public License requires our application to be licensed under the GNU Public License as well if we use another code which is licensed under this license. Therefore, the Apache 2.0 license is considered better, as it allows us to decide about the license for ourselves.

In the interest of the application, we want to use solutions that are up-to-date and are still under development. Therefore, the latest release date gives us insights about the activity on a project. Since a lot of open source projects suffer from missing developers, we expect longer development cycles. But the last version of GOCR has been released around 4 years ago. Hence we consider GOCR as not up-to-date anymore.

The following table shows the solutions again, but with underlying colors regarding their ability to fit to our problem. Green is used as a best fit, whereas red significates a major problem. Yellow shows that this attribute is not as good as others but no kick-out criterion.

As shown in the table, MathOCR will not fit our needs as it is a pre-alpha version. GOCR is

Application	**GOCR**	**Tesseract**	**OCROpus**	**MathOCR**	**OCRad**
Version	0.5	3.04.01	1.0	0.0.3	0.25
Release Date	03.2013	02.2016	11.2014	05.2015	04.2015
Supported Programming Languages	C	C/C++, Java (with Tess4J)	C++, Python	Java	C++
Supported Operating Systems	Windows, Linux, MacOS	Windows, Linux, MacOS	Linux	Windows, Linux, MacOS	Windows, Linux, MacOS
License	GNU Public License	Apache 2.0	Apache 2.0	GNU Public License	GNU Public License

Table 3.1: My caption

outdated and also supports only C as a programming language. OCROpus and OCRad only support C++ (and in the case of OCROpus Python). In addition to that, OCROpus could be of a problem when porting the application to other operating systems whereas OCRad is licensed under the GNU Public License.

Hence the Tesseract seems to be the best fit for our application. It is consistently updated and improved and by the history of it, the application itself has grown mature. The possibility to work with Tess4J enables the usage of it with Java. Multiple operating systems are supported and the Apache 2.0 license enables us to decide for ourselves under which license we will put the application.

Chapter 4

Machine Learning

The field of Machine Learning contains concepts how computers can obtain information without explicitly programming this kind of information retrieval. These concepts of ”Learning” can be divided in three main categories: Supervised learning, Unsupervised learning and Reinforcement Learning.

Supervised learning always deals with a user that ”feeds” input to the program as well as desired output. The program should recognize patterns that lead from the given input parameters to the desired output. Unsupervised learning instead, is an approach where the program does have an input and needs to find a structure in those data. The finding of some pattern can be the goal of the program itself.

Using reinforcement learning, every output of the program is being valued by the user again. Output that has been found correctly will be strengthened, whereas incorrect values will act repulsive on the algorithm. After multiple iterations of this process, the program can find the best answer (but not always the correct answer) using the attracting and repulsive values.

Our goal is to use one machine learning technique in order to improve the outcome of our application. One major objective that can be addressed with Machine Learning is the relation between accounting record positions and how they are assigned to all the accounts that are important for this position. We identify two major problems regarding this classification:

1. What does a position represent?

2. Which accounts should be assigned to this position?

As we are processing an invoice, we will retrieve a position as a String. An accountant would be able to identify the position (which means a semantic identification of the object) and assign it to the accounts that are important in this matter. But, as there is no concrete rule which position belongs to which accounts, every company can apply this position to different accounts.

For instance, the maintenance of a car in the car pool of a company could be booked as car costs, or (if the company defines it more specifically) as maintenance, car parts and worker time. Hence we need an algorithm that is capable of the following:

1. Assign involved accounts depending on the user (-¿ allow different account structure)
2. Learn relationships between a string and a set of accounts

While the algorithm should be able to deal with those problems, we will have another problem to deal with: OCR errors (e.g. ”CAB” instead of ”CAR”) and similar words (e.g. plural words such as ”apples” instead of ”apple”).

Keeping those constraints in mind, we can start thinking about a Machine Learning technique that satisfies our goal or at least helps us to reach it. To narrow our search, we also have to think about automation. As this application should be able to reduce the time an accountant needs to process an invoice, we want to make this process as automatically as possible. Using a supervised machine learning method would lead to an application, that requires to validate each invoice every time. Hence supervised machine learning algorithms will not be considered here.

4.1 Naive Bayes on accounting records

An invoice always contains one or more positions. Those positions are the reasons why the invoice even exist. But, depending on the company, those positions can be accounted in different ways. While one company would use only two accounts, another company could split the corresponding value on one side into two (or more) accounts. This behavior is dependent on the position. But same positions are accounted the same way. Therefore, we want a machine learning approach that is flexible enough to address this issue depending on the way a position has been booked before, but is also able to learn and to correctly classify a position.

Chapter 5

Implementation of the application

The application is structured by different packages. Each of them providing a specific benefit to the program as a whole. Before speaking about those modules, we will talk about the actual requirements of the application. After that, each module will be explained in detail and how it works. After that, the last section will deal with problems and possible solutions.

5.1 Requirements definition

Focus of this application is the possibility to automatically process forms and retrieve the data of the forms. Hence the application should be able to deal with several files and process them without human help. But, since there is a big variety of forms and every company have different structures, retrieving all necessary information can fail. If this happens, a user has to review scanned documents that contain errors. If it doesn't fail, data should be stored without any additional help.

To improve the process of gathering data, a machine learning approach should be implement that facilitates retrieving data and to speed-up form processing over time.

Output of the application should be a storage of the processed forms, appended with electronic invoice information that is valid against the basic- or comfort-level of the ZugfErd-Invoice standard.

5.2 Definition of modules

Following the Separation of Concerns principle (SoC) we want to separate all logical parts of the application into different modules. What the application will do is to take an invoice, read it (1), extract the information out of it (2), improve the information by using machine learning techniques (3) and eventually convert it to the ZugFerd-format (4). Hence we will define four different modules:

1. OCR: After the user has passed an document to the application, this module will process the document and read it using OCR techniques. Therefore, this module will be named ÖCR.̈

2. Extraction: This module will deal with the business logic regarding the retrieval of information from the processed document. Therefore, it will also give input and get output from the third module.

3. ML: In this module we will implement methods to improve future information extraction.

4. Transformation: Eventually, the extracted information and the processed document will be transformed into a new electronic invoice that is conform with the ZUGfERD-format.

5. GUI: In order to facilitate the process of entering and retrieving invoices, another module will be used that deals with all sorts of user interaction. As this application will have an graphical user interface, we will call this module GUI.

5.3 Architectural concept

The application will make use of several design patterns. One used pattern on an architectural level is the MVC-pattern. Hence the graphical user interface will be steered by a controller which retrieves data from the database and shows it to the user using javaFX and .fxml-Files. Input and changes the user makes in the view will be transported by the controller to the model which is stored in the database again.

To access the database we will use classes for each business object. The package BO contains classes that represent a table. A data-access-object (DAO) will be used to retrieve data from the database. To do that, this application will also make use of an object-relational mapping framework (Hibernate) which facilitates the conversation between table data and java objects.

5.4 Module 1 - OCR

The OCR module deals with the processing of the document. Therefore, we will use Googles Tesseract as described in chapter 3. In order to use it, we use Tess4J as a Java wrapper. TesseractWrapper.java is the class that initiates a tesseract instance. With initOcr() the tesseract instance is getting called. It returns a String as result.

We set HOCR to true, which means that our output will not only be a String containing the processed words, but in a structured way. HOCR is a xml-structured document first proposed by (TODO: CITE). Using this output we are not only able to retrieve the processed words, but also their position in the document.

The package hocr contains necessary java classes to represent this document in an objective-oriented way. The string output of the TesseractWrapper class can be given to the constructor of the HocrDocument class, that completely parses the string and divides it into multiple HocrAreas, HocrParagraphs, HocrLines and HocrWords. Before the actual step of processing the image, we want to improve its quality. Therefore, we use the ImagePreprocesser class. Any kind of document inserted will first converted to a BufferedImage. Then preprocess() can be called which executes multiple algorithms on the image:

```
public BufferedImage preprocess() {
    try {
        ...
        BufferedImage outputFile = this.resizeImage(image);
    ...
        outputFile = this.adjustDPI(image);
    ...
        outputFile = this.deSkewImage(image);
    ...
        outputFile = this.greyScaleImage(image);
    ...
        outputFile = this.deSpeckleImage(image);
    ...
        return outputFile;
        }
    ...
}
```

Most of those calculations are made using ImageMagick, a powerful open source library with several useful commands to apply on images. It is licensed under the Apache 2.0 license. In order to use it inside our application we are using IM4Java which is cited by ImageMagick itself (here: https://www.imagemagick.org/script/develop.php) and is licensed under the LGPL license.

In order to increase the performance of the application, we want to be able to perform the optical character recognition by using multiple instances of the tesseract at the same time. Hence we need to implement the Runnable interface provided by the JDK.

Seen from the outside, the TesseractWrapper class is just the Tesseract instance itself. So we need a worker class that can be given to a new Thread. The TesseractWorker class implements this interface. When we start a new Thread using start(), the run()-method of this worker is called internally. Run initiates a new Tesseract instance and executes OCR with the given ocr file:

```java
/**
 * Executes tesseract ocr using a wrapper
 * The result can be obtained using the getResultIfFinished() method
 */
@Override
public void run() {
    TesseractWrapper wrapper = new TesseractWrapper();
    if (this.imgToScan == null) {
        this.result = wrapper.initOcr(this.fileToScan, runWithHocr);
    } else {
        this.result = wrapper.initOcr(this.imgToScan, runWithHocr);
    }
    Logger.getLogger(this.getClass()).log(Level.INFO, ”Finished OCR”);
}
```

Since we want to be able to support not only pdf documents, but also images, we have to differentiate between this two. Depending what type of document, we have to parse it differently in order to get a BufferedImage out of it. Postprocessor:

5.5 Module 2 - Extraction

The core class that extracts the information from the hocr document is the DataExtractorService class. As we also want to retrieve information as fast as possible, we want to run it on different threads, so that we can extract the invoice information part on one thread and the accounting records information on another. Hence this class needs to implement the Runnable interface. When instantiated, a flag is set if this thread should extract the former or the latter:

```java
@Override
public void run() {
    ...
    if (this.extractInvoice) {
        this.threadInvoice = this.extractInvoiceInformationFromHocr();
    } else {
        this.threadRecord = this.extractAccountingRecordInformation();
    }
}
```

We will now start explaining the extractInvoiceInformationFromHocr() method in detail before continuing with the explanation of the extractAccountingRecordInformatio() method. As we built our invoice information extraction process on similar invoices of the same creditor, the extractInvoiceInformationFromHocr() method starts with a search for the creditor:

```java
...
result.setCreditor(this.getLegalPersonFromDatabase(this.getHocrDoument(),
    true));
if (result.getCreditor() != null) {
    result = this.getCaseInformation(result);
} else {
    String invNo = this.findInvoiceNumber();
    result.setInvoiceNumber(invNo);
    result.setIssueDate(this.findIssueDate());
    result.setDebitor(this.getLegalPersonFromDatabase(this.getHocrDoument(),
     false));
}
...
```

If we are not able to find the creditor in the database (because there was no invoice of this creditor yet) we will continue by searching for necessary invoice information by hand. This will be covered after the case information retrieval.

If a creditor is found, we get the case information of the corresponding creditor. A DocumentCase consists of a creditor to which it belongs as well as a keyword which relates the DocumentCase to one of the following:

- Document type: The DocumentCase contains information where to find a keyword that defines the document as an invoice, a proforma invoice or a credit note.
- Invoice number: The DocumentCase contains information where to find the corresponding invoice number of the invoice.
- Invoice date: The DocumentCase contains information where the invoice date is being placed on the document.
- Creditor: The DocumentCase contains information where the name of the creditor usually is. This is being used for new documents that are not classified yet in order to improve the recognition of creditors.
- Debitor: The DocumentCase contains information where the name of the debitor usually is.

Besides the keyword and the creditor, there is also the position stored where one of those keywords can be found, as well as the creation date of the DocumentCase, which is being used so that newer cases get a higher priority. This way we can react on changing designs for example when a company decides to restructure their invoice documents.

In addition to that, a case id clusters all DocumentCases that are created on one document. With five keywords at hand, a maximum of five DocumentCases should be related to one document.

A flag isCorrect is also existing but set to false in the beginning. After the user has reviewed missing information and wants to store the revised documents, the case is compared with the given information. If there are no changes, we expect the case to be correct. Hence at this time we set isCorrect to true. The getCaseInformation() method first retrieves all cases from the found creditor. Then, it sorts them to the corresponding cases.

For each keyword the corresponding cases contain position information of older documents where the keyword has been found. With that position at hand, the current HOCR document is being searched for a value at that position. The method findInCase() deals with this process:

```java
private HocrElement findInCase(List<DocumentCase> cases) {
    for (DocumentCase docCase : cases) {
        if (docCase.getIsCorrect()) {
            String[] position = docCase.getPosition().split(”\\+”);
            // 0: startX, 1: startY, 2: endX, 3: endY
            int[] pos = new int[] {
        Integer.valueOf(position[0]),
        Integer.valueOf(position[1]),
        Integer.valueOf(position[2]),
        Integer.valueOf(position[3])
      };

            HocrElement possibleArea = this.document.getPage(0).
   getByPosition(pos, 50);
            if (possibleArea != null) {
                HocrParagraph possibleParagraph = (HocrParagraph)
   possibleArea.getByPosition(pos, 30);
                if (possibleParagraph != null) {
                    HocrLine possibleLine = (HocrLine) possibleParagraph.
   getByPosition(pos, 30);
                    if (possibleLine != null) {
                        HocrWord possibleWord = (HocrWord) possibleLine.
   getByPosition(pos, 10);
                        if (possibleWord != null) {
                            return possibleWord;
                        } else {
                            // refine to multiple words, pixel threshold
   only a few pixels since we are searching for word
                            possibleWord = possibleLine.getWordsByPosition(
   pos, 10);
                            return possibleWord;
                        }
                    }
                }
            }
        }
    }
    return null;
}
```

We are only using the cases that have the flag isCorrect set to true. Then we compare all HocrElements in the document with the stored position. But, as there could also be some small differences (e.g. because the scans are hand-made and the document has not been placed on the exact same position every time) we apply a threshold value. Every element that is more or less consistent with the given position will be returned. Eventually, we will a word that matches the position, or, if the position stored contained multiple words, a combination of words. Those are concatenated and returned. If any of those steps fail, the method will return null.

This is repeated for each keyword. A new DocumentCase is created and the position added. Every keyword that has not been found will result in missing DocumentCases. After that, the invoice filled with the retrieved information will be returned.

As mentioned before, if we are unable to find a creditor, then we proceed with the document manually. Which means we are looking for keywords such as ”Rechnungsnummer” (invoice no.) or ”Rechnungsdatum” (invoice date) which are usually followed by the corresponding value. This is a fallback practice and will yield more errors due to missing position information. An invoice object with the found values will be returned all the same.

The extractAccountingRecordInformation() method deals with the problem of information retrieval with a different approach: It uses the extracted table information if a table has been found (TODO: Include in text). If not, the HocrDocument is searched for keywords that are usually appear in invoice tables. If we find those information, we iterate over the following lines until we find table end information, such as ”Gesamtbetrag” (total value), ”Lieferdatum” (delivery date) and others. Both, the table header words as well as table end words are stored in two textfiles (tablecontents.txt and tableendings.txt) which allows the user to add more words to improve the accuracy. Now, every line will be processed the following way:

```java
Record r = new Record();
String recordLine = this.removeFinancialInformationFromRecordLine(nextLine);
double value = this.getValueFromLine(nextLine);

Model m = service.getMostLikelyModel(recordLine);
if (m == null) {
    r.setEntryText(nextLine);
} else {
    r.setEntryText(m.getPosition());
    r.setRecordAccounts(m.getAsAccountRecord(value));
    r.setProbability(m.getProbability());
}
records.add(r);
index++;
```

We first want to remove all those additional information from the position so that we are able to store / retrieve it if it comes again more precisely. This is done by the removeFinancialInformationFromRecordLine() method. After that, we also retrieve the total amount of the position by searching in the line again for the financial information, but this time searching for the last numeric value that is proceeded by ”EUR” or ”€”.

After that, the machine learning module is called. What exactly happens there will be covered by the next section. We will retrieve a possible Model that applies to our position. We can assign the found value to every involved account as the Model also contains the percentual values of each account and add a probability value to the Record which will later presented to the user in order to facilitate his decision if the automatically made decision is correct or not.

5.6 Module 3 - Machine Learning

5.7 Module 4 - Transformation

We have now extracted required basic information of the invoice as well as accounting records based on the positions in the invoice. Everything has been labelled by a confidence level in the application. All documents with a confidence level lower than previously defined by the user had to be reviewed by the user manually. The final part of the use case is the transformation of those extracted information into the ZugFerd invoice format. Therefore, we need to order the given information in a predefined format and append it as xml-information to the invoice pdf.

We are using the Mustang project to generate the xml content for us. It is an open source project licensed under the Apache license version 2.0 and currently under version 1.3. This way, the amount of classes we need will be reduced to only one: The ZugFerdTransformator.java class.

Before giving an in-depth explanation about our implementation, we first want to explain how the ZugFerd-Format works.

5.7.1 About the ZugFerd Scheme

ZugFerd has been developed to close the gap between manually sent invoices in small companies and heavy electronic data interchange (EDI) between big companies. While EDI with its substandards can be a good solution for a big company, most of the small and medium sized companies can not make use of such a standard due to the overwhelming complexity that lies beyond this standard. But dealing with pdf documents manually is also a source of costs, errors and is time consuming.

ZugFerd stands in the middle between those two sides (TODO: ADD BILD). While documents can still be sent as a pdf, the underlying format enables automatic processing of the invoice. The extendibility with basic, comfort and extended levels enables also big companies to make use of this standard. This also improves the B2B relations between big and small companies.

Depending on the desired level of the ZugFerd format, more fields have to be filled out. But even the lowest level, the basic level, brings the possibility to provide additional information which would only be required on the comfort or extended level. But there are still some fields that even on the basic level are required. Those will be introduced now and explained shortly.

- Document Context Parameter: This field describes which level will be used in this document. A possible option would be the comfort-level.
- Exchanged Document Identifier: A unique identifier for an invoice. This is usually the invoice number that is present in the invoice document.
- Exchanged Document Type Code: The type code defines the invoice more in detail. There are currently three codes available: 380, 84 and 389.

  In the basic level, only code 380 is supported. All invoices regarding goods or services, as well as credit notes and payment requests should be labelled with this code. Beginning with the Comfort-level, code 84 is also supported. It refers to invoices without goods or values as well as credit notes without goods or values. Only the Extended-level supports code 389, which is a special case for self-filled invoices or credit notes. Exchanged Document Issue Date: The date when the invoice has been issued.
- Trade Agreement Seller Trade Party Name: The name of the company that is selling the goods or services in the invoice (also known as the creditor of the invoice).
- Trade Agreement Buyer Trade Party Name: The name of the company or person that bought

the goods or services and to whom this invoice is addressed at (also known as the debitor of the invoice).

- Supply Chain Trade Settlement Invoice Currency Code: This field describes the kind of currency that is used in the invoice. Countries in the European Union and Germany in particular will mostly be using ”EUR” as the Code for Euro currency, but there are also codes for US Dollar (”USD”), the Britain pound (”GBP”) and the Columbian peso (”COP”) available.
- Trade Settlement Monetary Summation Line Total: Line total is the total value of all positions combined.
- Trade Settlement Monetary Summation Charge Total: This field contains the sum of all additional charges to the invoice. These are not the price of the goods or services, but more additional costs (for instance: delivery costs, cancellation charges or reminder fees).
- Trade Settlement Monetary Summation Allowance Total: The sum of all allowances made on this invoice (e.g.: parts of the goods that are tax-free).
- Trade Settlement Monetary Summation Tax Basis Total: The net total on which the tax will be calculated.
- Trade Settlement Monetary Summation Tax Total: The total tax value that is applied on the invoice.
- Trade Settlement Monetary Summation Grand Total: The total sum of the invoice (usually the net total added by the tax that has been applied).

These are the most important fields in the ZugFerd format. Without them, it is not possible to create a conformal invoice document. This only applies on the Basic level of the ZugFerd format. Using the Comfort or even Extended-Level, several other fields are required. We will not further introduce these additional fields since the support of the other levels is not part of this thesis.

5.7.2 The transformation process

We have now introduced all the necessary fields to create an invoice document which fulfils the requirements of the ZugFerd-Scheme Basic level and can now introduce the ZugFerdTransformator.java class. The core method of this class is the createFullConformalBasicInvoice() method. In the first part, an invoice object is created and meta information are provided:

```
Invoice i = new Invoice(BASIC);

Context con = new Context(BASIC);
Profile guideline = new Profile(BASIC);
guideline.setVersion(ProfileVersion.V1P0);
con.setGuideline(guideline);
```

This information defines the invoice object to be of the Basic level (it has been described before as the Document Context Parameter). The ProfileVersion is currently 1.0 but could be increased when the ZugFerd format is further developed. A header containing the basic information of the invoice is now instantiated:

```
Header h = new Header();
h.setName(”RECHNUNG”);
h.setInvoiceNumber(inv.getInvoiceNumber());
h.setCode(_380);
h.setIssued(new ZfDateDay(inv.getIssueDate().getTime()));
```

As the application only deals with Invoices, we can set the name to ”RECHNUNG” (engl.: invoice). The invoice number has been extracted from the invoice object that has been given to the method. Since this method creates an invoice object of the Basic level, the only applicable code for this level is 380. Afterwards, the issue date of the given invoice object is used as well.

It is now time to add the actual invoice content. First, we have to define the creditor and debitor of this - in the terminology of the ZugFerd documentation - agreement. Both, the creditor and the debitor, are a TradeParty that are added to the agreement:

```
Agreement a = new Agreement();
a.setBuyer(new TradeParty().setName(inv.getDebitor().toString()));
a.setSeller(new TradeParty().setName(inv.getCreditor().toString()));
```

Hence we create a new Agreement object and set Buyer and Seller instances (respectively debitor and creditor) by using the given name of the legal person in the provided invoice object.

All the financial information such as Line Total or Tax Basis Total are now filled in to the MonetarySummation object:

```java
MonetarySummation sum = new MonetarySummation();
sum.setLineTotal(new Amount(BigDecimal.valueOf(inv.getLineTotal()), EUR));
sum.setChargeTotal(new Amount(BigDecimal.valueOf(inv.getChargeTotal()), EUR));
sum.setAllowanceTotal(new Amount(BigDecimal.valueOf(inv.getAllowanceTotal()), EUR));
sum.setTaxBasisTotal(new Amount(BigDecimal.valueOf(inv.getTaxBasisTotal()), EUR));
sum.setTaxTotal(new Amount(BigDecimal.valueOf(inv.getTaxTotal()), EUR));
sum.setGrandTotal(new Amount(BigDecimal.valueOf(inv.getGrandTotal()), EUR));

Settlement s = new Settlement();
s.setCurrency(EUR);
s.setMonetarySummation(sum);
```

The Settlement object holds this information. For each value, we also have to provide currency information. The application currently only supports invoices with the currency Euro, hence every amount will be added as the currency Euro.

To conclude the trade, we also have to define a delivery date. If no such information has been found in the invoice document, we will use the issue date as a fallback value:

```java
Delivery d;
if (inv.getDeliveryDate() == null) {
    d = new Delivery(new ZfDateDay(inv.getIssueDate().getTime()));
} else {
    d = new Delivery(new ZfDateDay(inv.getDeliveryDate().getTime()));
}

Trade tr = new Trade();
Item item = new Item();
tr.addItem(item);
tr.setAgreement(a);
tr.setDelivery(d);
tr.setSettlement(s);
```

After that, a Trade object is being instantiated and the information are added. Note that we create an empty Item object for the trade. This is necessary for the invoice object to be valid. But only in the higher levels actual information regarding specific items are required to be provided.

Eventually, we add the context, the header information as well as the trade object to the actual invoice object:

```
i.setContext(con);
i.setHeader(h);
i.setTrade(tr);
```

Before we now return the invoice document, we have to make sure that this document is valid against the ZugFerd-Scheme. Only if this invoice is valid, it will be returned, otherwise the method will return null:

```
if (this.isInvoiceValid(i)) {
    return i;
} else {
    return null;
}
```

The isInvoiceValid() method makes use of an InvoiceValidator, which is given by the Mustang framework and enables us to quickly validate the invoice object:

```
InvoiceValidator invoiceValidator = new InvoiceValidator();

Set<ConstraintViolation<Invoice>> violations = invoiceValidator.validate(i);
return violations.size() < 1;
```

The InvoiceValidator does not only check if the required fields are filled out, but also makes calculations on the MonetarySummation object. For instance, if the provided tax value does not sum up correctly to the grand total or the tax basis is smaller than the actual tax (which would mean a tax value over 100%) an error will be raised. With the correct validation of the invoice object the task of this module is completed.

5.8 Problems during the implementation

Chapter 6

Conclusion and outlook

6.1 Result of the application

6.2 Findings

6.3 Outstanding issues

Even though the application is finalized and working, several improvements could be made in the future. Each of them will be listed here, including the reasons why they should be made as well as possible ideas on how to achieve these improvements.

1. Improving the accuracy of OCR: As the application is highly dependent on the successful and accurate process of OCR, improving the accuracy of the OCR process will improve the usefulness of this application in general. Hence, every action made in this direction is an advantage. There are two ideas that could be realized in the future:

    - Improving the accuracy of the Tesseract by creating an own training set based on a representative amount of invoices (especially in German) of different companies.

    - Exchanging the open source solution for a proprietary solution that provides a higher accuracy and / or is specialized either on invoices or German text.

2. Refactor the overall design of the application: Various adjustments in the application could be made to make the application more extensible in the future. The following is a list of possible changes:

   - Using the strategy pattern on the OCR module: The application should be independent from which kind of OCR API it retrieves the String output. The strategy pattern would ideally lead to the possibility for the user to choose the preferred OCR reader from the settings view.

   - Removing unnecessary or unused Business Objects, such as the Address or CorporateForm classes, since those are not used at the moment. Or instead, extend the application to make use of those classes.

   - TO BE CONTINUED

3. Increase the performance of the processing step: The slowest part of the application is the process of scanning a document and extracting its information. Finding a way to speed-up this step would lead to a faster application. One idea would to parallelize the process of information retrieval with multiple documents and to make use of all processor cores the device the application runs on has.

Appendix A

Index of abbreviations

BO	Business Object
B2B	Business to Business
ERP	Enterprise Resource Planning
EDI	Electronic Data Interchange
EDIFACT	Electronic Data Interchange For Administration, Commerce and Transport
FeRD	Forum f"ur elektronische Rechnung Deutschland
SME	Small and medium enterprises
xCBL	XML Common Business Library
ZugFerd	Zentraler User Guide des Forums f"ur elektronische Rechnung Deutschland

List of Figures

List of Tables

Bibliography

[Bre02] Thomas M. Breuel. *Two Geometric Algorithms for Layout Analysis*, pages 188–199. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[Bre03] Thomas M. Breuel. High performance document layout analysis, 2003.

[Bre07] Thomas M. Breuel. The hocr microformat for ocr workflow and results. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 1063–1067, Sept 2007.

[Bre08] Thomas M. Breuel. The ocropus open source ocr system, 2008.

[Che08] Qiang Chen, Quan-sen Sun, Pheng Ann Heng, and De-shen Xia. A double-threshold image binarization method based on edge detector. *Pattern Recogn.*, 41(4):1254–1267, April 2008.

[Che13] Shuhan Chen, Weiren Shi, and Wenjie Zhang. An efficient universal noise removal algorithm combining spatial gradient and impulse statistic. *Mathematical Problems in Engineering*, 2013:1–12, 2013.

[CO00] Inc. Commerce One. Annual report of 2000, 2000. https://www.media.corporate-ir.net/media_files/NSD/CMRC/reports/10_k.pdf, last visited on 09.11.2016.

[Cov01] Robin Cover. Xml common business library (xcbl), 2001. https://www.xml.coverpages.org/cbl.html, last visited on 09.11.2016.

[Den14] Andreas Dengel and Faisal Shafait. *Analysis of the Logical Layout of Documents*, pages 177–222. Springer London, London, 2014.

[fE16] UN Economic Commission for Europe. Introducing un/edifact, 2016. https://www.unece.org/cefact/edifact/welcome.html, last visited on 08.11.2016.

[Ham07] Hatem Hamza, Yolande Belaïd, and Abdel Belaïd. Case-based reasoning for invoice analysis and recognition. In Rosina O. Weber and Michael M. Richter, editors, *7th International Conference on Case-based Reasoning - ICCBR 2007*, volume 4626, pages 404–418, Belfast, United Kingdom, August 2007. Springer Berlin / Heidelberg. The original publication is available at www.springerlink.com , ISBN 978-3-540-74138-1, ISSN 0302-9743 (Print) 1611-3349 (Online).

[Kau15] Achim Kauffmann. 5 punkte, die sie über zugferd wissen sollten, 2015. https://www.basware.de/blog/2015-07-10/ZUGFeRD-5-punkte-die-sie-wissen-sollten, last visited on 09.11.2016.

[Kle04] Bertin Klein, Stevan Agne, and Andreas Dengel. *Results of a Study on Invoice-Reading Systems in Germany*, pages 451–462. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[Men04] Kurt Menges. Commerce one declares bankruptcy: Does this foretell the fate of b2b e-commerce?, 2004. https://www.supplychainmarket.com/doc/commerce-one-declares-bankruptcy-does-this-fo-0001, last visited on 09.11.2016.

[Nag95] George Nagy. *Document image analysis: What is missing?*, pages 576–587. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

[Ram12] Cartic Ramakrishnan, Abhishek Patnia, Eduard Hovy, and Gully APC Burns. Layout-aware text extraction from full-text pdf of scientific articles. *Source Code for Biology and Medicine*, 7(1):7, 2012.

[Wan15] Chenyang Wang, Yanhong Xie, Kai Wang, and Tao Li. *OCR with Adaptive Dictionary*, pages 611–620. Springer International Publishing, Cham, 2015.

[Zhu05] Li Zhuang and Xiaoyan Zhu. *An OCR Post-processing Approach Based on Multi-knowledge*, pages 346–352. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.