HSB
Hochschule Bremen
City University of Applied Sciences

Hochschule Bremen
Fakultät 4 – Elektrotechnik und Informatik
Internationaler Studiengang Medieninformatik (B.Sc.)

Bachelor-Thesis

Entwicklung eines hOCR-Editors

– zur Bearbeitung von OCR-Ergebnissen –

---

vorgelegt von
Alexander Kopetsch

Erstgutachter:	Prof. Dr. Martin Hering-Bertram
Zweitgutachter:	Dipl.-Inf. Jean-Pierre Schober
Abgabetermin:	20. April 2020 in Bremen

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen wurden, sind durch Angaben der Herkunft kenntlich gemacht.

Diese Erklärung erstreckt sich auch auf in der Arbeit enthaltene Grafiken, Skizzen, bildliche Darstellungen sowie auf Quellen aus dem Internet. Die Arbeit habe ich in gleicher oder ähnlicher Form auch auszugsweise noch nicht als Bestandteil einer Prüfungs- oder Studienleistung vorgelegt.

Ich versichere, dass die eingereichte elektronische Version der Arbeit vollständig mit der Druckversion übereinstimmt.

Alexander Kopetsch

Matrikelnummer 5002589

Bremen, den 20. April 2020

_____

Inhaltsverzeichnis

1 Einleitung

Mit der vorliegenden Thesis zur Erlangung des wissenschaftlichen Grades Bachelor of Science soll die Entwicklung eines hOCR-Editors zur Bearbeitung von OCR-Ergebnissen dokumentiert, konzeptioniert und prototypisch realisiert werden.

1.1 Problemfeld

Die Texterkennung oder OCR (engl. *optical character recognition*) ist ein automatisiertes Verfahren, bei dem aus Bilddaten Textdaten gewonnen werden. Beispielsweise können Unternehmen mithilfe der Texterkennung Rechnungsnummern erkennen, die Post kann automatisch Adressen auf Briefumschlägen lesen und Bibliotheken können altes Schriftgut digitalisieren. Man erspart eine händische Transkription.

Wenngleich die Texterkennung gute Resultate liefert, kann nicht immer von einer perfekten Genauigkeit ausgegangen werden kann. Eine absolute Genauigkeit kann nur durch Korrekturlesen der Ergebnisse von Menschen sichergestellt werden. Die Anforderungen an die OCR und ihre Genauigkeit können sich je nach Anwendungsfall stark unterscheiden. Während eine vollständige Texterkennung in einigen Fällen erforderlich sein kann, mag eine teilweise Texterkennung in anderen Zusammenhängen schon ausreichend sein.

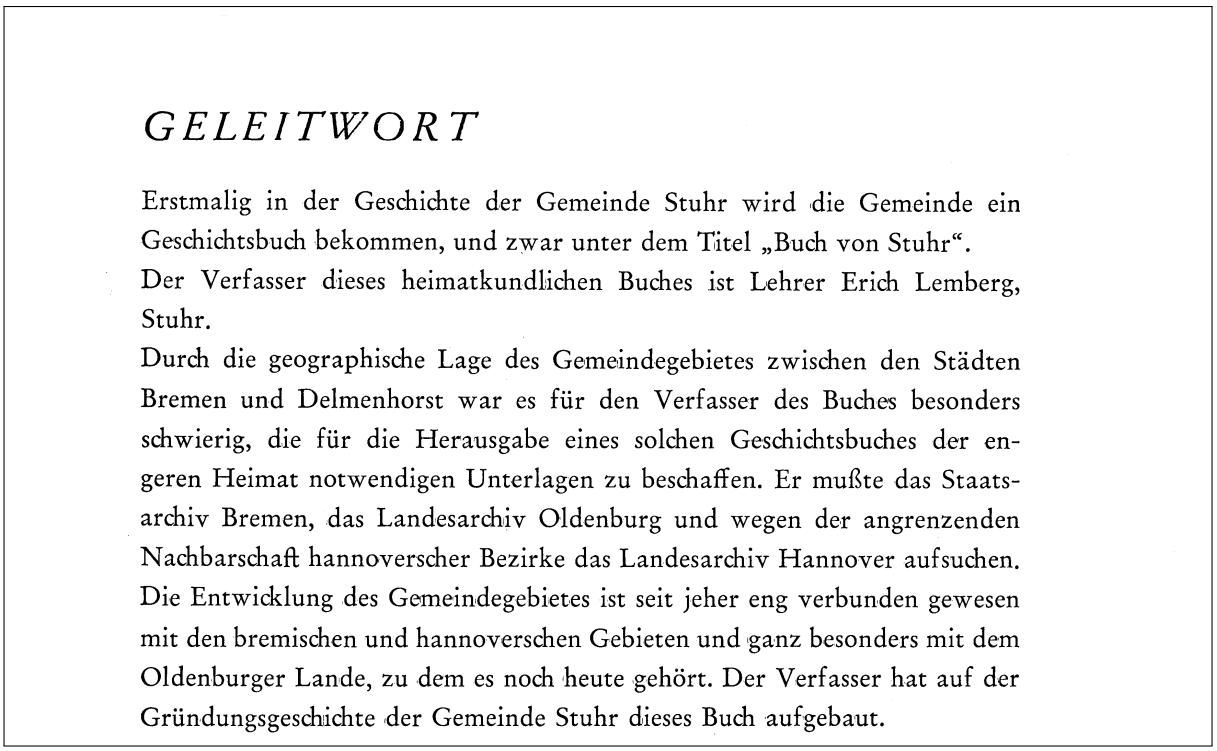
Wenn es z. B. darum geht, eingehende Rechnungen in Papierform maschinell zu identifizieren, müssen bei weitem nicht alle auf dem Papier enthaltenen Informationen mittels OCR gewonnen werden. Es kann genügen, damit die Rechnungsnummer zu ermitteln und zur Fehlerkorrektur eventuell noch einen Barcode oder den Briefkopf heranzuziehen. Metadaten aus den OCR-Ergebnissen wie die Position von erkannten Buchstaben oder erkannte Schriftarten sind eher uninteressant.

Geht es aber z. B. darum, die Retrodigitalisierung (Digitalisierung bisher analoger Schriftstücke) durchzuführen, ist Originalgetreue und damit die absolute Genauigkeit der OCR ein erstrebenswertes Ziel. Da dies derzeit nicht vollumfänglich gegeben werden kann, ist es sinnvoll, das Korrekturlesen (engl. *proofreading*) oder das Bearbeiten der OCR-Ergebnisse durch einen Menschen zu ermöglichen.

Als Beispiel dient die Retrodigitalisierung der Ortschronik „Das Buch von Stuhr“[1] aus dem Jahre 1966 von Erich Lemberg: Das Buch ist nur noch antiquarisch zu erwerben und wurde bisher noch nicht digitalisiert. Im privaten Rahmen dieser Arbeit wurden alle 292

[1] Lemberg, Erich (1966) *Das Buch von Stuhr*. Delmenhorst: Siegfried Rieck, Druckerei und Verlag

Buchseiten gescannt und für die Texterkennung vorbereitet. Das Buch wird in folgenden Teilen kurz als „Stuhr-Buch“ bezeichnet. Die folgende Abbildung zeigt auszugsweise die Überschrift und die ersten beiden Absätze der neunten Seite:



GELEITWORT

Erstmalig in der Geschichte der Gemeinde Stuhr wird die Gemeinde ein Geschichtsbuch bekommen, und zwar unter dem Titel „Buch von Stuhr“.
Der Verfasser dieses heimatkundlichen Buches ist Lehrer Erich Lemberg, Stuhr.
Durch die geographische Lage des Gemeindegebietes zwischen den Städten Bremen und Delmenhorst war es für den Verfasser des Buches besonders schwierig, die für die Herausgabe eines solchen Geschichtsbuches der engeren Heimat notwendigen Unterlagen zu beschaffen. Er mußte das Staatsarchiv Bremen, das Landesarchiv Oldenburg und wegen der angrenzenden Nachbarschaft hannoverscher Bezirke das Landesarchiv Hannover aufsuchen.
Die Entwicklung des Gemeindegebietes ist seit jeher eng verbunden gewesen mit den bremischen und hannoverschen Gebieten und ganz besonders mit dem Oldenburger Lande, zu dem es noch heute gehört. Der Verfasser hat auf der Gründungsgeschichte der Gemeinde Stuhr dieses Buch aufgebaut.

Abbildung 1.1: Das Buch von Stuhr – Anwendungsfall

Obwohl der Text für einen Menschen deutlich lesbar erscheint, werden per OCR bestimmte Wörter falsch erkannt. Es werden zwar viele Wörter richtig erkannt, jedoch wird bei sehr vielen Wörtern nur eine geringe Erkennungssicherheit zurückgegeben.

1.2 Ziele der Arbeit

Das Ziel dieser Arbeit ist die Entwicklung eines Editors, mit dem OCR-Ergebnisse manuell bearbeitet und korrigiert werden können. Das native Projektformat der Anwendung soll hOCR sein, welches als offenes Mikroformat erlaubt, OCR-Ergebnisse mit Meta-Informationen wie z. B. die Positionen von erkannten Wörtern in ein HTML-Dokument einzubetten. hOCR wurde im Jahre 2007 eingeführt und wird von einigen, v.a. quelloffenen OCR-Engines als Ausgabeformat unterstützt. Es sind bisher einige Tools für hOCR erschienen, ein ausgereifter Editor (nicht nur Betrachter) leider jedoch nie. Es gibt nur einige veraltete, experimentelle, webbasierte Anwendungen, die höchstens das Betrachten im Browser erlauben aber eben nicht das Bearbeiten. Diese Arbeit soll versuchen, diese Lücke, v.a. im Bereich der Desktop-Anwendungen, zu füllen.

1.3 Lösungsansatz

Die **Programmiersprache C#**[2], die Software-Plattform **.NET**[3] und die Entwicklungsumgebung **Visual Studio**[4] können als Mittel der Wahl für Entwicklung einer modernen desktopbasierten Editor-Anwendung für das Betriebssystem Windows 10 von Microsoft betrachtet werden.

Das **HtmlAgilityPack**[5] ist eine BSD-lizenzierte Bibliothek, die als Grundlage für das Software-Backend bzw. die hOCR-Bibliothek dienen könnte, welche die Serialisierung und Deserialisierung von HTML-Daten gemäß DOM (Document Object Model) erlaubt.

Die OCR-Engines **Tesseract**[6] und **Cuneiform** bieten hOCR als Datenformat zur Ausgabe der OCR-Ergebnisse. Tesseract kann mithilfe der gleichnamigen Bibliothek für .NET von Charles Weld eingebunden werden. Cuneiform könnte durch einen Wrapper eingebunden werden. Dabei kann unter Windows sogar die Linux-Variante (Cuneiform for Linux)[7] über das WSL (Windows Subsystem for Linux) eingebunden werden.

Die Bibliothek **YamlDotNet**[8] kann verwendet werden, um das YAML-Dokument, welches Definitionen der Begriffe aus dem hOCR-Standard enthält, zu parsen. Ein Softwaretest kann dann ggf. unter Einsatz von Reflection prüfen, ob die hOCR-Bibliothek alle Definitionen abdeckt.

Der Datensatz **Google 1000 Books**[9] erschien im Rahmen der ICDAR 2007 und umfasst tausend gemeinfreie Bücher als Digitalisate, welche jeweils Scans der Seiten, ein hOCR-Dokument und Metadaten beinhalten. [1] Dieser Datensatz sollte sich sehr gut eignen, um die hOCR-Bibliothek mit Testdaten zu versorgen. Wenn die „Reserialisierung“ – Deserialisierung mit anschließender Serialisierung – keine Unterschiede in den hOCR-Dateien verursacht und noch dazu keine Fehler bzw. Exceptions auftreten, kann davon ausgegangen werden, dass die Software fehlerfrei funktioniert.

Mit **WPF** (Windows Presentation Foundation)[10] kann das Software-Frontend grafisch realisiert werden. Das GUI-Framework erlaubt eine Trennung von GUI- und Anwendungslogik und bietet dazu viele allgemeine und spezielle Features zur Visualisierung. Es ist z. B. möglich, sog. grafische Primitive zu zeichnen, was genutzt werden könnte, um beispielsweise Rechtecke für erkannte Objekte darzustellen. Die Auszeichnungssprache XAML baut auf XML auf und dient dabei der GUI-Strukturierung.

[2] http://csharp.net
[3] https://dotnet.microsoft.com
[4] https://visualstudio.microsoft.com
[5] https://html-agility-pack.net
[6] https://github.com/tesseract-ocr/tesseract
[7] https://launchpad.net/cuneiform-linux
[8] https://github.com/aaubry/YamlDotNet
[9] http://commondatastorage.googleapis.com/books/icdar2007/README.txt
[10] https://docs.microsoft.com/en-us/dotnet/framework/wpf/

Bei der Entwicklung werden voraussichtlich zwei Aufgaben anfallen:

1. **Backend**: Es muss eine *hOCR-Bibliothek* entwickelt werden, die die Deserialisierung, Manipulation und Serialisierung von hOCR-Dateien erlaubt.
2. **Frontend**: Es muss der eigentliche *hOCR-Editor* entwickelt werden, also eine grafische, benutzerinteraktive Anwendung. Die Herausforderung wird vor allem darin bestehen, mittels mehrerer GUI-Elemente („Controls“) hOCR-Daten angemessen zu repräsentieren und eine Bearbeitung zu erlauben.

Die Begrenzungsrechtecke (engl. *bounding boxes*) erkannter Bereiche sollen grafisch dargestellt werden. Das Vertrauensniveau der Ergebnisse soll durch Einfärbung der Rechtecke veranschaulicht werden. Die folgende Abbildung zeigt, wie eine die Visualisierung einer hOCR-Datei aussehen könnte:

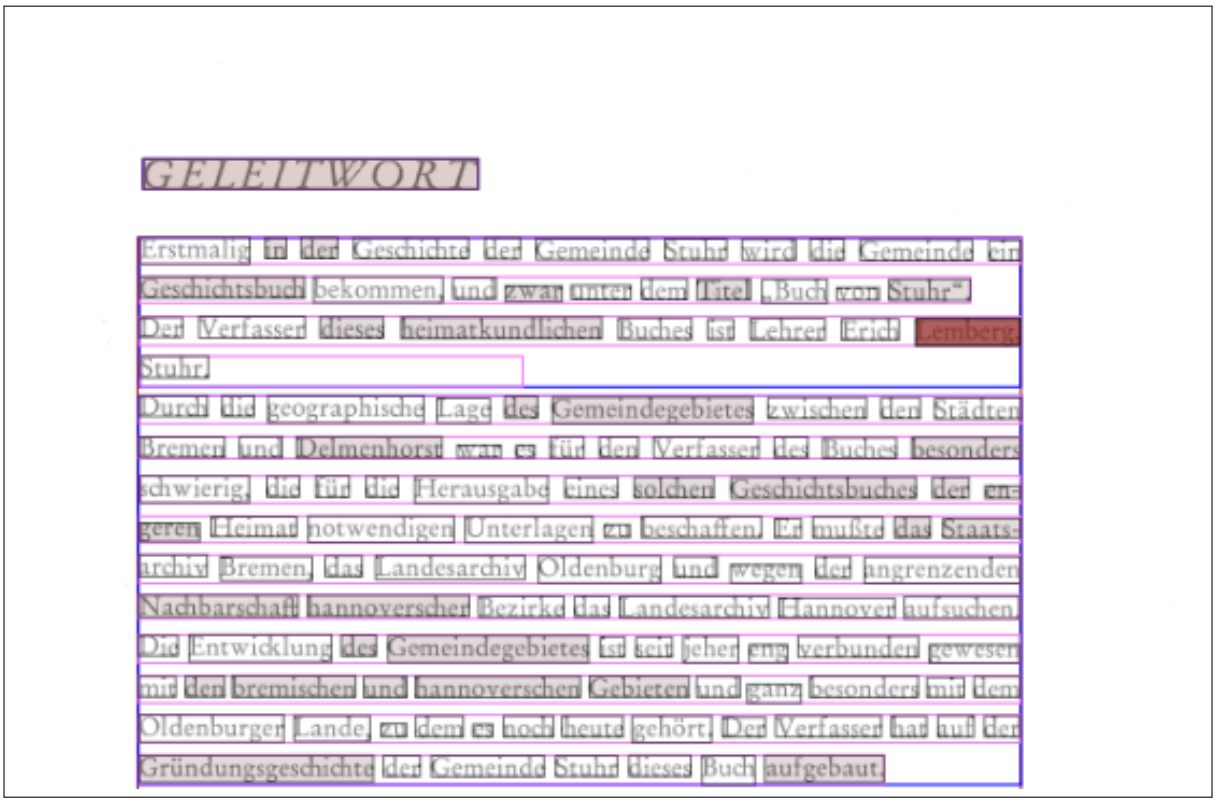


Abbildung 1.2: Das Buch von Stuhr – Proof of Concept

Der Screenshot wurde mit dem HtmlAgilityPack und WPF experimentell angefertigt. Dazu wurde ein Anzeigeprogramm entwickelt, das den hOCR-Standard nicht weiter implementiert. Wie bereits beschrieben, muss der vollwertige Editor weitaus mehr Anforderungen genügen, auf die im folgenden Kapitel eingegangen wird.

1.4 Aufbau der Arbeit

Der wesentliche Aufbau dieser Arbeit ergibt sich aus der Zweiteilung in Analyse und Synthese und umfasst die sieben Kapitel, die im Folgenden beschrieben werden:

1. **Einleitung** – Der Leser wird von der allgemeinen Thematik über das behandelte Problemfeld bis hin zu knappen Lösungsansätzen in die Arbeit eingeführt.
2. **Anforderungen** – Hiermit beginnt der Analyseteil der Arbeit. Es wird festgelegt, welchen Anforderungen die zu entwickelnde Anwendung genügen soll.
3. **Grundlagen** – Die Grundlage dieser Arbeit ist vorwiegend der Standard hOCR, der neben OCR-Engines und Tools, die das Format unterstützen, umfassend beschrieben wird, woraufhin alternative OCR-Formate knapp erläutert werden. Vergleichbare Editor-Anwendungen werden vorgestellt, um einen Überblick über den Software-Markt zu geben. Nach einer kurzen Beschreibung des wissenschaftlichen Umfeldes wird ein Ausblick geschaffen, der abschließend zum Syntheseteil dieser Arbeit überleitet.
4. **Konzeption** – Der Ansatz des Top-Down-Designs wird bei der Konzeption der Software-Anwendung verfolgt: Die Architektur wird zunächst abstrakt anhand des Schichten-Modells festgelegt. Es werden Empfehlungen für die Auswahl von Programmiersprache, Plattformen, Laufzeitumgebungen, Bibliotheken und Algorithmen beschrieben. Anhand eines Wireframes wird die grafische Oberfläche entworfen und veranschaulicht. Statt konkreter Code-Listings sind Teile eher noch in Pseudocode enthalten.
5. **Realisierung** – Die Software-Anwendung wird prototypisch realisiert. Prototypisch, da der Fokus mehr auf präziser Dokumentation und Reproduzierbarkeit liegt und weniger auf Vollständigkeit bzw. darauf, dass die Anwendung ausgereift ist. Details zur Implementation werden näher beschrieben. Anders als im vorigen Kapitel werden dazu vermehrt konkrete Code-Listings eingesetzt. Screenshots der Anwendung werden gezeigt.
6. **Evaluation** – Im Hinblick auf die zuvor festgelegten Anforderungen wird das Entwicklungsergebnis ausgewertet. Dabei wird aufgezeigt, inwiefern die Ziele der Arbeit erreicht werden konnten. Einschränkungen und auftretende Probleme werden beschrieben.
7. **Fazit** – Die gesamte Entwicklung der Arbeit und insbesondere dessen Ergebnis werden zusammengefasst. Für die Limitationen, die das vorangehenden Kapitel erklärt, werden im Ausblick alternative Ansätze aufgeführt.

Die Kapitel 2 u. 3 (Anforderungen und Grundlagen) bilden zusammen den Analyseteil der Arbeit. Der Syntheseteil besteht aus den Kapiteln 4 und 5 (Konzeption u. Realisierung). Der Vollständigkeit und Genauigkeit halber wird auf diese Unterteilung bestanden, sodass sich dem Leser der „rote Faden“ erst im Verlaufe erschließen mag.

2 Anforderungen

Die zu entwickelnde Anwendung soll bestimmten Anforderungen genügen, damit auf das beschriebene Problemfeld durch eine prototypische Umsetzung des Lösungsansatzes eingegangen werden kann. Die Anforderungen werden im Folgenden beschrieben.

2.1 Anwendungstyp und Plattform

Das Software-System soll eine desktopbasierte Editor-Anwendung für das Betriebssystem Windows 10 von Microsoft darstellen. Diese Anforderung wird sich v.a. auf die Wahl der Programmiersprache und des Entwicklungsumfeldes auswirken.

2.2 Grafische Benutzeroberfläche

Es sollen die sog. Human Interface Guidelines, also Richtlinien für die Entwicklung der Benutzerschnittstelle befolgt werden. Diese ergeben sich aus Anwendungstyp und Plattform ergeben und erfordern z. B. eine Menü-Leiste (`File` , `Edit` , `View` und `Help`) und ein „About“-Fenster. Die Anwendung soll in das Betriebssystem integrierbar sein und wie eine „klassische“ Windows-Anwendung erscheinen: Bestenfalls soll die Anwendung als Standard-Anwendung zum Öffnen von Dateien mit der Endung `.hocr` dienen können und den Konventionen für Windows-Anwendungen folgen.

2.3 Natives Dateiformat hOCR

Die Anwendung soll hOCR als *natives* Dateiformat nutzen, womit Anforderungen an die Benutzererfahrung (engl. *user experience*, kurz UX) gestellt werden. Editor-Anwendungen erlauben das Öffnen (`Open`) und Speichern (`Save`) von Dateien. Das Dateiformat, das für diese Benutzeraktionen vorgesehen ist, wird als natives, anwendungseigenes Dateiformat bezeichnet. Dateiformate, die nicht nativ bzw. anwendungsfremd sind, werden ggf. durch Aktionen wie `Import` und `Export` , manchmal auch durch „Speichern Unter“ (`Save As`), unterstützt.

2.4 Integrität der Daten

Das *native* Dateiformat hOCR soll über die Anforderungen an die Benutzererfahrung in vorigem Abschnitt hinaus auch *Integrität* bei der Verarbeitung von hOCR-Dateien erfordern: Wenn mit dem Editor eine hOCR-Datei geöffnet wird, anschließend effektiv vom Benutzer *nicht* bearbeitet wird und schließlich wieder gespeichert wird, soll sie sich nicht verändert haben. Wird sie hingegen geöffnet und anschließend nur an einer bestimmten Stelle vom Benutzer im Editor manipuliert, bspw. indem die Koordinaten einer Zeile verschoben werden, dann soll die Datei nach dem Abspeichern auch nur an genau derselben Stelle anders sein. Das erlaubt dann eine einfache Versionskontrolle, da hOCR (HTML) textbasiert ist und in den hOCR-Dateien relativ simpel Änderungen durch sog. Diffing (zeilenbasiertes Vergleichen) festgestellt werden können.

2.5 Unterstützung für hOCR

Der Standard hOCR soll voll unterstützt werden: Die verschieden „Dialekte“, die von hOCR-Engines produziert werden und sich Erweiterungen des sehr flexiblen Formats zunutze machen, sollen ohne Programmabsturz und Fehlermeldung verarbeitet werden können. Ebenso fehlerfrei sollen aber auch solche Daten behandelt werden, die gemäß Spezifikation des Standards eigentlich nicht valide sind, aber dennoch durchaus aufzufinden sind.

Es soll immer möglich sein, alle hOCR-Daten in einem Dokument bearbeiten zu können. Die Editor-Anwendung muss dem Benutzer also alle Daten präsentieren und eine Manipulation erlauben. Neben den hOCR-Elementen und hOCR-Properties muss es im Sinne des Korrekturlesens möglich sein, den Text bearbeiten zu können. Die Möglichkeit der Bearbeitung der Metadaten eines hOCR-Dokuments ist wünschenswert aber nicht zwingend notwendig.

Eine anschauliche Darstellung der Daten, etwa des Vertrauensniveaus der Texterkennung durch Einfärbung der erkannten Wörter, ist wünschenswert aber nicht zwingend notwendig. Es wird ohnehin nicht möglich sein, alle Daten darzustellen, da allein die zugrundeliegende Spezifikation des Standards sehr flexibel ist, teilweise Definitionslücken aufweist und Spielraum für verschiedene Interpretationen bietet.

2.6 Testabdeckung

Die Testabdeckung (engl. *test coverage*) des Software-Systems soll mithilfe von Modultests (engl. *unit tests*) gewährleistet werden. Der Fokus soll dabei v.a. auf dem Software-Backend liegen, welches die hOCR-Dokumente verarbeitet und unabhängig von Anzeigegrafik und Benutzerinteraktionen funktioniert. Die Testabdeckung des Software-Frontends rückt hingegen eher in den Hintergrund.

3 Grundlagen

Die Grundlage dieser Arbeit ist vorwiegend der Standard hOCR, der neben OCR-Engines und Tools, die das Format unterstützen, in folgenden Abschnitten umfassend beschrieben wird, woraufhin alternative OCR-Formate knapp erläutert werden. Vergleichbare Editor-Anwendungen werden vorgestellt, um einen Überblick über den Software-Markt zu geben. Nach einer kurzen Beschreibung des wissenschaftlichen Umfeldes wird ein Ausblick geschaffen, der abschließend zum Syntheseteil dieser Arbeit überleitet.

3.1 Standard hOCR

Der Standard **hOCR** („HTML-OCR“)[1] beschreibt ein offenes Datenformat, welches es ermöglicht, OCR-Ergebnisse in ein HTML-Dokument einzubetten. [2], [3, § 3.5]

3.1.1 Mikroformat hOCR

Ein **Mikroformat** (engl. *microformat*, kurz µF) definiert, wie ein HTML-Dokument mit bestimmten semantischen Informationen angereichert werden kann, die von Programmen verarbeitet werden können, die das Mikroformat implementieren. Die Mikroformate hCalendar oder hCard erlauben bspw. das Einbetten von Kalenderdaten bzw. Kontaktdaten. Dabei werden HTML-Features nicht zweckentfremdet oder neu definiert, sondern sinngemäß wiederverwendet. HTML-Dokumente, welche Daten durch Mikroformate enthalten, sind gemäß HTML-Standard immer noch gültig. [4], [5]

Durch hOCR wird der Ansatz eines Mikroformats verfolgt, also ein bestehendes HTML-Dokument semantisch um Informationen zur Texterkennung erweitert. Damit sind hOCR-Dokumente auch HTML-Dokumente. [2]

3.1.2 Standard HTML

Die Auszeichnungssprache **HTML** (Hypertext Markup Language)[2] wird von der Arbeitsgruppe WHATWG (Web Hypertext Application Technology Working Group) entwickelt. Die aktuellste Version HTML5 erschien im Oktober 2014. Im Jahre 2007 und in

[1] http://kba.cloud/hocr-spec/1.2/
[2] https://html.spec.whatwg.org

den Folgejahren, als hOCR und Software dazu hauptsächlich entwickelt wurde, waren HTML 4.01 und XHTML 1.0 und 1.1 die gängigen Varianten, sowohl im WWW (World Wide Web) als auch in hOCR-Dokumenten. hOCR wurde bisher noch nicht angepasst, um sich die Weiterentwicklungen durch HTML5 zunutze zu machen. [6]

Grundlegend besteht jedes HTML-Dokument (und damit jedes hOCR-Dokument) aus einem Wurzelelement `<html>`, welches den Kopf (engl. *head*) `<head>` und den HTML-Körper `<body>` beinhaltet. Dem Wurzelelement vorangestellt ist die DTD (Document Type Definition) (dt. *Dokumenttypdefinition*), welche die genaue HTML-Version bzw. HTML-Variante angibt.

Um dem in einem HTML-Dokument enthaltenen Text (Hypertext) eine logische oder semantische Struktur zu verleihen, wird er durch bestimmte HTML-Elemente ausgezeichnet. Auszeichnung (engl. *markup*) von Text durch ein HTML-Element bedeutet, dass sich der Text zwischen öffnenden und schließendem Text des Elements befindet. Das folgende Beispiel zeigt, wie die Überschrift und der Anfang des ersten Absatzes von Seite 9 aus dem Stuhr-Buch (s. Abschnitt 1.1) mittels HTML-Tags ausgezeichnet werden könnten:

```
<!DOCTYPE html>
<html>
  <head>...</head>
  <body>
    ...
    <h1>GELEITWORT</h1>
    <p>
      Erstmalig in der Geschichte der Gemeinde Stuhr wird die Gemeinde ein
      Geschichtsbuch bekommen, und zwar unter dem Titel...
    </p>
    ...
  </body>
</html>
```

Listing 3.1: HTML: Text-Auszeichnung

Es wird auf sog. Tags zurückgegriffen, um mithilfe von HTML-Elementen im HTML-Körper die logische und semantische Struktur des Dokuments wiederzugeben. [7, § 2.1]

- Überschriften werden mit `<h1>` bis `<h6>` (engl. *heading*) kodiert.
- Textblöcke werden allgemein mit dem Tag `<div>` (engl. *division*) beschrieben.
- Absätze (Paragraphen) können mit `<p>` (engl. *paragraph*) dargestellt werden.
- Bestimmte Phrasen können vom `<span>` (engl. *span*) umspannt werden.

Weitere Informationen, wie z. B. Position von Wörtern, Seitenabmessungen und Verweise auf die gescannten Rastergrafiken, werden mithilfe von hOCR-Elementen und deren hOCR-Properties kodiert, welche im Folgenden beschrieben werden. [7, § 2 u. 4]

3.1.3 Elemente

Die **hOCR-Elemente** bilden eine Untermenge der HTML-Elemente: Ein solches Element ist ein hOCR-Element, wenn es zu *einer* Klasse gehört, deren Name mit `ocr_` oder `ocrx_` beginnt.[3] Dieser Name benennt dann das hOCR-Element. In dieser Arbeit werden hOCR-Elemente kurz als *Elemente* bezeichnet. [7, § 2 u. 3]

Das Element **ocr_page** beschreibt eine Seite (engl. *page*) mit OCR-Ergebnissen und ist in jedem hOCR-Dokument mindestens einmal enthalten. Im folgenden Listing werden die Seiten 9 und 10 des Stuhr-Buchs beschrieben:

```
<div class='ocr_page' id='page_9'  title='image "009.png"; bbox 0 0 3472 4937; ppageno 9' >...</div>
<div class='ocr_page' id='page_10' title='image "010.png"; bbox 0 0 3472 4937; ppageno 10'>...</div>
```

Listing 3.2: hOCR: Beispiel auf Dokumentenebene

Das Element **ocr_carea** beschreibt innerhalb einer Seite einen Inhaltsbereich (engl. *content area*) wie etwa einen Textblock oder eine Textspalte. Erkannter Text darf nur darin vorkommen. Das folgende Listing zeigt die vier Inhaltsbereiche, die für Seite 9 des Stuhr-Buchs erkannt wurden:

```
1 <div class='ocr_page' id='page_9' title='...'>
2   <div class='ocr_carea' id='block_1_1' title="bbox  388 1040 1360 1127">...</div>
3   <div class='ocr_carea' id='block_1_2' title="bbox  371 1268 2937 4227">...</div>
4   <div class='ocr_carea' id='block_1_3' title="bbox  370 4264 2929 4339">...</div>
5   <div class='ocr_carea' id='block_1_4' title="bbox 2896 4465 2929 4519">...</div>
6 </div>
```

Listing 3.3: hOCR: Beispiel auf Seitenebene

Durch **ocr_par**, **ocr_line** und **ocrx_word** wird ein Inhaltsbereich weiter in Absätze (engl. *paragraphs*) unterteilt, die wiederum aus mehreren Zeilen (engl. *lines*) bestehen, welche schließlich mehrere Wörter (engl. *words*) umfassen. Das folgende Listing zeigt von Dokumenten- bis auf Wortebene, wie die ersten beiden Wörter im ersten Absatz auf der Seite 9 des Stuhr-Buchs beschrieben werden:

[3] Die Klassennamen eines HTML-Elements sind im Wert seines `class`-Attributs enthalten.

```
<div class='ocr_page' id='page_9' title='...'>
  <div class='ocr_carea' id='block_1_1' title="...">...</div>
  <div class='ocr_carea' id='block_1_2'
       title="bbox 388 1040 1360 1127">
    <p class='ocr_par' id='par_1_2' lang='deu'
       title="bbox 388 1040 1360 1127">
      <span class='ocr_line' id='line_1_2'
            title="bbox 379 1268 2936 1344;
                   baseline 0.003 -18;
                   x_size 77;
                   x_descenders 17;
                   x_ascenders 25">
        <span class='ocrx_word' id='word_1_2'
              title='bbox 379 1268 697 1344; x_wconf 96'>Erstmalig</span>
        <span class='ocrx_word' id='word_1_3'
              title='bbox 742 1269 803 1328; x_wconf 95'>in</span>
        ...
      </span>
      ...
      </p>
  </div>
  <div class='ocr_carea' id='block_1_3' title="...">...</div>
  <div class='ocr_carea' id='block_1_4' title="...">...</div>
</div>
```

Listing 3.4: hOCR: Beispiel bis auf Wortebene

3.1.4 Properties

Die **Properties** eines Elements werden als Zeichenfolge im Wert des HTML-Attributs `title` kodiert. Jede Property trägt einen Namen und hat einen Wert. wobei das Erweiterungspräfix `x_` frei verwendet werden darf, die nicht bereits vom Standard spezifiziert werden. Alle 37 vom Standard vordefinierten Properties werden im Anhang aufgeführt. Auf drei besonders wichtige Properties wird im Folgenden eingegangen:

3.1.4.1 bbox

Die Property **bbox** beschreibt ein achsenparalleles, minimal umgebendes Rechteck (engl. *minimum bounding rectangle*, auch *bounding box*), welches die Position und das Ausmaß eines bestimmten Seitenbereichs definiert. Das Rechteck wird durch die vier nichtnegativen Werte x0, y0, x1 und y1 beschrieben, welche seine linke obere Ecke (x0, y0) und seine rechte untere Ecke (x1, y1) als Koordinaten definieren, wobei der Koordinatenursprung die linke obere Ecke der Seite ist. Der Begriff „Bounding Box“ wird in dieser Arbeit für die Property bbox verwendet. [7, § 4.2]

Die Bounding Box wird im Folgenden durch den String `bbox 10 20 160 30` dargestellt:

```
<span class='ocr_line' id='line_1' title="bbox 10 20 160 30">...</span>
```

Die folgende Grafik wurde als gemeinfreies Werk der hOCR-Website entnommen und veranschaulicht die Koordinaten (x0, y0) und (x1, y1) bzw. (10, 20) und (160, 30):

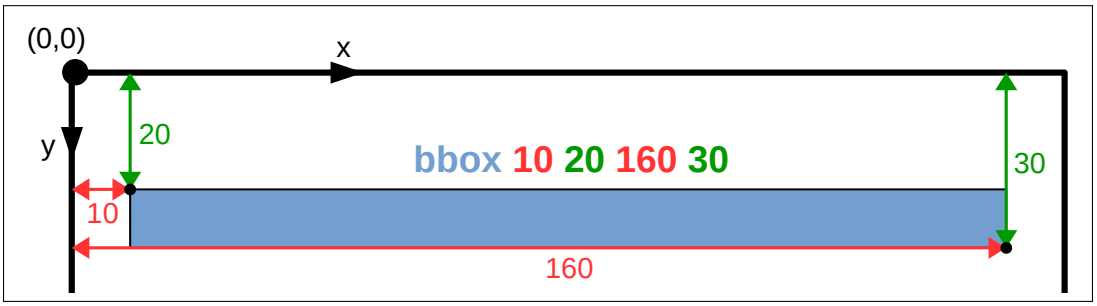


Abbildung 3.1: hOCR-Standard – Bounding Box

3.1.4.2 image

Die Property **image** verweist durch einen unixartigen Pfad auf die Bilddatei, auf welche sich die OCR-Ergebnisse der Seite beziehen. Der Pfad kann relativ sein und muss nicht auflösbar sein. Wenn sich die hOCR-Datei und die Bilddatei im selben Verzeichnis oder Dateiarchiv befinden, sollte die Pfadauflösung jedoch möglich sein. [7, § 4.6]

3.1.4.3 x_wconf

Die Property **x_wconf** (engl. *word confidence*) gibt die Erkennungssicherheit durch einen Gleitkommawert zwischen 0 und 100 wieder, wobei höhere Werte eine höhere Sicherheit bedeuten. [7, § 4.6]

3.1.5 Properties-Syntax

Die **ABNF** (angereicherte Backus-Naur-Form)[4] ist eine Variante der BNF (Backus-Naur-Form) und wird als formale Metasprache für Syntax-Notationen verwendet. [8] Der hOCR-Standard beschreibt mithilfe der ABNF syntaktisch die Darstellung von hOCR-Properties als Zeichenfolgen und verweist dabei speziell auf den Standard RFC 5234, welcher neben anderen RFC-Standards zur Entwicklung beigetragen hat. [7, § 2.4] Die Grammatik wird im Folgenden mittels gekürzter Ausschnitte wiedergegeben. Drei Punkte (`...`) zeigen an, dass aus Platzgründen etwas ausgelassen wurde. Die vollständige Fassung der Grammatik befindet sich im Anhang dieser Arbeit. (s. Abschnitt 8.1.3)

[4] https://tools.ietf.org/html/rfc5234

Mehrere **Properties** werden gemäß der folgenden ABNF-Regel `properties-format` zu einer Zeichenfolge zusammengefasst, indem die Zeichenfolgen der einzelnen Properties mit Semikolon als Trennzeichen aneinandergereiht werden, wobei der Einsatz von Leerraum vor und nach einem Semikolon erlaubt ist. Diese zusammenfassende Zeichenfolge ist der Wert des HTML-Attributs `title`.

```
properties-format = key-value-pair *(*whitespace semicolon *whitespace
                    key-value-pair)
```

Eine **Property** wird als Schlüssel/Wert-Paar (`key-value-pair`) beschrieben, wobei der Name (`property-name`) der Schlüssel ist, welcher durch Leerraum (`whitespace`) vom Wert (`property-value`) getrennt wird:

```
key-value-pair = property-name whitespace property-value
```

Der **Property-Name** (`property-name`) ist entweder explizit vom Standard aufgeführt (`spec-property-name`) oder aber eine Erweiterung für eine bestimmte OCR-Engine (`engine-property-name`), die aus dem Präfix `"x_"` und einem darauffolgenden alphanumerischen Wort (`alnum-word`) besteht:

```
1   spec-property-name = ("bbox" / "baseline" / "cflow" / ... )
2  engine-property-name =  "x_" alnum-word
3         property-name =  spec-property-name / engine-property-name
```

Der **Property-Wert** (`property-value`) ist eine mittels Whitespace trennende Aneinanderreihung von Zeichenfolgen (`ascii-word` oder `delimited-string`), die in dieser Arbeit als Token bezeichnet werden:

```
property-value = (ascii-word / delimited-string) *(whitespace
                 (ascii-word / delimited-string) )
```

Der Token **ascii-word** (dt. ASCII-Wort) ist eine Folge von druckbaren ASCII-Zeichen außer dem Leerzeichen und dem Semikolon:

```
ascii-word = +(%x21-7E - semicolon) ; printable w/o space/semicolon
```

Der Token **delimited-string** (dt. begrenzte Zeichenfolge) ist eine Folge von druckbaren ASCII-Zeichen (`ascii-string`) außer dem doppelten Anführungszeichen, welches die eigentliche Zeichenfolge umschließt und begrenzt.

```
doublequote      =   %x22                    ; double quote '"'
ascii-string     = +(%x20-7E - doublequote) ; printable ascii without doublequote
delimited-string = doublequote ascii-string doublequote
```

Der Standard beschreibt für alle von ihm aufgeführten Properties jeweils den Wert durch eine eigene, spezielle Grammatik, welche die allgemeine Grammatik „überschreibt“. Dabei wird allerdings auf ABNF-Regeln der allgemeinen Syntax zurückgegriffen, welche vom folgenden Listing vollständig wiedergegeben werden:

```
digit            =    %x30-39
uint             =     +digit
int              = *1"-" uint
nint             =   "-" uint
fraction         =   "." uint
float            =      *uint fraction

whitespace       = +%20    ; one or more spaces ' '
comma            =  %2C    ; comma ','
semicolon        =  %3B    ; semicolon ';'
doublequote      =  %22    ; double quote '"'
lowercase-letter =  %x41-5A
alnum-word       = +(lowercase-letter / digit)
ascii-word       = +(%x21-7E - semicolon)   ; printable w/o space/semicolon
ascii-string     = +(%x20-7E - doublequote) ; printable ascii without doublequote
delimited-string = doublequote ascii-string doublequote
```

3.1.6 Metadaten

Die **Metadaten** werden durch Elemente mit dem Tag `<meta>` im HTML-Kopf kodiert. Das Attribut `name` enthält dabei den Namen des Metadatenfeldes und das Attribut `content` dessen Wert. Es werden fünf Metadatenfelder definiert, wobei die folgenden beiden in einem hOCR-Dokument zwingend erforderlich sind. [7, § 6]

- **ocr-system** nennt die OCR-Software, welche das hOCR-Dokument erzeugt hat.
- **ocr-capabilities** nennt die hOCR-Elemente, welche diese Software erzeugen kann.

Die OCR-Engine Tesseract (Version 4.0.0 Beta 1) generiert die folgenden Metadaten:

```
<meta name='ocr-system'       content='tesseract 4.0.0-beta.1' />
<meta name='ocr-capabilities' content='ocr_page ocr_carea ocr_par
                                       ocr_line ocrx_word' />
```

Listing 3.5: hOCR: Metadaten: Minimalbeispiel

Mithilfe von **Dublin Core**, einem bibliographischen Datenformat, können weitere Metadaten eingebunden werden. [7, § 6.1] Das folgende Listing zeigt, wie z. B. die Dublin-Core-Metadatenfelder für die englischsprachige Märchensammlung „Popular Tales of the West Highlands“ von John Francis Campbell aus dem Jahre 1860 aussehen könnten:

```
<meta name='DC.title'     content='Popular Tales of the West Highlands'>
<meta name='DC.creator'   content='John Francis Campbell'>
<meta name='DC.publisher' content='Edmonston and Douglas'>
<meta name='DC.language'  content='EN'>
<meta name='DC.date'      content='1860'>
```

Listing 3.6: hOCR: Metadaten: Dublin Core

3.1.7 YAML-Definition

Die Auszeichnungssprache **YAML**[5] dient der Serialisierung von Daten und zielt insbesondere darauf ab, menschenlesbar zu sein. Der Name ist ein rekursives Akronym für „YAML Ain’t Markup Language“. [9]

Das YAML-Dokument `defs.yml`[6] im GitHub-Repository der hOCR-Spezifikation enthält Definitionen der Elemente, Properties und Metadaten. Daneben enthalten sind ein Makefile und das Python-Skript `gen-defs.py`, welches anhand der Definitionen Teile der Website des hOCR-Standards erzeugt. Das YAML-Dokument enthält auf oberster Ebene drei assotiative Arrays:

- `element` ist eine Auflistung aller Elemente, die ihrerseits weiter spezifiziert werden.
- `property` listet alle Properties, die ihrerseits ebenfalls weiter spezifiziert werden.
- `metadata` enthält die Namen aller fünf Metadatenfelder ohne weitere Details.

Dem Element **ocr_page** wird in der YAML-Spezifikation eine Kategorie zugewiesen und es wird beschrieben, welche Properties für das Element erlaubt, empfohlen oder sogar erforderlich sind:

```
ocr_page:
  categories: ['Typesetting']
  properties:
    required:    ['bbox']
    recommended: ['image', 'imagemd5', 'ppageno', 'lpageno']
    allowed:     ['x_source', 'x_scanner', 'scan_res']
```

Listing 3.7: hOCR: YAML-Definitionen für ocr_page

[5] https://yaml.org
[6] https://github.com/kba/hocr-spec/blob/master/1.2/defs.yml

Der Property **bbox** werden ebenso die Kategorien zugewiesen. Der Eintrag enthält ein Beispiel für die Darstellung der Property in einer Zeichenfolge und definiert dafür die ABNF-Syntax:

```
bbox:
  categories: ['General', 'Layout']
  example: 'bbox 0 0 100 200'
  grammar: |
    <a>property-value</a> = <a>uint</a> <a>uint</a> <a>uint</a> <a>uint</a>
```

Listing 3.8: hOCR: YAML-Definition für bbox

3.1.8 Versionshistorie

Durch **Thomas Breuel** wurde der Standard zusammen mit dem OCR-System OCRopus im Rahmen der ICDAR (International Conference on Document Analysis and Recognition) 2007 eingeführt. [2] Eine wissenschaftliche Veröffentlichung von Breuel zu OCRopus im Folgejahr enthielt eine weitere Beschreibung. [3, § 3.5]

Über **Google Docs** veröffentlichte Breuel im Dezember 2007 die erste vollständige Spezifikation, überarbeitete sie im März 2010 mit Erläuterungen und Fehlerbehebungen und versah sie im Mai 2010 mit den letzten wesentlichen Änderungen. [10]

Die **Version 1.1** entstand, indem Konstantin Baierer im März 2016 auf GitHub eine Portierung der Spezifikation in der Auszeichnungssprache Markdown veröffentlichte. Die Spezifikation auf Google Docs wurde rückwirkend als Version 1.0 bezeichnet und verweist auf der ersten Seite auf die Spezifikation auf GitHub, die mit der Versionsnummer 1.1 versehen wird. [10], [11]

Die **Version 1.2** erschien im September 2006 und ist als aktuellste Version zu den nun veralteten Versionen 1.0 und 1.1 abwärtskompatibel. Die Spezifikation liegt nicht mehr als Markdown-Dokument vor, sondern als Website, welche von Bairerer gehostet wird. Im GitHub-Repository befinden sich Dateien, die benötigt werden, um die Website quelloffen erzeugen zu können. Die letzte Änderung gab es am 26. Februar 2020. [7]

3.2 Standards außerdem

Durch die folgenden drei offenen Standards werden ebenfalls OCR-Formate definiert:

3.2.1 ALTO

Der Standard **ALTO** (Analyzed Layout and Text Object)[7] ist ein auf XML basierender Standard zur Beschreibung von Layout-Informationen und OCR-Ergebnissen in digitalen Objekten. Die Entwicklung des Standards begann durch das Hamburger Unternehmen CCS Content Conversion Specialists GmbH, welche von Juni bis Dezember 2004 die Versionen 1.0 bis 1.4 veröffentlichte und bis 2010 den Standard verwaltete und hostete, als die US-amerikanische Library of Congress (dt. *Kongressbibliothek*) diese Aufgaben übernahm. Daraufhin erschienen die Versionen 2.0 im Jahre 2010 bis zur aktuellen Version 4.1 im Mai 2019. [12]

3.2.2 PAGE

Der Standard **PAGE** (Page Analysis and Ground-Truth Elements) wird seit 2010 am PRImA (Pattern Recognition and Image Analysis) Research Lab[8] an der University of Salford in Manchester im Vereinigten Königreich entwickelt. [13]

3.2.3 TEI

Der Standard **TEI** (Text Encoding Initiative)[9] wird seit dem Jahre 2000 vom TEI-Konsortium entwickelt und definiert ein Dokumentenformat für die digitale Repräsentation von Texten. Das Konsortium ging aus der gleichnamigen Organisation TEI hervor, welche im Jahre 1987 als textzentrische Community of Practice gegründet wurde. TEI hat sich innerhalb der sog. Digital Humanities (dt. *digitale Geisteswissenschaften*) zu einem De-facto-Standard entwickelt. [14]

[7] https://www.loc.gov/standards/alto/
[8] https://www.primaresearch.org
[9] https://www.tei-c.org

3.3 hOCR-Engines

Als **hOCR-Engines** werden in dieser Arbeit OCR-Engines bezeichnet, welche hOCR als Ausgabeformat unterstützen. Im Folgenden werden drei Beispiele genannt:

3.3.1 OCRopus

OCRopus[10] ist ein freies, in C/C++ und Python entwickeltes, modulares OCR-System, das über Texterkennung hinaus noch Features für die Vor- und Nachverarbeitung bietet. Die Entwicklung begann 2007 durch Thomas Breuel im Rahmen der ICDAR 2007 und ist immer noch aktiv. OCRopus ist auf Projekte zugeschnitten, bei denen in sehr großem Umfang Bücher digitalisiert werden, wie z. B. bei Google Books. [3], [15]

3.3.2 Tesseract

Tesseract[11] ist eine in C++ programmierte OCR-Engine, die zwischen 1985 und 1995 von Hewlett Packard (HP) entwickelt wurde und seit 2006 als Open-Source-Projekt von Google gesponsert wird. [16], [17] Tesseract ist freie Software unter der Apache-Lizenz, befindet sich in aktiver Entwicklung, unterstützt weit mehr als 100 Sprachen und wird mitunter zu den besten OCR-Engines gezählt. [17], [18] Seit 2011 wird hOCR unterstützt. [19]

3.3.3 Cuneiform

Cuneiform ist eine OCR-Software, die seit 1993 vom russischen Unternehmen Cognitive Technologies entwickelt wurde. [20] 2008 wurde sie erstmals frei unter der BSD-Lizenz veröffentlicht. [21] Die Linux-Portierung *Cuneiform for Linux*[12] wurde bis 2011 weiterentwickelt. [22] Obwohl Cuneiform als veraltet betrachtet werden kann, gilt die Software als sehr ausgereift und tritt z. B. noch in Empfehlungen für OCR-Workflows unter Linux auf. [23]

3.4 hOCR-Software

Als **hOCR-Software** werden in dieser Arbeit Anwendungen bezeichnet, welche die Auswertung, die Bearbeitung oder das Betrachten von hOCR-Dokumenten erlauben. Die Anwendungen, die im Rahmen der Recherche dieser Arbeit entdeckt wurden, werden im Folgenden vorgestellt:

[10] https://github.com/tmbdev/ocropy
[11] https://github.com/tesseract-ocr/tesseract
[12] https://launchpad.net/cuneiform-linux

3.4.1 hocrjs

Die Anwendung **hocrjs**[13] wurde von Konstantin Baierer in JavaScript geschrieben und ermöglicht die browserbasierte Anzeige von hOCR-Dokumenten. [24] Die folgenden beiden Screenshots visualisieren die Begrenzungsrechtecke erkannter Bereiche.

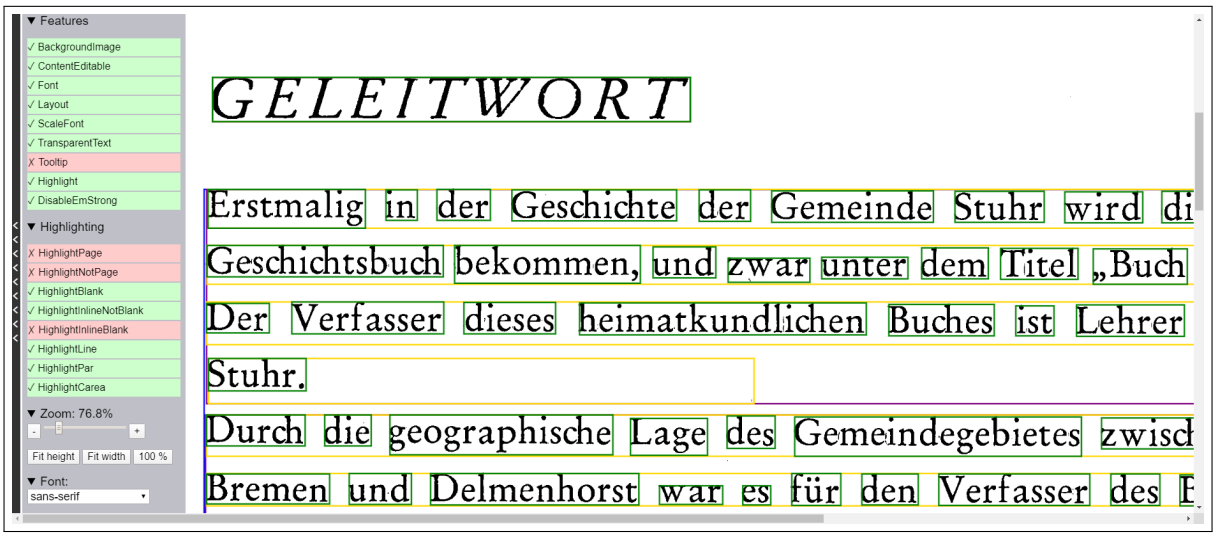


Abbildung 3.2: hOCR-Software: hocrjs – mit Hintergrundbild

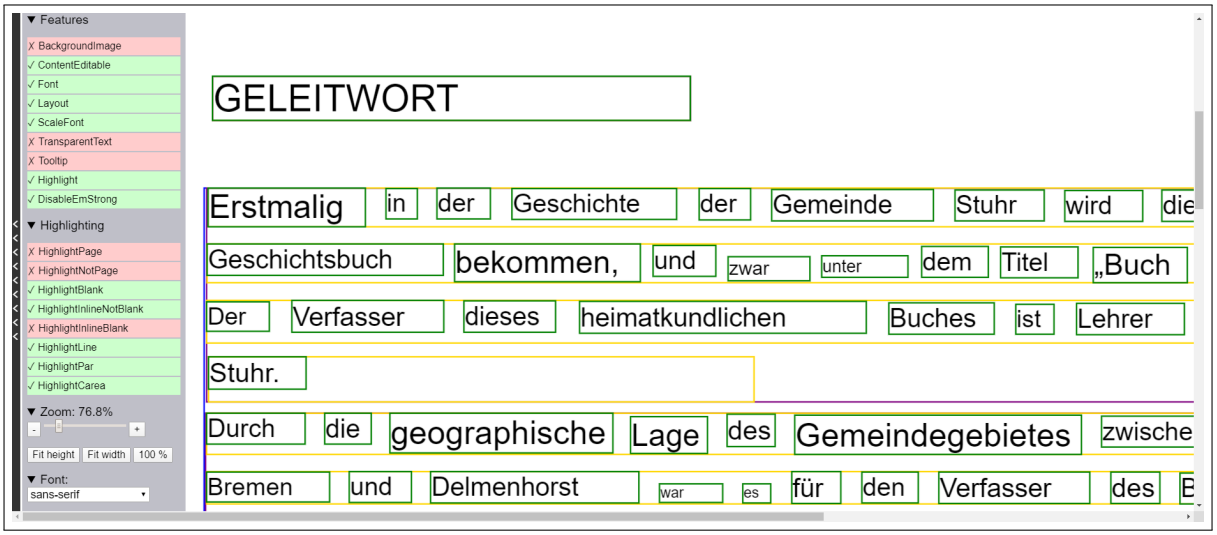


Abbildung 3.3: hOCR-Software: hocrjs – mit erkanntem Text

Die Anwendung wird quelloffen auf GitHub unter der MIT-Lizenz entwickelt und ist damit freie Software. Die Anwendung befindet sich in aktiver Entwicklung und wurde im August 2019 das letzte Mal aktualisiert. [24]

[13] https://github.com/kba/hocrjs

3.4.2 hOCR-Proofreader

Die Anwendung **hOCR-Proofreader**[14] ist eine von Mark Plömer in JavaScript geschriebene GUI-Bibliothek für webbasiertes Korrekturlesen und Bearbeiten von hOCR-Dokumenten. Die Bibliothek wird auf GitHub unter MIT-Lizenz veröffentlicht und ist damit freie Software. In der Readme ist ein Link zu einer Online-Demo[15] angegeben. [25]

Die folgende Abbildung zeigt einen Screenshot für die neunte Seite des Stuhr-Buchs:

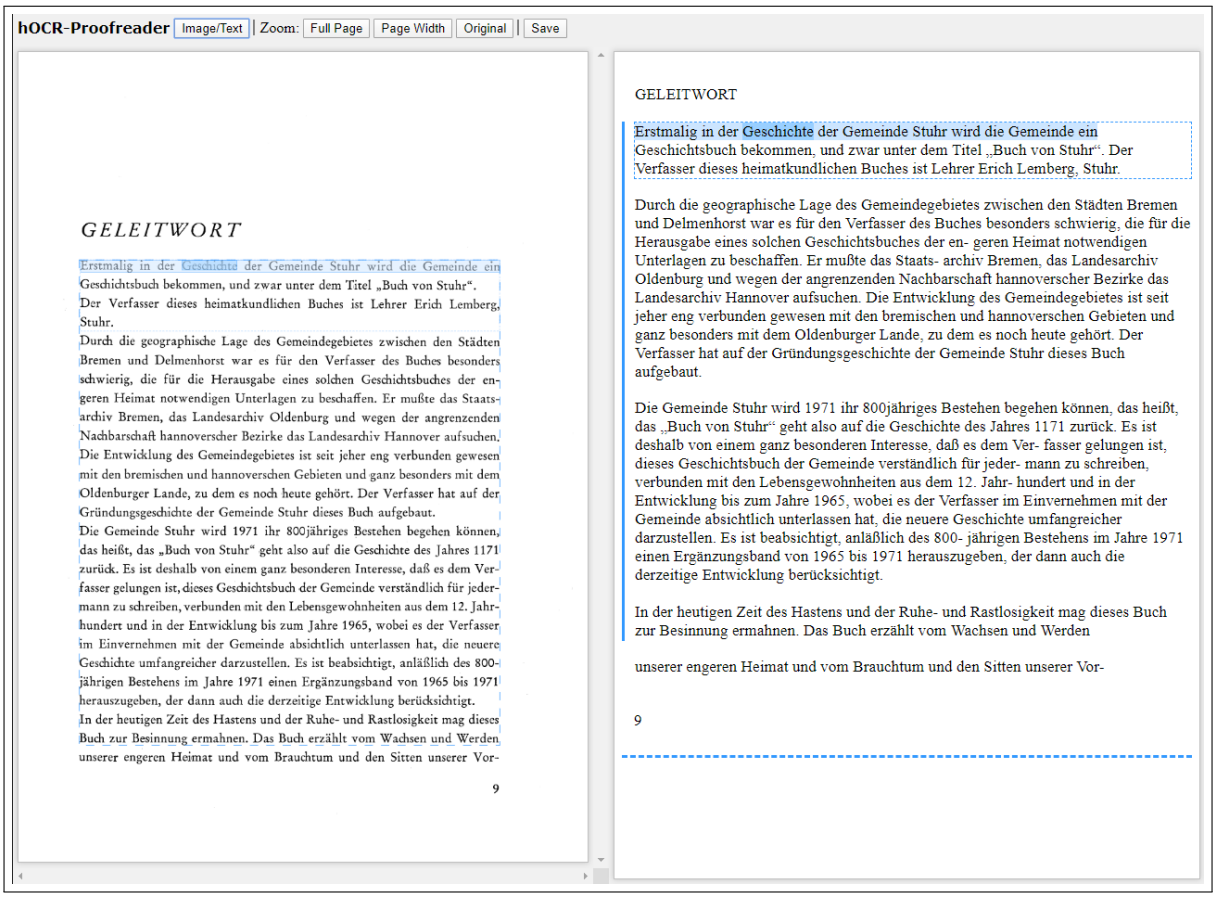


Abbildung 3.4: hOCR-Software: hOCR-Proofreader

Die letzte Aktualisierung erhielt das GitHub-Repository im Januar 2017. Der ausgelieferte Quelltext enthält im Skript `main.js` noch hartkodierte Verweise auf das hOCR-Dokument `demo.hocr` zum Bearbeiten und auf die PHP-Datei `save.php` zum Speichern von Ergebnissen. Daneben deutet die Readme darauf hin, dass das Projekt bisher nicht fertiggestellt wurde. [25]

[14] https://github.com/not-implemented/hocr-proofreader
[15] http://www.not-implemented.de/hocr-proofreader/

3.4.3 moz-hocr-edit

Die Anwendung **moz-hocr-edit**[16] ist eine von James „Jim“ Garrison in JavaScript geschriebene Erweiterung für den Browser Mozilla Firefox (kurz Firefox), um damit hOCR-Dokumente Zeile für Zeile zu korrigieren. Das Bearbeiten ist sowohl über das HTTP-Protokoll als auch im lokalen Dateisystem möglich. Die Erweiterung steht unter den drei Lizenzen MPL, GPL und LGPL und ist damit freie Software. [26], [27]

Die folgende Abbildung zeigt einen Screenshot für die neunte Seite des Stuhr-Buchs:

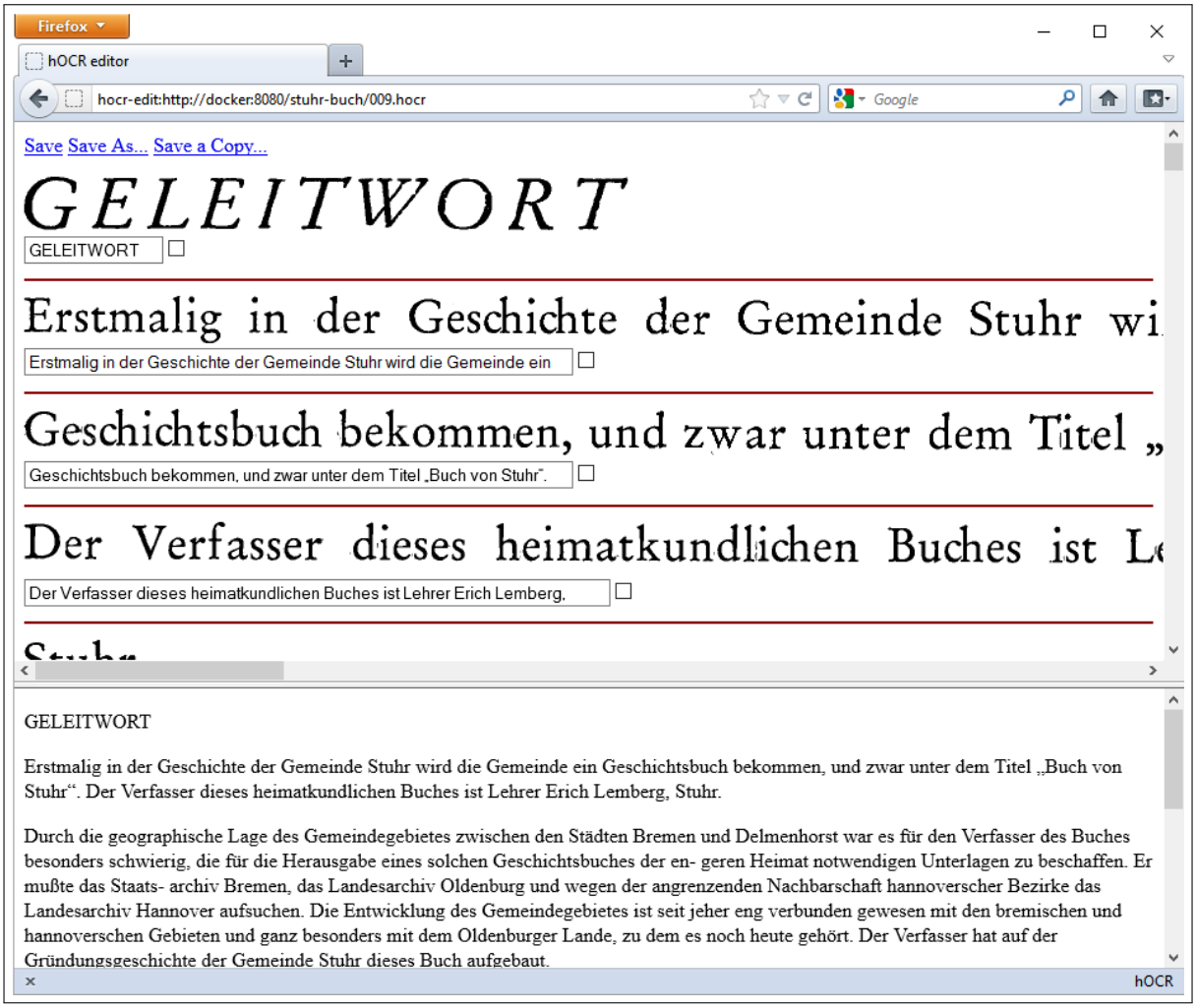


Abbildung 3.5: hOCR-Software: moz-hocr-edit

Die letzte Version (0.4.3) wurde im September 2010 veröffentlicht. [26] Seitdem Garrison die Entwicklung im Janar 2016 ganz einstellt hatte, ist die Erweiterung mit neueren Firefox-Versionen nicht mehr kompatibel. [28]

[16] https://jimgarrison.org/moz-hocr-edit/

3.4.4 hocr-tools

Unter dem Namen **hocr-tools**[17] wird auf GitHub eine freie, quelloffene Sammlung von Tools unter der Apache-Lizenz zur Auswertung, Manipulation und Validierung von hOCR-Dokumenten entwickelt. Programmiersprache ist dabei Python und es wird ein gleichnamiges Python-Paket veröffentlicht. Das Projekt entstand im Jahre 2007 auf der Online-Plattform Google Code, welche Ende 2016 eingestellt worden ist,[29] sodass die Projekt-URL nun auf das aktuelle GitHub-Repository umleitet, welches von Thomas Breuel verwaltet wird, der auch bereits das ehemalige Google-Repository mitbegründete. Die Tool-Sammlung wird in dieser Arbeit als Referenzimplementation des hOCR-Standards betrachtet. [30], [31]

3.4.5 ocr-fileformat

Unter dem Namen **ocr-fileformat**[18] veröffentlicht die Universitätsbibliothek Mannheim (UB Mannheim) auf GitHub eine Sammlung von Befehlszeilenprogrammen zur Validierung und Umwandlung verschiedener OCR-Formate wie hOCR, ALTO und PAGE, zusammen mit einem Web-Interface, das alternativ zur Befehlszeilen-Schnittstelle verwendet werden und bereits öffentlich und frei-zugänglich gehostet wird. [32]

3.4.6 ocr-gt-tools

Die Web-Anwendung **ocr-gt-tools**[19] wird seit April 2016 frei und quelloffen auf GitHub unter der Lizenz GPLv3 von der UB Mannheim entwickelt. Ähnlich wie die Firefox-Erweiterung moz-hocr-tools ermöglicht die Anwendung die zeilenweise Korrektur. Die folgende Abbildung ist ein Ausschnitt des Screenshots aus dem GitHub-Repository: [33]

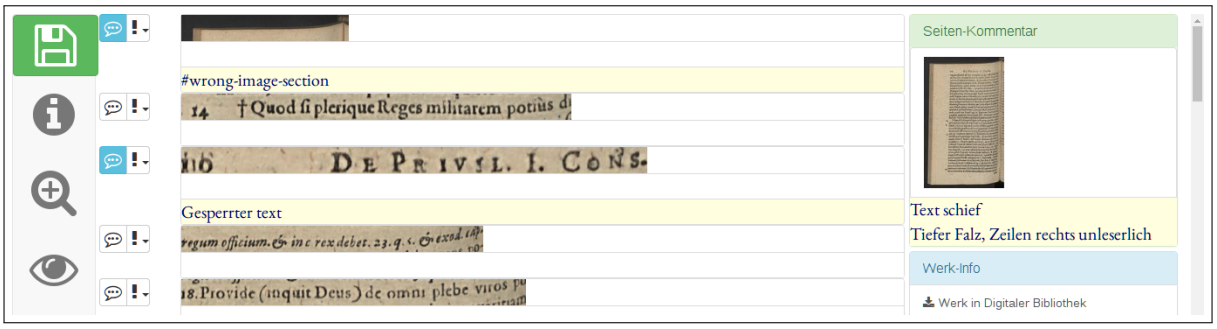


Abbildung 3.6: hOCR-Software: ocr-gt-tools

[17] https://github.com/tmbdev/hocr-tools
[18] https://github.com/UB-Mannheim/ocr-fileformat
[19] https://github.com/UB-Mannheim/ocr-gt-tools

3.5 Software-Bedienkonzepte

Mehrere wiederkehrende **Bedienkonzepte** lassen sich nach der Recherche, welche die Auswertung verschiedener Anwendungen für hOCR umfasst, feststellen:

- Bei der **zeilenweisen Bearbeitung** gibt es pro Zeile jeweils ein Eingabefeld, das den erkannten Text enthält und zur Korrektur bearbeitet werden kann. Anhand der jeweiligen Bounding Box wird direkt darüber ein Seitenausschnitt aus der Bilddatei angezeigt. Auf diese Weise werden die Zeilen der Reihenfolge nach (ohne Berücksichtigung der Positionierung) untereinander aufgeführt. Die zeilenweise Bearbeitung wird z. B. zum Korrekturlesen von den Anwendungen moz-hocr-edit und ocr-gt-tools eingesetzt.
- Bei der **Seitenanzeige** steht die Darstellung von Positionierungen und Abmessungen in Bezug zur Seite im Vordergrund. Die Navigation wird dabei durch das Vergrößern und Verkleinern und durch Bewegung auf der Seite ermöglicht. Diese Form der Darstellung lässt sich mit Bildbetrachtern oder Programmen zur Anzeige von PDF-Dokumenten vergleichen. Sie eignet sich v.a. für das Betrachten und die Bearbeitung geometrischer Informationen. Rechtecke können proportionsgerecht dargestellt werden. Die Anwendung hocrjs erlaubt dabei z. B. das Ein- und Ausschalten folgender Features der Seitenanzeige:
  - Bildanzeige – Die Seite wird als Rastergrafik im Hintergrund dargestellt.
  - Rechtecke – Für erkannte Bereiche wird jeweils die Bounding-Box angezeigt.
  - Textanzeige – Der erkannte Text wird in den Rechtecken gerendert.
- Die **geteilte Ansicht** (engl. *split view*) trennt die Benutzeroberfläche in verschiedene Komponenten. Die Anwendung hOCR-Proofreader unterteilt z. B. in zwei Bereiche, welche nebeneinander (engl. *side by side*) angezeigt werden: Der linke Bereich enthält eine Seitenanzeige als Rastergrafik, der Bereich rechts daneben die OCR-Ergebnisse in Textform. Bei moz-hocr-edit gibt es eine vertikale Teilung der Anzeigefläche: Der obere Bereich dient der zeilenweisen Bearbeitung. Der untere Bereich ist ein Textfeld, das den gesamten erkannten Seitentext enthält.
- Bei der **Synchronisation** von Komponenten wirken sich Änderungen in einer Komponente auch in anderen Komponenten aus, sodass diese also untereinander synchronisiert werden. Ein Beispiel ist die geteilte Ansicht der Anwendung hOCR-Proofreader: Wählt man Text im rechten Bereich aus, so wird die Auswahl durch eine rechteckige Umrandung im linken Bereich visualisiert. Umgekehrt wird rechts automatisch Text ausgewählt, wenn man mit der Maus über die leicht angedeuteten Rechtecke im linken Bereich fährt.

Die verschiedenen Bedienkonzepte haben unterschiedliche Vor- und Nachteile. Alle Anwendungen, die im Rahmen der Recherche dieser Arbeit untersucht wurden, konzentrieren sich jedoch meistens auf nur ein Bedienkonzept. Es wurde keine Anwendung gefunden, welche bspw. die Zeilenbearbeitung mit einer Seitenanzeige synchronisiert.

3.6 Software außerdem

Die im Folgenden beschriebenen Anwendungen erlauben auch die Bearbeitung von OCR-Ergebnissen, jedoch nicht das OCR-Format hOCR:

3.6.1 OCRFeeder

Die Anwendung **OCRFeeder**[20] wurde von Joaquim Rocha im Rahmen seiner unveröffentlichten Master-Thesis in Python für die Desktop-Umgebung GNOME unter Linux und anderen unixähnliche Systemen entwickelt und soll als universelle Software-Suite für OCR-Arbeitsläufe dienen. Die folgenden Punkte beschreiben einen typischen Ablauf: [34]

- **Import**: Bilddaten werden direkt vom Scanner oder als Dateien importiert.
- **Texterkennung**: Für die Bilddaten wird eine Texterkennung durchgeführt, wobei praktisch jede OCR-Software mit Kommandozeilen-Schnittstelle als OCR-Engine dienen kann. Tesseract und Cuneiform werden automatisch unterstützt.
- **Bearbeitung**: Die OCR-Ergebnisse können anschließend vom Benutzer gesichtet und bearbeitet werden. Die Anwendung erlaubt das Laden und Speichern von Benutzerprojekten in einem nativen Dateiformat mit der Namenserweiterung „.ocrf“ das nicht näher dokumentiert ist und von keinen weiteren Anwendungen unterstützt wird.
- **Export**: Das OCR-Projekt kann als OpenDocument-Datei oder PDF-Dokument exportiert werden. Die Unterstützung für hOCR (als Format für den Export) wurde im Jahre 2011 zwar angekündigt, bisher jedoch nicht umgesetzt. [35]

Seit der Erstveröffentlichung im März 2009 befindet sich die GPL-lizenzierte Software in aktiver Entwicklung, wobei jedoch die letzte stabile Version im Jahre 2014 (0.8.1) erschienen ist. [36]

3.6.2 ABBYY FineReader

Die Desktop-Anwendung **ABBYY FineReader**[21] ist eine proprietäre und kostenpflichtige OCR-Software, die vom multinationalen Unternehmen ABBYY entwickelt wird. [37] Mit der beinhalteten Komponente „OCR Editor“ können die Ergebnisse der Texterkennung bearbeitet werden. [38] ABBYY definiert ein OCR-Format, das auf XML basiert und für das Exportieren der Ergebnisse verwendet werden kann. [39]

[20] https://wiki.gnome.org/Apps/OCRFeeder
[21] https://www.abbyy.com/en-eu/finereader/

4 Konzeption

Die **Konzeption** der Arbeit verfolgt das Top-Down-Design: Zunächst wird die Software-Architektur und das Entwicklungsumfeld geplant, das sich aus Programmiersprache, Entwicklungsumgebung und Software-Plattform ergibt.

4.1 Software-Architektur

Die **Software-Architektur** des gesamten Software-Softwares lässt anhand des Schichten-Modells beschreiben, dessen herkömmliche Variante die Drei-Schichten-Architektur ist. Das zu entwickelnde Software-System besteht in dieser Arbeit jedoch der Einfachheit halber aus nur zwei Schichten: [40]

- Die **unterste Schicht** – das Software-Backend – umfasst die Bereiche Anwendungslogik und Datenzugriff, welche in einer Drei-Schichten-Architektur jeweils einzelne Schichten wären. [40] Das beinhaltet die hOCR-Bibliothek, die Anbindung an OCR-Engines und die Softwaretests.
- Die **oberste Schicht** – das Software-Frontend – vereint Anwendungs- und Präsentationsschicht, welche in der herkömmlichen Variante ebenso jeweils einzelne Schichten wären. [40] Das beinhaltet den eigentlichen hOCR-Editor, also die grafische Benutzeroberfläche.

Eine **Solution-Datei** (engl. *solution file*) organisiert die Code-Basis der Software-Lösung (engl. *solution*) und verweist auf die eigentlichen *Projekte*. In dieser Arbeit verweist die Solution-Datei `hocr-editor.sln` auf C#-Projekte, welche wiederum durch C#-Projektdateien mit der Endung `.csproj` organisiert werden. Beim *Erstellen* („Kompilieren“) der Solution werden je nach Konfiguration alle oder nur bestimmte Projekte erstellt bzw. kompiliert. Beim Kompilieren von C#-Projekten werden sog. Assemblies erzeugt. [41]

Ein **Assembly** ist gemäß CLI eine Bibliothek, die kompilierten Code, Daten und Resourcen enthalten kann. Auf Dateisystemebene erscheinen Assemblies als dynamische Bibliothek mit der Dateiendung `.dll` (engl. *dynamic-link library*) oder als ausführbares Programm mit der Dateiendung `.exe` (engl. *executable file*). [42]

4.1.1 Programmiersprache

Die **Programmiersprache C#** (gespr. „see sharp“)[1] ist typensicher, multiparadigmatisch und universell. Sie wurde für Microsoft von Anders Hejlsberg entworfen und wird aktuell unter der Leitung von Mads Torgersen weiterentwickelt. Bis zur Version 5.0 wurde C# durch die einander entsprechenden Standards ECMA-334 (Ed. 5) und ISO/IEC 23270:2018 spezifiziert. Seit September 2019 ist C# 8.0 die aktuellste Version. [43]–[46]

4.1.2 Entwicklungsumgebung

Die Entwicklungsumgebung **Microsoft Visual Studio**[2] wird von Microsoft entwickelt und vertrieben. Für die Entwicklung in C# 8.0 ist Visual Studio 2019 (VS 2019) erforderlich, welches seit April 2019 die aktuellste Version ist. [46]

4.1.3 Software-Plattform

Unter **.NET** (gespr. „dot net“)[3] werden mehrere Software-Plattformen für die Entwicklung und Ausführung von Anwendungsprogrammen zusammengefasst, welche die Common Language Infrastructure (CLI) implementieren. Die CLI wird durch die einander entsprechenden Standards ECMA-335 (Ed. 6) und ISO/IEC 23271:2012 spezifiziert. Die Programmiersprache C# ist für den Einsatz unter .NET vorgesehen. [42]

- Das **.NET Framework** ist eine von Microsoft entwickelte und im Jahr 2002 erstmalig veröffentlichte proprietäre Plattform für das Betriebssystem Microsoft Windows, die zuletzt als Version 4.8 im Juli 2019 erschien. Anfangs war mit dem einfachen Begriff .NET meist das .NET Framework gemeint. Erst später entwickelte er sich zum Sammelbegriff. Dazu beigetragen hatte z. B. die Gründung der .NET Foundation durch Microsoft im März 2014, um die quelloffene Entwicklung im Umfeld des .NET Frameworks zu verbessern. [47] Im Rückblick gilt das .NET Framework als die klassische, monolithische Implementation. Das .NET Framework bietet Unterstützung bis C# 7.3. [46], [48]
- **Mono**[4] erschien im Jahr 2004 als alternative, quelloffene und plattformübergreifende Implementation des .NET Frameworks. Mono wurde anfangs vom Unternehmen Ximian entwickelt, das von Miguel de Icaza mitbegründet wurde, und noch vor der Erstveröffentlichung vom Unternehmen Novell aufgekauft wurde. Im Juli 2011 wurde das Software-Projekt dem Unternehmen Xamarin übergeben, das auf .NET-Entwicklung bei mobilen Anwendungen abzielt und mittlerweile ein Tochterunternehmen von Microsoft ist. In der plattformübergreifenden Spiele-Engine

[1] https://docs.microsoft.com/en-us/dotnet/csharp/
[2] https://visualstudio.microsoft.com
[3] https://dotnet.microsoft.com
[4] https://www.mono-project.com

Unity wird Mono zur Code-Ausführung verwendet. Mono bietet volle Unterstützung bis C# 6.0 und teilweise Unterstützung für C# 7.x. [49]–[51]

- **.NET Core** ist Microsofts modular aufgebauter, quelloffener und plattformübergreifender Nachfolger des .NET Frameworks. Auf die Erstveröffentlichung im Juni 2016 folgten weitere Versionen bis zur aktuellsten Version .NET Core 3.1 im Dezember 2019. Unterstützung für C# 8.0 besteht seit .NET Core 3.0. [52], [53]
- **.NET 5.0** soll ab Dezember 2020 die drei Plattformen, die zuvor genannt wurden, vereinigen und weiterführen. [54]

Um die Verwendung von C# 8.0 zu ermöglichen, wurde im Rahmen dieser Arbeit entschieden, die Plattform .NET Core 3.0 oder neuer einzusetzen, was ebenfalls wie die Version der Programmiersprache den Einsatz von Visual Studio 2019 erfordert. [46]

4.2 hOCR-Bibliothek

Die **hOCR-Bibliothek** soll den programmatischen Zugriff auf hOCR-Dokumente zur Auswertung und Veränderung derselben ermöglichen. Weil hOCR auf HTML aufbaut, bedient man sich in dieser Arbeit der Programmierschnittstelle DOM, welche im Folgenden behandelt wird:

4.2.1 DOM-Standard

Das **DOM** (Document Object Model)[5] ist eine durch die WHATWG und das W3C (World Wide Web Consortium) standardisierte API für HTML- und XML-Dokumente, welche diese als Baumstruktur repräsentiert, um Datenzugriff und -manipulation zu erlauben. Begriffe wie *Knoten*, *Kinder*, *Eltern* und *Wurzel* sind im Folgenden daher im graphentheoretischen Sinne zu verstehen. Jedes XML- oder HTML-Element wird durch einen Knoten (engl. *Node*) in der Baumstruktur repräsentiert. Kommentare und Zeichenfolgen werden ebenso als Knoten erfasst. Die API ist plattform- und programmiersprachenunabhängig. Bei objektorientierter Programmierung werden die Knoten z. B. durch Klassen bzw. Objekte modelliert, wobei die Methoden in den Klassen Datenzugriff und -manipulation ermöglichen. [55]

4.2.1.1 DOM-Implementation

Eine **DOM-Implementation** für .NET bietet die Bibliothek HtmlAgilityPack (HAP)[6], welche als freie Software unter der MIT-Lizenz auf GitHub[7] aktiv entwickelt wird und

[5] https://dom.spec.whatwg.org
[6] https://html-agility-pack.net
[7] https://github.com/zzzprojects/html-agility-pack

als NuGet-Paket [8] bereits mehr als 24,5 Millionen mal installiert wurde. [56]–[58] Das folgende C#-Listing zeigt, wie der Titel des HTML-Dokuments `index.html` damit extrahiert und anschließend auf der Konsole ausgegeben werden kann:

```
1 var doc = new HtmlDocument();
2 doc.Load("index.html");
3 HtmlNode root = doc.DocumentNode;
4 HtmlNode title = root.SelectSingleNode("//html/head/title");
5 Console.WriteLine(title.InnerText);
```

Listing 4.1: C# – HtmlAgilityPack – DOM-Implementation

Durch **HtmlDocument** wird ein HTML-Dokument repräsentiert, wobei die Klasseneigenschaft `DocumentNode` die Dokumenten-Wurzel darstellt, also einen virtuellen Knoten, der alle tatsächlichen Knoten oberster Ebene als Kinder hat.

Durch **HtmlNode** wird ein HTML-Knoten mit folgenden Eigenschaften dargestellt:

- `NodeType` unterscheidet zwischen Wurzel, Element, Kommentar und Text.
- `ChildNodes` ist die Liste der Kinder des Knotens.
- `ParentNode` verweist auf den Elternknoten.
- `Attributes` ist die Liste HTML-Attribute des Knotens.

4.2.1.2 DOM-Erweiterung

Die **DOM-Erweiterung**, welche im Rahmen dieser Arbeit entwickelt wird, ermöglicht den programmatischen Zugriff auf hOCR-Elemente, hOCR-Properties und hOCR-Metadaten, damit diese ausgewertet und manipuliert werden können. Elemente, Properties und Metadaten werden dazu durch ein Objektmodell abgebildet, das in folgenden Abschnitten beschrieben wird.

Das allgemeine Objektmodell reicht für eine programmatische Auswertung und Manipulation von hOCR-Elementen -Properties und -Metadaten nicht aus. Das folgende C#-Listing ermittelt mittels XPath das erste hOCR-Element im hOCR-Dokument `009.hocr` und gibt dessen hOCR-Properties als String (s. letzte Kommentarzeile) aus:

[8] https://www.nuget.org/packages/HtmlAgilityPack/

```
var doc = new HtmlDocument();
doc.Load("009.hocr");
HtmlNode root = doc.DocumentNode;
HtmlNode element = root.SelectSingleNode("//*[" +
  "starts-with(@class, 'ocr_') or " +
  "starts-with(@class, 'ocrx_')]");
string properties = element.Attributes["title"].Value;
Console.WriteLine(properties);
// image "009.png"; bbox 0 0 3472 4937; ppageno 9
```

Listing 4.3: C# – HtmlAgilityPack – DOM-Limitation

Wenn nun die Aufgabe bspw. die Manipulation der Koordinaten der Bounding-Box wäre, ginge dies nur über aufwändige String-Manipulation. Dieser Umstand ist zu erwarten, weil das DOM ja nur ein Objektmodell für HTML bietet, nicht aber für die Erweiterung hOCR bzw. dessen Grammatik.

Der **Token** (vgl. Abschnitt 3.1.5) gemäß allgemeiner Grammatik für die hOCR-Properties trägt daher bei der Umsetzung dieser Arbeit eine bedeutende Rolle. Das folgende Klassendiagramm modelliert die beiden möglichen Token `ascii_word` und `delimited_string`:

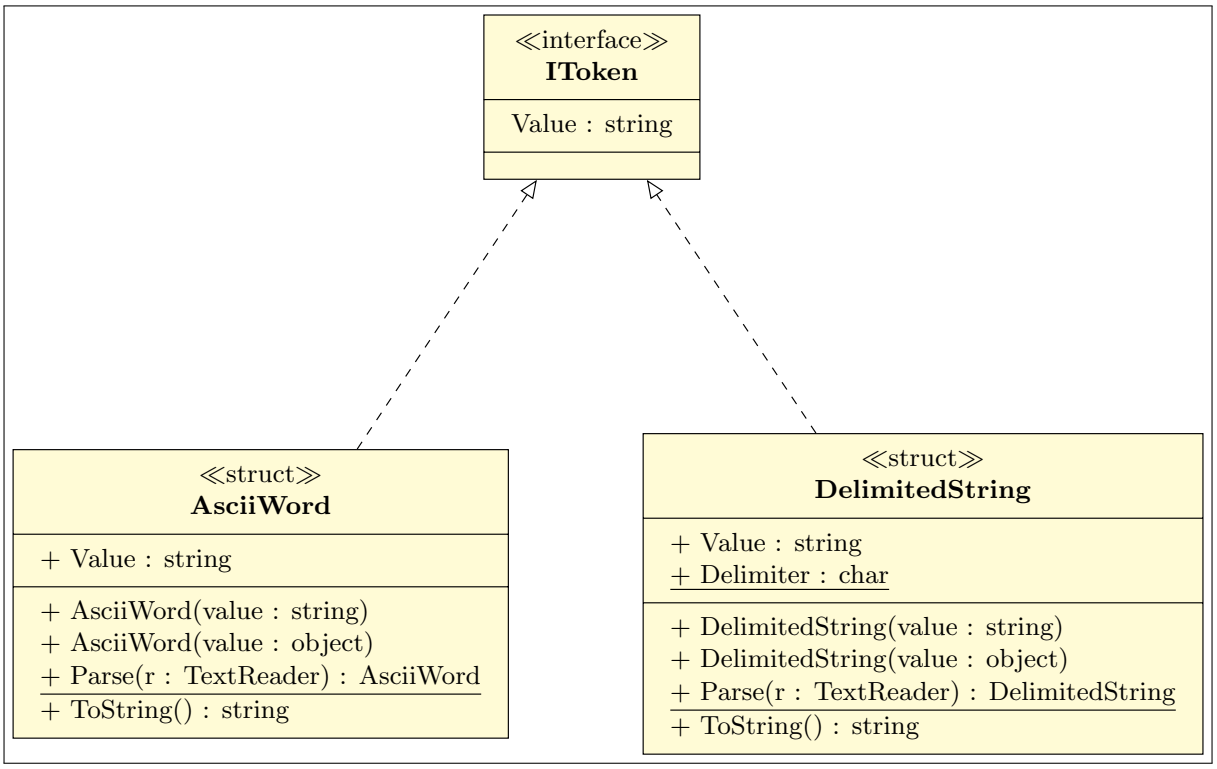


Abbildung 4.1: Klassendiagramm: Properties-Grammatik

Alle **Elemente und Properties** sollen durch Klassen modelliert werden, welche das folgende Diagramm exemplarisch darstellt:

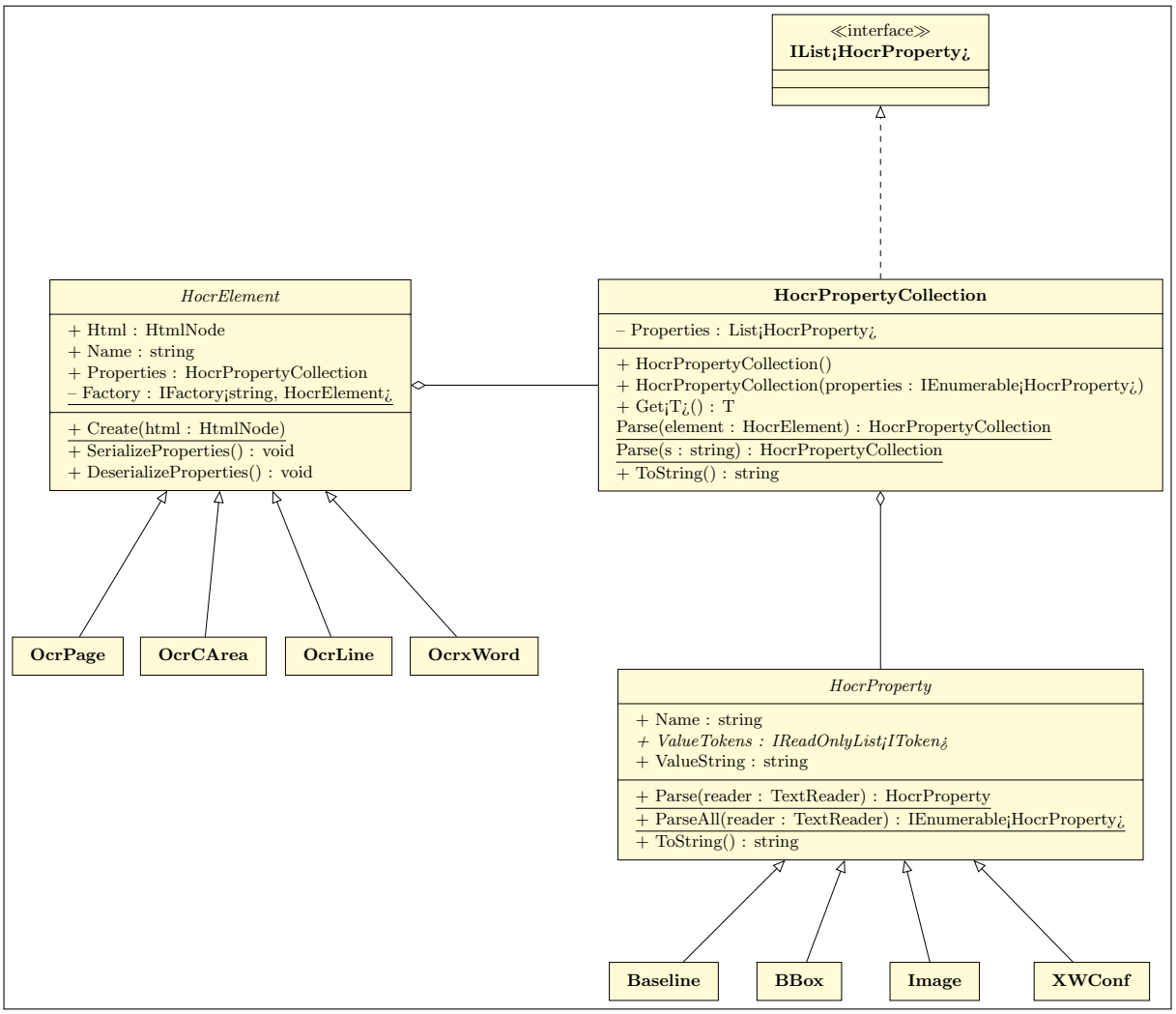


Abbildung 4.2: Klassendiagramm: Properties und Elemente

4.3 hOCR-Editor

Der **hOCR-Editor** stellt das Software-Frontend bzw. die grafische Benutzeroberfläche der Anwendung dar. In folgenden Abschnitten wird dazu der Einsatz von Frameworks und Architekturmustern geplant. Ein Wireframe wird zur Veranschaulichung entwickelt und die grafischen Komponenten der Oberfläche werden damit entworfen und definiert.

4.3.1 WPF und Avalonia

Das Grafik-Framework **WPF** (Windows Presentation Foundation)[9] wird von Microsoft für .NET entwickelt und wurde im November 2006 für das .NET Framework 3.0 erstveröffentlicht. Die aktuellste Version wird auch von .NET Core 3.x unterstützt. Seit Oktober 2018 steht WPF quelloffen als freie Software unter der MIT-Lizenz auf GitHub zur Verfügung. WPF erlaubt eine Trennung von GUI- und Anwendungslogik und bietet dazu noch viele allgemeine und spezielle Features zur Visualisierung. Beispielsweise ist es mit WPF möglich, grafische Primitive zu zeichnen, was genutzt werden könnte, um z. B. Rechtecke für erkannte Objekte darzustellen. Als Auszeichnungssprache zur GUI-Strukturierung dient XAML, welche auf XML aufbaut. Es gibt daher keinen Code der vom Designer generiert wird und nicht von Hand gepflegt werden sollte, sondern XML-Dateien, die durchaus für die händische Bearbeitung gedacht sind, ähnlich der Entwicklung mit Android Studio.[10] [59]–[62]

Avalonia[11] ist ein plattformübergreifendes XAML-Framework für .NET. [63]

4.3.2 Model View ViewModel

Das Architekturmuster **MVVM** (Model View ViewModel) erleichtert bei einem Software-System die Trennung von Anwendungslogik und (grafischer) Benutzerschnittstelle. Die drei Komponenten des Entwurfsmusters werden folgend beschrieben: [64]

- Das **Model** (dt. *Modell*) beinhaltet die Geschäftslogik bzw. Anwendungslogik und hat keinen Bezug zur Darstellung durch die Benutzerschnittstelle. Bei der zu entwickelnden Anwendung kann die hOCR-Bibliothek als Model bezeichnet werden.
- Das **View** (dt. *Ansicht*) umfasst die Teile des Software-Systems, die der reinen Darstellung dienen und unabhängig von der Geschäftslogik bestehen. Die „höhere“ Darstellungslogik ist ebenso davon ausgeschlossen. Bei WPF kann XAML-Quelltext in diesem Sinne als View bezeichnet werden.
- Das **ViewModel** (dt. *Ansichts-Modell*) dient der Vermittlung zwischen Model und View, indem es eine Darstellung von Daten aus der Geschäftslogik abstrahiert. Die Daten des ViewModels werden mit den Daten des Models synchronisiert. Zwischen dem View und dem ViewModel sorgt ein sog. *data binding* ebenso für die Synchronisation von Daten. Das ViewModel fungiert damit als Vermittler.

Das MVVM wurde von Microsoft entwickelt und wird für den Einsatz im Kontext von WPF empfohlen. Das MVVM kann als Variante des Musters Model-View-Controller (MVC) betrachtet werden und ähnelt dem Entwurfsmuster Presentation Model, das von Martin Fowler entwickelt wurde. [64], [65]

[9] https://docs.microsoft.com/en-us/dotnet/framework/wpf/
[10] https://developer.android.com/guide/topics/ui/declaring-layout
[11] https://avaloniaui.net

4.3.3 MVVM-Frameworks

Das Framework **ReactiveUI**[12] erleichtert auf .NET-Plattformen die Umsetzung des Entwurfsmusters MVVM und verfolgt dabei außerdem die sog. reaktive Programmierung (engl. *reactive programming*) als Programmierparadigma. Alternative MVVM-Frameworks sind z. B. das minimalistische MVVM Light Toolkit[13] und Prism[14]. Bei grafischen Anwendungen höher Komplexität wird der Einsatz von ReactiveUI empfohlen, das außerdem besonders aktiv entwickelt wird. [66]–[68]

4.3.4 Wireframe

Ein **Wireframe** wird zur Konzeption grafischen Benutzeroberfläche entwickelt. Das Augenmerk liegt dabei hauptsächlich auf der Anordnung der Bedienelemente und weniger auf deren Gestaltung im Detail. Die folgende Abb. 4.3 zeigt den Wireframe, der in dieser Arbeit für das Hauptfenster der Editor-Anwendung entwickelt wurde.

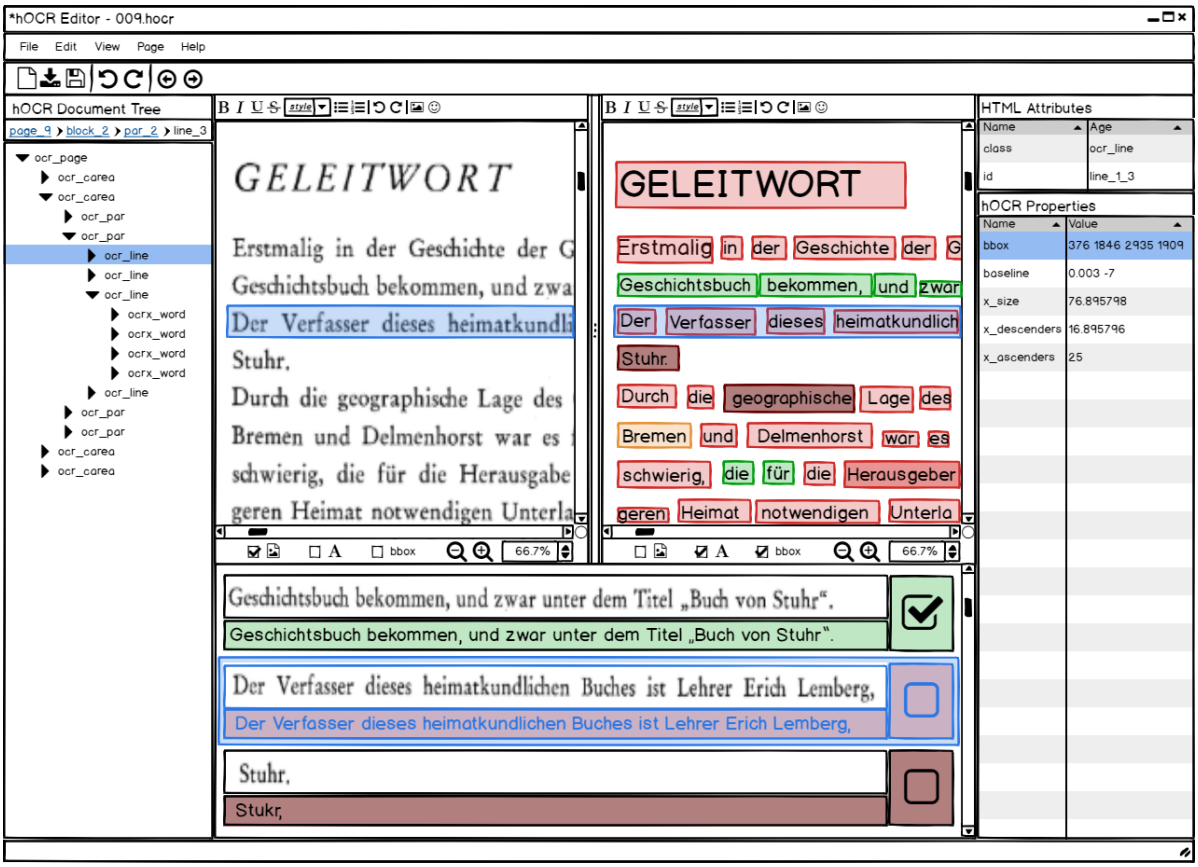


Abbildung 4.3: hOCR-Editor – Wireframe

[12] https://reactiveui.net
[13] http://www.mvvmlight.net
[14] https://prismlibrary.com

4.3.5 Komponenten

Mehrere **Komponenten** sind im Hauptfenster enthalten. Ihr Entwurf ist teilweise an die Bedienkonzepte der Software-Anwendungen angelehnt, welche im Rahmen der Recherche dieser Arbeit ermittelt wurden. (vgl. Abschnitt 3.5)

- Die **Baumansicht** mit dem Titel „hOCR Document Tree“ im linken Bereich des Fensters gibt die hierarchische Struktur des geöffneten hOCR-Dokuments wieder. Dazu kann die Baumstruktur mittels DOM gewonnen werden und dann anschließend so weit „gekürzt“ werden, sodass nur noch die hOCR-Elemente enthalten sind. HTML-Knoten wie `<head>` und `<body>` werden daher nicht angezeigt. Außerdem wird der ausgewählte Hierarchiepfad mithilfe einer Navigationszeile über der Baumansicht im Sinne der sog. Brotkrümelnavigation (engl. *bread crumb navigation*) veranschaulicht.
- Bei den **Seitenanzeigen** in der Mitte des Fensters steht die Darstellung von Positionierungen und Abmessungen in Bezug auf die aktuell ausgewählte Seite im Vordergrund. Wichtigstes Feature ist die Visualisierung der Bounding-Boxes. in Bezug zur Seite im Vordergrund. Die hOCR-Elemente werden nicht direkt sondern nur implizit durch ihre Bounding-Boxes visualisiert.
- Die **hOCR-Properties** werden allesamt in einer Tabelle im rechten Fensterbereich aufgeführt, wobei die linke Spalte die Namen der Properties enthält und die rechte Spalte die Werter der Properties aufführt. Die Zellen der Tabelle erlauben als Textfelder die Eingabe und Veränderung von Property-Namen und -Werten. Bedienelemente dieser Art werden auch als Rasteransicht (engl. *grid view*) bezeichnet.
- Die **HTML-Attribute** werden in einer Tabelle darüber gleichermaßen aufgelistet. Das Attribut `title` ist davon ausgeschlossen, weil es bereits durch die Tabelle der hOCR-Properties wiedergegeben wird.
- Die **zeilenweise Bearbeitung** wird durch den unteren Fensterbereich ermöglicht.

Eine **Synchronisation** der Komponentenzustände untereinander sorgt dafür, dass sich Veränderungen in einem Fensterbereich auch in allen anderen betroffenen Bereichen auswirkt. Damit ist z. B. sichergestellt, dass die Löschung eines hOCR-Elements in der Baumstruktur dazu führt, dass auch die entsprechende Bounding-Box in beiden Seitenanzeigen nicht mehr angezeigt wird.

Die **Auswahl** bestimmter hOCR-Inhalte wird in allen Komponenten mittels blauer Farbtöne gekennzeichnet und ist z. B. von der Synchronisation betroffen. Es wird zwischen der Auswahl von hOCR-Elementen und der Auswahl von hOCR-Properties des ausgewählten Elements unterschieden. Bei Auswahl eines Elements wird standardmäßig (wenn verfügbar) dessen Bounding-Box ausgewählt.

Die **Rot-Grün-Färbung** von Bedienelementen soll das Korrekturlesen visuell unterstützen: Ein Grüntön dient dazu, die Zeilen und Wörtern zu markieren, welche manuell

(ggf. nach einer Korrektur) vom Benutzer akzeptiert wurden. Mit Rottönen werden dagegen alle anderen OCR-Ergebnisse versehen, wobei die Helligkeit des Rottons jeweils die Erkennungssicherheit anhand der hOCR-Property `x_wconf` (s. Abschnitt 3.1.4.3) veranschaulichen soll. Dunklere Rottöne verdeutlichen geringere Sicherheiten.

4.4 Softwaretests

Die **Softwaretests**, welche entwickelt werden, sollen das Framework xUnit[15] nutzen.

Unter der **Reserialisierung** wird in dieser Arbeit die Deserialisierung mit anschließender Serialisierung verstanden: Aus einer (seriellen) Bytefolge wird ein hOCR-Dokument deserialisiert, welches dann wieder zu einer Bytefolge serialisiert wird. Das Dokument wird durch eine Bytefolge repräsentiert. Die entwickelten Softwaretests führen die Reserialisierung durch und prüfen daraufhin hinsichtlich zweier Aspekte:

- Die **Integrität** der deserialisierten Bytefolge bedeutet, dass sich diese nicht von der serialisierten Bytefolge unterscheidet und damit unverändert (integer) ist.
- Die **Manipulation** des hOCR-Dokuments kann nur dann als erfolgreich betrachtet werden, wenn sie zu einer *Veränderung* der Bytefolge führt.

Der Datensatz **Google 1000 Books** wurde (zusammen mit hOCR und OCRopus) im Rahmen der ICDAR 2007 veröffentlicht und umfasst mit etwa 119 GB tausend gemeinfreie Bücher mitsamt Metadaten, Scans der Seiten und hOCR-Dateien umfasst. [1], [69] Dieser Datensatz sollte sich sehr gut eignen, um die hOCR-Bibliothek mit Testdaten zu versorgen. Wenn die „Reserialisierung“ – Deserialisierung mit anschließender Serialisierung – keine Unterschiede in den hOCR-Dateien verursacht und noch dazu keine Fehler bzw. Exceptions auftreten, kann davon ausgegangen werden, dass die Software fehlerfrei funktioniert.

[15] https://xunit.net

5 Realisierung

Die **Realisierung** ist die Umsetzung der Bestandteile, welche in der Konzeption im vorigen Kapitel planend ausgearbeitet wurden.

5.1 hOCR-Bibliothek

Die **hOCR-Bibliothek** wird durch das Assembly `hOCR` realisiert, welches eine Software-Bibliothek für den programmatischen Zugriff auf hOCR-Dokumente darstellt.

5.1.1 hOCR-Elemente

Die Klasse **HocrElement** ist die abstrakte Basisklasse zur Repräsentation eines hOCR-Elements. Weil jedes hOCR-Element auch ein HTML-Element ist und über einen Namen und eine Menge von Properties verfügt, umfasst die Klasse dementsprechend über die drei Eigenschaften `Html`, `Name` und `Properties`:

```
1 public abstract class HocrElement
2 {
3     public HtmlNode Html { get; internal set; }
4     public string Name { get; internal set; }
5     public HocrPropertyCollection Properties { get; set; }
6     ...
7 }
```

Listing 5.1: C# – Klasse `HocrElement` – Eigenschaften

Es werden alle vom Standard spezifizierten Elemente durch jeweils eine Klasse abgebildet, welche von der der abstrakten Basisklasse `HocrElement` erbt (s. Anh. 8.1.1).

5.1.1.1 OcrPage – Implementationsklasse

Die Klasse **OcrPage** repräsentiert als konkrete Implementationsklasse z. B. ein `ocr_page`-Element. Sie verfügt der Einfachheit halber über die Eigenschaft `Image`, zumal der Standard für `ocr_page`-Element den Gebrauch der `image`-Property empfiehlt:

```
[HocrName("ocr_page")]
public class OcrPage : HocrElement
{
    public Image Image => Properties.Get<Image>();
}
```

Listing 5.3: C# – Klasse `OcrPage` – Eigenschaften

5.1.1.2 HocrElement.Create – Fabrikmethode

Die Methode **HocrElement.Create** ist eine Fabrikmethode, um aus einem beliebigen HTML-Knoten das entsprechende hOCR-Element zu erzeugen. (s. Abschnitt 5.1.1.2) Es wird zunächst der Elementname wird aus dem `class`-Attribut gewonnen und wiederum der Fabrikmethode `Create` der Fabrikklasse `HocrElementFactory` mitgeteilt, damit diese die entsprechende Implementationsklasse instanziiert. Die Instanz wird dann mit Werten initialisiert und zurückgegeben. Wenn der HTML-Knoten kein hOCR-Element ist, wird direkt zu Beginn der Methode `null` zurückgegeben.

```
public static HocrElement Create(HtmlNode node)
{
    if (node.IsHocrElement())
    {
        string name = node.GetAttributeValue("class", null);
        HocrElement element = Factory.Create(name);

        element.Html = node;
        element.Name = name;
        element.DeserializeProperties();

        return result;
    }
    return null;
}
```

Listing 5.5: C# – Klasse `HocrElement` – Fabrikmethode

5.1.1.3 HocrElementFactory – Fabrikklasse

Die Klasse **HocrElementFactory** ist eine Fabrikklasse, die das Feld `dictionary` enthält, ein assoziatives Datenfeld, welches jedem hOCR-Element-Namen (`string`) den Typ (`System.Type`) der entsprechenden Implementationsklasse zuordnet. Die Fabrikmethode `Create` ermittelt anhand des Namens über das assoziative Datenfeld den Typen und erzeugt dann mittels Reflection (über die Klasse `Activator`) eine Instanz davon, welche schließlich zurückgegeben wird:

```
    public HocrElement Create(string name) =>
    (HocrElement)Activator.CreateInstance(dictionary[name]);
```

5.1.1.4 HocrNameAttribute – Namenszuordnung

Die **Namenszuordnung** geschieht über das assoziatives Datenfeld `dictionary` der Fabrikklasse. Das Datenfeld muss befüllt werden. Das *imperative* und hartkodierte Befüllen könnte folgendermaßen geschehen:

```
dictionary.Add("ocr_page", typeof(OcrPage));
dictionary.Add("ocr_line", typeof(OcrLine));
...
```

Listing 5.8: C# – Namenszuordnung – imperativer Ansatz

Die Klasse **HocrNameAttribute** ermöglicht eine *deklarative* Namenszuordnung. Die Implementationsklassen der hOCR-Elemente werden direkt mit Metadaten in Form dieses Attributs versehen:

```
[AttributeUsage(AttributeTargets.Class)]
public class HocrNameAttribute : Attribute
{
    public string Name { get; }

    public HocrNameAttribute(string name) =>
        Name = name;
}
```

Listing 5.10: C# – Namenszuordnung – Klasse `HocrNameAttribute`

Die Klasse OcrPage wird z. B. mit der Zeichenfolge „ocr_page“ versehen:

```
[HocrName("ocr_page")]
public class OcrPage : HocrElement
{
    ...
}
```

Listing 5.12: C# – Namenszuordnung – Beispiel `ocr_page`

Mittels Reflection kann dann zur Laufzeit ermittelt werden, welche Klassen im aktuell ausführenden Assembly von der Basisklasse HocrElement erben:

```
IEnumerable<Type> types = Assembly.GetExecutingAssembly().GetTypes()
    .Where(t => typeof(HocrElement).IsAssignableFrom(t));
```

Listing 5.14: C# – Namenszuordnung – Aufzählung aller Implementationsklassen

Den Klassen wird dann per Reflection der Name aus dem Attribut entnommen. Der Name bildet dann mit der Klasse ein Schlüssel-Wert-Paar für das assoziative Datenfeld `dictionary`:

```
foreach (Type t in types)
{
    string name = type.GetCustomAttribute<HocrNameAttribute>()?.Name;
    if (name != null)
        dictionary.Add(name, type);
}
```

Listing 5.16: C# – Namenszuordnung – Bildung der Schlüssel-Wert-Paare

5.1.1.5 Erweiterungsmethoden

Die **Identifikation** eines hOCR-Elements geschieht mithilfe der Erweiterungsmethode `IsHocrElement`, welche für einen HTML-Knoten einen booleschen Wert zurückgibt, der nur dann `true` ist, wenn der Knoten ein hOCR-Element ist. Dazu wird geprüft, ob ein HTML-Attribut namens `title` vorhanden ist, dessen Wert mit `ocr_` oder `ocrx_` beginnt:

```
public static bool IsHocrElement(this HtmlNode node)
{
    var a = node.GetAttributeValue("class", string.Empty);
    return a.StartsWith("ocr") || a.StartsWith("ocrx_");
}
```

Listing 5.18: C# – Erweiterungsmethode `IsHocrElement`

Die **Aufzählung** (engl. *enumeration*) von hOCR-Elementen erfolgt mithilfe der Erweiterungsmethode `HocrElements`, welche aus einer Aufzählung von HTML-Knoten nur die hOCR-Elemente zurückgibt:

```
public static IEnumerable<HocrElement> HocrElements(
    this IEnumerable<HtmlNode> nodes)
    => nodes
       .Where(n => n.IsHocrElement())
       .Select(n => n.GetHocrElement());
```

Listing 5.20: C# – Erweiterungsmethode `HocrElements`

```
public static IEnumerable<T> HocrElements<T>(
    this IEnumerable<HtmlNode> nodes)
    where T : HocrElement
    => nodes
       .HocrElements()
       .OfType<T>();
```

Listing 5.22: C# – Erweiterungsmethode `HocrElements<T>`

5.1.2 hOCR-Properties

Die Basisklasse **HocrProperty** dient als abstrakte Basisklasse für jede hOCR-Property. Alle spezifizierten Properties werden durch jeweils eine Implementationsklasse abgebildet, welche von dieser abstrakten Basisklasse erben. (s. Anh. 8.1.2) Die Klasse **BBox** repräsentiert z. B. die Property `bbox`, wobei über Klasseneigenschaften die vier Koordinatenwerte `x0`, `y0`, `x1` und `y1` abgefragt und geändert werden können:

```
public int X0 { get; set; }
public int Y0 { get; set; }
public int X1 { get; set; }
public int Y1 { get; set; }
```

Listing 5.24: C# – Klasse `BBox` – Properties

Die Klasse **HocrPropertyCollection** repräsentiert eine Liste von hOCR-Properties. Sie implementiert das Interface `IList<HocrProperty>`, sodass Methoden wie `Clear()` oder `RemoveAt(int)` und Eigenschaften wie `Count` verfügbar sind.

5.1.2.1 Tokenization

Ein **Token** ist gemäß der der Regel `property-value` in der allgemeinen Property-Grammatik entweder ein `ascii-word` oder ein `delimited-string`. Eine Anmerkung weist darauf hin, dass die individuellen Properties jeweils über eigene, spezielle Grammatiken verfügen, welche die allgemeine Regel `property-value` „überschreiben“. Bei der Property `bbox` ist das z. B. die Regel `uint uint uint uint` für die vier nichtnegativen Zahlen, bei `image` ist es `delimited-string` für den Verweis auf eine Bilddatei. Wichtig, jedoch nicht vom Standard erwähnt ist dabei die Tatsache, dass die allgemeine Regel `property-value` durch das „Überschreiben“ *nicht* ungültig wird, sondern nur spezialisiert wird. Das gilt zumindest für alle spezifizierten Properties; ob Erweiterungs-Properties die allgemeine Regel durch „Überschreiben“ entwerten können, ist nicht bekannt. Das Interface **IToken** (s. Abschnitt 5.1.2.1) modelliert einen Token:

```
public interface IToken
{
    string Value { get; set; }
}
```

Listing 5.26: C# – Interface `IToken`

Der Strukturdatentyp **AsciiWord** implementiert die ABNF-Regel `ascii-word`. Der Typ **DelimitedString** implementiert die Regel `delimited-string`.

Ein unter .NET geläufiges Schema für die Serialisierung und Deserialisierung wird mittels der folgenden beiden Methoden umgesetzt

- Die Methode **Parse** ist unter .NET für alle primitiven Datentypen vorhanden. Der statischen Methode wird eine Zeichenfolge (engl. `string`) zum *Parsen* übergeben, woraufhin eine Instanz des Datentyps mit entsprechendem Wert erzeugt und zurückgegeben wird.
- Die Methode **ToString** ist unter .NET für jedes Objekt verfügbar. Alle Klassen erben von der Basisklasse `Object` und können die geerbte Methode überschreiben.

5.1.2.2 Serialisierung

Bei der **Serialisierung** von Daten wird aus einem Objektgraphen eine (serielle) Folge von Bytes oder Zeichen generiert.

```
public void SerializeProperties()
{
    if (Properties != null)
        Html.SetAttributeValue("title", Properties.ToString());
}
```

Listing 5.28: C# – Klasse `HocrElement` – Methode `SerializeProperties`

```
public override string ToString() =>
    string.Join("; ", Properties);
```

Listing 5.30: C# – Klasse `HocrPropertyCollection` – Methode `ToString`

```
public override string ToString() =>
    $"{Name} {string.Join(" ", ValueTokens)}";
```

Listing 5.32: C# – Klasse `HocrProperty` – Methode `ToString`

5.1.2.3 Deserialisierung

Bei der **Deserialisierung** wird aus einer *seriellen* Byte- oder Zeichenfolge ein Objektgraph rekonstruiert.

```
1 public void DeserializeProperties() =>
2     Properties = HocrPropertyCollection.Parse(
3         Html.GetAttributeValue("title", null));
```

Listing 5.34: C# – Klasse `HocrElement` – Methode `DeserializeProperties`

```
1  public static HocrPropertyCollection Parse(string s)
2  {
3      if (s == null)
4          return null;
5
6      var properties = new HocrPropertyCollection();
7      using (var r = new StringReader(s))
8      {
9          while (r.Peek() != -1)
10         {
11             var property = HocrProperty.Parse(r);
12             properties.Add(property);
13         }
14     }
15     return properties;
16 }
```

Listing 5.36: C# – Klasse `HocrPropertyCollection` – Methode `Parse`

```
public static HocrProperty Parse(TextReader reader)
{
    // name
    string name = AsciiWord.Parse(reader);

    // tokens
    var tokens = new List<IToken>();
    while (reader.Peek() != -1 && reader.Peek() != ';')
    {
        reader.ReadAny(' ');

        // token
        IToken token;
        if (reader.Peek() == DelimitedString.Delimiter)
            token = DelimitedString.Parse(reader);
        else
            token = AsciiWord.Parse(reader);

        tokens.Add(token);
    }
    reader.ReadAny(';', ' ');

    var result  = Factory.Create(name);
    result.Name = name;
    result.ValueTokens = tokens;
    return result;
}
```

Listing 5.38: C# – Klasse `HocrProperty` – Methode `Parse`

Die folgende Tabelle gibt die Sequenz der Methodenaufrufe – jeweils für Serialisierung und Deserialisierung – wieder:

Klasse / Interface	**Serialisierung**	**Deserialisierung**
`HocrElement`	`SerializeProperties()`	`DeserializeProperties()`
`HocrPropertyCollection`	`ToString()`	`Parse(string)`
`HocrProperty`	`ToString()`	`Parse(TextReader)`
`IToken`	`ToString()`	`Parse(TextReader)`

Tabelle 5.1: Serialisierung / Deserialisierung

5.2 hOCR-Editor

Das Assembly **hOCR.Editor** stellt das Software-Frontend der Editor-Anwendung dar. Die folgende Abbildung zeigt den die grafische Oberfläche als visuellen Prototypen:

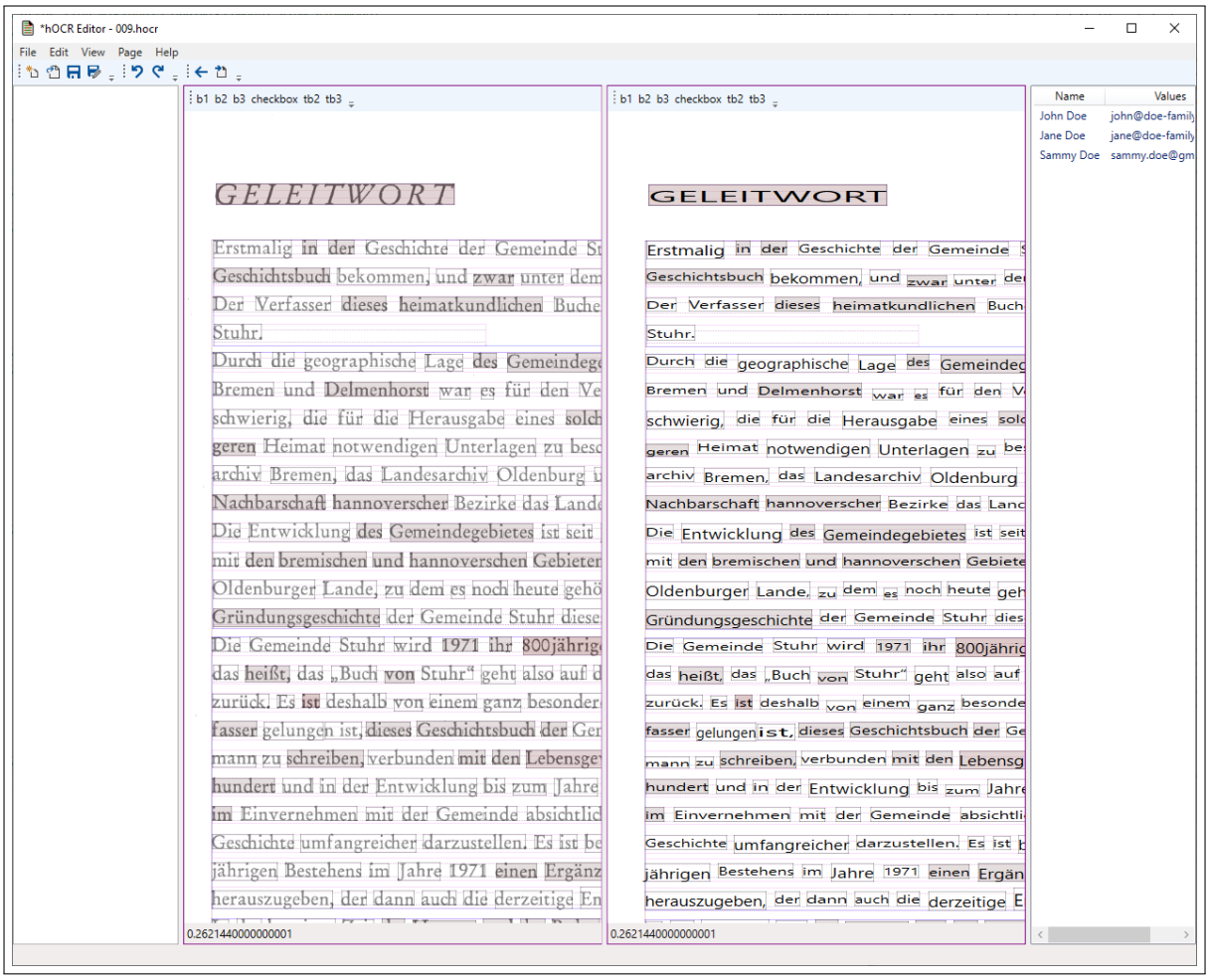


Abbildung 5.1: hOCR Editor – Visueller Prototyp

5.3 hOCR-Engines

Die **hOCR-Engines** werden durch das Assembly `hOCR.Engines` angebunden.

Das Interface **IHocrEngine** (s. Abschnitt 5.3) definiert eine einheitliche Schnittstelle für den Zugriff auf hOCR-Engines. Dabei gibt die Methode `GetHocr` ein hOCR-Dokument zurück, das anhand der hOCR-Engine für eine Bilddatei erzeugt wird. Die Sprache und das Schriftsystem des zu erkennenden Textes können durch den Parameter `cultureInfo` bestimmt werden. Die Standardeinstellung soll dabei Amerikanisches

Englisch mit modernem englischem Alphabet sein, was als Gebietsschema (engl. *Locale*) gewöhnlicherweise durch den Code `en-US` beschrieben wird. [70]

```
1  public interface IHocrEngine
2  {
3      string Name { get; }
4
5      HtmlDocument GetHocr(
6          string imageFilename, CultureInfo cultureInfo = null);
7  }
```

Listing 5.40: C# – Interface `IHocrEngine`

Die Klasse **CultureInfo** dient unter .NET der Lokalisierung von Programmen.[1] Ein Klassenobjekt enthält u.a. Informationen zu Sprache und Schriftsystem, welche mithilfe der internationalen Standards ISO 639 und ISO 15924 kodiert werden. Die Klasse eignet sich daher für einen vereinheitlichten Übergabemechanismus und wird in dieser Arbeit dafür wiederverwendet. [70]

Die Klasse **HocrEngineManager** (s. Abschnitt 5.3) wurde als Singleton realisiert, um als zentrale Sammelstelle für alle hOCR-Engines, die zur Laufzeit der Anwendung zur Verfügung stehen, zu dienen. Mithilfe der Listeneigenschaft `Engines` können die Engines abgefragt werden.

```
1
2  public ReadOnlyCollection<IHocrEngine> Engines { get; }
3  public IHocrEngine DefaultEngine { get; set; }
4
5  private HocrEngineManager()
6  {
7      Engines = new
           ReadOnlyCollection<IHocrEngine>(GetEngines().ToList());
8      DefaultEngine = Engines.FirstOrDefault();
9  }
10
11 private IEnumerable<IHocrEngine> GetEngines()
12 {
13     yield return TesseractHocrEngine.Instance;
14     yield return CuneiformHocrEngine.Instance;
15 }
```

Listing 5.42: C# – Klasse `HocrEngineManager`

Die *hartkodierten* Einträge umfassen die Engines Tesseract und Cuneiform, wobei die erstere als Standardeinstellung festgelegt wird. In weiterführenden Arbeiten könnte eine

[1] https://docs.microsoft.com/en-us/dotnet/api/system.globalization.cultureinfo?view=netcore-3.1

Plugin-Architektur zur *dynamischen* Erkennung und Aufzählung von hOCR-Engines während Laufzeit umgesetzt werden. Die Community-Dokumentation für .NET enthält bereits eine Anleitung[2] zum Erstellen einer .NET-Core-Anwendung mit Plugins. [71]

5.3.1 Tesseract

Mit **Tesseract**[3], einer von Charles Weld entwickelten Wrapper-Bibliothek, kann die Version 3.05.02 der gleichnamigen OCR-Engine in .NET eingebunden werden. Die Bibliothek wird als freie Software unter der Apache-Lizenz quelloffen auf GitHub entwickelt und als NuGet-Paket[4] veröffentlicht. [72], [73]

5.3.2 Cuneiform

Mit **Puma.NET**[5] existiert zwar eine Wrapper-Bibliothek für das .NET Framework, jedoch können damit keine hOCR-Daten gewonnen werden. Puma.NET wurde bis 2010 als Beta-Version von Maxim Saplin entwickelt und quelloffen unter der BSD-Lizenz auf der von Microsoft betriebenen Entwicklungsplattform CodePlex veröffentlicht. Seitdem der Betrieb der Plattform im Dezember 2017 vollständig eingestellt wurde, befindet sich das Projekt im CodePlex Archive. [74], [75]

Mit **Cuneiform for Linux** kann ein plattformübergreifender Wrapper realisiert werden. Das Programm kann nativ unter Linux und mithilfe der Kompatibilitätsschicht WSL (Windows Subsystem for Linux)[6] auch unter Windows ausgeführt werden. [76]

5.4 Softwaretests

Die **Softwaretests**, welche in dieser Arbeit entwickelt wurden, befinden sich in Assemblies, deren Namen bzw. Namespace mit `hOCR.Tests` beginnen.

5.4.1 YAML-Definitionen

Mit **YamlDotNet**[7] können YAML-Daten deserialisiert werden. Die Software wird auf GitHub entwickelt, steht unter der MIT-Lizenz und ist als NuGet-Paket verfügbar.[8] [77]

[2] https://docs.microsoft.com/en-us/dotnet/core/tutorials/creating-app-with-plugin-support
[3] https://github.com/charlesw/tesseract
[4] https://www.nuget.org/packages/Tesseract/
[5] https://pumanet.codeplex.com
[6] https://docs.microsoft.com/de-de/windows/wsl/about
[7] https://github.com/aaubry/YamlDotNet
[8] https://www.nuget.org/packages/YamlDotNet/

Die **Deserialisierung** in einen Objekt-Graphen[9] ist ein Verfahren zur Auswertung von YAML-Daten, das im GitHub-Wiki erläutert wird. Bei der Deserialisierung werden die Inhalte eines YAML-Dokuments durch Klassen abgebildet, wobei die Klassennamen und die Namen der Klassenmember üblicherweise so lauten wie die entsprechenden YAML-Knoten. [78]

Eine **imperative** Deserialisierung ist vonnöten, weil die von YamlDotNet gebotene Funktionalität nicht ausreicht, um `defs.yml` deklarativ bzw. „automatisch“ deserialisieren zu lassen.

Die YAML-Definitionen können über die folgende URL direkt heruntergeladen werden:

https://github.com/kba/hocr-spec/raw/master/1.2/defs.yml

Der **Projektkonfigurationsdatei** werden die folgenden Zeilen angehängt, damit das YAML-Dokument beim Erstellen der Bibliothek immer in das Ausgabeverzeichnis kopiert wird, sodass es sich bei der Ausführung einer Anwendung, welche die Bibliothek verwendet, unmittelbar im Arbeitsverzeichnis befindet:

```
<ItemGroup>
  <None Update="defs.yml">
    <CopyToOutputDirectory>PreserveNewest</CopyToOutputDirectory>
  </None>
</ItemGroup>
```

Das Assembly **hOCR.YamlDefinitions** enthält die Klasse **HocrYamlDefinitions**.

Drei **Eigenschaften** enthalten die Namen der Elemente, Properties und Metadaten:

```
    public List<string> Elements { get; }
    public List<string> Properties { get; }
    public List<string> Metadata { get; }
```

Die **YAML-Wurzel** wird zunächst im Konstruktor aus `defs.yml` geladen:

```
private HocrYamlDefinitions()
{
  var yaml = new YamlStream();
  using (var text = File.OpenText("defs.yml"))
      yaml.Load(text);
  var doc = yaml.Documents.Single();
  var root = (YamlMappingNode)doc.RootNode;
  ...
```

Daraufhin erzeugt der Konstruktor anhanddessen die Listen der Namen:

[9] https://github.com/aaubry/YamlDotNet/wiki/Samples.DeserializeObjectGraph

```
  ...
  Elements   = GetYamlKeys(root, "element").ToList();
  Properties = GetYamlKeys(root, "property").ToList();
  Metadata   = GetYamlKeys(root, "metadata").ToList();
}

private IEnumerable<string> GetYamlKeys(YamlMappingNode root, string name) =>
    (root[name] as YamlMappingNode).Select(n => n.Key.ToString());
```

Das **Singleton-Pattern** wird verwendet, weil nur *ein* YAML-Dokument mit Definitionen existiert und damit der Zugriff auf `HocrYamlDefinitions` erleichtert wird:

```
public sealed class HocrYamlDefinitions
{
    public static HocrYamlDefinitions Instance { get; } = new HocrYamlDefinitions();
    ...
}
```

Der Datensatz **Google 1000 Books** dient der Prüfung der Integrität:

```
private GoogleBooks Books { get; } =
    new GoogleBooks("replace-me/");
private HocrReserializer Reserializer { get; } =
    HocrReserializer.Instance;

private void TestVolume(int i)
{
    var bytes = Books[i].ReadHocrBytes();
    Assert.True(Reserializer.IsReserializedEqually(bytes));
}

[Fact] public void TestVolume000() => TestVolume(0);
[Fact] public void TestVolume001() => TestVolume(1);
[Fact] public void TestVolume001() => TestVolume(2);
...
[Fact] public void TestVolume999() => TestVolume(999);
```

Listing 5.44: C# – Testklasse `GoogleBooksTest`

6 Evaluation

Die Auswertung des entwickelten Software-Systems geschieht durch das Betrachten der realisierten Ergebnisse unter Berücksichtigung der gestellten Anforderungen. Die vorliegende Arbeit konnte diese Anforderungen erfüllen und die Lösung im anfangs vorgestellten Problemfeld bereichern.

Die **YAML**-Definitionen des Standards werden durch einen Softwaretest mithilfe der Bibliothek YamlDotNet auswertet und mit den verfügbaren Implementationsklassen abgeglichen, sodass die Vollständigkeit der Implementation sichergestellt ist. Die hOCR-Bibliothek, die in dieser Arbeit entwickelt wurde, umfasst somit eine Implementation aller Properties und Elemente, die vom hOCR-Standard spezifiziert werden.

Die **Integrität** der hOCR-Dokumente wird durch Softwaretests sichergestellt: Wird ein Dokument mithilfe der Bibliothek geladen und anschließend wieder abgespeichert, so verändert es sich nicht. Dieser Test wurde mit dem 119 GB großen Datensatz Google 1000 Books, welcher im Rahmen der ICDAR 2007 veröffentlicht wurde. Der hOCR-Standard und das OCR-System OCRopus – beides Werke von Thomas Breuel – sind z. B. durch dieselbe wissenschaftliche Konferenz veröffentlicht worden.

Die **Manipulation** der hOCR-Dokumente wird ebenfalls anhand eines Softwaretests sichergestellt. Dabei wird ein hOCR-Dokument als Bytefolge aus dem Dateisystem in die Laufzeit geladen, darin verändert und schließlich wieder in eine Bytefolge gespeichert. Die Veränderung der Bytefolge dient als Beleg dafür, dass sich das hOCR-Dokument tatsächlich verändert hat.

Die **OCR-Engines** Tesseract und Cuneiform können mithilfe der Entwicklungen dieser Arbeit angebunden werden. Beide OCR-Engines unterstützen hOCR als Datenformt bei der Ausgabe von OCR-Ergebnissen. Bisher war keine Lösung bekannt, welche die Anbindung von Cuneiform zur Ausgabe von hOCR-Daten ermöglichte. In dieser Arbeit wurde dies durch einen Wrapper mithilfe des WSL (Windows Subsystem for Linux) bewerkstelligt.

Der **hOCR-Editor**, die grafische Benutzeroberfläche also, verbindet die Bedienkonzepte verschiedener Anwendungen, die bereits für das Format hOCR bestehen. Neben dieser Inspiration wurden im Rahmen dieser Arbeit auch Neuerungen entwickelt, die nicht in ähnlicher Form bei bestehenden Lösungen im Rahmen der Recherche entdeckt werden konnten. Ein Beispiel ist die Nutzung von Farbtönen zur Visualisierung der Erkennungssicherheit. Bedienelemente wie z. B. die Baumansicht für die hierarchische Struktur des hOCR-Dokuments oder die zweispaltige Rasteransicht für die hOCR-Properties werden in dieser Arbeit erstmals im Zusammenhang mit hOCR eingesetzt.

7 Fazit

Die Entwicklung eines hOCR-Editors zur Bearbeitung von OCR-Ergebnissen konnte im Rahmen der vorliegenden Arbeit prototypisch und exemplarisch realisiert werden.

Damit wird eine **Marktlücke** gefüllt: Bisher ist noch kein ausgereifter hOCR-Editor erschienen, die hOCR als natives Datenformat unterstützt. Die in dieser Arbeit entwickelte Anwendung kombiniert mehrere Bedienkonzepte, sodass *alle* Aspekte eines hOCR-Dokuments bearbeitet werden können und bestenfalls auch anschaulich visualisiert werden. Das Bedienkonzept der zeilenweisen Korrektur wurde integriert; gleichzeitig können aber auch die Bounding-Boxes verschieben werden. Alle anderen untersuchten Anwendungen konzentrieren sich meist auf Aspekte wie z. B. nur auf das Korrekturlesen.

Die **Modularität** der Anwendung erleichtert die Wiederverwendung von Komponenten, die in dieser Arbeit verwendet wurden. Die hOCR-Bibliothek – das Software-Backend – funktioniert z. B. unabhängig von jeglicher Benutzerinteraktion und könnte in weiterführenden Arbeiten bspw. in einem Web-Dienst wiederverwendet werden, der mittels ASP.NET Core realisiert wird. Eine OCR-Engine wird modular durch jeweils ein Assembly repräsentiert, das mindestens eine Klasse enthält, welche eine geteilte Schnittstelle implementiert. In weiterführenden Arbeiten könnte daraus eine „echte“ Plugin-Architektur entwickelt werden, sodass Assemblies allein dadurch, dass sie sich im Arbeitsverzeichnis befinden, erkannt und eingebunden werden.

Die **Nativität** des Datenformats hOCR erhält ein besonderes Augenmerk und wird anhand mehrerer Modultests sichergestellt, die wiederum sowohl die Integrität als auch gezielte Manipulation von hOCR-Dokumenten prüfen. Das bedeutet für den Benutzer, dass sich hOCR-Dokumente, welche geöffnet und direkt wieder gespeichert werden, unverändert (integer) bleiben. Anderseits sind nur ganz bestimmte Änderungen in einem hOCR-Dokument zu erwarten, wenn dieses geöffnet wird, dann nur ganz bestimmte Informationen verändert werden und das Dokument schließlich wieder gespeichert wird. Wenn z. B. nur die Koordinaten einer Bounding-Box verändert werden, verändert sich nur eine entsprechende Zeile, was z. B. die Verwendung einer textbasierten Versionskontrolle erleichtert. Bearbeitet man hingegen mit einem Programm eine Datei, welche nicht im nativen Datenformat der Anwendung vorliegt, ist in dem Sinne ein Import bzw. ein Export von Daten vonnöten, was wiederum Datenverlust durch die Datentransformation (Generationsverlust) bedeuten kann.

Die vorliegende Arbeit soll eine Grundlage für weiterführende Arbeiten bieten. Die modulare Architektur sollte dabei die Ausarbeitung bestimmter Aspekte und Komponenten erleichtern.

Anhang

8 Dokumentanhänge

8.1 Standard hOCR

8.1.1 Elemente

Der hOCR-Standard spezifiziert insgesamt vierzig Elemente und verteilt diese in folgende fünf Kategorien, wobei ein Element mehreren Kategorien angehören kann: [7, § 3]

- **Typesetting**[1] (dt. *das Setzen*) sind alle Elemente, die sich auf den Textsatz (engl. *typesetting*) beziehen, wie bspw. ocr_page oder ocr_line. Die Definitionen der Elemente orientieren sich dabei an den Modellen von Textsatz-Systemen wie LaTeX, LibreOffice oder Microsoft Word.
- **Floating**[2] (dt. *gleiten*) sind Elemente wie Fuß- und Kopfnoten, Seitenzahlen, Abbildungen oder Tabellen. Sie sind aus dem Fließtext ausgelagert und *gleiten* somit zwischen den Absätzen. Im Deutschen bezeichnet man solche Bestandteile z. B. als „Gleitobjekte“. [79]
- **Logical**[3] (dt. *logisch*) sind Elemente, die eine logische Bedeutung haben. Durch ocr_chapter, ocr_section und ocr_subsection kann ein Dokument z. B. logisch in Kapitel, Abschnitte und Unterabschnitte unterteilt werden.
- **Inline**[4] (dt. *innerhalb einer Zeile*) sind alle Elemente, die im Gegensatz zu den „floating elements“ ein Bestandteil des Fließtextes sind. Dazu zählen z. B. Wörter und nicht näher bestimmte Zeichen (Glyphen) bzw. ocrx_word und ocr_glyph.
- **OCR-Engine-Specific**[5] sind Elemente, deren genaue Bedeutung von der *spezifischen* OCR-Engine abhängt. Ein ocrx_word zeichnet z. B. ein Wort aus, jedoch ist die Definition, was ein Wort ist, Sache der OCR-Engine. Der Name dieser Elemente beginnt mit `ocrx_`.

Allen Elementen wird in der folgenden Tabelle 8.1 jeweils eine C#-Klasse zugordnet, die im Rahmen dieser Arbeit zur Modellierung entwickelt wurde. Die Zugehörigkeit zu den Kategorien wird durch Anfangsbuchstaben der Kategorienamen kenntlich gemacht:

[1] http://kba.cloud/hocr-spec/1.2/#sec-typesetting-elements
[2] http://kba.cloud/hocr-spec/1.2/#sec-float-elements
[3] http://kba.cloud/hocr-spec/1.2/#sec-logical-elements
[4] http://kba.cloud/hocr-spec/1.2/#sec-inline-elements
[5] http://kba.cloud/hocr-spec/1.2/#sec-engine-elements

Name	Klasse	Kategorien				
ocr_abstract	OcrAbstract			L		
ocr_author	OcrAuthor			L		
ocr_blockquote	OcrBlockQuote			L		
ocr_caption	OcrCaption			L		
ocr_carea	OcrCArea	T				
ocr_chapter	OcrChapter			L		
ocr_chem	OcrChem		F			
ocr_cinfo	OcrCInfo				I	
ocr_column	OcrColumn	T				
ocr_display	OcrDisplay		F			
ocr_document	OcrDocument			L		
ocr_dropcap	OcrDropCap				I	
ocr_float	OcrFloat		F			
ocr_footer	OcrFooter		F			
ocr_glyph	OcrGlyph				I	
ocr_glyphs	OcrGlyphs				I	
ocr_header	OcrHeader		F			
ocr_image	OcrImage		F			
ocr_line	OcrLine	T				
ocr_linear	OcrLinear	T				
ocr_linedrawing	OcrLineDrawing		F			
ocr_math	OcrMath		F			
ocr_noise	OcrNoise				I	
ocr_page	OcrPage	T				
ocr_pageno	OcrPageNo		F			
ocr_par	OcrPar			L		
ocr_part	OcrPart			L		
ocr_photo	OcrPhoto		F			
ocr_section	OcrSection			L		
ocr_separator	OcrSeparator	T	F			
ocr_subsection	OcrSubsection			L		
ocr_subsubsection	OcrSubsubsection			L		
ocr_table	OcrTable		F			
ocr_textfloat	OcrTextFloat		F			
ocr_textimage	OcrTextImage		F			
ocr_title	OcrTitle			L		
ocr_xycut	OcrXYCut				I	
ocrx_block	OcrxBlock				I	O
ocrx_line	OcrxLine				I	O
ocrx_word	OcrxWord				I	O

Tabelle 8.1: hOCR-Elemente

8.1.2 Properties

Es werden insgesamt 21 Properties vom hOCR-Standard definiert, wobei die Zeichenfolgen zur Darstellung der Werte jeweils mithilfe der ABNF beschrieben wird. Die folgende Tabelle 8.2 ordnet den Properties ihre ABNF-Grammatik und eine entsprechende C#-Klasse zu, die im Rahmen dieser Arbeit zur Modellierung entwickelt wurde. [7, § 3]

Name	Klasse	Grammatik
baseline	Baseline	`float int`
bbox	BBox	`uint uint uint uint`
cflow	CFlow	`delimited-string`
cuts	Cuts	`+(uint *1(comma uint *1(comma nint)))`
hardbreak	HardBreak	`"0/ "1"`
image	Image	`delimited-string`
imagemd5	ImageMD5	`doublequote 32(%x41-46 / digit) doublequote`
lpageno	LPageNo	`delimited-string / uint`
ppageno	PPageNo	`uint`
nlp	NLP	`+float`
order	Order	`+uint`
poly	Poly	`2uint 2int *(2int)`
scan_res	ScanRes	`2(uint)`
textangle	TextAngle	`float`
x_bboxes	XBBoxes	`1*(4uint)`
x_font	XFont	`delimited-string`
x_fsize	XFSize	`uint`
x_confs	XConfs	`+float`
x_scanner	XScanner	`delimited-string`
x_source	XSource	`1*delimited-string`
x_wconf	XWConf	`float`
Erweiterungen		
x_ascenders	XAscenders	`float`
x_descenders	XDescenders	`float`
x_size	XSize	`float`

Tabelle 8.2: hOCR-Properties

Die drei Erweiterungen entstammen der OCR-Engine Tesseract. Sie kodieren gemäß Liniensystem aus der Typografie die Oberlänge (engl. *ascender*), die Unterlänge (engl. *descender*) und die Mittellänge, die auch als x-Höhe (engl *x-size*) bezeichnet wird. [80]

8.1.3 Properties-Grammatik

Die speziellen Grammatiken in Tabelle 8.2 greifen auf die allgemeine Grammatik[6] zurück, welche durch das folgende Listing 8.1 vollständig wiedergegeben wird. [7, § 2.4]

```
1  digit            =    %x30-39
2  uint             =     +digit
3  int              = *1"-" uint
4  nint             =   "-" uint
5  fraction         =   "." uint
6  float            =      *uint fraction
7
8  whitespace       = +%x20  ; one or more spaces ' '
9  comma            =  %x2C  ; comma ','
10 semicolon        =  %x3B  ; semicolon ';'
11 doublequote      =  %x22  ; double quote '"'
12 lowercase-letter =  %x41-5A
13 alnum-word       = +(lowercase-letter / digit)
14 ascii-word       = +(%x21-7E - semicolon)    ; printable w/o space/semicolon
15 ascii-string     = +(%x20-7E - doublequote)  ; printable ascii without doublequote
16 delimited-string =   doublequote ascii-string doublequote
17
18 properties-format = key-value-pair *(*whitespace semicolon *whitespace
19                     key-value-pair)
20 spec-property-name = ("bbox" / "baseline" / "cflow" / "cuts" / "hardbreak" /
21                       "image" / "imagemd5" / "lpageno" / "nlp" / "order" /
22                       "poly" / "ppageno" / "scan_res" / "textangle" /
23                       "x_bboxes" / "x_confs" / "x_font" / "x_fsize" /
24                       "x_scanner" / "x_source" / "x_wconf" )
25 engine-property-name = "x_" alnum-word
26 key-value-pair = property-name whitespace property-value
27 property-name  = spec-property-name / engine-property-name
28 property-value = (ascii-word / delimited-string) *(whitespace
29                  (ascii-word / delimited-string) )
```

Listing 8.1: ABNF – Grammatik für hOCR-Properties (allgemein)

Mithilfe der Regel `properties-format` (Z. 18) beschreibt die allgemeine Grammatik darüber hinaus auch, wie die Zeichenfolgen mehrerer Properties in einer Zeichenfolge zusammengefasst werden können.

8.2 TextReader-Erweiterungsmethoden

Die Klasse **TextReader** wird mit vier Erweiterungsmethoden versehen, welche das Parsen von Zeichenfolgen im Rahmen der Deserialisierung von Properties erleichtern sollen:

[6] http://kba.cloud/hocr-spec/1.2/#grammar

8.2.1 Erweiterungsmethode Read

Mit **Read** wird das nächste Zeichen gelesen. Die Methode hat keinen Rückgabewert und dient nur als Assertion (dt. *Zusicherung*): Bei einem Zeichen, welches nicht erwartet (engl. `expected`) wurde, wird eine `FormatException` ausgelöst.

```
public static void Read(
    this TextReader r, char c)
{
    int i = r.Read();
    if (i == -1 || i != c)
        throw new FormatException(
            $"read: '{(char)i}' expected: '{c}'");
}
```

Listing 8.2: C# – Erweiterungsmethode `Read(char)`

8.2.2 Erweiterungsmethode ReadUntil

Mit **ReadUntil** werden solange Zeichen gelesen, bis (engl. *until*) eine bestimmte Bedingung in Bezug auf das vorangehende Zeichen eintritt: Diese Bedingung kann mit `nextCharPredicate` als logisches Prädikat (engl. *predicate*) direkt übergeben werden und es wird dann solange gelesen, bis dieses Prädikat für das nächste Zeichen wahr ist:

```
public static string ReadUntil(
    this TextReader r, Predicate<char> nextCharPredicate)
{
    var sb = new StringBuilder();

    char next;
    while (r.Peek() != -1 && !nextCharPredicate(next = (char)r.Peek()))
    {
        sb.Append(next);
        r.Read();
    }

    return sb.ToString();
}
```

Listing 8.4: C# – Erweiterungsmethode `ReadUntil(Predicate<char>)`

Alternativ kann mit `chars` eine Menge von Zeichen übergeben werden, die allesamt *nicht* gelesen werden sollen, sodass nur solange gelesen wird, bis das nächste Zeichen, das gelesen *würde*, in der Menge enthalten ist.

```
1 public static string ReadUntil(
2     this TextReader r, params char[] chars)
3     => r.ReadUntil(c => chars.Contains(c));
```

Listing 8.6: C# – Erweiterungsmethode `ReadUntil(params char[])`

8.2.3 Erweiterungsmethode ReadAny

Mit **ReadAny** wird solange gelesen, bis das nächste Zeichen nicht mehr in der übergebenen Zeichenmenge enthalten ist. Die gelesenen Zeichen werden dann als Zeichenfolge zurückgegeben. Verglichen mit dem Prädikat in Abschnitt 8.2.2, ist das Prädikat negiert, weil die übergebenen Zeichen hier durchaus gelesen werden:

```
1 public static string ReadAny(
2     this TextReader r, params char[] chars)
3     => r.ReadUntil(c => !chars.Contains(c));
```

Listing 8.8: C# – Erweiterungsmethode `ReadAny(params char[])`

Verzeichnisse

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Literaturverzeichnis

Publikationen

[1] L. Vincent, „Google Book Search: Document Understanding on a Massive Scale“, in *9th International Conference on Document Analysis and Recognition*, (23.–26. Sep. 2007), Bd. 2, IEEE Computer Society, 2007, S. 819–823, ISBN: 978-0-7695-2822-9. DOI: 10.1109/ICDAR.2007.4377029.

[2] T. M. Breuel, „The hOCR Microformat for OCR Workflow and Results“, in *9th International Conference on Document Analysis and Recognition*, (23.–26. Sep. 2007), Bd. 2, IEEE Computer Society, 2007, S. 1063–1067, ISBN: 1520-5363, 0-7695-2822-8. DOI: 10.1109/ICDAR.2007.4377078.

[3] T. M. Breuel, „The OCRopus open source OCR system“, in *Document Recognition and Retrieval XV*, B. A. Yanikoglu und K. Berkner, Hrsg., International Society for Optics und Photonics, Bd. 6815, SPIE, 2008, S. 120–134. DOI: 10.1117/12.783598.

[4] C. M. Schulze, *Was sind Mikroformate und wozu braucht man sie?* De Gruyter Saur, 27. Sep. 2010, S. 129–142, ISBN: 978-3-11-023210-3. DOI: 10.1515/9783110232103.

[8] D. Crocker und P. Overell, „Augmented BNF for Syntax Specifications: ABNF“, Techn. Ber., Jan. 2008, Internet Standard. Adresse: https://tools.ietf.org/html/rfc5234.

[9] O. Ben-Kiki, C. Evans und I. döt Net, „YAML Ain't Markup Language (YAML™) Version 1.2“, Techn. Ber., 1. Okt. 2009. Adresse: https://yaml.org/spec/1.2/spec.html (besucht am 01. 04. 2020).

[13] S. Pletschacher und A. Antonacopoulos, „The PAGE (Page Analysis and Ground-Truth Elements) Format Framework“, in *20th International Conference on Pattern Recognition*, (23.–26. Aug. 2010), IEEE Computer Society, 2010, S. 257–260, ISBN: 978-0-7695-4109-9. DOI: 10.1109/ICPR.2010.72.

[15] T. M. Breuel, „Recent Progress on the OCRopus OCR System“, in *Proceedings of the International Workshop on Multilingual OCR*, Ser. MOCR '09, Barcelona, Spain: ACM, 2009, 2:1–2:10, ISBN: 978-1-60558-698-4. DOI: 10.1145/1577802.1577805.

[16] R. Smith, „An Overview of the Tesseract OCR Engine“, in *9th International Conference on Document Analysis and Recognition*, (23.–26. Sep. 2007), Bd. 2, IEEE Computer Society, 2007, S. 629–633, ISBN: 0-7695-2822-8. DOI: 10.5555/1304596.1304846.

[40] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, Nov. 2002, ISBN: 978-321-12742-6. Adresse: https://martinfowler.com/books/eaa.html.

[42] ECMA International, *Standard ECMA-335 - Common Language Infrastructure (CLI)*, 6. Aufl., Geneva, Switzerland: ECMA International, Juni 2012, S. 1–574. Adresse: https://ecma-international.org/publications/standards/Ecma-335.htm (besucht am 27. 10. 2019).

[43] A. Hejlsberg, M. Torgersen, S. Wiltamuth und P. Golde, *The C# Programming Language*, 4. Aufl. Upper Saddle River, NJ, USA: Addison-Wesley, Okt. 2010, ISBN: 978-0-321-74176-9. DOI: 10.5555/1951915.

[44] ECMA International, *Standard ECMA-334 – C# Language Specification*, 5. Aufl., Geneva, Switzerland: ECMA International, Dez. 2017, S. 1–516. Adresse: https://ecma-international.org/publications/standards/Ecma-334.htm (besucht am 28. 11. 2019).

[81] *9th International Conference on Document Analysis and Recognition*, (23.–26. Sep. 2007), IEEE Computer Society, 2007, ISBN: 978-0-7695-2822-9.

Online-Quellen

[5] F. Berriman, T. Çelik, D. Cederholm u. a. (1. Apr. 2020). Microformats Wiki – About, Adresse: http://microformats.org/wiki/about (besucht am 01. 04. 2020).

[6] WHATWG. (14. März 2020). HTML Standard. Living Standard, Adresse: https://html.spec.whatwg.org (besucht am 15. 03. 2020).

[7] K. Baierer und T. M. Breuel. (26. Feb. 2020). hOCR - OCR Workflow and Output embedded in HTML, Adresse: http://kba.cloud/hocr-spec/1.2/ (besucht am 14. 03. 2020).

[10] T. M. Breuel. (März 2010). The hOCR Embedded OCR Workflow and Output Format, Adresse: https://docs.google.com/View?docid=dfxcv4vc_67g844kf (besucht am 14. 03. 2020).

[11] K. Baierer und T. M. Breuel. (23. Okt. 2016). The hOCR Embedded OCR Workflow and Output Format, version 1.1, Adresse: https://github.com/kba/hocr-spec/blob/master/1.1/spec.md (besucht am 14. 03. 2020).

[12] Library of Congress. (7. Mai 2019). ALTO: Technical Metadata for Layout and Text Objects, Adresse: https://www.loc.gov/standards/alto/ (besucht am 15. 03. 2020).

[14] TEI-Konsortium. (13. Feb. 2020). P5: Guidelines for Electronic Text Encoding and Interchange. Historical Background, Adresse: https://tei-c.org/release/doc/tei-p5-doc/en/html/AB.html#ABTEI (besucht am 04. 04. 2020).

[17] L. Vincent. (30. Aug. 2006). Announcing Tesseract OCR, Adresse: http://googlecode.blogspot.com/2006/08/announcing-tesseract-ocr.html (besucht am 29. 12. 2019).

[18] A. Kay. (1. Juli 2007). Tesseract: an Open-Source Optical Character Recognition Engine, Adresse: https://www.linuxjournal.com/article/9676 (besucht am 29. 12. 2019).

[19] Tesseract Project. (Feb. 2011). Issue 263: patch to enable hOCR output, Adresse: http://code.google.com/p/tesseract-ocr/issues/detail?id=263 (besucht am 26. 02. 2011).

[20] Cognitive Technologies. (). OCR-Cuneiform. Support, Adresse: http://cuneiform.ru/products/index.html (besucht am 04. 05. 2010).

[21] Cognitive Technologies. (2. Apr. 2008). OpenOCR – Project News (2008-04-02), Adresse: http://en.openocr.org/news/2008-04-02/3/ (besucht am 14. 07. 2014).

[22] J. Pakkanen. (19. Apr. 2011). Cuneiform for Linux, Adresse: https://launchpad.net/cuneiform-linux (besucht am 15. 03. 2020).

[23] E. Bärwaldt. (Apr. 2011). Eingescannte Texte automatisch erkennen, Adresse: https://www.linux-community.de/ausgaben/linuxuser/2011/04/eingescannte-texte-automatisch-erkennen/ (besucht am 15. 03. 2020).

[24] K. Baierer. (22. Aug. 2019). kba/hocrjs, Adresse: https://github.com/kba/hocrjs (besucht am 03. 04. 2020).

[25] M. Plömer. (23. Jan. 2017). not-implemented/hocr-proofreader, Adresse: https://github.com/not-implemented/hocr-proofreader (besucht am 03. 04. 2020).

[26] J. Garrison. (19. Juni 2016). moz-hocr-edit, Adresse: https://jimgarrison.org/moz-hocr-edit/ (besucht am 03. 04. 2020).

[27] J. Garrison. (9. Apr. 2015). garrison/moz-hocr-edit, Adresse: https://github.com/garrison/moz-hocr-edit (besucht am 03. 04. 2020).

[28] J. Garrison. (26. Jan. 2016). end of development, Adresse: https://groups.google.com/forum/#!topic/moz-hocr-edit/ijtaEAznOUs (besucht am 03. 04. 2020).

[29] C. DiBona. (12. März 2015). Bidding farewell to Google Code. Google Open Source Blog, Adresse: https://opensource.googleblog.com/2015/03/farewell-to-google-code.html (besucht am 04. 04. 2020).

[30] T. M. Breuel. (13. Aug. 2019). hocr-tools, Adresse: https://github.com/tmbdev/hocr-tools (besucht am 04. 04. 2020).

[31] T. M. Breuel. (2007-04-02). hocr-tools, Adresse: http://code.google.com/p/hocr-tools/ (besucht am 20. 04. 2007).

[32] Universitätsbibliothek Mannheim. (9. Jan. 2020). ocr-fileformat, Adresse: https://github.com/UB-Mannheim/ocr-fileformat (besucht am 09. 03. 2020).

[33] Universitätsbibliothek Mannheim. (17. Dez. 2019). ocr-gt-tools, Adresse: https://github.com/UB-Mannheim/ocr-gt-tools (besucht am 09. 03. 2020).

[34] J. Rocha. (10. Jan. 2020). OCRFeeder, Adresse: https://wiki.gnome.org/Apps/OCRFeeder (besucht am 15. 03. 2020).

[35] J. Rocha. (27. Juni 2011). Issue #17 – Export to hOCR, Adresse: https://code.google.com/archive/p/ocrfeeder/issues/17 (besucht am 04. 04. 2020).

[36] J. Rocha. (3. Apr. 2009). My Master Thesis, Adresse: https://www.joaquimrocha.com/2009/03/04/my-master-thesis/ (besucht am 04. 04. 2020).

[37] ABBYY. (). ABBYY FineReader 15, Adresse: https://www.abbyy.com/en-eu/finereader (besucht am 15. 03. 2020).

[38] ABBYY. (16. Jan. 2020). FineReader 15 User's Guide: OCR Editor, Adresse: https://help.abbyy.com/en-us/finereader/15/user_guide/workwithocr (besucht am 04. 04. 2020).

[39] ABBYY. (). ABBYY XML Export, Adresse: https://abbyy.technology/en:features:ocr:xml (besucht am 15. 03. 2020).

[41] Microsoft. (15. März 2019). Solution (.sln) file. Visual Studio Docs, Adresse: https://docs.microsoft.com/en-us/visualstudio/extensibility/internals/solution-dot-sln-file?view=vs-2019 (besucht am 24. 03. 2020).

[45] Microsoft. (20. Sep. 2019). What's new in C# 8.0, Adresse: https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-8 (besucht am 21. 03. 2020).

[46] Microsoft. (21. Feb. 2020). C# language versioning, Adresse: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/configure-language-version (besucht am 21. 03. 2020).

[47] .NET Foundation. (). About the .NET Foundation, Adresse: https://www.dotnetfoundation.org/about (besucht am 21. 03. 2020).

[48] Microsoft. (). What is .NET Framework?, Adresse: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework (besucht am 21. 03. 2020).

[49] Mono Project. (8. Mai 2019). Documentation, Adresse: https://www.mono-project.com/docs/ (besucht am 21. 03. 2020).

[50] Mono Project. (10. Apr. 2018). C# Compiler, Adresse: https://www.mono-project.com/docs/about-mono/languages/csharp/ (besucht am 21. 03. 2020).

[51] Unity Technologies. (18. März 2020). Mono, Adresse: https://github.com/Unity-Technologies/mono (besucht am 21. 03. 2020).

[52] Microsoft. (12. Apr. 2019). .NET Core guide, Adresse: https://docs.microsoft.com/en-us/dotnet/core/ (besucht am 21. 03. 2020).

[53] .NET Foundation. (20. März 2020). .NET Core Home, Adresse: https://github.com/dotnet/core (besucht am 21. 03. 2020).

[54] R. Lander. (6. Mai 2019). Introducing .NET 5, Adresse: https://devblogs.microsoft.com/dotnet/introducing-net-5/ (besucht am 21. 03. 2020).

[55] WHATWG. (14. März 2020). DOM Standard. Living Standard, Adresse: https://dom.spec.whatwg.org (besucht am 15. 03. 2020).

[56] ZZZ Projects. (). Html Agility Pack, Adresse: https://html-agility-pack.net (besucht am 15. 03. 2020).

[57] ZZZ Projects. (11. März 2020). Html Agility Pack, Adresse: https://github.com/zzzprojects/html-agility-pack (besucht am 15. 03. 2020).

[58] NuGet. (22. März 2020). Html Agility Pack, Adresse: https://www.nuget.org/packages/HtmlAgilityPack/ (besucht am 26. 03. 2020).

[59] Microsoft. (25. Jan. 2018). Windows Presentation Foundation, Adresse: https://docs.microsoft.com/en-us/dotnet/framework/wpf/ (besucht am 21. 03. 2020).

[60] .NET Foundation. (21. März 2020). Windows Presentation Foundation (WPF), Adresse: https://github.com/dotnet/wpf (besucht am 21. 03. 2020).

[61] H. Schwichtenberg. (5. Dez. 2018). Microsoft Connect() 2018: Erste Vorschauversion auf .NET Core 3.0, Adresse: https://www.heise.de/developer/meldung/Microsoft-Connect-2018-Erste-Vorschauversion-auf-NET-Core-3-0-4239399.html (besucht am 21. 03. 2020).

[62] Microsoft. (8. Aug. 2019). XAML overview in WPF, Adresse: https://docs.microsoft.com/en-us/dotnet/desktop-wpf/fundamentals/xaml (besucht am 21. 03. 2020).

[63] Avalonia. (19. März 2020). Documentation, Adresse: https://avaloniaui.net/docs/ (besucht am 21. 03. 2020).

[64] J. Smith. (Feb. 2009). WPF Apps With The Model-View-ViewModel Design Pattern. MSDN Magazine, Adresse: https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern (besucht am 18. 04. 2020).

[65] M. Fowler. (19. Juli 2004). Presentation Model, Adresse: https://martinfowler.com/eaaDev/PresentationModel.html (besucht am 18. 04. 2020).

[66] ReactiveUI. (14. März 2020). Documentation, Adresse: https://reactiveui.net/docs/ (besucht am 21. 03. 2020).

[67] T. Moreira. (12. Nov. 2012). MVVM PRISM or ReactiveUI? Choose one or both? Thoughts about Software Engineering, Adresse: http://toughtsaboutsoftware.blogspot.com/2013/08/mvvm-prism-or-reactiveui-choose-one-or.html (besucht am 18. 04. 2020).

[68] ReactiveUI. (3. Apr. 2020). reactiveui/ReactiveU, Adresse: https://github.com/reactiveui/ReactiveUI (besucht am 18. 04. 2020).

[69] Y. Bulatov. (9. Nov. 2011). Google1000 dataset, Adresse: http://yaroslavvb.blogspot.com/2011/11/google1000-dataset_09.html (besucht am 14. 03. 2020).

[70] Microsoft. (18. März 2020). CultureInfo Class, Adresse: https://docs.microsoft.com/en-us/dotnet/api/system.globalization.cultureinfo?view=netcore-3.1 (besucht am 17. 04. 2020).

[71] Microsoft. (16. Okt. 2019). Create a .NET Core application with plugins, Adresse: https://docs.microsoft.com/en-us/dotnet/core/tutorials/creating-app-with-plugin-support (besucht am 17. 04. 2020).

[72] C. Weld. (16. Dez. 2018). A .Net wrapper fro tesseract-ocr, Adresse: https://github.com/charlesw/tesseract (besucht am 21. 03. 2020).

[73] C. Weld. (16. Dez. 2018). Tesseract 3.3.0, Adresse: https://www.nuget.org/packages/Tesseract/ (besucht am 21. 03. 2020).

[74] M. Saplin. (9. Jan. 2010). Puma.NET, Adresse: https://pumanet.codeplex.com (besucht am 15. 03. 2020).

[75] B. Harry. (31. März 2017). Shutting down CodePlex, Adresse: https://devblogs.microsoft.com/bharry/shutting-down-codeplex/ (besucht am 06. 04. 2020).

[76] Microsoft. (11. Juli 2016). Windows-Subsystem für Linux: Dokumentation, Adresse: https://docs.microsoft.com/de-de/windows/wsl/about (besucht am 06. 04. 2020).

[77] A. Aubry. (23. März 2020). aaubry/YamlDotNet, Adresse: https://github.com/aaubry/YamlDotNet (besucht am 10. 04. 2020).

[78] A. Aubry und F. Bin Hasan. (9. März 2020). aaubry/YamlDotNet, Adresse: https://github.com/aaubry/YamlDotNet/wiki/Samples (besucht am 10. 04. 2020).

[79] J. Clausen. (9. Jan. 2006). LaTeX-Praxis – Gleitobjekte und Abbildungen, Adresse: https://www.techfak.uni-bielefeld.de/~joern/edu/tex/latexpraxis/latex5-print.pdf (besucht am 07. 04. 2020).

[80] P. Zumstein. (23. Apr. 2017). Calculate font size. hocr-tools Wiki, Adresse: https://github.com/tmbdev/hocr-tools/wiki/Calculate-font-size (besucht am 08. 04. 2020).