SIEMENS
Ingenuity for life

OST
Ostschweizer
Fachhochschule

AI for Relay Interlocking

PROJEKTDOKUMENTATION STUDIENARBEIT HS21

DOMINIC KLINGER, CHRISTIAN BISIG

FARHAD MEHTA

Abstract

Ausgangslage

Ein Relaisstellwerk ist eine ältere Stellwerks-Bauform, bei welcher die Abhängigkeiten zwischen den einzelnen Komponenten wie beispielsweise Weichen, Gleissperren, Signalen oder Bahnübergangssicherungsanlagen vollständig elektrisch durch komplexe Relaisschaltungen umgesetzt werden. In der Schweiz sind immer noch viele solche Relaisstellwerke im Einsatz und es werden auch noch immer neue Stellwerke dieses Typs geplant und gebaut. Die Dokumentationen solcher Anlagen werden im PDF Format abgelegt. Dabei ist es keine Seltenheit, dass die Dokumentation einer einzelnen Anlage mehrere tausend Seiten Stromlaufpläne beinhaltet. Bei älteren Anlagen ist es zudem üblich, dass gescannte Tuschzeichnungen und moderne CAD-Zeichnungen in der gleichen Dokumentation vorhanden sind. Diese Mischform führt dazu, dass die PDF-Dokumentationen nicht digital durchsuchbar sind und ein effizientes Arbeiten nur schwer möglich ist.

Ziel

Das im Rahmen dieser SA durchgeführte Projekt wurde in Zusammenarbeit mit Siemens Mobility AG durchgeführt. Das Ziel dieser Arbeit war, mit Hilfe von Bildverarbeitungsverfahren, Machine Learning oder Artificial Intelligence, einen Weg zu finden, um Informationen aus den Stromlaufplänen zu extrahieren. Diese neu gewonnenen Informationen sollten in einem zweiten Schritt für zukünftige Analysen verfügbar gemacht werden. Zudem sollte ein Weg gefunden werden, die extrahierten Informationen so im ursprünglich analysierten Dokument zu hinterlegen, dass nach den erkannten Texten gesucht werden kann. Mit diesem Schritt soll das Arbeiten mit den Stellwerkdokumentationen effizienter gemacht werden.

Methode / Vorgehen

In einem ersten Schritt ging es darum, verschiedene Methoden und Technologien zum Detektieren von Symbolen und Texten zu evaluieren. Dabei sollte herausgefunden werden, wie die besten Ergebnisse erzielt werden können und möglichst viele Informationen korrekt aus den Plänen gelesen werden können. In dieser Phase wurden verschiedene Prototypen basierend auf Technologien wie ML.net, TensorFlow, ONNX, QATM, OpenCV oder Tesseract erstellt und die dabei erzielten Resultate verglichen. In einer Nächsten Phase ging es darum, die zuvor als vielversprechend identifizierten Methoden auf den sehr spezifischen hier vorliegenden Anwendungsfall der Relaisschaltpläne zu optimieren. Nachdem diese Optimierung abgeschlossen war, konnte ein Domainmodel geplant werden, welches das Persistieren der gefundenen Informationen in einer Datenbank erlaubt hat. In der abschliessenden Phase wurden die einzelnen Komponenten zu einem einfach anwendbaren Produkt zusammengeführt.

Wesentliche Ergebnisse

Der, bei diesem Projekt entstandenen, Applikation kann eine Anlagedokumentation übergeben werden, was den vollautomatischen Analyseprozess startet. Das Analyseresultat kann nach Abschluss des Analyseprozesses als PDF oder in Bildform exportiert werden. Das Analyseresultat ist zudem in einer Datenbank gespeichert und kann von dort aus für weiterführende Analysen und Verknüpfungen verwendet werden. Dies erlaubt eine universelle Verwendung der extrahierten Informationen. Zudem wird das Arbeiten mit Anlagedokumentationsdateien viel effizienter, da im exportierten PDF nach den detektierten Texten gesucht werden kann. Die Erkennungsrate der Schemasymbole liegt bei über 90% und die korrekte Texterkennungsrate auf sauberen, digital erstellten Plänen liegt ebenfalls bei über 85%.

Inhalt

Projektbeschrieb

Ausgangslage

Ein Stellwerk ist eine ortsfeste Bahnanlage der Eisenbahn zur Steuerung des Bahnbetriebs. Es dient der Stellung von Fahrwegelementen wie Weichen und Gleissperren, stellt Abhängigkeiten zwischen den Fahrwegelementen und Signalen her und bindet Bahnübergangssicherungsanlagen in die Sicherungslogik ein. Mittels Gleisfreimeldeeinrichtungen prüft das Stellwerk jederzeit welche Gleisabschnitte von Fahrzeugen belegt sind.

Das Relaisstellwerk ist eine ältere Stellwerks-Bauform, in der die sicherungstechnischen Abhängigkeiten vollständig elektrisch durch Relais in komplexen Schaltungen implementiert werden. In der Schweiz sind noch immer viele Relaisstellwerke im Betrieb und es werden weiterhin auch neue gebaut.

In der Schweiz sind noch immer viele Relaisstellwerke im Betrieb und es werden weiterhin auch neue gebaut. Es ist daher wünschenswert, die Projektabwicklung in diesem Geschäft mittels innovativer Ansätze fit für die Zukunft zu machen.

Problemstellung

Die Dokumentation älterer Anlagen umfasst mehrere tausend Seiten Stromlaufpläne als PDF, in denen sich gescannte Tuschzeichnungen und moderne CAD-Zeichnungen abwechseln (siehe Abbildungen unten). Durch diese Mischform sind diese PDF's nicht digital durchsuchbar, was das Arbeiten bei Umbauten und Anpassungen sehr ineffizient macht.

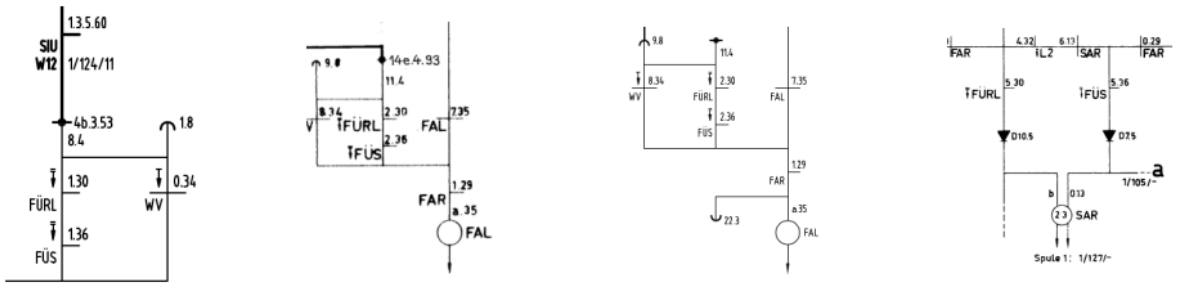


Figure 1Schema Aufgabenstellung

Lösungsansatz

Methoden aus den Gebieten der Bildverarbeitung und der künstlichen Intelligenz werden angewendet um aus diesen Stromlaufplänen automatisiert Informationen zu extrahieren und diese als Metadaten in den PDF's zu hinterlegen. Dies erlaubt die digitale Suche nach Schaltungsteilen, was das Arbeiten viel effizienter macht. Ausserdem können diese Informationen intelligent verknüpft werden was die Projektierungsarbeit teilweise automatisiert oder als qualitätssichernde Kontrollmassnahme dienen kann.

Ziel

Das Ziel dieser Arbeit ist es, die Umsetzbarkeit des obigen Ansatzes zu untersuchen und den besten Lösungsweg zu identifizieren. Dabei gibt es folgende Herausforderungen zu bewältigen:

1. Welche Methoden und Ansätze führen zu den besten und robustesten Resultaten?
2. Wie können effizient gelabelte Datensets aus CAD-Dateien erstellt werden?
3. Wie könnte eine Anwendung aussehen, welche die identifizierten Lösungen in den Geschäftsprozess einbindet?
4. Wie würde der nächste Schritt auf dem Weg zu Automatisierung der Projektierungsarbeit aussehen?

Team

- Dominic Klinger dominic.klinger@ost.ch
- Christian Bisig christian.bisig@ost.ch

Projektplan

Einführung

In diesem Dokument wird der Projektplan des Projektes «AI for Relay Interlocking» beschrieben, welcher eine Übersicht für die getätigten Arbeiten geben soll. Die Planung, Organisation und weitere Bereiche des Projektaufbaus werden thematisiert und erläutert. Dieser Projektplan dient als Grundlage für die hier durchgeführte SA.

Gültigkeitsbereich

Der Gültigkeitsbereich beschränkt sich auf die ganze Projektdauer des Projektes «AI for Relay Interlocking», welches im Rahmen einer SA im Semester HS2021 an der OST durchgeführt wird. Das Dokument wird stetig aktualisiert.

Projektorganisation

Projektmitarbeiter

- Dominic Klinger dominic.klinger@ost.ch
- Christian Bisig christian.bisig@ost.ch

Projektbetreuer

- Farhad Mehta farhad.mehta@ost.ch

Auftraggeber

- Till Stöckli till.stoeckli@siemens.com
- Andreas Fürst andreas.fuerst@siemens.com
- Peter Schudel peter.schudel@siemens.com

Projektstrukturplan

Unterhalb ist der initiale Projektstrukturplan ersichtlich. Darin wurde zu Beginn der Elaborationsphase die Aufgabenstellung analysiert und in einzelne Komponenten aufgeteilt. In einem weiteren Schritt wurden diese Pakete in die Phasen von Scrum+ aufgeteilt und daraus die Epics definiert. Die Pakete wurden in die drei grundsätzlich unterschiedlichen Kategorien Koordination, Infrastruktur und Applikation unterteilt.

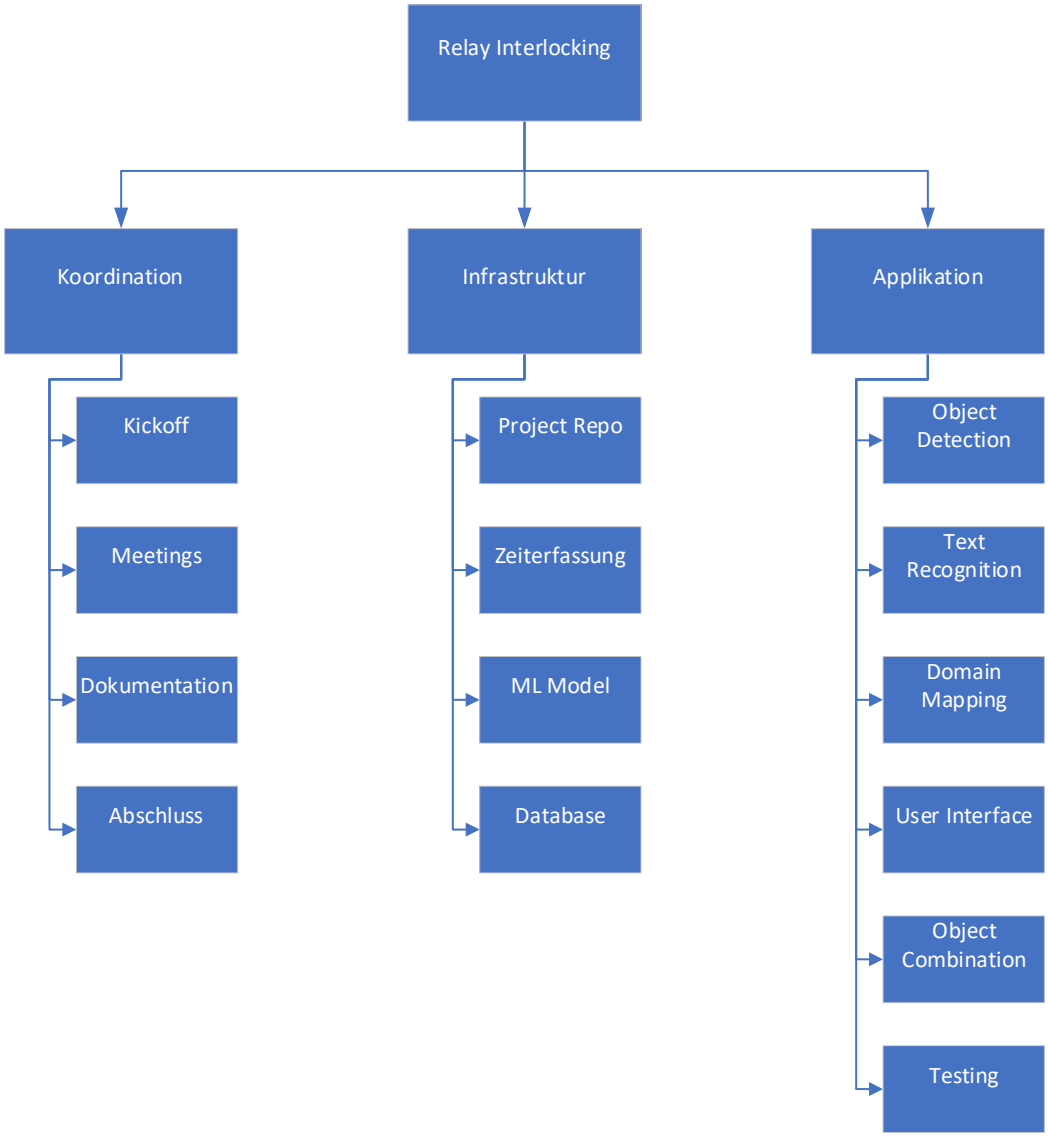


Figure 2Projektstrukturplan

Management Abläufe

Kostenvoranschlag

Projektdauer	14 Wochen
Anzahl Projektmitarbeiter	2 Personen
Arbeitspensum pro Woche	16h
Arbeitspensum insgesamt	448h
Projektstart	Mo. 20.09.2021
Projektende	Do. 23.12.2021

Zeitliche Planung

Unsere Zeitplanung beruht auf Scrum+ mit den vier Phasen Inception, Elaboration, Construction und Transition. Alle Phasen sind in Iterationen von zwei Wochen unterteilt.

Phasen

Die nachfolgenden Phasen wurden als Eckpunkte im Projektverlauf festgelegt. Dabei wurde vereinbart, dass jeder Phasenabschluss mit einem Meilenstein gleichgesetzt werden kann. Die Phasendauern wurden gemäss Scrum+, bei dem die Phasen von RUP und die Agilität von Scrum vereint werden, festgelegt. Dabei ist der Hauptbestandteil der verfügbaren Zeit auf die beiden Phasen Elaboration, in welcher die Machbarkeit geprüft und verschiedene Technologien evaluiert werden, und Construction, in der die eigentliche Entwicklungsarbeit stattfindet, aufgeteilt. In der Tabelle unterhalb können die einzelnen Phasen mit dem jeweils dazugehörigen Start- und Enddatum gefunden werden. In einem weiteren Schritt wurden zusätzliche Meilensteine definiert und den unten ersichtlichen Phasen zugewiesen. Die Definition der Meilensteine kann auf der nachfolgenden Seite gefunden werden.

Phase	Startdatum	Enddatum
Inception	Mo 20.09.2021	So 26.09.2021
Elaboration	Mo 27.09.2021	So 31.10.2021
Construction	Mo 01.11.2021	So 12.12.2021
Transition	Mo 13.12.2021	Do 23.12.2021

Table 1 Projektphasen

Meilensteine

Bezeichnung	Inhalt	Datum
M1 Projektplan	- Projektplan ausgearbeitet - Grobe Projektplanung auf ebene Epic durchgeführt - Entwicklungsumgebung gemäss Projektplan angepasst - Projektplan besprochen - Zeiterfassung aufgesetzt	07.10.2021
M2 Anforderungsanalyse	- Anforderungen aufgeführt - Wichtigste Use Cases dokumentiert - UserStories basierend auf Use Cases erstellt (Für Sprint Planung) - Risikoanalyse durchgeführt	17.10.2021
M3 End of Elaboration	- Prototypen für OCR / Objektdetektion / PDF-Manipulation evaluiert - Grösste Risiken beseitigt - Plan für tatsächliches Produkt basierend auf neuen Erfahrungen ausgearbeitet	31.10.2021
M4 Alpha	- Erste Version der Applikation ist verfügbar - Bauteile können auf Schaltplänen detektiert werden - Bauteilbeschriftungen werden erkannt und können dem Plan entnommen werden - Bauteilinformationen können dem PDF hinterlegt werden - Suche im PDF nach hinterlegten Attributen funktioniert	07.11.2021
M5 Zwischenpräsentation	- Präsentation der Alphaversion bei Siemens Mobility AG - Kundenwünsche basierend auf Präsentation festgehalten - Plan für weiteres Vorgehen erstellt basierend auf Rückmeldung der Auftraggeber	11.11.2021
M6 Beta	- Neue lauffähige Version verfügbar - Version ist getestet und entspricht den Qualitätsanforderungen - Funktionalität gemäss Rückmeldung Zwischenpräsentation	12.12.2021
M7 Projektabgabe	- Release Version verfügbar - Abschliessende Bugfixes wurden auf Betaversion durchgeführt - Dokumentation nachgeführt	17.12.2021
M8 Projektpräsentation	- Präsentation des Produktes bei Siemens Mobility AG	23.12.2021

Table 2 Meilensteine

Timeline

Unterhalb ist eine Grafik mit einer Übersicht über alle Phasen, Iterationen und Meilensteine inkl. den wichtigsten Aufgaben pro Iteration ersichtlich. Sie soll einen groben Überblick darüber geben, welche Aufgaben in welchem Zeitraum abgearbeitet werden müssen, damit das Projekt im vorgegebenen Zeitraum erfolgreich abgeschlossen werden kann.

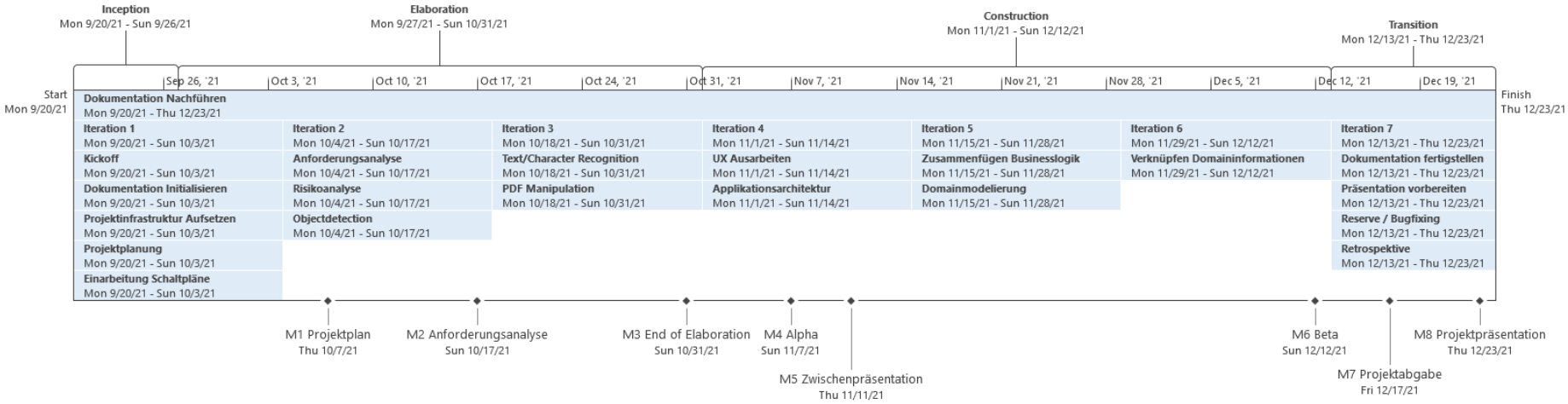


Figure 3 Timeline

Epic Schätzungen

In der unterhalb ersichtlichen Tabelle sind erste grobe Schätzungen zu den wichtigsten Epics ersichtlich. Basierend auf den hier ersichtlichen Epics werden in den jeweiligen Iterationen feingranulare Tasks erstellt, welche dann für die Zeiterfassung verwendet werden.

Epic Name	Dauer
Inception	**8 days**
Kickoff	6 hrs
Dokumentation Initialisieren	2 hrs
Projektinfrastruktur Aufsetzen	1 day
Projektplanung	1 day
Einarbeitung in Grundlagen der Schaltpläne	1 day
Elaboration	**20 days**
Anforderungsanalyse	1 day
Risikoanalyse	1 day
Objectdetection	6 days
Text/Character Recognition	6 days
PDF Manipulation	6 days
Construction	**24 days**
UX Ausarbeiten	6 days
Applikationsarchitektur	3 days
Domainanalyse und Modellimplementierung	3 days
Zusammenfügen der evaluierten Varianten zur Objectdetection, OCR sowie PDF Manipulation	6 days
Verknüpfen der Domaininformationen	6 days
Transition	**8 days**
Dokumentation fertigstellen	3 days
Präsentation vorbereiten	2 days
Reserve / Bugfixing	2 days
Projektretrospektive	1 day
Total	**60 days**

Table 3 Zeitschätzung Epics

Arbeitspakete

Die Verwaltung der Arbeitspakete findet auf GitLab statt. Es gibt projektspezifische Pakete, welche immer einem Meilenstein zugewiesen sind und eine Zeiteinschätzung haben. Je weiter in der Zukunft die Implementation der Arbeitspakete geplant ist, desto generischer und ungenauer sind diese definiert. Die Pakete werden mit der Zeit immer genauer und zu kleineren Tickets umgewandelt. Zusätzlich gibt es noch Arbeitspakete für den ganzen Overhead, wie Meetings und Reviews.

Qualitätsmassnahmen

Um die Qualität unseres Produktes sicherzustellen haben wir die folgenden Massnahmen definiert:

Massnahme	Zeitraum	Ziel der Massnahme
Systemtests (Checkliste)	Ganzes Projekt	Funktionsumfang der Applikation sicherstellen. Alle Use Cases können mit der Applikation abgedeckt werden.
Unit-Tests	Bei Implementierung	Stellen sicher, dass die einzelnen Elemente so funktionieren, wie sie sollen. Diese Massnahme soll dazu beitragen, dass der Code robust bleibt.
Integration-Tests	Bei Implementierung	Diese Massnahme soll sicherstellen, dass die einzelnen Komponenten sauber miteinander Interagieren können.
Codereviews	Bei Implementierung (Merge Request)	Code Reviews führen zu wichtigem Knowhow Austausch. Zudem können diese Reviews die Wahrscheinlichkeit, dass der produktive Master Branch nicht mehr funktioniert, verkleinert werden.
Code Metriken	Ganzes Projekt	Von Zeit zu Zeit sollen die Code Metriken des Projektes ermittelt werden. Dabei sollen Code Smells, Security Probleme und duplizierter Code ermittelt werden, damit diese Problemstellen in einem Refactoring verbessert werden können.

Table 4 Qualitätsmassnahmen

Eingesetzte Tools

Zur Umsetzung der oben definierten Massnahmen werden die folgenden Tools eingesetzt:

- XUnit als TestFramework zur Implementation von Unit- und Integrationstests
- Moq als Library für das Mocking von Objekten bei Unittests
- Sonarqube zur Ermittlung der Code Metriken.

Code Style Guidelines

Für die Implementation der .NET Logik wurde entschieden, grundsätzlich auf die C# Coding Standards und Naming Conventions von dofactory zu setzen. Diese Guidelines decken sich grösstenteils mit den C# Coding Conventions von Microsoft, sind jedoch in einigen Themen etwas spezifischer. Darin werden unter anderem die folgenden Punkte genauer spezifiziert:

– PascalCasing für Klassen und Methoden Namen
– camelCasing für lokale Variablen und Methodenargumente
– Starte Interfacenamen mit einem I
– Deklaration aller Member Variablen am Anfang von Klassen

Die oben referenzierten Coding Standards beinhalten keine abschliessende Empfehlung bezüglich Underscores vor privaten Klassenvariablen. Aus diesem Grund wurde entschieden, dass private Klassenmember mit einem _ am Anfang des Namens benannt werden sollen. Dieser Entscheid hat zur Folge, dass Membervariablen nicht mit this. angesprochen werden müssen, was sich positiv auf die Lesbarkeit des Codes auswirkt.

Risikoanalyse

R1: Wahl eines falschen Ansatzes zur Analyse der Bilddaten

Beschreibung	Um die gewünschten Daten aus den gestellten PDF-Dokumenten zu generieren, stehen verschiedene Methoden zur Verfügung. Dies kann dazu führen, dass auf der falschen Methode aufgebaut werden kann.
Maximaler Schaden	65 Stunden
Eintrittswahrscheinlichkeit	20%
Gewichteter Schaden	13
Vorbeugung	Um einen passenden Ansatz zu finden, welcher für dieses Projekt am besten geeignet ist, werden verschiedene Prototypen und Ansätze aufgebaut und ungesetzt, damit hier die passendste Variante ausfindig gemacht werden kann.
Verhalten bei Eintreten	Den Scope des Projekts verringern, alternative Prototypen bauen und testen.

R2: Falscher Umgang mit den Anforderungen

Beschreibung	Wichtige Anforderungen oder Use Cases gehen vergessen oder werden im Zeitaufwand unterschätzt
Maximaler Schaden	30 Stunden
Eintrittswahrscheinlichkeit	15%
Gewichteter Schaden	4.5
Vorbeugung	Die Anforderungen ans Projekt sollen in der Elaborationsphase als erstes genau analysiert werden. Bei Unklarheiten sollen Abklärungen zusammen mit den Ansprechpersonen bei Siemens Mobility AG gestartet werden. Um möglichst genaue Zeitschätzungen erzielen zu können, sollen diese im Team mit Hilfe von «Planning Poker» durchgeführt werden.
Verhalten bei Eintreten	Den Scope in Ansprache mit dem Industriepartner anpassen.

R3: Technologien

Beschreibung	Da bei diesem Projekt Technologien zum Einsatz kommen, welche dem Team noch nicht umfänglich bekannt sind, kann dies zu unvorhergesehenen Problemen führen.
Maximaler Schaden	65 Stunden
Eintrittswahrscheinlichkeit	30%
Gewichteter Schaden	19.5
Vorbeugung	Durch den Aufbau mehrerer Prototypen und dem Testen dieser, sollen grössere Probleme mit den gewählten Technologien aus dem Weg geschafft werden.
Verhalten bei Eintreten	Sollten Probleme auftauchen, so ist es daran, eine geeignete Alternative zur gewählten Technologie zu finden und sich das nötige Wissen dazu anzueignen.

R4: Quelldaten in schlechter Qualität

Beschreibung	Die vom Projektpartner gelieferten Daten eignen sich nicht dazu, die gewünschten Daten zu extrahieren.
Maximaler Schaden	30 Stunden
Eintrittswahrscheinlichkeit	10%
Gewichteter Schaden	3
Vorbeugung	Es wurden von Projektbeginn weg viele Daten zur Verfügung gestellt. Zudem wurde die Möglichkeit geboten jederzeit zusätzliche Daten zu erhalten. Neben den PDF-Anlagedokumentation wurde auch darüber gesprochen, die Daten aus CAD Zeichnungen zu extrahieren.
Verhalten bei Eintreten	Alternative Varianten suchen, die weniger Daten benötigen, um gute Resultate zu liefern.

Risikomatrix

Unterhalb ist die Risikomatrix, in der die zuvor erkannten Risiken eingetragen sind, zu finden. Dabei zeigt sich, dass es sich beim Risiko R3 um das Risiko handelt, das am gefährlichsten werden könnte. Aus diesem Grund wurde entschieden, in der Elaborationsphase möglichst viele Prototypen basierend auf einem möglichst breiten Spektrum an Technologien umzusetzen. Daneben muss auch Risiko R1 im Auge behalten werden, da auch dieses Risiko ein erhöhtes Potential hat, den Projektverlauf negativ zu beeinflussen. Mit der zuvor beschriebenen, ausgedehnten Prototypisierung in der Elaborationsphase sollte aber auch dieses Risiko in einem unproblematischen Bereich gehalten werden können. Dennoch ist es besonders wichtig, möglichst früh im Projekt eine technische Möglichkeit zu finden, mit welcher das Projekt umsetzbar ist. Die Risiken R2 und R4 liegen im Moment im grünen Bereich und können daher mit den zuvor definierten vorbeugenden Massnahmen belassen werden. Die Risiken werden im Projektverlauf stetig beobachtet und gegebenenfalls angepasst. Ziel ist es, nach der Elaborationsphase möglichst geringe Restrisiken zu haben, damit der restliche Projektverlauf nicht gefährdet wird.

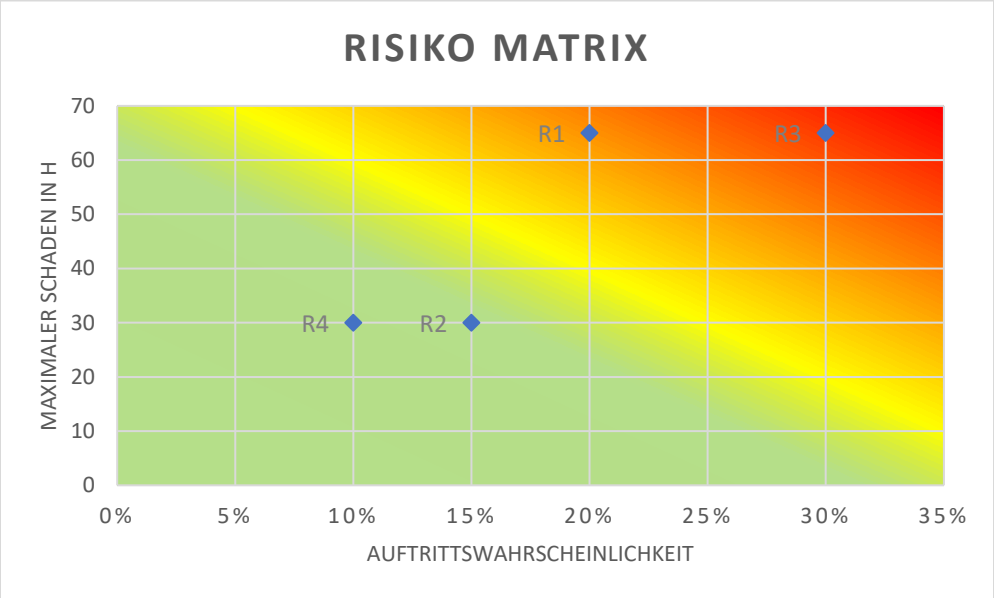


Figure 4 Risikomatrix

Anforderungsspezifikation

Einführung

Zweck

Dieser Abschnitt soll einen Überblick über die Anforderungen an das bei diesem Projekt entwickelte Produkt geben. Die Anforderungen sollen dabei in einzelne Anwendungsfälle unterteilt und von dort aus weiteranalysiert werden. Basierend auf den hier definierten Anforderungen werden Testfälle definiert, die positiv zur Qualitätssicherung beitragen sollen.

Übersicht

Nach einer Einführung beinhaltet das Dokument eine allgemeine Beschreibung, sowie eine genaue Erklärung der funktionalen und nicht-funktionalen Anforderungen. Eine Übersicht zu den definierten Use Cases kann dem Use Case Diagramm entnommen werden. Zudem wurden die Use Cases im Anschluss ans Diagramm kurz beschrieben. Zum Schluss werden im letzten Teil des Dokumentes die nicht funktionalen Anforderungen in Bezug auf Zuverlässigkeit, effiziente Performance, Benutzbarkeit sowie Änderbarkeit definiert.

Allgemeine Beschreibung

Produktperspektive

«Ai Interlocking» ist eine Applikation, welche die Extraktion von Informationen aus bestehenden Schemas übernimmt. Bei den Schemas handelt es sich um die Anlagendokumentationen verschiedener Schweizer Bahnhöfe. Es soll nun möglich sein, Symbole und deren Bezeichnungen aus dem Schema zu entnehmen und diese zu digitalisieren. Es handelt sich bei den Schemas teilweise auch um handgezeichnete Dokumentationen. Am Ende des Digitalisierungsprozesses soll ein umfangreiches Set an Informationen zur Verfügung stehen, um das Durchsuchen der Informationen zu ermöglichen.

Produktfunktion

Der Anwendung wird eine bestehende Anlagendokumentation im PDF-Format übergeben. Das PDF wird anhand der Lesezeichen oder Schemanummern in relevante Bereiche unterteilt und zerlegt, anschliessend in Bildinformation umgewandelt. Im Anschluss wird nun versucht mithilfe von Templatematching und OCR (optical character recognition – Texterkennung) Informationen zu extrahieren und in einem neuen PDF verfügbar zu machen. Ebenso werden die Daten in einer internen Datenbank abgelegt, um so die zukünftige Editierung und erneute Ausgabe in ein PDF zu ermöglichen. In einem späteren Zuge soll es zusätzlich möglich sein, relevante Informationen zusammenzutragen und eine Statistik ausgeben zu lassen.

Benutzercharakteristik

Die Anwender dieser Software ist Siemens Mobility im Rahmen der Erstellung und Bearbeitung von Anlagendokumentation von Bahnhöfen.

Einschränkungen

Bei dem Endprodukt handelt es sich nicht um eine Software, die eine 100%ige Zuverlässigkeit bietet. Dieses Projekt hat einen Forschungscharakter, bei dem verschiedene Technologien zur Erkennung geprüft werden.

Abhängigkeiten

Diese Anwendung basiert auf verschiedenen Bibliotheken, die das Umwandeln und Bearbeiten von PDF-Dateien sowie das Editieren und Verarbeiten von Bildern ermöglichen.

Use Cases

Use Case Diagramm

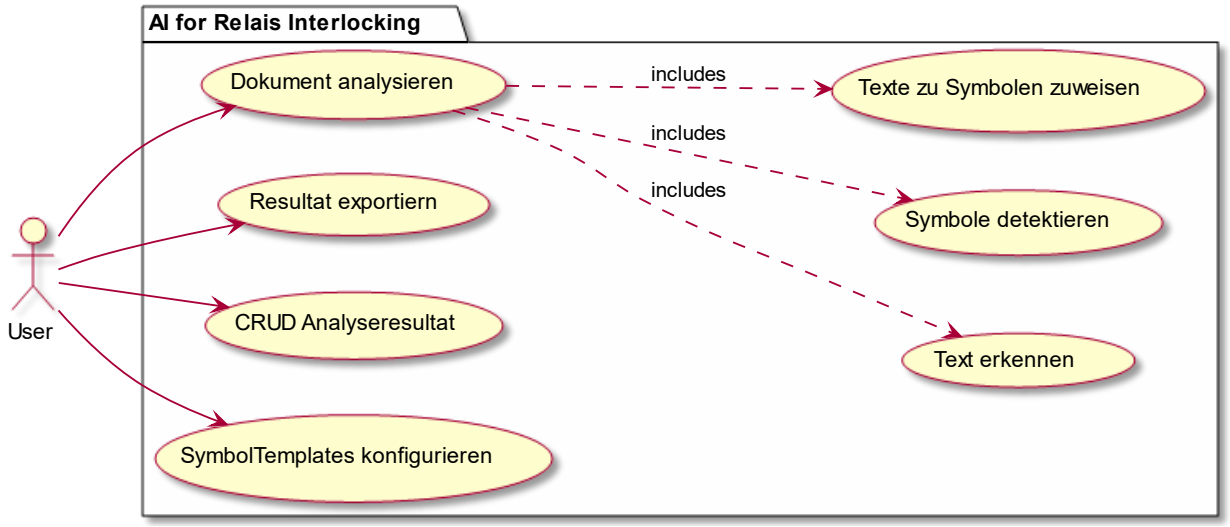


Figure 5 UseCase Diagramm

Use Case UC-1: Anlagedokumentationen analysieren

Als Eingabe wird eine Anlagendokumentation angegeben. Diese wird von der Applikation vollautomatisch analysiert und ausgewertet. Die Daten werden alle in einer Datenbank abgelegt. Bei der Analyse soll der Text auf der Schemaseite erkannt und die vorhandenen Symbole detektiert und korrekt identifiziert werden. Nachdem die Symbole detektiert und die Texte gelesen worden sind, sollen diese automatisch einander zugeordnet werden. Spezielle detektierte Texte sollen zudem einer Plausibilitätsprüfung unterzogen werden, um die Qualität der Analyseresultate so hoch wie möglich zu halten.

Use Case UC-1.1: Texte erkennen

Einzelne Schemaseiten der Anlagedokumentation, die analysiert wird, sollen von einer Texterkennungsfunktionalität untersucht werden. Dabei soll so viel Text wie möglich aus der Seite extrahiert werden. Bei den Texten auf den Schemaseiten handelt es sich um Bezeichnungen, Identifier und zusätzliche Beschriftungen. Standardmässige Texterkennung wird also nicht funktionieren, da es sich weder um ganze Sätze noch um eigentliche Wörter, wie sie in einem Wörterbuch vorkommen, handelt.

Use Case UC-1.2: Symbole detektieren

Einzelne Schemaseiten der Anlagedokumentation, die analysiert wird, sollen bezüglich der in den Schemas enthaltenen Symbole untersucht werden. Dabei sollen die Koordinaten sowie die Symboltypen der Schemasymbole ermittelt werden. Die Liste von verschiedenen Symboltypen soll möglichst einfach erweiterbar sein, um die Applikation zukunftssicher zu machen.

Use Case UC-1.3: Texte zu Symbolen zuweisen

Die zuvor erkannten Texte und detektierten Symbole sollen einander vollautomatisch zugewiesen werden können. Dabei sollen auf den Symboltypen basierende Regeln zum Einsatz kommen, welche entscheiden können, ob ein erkannter Text zu einem Symbol gehören kann oder nicht. Die Zuweisungsmöglichkeiten sollen einfach konfigurierbar sein, damit auch neu hinzugefügte Symbole korrekt den zugehörigen Texten assoziiert werden können.

Use Case UC-2: Analyseresultate exportieren

Analyseresultate können als PDF exportiert werden, wobei die erkannten Texte von den Suchfunktionalitäten gebräuchlicher PDF-Reader gefunden werden sollen. Dies erleichtert die Arbeit mit den sehr umfangreichen Anlagedokumentationen. Der Umfang der zu exportierenden Informationen soll bestimmt werden können. Es soll entschieden werden können, ob Texte, Symbole, Zuweisungen oder eine beliebige Kombination daraus ins PDF geschrieben werden sollen. Zudem soll es für Debugging und Bearbeitungszwecke die Möglichkeit geben, einzelne Seiten aus dem analysierten Dokument separat zu exportieren.

Use Case UC-3: Analyseresultate bearbeiten

Die Resultate abgeschlossener Analysen sollen bearbeitet werden können, um falsch erkannte Texte oder Symbole korrigieren zu können. Dabei soll es möglich sein, fehlende Texte oder Symbole hinzuzufügen, fehlerhafte Texte oder Symboleigenschaften anzupassen und zu viel erkannte Texte oder Symbole zu löschen. Natürlich sollen diese Analyseresultate auch dargestellt werden können, um eine effiziente Bearbeitung zu ermöglichen. Fertig bearbeitete Analyseresultate sollen zurück in die Datenbank geschrieben werden können, damit zukünftige erweiterte Analysen darauf ausgeführt werden können. Zudem sollen die angepassten Resultate gleich wie ursprünglich erkannte Resultate exportiert werden können.

Use Case UC-4: SymbolTemplates konfigurieren

Zusätzliche Symbole sollen einfach als Templates in die Analyse miteinbezogen werden können. Diese Konfigurationen sollen von der Analysesoftware verwendet werden, um die Symbole zu detektieren. Diese Funktionalität erlaubt das Hinzufügen von neuen Symboltypen. Zudem sollen existierende Symboltyp Konfigurationen angepasst werden können, wenn sich herausstellen sollte, dass die Erkennungs- oder Zuweisungsrate unzureichend ist. Ausserdem soll es möglich sein, bestimmte Eigenschaften für jedes Template von aussen zu konfigurieren. Dabei handelt es sich beispielsweise um Parameter, welche die Textzuweisung beeinflussen können oder auch die Zuverlässigkeit bei der Symbolerkennung.

Weitere Anforderungen

Qualitätsmerkmale

In diesem Abschnitt wird auf die zusätzlichen Qualitätsanforderungen eingegangen, die an das Produkt gestellt worden sind. Dabei handelt es sich zum einen um nicht funktionale Anforderungen (NFR) und zum anderen um Nebenbedingungen, wie vorgegebene Schnittstellen oder weitere Randbedingungen. Die nicht funktionalen Anforderungen sind dabei in die Gruppen Zuverlässigkeit, Performanz, Benutzbarkeit sowie Änderbarkeit unterteilt. Die NFRs wurden basierend auf der ISO 25010 Norm analysiert.

Zuverlässigkeit (reliability)

NFR-1: Reife (maturity)

Business Goal	Auftretende Fehler sollen geloggt werden und somit möglichst reproduzierbar sein, damit diese in einem ersten Schritt nachvollzogen und in einem zweiten Schritt behoben werden können.
Scenario	Während dem Betrieb treten unbekannte Fehler auf, die nicht reproduziert und somit nicht behoben werden können.
Reaction	Unerwartete Fehler werden mit den zum Reproduzieren benötigten Informationen in eine Datei geschrieben. Diese Datei kann im Fehlerfall gelesen werden, wobei der darin enthaltene Inhalt Aufschluss über das aufgetretene Problem geben soll.
Measure	Folgende Informationen sollen gesammelt und in ein Logfile geschrieben werden: - Datum und Uhrzeit des Fehlerauftrittes - Error Message - Informationen zum Fehlertyp - Informationen zur Komponente, in welcher der Fehler aufgetreten ist.

NFR-2: Fehlertoleranz (fault tolerance)

Business Goal	Die Applikation soll mit Fehleingaben oder nicht unterstützten Dokumenten umgehen können, ohne dabei abzustürzen.
Scenario	Eine Benutzerinteraktion bringt die Applikation zum unerwarteten Absturz.
Reaction	Fehler, die das Programm zum Absturz bringen könnten, werden abgefangen und dem User mitgeteilt. Genaue Informationen zu den aufgetretenen Fehlern werden in die Logfiles geschrieben. Aus den darin enthaltenen Informationen können Patches entwickelt werden.
Measure	Die Applikation soll in keinem Fall abstürzen, ohne den User in irgendeiner Form über das Problem zu benachrichtigen. Bei einem Absturz sollen die Resultate bereits vollständig durchgeführter Analysen auf keinen Fall verloren gehen können.

Performanz (performance efficiency)

NFR-3: Zeitverhalten (time behaviour)

Business Goal	Die Analyse einer 5000-seitigen Anlagedokumentation soll in einer sinnvollen Zeit durchgeführt werden können. Es soll möglich sein, eine Anlagedokumentation über die Nacht analysieren zu lassen, sodass am nächsten Morgen die Resultate vorliegen.
Scenario	Die Analyse einer Anlagedokumentation brauch so viel Zeit, dass die Analyseapplikation dadurch unbrauchbar wird.
Reaction	Die Applikation soll so weit wie möglich parallelisiert ausführbar sein. Es soll möglich sein, mehrere Schemaseiten gleichzeitig analysieren zu lassen. Dies soll dazu führen, dass der gesamte Analyseprozess zeitlich nicht aus dem Rahmen läuft.
Measure	Die Analyse eines 5000-seitigen Dokumentes soll nicht länger als 8 Stunden dauern. Die verfügbaren Ressourcen sollen so gut wie möglich ausgenutzt werden, um die Analyse so schnell wie möglich voranzutreiben.

Benutzbarkeit (usability)

NFR-4: Bedienbarkeit (operability)

Business Goal	Die GUI Applikation soll möglichst einfach die benötigte Funktionalität zur Verfügung stellen. Die Bedienung soll so intuitiv wie möglich gestaltet werden.
Scenario	Die Fenster sind mit Informationen überladen, worunter die Übersichtlichkeit leidet. Gewünschten Informationen oder Operationen können nicht gefunden werden.
Reaction	Das User-Interface soll möglichst hierarchisch aufgebaut werden. Dabei soll der Grundaufbau der Logik des Domain-Modells entsprechen. (Fileübersicht, Pageübersicht, einzelne Symbolübersicht)
Measure	Das GUI ist gemäss Domain-Modell hierarchisch aufgebaut. Es gibt eine Seite für die Dateiübersicht, eine für die Seitenübersicht und eine mit einer Übersicht über einzelne Symbole.

Änderbarkeit (maintainability)

NFR-5: Modularität (modularity)

Business Goal	Die Applikation soll so modular aufgebaut werden, dass einzelne Komponenten ausgewechselt werden können, ohne Anpassungen an anderen Komponenten durchführen zu müssen. Dasselbe soll auch für die Layer innerhalb der einzelnen Komponenten gelten.
Scenario	Änderungen an der «Schema_Analyzer» Komponente (Python) ziehen Änderungen an der .NET Komponente mit sich und beeinflussen deren Funktionalität.
Reaction	Die Komponenten und Layer aus welchen die Applikation aufgebaut ist, sind möglichst weitgehend voneinander getrennt und somit unabhängig.
Measure	Der modulare Aufbau der Applikation soll von Sonarqube kontrolliert werden. Damit soll sichergestellt werden, dass einzelne Teile problemlos ausgewechselt werden können, ohne die Funktionalität anderer Teile zu beeinträchtigen. Folgenden Sonarqube Metriken sollen dabei eingehalten werden: - Test-Coverage: Min. 80 % - Duplicated Lines: Max: 5 % - Maintainability Rating: Min. A - Reliability Rating: Min. A

NFR-6: Modifizierbarkeit (modifiability)

Business Goal	Die Applikation soll einfach erweiterbar sein und Komponenten sollen einfach angepasst werden können. Nach den Änderungen soll sichergestellt werden können, dass die Applikation immer noch wunschgemäss funktioniert.
Scenario	Nach der Anpassung einer Komponente treten Fehler auf, die hätten vermieden werden können. Diese auftretenden Fehler machen die Applikation unbrauchbar.
Reaction	Tests werden automatisch ausgeführt, nachdem Änderungen vorgenommen worden sind. Damit soll sichergestellt werden können, dass die Applikation noch gemäss Anforderungen funktioniert.
Measure	Automatisierte Tests erlauben es Änderungen zu überprüfen und sicherzustellen, dass diese keinen Einfluss auf die korrekte Funktionsweise der Applikation haben. Mit diesen Tests soll mindestens 80% des Codes abgedeckt werden. Ein weiterer Punkt, der das oben beschriebene Problem verhindern soll, sind die MergeRequests, bei welchen jeweils ein Review vom anderen Teammitglied durchgeführt wird bevor die Änderungen in den produktiven Master Branch kommen.

Schnittstellen

Da es sich um eine neu entwickelte Applikation handelt, welche die Machbarkeit von Schemaanalysen evaluieren soll, muss auf keine bestehenden Schnittstellen zugegriffen werden. Die Applikation muss jedoch mit sehr grossen PDF-Dateien umgehen können, die teilweise über 7000 Seiten lang und über 1 GB gross sind. Informationen zu applikationsinternen Schnittstellen, über welche die Kommunikation zwischen den verschiedenen Komponenten läuft, können dem Kapitel «Architektur» entnommen werden.

Randbedingungen

Grundsätzlich wurden keine Randbedingungen für das zu erarbeitende Produkt festgelegt. Vonseiten Siemens Mobility AG wurde aber erwähnt, dass bei ihnen bereits viele Applikationen aus dem .NET Umfeld im Einsatz sind. Aus diesem Grund wurde das .NET 5.0 Framework unter der Verwendung von C# 9.0 als initialen Technologie Stack gewählt. Damit verbunden wurde entschieden das User-Interface mit WPF in XAML umzusetzen. Weitere Informationen zu den gefällten Entscheidungen können dem Kapitel «Entscheidungen» entnommen werden.

Domainanalyse

Einführung

Zweck

Dieser Abschnitt soll dabei helfen, eine Übersicht über das Domainmodell von «AI Interlocking» zu erhalten. Auf den in dieser Analyse gefundenen Erkenntnisse wurden in einem späteren Schritt die Modellkassen der «AI Interlocking» Applikation aufgebaut. Damit verbunden wird die hier durchgeführte Domainanalyse auch einen Einfluss auf das zugrundeliegende Datenbankschema haben.

Übersicht

Dieser Abschnitt beinhaltet nachfolgend ein UML Domaindiagramm mit einer Beschreibung zu den wichtigsten Konzepten im Anschluss. Unterhalb der Domainanalyse ist ein Sequenzdiagramm auffindbar, welches das Zusammenspiel der Systemkomponenten zur Erfüllung der zuvor definierten Anwendungsfälle visualisiert.

Domain Modell

Strukturdiagramm – «AI Interlocking»

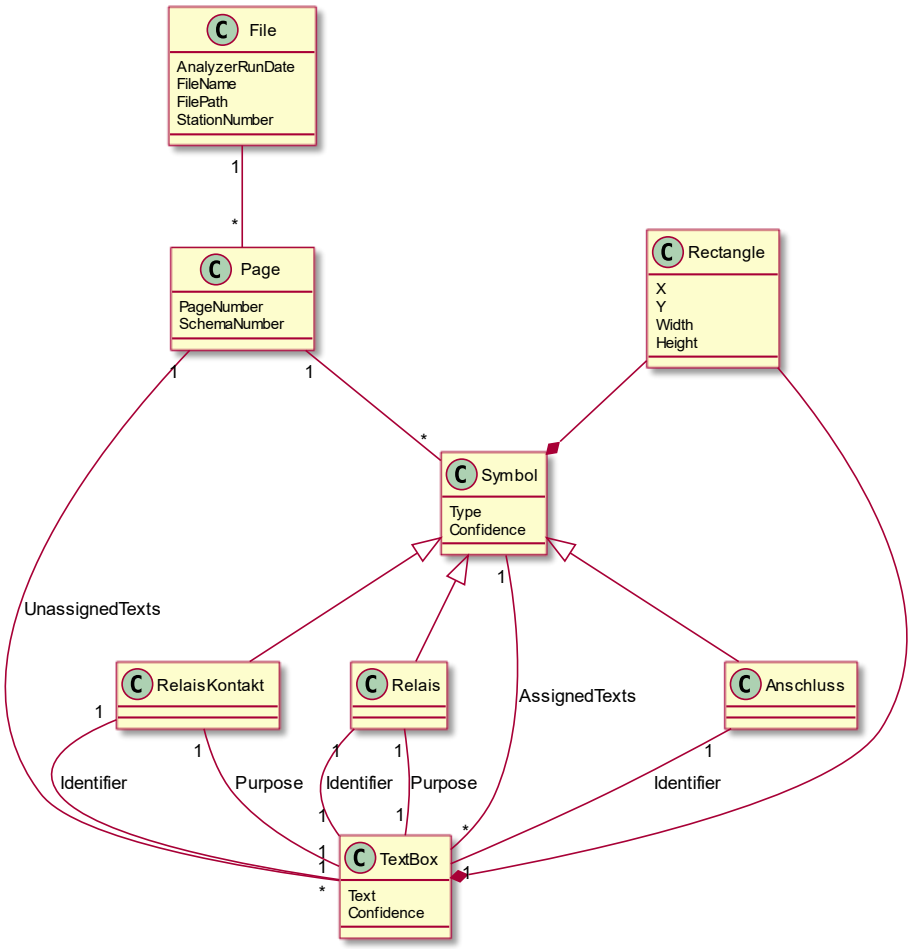


Figure 6 Domain Modell

Im oben dargestellten Diagramm sind die Domainklassen des «AI Interlocking» .NET Teils ersichtlich. Diese wurden ausgehend von einer analysierten Dokumentation geplant und umgesetzt. Jede analysierte Dokumentation entspricht einem «File» Objekt. Darin enthalten sind neben dem Namen und Pfad der analysierten Datei auch das Analysedatum sowie die Stationsnummer, die aus den Plänen ermittelt wird. Jedes «File» Objekt beinhaltet eine Liste von «Page» Objekten. Dabei handelt es sich jeweils um die Informationen zu einer analysierten Schemaseite. Abgelegt werden dabei jeweils die Seitennummer in der Dokumentation sowie die Schemanummer, die aus dem Schema gelesen wird. Jedes «Page» Objekt besitzt seinerseits eine Liste von «TextBox» Objekten, die TextBoxen beinhaltet, welche keinem Symbol zugewiesen werden konnten. Neben diesen UnassignedTexts sind auch die DetectedSymbols für jede Seite hinterlegt. Für jedes Symbol sind die ihm zugewiesenen TextBoxen hinterlegt. Zudem haben die Spezialsymbole RelaisKontakt, Relais und Anschluss jeweils noch weitere Properties wie den Identifier oder Purpose. Diese Properties verweisen jeweils auf eines der, dem Symbol zugewiesenen TextBoxen.

Strukturdiagramm – «Schema Analyzer»

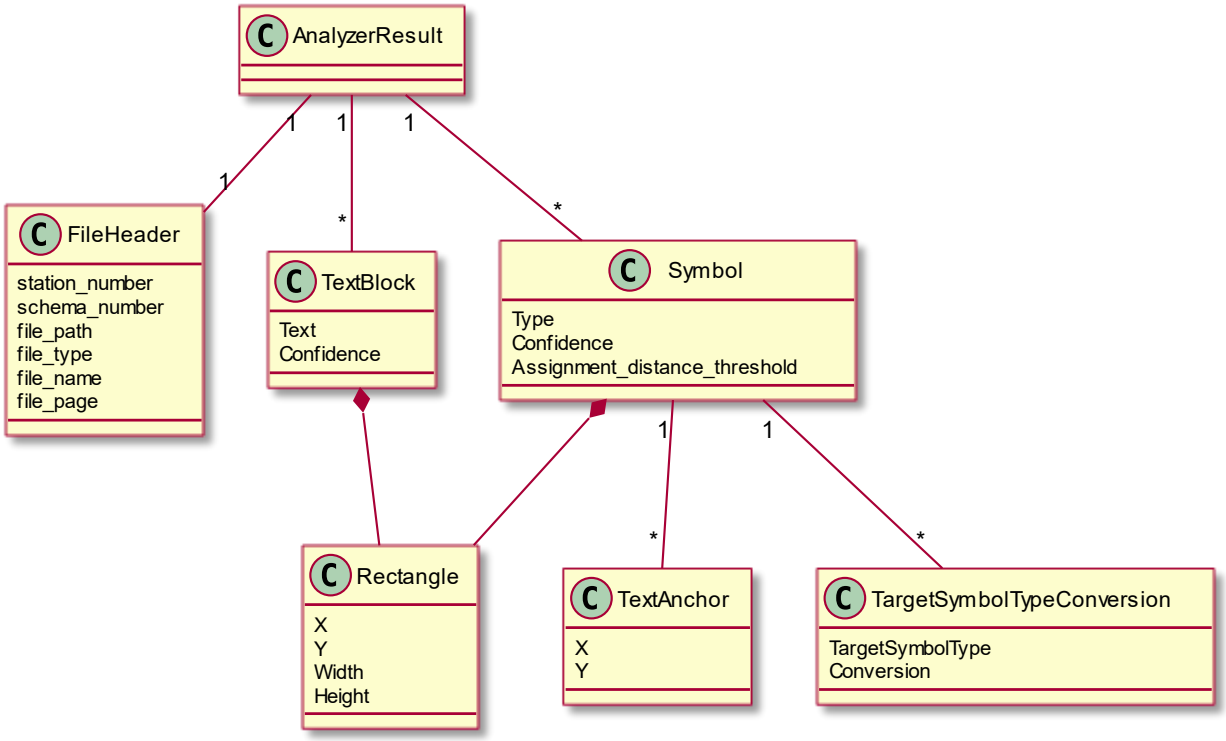


Figure 7 Schema_Analyzer Domain Modell Ausschnitt

Das oben dargestellte Diagramm zeigt das Domain-Modell des Python-Teiles (Schema_Analyzer). Dabei handelt es sich im Prinzip um einen Ausschnitt aus dem im letzten Abschnitt vorgestellten Domain Modell. Die Python Komponente ist zuständig für die Analyse einzelner Schemaseiten und kann auch unabhängig von der «AI Interlocking» Applikation verwendet werden. Aus diesem Grund wird hier kurz auf das Domain-Modell dieser Komponente eingegangen. Eine analysierte Seite entspricht einem «AnalyzerResult» Objekt. Dieses beinhaltet Metainformationen über die analysierte Datei (Bilddatei) in Form eines «FileHeader» Objektes. Neben diesen Metainformationen werden die detektierten TextBoxen und Symbole vom «AnalyzerResult» referenziert. Da sich die Python Komponente lediglich um die Analyse einer Schemaseite kümmert, sind in diesem Modell noch keine Abhängigkeiten zwischen TextBoxen und Symbolen vorhanden. Die Zuweisung der Symbole und TextBoxen wird erst später in der .NET Komponente durchgeführt. Jedes Symbol und jede TextBox besitzen Koordinaten in Form eines «Rectangle» Objektes. Zudem besitzen «Symbol» Objekte eine Liste von «TextAnchors», die von anderen Programmen oder der «AI Interlocking» Applikation für die Zuweisung der Symbole und TextBoxen verwendet werden kann. Die «TargetSymbolTypeConversion» sind lediglich für «AdditionalInfo» -Symbole relevant, deren genaue Funktionalität später im Dokument erläutert wird.

Sequenzdiagramm

UC-1: Dokument analysieren

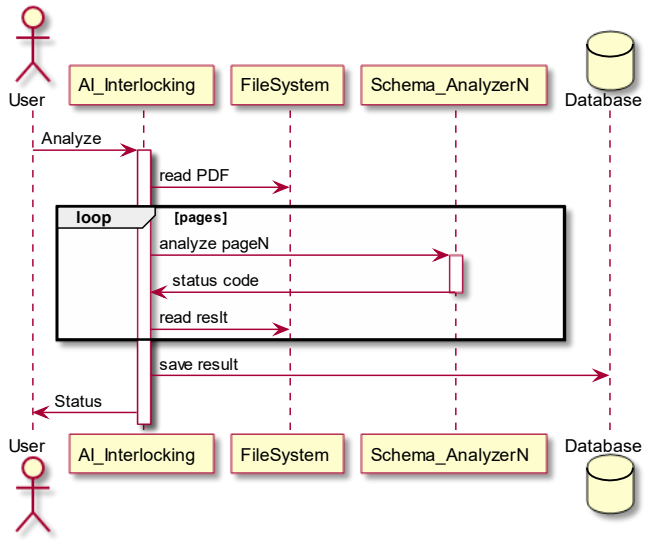


Figure 8 UC-1 Sequenzdiagramm

Im links dargestellten Sequenzdiagramm sind die beiden Teile ersichtlich, welche benötigt werden, um den Analyse Anwendungsfall erfüllen zu können. Dabei soll dieses Sequenzdiagramm lediglich einen Überblick über das System verschaffen und verdeutlichen, warum das Domainmodell in zwei Teile unterteilt worden ist. Beim ersten Teil handelt es sich um das für die Gesamtapplikation zutreffende Domain. Beim zweiten Modell handelt es sich um einen Ausschnitt daraus, der für die Analyse einzelner Seiten benötigt wird. Genauere Informationen zum Aufbau des Systems können dem Kapitel «Architektur» entnommen werden.

UC-2: Analyseresultate exportieren

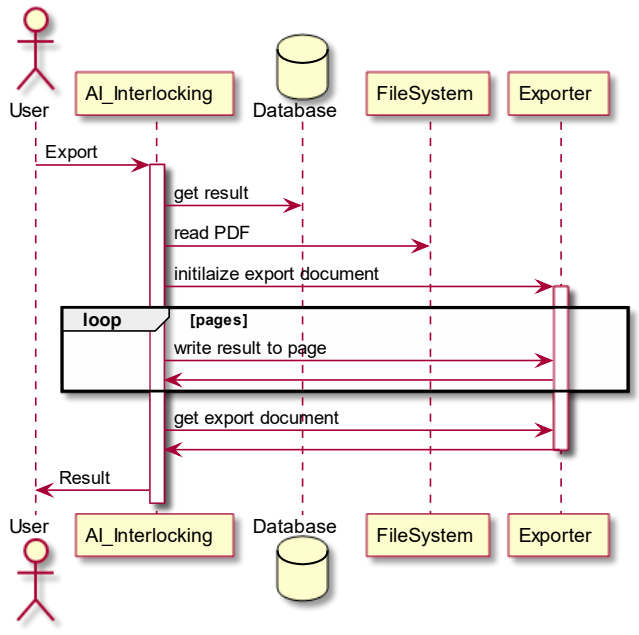


Figure 9 UC-2 Sequenzdiagramm

Im links dargestellten Sequenzdiagramm ist der Ablauf ersichtlich, der benötigt wird, um den Export Anwendungsfall erfüllen zu können. Dabei werden analysierte Anlagedokumentationen aus Dateisystem geladen und die dazugehörigen Analyseresultate aus der Datenbank abgefragt. Die wählbaren Informationen werden anschliessend Seite für Seite sequenziell ins zu exportierende PDF geladen. Wenn das gesamte Analyseresultat zurückgeschrieben wurde, wird das erstellte Dokument dem User zurückgeliefert.

Schnittstellen

Systemarchitektur

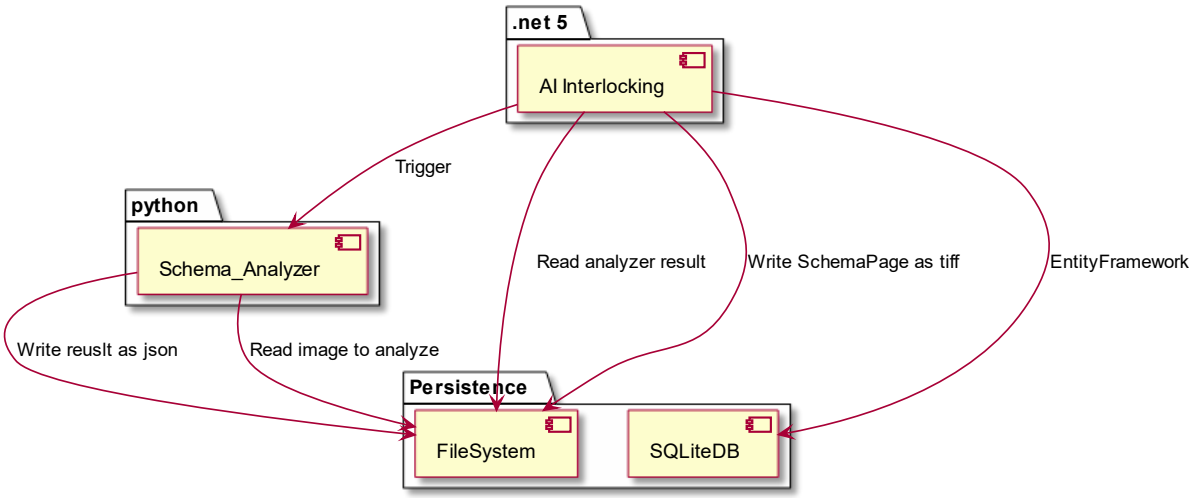


Figure 10 Systemarchitektur Übersicht

Im oben abgebildeten Diagramm ist ersichtlich, wie die einzelnen Komponenten der «AI Interlocking» Applikation interagieren. Es gibt speziell zu erwähnen, dass es sich bei den beiden Komponenten, die das «AI Interlocking» System ausmachen, um technologisch absolut unterschiedliche Implementierungen handelt. Die Python Komponente ist dabei für die eigentliche Analyse einzelner Schemabilder verantwortlich, wogegen sich die .NET Komponente mit der Assoziation von Texten und Symbolen, der Speicherung, Bearbeitung und Exportierung der Resultate sowie der Vorbereitung der PDF-Dokumentation für die Analyse kümmert.
Für eine Dokumentationsanalyse nimmt die .NET Komponente den Pfad der zu analysierende PDF-Datei entgegen und lädt dieses in den Speicher. Das Dokument wird zuerst in einzelne einseitige PDF-Dokument unterteilt. Anschliessend werden diese Dateien in, von der Python Komponente analysierbare Bilddateien im TIFF-Format umgewandelt und im Filesystem abgespeichert. Sobald dieser Umwandlungsschritt abgeschlossen ist, stösst die .NET Komponente die Python Komponente an und übergibt ihr den Pfad zu der jeweiligen Bilddatei. Dabei startet die .NET Komponente für jede Seite einen neuen Python Prozess, in dem der «Schema_Analyzer» ausgeführt wird. Die Python Komponente analysiert jetzt die angegebene Bilddatei und schreibt das Resultat als Datei im JSON Format zurück ins Filesystem. Sobald sich ein zuvor gestarteter Python Prozess beendet hat, lädt die .NET Komponente das Resultat aus dem Filesystem, führt die Assoziationsanalyse durch und schreibt das dabei erhaltene Resultat über das Entity Framework in eine SQLite Datenbank.

«AI Interlocking .NET»

Graphical User Interface (GUI)

In diesem Abschnitt wird auf die einzelnen Views eingegangen, die von der GUI Applikation zur Analyse von PDF-Dokumentationen und Bearbeitung von Analyse Resultaten angeboten werden. Neben der hier beschriebenen graphischen Nutzeroberfläche gibt es auch ein Kommandozeilenprogramm, welches es erlaubt, den Analyseprozess automatisiert aus einem Scheduler zu starten oder ihn in ein anderes Programm oder Skript einzubinden. Auf die CLI Definition wird im nächsten Abschnitt genauer eingegangen.

FileSelectorWindow

Beim unten ersichtlichen Fenster handelt es sich um den Einstiegspunkt der GUI-Applikation. Darin werden die bereits zuvor analysierten Dokumentationen aufgelistet. Diese View bietet die Möglichkeit auf das FileProcessorWindow zu navigieren, um neue Dokumentationen zu analysieren. Zudem besteht die Möglichkeit, bestehende Analyseresultate zu exportieren oder zu löschen. Neben diesen Funktionalitäten können die einzelnen Resultate auch geöffnet werden, um die Analyseresultate aufgeschlüsselt nach Seite anzeigen zu lassen.

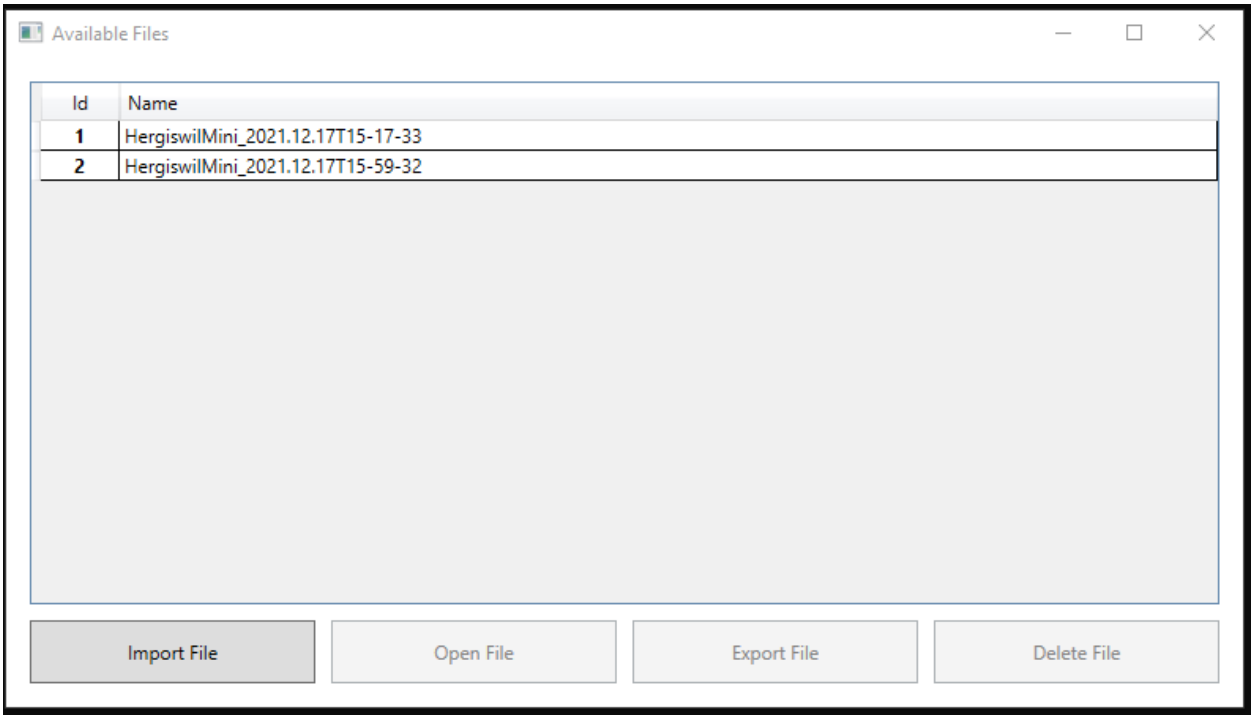


Figure 11 FileSelectorWindow

FileProcessorWindow

Beim unten ersichtlichen Pop-up-Fenster handelt es sich um die View, welche es erlaubt neue Dokumentation zu analysieren. Sie bietet die Möglichkeit, mit dem File Explorer eine PDF-Datei zu selektieren und danach den Analyseprozess zu starten. Es wird jeweils angezeigt, wie viele Seiten insgesamt analysiert werden müssen und wie viele Seiten davon bereits analysiert worden sind. Da der hier angestossene Analyseprozess einige Zeit und Ressourcen in Anspruch nehmen kann, gibt es zudem die Möglichkeit ihn jederzeit abzubrechen.

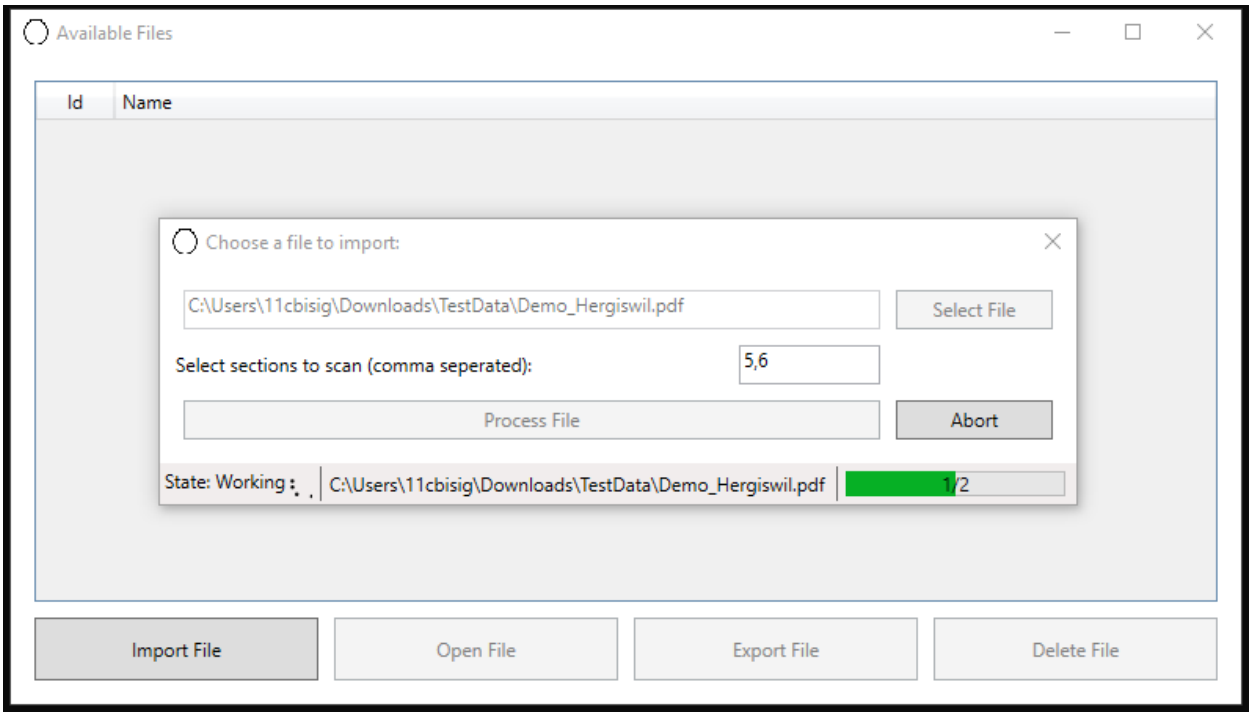


Figure 12 FileProcessorWindow

FileContentWindow

Beim unten ersichtlichen Fenster handelt es sich um die View, welche es erlaubt, ein Analyseresultat aufgeschlüsselt auf die einzelnen Seiten einzusehen. Oben im Fenster wird zudem eine Übersicht über alle gefundenen Elemente gegeben. Der Nutzer hat hier die Möglichkeit, eine Seite zu öffnen, um sie zu bearbeiten. Nach dem Bearbeiten kann der User seine Änderungen speichern, indem er auf den Save-Button klickt. Dabei werden die Datenbankeinträge aktualisiert und die Änderungen stehen in Zukunft zur Verfügung und können somit auch exportiert werden.

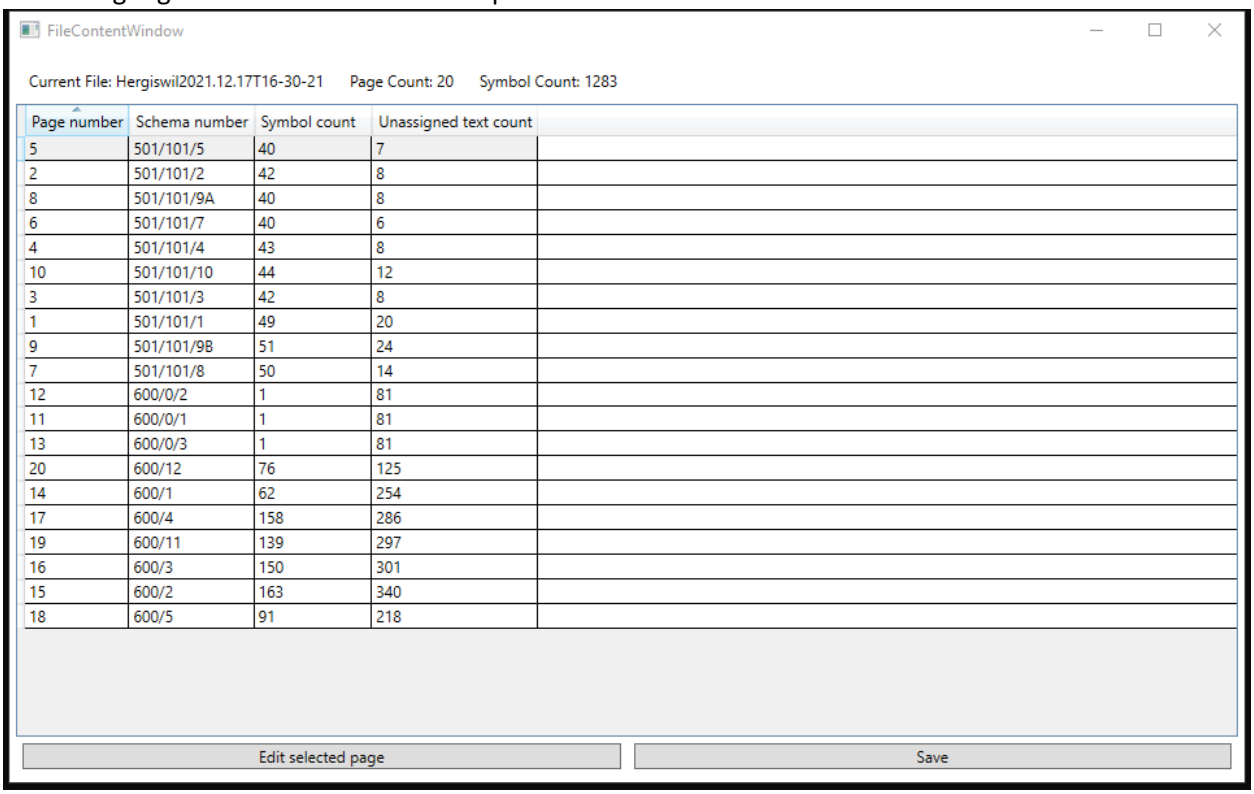


Figure 13 FileContentWindow

PageContentWindow

Das unterhalb ersichtliche Fenster zeigt die gefundenen Symbole mit den ihnen zugewiesenen TextBoxen an. Zudem sind die nicht zugewiesenen Texte ersichtlich und können den gefundenen Symbolen zugewiesen werden. Fälschlich zugewiesene Texte können von einem Symbol entfernt werden und gehören danach wieder zu den «UnassignedTexts» und können somit erneut zugewiesen werden. Falsch erkannte Texte können auf dieser Ansicht verbessert oder gelöscht werden. Mit einem Klick auf den «Visualize» Button kann sich ein User das Analyseergebnis als Bild anzeigen lassen, um allfällige Fehler einfacher entdecken zu können. Nachdem der User Änderungen vorgenommen hat, kann er sich die aktualisierte Version erneut visualisieren lassen.

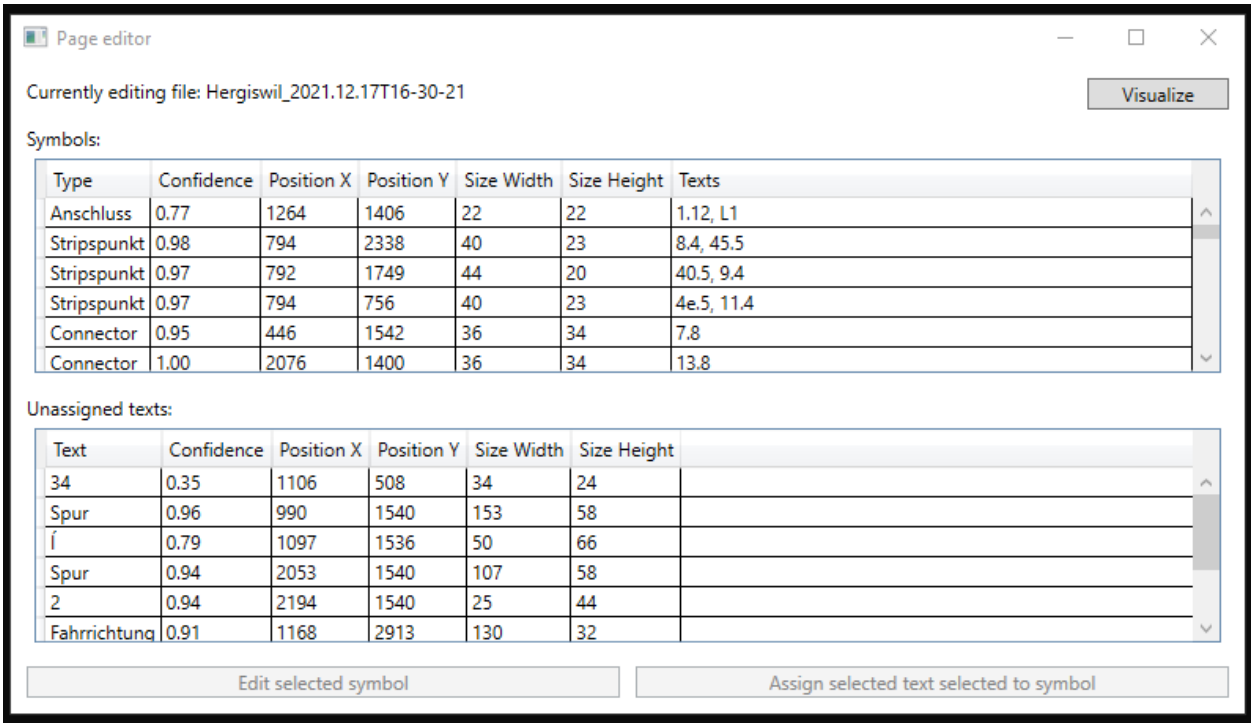


Figure 14 PageContentWindow

FileExporterWindow

Mit dem unten ersichtlichen Pop-up-Fenster können Analyseresultate exportiert werden. Dabei kann ausgesucht werden, welche Informationen im Export enthalten sein sollen (Symbole, Texte, Symbol-Text Zuweisungen, sowie Boxen rund um die erkannten Objekte). Dem User stehen zwei mögliche Formate für den Export zur Verfügung. Zum einen kann das Resultat als PDF-Datei exportiert werden, worin nach den einzelnen Texten gesucht werden kann. Zum anderen besteht die Möglichkeit, das Resultat als Bilddatei zu exportieren. Daraus heraus können beispielsweise zusätzliche Symbol Templates erstellt werden, um die Analyse zu verbessern.

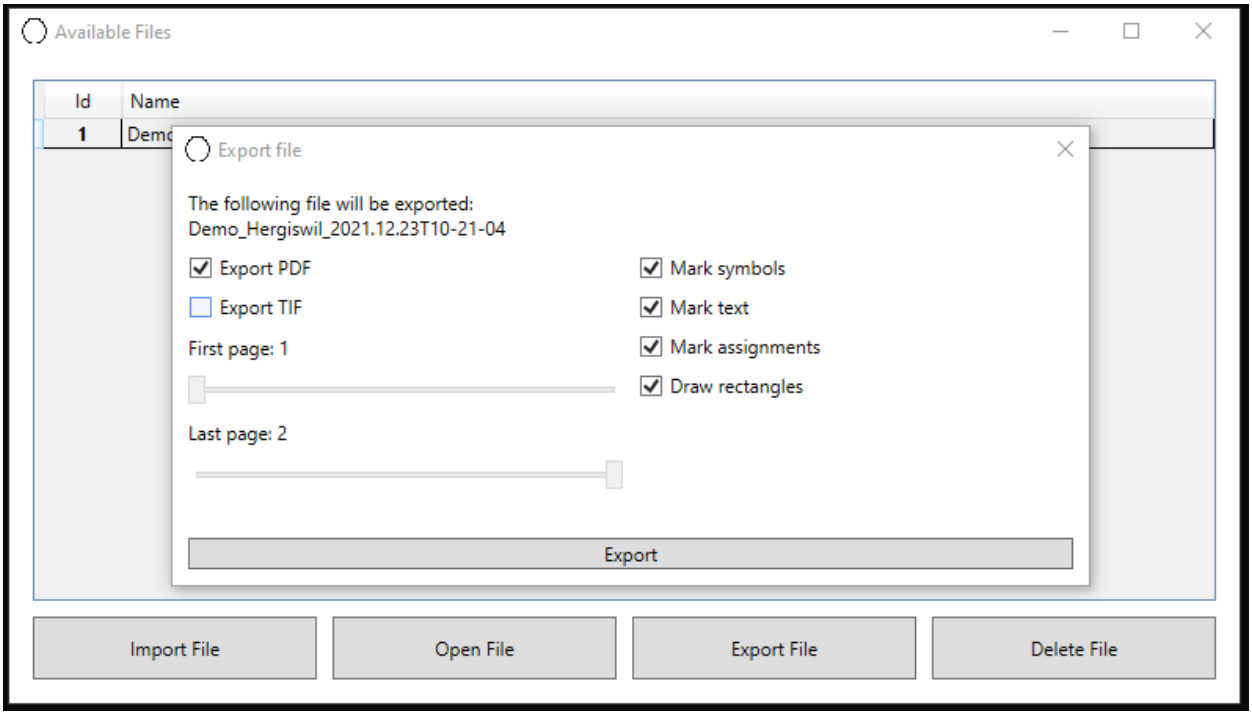


Figure 15 FileExporterWindow

Command Line Interface (CLI)

Wie bereits erwähnt, steht neben dem GUI Programm auch ein CLI zur Verfügung, mit dem der Analyseprozess angestossen werden kann. Dabei muss der Pfad zum zu analysierende PDF im «InputPath» Argument übergeben werden. Daneben besteht die Möglichkeit, Sektionen zu übergeben, die analysiert werden sollen, z.B. 5, um alle 5xx Schemas zu analysieren. Es können mehrere Sektionen angegeben werden, z.B. 4,7. Standardmässig werden alle 5xx und 6xx Dateien analysiert. Dieses Command Line Interface erlaubt es, den Analyseprozess aus einem anderen Programm oder einem Scheduler anzustossen.

```
Interlocking.CLI 1.0.0
Copyright (C) 2021 Interlocking.CLI

  -s, --section         Define the sections that should be analyzed

  --help                Display this help screen.

  --version             Display version information.

  InputPath (pos. 0)    Required. Defines the input file path
```

Figure 16 Analyzer CLI

Erwartetes AnalyzerResult Format

Die .NET Komponente der «AI Interlocking» Applikation erwartet von der Python Komponente eine JSON Datei im unten ersichtlichen Format. Darin sind Informationen zum analysierten Schema sowie alle detektierten Symbole und TextBoxen abgelegt. Im Architektur Teil dieser Dokumentation wird genauer auf die einzelnen Properties eingegangen.

```
{
    "Header": {
        "StationNumber": "",
        "SchemaNumber": "",
        "File": "",
        "Page": "",
        "Type": "",
        "Path": ""
    },
    "Symbols": [],
    "Texts": []
}
```

Figure 17 AnalyzerResult Format

«Schema_Analyzer Python»

Bei der Python Komponente steht ebenfalls ein Command Line Interface (CLI) zur Verfügung. Diese Komponente kann unabhängig von der .NET Komponente betrieben werden, um einzelne Schemabilder zu analysieren. Dafür muss der Dateipfad zur Bilddatei, die analysiert werden soll, als Parameter übergeben werden. Zusätzlich können auch hier Sektionen angegeben werden, die darüber entscheiden, ob ein Schema analysiert wird oder nicht. Wenn beispielsweise -s 6 angegeben wird, aber ein Pfad zu einem 5xx Schema übergeben worden ist, so wird dieses Schemabild nicht weiter analysiert. Es besteht auch bei der Python Komponente die Möglichkeit, mehrere Sektionen zu übergeben. Diese müssen bei diesem CLI kommagetrennt übergeben werden.

```
usage: SchemaAnalyzer.py -f FILE [-s SECTIONS] [--save_res_img] [-h]

optional arguments:
  -h, --help            show this help message and exit
  -f FILE, --file FILE  path to image
  -s SECTIONS, --sections SECTIONS
                        select sections (Klassierungsnummer) e.g. 5,6
  --save_res_img, --no-save_res_img
                        save result as image
```

Figure 18 Schema_Analyzer CLI

Softwarearchitektur

Einführung

Zweck

Dieser Abschnitt soll die grundlegende Architektur des «Ai Interlocking» Systems aufzeigen und erklären. Dabei wird zuerst ein Gesamtüberblick über das System vermittelt, wonach tiefer auf die Architekturen der einzelnen Komponenten eingegangen wird. Zudem werden in diesem Abschnitt die komplexen Abläufe mittels Sequenzdiagrammen aufgezeigt.

Übersicht

Am Anfang dieses Abschnittes kann eine Grobübersicht des Systems gefunden werden. Anschliessend wird auf die Architekturen der einzelnen Komponenten eingegangen, begonnen bei der «Schema_Analyzer» Komponente, die verwendet wird, um einzelne Schemaseiten zu analysieren bis hin zur «Ai Interlocking» Komponente, mit welcher der User interagiert. Bei der Beschreibung der Architektur wurde das C4 Model als Grundlage verwendet. Aus diesem Grund wird nach der Kontextübersicht der einzelnen Container auf deren Komponenten und zum Schluss deren Code eingegangen. Am Ende dieses Dokumentes können komplexe Abläufe visualisiert in Sequenzdiagrammen gefunden werden.

Systemübersicht

Beschreibung

Das «AI Interlocking» System besteht grundsätzlich aus zwei vollständig unabhängigen Komponenten. Zum einen ist dies die .NET Komponente, die sich um die Analyseresultat-Verwaltung kümmert und die Symbol-Text Zuweisungen durchführt. Zudem teilt die .NET Komponente die zu analysierenden PDF-Dateien in einem ersten Schritt in mehrere einseitige PDF-Dokumente auf und wandelt diese in einem zweiten Schritt in analysierbare Bilddateien um. Die Python Komponente auf der anderen Seite wurde dafür konzipiert, einzelne Bilddateien von Schaltplänen zu analysieren. Dazu gehören neben der eigentlichen Analyse, zu welcher die Symboldetektion sowie die Texterkennung zählen, auch die Vorverarbeitung der Bilddateien, um sicherzustellen, dass die Analyseergebnisse so gut wie möglich sind.
Initial wurde geplant, das komplette System im .NET Umfeld umzusetzen. In der Elaborationsphase hat sich bei der genaueren Analyse des Problems, respektive beim Evaluieren von Technologien zur Lösung der Teilprobleme gezeigt, dass dieser initiale Ansatz nicht zielführend sein wird. Es hat sich relativ schnell herauskristallisiert, dass im Python Umfeld umfangreichere und besser zu unseren Ansätzen passende Möglichkeiten vorhanden sind. Nachdem erste Prototypen erfolgreich aufgezeigt haben, dass mit OpenCV eine sehr präzise Symboldetektion sowie eine performante Bildvorverarbeitung möglich ist, wurde das unterhalb ersichtliche System mit den zwei zuvor beschriebenen unabhängigen Komponenten entworfen und genauer spezifiziert. Gestärkt wurde der Entscheid dadurch, dass sich herausgestellt hatte, dass Tesseract, das für die Texterkennung verwendet wird, im Python Umfeld einfacher eingesetzt werden kann und um ein vielfaches mehr Konfigurationsmöglichkeiten bestehen als im .NET Wrapper für Tesseract.
Der Datenaustausch zwischen den beiden Komponenten findet ausschliesslich über Dateien statt,

welche von der einen Komponente ins Dateisystem gespeichert werden und anschliessend von der anderen Komponente ebenda wieder gelesen werden.
Die nahezu vollständige Entkopplung der Komponenten, welche mit diesem Ansatz erzielt werden konnte, führt dazu, dass beide Komponenten für sich alleine verwendet werden können und somit auch einfach ersetzt werden könnten. Wenn beispielsweise in Zukunft eine erweiterte Schemaanalyse durchgeführt werden sollte, könnte die Python Komponente einfach durch eine neue Komponente, die sich an die geforderten Dateiformate hält, ersetzt werden, ohne Änderungen an der .NET Komponente vornehmen zu müssen. Zudem kann mit dieser Architektur die Python Komponente von anderen Programmen verwendet werden, um Schemabilder aus anderen Quellen zu analysieren.

Diagramm

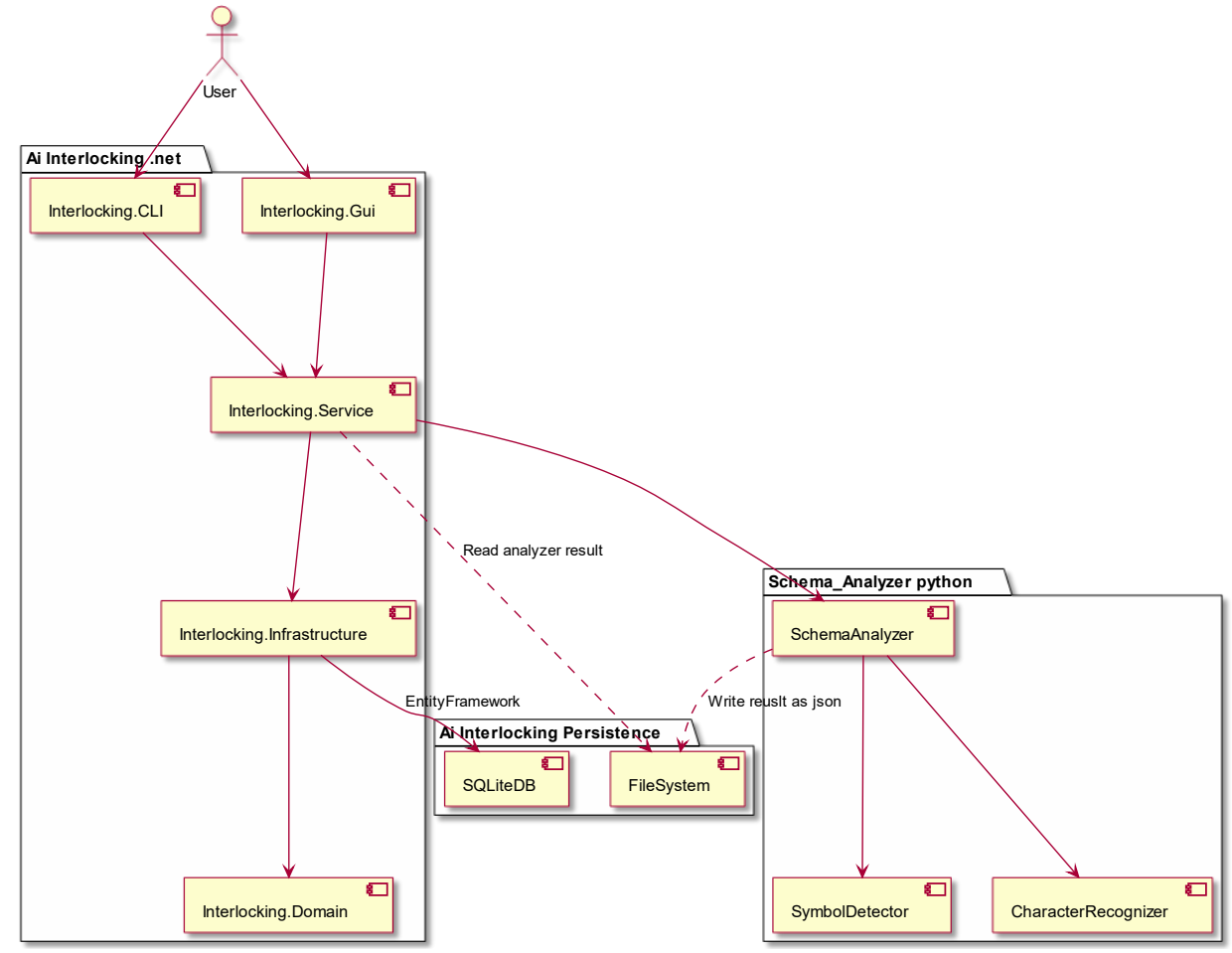


Figure 19 Architekturübersicht

Architektonische Ziele und Einschränkungen

Ziel der Architektur war es, ein möglichst vielseitig einsetzbares System zu erstellen, in welchem die Komponenten lediglich eine geringe Kopplung aufweisen. Auf die Architekturen innerhalb der einzelnen Komponenten wird in den nachfolgenden Abschnitten noch genauer eingegangen.
Ein weiteres Ziel der geplanten Architektur war eine Möglichkeit zu finden, die Analyse der einzelnen Seiten parallelisieren zu können. Die gewählte Architektur erfüllt dieses Ziel vollumfänglich, da vom .NET Teil für jede zu analysierende Seite ein Python Prozess gestartet werden kann. Diese Prozesse laufen komplett unabhängig und können somit auch parallel ausgeführt werden. Als Einschränkung dieser hier verwendeten Architektur muss der Overhead erwähnt werden, der erzeugt wird, wenn für jeden Schemabild-Analyseprozess ein neuer Prozess gestartet werden muss. Dazu kommt, dass der Datenaustauch über das Dateisystem gegebenenfalls zu Abstrichen in der Performanz führen kann. Diese Einschränkungen wurden in der Analyse genauestens gegen die Vorteile abgewogen, wobei sich gezeigt hat, dass die Vorteile die Nachteile ganz klar übertreffen.

Parallelisierungsmöglichkeit

Das unten ersichtliche Diagramm zeigt die Möglichkeit zur Parallelisierung der Seitenanalyse auf. Dabei werden von der .NET Komponente mehrere Python-Prozesse gleichzeitig gestartet, in welchen jeweils eine «Schema_Analyzer» Instanz ausgeführt wird, die jeweils eine spezifische Seite analysiert und anschliessend das Resultat als JSON Datei im Dateisystem ablegt. Die .NET Komponente liest diese JSON Dateien, sobald sich ein Python Prozess beendet hat und Kapazität zur weiteren Verarbeitung des Resultates vorhanden ist. Diese Parallelisierungsmöglichkeit bringt Gewinne im Bereich der Performanz mit sich, da es sich beim Analyseprozess der einzelnen Seiten um den zeitaufwändigsten Teil der gesamten Dokumentationsanalyse handelt.

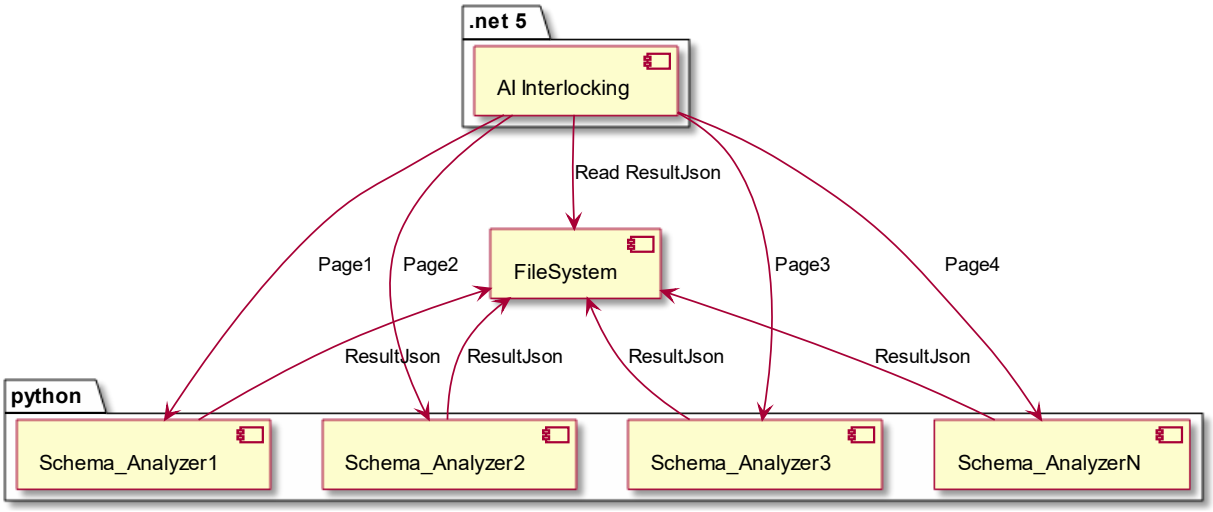


Figure 20 Parallelisierungsübersicht

Logische Architektur «AI Interlocking»

In diesem Abschnitt dieser Dokumentation wird auf die zuvor erwähnte .NET Komponente eingegangen. Dabei wird gemäss C4 Modell zuerst auf einem hohen Abstraktionslevel ein Überblick über die Komponente verschafft. Anschliessend wird die Abstraktionsebene stetig verkleinert, bis Informationen über die Klassenarchitektur der einzelnen Subkomponenten ersichtlich werden.

Grundlegende Architekturentscheide

Bei der für diese Komponente verwendeten Architektur wurde auf einer «Clean Architecture» aufgebaut. Dabei bildet das Interlocking.Domain Projekt der Kern der Komponente. Die Interlocking.Service Schale beinhaltet die gesamte Businesslogik, von der Vorbereitung der Dokumentation für die Analyse bis hin zur Zuweisung der TextBoxen zu den Symbolen. Der unten gelb gefärbte Teil wird im Projekt durch das Interlocking.Infrastructure Projekt widerspiegelt und beinhaltet Technologiespezifische Logik, wie den Datenbankkontext für die aktuell verwendete SQLite Anbindung über EF Core. Im roten Teil der äusserten Schale können die beiden Projekte gefunden werden, die dem Nutzer für die Interaktion mit der Applikation zur Verfügung stehen. Der hier gewählte Ansatz bringt eine grosse Entkopplung der einzelnen Teilprojekte mit sich und erlaubt das einfache Austauschen von schnelllebigen Technologien, beispielsweise im Bereich des Userinterfaces. Auch der Umstieg auf ein anderes Produkt zur Datenspeicherung wäre mit der gewählten Architektur kein Problem. Bei solchen Änderungen müssten die inneren Schalen jeweils nicht angepasst werden und sind somit offen für Erweiterungen, ohne sie abändern zu werden müssen. Diese Architekturgrundlage erleichtert auch das Testen einzelner Komponenten.

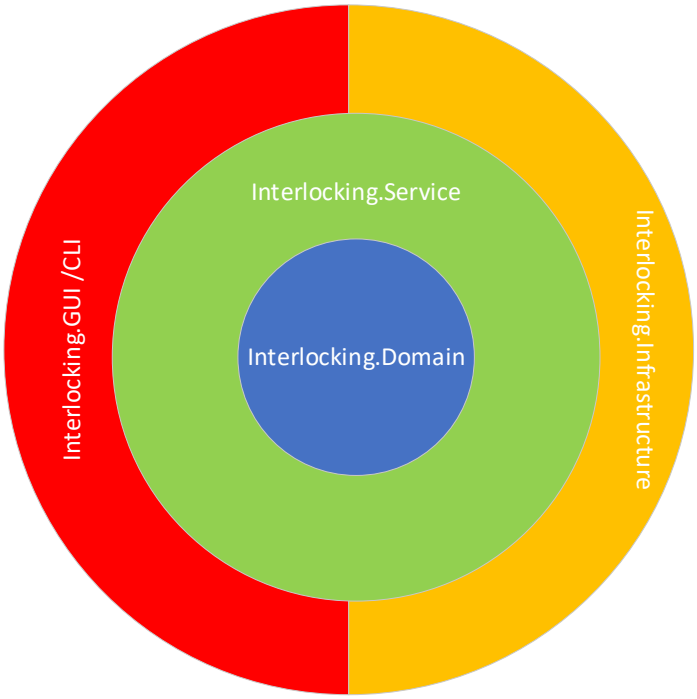


Figure 21 Clean Architecture Schema

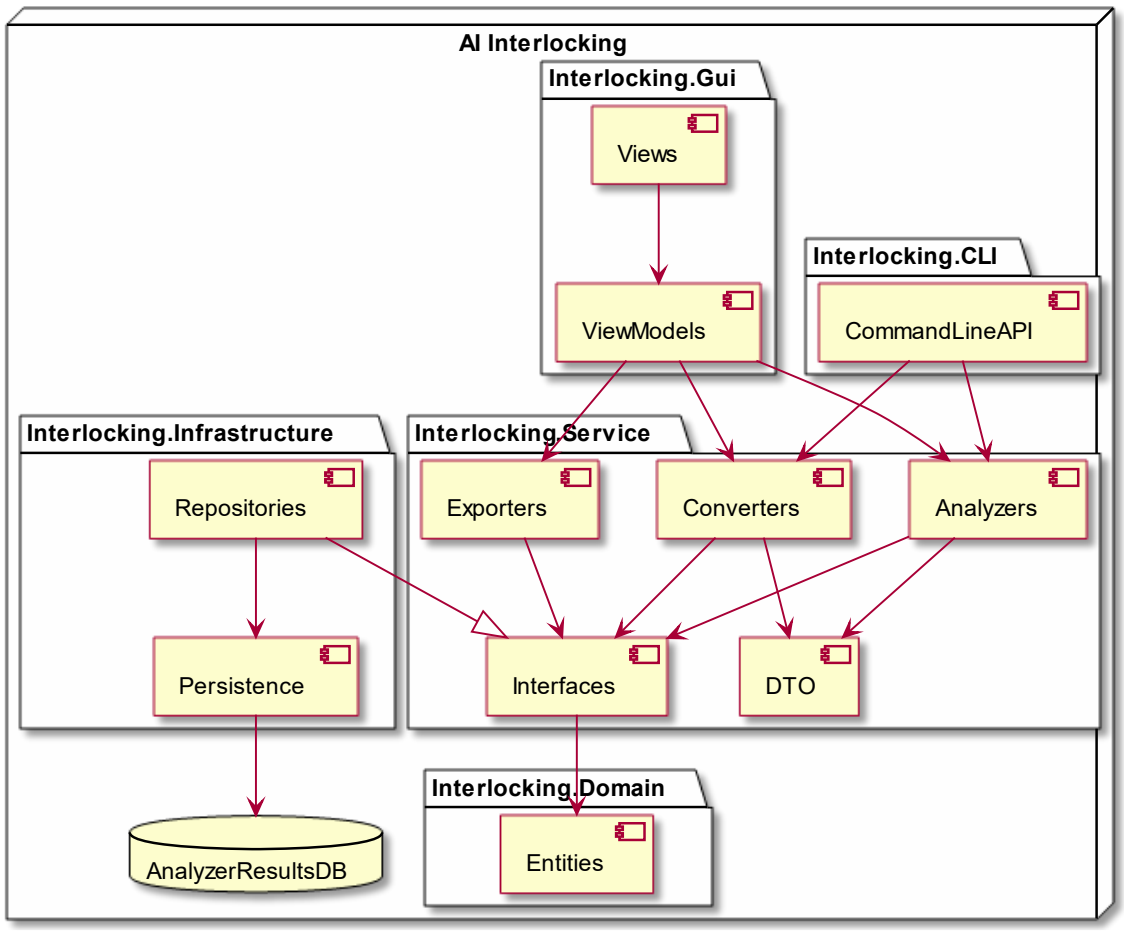


Figure 22 .NET Komponente Paketübersicht

Im oben abgebildeten Diagramm sind die einzelnen Teilprojekte der .NET Komponente ersichtlich. Die oberste Schicht wird gebildet durch das Interlocking.Gui, sowie das Interlocking.CLI Projekt, alle Teilprojekte, mit denen der User interagiert. Im Gui Projekt wird unterschieden zwischen den Views und den daran gebundenen ViewModels, die sich um die Propagierung von Änderungen kümmern und jeweils das UI aktualisieren. Beim Interlocking.Gui Projekt wurde auf der MVVM (Model View ViewModel) Architektur aufgebaut. Das Interlocking.Gui und Interlocking.CLI Teilprojekt greifen jeweils nur auf das Interlocking.Service Teilprojekt zu und arbeiten so mit der Businesslogik der Applikation. Die Businesslogik wurde in die drei sich im Usecase unterscheidenden Teile Exporters, Converters sowie Analyzers unterteilt. Diese Logikkomponenten arbeiten nur gegen Interfaces und sind somit unabhängig von den technologiespezifischen Implementationen im Interlocking.Infrastructure Teilprojekt. Neben den Entity Modellen, die vom Interlocking.Service Projekt verwendet werden, gibt es auch noch DTO Modelle zur Kommunikation mit der Python Komponente. Im Interlocking.Infrastructure Teilprojekt, wurde auf das Repository Pattern gesetzt, um den Datenzugriff zu abstrahieren und somit einfacher testbar zu machen und die Kopplung im Projekt niedrig zu halten.
In den nachfolgenden Abschnitten wird genauer auf die Klassenarchitektur innerhalb dieser einzelnen Teilprojekte eingegangen.

«AI Interlocking» Userinteraction-Layer

Im unten ersichtlichen Diagramm ist zu erkennen, wie das Zusammenspiel der Klassen innerhalb der beiden Userinteraction-Layer Teilprojekten aufgebaut ist. Zudem ist ersichtlich, auf welche Komponenten aus der nächsttieferliegenden Schicht zugegriffen wird. Wie bereits erwähnt, wurde im Interlocking.Gui Projekt auf dem MVVM Pattern aufgebaut, um die Daten ans Userinterface zu binden. Das führt dazu, dass jede View Klasse eine dazugehörige Viewmodel Klasse besitzt, welche sich um das Propagieren von Änderungen und somit das Aktualisieren der View kümmert. Daneben wird in den Viewmodels auf Userinteraktionen reagiert und die entsprechende Businessaktion im Interlocking.Service Projekt angestossen.

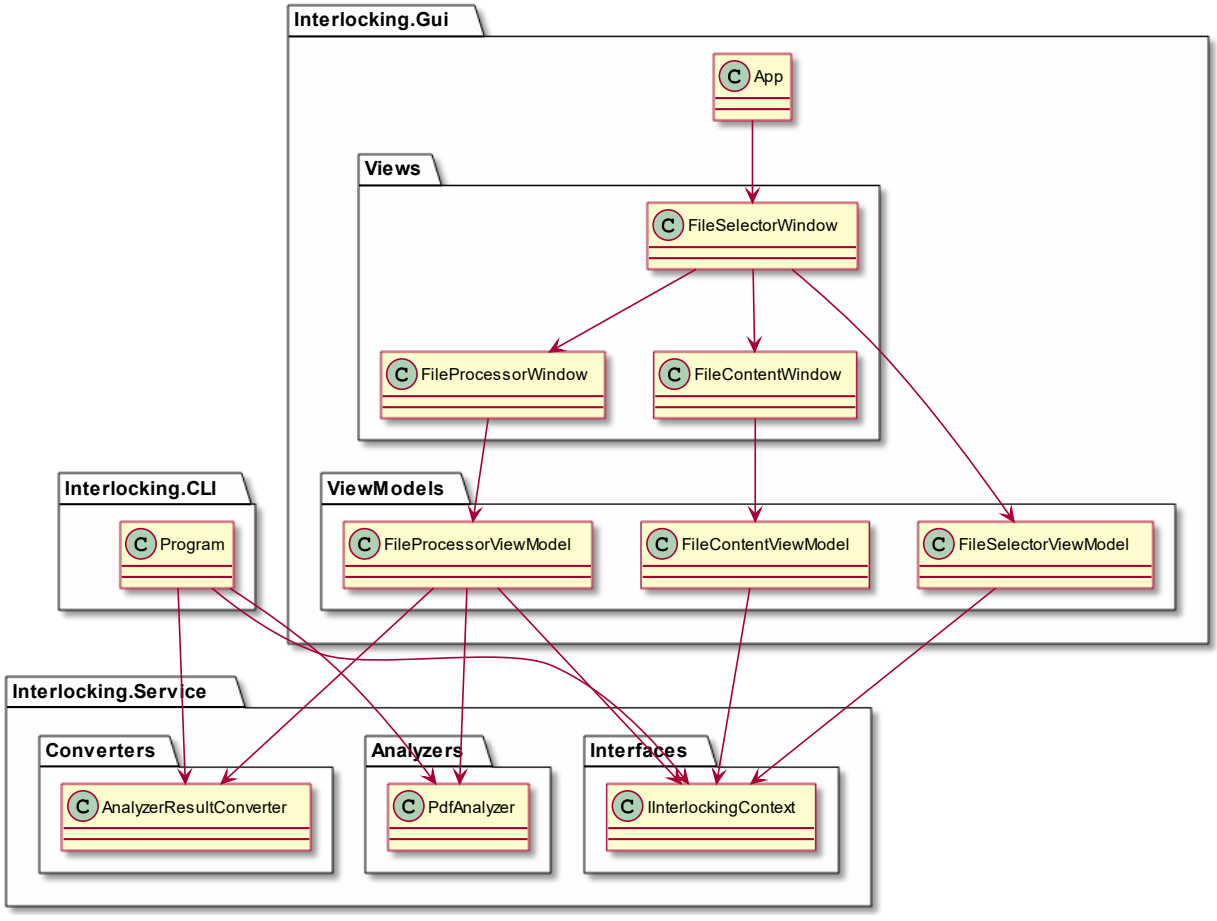


Figure 23 .NET Komponente Userinteraction-Layer Klassenübersicht

«AI Interlocking» Service Layer

Dem unten ersichtlichen Diagramm können die Klassen des Interlocking.Service Projekt entnommen werden. Die Businesslogik wurde wie bereits erwähnt in die drei sich im Anwendungsfall unterscheidenden Komponenten Analyzers, Exporters und Converters unterteilt. Die Analyzer Logik beinhaltet die Funktionalität eine PDF-Dokumentation zu analysieren. Dazu gehören das Aufteilen und Umwandeln der PDF-Dateien in Bilddateien, das Anstossen der Python Komponente zur Analyse einzelner Seiten sowie die Assoziation von TextBoxen und Symbolen. Für die Kommunikation mit der Python Komponente, welche über Dateien im Dateisystem funktioniert, werden die DTO Klassen benötigt. Die PredictionReader Klasse liest das vom Schema_Analyzer geschriebene Resultat aus der JSON Datei und gibt diese dem PdfSchemaPageAnalyzer Objekt zurück. Dieses führt die Zuweisung der TextBoxen und Symbolen aus und gibt das Resultat der PdfAnalyzer Klasse zurück, das weitere Analysen basierend auf den erhaltenen Resultaten ausführt.
Die Exporters Businesslogik beinhaltet die Funktionalität ein vollständiges Analyseresultat als durchsuchbares PDF oder als Bilddateien zu exportieren. Damit können die Analyseresultate visualisiert und weiterverarbeitet werden. Der jeweilige Exporter lädt die Resultate über den InterlockingContext aus der Datenbank und verarbeitet diese anschliessend weiter.
Die Converters Businesslogik ist dafür zuständig gefundene Symbole in vordefinierte Spezialsymbole wie Anschlüsse, Relais oder Relaiskontakte umzuwandeln. Die AnalyzerResultConverter Klasse ist dafür zuständig die DTOs, die zuvor aus der JSON Datei geladen wurden in Domain Entities umzuwandeln, damit diese anschliessend in der Datenbank abgelegt werden können.

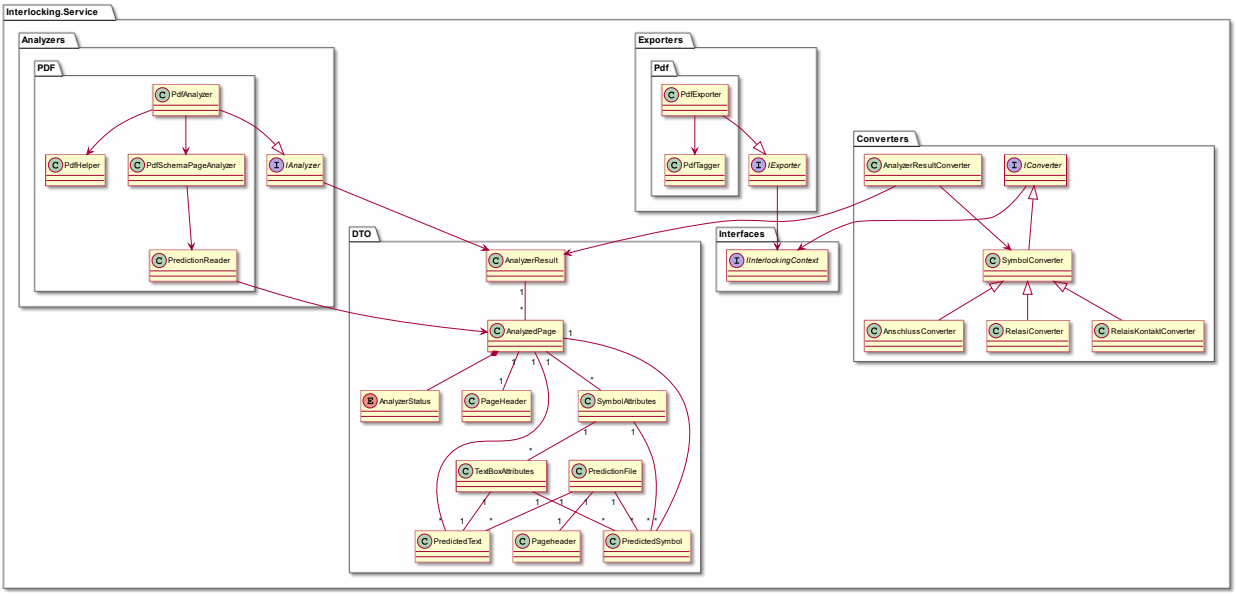


Figure 24 .NET Komponente Service Layer Klasssenübersicht

PDF Analyzer

In diesem Abschnitt wird auf die Funktionalität der PDF Analyzer Businesslogik eingegangen. Dabei werden die einzelnen Schritte, die nötig sind, um eine PDF-Dokumentation zu analysieren, in ihrer chronologischen Reihenfolge abgearbeitet und genauer erklärt.

Kapitel Selektion

Es wird die Möglichkeit geboten, die zu analysierenden Kapitel zu selektieren. Wenn beispielsweise nur die 5xx Schemas analysiert werden sollen, kann der zugehörige Parameter «Section» auf 5 gesetzt werden. Für die Unterscheidung der Kapitel stehen zwei Möglichkeiten zur Verfügung, die sich in ihrer Performanz unterscheiden.

Variante 1: Kapitelselektion basierend auf Bookmarks

Der performantere Ansatz basiert auf Bookmarks, die in der PDF-Dokumentation vorhanden sein können. Dabei wird die Bookmark Hierarchie abgearbeitet und nach einer Ebene gesucht, auf der alle gewünschten Kapitel genau einmal vorhanden sind. Die Bookmarks werden mit dem folgenden Regex Pattern verglichen: `"{section}([0-9]|x){2}($|-|/)"`
Wird ein Layer gefunden, das für alle Kapitel Patterns passt, wird der Abschnitt zwischen dem passenden Bookmark und dem nachfolgenden Bookmark extrahiert und analysiert. Der restliche Teil der Dokumentation muss dabei nicht in Bilddateien umgewandelt und vom «Schema_Analyzer» analysiert werden. Dies bringt grosse Gewinne im Bereich der Performanz mit sich.
Unten ersichtlich ist ein Ausschnitt der Bookmarks aus einer Anlagedokumentation, deren Format passen würde und somit diese Variante der Kapitelselektion verwendet werden würde.

Figure 25 Sectionbookmarks

Variante 2: Kapitelselektion basierend auf Schemanummern

Die Fallbackvariante funktioniert basierend auf der Blattnummer, welche im Footer jeder Dokumentationsseite vermerkt ist. Bei dieser Variante wird jede Seite der Anlagedokumentation in eine Bilddatei umgewandelt und einer «Schema_Analyzer» Instanz (Python Komponente) zur Analyse übergeben. Bevor das eigentliche Schema analysiert wird, findet eine Footer Analyse statt. Die Python Komponente trennt dafür den Footer vom Rest der Seite und führt darauf eine Texterkennung durch (Informationen zu dieser Funktionalität kann dem Kapitel «Schema_Analyzer Python» entnommen werden). Die detektierten Texte werden gegen das folgende Regex Pattern verglichen, um die Blattnummer zu detektieren: `"[0-9]{3}(/|-)[0-9]+((/|-)[0-9]+)?(/|-)?[0-9]*"` . Basierend auf der ersten Ziffer der detektierten Blattnummer wird dann entschieden, ob diese Seite komplett analysiert werden soll oder nicht.

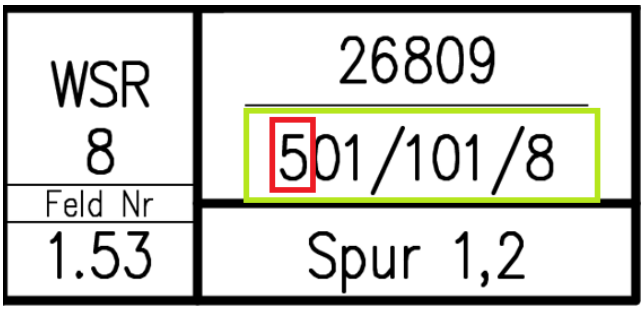


Figure 26 Blattnummer Sektionsauswahl

Documentation Splitting – Aufteilen der Datei in einseitige PDF-Dateien

Die Python Komponente analysiert Seite für Seite. Aus diesem Grund muss eine zu analysierende Seite in einem ersten Schritt aus der Anlagedokumentation extrahiert werden und in ein einseitiges PDF-Dokument ausgelagert werden. Wenn Variante 1 der Kapitelselektion zum Einsatz kommt, werden nur diejenigen Seiten extrahiert, die in den selektierten Kapiteln vorhanden sind und analysiert werden sollen. Für die PDF-Verarbeitung wurden diverse Bibliotheken evaluiert (itext7, Spire.Pdf, PdfSharpCore, Aspose.PDF, IronPdf). Dabei war das Ziel eine Bibliothek zu finden, welche die benötigte Funktionalität wie einzelne Seiten zu exportieren, Seiten in Bilder umzuwandeln, auf existierende Seiten zu schreiben oder diese mit Tags zu versehen zur Verfügung stellt. Bei der genaueren Evaluation hat sich gezeigt, dass es sich bei den meisten der oben genannten Libraries um kommerzielle Produkte handelt, was Lizenzkosten mit sich bringen würde. Zudem hat sich beim Prototypisieren gezeigt, dass keine der genannten Libraries die gesamte benötigte Funktionalität zur Verfügung stellt. Mit dem Ausschlussverfahren wurde PDFsharp als beste verfügbare Möglichkeit erkoren. PDFsharp ist ein Open Source Projekt, das unter der MIT Lizenz frei verwendet werden kann. Nachteil dieser Wahl ist, das PDFsharp im Gegensatz zu anderen getesteten Bibliotheken nicht mit Korrupten PDF-Dokumenten umgehen kann. Korrupte Anlagedokumentation können also nicht direkt analysiert werden, es gibt jedoch die Möglichkeit diese zuerst in einem PDF-Reader zu öffnen und eine standartkonforme Kopie des Dokumentes abzuspeichern, welches dann analysiert werden kann.

PDF – Image Konvertierung

Nachdem die zu analysierende Seite in ein einseitiges PDF-Dokument exportiert worden ist, muss diese in eine analysierbare Bilddatei umgewandelt werden. Da PDFSharp die Funktionalität zum Konvertieren der Seiten in Bilddateien standardmässig nicht im geforderten Format unterstützt, musste eine zusätzliche Bibliothek evaluiert werden. Bei der Evaluation verschiedener Produkte hat sich gezeigt, dass Magick.NET, ein .NET Wrapper für ImageMagick, die passendste Möglichkeit ist. Magick.NET ist unter der Apache-2.0 Lizenz frei verfügbar und kann somit problemlos eingebunden werden. Die Bibliothek verwendet intern, wie alle anderen evaluierten Produkte, Ghostscript zur Interpretation von PDF-Seiten. Da Ghostscript mit der «AI Interlocking» Applikation ausgeliefert wird, muss gemäss GNU AGPL Lizenzbedingungen die Software frei verfügbar gemacht werden oder als Alternative eine kommerzielle Lizenz von Artifex vorhanden sein. Die PDF-Seiten werden im TIFF Format mit 300 DPI exportiert, da diese Konfiguration im Test mit der Python Komponente, dem «Schema_Analyzer», die besten Ergebnisse geliefert hat. Unterhalb ist ein Beispiel einer exportierten Schemaseite, wie sie von der «AI Interlocking» Applikation erstellt wird, ersichtlich.

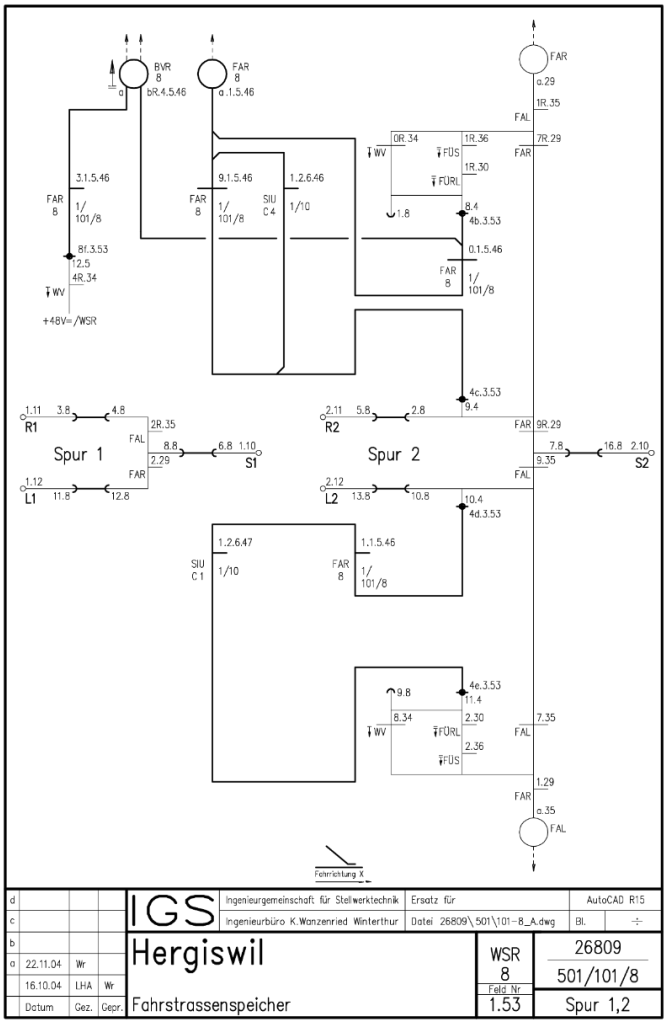


Figure 27 Konvertiertes Schemabild

Symbol – Text Zuweisung

Nachdem die «PredictionReader» Klasse das Resultat von der Python Komponente eingelesen hat, werden die detektierten Symbole und TextBoxen einander zugwiesen. Für jedes Symbol können Distanzgrenzen (AssignmentDistanceThreshold) und sogenannte TextAnchors definiert werden. Standardmässig wird ein TextAnchor im Zentrum des Template-Bildes festgelegt. Von den TextAnchors aus werden die Distanzen zu den umliegenden TextBoxen berechnet und wenn diese Distanz kleiner als die ebenfalls definierte Distanzgrenze ist, wird diese TextBox als zuweisbar markiert. Zum Schluss werden die TextBoxen jeweils dem zuweisbaren Symbol zugewiesen, welches die kleinste Distanz zur TextBox aufweist. Unterhalb ist diese Zuweisungslogik schematisch dargestellt. Die grünen Boxen stellen die detektierten Texte dar, die roten Punkt widerspiegeln die TextAnchors und die gelben Kreise werden vom AssignmentDistanceThreshold gebildet. Alle grünen Rechtecke, die einen Punkt innerhalb oder auf dem gelben Kreis haben, kommen also als zuweisbar infrage.

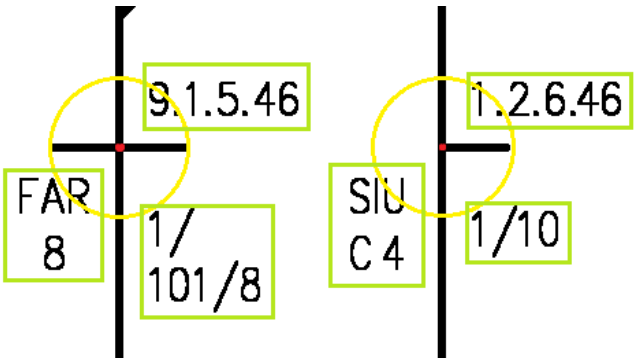


Figure 28 TextBox-Symbol Zuweisung

Unterhalb ist beispielhaft dargestellt, wie sich die Zuweisungslogik verhält, wenn mehrere TextAnchors für ein Symbol definiert worden sind. Die roten Punkte bilden dabei wieder die TextAnchors, der AssignmentDistanceThreshold kann so viel kleiner gewählt werden, als wenn wie standardmässig nur ein TextAnchor in der Mitte des Symbols definiert worden wäre. Diese Vorgehensweise bringt den Vorteil mit sich, dass viel genauer definiert werden kann, wo mögliche zuweisbare TextBoxen vorhanden sein könnten.

Figure 29 Mehrfach TextAnchor Zuweisung

Spezialsymbol Zuweisung

Neben den TextBoxen, welche den Symbolen zugewiesen werden können, gibt es sogenannte AdditionalInfoSymbols, welche Zusatzinformation zu Texten oder Symbolen mit sich bringen. Für jedes dieser AdditionalInfoSymbols kann definiert werden, welchen Symboltypen sie zugewiesen werden können und ob sie auch TextBoxen zugewiesen werden können. Genauere Informationen zur Konfiguration dieser Properties können dem Kapitel «Schema_Analyzer Python» entnommen werden. Wie bereits bei den TextBoxen zuvor, wird die Distanz zwischen den AdditionalInfoSymbols und den möglichen Zuweisungspartnern berechnet. Als zuweisbar gelten hier nur Objekte (Symbole/TextBoxen) bei welchen das AdditionalInfoSymbol innerhalb des AssignmentDistanceThreshold liegt. Bei TextBoxen ist dieser Threshold standardmässig auf 50px gesetzt, bei den Symbolen kann er frei konfiguriert werden. Das AdditionalInfoSymbol wird dem zuweisbaren Objekt mit der geringsten Distanz zugewiesen. Unterhalb sind schematisch drei Beispiele ersichtlich, bei denen die blauen AdditionalInfoSymbols verschiedenen Objekten zugewiesen worden sind. Im linken Beispiel beziehen sich die AdditionalInfoSymbols auf die TextBoxen, neben welchen sie stehen und in den rechten beiden Beispielen hingegen auf die Symbole.

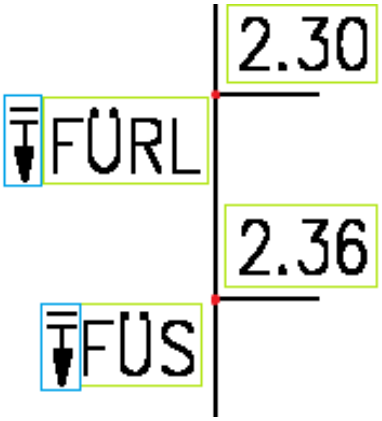


Figure 30 Additional Info TextBox

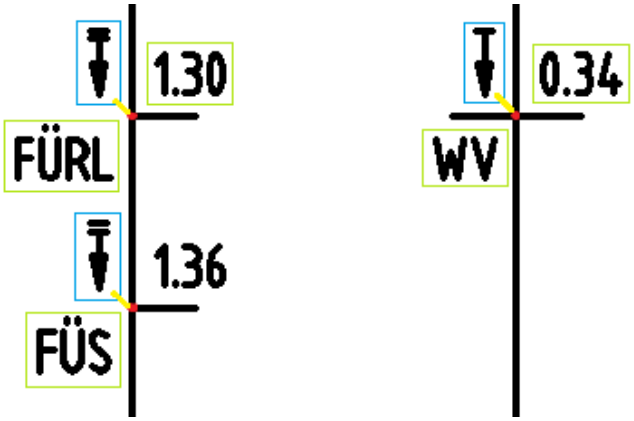


Figure 31 Additional Info Symbol

Algorithmus zur Minimierung des nicht zugewiesenen Textes

Da es vorkommen kann, dass TextBoxen von der Texterkennungslogik willkürlich unterteilt werden, zum Beispiel weil die Abstände zwischen den einzelnen Ziffern nicht immer genau gleich sind, wurde eine Logik entwickelt, um die fälschlicherweise getrennten TextBoxen wieder zu vereinen. Im unten ersichtlichen Beispiel ist zu erkennen, dass die orangen Boxen [.46] sowie [C4] nicht dem Symbol zugewiesen werden können, da sie ausserhalb des AssignmentDistanceThresholds (gelber Kreis) liegen. Diese beiden Texte werden somit fälschlicherweise zu «Unassigned Text». Um diesen Fehler zu beheben, wird zum Schluss der Zuweisungsanalyse für alle nicht zugewiesenen TextBoxen geprüft, ob diese auf der gleichen Zeile oder in der gleichen Spalte sind, mit einer zugewiesenen TextBox zu welcher die Distanz weniger als 25px beträgt. Wird eine solche TextBox gefunden, wird die nicht zugewiesene TextBox dem gleichen Symbol zugewiesen, wie die gefundene TextBox. Somit werden die unten orangen gefärbten Boxes nach dieser Analyse korrekterweise auch dem zugehörigen Symbol zugewiesen.

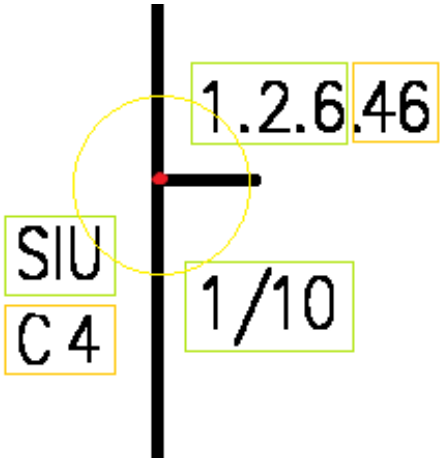


Figure 32 Minimierung von nicht zugewiesenem Text

Converters

Für bestimmte Symboltypen wie Relais, Relaiskontakte oder Anschlüsse wurden Converter implementiert. Diese haben die Aufgabe aus den, dem Symbol zugewiesenen TextBoxen, spezielle Properties zu definieren wie beispielsweise den Identifier oder einen Purpose. Im folgenden Abschnitt wird auf die aktuell verfügbaren Converter eingegangen und deren Funktionsweise erläutert. Wenn der detektierte Symboltyp keinem der unten spezifizierten Converter zugeteilt ist, kommt der SymbolConverter zum Einsatz. Dieser konvertiert die detektierten DTO Objekte in Domain Entity Symbol Objekte und weist ihnen die zugewiesenen Texte zu. Der SymbolConverter ist die Basisklasse aller unten aufgeführten Spezial-Converter. Diese führen lediglich zusätzlich zur Basislogik noch zusätzliche Operationen durch.

RelaisConverter

Wenn der Typ eines detektierten Symbols «Einzelrelais» ist, wird der RelaisConverter verwendet. Dieser sucht aus den, dem Symbol zugewiesenen TextBoxen zum einen, einen Identifier welcher mit dem folgenden Regex Pattern übereinstimmt: "(\w+\.)(\d+\.)*(\d+)" . Zum anderen sucht der RelaisConverter mithilfe einer Liste von möglichen Relaisbezeichnungen ein Purpose Property. Dabei kommt der sogenannte FuzzyWuzzy Algorithmus zum Einsatz, welcher versucht die ähnlichste mögliche Bezeichnung zu ermitteln. Dies hat zum Vorteil, dass nicht ganz korrekt erkannte Bezeichnungen auf die bestpassendste mögliche Bezeichnung geändert und somit korrigiert werden. Als mögliche Kandidaten für dieses Purpose Property kommen TextBoxen infrage, deren Text zu mindestens 90% mit einer möglichen Relaisbezeichnung übereinstimmt. Genauer Informationen zum verwendeten Algorithmus und den möglichen Bezeichnungen können dem Abschnitt «Mögliche Relais-Bezeichnungen» entnommen werden.

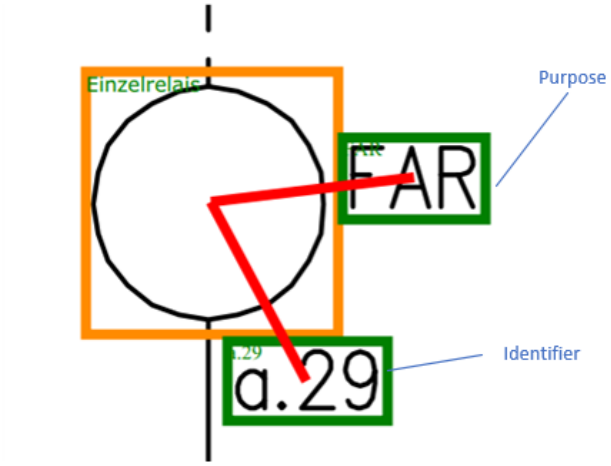


Figure 33 Relais-Symbol Konvertierung

RelaisKontaktConverter

Der RelaisKontaktConverter kommt zum Einsatz, wenn der Typ eines detektierten Symbols «Oeffner» oder «Schliesser» ist. Auch hier wird, wie bereits beim RelaisConverter, in den zugewiesenen TextBoxen nach einem Identifier und einem Purpose gesucht. Der Identifier wird der TextBox gleichgesetzt, deren Text mit dem folgenden Regex Pattern übereinstimmt: `"(\w+\.)(\d+\.)*(\d+)"` . Der Purpose wird wiederum mit der Liste an möglichen Bezeichnungen unter Einsatz des FuzzyWuzzy Algorithmus ermittelt. Als mögliche Kandidaten für dieses Purpose Property kommen TextBoxen infrage, deren Text zu mindestens 85% mit einer möglichen Relaisbezeichnung übereinstimmt. Die TextBox mit der grössten Übereinstimmung wird zum Schluss als Purpose gesetzt. Genauer Informationen zum verwendeten Algorithmus und den möglichen Bezeichnungen können dem Abschnitt «Mögliche Relais-Bezeichnungen» entnommen werden.

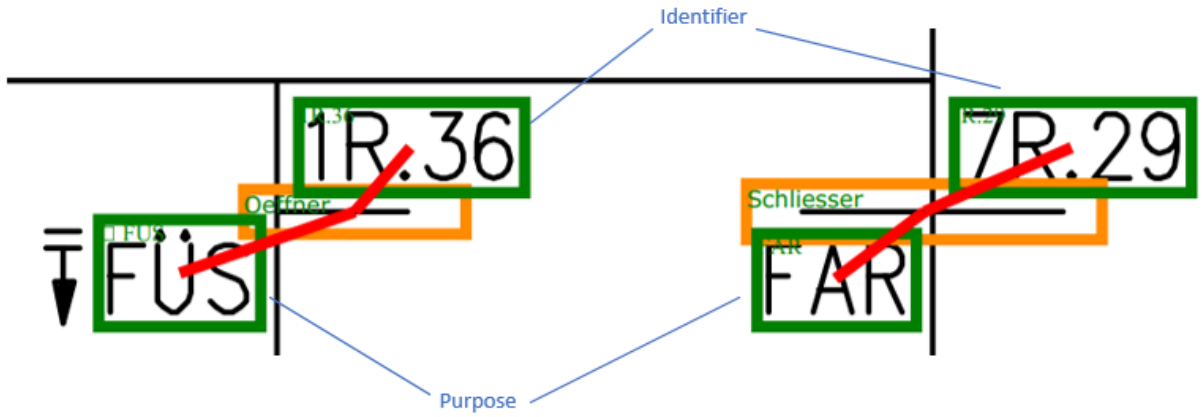


Figure 34 RelaisKontakt-Symbol Konvertierung

AnschlussConverter

Der AnschlussConverter kommt zum Einsatz, wenn der Typ des detektierten Symbols einem «Anschluss», einem «Connector», einer «Klemme» oder einem «Stripspunkt» entspricht. Hier werden, wie bereits bei den zuvor beschriebenen Convertern, die detektierten TextBoxen nach einem Identifier durchsucht. Dabei wird das Identifier Property mit der TextBox gleichgesetzt, deren erkannter Text dem folgenden Regex Pattern entspricht: "^\d+.\d+$" . Neben dem Identifier wird nach einer TextBox gesucht, die Informationen über die Installationslokation des Anschlusses enthält. Beispielsweise bei einem Stripspunkt über den Stripspunkt-Platz auf dem Strips, dem Strips-Platz auf dem Satz usw. Dabei wird die TextBox als Location genommen, deren Text dem folgenden Regex Pattern entspricht: "\w+\.\d+\.\d*" . Bei der Location handelt es sich um ein optionales Property, das leer gelassen wird, wenn kein passender Text gefunden wird.

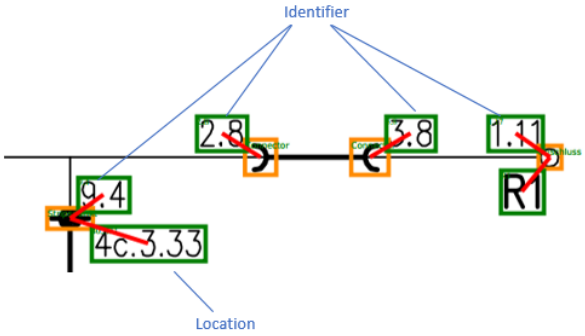


Figure 35 Anschluss-Symbol Konvertierung

SymbolConverter

Beim SymbolConverter handelt es sich um den Standard-Converter. Er bildet die Basis für alle oben erwähnten Spezial-Converter. Er kommt zum Einsatz, wenn der detektierte Symboltyp keinem der oben beschriebenen Typen entspricht. Seine Aufgabe ist die Umwandlung des Analyseresultates das als DTO Objekt vorliegt in ein Entity Objekt, das in der Datenbank gespeichert werden kann. Den Symbolen werden alle zuvor zugewiesenen TextBoxen hinterlegt, damit diese Assoziationen abgespeichert werden können.

Exporters

Exporter sind dafür zuständig, die Analyseresultate in einem vom User gewünschten Format verfügbar zu machen. Aktuell sind Exporter für den PDF- sowie den Bild-Export implementiert. Es können jeweils einzelne Seiten oder ein gesamtes Dokument exportiert werden. Genauere Informationen zu der jeweiligen Implementation können im nachfolgenden Abschnitt gefunden werden. Alle Exporter-Implementation bieten die Möglichkeit folgende Informationen zu exportieren:

- Erkannte Texte
- Detektierte Symbole
- Text – Symbol Assoziationen
- BoundingBoxes rund um die detektierten Symbole und erkannten Texte

PDF Exporter

Der PDF Exporter ist die Exporter Implementation, die es erlaubt, Analyseresultate in PDF-Dateien zu exportieren. Dazu wird eine Kopie der ursprünglich analysierten Dokumentation geladen und mit den Analyseresultaten angereichert. Ursprünglich war die Idee, die erkannten Texte als Tags im PDF zu hinterlegen, bei der Evaluation der verschiedenen Möglichkeiten hat sich jedoch gezeigt, dass solche Tags nicht von allen PDF-Readern gelesen werden können, respektive nicht nach ihnen gesucht werden kann. Aus diesem Grund wurde entschieden, dass die Texte auf die PDF-Seite selbst geschrieben werden. Somit sind sie am exakten Ort im Schema und können problemlos von allen PDF-Readern gesucht werden. Zur Bearbeitung der PDF-Dokumente wird das zuvor beschriebene PdfSharpCore verwendet. Unterhalb ist ein Screenshot einer exportierten PDF-Seite ersichtlich, auf der alle möglichen Informationen gekennzeichnet worden sind.

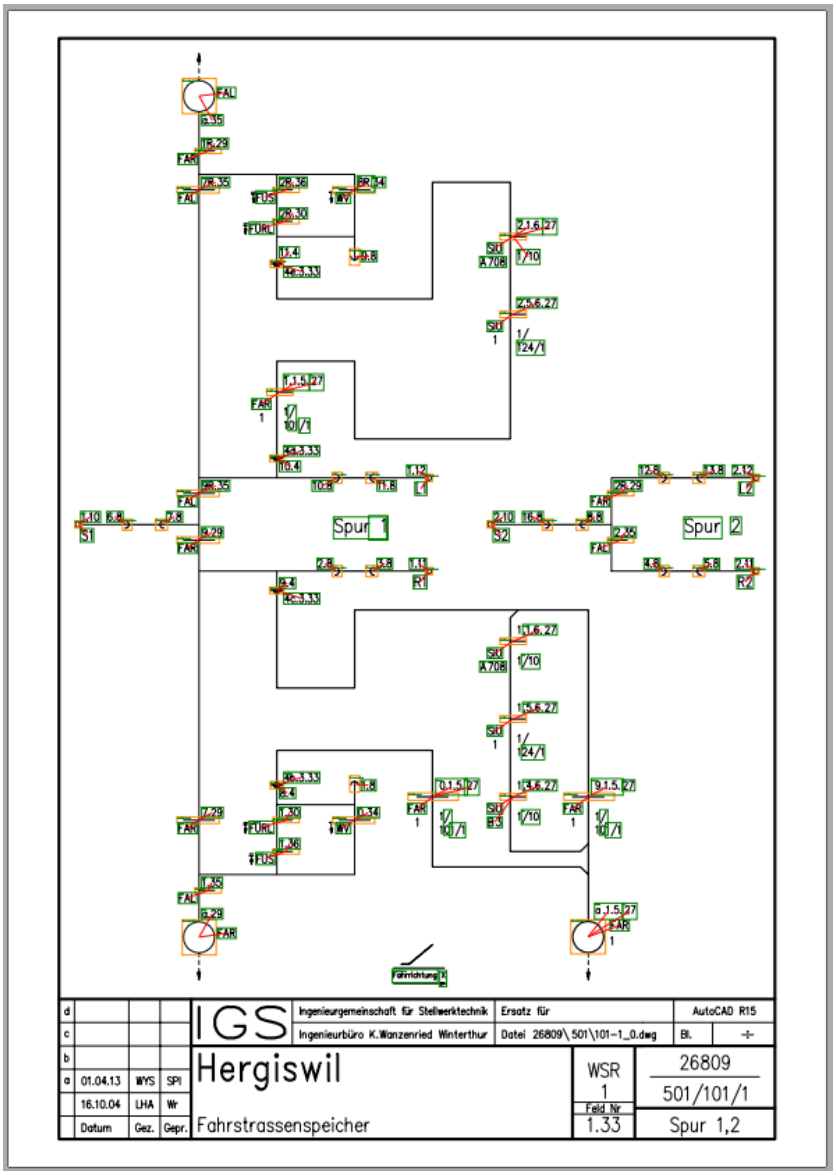


Figure 36 Exportiertes PDF-Analyseresultat

TIFF Exporter

Der TIFF-Exporter erlaubt es, die Analyseresultate als Bilder im TIFF Format zu exportieren. Dabei wird pro exportierte Seite ein Bild erstellt. Wie beim PDF-Exporter wird dazu zuerst eine Kopie des ursprünglich analysierten Dokuments geladen und die gewünschte Seite daraus in ein Bild umgewandelt. Dieses Bild wird anschliessend mit den Analyseresultaten erweitert, indem diese auf das Bild gezeichnet werden. Das Level an Informationen, die exportiert werden sollen, kann auch bei diesem Exporter gewählt werden. Die exportierten Bilder sind sehr hilfreich bei der Erstellung von neuen Symbol-Templates, da diese direkt aus diesen Bildern ausgeschnitten werden können. Die Bilder werden mit 300 DPI und im gleichen Format exportiert, wie dies beim Analyseprozess geschieht. Bilder, die ohne Resultatinformationen exportiert worden sind, können auch manuell einer «Schema_Analyzer» Python Instanz übergeben werden, um sie erneut analysieren zu lassen. Für die Bildbearbeitung wird die zuvor beschriebene Magick.NET Bibliothek verwendet. Unterhalb ist ein exportiertes Bild ersichtlich, bei dem alle verfügbaren Informationen gekennzeichnet worden sind.

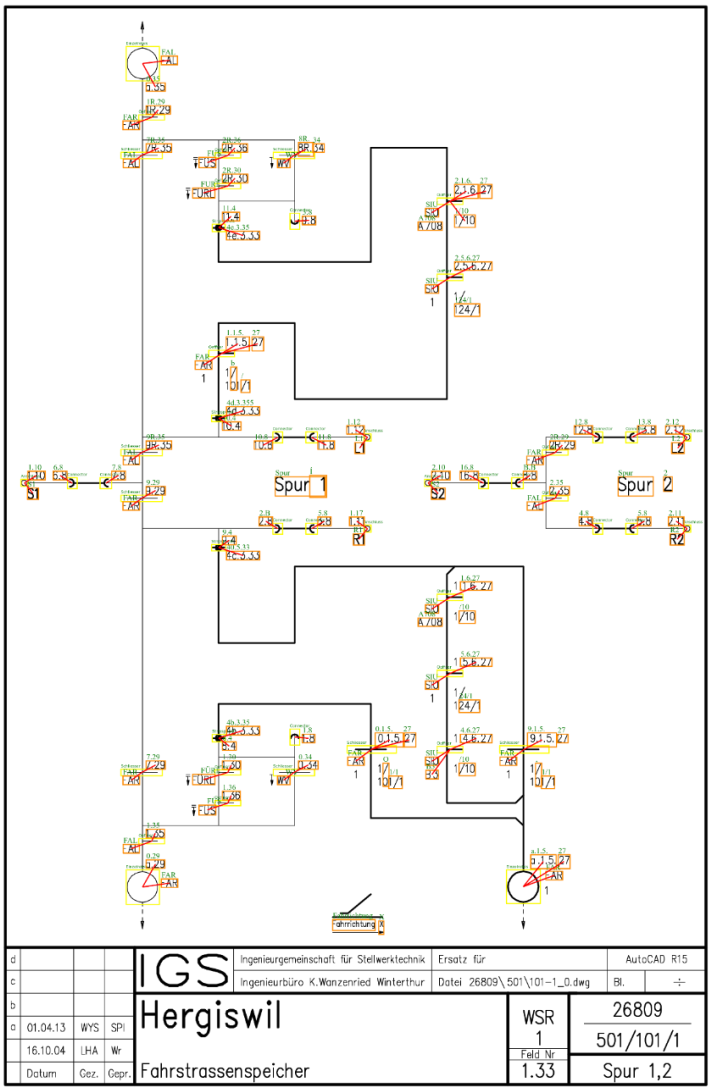


Figure 37 Exportiertes TIFF-Analyseresultat

«AI Interlocking» Infrastructure Layer

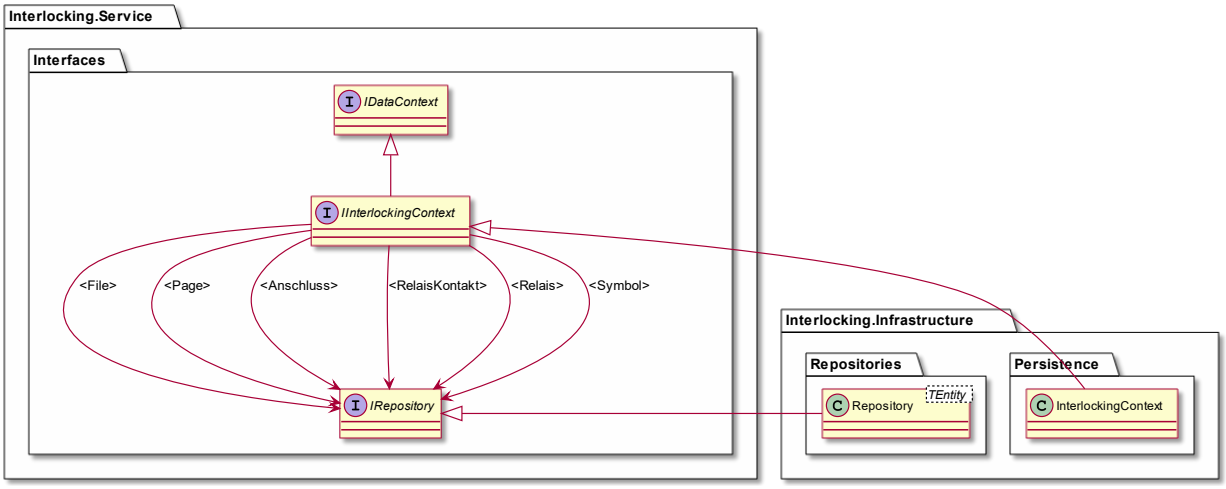


Figure 38 .NET Komponente Infrastructure Layer Klassenübersicht

Der Infrastruktur Layer beinhaltet die technologiespezifische Datenbankanbindungslogik. Für die Datenbankanbindung wird auf EntitiyFramework Core als ER-Mapper gesetzt. In der aktuellen Version ist der DataContext für eine SQLite Datenbankanbindung konfiguriert. Durch die gewählte Architektur kann die Datenbanktechnologie jederzeit problemlos ausgetauscht oder um eine weitere Technologie erweitert werden. Alle in diesem Layer vorhandenen Klassen implementieren Interfaces aus der Interlocking.Service Schale. Alle Businessservices arbeiten nur gegen diese Interfaces und wissen damit nichts von der tatsächlichen Implementation oder der dabei verwendeten Technologie. Dies führt zu einer sehr losen Kopplung der einzelnen Komponenten und damit zu einem modularen Applikationsaufbau. Zudem wird für den Datenzugriff auf das Repository Pattern gesetzt, welche zu einer weitern Abstraktion des Datenzugriffes führt und damit die Logik universeller einsetzbar macht. Zusätzlich zu den bereits erwähnten Vorteilen, führt die gewählte Architektur zu einer guten Testbarkeit, da die meisten Komponenten einzeln und somit unabhängig von Anbindungen jeglicher Art getestet werden können.

«AI Interlocking» Domain Layer

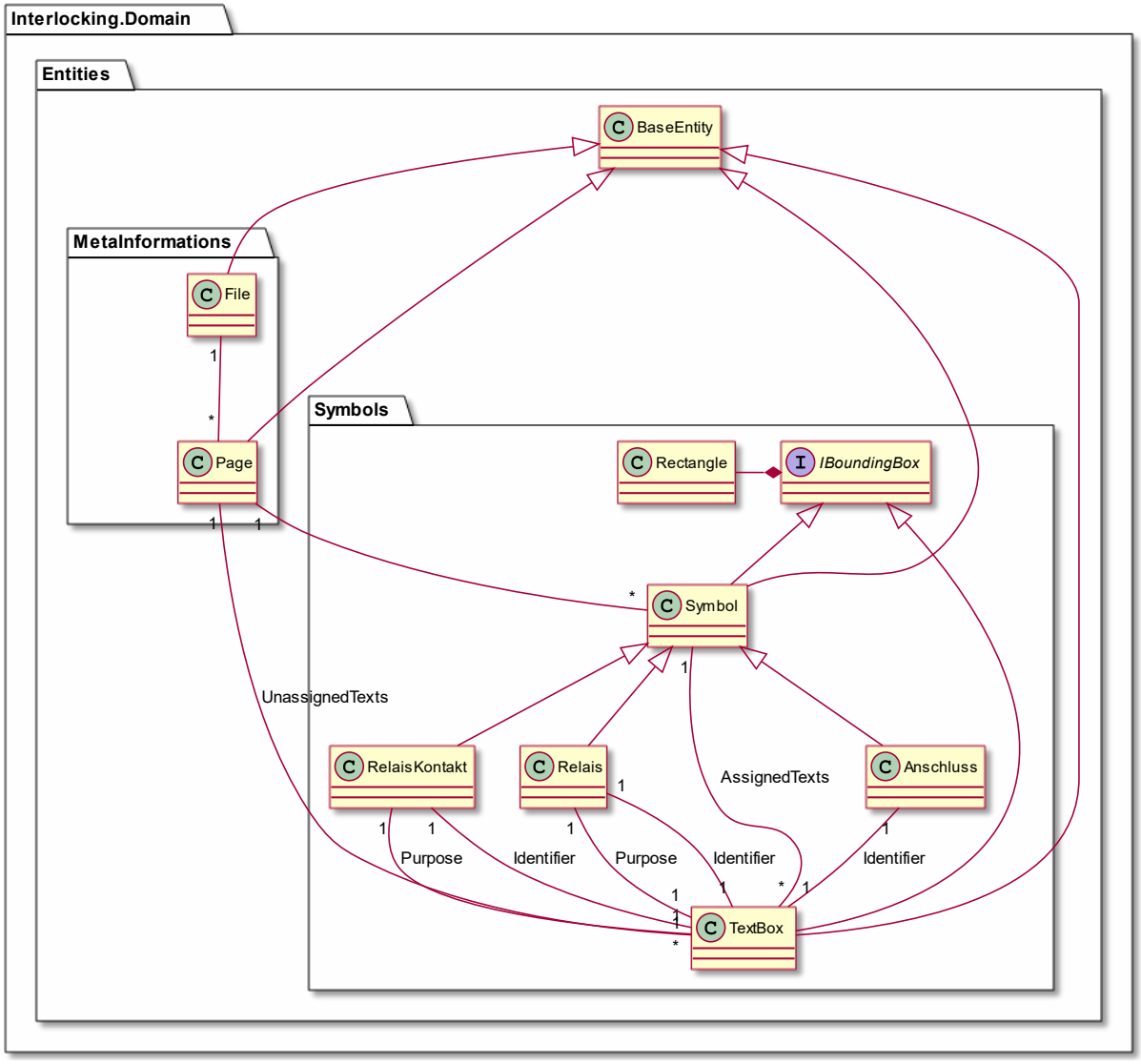


Figure 39 .NET Komponente Domain Layer Klassenübersicht

Im Domain Layer befinden sich die Domain Modell Klassen, welche sich aus der im Kapitel «Domainanalyse» ersichtlichen durchgeführten Analyse ableiten. Die dabei definierten Domain Klassen, wurden 1:1 in Entity Klassen umgewandelt und in diesem Domain Layer abgelegt. Speziell zu erwähnen, gibt es die BaseEntity Klasse, von welcher alle Entity Klassen ableiten. Dieses Vorgehen erlaubt es, die Entity Klassen übersichtlich zu halten und geteilte Properties auszulagern. Die Domain Entity Klassen wurden so aufgebaut, dass TextBoxen und Symbole beliebig gemischt werden können, indem gegen das Interface IBoundingBox gearbeitet wird. Alle Symbol- sowie die TextBox-Klasse implementieren dieses erwähnte Interface, welches die Koordinaten der jeweiligen Box abstrahiert. Eine weitere erwähnenswerte Eigenschaft ist die Aufteilung der Metainformationen und den symbolverwandten Entity Klassen. Bei dieser Trennung handelt es sich um eine logische Unterteilung des Domain-Modells gemäss dem Verwendungszweck der einzelnen Klassen.

Logische Architektur «Schema_Analyzer Python»

In diesem Kapitel wird auf die Architektur und Funktionalität der Python Komponente, dem «Schema_Analyzer» eingegangen. Diese Komponente ist für die eigentliche Analyse von einzelnen Schemaseiten verantwortlich. Hauptbestandteil dabei ist zum einen die Symboldetektion und zum anderen die Texterkennung. Diese Komponente wurde in Python geschrieben, da sich in der Elaborationsphase gezeigt hat, dass in Python besser geeignete Bibliotheken verfügbar sind, um die Analysefunktionalität umzusetzen. Bei der Evaluation der verschiedenen Möglichkeiten zur Bildverarbeitung und Analyse, hat sich OpenCV als sehr gut geeignete Bibliothek herauskristallisiert. Für die Texterkennung wurde Tesseract, als sehr oft verwendete und gut dokumentierte Bibliothek, gewählt. Genauere Informationen zur Evaluation der Bibliotheken und deren Konfiguration und Verwendung im Projekt können den untenstehenden Abschnitten entnommen werden.

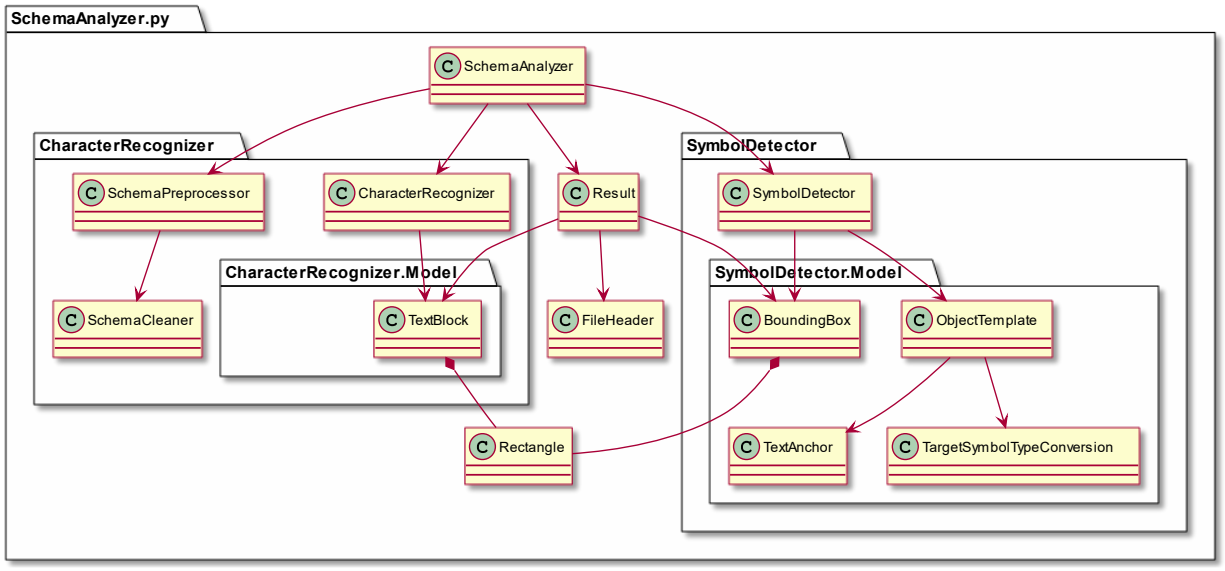


Figure 40 Python Komponente Paketübersicht

Diese Komponente kann grundsätzlich in zwei Pakete unterteilt werden. Zum einen ist dies das CharacterRecognizer Paket, in dem die Logik für die Texterkennung sowie für das dafür benötigte Preprocessing der Bilddateien implementiert worden ist. Zum anderen gibt es das SymbolDetector Paket, in dem die Funktionalität zum Detektieren von Schemasymbolen enthalten ist. Beide Pakete verfügen über die jeweils von ihnen benötigten Model Klassen. Bei diesen Modelklassen handelt es sich jeweils um einen Ausschnitt aus dem Domainmodel, welches im Kapitel Domainanalyse gefunden werden kann. Die SchemaAnalyzer Klasse bildet den Einstiegspunkt in die Applikation und kümmert sich um die Koordination der beiden Pakete, sowie das Zusammensetzen des Analyseresultates aus den Resultaten der beiden Pakete. Dieses Analyseresultat wird als JSON-Datei ins Dateisystem gespeichert und somit verfügbar gemacht.

Preprocessing

In diesem Abschnitt wird auf die Vorverarbeitungslogik eingegangen. Diese wird benötig, um die Schemabilder so vorzubereiten, dass eine zuverlässige Texterkennung möglich ist. Untenstehend wird auf die einzelnen Schritte dieses Preprocessings eingegangen. Wie bereits erwähnt, wird für die Bildverarbeitung OpenCV verwendet, das sich bei der Prototypisierung als sehr hilfreich erwiesen hat. Die Analyse und somit auch die Vorverarbeitung kann in zwei Teile unterteilt werden. Zum einen ist dies die Analyse der Fusszeile, aus der die Informationen zur Stationsnummer sowie die Blattnummer gelesen werden. Zum anderen gibt es die Analyse des eigentlichen Schemas, wobei Symbole detektiert werden und alle Beschriftungen detektiert werden sollen.

Footer

In diesem Abschnitt wird auf die Vorverarbeitung der Fusszeile eingegangen. Diese ist nötig, um eine erfolgreiche Erkennung der Stations- sowie Blattnummer zu ermöglichen. Unterhalb sind die dabei nacheinander ausgeführten Schritte mit dem jeweiligen Resultat beschrieben und erklärt.

Cropped Footer

In einem ersten Schritt muss die Fusszeile von der restlichen Seite getrennt werden. Dies wird erreicht, indem horizontale Linien gesucht werden im Bild. Dabei kommt die findContours Funktion von OpenCV mit dem passenden Kernel als Parameter zum Einsatz. Für die gefundenen Linien wird anschliessend geprüft, ob sie als Dokumentunterteiler in Frage kommen. Dies ist der Fall, wenn sie entweder zwischen 2000 und 2500 Pixel (A4 Seiten) oder zwischen 4500 und 5000 Pixel (A3 Seiten) breit ist. Der unterste mögliche Dokumentunterteiler wird als Seitenende angeschaut, der oberste mögliche Dokumentunterteiler wird als Seitenanfang angeschaut. Der Beginn der Fusszeile wird standardmässig auf den zweituntersten möglichen Dokumentunterteiler gesetzt. Von dort weg wird iterativ überprüft, ob sich innerhalb der darüberliegenden 150px (12.7mm) ein weiterer möglicher Dokumentenunterteiler liegt. Ist dies der Fall, kann davon ausgegangen werden, dass man sich in einer Tabellenstruktur innerhalb der Fusszeile befindet. In diesem Fall wird der Beginn der Fusszeile auf den nächsthöheren möglichen Dokumentunterteiler gesetzt und die Überprüfung geht in die nächste Iteration. Unterhalb ist eine korrekt abgetrennte Fusszeile ersichtlich.

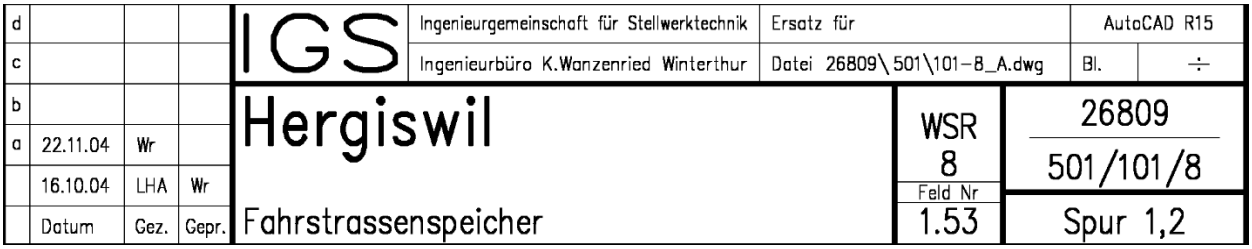


Figure 41 Extrahierte Fusszeile

Linienbereinigter Footer

In der Elaborationsphase hat sich gezeigt, dass Tesseract teilweise durch die Linien auf einem Bild bei der Texterkennung gestört wird. Um die bestmögliche Erkennungsrate zu erhalten, werden aus diesem Grund in einem zweiten Schritt alle horizontalen und vertikalen Linien mit einer Mindestlänge von 120px (ca. 1cm) entfernt. Dabei kommt wieder die gleiche OpenCV Funktionalität wie bei der Fusszeilendetektion zum Einsatz. Die dabei gefundenen Linien werden weiss übermalt und somit vom Bild entfernt. Unterhalb ist ein Beispiel einer linienbereinigten Fusszeile zu sehen.

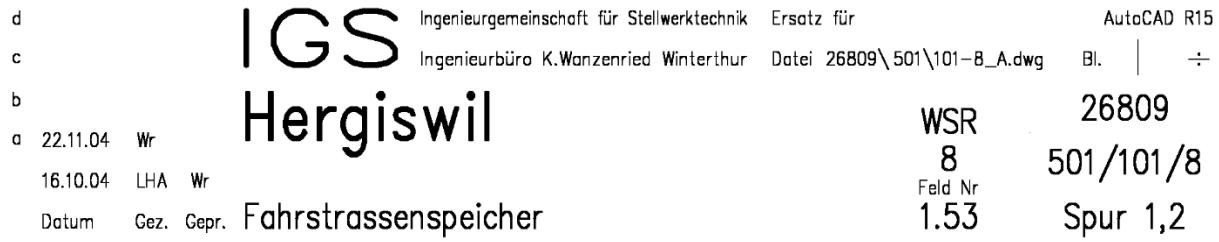


Figure 42 Linienbereinigte Fusszeile

Abgetragener Footer (Schriftverkleinerung)

Um eine optimale Texterkennung gewährleisten zu können, auch wenn es sich um handschriftliche Fusszeilen handelt, muss die Strichbreite der Schrift verkleinert werden. Damit wird erreicht, dass mit unterschiedlichen Schreibstylen umgegangen werden kann.

Figure 43 Beispiel OpenCV cv::erode

Bei der Evaluation der benötigten Vorverarbeitungsschritte gezeigt, dass bei der Fusszeilenanalyse nur eine geringe Verdünnung notwendig ist, um gute Resultate zu erzielen. Unterhalb ist eine Fusszeile nach dem Verdünnungsschritt abgebildet.

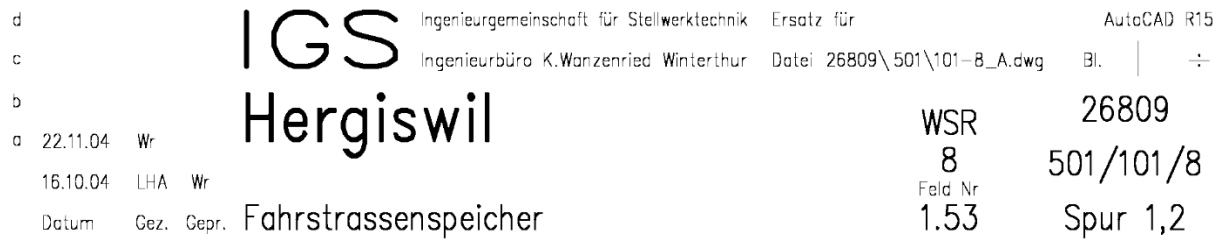


Figure 44 Abgetragene Fusszeile

Entrauschter Footer

Der letzte Schritt der Fusszeilenvorbereitung ist das Entrauschen des Bildes, wobei dieses geglättet wird. Dies wird erreicht, indem kleine Punkte und Flecken aus dem Bild entfernt werden. Für diesen Schritt wird aktuell die OpenCV fastNlMeansDenoisingColored Funktion eingesetzt, welche die gewünschten Effekte mit einem Mean Filter erreicht. Unterhalb ist ein entrauschtes und geglättetes Beispiel einer Fusszeile zu sehen. Dabei ist bei dem hier vorliegenden Beispiel kein grosser Unterschied zu erkennen, da es sich bereits vor diesem Schritt um ein ziemlich sauberes Bild gehandelt hat. Dennoch hat sich bei der Validierung der Texterkennung gezeigt, dass dieser Schritt einen positiven Effekt auf die Erkennungsrate hat.

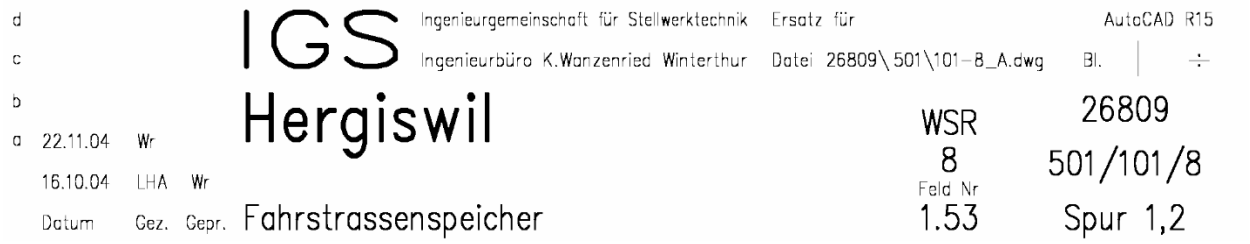


Figure 45 Entrauschte Fusszeile

Schema

In diesem Abschnitt wird auf die Vorverarbeitung des Schemas eingegangen. Diese ist nötig, um eine erfolgreiche Erkennung der Symbolbeschriftungen und weiteren Textinformationen zu ermöglichen. Unterhalb sind die dabei nacheinander ausgeführten Schritte mit dem jeweiligen Resultat beschrieben und erklärt. Die bei den einzelnen Schritten ersichtlichen Resultate entsprechen aus Platzgründen nicht ihrer Originalgrösse, wobei es sich beim gegebenen Beispiel um die Abmessungen einer A4 Seite handelt. Ausgangspunkt für diesen Vorverarbeitungsschritt ist das zuvor aus einer PDF-Seite generierte Bild im TIFF-Format. Genauere Informationen zu diesem Konvertierungsschritt und dem dabei erzielten Resultat können dem Abschnitt «PDF – Image Konvertierung» im Kapitel Architektur entnommen werden.

Cropped Schema

Beim ersten Schritt geht es wie bereits bei der Fusszeile darum, den interessanten Bereich zu extrahieren. In diesem Fall ist dies der Schemaausschnitt, welcher analysiert werden soll. Um dieses Ziel zu erreichen, wird dieselbe Logik verwendet, um die möglichen Dokumentunterteiler zu finden, wie dies bereits bei der Fusszeile gemacht worden ist. Der oberste mögliche Dokumentenunterteiler wird als Schemabeginn gewählt und definiert zudem die Breite des zu analysierenden Bereiches. Dies führt dazu, dass nur der Teil der Seite innerhalb des Rahmens analysiert wird. Die zuvor detektierte Fusszeile wird weiss überdeckt und somit aus dem Bild entfernt, damit bei der Texterkennung keine überflüssigen Texte detektiert werden. Unterhalb ist eine Schemaseite nach diesem ersten Vorverarbeitungsschritt ersichtlich.

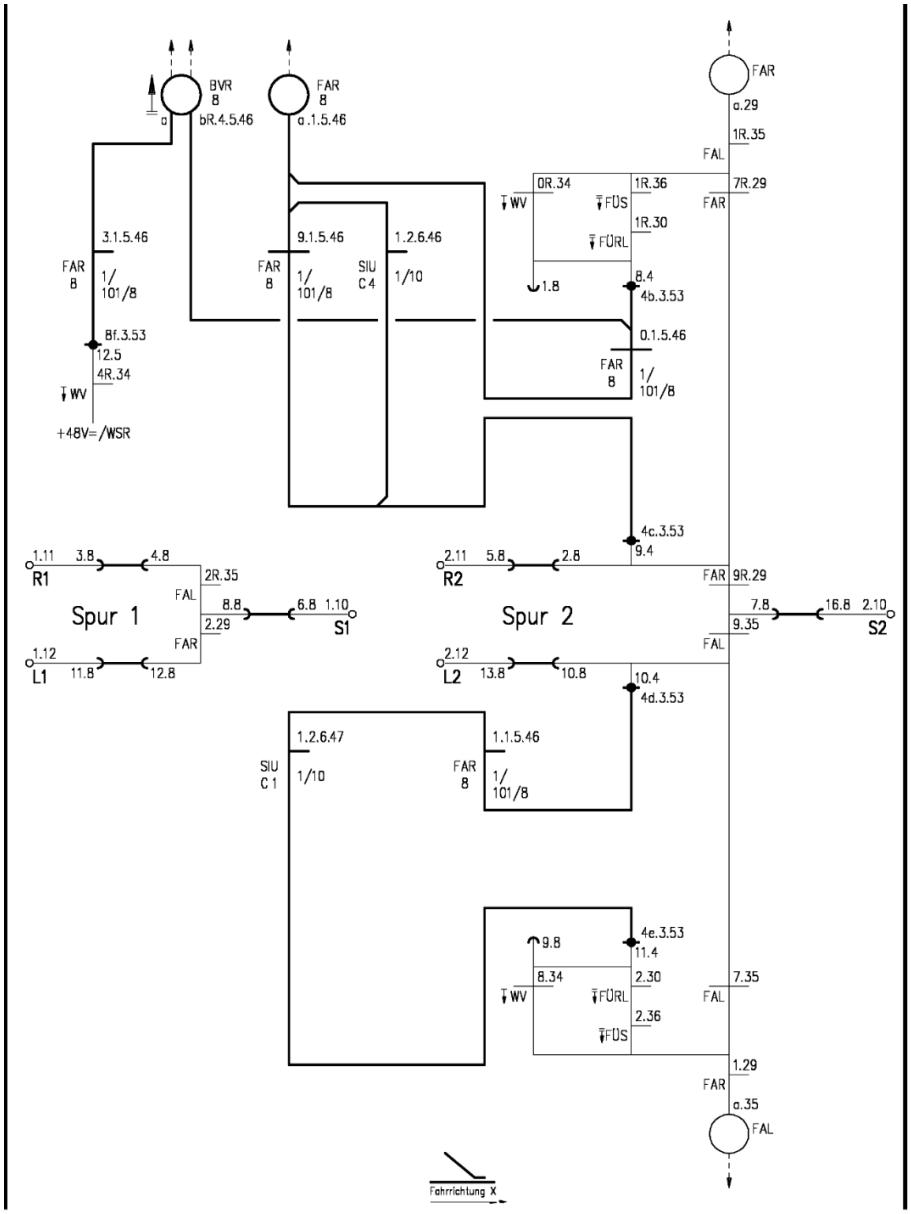


Figure 46 Extrahiertes Schema

Linienbereinigtes Schema

Wie bereits bei der Fusszeilenbereinigung werden auch beim Schemateil im zweiten Schritt die Linien enfernt. Dieser Schritt hat einen sehr grossen Einfluss auf die korrekte Erkennungsrate, da sich herausgestellt hat, dass Tesseract durch die Linien beeinflusst wird und diese Teilweise als Textkomponenten respektive Textunterteiler erkennt. Für diesen Schritt wird die exakt gleiche Logik verwendet wie bei der Fusszeilenbereinigung. Somit werden hier wieder die im Abschnitt «Footer Bereinigung» beschriebenen OpenCV Funktionen zum detektieren der Linien und dem anschliessenden weissen Übermalen verwendet. Unterhalb ist eine Dokumentationsseite ersichtlich, welche die ersten zwei Vorverarbeitungschritte durchlaufen hat.

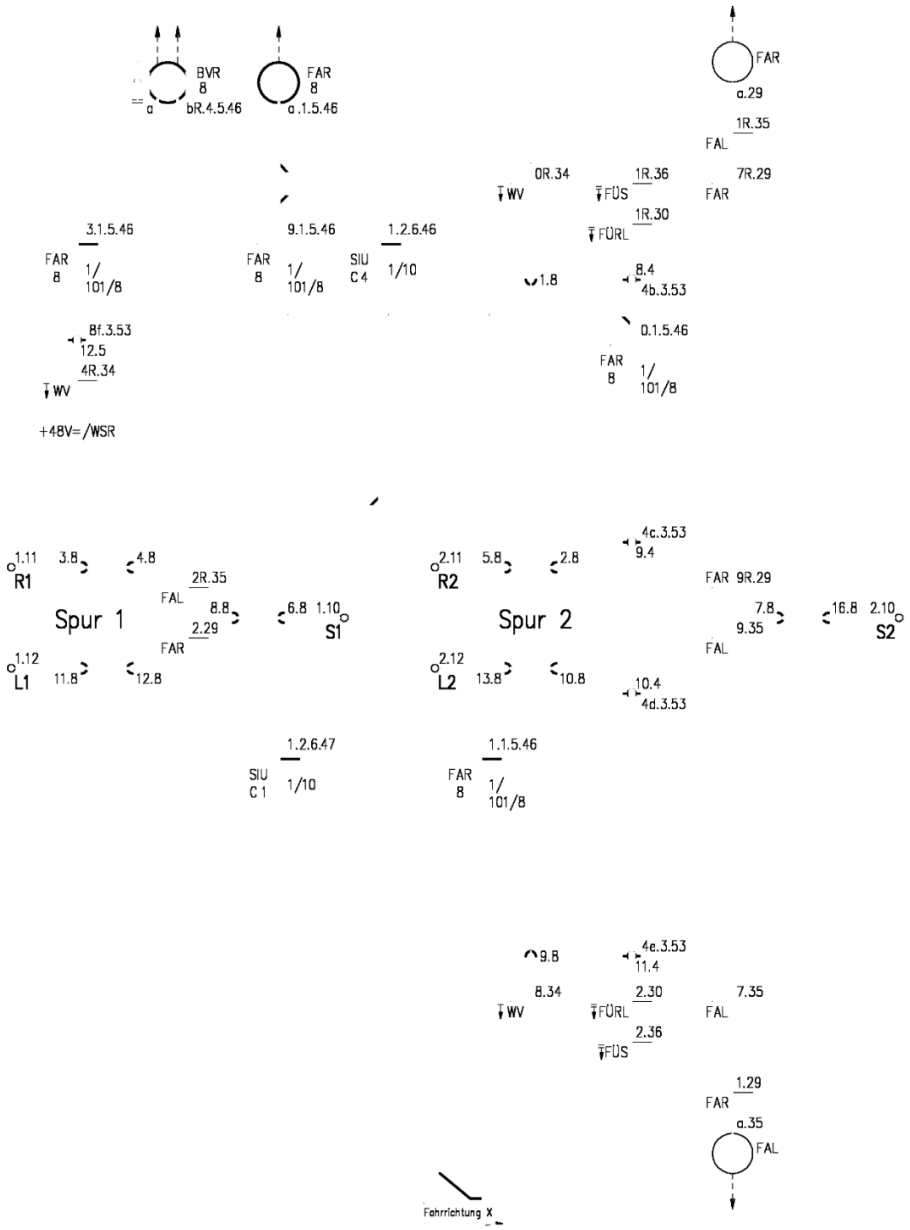


Figure 47 Linienbereinigtes Schema

Symbolbereinigtes Schema

Beim dritten Vorverarbeitungsschritt handelt es sich um einen speziell für die Schemaanalyse entwickelten Prozess. Bei der Evaluation der Möglichkeiten für eine genaue Texterkennung hat sich gezeigt, dass es sich bei den Schemasymbolen um den grössten Störfaktor handelt. Diese wurden in allen evaluierten Textverarbeitungsvarianten teilweise als Buchstaben oder Wortteile erkannt und haben somit die Qualität des Analyseresultates stark beeinflusst. Um diese Fehlerquelle so klein wie möglich zu halten, werden die zuvor im Symboldetektionsschritt gefundenen Symbole in diesem Preprocessing Schritt weiss überdeckt und somit aus dem Bild entfernt. Genauere Informationen zur Symboldetektion und die damit verbundenen Konfigurationsmöglichkeiten können dem Kapitel «SymbolDetector» entnommen werden. Es ist zu beachten, dass Symboltemplates so klein wie möglich sind und nur das tatsächliche Symbol beinhalten. Ansonsten kann es vorkommen, dass dieser Symbolbereinigungsschritt nicht nur die Symbole entfernt, sondern auch interessanten Text löscht, der danach nicht mehr erkannt werden kann. Speziell zu erwähnen, gibt es hier die Symboltemplate-Kategorie «Unknown» die darin enthaltenen Symbole werden nicht ins Analyseresultat aufgenommen, werden aber in diesem Schritt ebenfalls aus dem Schema entfernt. Damit ist eine Möglichkeit gegeben, störende Symbole, die nicht interessant fürs Resultat sind auszublenden und damit die Erkennungsrate zu optimieren. Unterhalb ist eine Beispielseite zu sehen, welche die ersten drei Vorverarbeitungsschritte durchlaufen hat und somit symbolbereinigt ist.

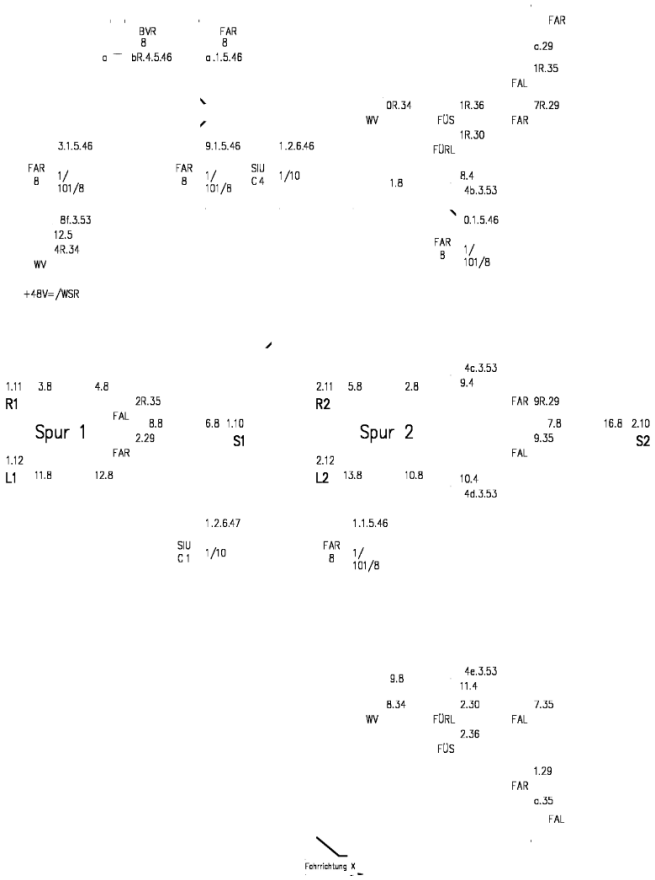


Figure 48 Symbolbereinigtes Schema

Abgetragenes Schema (Schriftverkleinerung)

Um eine optimale Texterkennung gewährleisten zu können, auch wenn es sich um handschriftliche Schemazeichnungen handelt, muss die Strichbreite der Schrift verkleinert werden. Damit wird erreicht, dass mit unterschiedlichen Schreibstylen umgegangen werden kann. In diesem Schritt wird die gleiche Funktionalität wie bei der Fusszeilenbereinigung verwendet. Dabei werden die morphologischen OpenCV Funktionen «erode» und «dilate» mit einem passenden Kernel aufgerufen.
Im Gegensatz zur Fusszeilenbereinigung, bei der nur eine geringe Verdünnung der Strichstärke notwendig war, hat sich bei der Validierung der Schemaanalysen gezeigt, dass ein Kernel der zu einer stärkeren Verdünnung führt die beste Erkennungsrate des Textes bietet. Im unten ersichlichen Beispiel einer Dokumentationsseite nach diesem Vorverarbeitungsschritt ist bei der Strichstärke ein deutlicher Unterschied feststellbar.

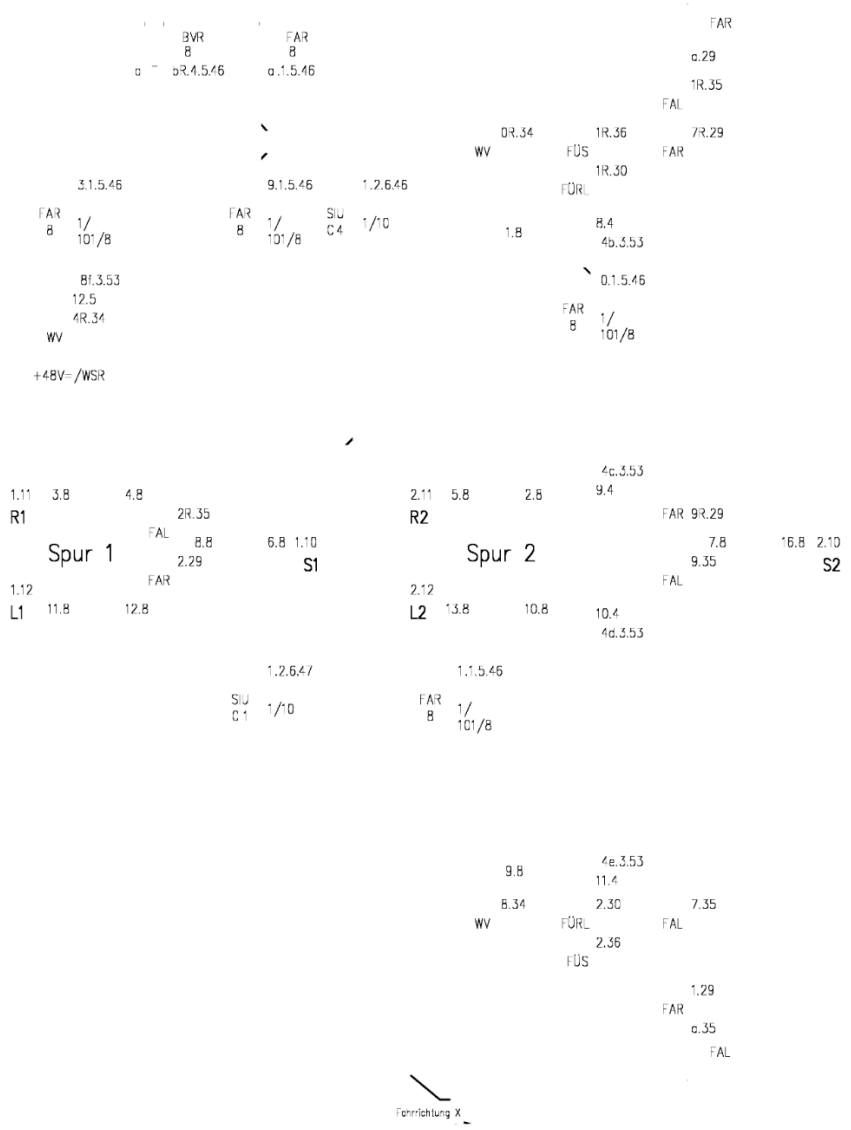


Figure 49 Abgetragenes Schema

Entrauschtes Schema

Als letzter Schritt der Schemavorverarbeitung wird wie bei der Fusszeile eine Entrauschung durchgeführt, welche das Bild glättet und kleine Punkte oder Flecken entfernt. Bei diesem Schritt kommt die exakt gleiche OpenCV Funktionalität zum Einsatz, wie sie bei der Fusszeilenbereinigung angewendet wird. Weitere Informationen dazu können dem entsprechenden Abschnitt unter «Footer Bereinigung» entnommen werden. Unterhalb ist ein entrauschtes Schemabild ersichtlich, nachdem es alle Vorverarbeitungsschritte durchlaufen hat. Dieses Bild wird im Anschluss der Texterkennungskomponente übergeben und dementsprechend analysiert.

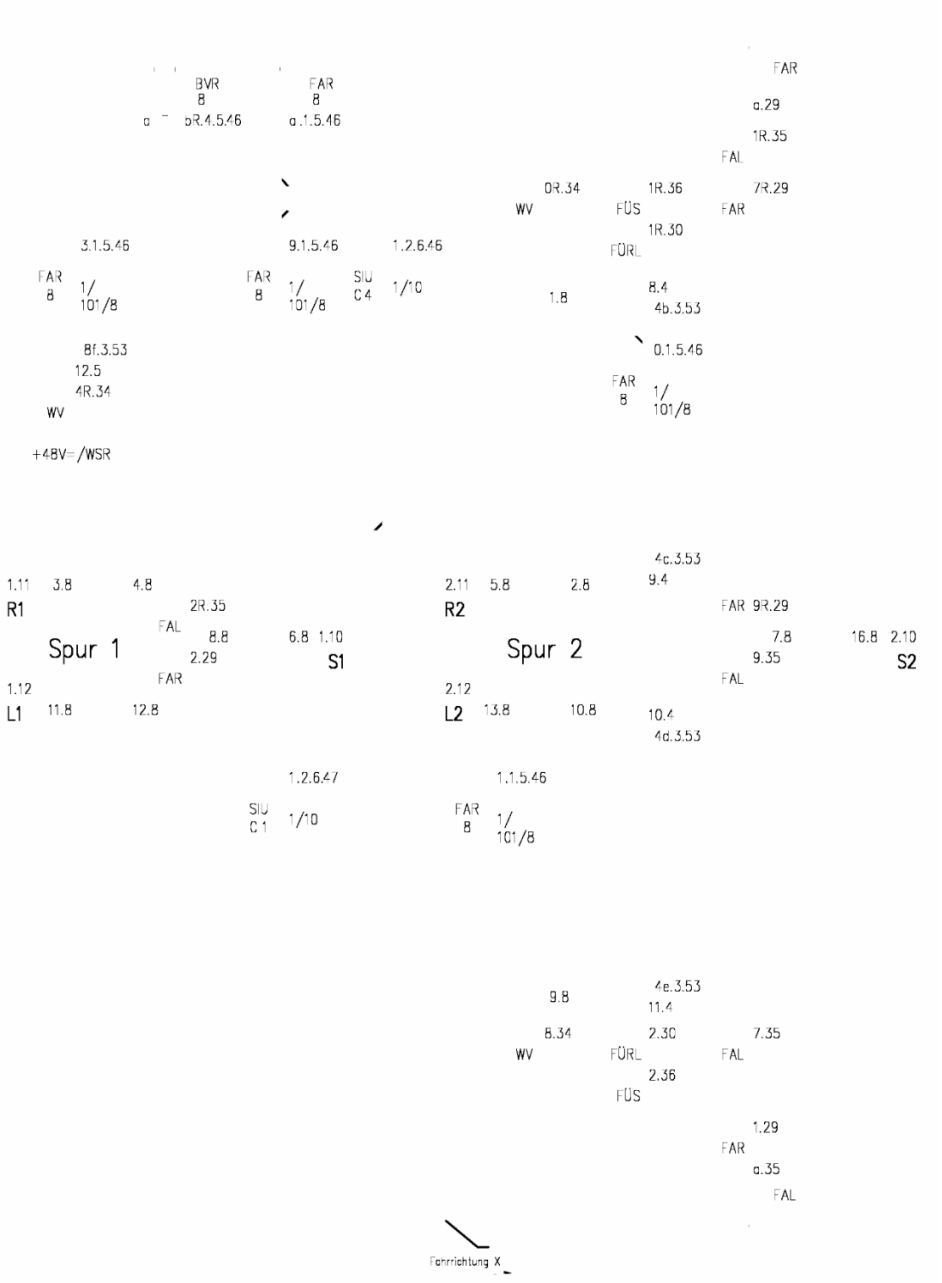


Figure 50 Entrauschtes Schema

SymbolDetector

Das SymbolDetector Paket der Python Komponente wird verwendet, um konfigurierbare Symbole auf einer Schemaseite zu finden. In der Elaborationsphase wurden diverse Technologien evaluiert, welche zum gewünschten Ziel hätten führen können.
Der erste Ansatz war die Verwendung von einem selbst trainierten Neuronalen Netz zur «Object Detection» in ML.NET. Für das Training wurden verschiedene Varianten von TensorFlow über Pytorch bis hin zu auf Azure trainierten ML.NET Netzen ausprobiert. Diese Netze sollten im ONNX Format mit ML.NET direkt in die .NET Komponente der Applikation integriert werden und somit verhindern, dass verschiedene Programmiersprachen eingesetzt werden müssen. Einige dabei ausgearbeitete Prototypen lieferten akzeptierbare, aber nicht hervorragende Ergebnisse. Andere Prototypen erkannten nur wenige Symbole und lieferten schlechte Ergebnisse. Zudem hat sich gezeigt, dass für gute Ergebnisse sehr viele getaggte Daten benötigt würden, welche mit einem grossen Zeitaufwand hätten erstellt werden müssen.
In einem zweiten Ansatz wurde die Verwendung von Template-Matching Algorithmen für die Symbolerkennung evaluiert. Dabei wurde unter anderem die Einsetzbarkeit von QATM (Quality Aware Template Matching for Deep Learning) geprüft. Parallel dazu wurde ein Prototyp entwickelt, der auf der OpenCV Template-Matching Funktionalität aufbaut. Die Erkennungsrate der verschiedenen Prototypen hat gezeigt, dass diese OpenCV Funktionalität sehr gute und im Vergleich zu den anderen Prototypen sogar hervorragende Ergebnisse liefert. Aus diesem Grund wurde entschieden, OpenCV für die Symboldetektion bei der Schemaanalyse einzusetzen.

Template Matching

Beim eingesetzten OpenCV Template Matching wird das Template Bild über das zu analysierende Bild verschoben und dabei Pixel für Pixel verglichen. Bei diesem Vergleich wird für jede Koordinate ein Wert berechnet, der die Übereinstimmung des Templates mit dem Bild an diesem Punkt widerspiegelt. Dabei stellt OpenCV verschiedene Methoden zur Berechnung dieses Wertes zur Verfügung. Bei der Evaluation der verschiedenen Möglichkeiten hat sich gezeigt, dass die «TM_CCOEFF_NORMED» Variante die besten Ergebnisse liefert und nahezu alle Symbole korrekt erkennt. Unten ist die Formel zur Berechnung des Ähnlichkeitswertes ersichtlich. Dabei handelt es sich im Wesentlichen um einen normierten Korrelationskoeffizienten, bei dem jeweils ein Skalarprodukt zwischen dem Template Bild und dem darunterliegenden Ausschnitt aus dem zu analysierenden Bild verwendet wird.

$$R(x,y) = \frac{\sum_{x',y'}(T'(x',y') \cdot I'(x+x',y+y'))}{\sqrt{\sum_{x',y'} T'(x',y')^2 \cdot \sum_{x',y'} I'(x+x',y+y')^2}}$$

Formel 1 Template-Matching

Um die richtigen Punkte zu finden, an denen ein Template gefunden worden ist, werden die berechneten Werte mit einem definierbaren Threshold verglichen. Liegt der Wert über dem definierten Threshold kann davon ausgegangen werden, dass es sich um eine Übereinstimmung handelt.
Template Bilder und Thresholds können beliebig hinterlegt und konfiguriert werden. Dabei gibt es zu beachten, dass die Template Bilder eine Pixeldichte von 300 DPI aufweisen sollten, da es sich dabei um den Wert handelt, mit dem die PDF-Seiten in Bilder umgewandelt werden. Weitere Informationen zu den Konfigurationsmöglichkeiten der Templates können dem Abschnitt «Schema_Analyzer Konfiguration» entnommen werden.
Unten ist ein Ausschnitt aus dem Resultat des SymbolDetectors ersichtlich. Dabei werden alle mit der zuvor beschriebenen Methode detektierten Symbole in einem Array abgelegt. Dabei wird für jedes detektierte Symbol der Symboltyp, der durch den Ordnernamen, in dem das Template Bild abgelegt wurde, definiert ist, die Konfidenz, mit welcher es sich um eine Übereinstimmung handelt, sowie die zugehörigen Koordinaten vermerkt. Zusätzlich werden noch einige weitere Properties abgespeichert, welche sich aus der Konfiguration der Templates ergeben. Mehr zu den Konfigurierbaren Parameter können dem Kapitel «Schema_Analyzer Konfiguration» entnommen werden.


```
{
    "Type": "Einzelrelais",
    "Confidence": "0.9605915",
    "Rectangle": {
        "X": "527",
        "Y": "2758",
        "Width": "103",
        "Height": "107"
    },
    "TextAnchors": [{
        "X": "578",
        "Y": "2811"
    }],
    "TargetSymbolTypeConversions": null,
    "AssignmentDistanceThreshold": 100
}
```


Figure 51 SymbolDetector detektiertes Symbol

Resultat

Wie die .NET Komponente bietet auch die Python Komponente die Möglichkeit ein Analyseresultat zu visualisieren. Der «Schema_Analyzer» kann als unabhängige Software eingesetzt werden, um einzelne Schemabilder zu analysieren. Unterhalb ist das visualisierte Resultat des SymbolDetectors zu finden. Darauf sind diverse Symboltypen zu erkennen mit der zugehörigen Konfidenz. Bei moderneren digital erstellten Plänen liegt die Erkennungsrate von Symbolen bei über 90%. Durch die individuelle Konfigurierbarkeit kann die Symbolerkennung so optimiert werden, dass auch Schemas mit schlechterer Qualität mit überzeugenden Ergebnissen analysiert werden können. Ein weiterer Vorteil der umfänglichen Konfigurationsmöglichkeiten ist die Möglichkeit, neue Symbole einfach hinzuzufügen und diese in die Symbolerkennung miteinzubeziehen. Diese Funktionalität erlaubt es auch zukünftige Pläne problemlos analysieren lassen zu können.

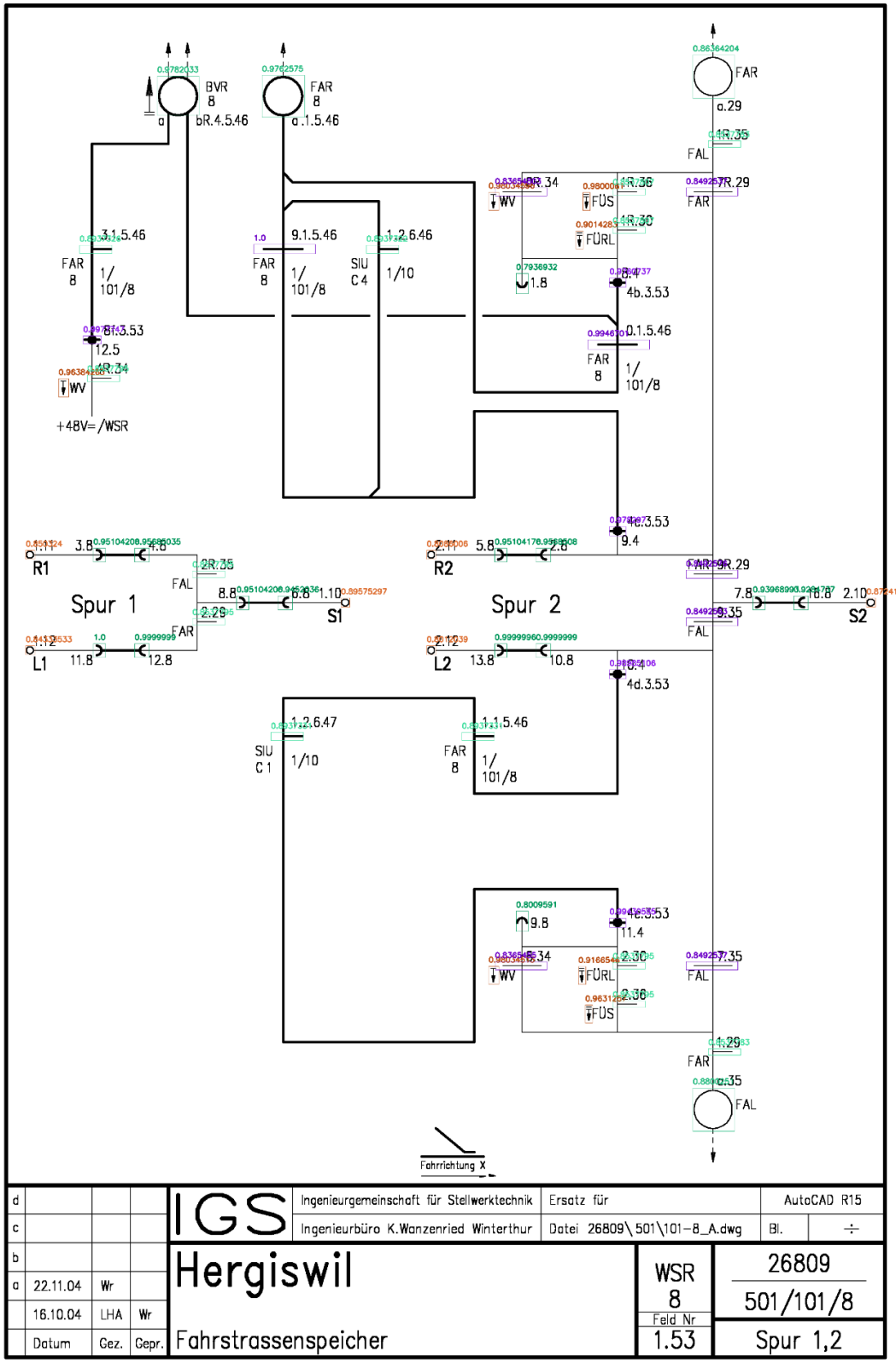


Figure 52 SymbolDetector Resultat (visualisiert)

CharacterRecognizer

Das CharacterRecognizer Paket ist für die Texterkennung auf den Schemabildern und den Fusszeilen verantwortlich. In der Elaborationsphase wurden diverse Varianten für die Texterkennung evaluiert, wobei sich Tesseract ziemlich schnell als ausgereifteste Möglichkeit erwiesen hatte. Bei Tesseract handelt es sich um eine Open Source Bibliothek, die unter der Apache Lizenz frei verfügbar ist. Tesseract hat sich in den vergangenen 20 Jahren in eine gut bekannte, relativ zuverlässige und sehr umfangreich konfigurierbare Bibliothek für die optische Zeichenerkennung entwickelt. Die ausführliche Dokumentation, die verfügbaren Beispiele sowie die Möglichkeit eigene Modelle zu trainieren haben zum Entscheid geführt Tesseract für die Texterkennung einzusetzen. In einem der ersten Prototypen wurde ein .NET Wrapper für Tesseract verwendet, damit die gesamte Applikation im .NET Umfeld entwickelt werden könnte. Dieser Wrapper hatte im Vergleich zur schlussendlich verwendeten pytesseract Python Bibliothek den Nachteil, dass Konfigurationen nicht im gleichen Umfang verfügbar sind und somit die Resultate nicht zuverlässig genug waren. Mit der Implementation des SymbolDetectors im Python Umfeld, konnte auch die Texterkennung in die Python Komponente ausgelagert werden, was zu einer deutlichen Verbesserung des Prozentsatzes der korrekt Erkannten Texte führte.

OCR

Wie bereits erwähnt sind die Konfigurationsmöglichkeiten von Tesseract sehr umfangreich. Neben drei verschiedenen OCR Engine Modes (OEM) sind 14 Page Segmentation Modes (PSM) sowie eine Liste unzähliger Parameter verfügbar. Für die Evaluation, der für den Anwendungsfall am besten geeigneten Konfiguration, wurde ein Prototyp erstellt, der die Verwendung aller OEMs mit jeweils allen PSMs zur Analyse eines Bildes erlaubt hat. Beim Testen mit einem Set von 20 Schemaseiten, das aus einer Vielzahl verschiedener Schematypen bestand (A4 Seiten, A3 Seiten, CAD Zeichnungen, Handschriftliche Zeichnungen) hat sich gezeigt, dass der OCR Engine Mode 1 (Neural nets LSTM engine only), der seit Tesseract 4 verfügbar ist, die besten Ergebnisse liefert. Die meisten korrekt erkannten Texte konnten bei der Verwendung des OEM 1 in Zusammenarbeit mit dem Page Segmentation Mode 12 (Sparse text with OSD) erzielt werden. Zusätzlich wurden die Parameterliste durchgearbeitet und analysiert, wobei sich die folgenden Parameter als qualitätsoptimierend herausgestellt haben:

Parameter	Wert	Beschreibung
tessedit_char_blacklist	|[]{}%	Blacklist of chars not to recognize
load_system_dawg	0	Load system word dawg
load_freq_dawg	0	Load frequent word dawg
load_unambig_dawg	0	Load unambiguous word dawg
load_punc_dawg	0	Load dawg with punctuation patterns
load_number_dawg	0	Load dawg with number patterns
load_bigram_dawg	0	Load dawg with special word bigrams

Table 5 Verwendete Tesseract Parameter

DAWG = Directed Acyclic Word Graphs

Bei den meisten der oben genannten Parametern geht es darum, dass keine Wörterbuchwörter erkannt werden sollen und auch die Punktsetzung oder die Zahlenzusammensetzung nicht dem sprachlichen Standard entsprechen. Die meisten zu detektierenden Texte sind Identifier oder Bezeichnungen, wobei es sich nicht um Wörter handelt, die in einem Wörterbuch vorhanden sind.

Neben diesen Konfigurationsmöglichkeiten wurden auch Versuche durchgeführt, bei denen der Unterschied zwischen der Textblock Erkennung und der Erkennung von einzelnen Zeichen untersucht worden ist. Bei der Erkennung einzelner Zeichen hat sich dabei kein gewinnbringender Effekt gezeigt. Durch die Performanz beeinträchtigende Logik zum Zusammenfügen der einzelnen Zeichen hat sich diese Variante im Gegenteil als schlechtere Option herausgestellt, was zum Entscheid geführt hat, Tesseract nach Textblöcken suchen zu lassen.

Für die Texterkennung werden in der aktuellen Version werden die vortrainierten Modelle für Latein und Deutsch verwendet, womit problemlos alle Standardzeichen erkannt werden können. Für die Erkennung der Pfeile, die Teilweise zu Bezeichnungen gehören, wurde eine separate Logik entwickelt, welche die SymbolDetector Logik verwendet. Genauere Informationen zur Spezialsymbolerkennung können dem Kapitel «Schema_Analyzer Konfiguration» entnommen werden. Dieses Vorgehen hat das aufwändige Trainieren eines eigenen Texterkennungsmodelles für den Moment überflüssig gemacht. Mit der aktuellen Konfiguration und den verwendeten Modellen werden auf digital erstellten Schemaseiten über 85% des Textes richtig erkannt, wobei die grössten Fehlerquellen bei Texten liegen, die sich über mehrere Zeilen erstrecken. Teilweise kann auch die Kombination von Zahlen und Buchstaben zu Falscherkennungen führen, wenn beispielsweis ein a als 0 erkannt wird. Möglicherweise könnte mit einem speziell für die verwendete Schrift trainierten Modell noch eine Qualitätssteigerung erzielt werden. Bei handschriftlichen Dokumenten kommt die Erkennungsrate auf die Qualität der Dokumente an. Bei sauber strukturierten und einigermassen der Standardschrift entsprechenden Beschriftungen liegt die Erkennungsrate nicht weit entfernt von der bei digital erstellten Plänen. Bei Plänen mit schlechter Scanqualität oder unsauberer Schrift, kann diese Rate jedoch auch unter 50% richtig erkannte Texte sinken.

Unten ist ein Ausschnitt aus dem Resultat des CharacterRecognizer ersichtlich. Dabei werden alle mit der zuvor beschriebenen Methode erkannten Texte in einem Array abgelegt. Für jeden erkannte Textblock wird der erkannte Text, die Konfidenz, mit welcher der Text erkannt worden ist, sowie die zugehörigen Koordinaten vermerkt.


```
{
    "Text": "bR.4.5.46",
    "Confidence": "0.85",
    "Rectangle": {
        "X": "625",
        "Y": "354",
        "Width": "125",
        "Height": "25"
    }
}
```


Figure 53 CharacterRecognizer erkannter Text

Resultat

Wie beim SymbolDetector Paket kann auch beim CharacterRecognizer Paket das Analyseresultat visualisiert werden. Dabei werden die erkannten Textblöcke eingerahmt und der dazugehörige Text vermerkt. Unterhalb ist das visualisierte Resultat einer CharacterRecognizer Analyse einsehbar. Bei der analysierten Seite handelt es sich um eine AutoCAD Zeichnung, auf welcher der Text in einer sauberen Schrift vorhanden ist, was eine nahezu perfekte Erkennungsrate mit sich bringt. Zudem wurden bei dieser Analyse auch die Spezialsymbole korrekt erkannt und den zugehörigen Texten angehängt.

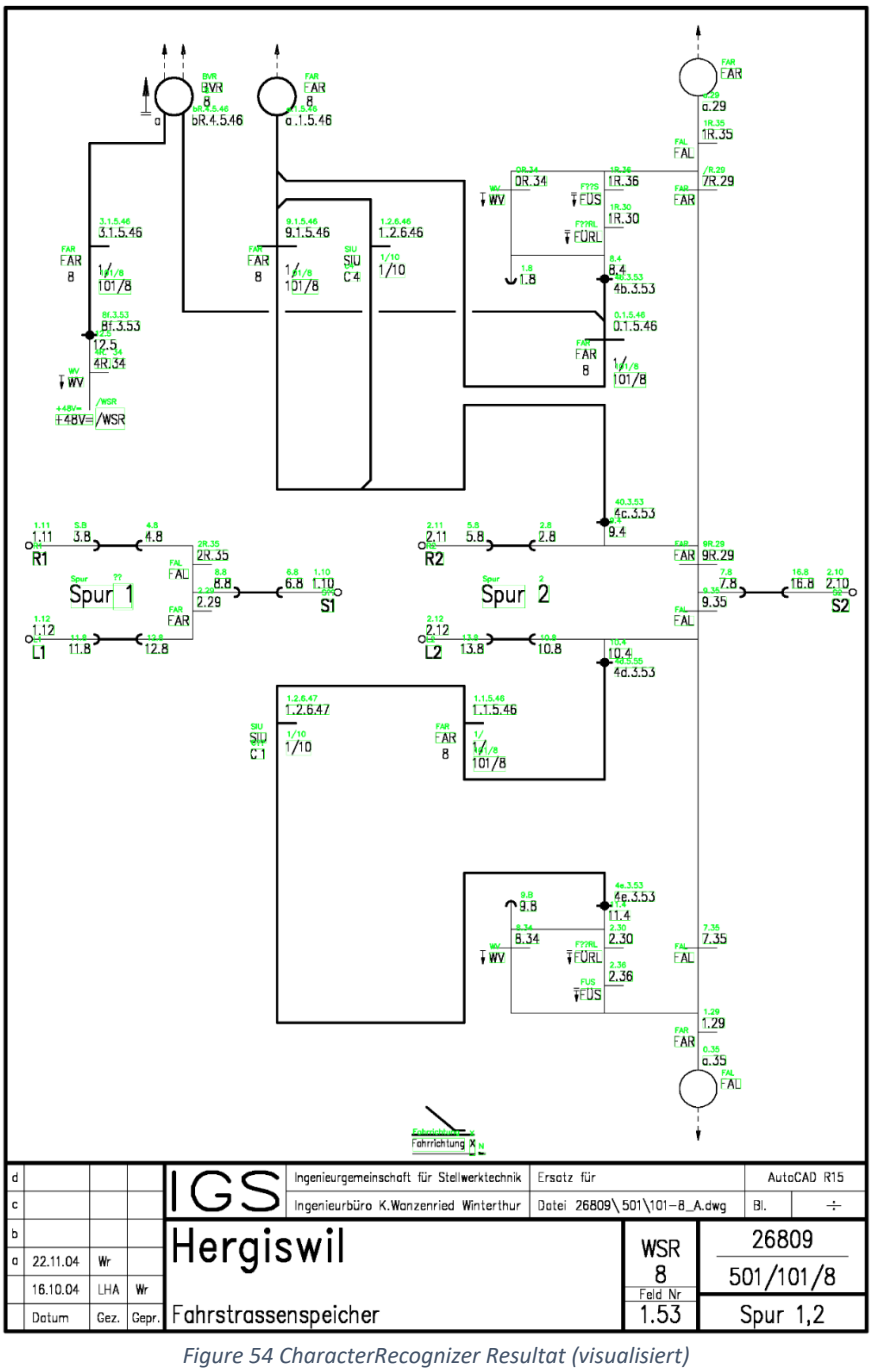


Figure 54 CharacterRecognizer Resultat (visualisiert)

Interface (Verwendung)

Wie bereits erwähnt, kann der Schema_Analyzer (Python Komponente) unabhängig vom Rest der Applikation eingesetzt werden. Im Kapitel «Schnittstellen» kann die Beschreibung des Command Line Interface gefunden werden.

Metadaten Detektion

Um die Stations- und Blattnummer aus der Fusszeile zu lesen, werden mit Tesseract alle Textblöcke gesucht und anschliessend gegen ein Regex Pattern verglichen. Das Pattern zum Finden Der Blattnummer ist wie folgt aufgebaut: `"[0-9]{3}(/|-)[0-9]+((/|-)[0-9]+)?(/|-)?[0-9]*"` . Es kann mit Blattnummern umgegangen werden, die mit / oder – unterteilt sind. Um die Stationsnummer (grüne Nummer) zu finden, wird die TextBox gesucht, welche direkt über der Blattnummer liegt. Die Referenzwerte zum Finden dieser TextBox sind eine maximale Verschiebung von 100 Pixel in y-Richtung und einen Spielraum von je 50 Pixeln links und rechts in x-Richtung. Bei Tests hat sich gezeigt, dass dabei praktisch bei jeder Fusszeilenvariante der richtige Textblock als Stationsnummer festgelegt wird. (grüne Box)

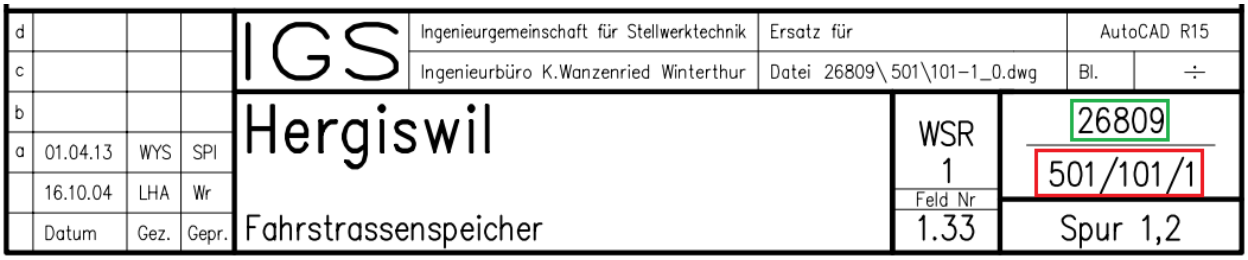


Figure 55 Fusszeilen Metadaten

Resultat

Unterhalb ist ein Beispiel des Inhalts einer Resultat JSON-Datei ersichtlich. Diese setzt sich aus den zuvor beschriebenen SymbolDetector und CharacterRecognizer Resultaten zusammen. Neben diesen beiden Resultaten, werden zusätzlich Metainformationen zur analysierten Seite mit ins Resultat gespeichert. Dabei handelt es sich zum einen um die, aus der Fusszeile gelesenen Informationen zur Stations- sowie Blattnummer. Zum anderen werden Informationen wie die Seitennummer im Dokumentations-PDF oder den Namen und den Pfad zur dazugehörigen Bilddatei hinterlegt. Dieses Resultat wird ins Dateisystem an den Ort gespeichert, wo sich die zu analysierende Bilddatei befindet. Von dort wird das Resultat von der .NET Komponente gelesen, weiter analysiert und schlussendlich abgespeichert. Sollte es nicht möglich sein, die Stations- oder Blattnummer zu detektieren, werden diese beiden Werte auf -1 gesetzt. Bei der nachfolgenden weiterführenden Analyse in der .NET Komponente wird die Stationsnummer einer Anlagedokumentation auf den meistvorkommenden detektierten Wert aus den einzelnen Seiten gesetzt.

```
{
    "Header": {
        "StationNumber": "26809",
        "SchemaNumber": "501/101/8",
        "File": "Hergiswil_page_287",
        "Page": "287",
        "Type": "tif",
        "Path": ".\\Hergiswil_page_287.tif"
    },
    "Symbols": [{
        "Type": "Einzelrelais",
        "Confidence": "0.86364204",
        "Rectangle": {
            "X": "1850",
            "Y": "205",
            "Width": "103",
            "Height": "107"
        },
        "TextAnchors": [{
            "X": "1901",
            "Y": "258"
        },
        {...}
        ],
        "TargetSymbolTypeConversions": null,
        "AssignmentDistanceThreshold": 100
    }],
    "Texts": [{
        "Text": "FAR",
        "Confidence": "0.96",
        "Rectangle": {
            "X": "1959",
            "Y": "236",
            "Width": "50",
            "Height": "25"
        }
    },
    {...}
    ]
}
```


Figure 56 Python Komponente Resultat einer Seitenanalyse

«Schema_Analyzer» Konfiguration

In diesem Abschnitt wird auf die verschiedenen Konfigurationsmöglichkeiten der «Schema_Analyzer» Komponente eingegangen. Diese Konfigurationsmöglichkeiten erlauben es die Python Komponente universell einzusetzen und fortlaufend weiter für den gegebenen Anwendungsfall zu optimieren.

Ordnerstruktur

Im Distributionsordner der «AI Interlocking» Applikation gibt es ein Unterverzeichnis «SchemaAnalyzer_py». Darin ist die Python Komponente für die Analyse einzelner Schemaseiten abgelegt. Die hauptsächlichen Konfigurationsmöglichkeiten sind mit dem SymbolDetector verbunden. Im SymbolDetector Ordner kann eine CSV-Datei gefunden werden, die alle möglichen Relaisbezeichnungen beinhaltet. Alle erkannten Relaisbezeichnungen müssen einem in dieser Liste vorhandenen Eintrag entsprechen. Weitere Informationen zur Konfiguration der Relaisbezeichnungen können dem Abschnitt «Mögliche Relais-Bezeichnungen» entnommen werden. Weiter können im SymbolTemplates Ordner die zu detektierenden Symbol-Templates gefunden werden. Jeder Ordnername entspricht einem Symboltypen, wobei es sich bei den «Unknown» und «AdditionalInfoSymbols» um spezielle Typen handelt. Die «Unknown» Symbole werden nicht ins Analyseresultat aufgenommen, aber bei der Bildvorverarbeitung aus dem Schema entfernt, um Probleme bei der Texterkennung zu verhindern. Die «AdditionalInfoSymbols» bieten zusätzliche Konfigurationsmöglichkeiten, auf welche unterhalb genauer eingegangen wird. Dem SymbolTemplates Ordner können beliebig neue Unterverzeichnisse hinzugefügt werden, um damit neue Symboltypen zu erstellen. In den Verzeichnissen der einzelnen Typenordner, sind zum einen alle Template-Bilder abgelegt, und zum anderen gibt es eine «properties.json» Datei, welche die Konfiguration eines spezifischen Symboltypen und den dazugehörigen Templates erlaubt. Unterhalb wird vertieft auf die einzelnen Konfigurationsmöglichkeiten eingegangen.

```
SymbolDetector
    RelaisPurposes.csv

SymbolTemplates
    AdditionalInfoSymbols
        arrow_down.png
        properties.json

    Anschluss
        anschluss.png
        properties.json

    Connector
        connector.png
        properties.json

    Einzelrelais
        einzelrelais.png
        properties.json

    Oeffner
        oeffner1.png
        oeffner2.png
        properties.json

    Schliesser
        schliesser1.png
        schliesser2.png
        properties.json

    Stripspunkt
        strips.png
        properties.json

    Unknown
        properties.json
```

Figure 57 SymbolDetector Konfiguration Ordnerstruktur

Mögliche Relais-Bezeichnungen

Wie bereits erwähnt, werden Relaisbezeichnungen unter Verwendung des FuzzyWuzzy Algorithmus gegen eine Liste möglicher Bezeichnungen verglichen. Diese möglichen Bezeichnungen sind als CSV-Datei unter «RelaisPurposes.csv» abgelegt. Wenn zusätzliche Bezeichnungen möglich werden sollen, kann die Liste einfach ergänzt werden. Auch das Entfernen von Einträgen aus der Liste ist möglich. Bei den einzelnen Einträgen handelt es sich UTF-8 codierte Bezeichnungen. Diese können also Unicode Zeichen beinhalten, beispielsweise für Pfeile. Unterhalb ist eine Auflistung aller zurzeit konfigurierten Möglichkeiten ersichtlich:

```
fs , fs (KRK) , (SG) , GRS , HSP , HSR , KSP , RE , SPZ , SR , SSR1 , SSR2 , VSR
, WSP , WSR , WÜR , ZGR , ZSR , ZVF , ZVR , ZVZ , 1F , 2F , 3F , 4F , 5.1F , 5.2F
, 6F , A , A1 , A2 , A3 , AA , Aa , AA/NS , AA/SH , AA/ZS , AA1/T , AA2/T , AAT ,
AAW , AH , AM , AM/GT , AM/T , AM/ZK , AM1.1 , AM1.2 , AM1.3 , AM2.1 , AM2.2 ,
ARL , AS , AU , AU1 , ↥ AV , BA , BA/GT , BA/GTR , BA/TFU1 , BA/TFU2 , BAD , BAH
, ↧ Bes.E , ↥ Bes.E , BLI/S , BS/GT , D , DFÜ , Dfü , DSA , DSÜ , ↟ DSV , ↧ DV ,
↥ DV , EÜL , EÜR , ↟ EV , EVA/GT , EVE/GT , EVA/TFU1 , EVA/TFU2 , F , F/RM1 ,
F/RM2 , F/RM3 , f/rm4 , F/RM5 , F1 , F1 , 3 , f2 , F2 , F3 , F5 , FAL , FAR , ↧
FB/1 , ↥ FB/1 , ↧ FB/2 , ↥ FB/2 , ↧ FB/3 , ↥ FB/3 , FB/T1 , FB/T2 , FB/TFU , FBN
, FBN/TFU , ↧ FBZ/1 , ↥ FBZ/1 , ↧ FBZ/2 , ↥ FBZ/2 , FBZ/T , FBZ/TFU , ff , ff1 ,
FF , FRM , FÜ , fü , FÜL , füL , FÜR , füR , ↡ FÜRL , füS , ↡ FÜS , ↧ FV , ↥ FV ,
FW , GN , GNRM , GNRM1 , GNS , GSA/GT , GSE/GT , GSA/TFU1 , GSA/TFU2 , ↥ GSP1 ,
GSP2 , ↥ H/RM1 , ↥ H/RM2 , HI/GT , HI/T , HIRM , HIRM1 , HIS1 , HIS2 , ↥ HRM , ↥
IS , ↥ IS1 , ↥ Is , ↥ Is.1 , ↥ Is.2 , ↧ L1 , ↥ L1 , ↧ L2 , ↥ L2 , ↧ L3 , ↥ L3 , L4
, M , MR , MRZ , MZ , MZ (2F) , NA , NA/GT , NA/T , ↥ NAV1 , ↥ NAV1/W , ↥ NAV2 ,
↥ NAV2/W , NH/RM , ↥ NHW , OB/T , OB/TFU , OBN/T , OG , OGRM , OGRM1 , OGS (4F) ,
P , PR , ↥ RM1 , ↥ RM2 , RT , s , S , ↧ S1 , ↥ S1 , ↧ S1.1 , ↥ S1.1 , ↧ S1.2 , ↥
S1.2 , ↧ S1.3 , ↥ S1.3 , S2 , S2.1 , S2.2 , S2.3 , SAL , SAL1 , SAR , SAR1 , SF ,
SH , ↡ SH , ↥ SHA/NG , SHE , SHE/NG , SHE1 , ↡ SHH , SHRM , SHT , SL/GT , SL/GTR
, SNH/GT , SNH/GTR , SPS , ↥ SR , ↥ SR/NS , ↥ SR/SH , ↥ SSR , ↥ SSR1 , ST , ST/GT
, ↥ ST/PR , ↥ ST/PZ , ST/TFB , ST/TFU , STP , ↥ SU , ↥ SU 12 , ↥ SU 48 , SÜL ,
SÜR , ↧ SV , ↥ SV , SV , SVP , ts , TS , ↥ TSR , TZ , ↥ Ü , ↥ Ü1 , ↥ Ü2 , ↥ Ü3 , ↥
V/RM1 , ↥ V/RM2 , VAa , ↡ VFÜ , vp , VP , ↧ VR , ↥ VR , VS1.1 , VS1.2 , VS2 , VS3
, VS4 , VS5 , ↟ VSW , ↡ WA1 , WA2 , ↥ WEVA , WIU/GT , WIU/T , ↥ WSR , WT , ↥ WTP1
, WTP2 , ↧ WV , ↥ WV , ↧ WVE , ↥ WVE , z , Z , ZE , ZIRM , ZIRM1 , ZIS (1F) , ZK
, ZR , ZR/NAV , ZR/NAV/W , ↥ ZSR , ZT , ZU , ↧ ZV , ↥ ZV , ↧ ZV1 , ↥ ZV1 , ↡ ZV2
```

Figure 58 Auflistung der möglichen Relaisbezeichnungen

SymbolTemplate Konfiguration

In diesem Abschnitt wird genauer auf die Konfigurationsmöglichkeiten einzelner Symbol-Templates und den damit verbundenen Template-Bilder eingegangen. Alle verfügbaren konfigurierbaren Parameter werden dabei aufgelistet, Beschrieben und deren Verwendung mit einem Beispiel demonstriert.

Konfigurierbare Parameter:

ConfidenceThreshold

Beschreibung	Beim ConfidenceThreshold handelt es sich um den minimal benötigten Prozentsatz, der aussagt, wie sicher sich der SymbolDetector ist, dass es sich um ein bestimmtes Symbol handelt. Nur Symbole, deren Konfidenz über dem hier konfigurierten Wert liegen, werden ins Resultat aufgenommen. Dieser Parameter kann pro Symbol festgelegt werden und wird dann für alle zu diesem Symbol hinterlegten Templates verwendet.
Default Wert	0.8 = 80%
Beispiel	`"ConfidenceThreshold": 0.75,`

Figure 59 ConfidenceThreshold Beispiel

AssignmentDistanceThreshold

Beschreibung	Beim AssignmentDistanceThreshold handelt es sich um die Grenze, bis zu welcher Objekte wie TextBoxen oder AdditionalInfoSymbols einem Symbol zugewiesen werden. Alle zuweisbaren Objekte im Umkreis dieses DistanceThresholds, gemessen von den TextAnchors des zugehörigen Symbol-Templates kommen als Eigenschaften dieser Symbolinstanz in Frage. Dieser Parameter kann pro Symbol festgelegt werden und wird dann für alle zu diesem Symbol hinterlegten Templates verwendet.
Default Wert	100px auf 300dpi Schema-Bild (ca. 8.5mm)
Beispiel	`"AssignmentDistanceThreshold": 50`

Figure 60 AssignmentDistanceThreshold Beispiel

Templates

Beschreibung	Beim Templates Parameter handelt es sich um ein Objekt, das SymbolTemplates als Subobjekte beinhaltet. Zu jedem der hier aufgeführten Templates gehört ein Template-Image, das sich im gleichen Ordner befinden muss wie das Properties.json, in dem dieser Parameter gesetzt werden kann. Der Name der jeweiligen Template-Image Datei ist der Property Name der dazugehörigen Konfiguration in diesem Templates Objekt. Auf die einzelnen Konfigurationsmöglichkeiten pro Template wird unterhalb eingegangen.

Default Wert	Wenn keine speziellen Template Konfigurationen gemacht werden sollen, kann dieser Parameter leer gelassen werden. In diesem Fall werden für alle Template-Images die Default-Konfigurationen verwendet. Auf die Default-Konfigurationen der Templates wird unterhalb eingegangen.
Beispiel	

```
"Templates": {
    "template1.png": {
      “Template Konfiguration”
    },
    "template2.png": {
      “Template Konfiguration”
    },
    "templateN.png": {
      “Template Konfiguration”
    }
}
```

Figure 61 Templates Konfigurationsbeispiel

Templates:“TemplateFileName”:TextAnchors

Beschreibung	Beim TextAnchors Parameter handelt es sich um ein Array, das (X/Y) Koordinaten beinhaltet. Jede der in diesem Array hinterlegten Koordinaten wird verwendet, um die Distanz zu zuweisbaren Objekten (TextBoxen) zu berechnen. Wenn die gemessene Distanz kleiner ist als der zuvor erwähnte AssignmentDistanceThreshold, wird das zuweisbare Objekt als Kandidat für ein Symbol-Property verwendet.
Default Wert	Die X und Y Werte werden in Pixel angegeben relativ zum Template-Image (300dpi). Wenn dieses Property leer gelassen wird, wird der Mittelpunkt des jeweiligen Template-Image als TextAnchor verwendet.
Beispiel	

```
"TextAnchors": [{
    "X": 66,
    "Y": 10
    },{
    "X": 250,
    "Y": 110
}]
```

Figure 62 TextAnchors Konfigurationsbeispiel

Templates:“TemplateFileName”:TargetSymbolTypeConversions

Beschreibung	Das TargetSymbolTypeConversions Property kommt bei AdditionalInfoSymbols zum Einsatz. Bei diesem Property handelt es sich um ein Objekt, in dem definiert wird, welchen Symbolen das zugehörige AdditionalInfoSymbol zugewiesen werden kann. Zudem wird pro zuweisbarem Objekt eine Transformation definiert. Wenn es sich beim zuweisbaren Objekt um ein Symbol handelt, wird der Symboltyp durch den zweiten Teil der Konfiguration ersetzt. Mit der unterhalb ersichtlichen Konfiguration würde beispielsweise ein Schliesser in einen Öffner umgewandelt werden und umgekehrt, wenn der Pfeil (zugehöriges Template-Image) am nächsten bei einem dieser Symbole ist. Ist der Pfeil näher bei einer TextBox, wird deren Text durch den zweiten Teil der Konfiguration ersetzt, wobei {$} durch den bisherigen Text ersetzt wird. In diesem Fall würde also ein Unicode Pfeil vor den detektierten Text gesetzt.
Default Wert	Dieser Parameter hat nur Einfluss auf AdditionalInfoSymbols und muss dort gesetzt werden. Bei allen anderen Symbolen kann dieser Parameter weggelassen werden oder er wird ignoriert.
Beispiel	

```
"TargetSymbolTypeConversions": {
    "Schliesser": "Oeffner",
    "Oeffner": "Schliesser",
    "TextBox": "↥ {$}"
}
```

Figure 63 TargetSymbolTypeConversions Konfigurationsbeispiel

Initial konfigurierte Symbole

In diesem Abschnitt sind die aktuell konfigurierten Symbol-Templates mit den dazugehörigen Template-Bilder und Konfigurationen aufgelistet. Dabei gibt es zu beachten, dass die Template-Bilder für die Symboldetektion der originalen Plangrösse entsprechen müssen.

Einzelrelais	
	einzelrelais.png
	einzelrelais_double.png
	einzelrelais_thick.png
	einzelrelais_thin.png

Konfiguration

```
{
    "ConfidenceThreshold": 0.7,
    "AssignmentDistanceThreshold": 100
}
```

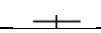
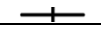
Figure 64 Einzelrelais Konfiguration

Oeffner	
	oeffner1.png
	oeffner2.png

Konfiguration

```
{
    "ConfidenceThreshold": 0.82,
    "AssignmentDistanceThreshold": 50,
    "Templates": {
        "oeffner1.png" : {
            "TextAnchors": [{
                "X": 12,
                "Y": 8
            }]
        },
        "oeffner2.png" : {
            "TextAnchors": [{
                "X": 30,
                "Y": 10
            }]
        }
    }
}
```

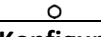
Figure 65 Öffner Konfiguration

Schliesser	
	schliesser1.png
	Schliesser2.png

Konfiguration

```
{
    "ConfidenceThreshold": 0.82,
    "AssignmentDistanceThreshold": 50,
    "Templates": {
        "schliesser1.png" : {
            "TextAnchors": [{
                "X": 66,
                "Y": 10
            }]
        },
        "schliesser2.png" : {
            "TextAnchors": [{
                "X": 73,
                "Y": 11
            }]
        }
    }
}
```

Figure 66 Schliesser Konfiguration

Anschluss	
	anschluss.png

Konfiguration

```
{
    "ConfidenceThreshold": 0.7,
    "AssignmentDistanceThreshold": 30,
    "Templates": {
        "anschluss.png" : {
            "TextAnchors": [{
                "X": 11,
                "Y": 11
            }]
        }
    }
}
```

Figure 67 Anschluss Konfiguration

Connector	
	connector_links.png
	connector_rechts.png
	connector_oben.png
	connector_unten.png

Konfiguration

```
{
    "ConfidenceThreshold": 0.75,
    "AssignmentDistanceThreshold": 30,
    "Templates": {
        "connector_links.png" : {
            "TextAnchors": [{
                "X": 30,
                "Y": 15
            }]
        },
        "connector_oben.png" : {
            "TextAnchors": [{
                "X": 15,
                "Y": 33
            }]
        },
        "connector_rechts.png" : {
            "TextAnchors": [{
                "X": 19,
                "Y": 15
            }]
        },
        "connector_unten.png" : {
            "TextAnchors": [{
                "X": 15,
                "Y": 17
            }]
        }
    }
}
```

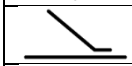
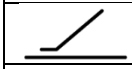
Figure 68 Connector Konfiguration

AdditionalInfoSymbols	
↧	arrow_down.png
↧	arrow_thin.png
↥	arrow_up.png
⇟	double_arrow_small.png
⇟	double_arrow_thick.png

Konfiguration

```
{
    "ConfidenceThreshold": 0.8,
    "Templates": {
        "arrow_thin.png": {
            "TargetSymbolTypeConversions": {
                "Schliesser": "Oeffner",
                "Oeffner": "Schliesser",
                "TextBox": "↧ {$}"
            }
        },
        "arrow_down.png": {
            "TargetSymbolTypeConversions": {
                "Schliesser": "Oeffner",
                "Oeffner": "Schliesser",
                "TextBox": "↧ {$}"
            }
        },
        "arrow_up.png": {
            "TargetSymbolTypeConversions": {
                "Schliesser": "Oeffner",
                "Oeffner": "Schliesser",
                "TextBox": "↥ {$}"
            }
        },
        "double_arrow_small.png": {
            "TargetSymbolTypeConversions": {
                "TextBox": "⇟ {$}"
            }
        },
        "double_arrow_thick.png": {
            "TargetSymbolTypeConversions": {
                "TextBox": "⇟ {$}"
            }
        }
    }
}
```

Figure 69 AdditionalInfoSymbols Konfiguration

Unknown	
	arrow_end_relais.png
	arrow_end_relais2.png
	fahrrichtung1.png
	fahrrichtung2.png

Konfiguration

```
{
    "ConfidenceThreshold": 0.75
}
```

Figure 70 Unknown Symbol Konfiguration

Installationsanleitung

Die Installationsanleitung befindet sich in einem separaten Dokument, das mit der Applikation ausgeliefert wird. Darin werden alle notwendigen Schritte zur Installation und initialen Konfiguration der Anlagendokumentationsanalyse-Software beschrieben. Genauere Informationen zum Installations- respektive Konfigurationsprozess können dem Dokument Betreibsanleitung.pdf unter dem Kapitel «Installationsanleitung» entnommen werden.

Gebrauchsanweisung

Die Gebrauchsanweisung der Applikation befindet sich in einem separaten Dokument, das mit der Applikation ausgeliefert wird. Darin werden alle grundlegenden Funktionalitäten und deren Verwendung über das GUI genauer erklärt. Zudem wird dort auch noch einmal ein Überblick über die Konfigurationsmöglichkeiten der Applikation geboten. Genauere Informationen zum Gebrauch der Anlagendokumentationsanalyse-Software können dem Dokument Betreibsanleitung.pdf unter den Kapiteln «Konfiguration des Analyzers», «Handhabung der grafischen Oberfläche» entnommen werden.

Datenspeicherung

Die Analyseresultate werden in einer in der aktuellen Version in einer relationalen SQLite Datenbank gespeichert. Für die DB-Anbindung wird Entity-Framework Core 5 als ER-Mapper verwendet. Dabei wurde ein Code-First Ansatz gewählt, bei dem die Datenbankversionen aus den Modelklassen erstellt werden. Bei diesem Vorgang werden die verschiedenen Datenbankversionen über EF-Migrations verwaltet, was den grossen Vorteil mit sich bringt, dass einfach zwischen den verschiedenen Versionen vor und zurück migriert werden kann und diese ein Teil der Quellcodeverwaltung sind. Dies funktioniert, weil sich jede Datenbankinstanz merkt, welche Migrationen bei ihr eingespielt sind. (__EFMIgrationsHistory Tabelle) Unterhalb ist das generierte Datenbankschema als ER-Diagramm ersichtlich. Dabei handelt es sich grundsätzlich um eine 1:1 Abbildung des Domain Modells, das im Kapitel «Domainanalyse» genauer beschrieben ist. Speziell zu erwähnen, gibt es die die Umsetzung der Vererbungshierarchie, welche Rund um die Symbolklasse existiert. Dabei wurde entschieden die Klassen in einer «Table-per-type» Konfiguration umzusetzen, was erst mit EF-Core 5.0 eingeführt worden ist. Dieser Entscheid wurde trotz den möglichen Performanz Einbussen mit der Begründung einer verbesserten Übersichtlichkeit gefällt. Dabei wird im Gegensatz zum «Table-per-hierarchy» Ansatz jede Klasse als eigenständige Tabelle umgesetzt, wobei die Primärschlüssel der Abgeleiteten Tabellen auf die Basistabelle verweisen, in der die geteilten Daten abgelegt sind.

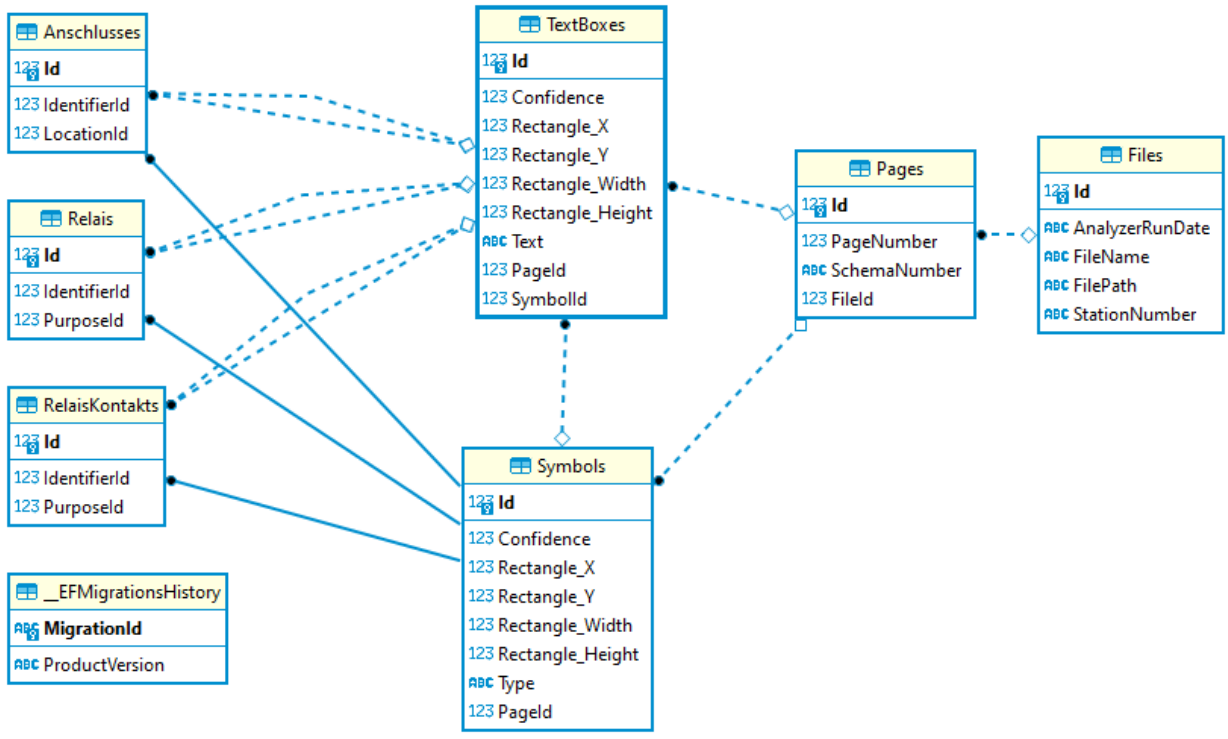


Figure 71 Datenbankschema

Neben den in der Datenbank gespeicherten Analyseresultaten wird jeweils eine Kopie der ursprünglich analysierten Anlagedokumentation im Dateisystem unter «.\\AnalyzedFiles\\» gespeichert. Der PDF-Exporter verwendet diese als Grundlage und reichert diese mit den detektierten Symbolen und Texten an.

Qualitätssicherung

Einführung

In diesem Dokument wird die Qualitätssicherung des Projektes «AI for Relay Interlocking» beschrieben, welches eine Übersicht für die getätigten Q-Massnahmen geben soll. Diese Qualitätssicherung dient als Grundlage für alle im Projektverlauf durchgeführten Qualitätssicherungsmassnahmen.

Q-Massnahmen

Unterhalb ist eine Liste mit allen Q-Massnahmen ersichtlich, welche angewendet worden sind, um ein möglichst qualitativ hochwertiges Produkt abliefern zu können.

- Bugs werden immer auf GitLab erfasst
- Stunden-Erfassung auf den Arbeitspaketen / Issues im GitLab
- Code-Reviews werden bei jedem Merge Request gemacht
- Unit Tests (Microtesting und Integration Tests) mit einer Mindestabdeckung von 80%. Auswertung auf Sonarqube
- Metriken beim Build auf CI Server. Auswertung auf Sonarqube
- Static Code Analysis mittels Sonarqube laufen auf dem CI Server.
- Github Workflow mit Feature Branches und Merge Requests als minimales Vier-Augen-Prinzip
- Systemtests (Use Case) werden immer vor Produktionsübernahme (Production Merge Request) gemacht
- Sonarqube Instanz wird für die Auswertung genutzt
    - Test-Coverage: Min. 80 %
    - Duplicated Lines: Max: 5 %
    - Maintainability Rating: Min. A
    - Reliability Rating: Min. A

Sicherung der Entwicklungshistorie

Die nachfolgenden Informationen werden für eine gute Nachvollziehbarkeit des Entwicklungsprozesses gesichert:

- Ganze Code Basis ist auf GitLab versioniert.
- Projektdokumentationen werden in einem Microsoft Teams Team verwaltet, geteilt und versioniert.
- Unterlagen, die bei der Entwicklung des Produktes benötigt worden sind (Schulungsunterlagen, Anlagedokumentation etc.) wurden über Microsoft Teams von Siemens Mobility AG geteilt und somit allen beteiligten Personen zur Verfügung gestellt und versioniert.
- Metrik- und Code Analysis-Auswertungen sind im Sonarqube ersichtlich. Zusätzlich wird für die Schlussabgabe ein Export aus Sonarqube generiert.

Systemtest Spezifikation

Einführung

In diesem Abschnitt wird die Systemtestspezifikation des Projektes «AI for Relay Interlocking» beschrieben, welche eine Übersicht über die geplanten Systemtests geben soll. Informationen zu den Testergebnissen können dem Kapitel «Systemtest Protokoll» entnommen werden.

AI Interlocking .NET Komponente

Test 1: Analyse einer Anlagedokumentation GUI

Beschreibung	Eine Anlagedokumentation soll analysiert werden. Dabei soll sichergestellt werden, dass dieser Prozess erfolgreich abgeschlossen wird und das Analyseresultat verfügbar ist.
Voraussetzungen	- Im Ordner «Interlocking.Service.Test\Data» befindet sich eine verkleinerte Anlagedokumentation. (HergiswilMini.pdf) - GUI ist geöffnet und FileProcessor Fenster ist aktiv.
Input / Interaktion	1. Auf «Select File» klicken und zuvor erwähnte Datei im Dateiexplorer suchen 2. Auf «Process File» klicken
Erwarteter Output	Analyseprozess wird gestartet und Fortschritt wird angezeigt (0/2). Sobald die Analyse abgeschlossen ist, kann das Fenster geschlossen werden und der analysierte Dateiname wird mit dem Datum und der Urzeit in der Resultatauflistung angezeigt.

Test 2: Abbruch eines Analyseprozesses

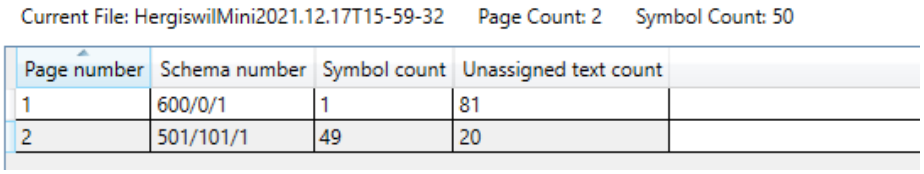
Beschreibung	Ein laufender Analyseprozess soll abgebrochen werden. Dabei soll sichergestellt werden, dass alle Subprozess erfolgreich beendet werden und das System in einem sauberen Status ist.
Voraussetzungen	- Im Ordner «Interlocking.Service.Test\Data» befindet sich eine verkleinerte Anlagedokumentation. (HergiswilMini.pdf) - GUI ist geöffnet und FileProcessor Fenster ist aktiv.
Input / Interaktion	1. Auf «Select File» klicken und zuvor erwähnte Datei im Dateiexplorer suchen 2. Auf «Process File» klicken 3. Warten bis Progress (1/2) anzeigt 4. Auf «Abort» klicken
Erwarteter Output	Der Analyseprozess wird abgebrochen und alle dabei aktiven Subprozesse (Python Komponenten Instanzen) werden beendet. Nach dem Abbruch ist kein Analyseresultat zu diesem Analysevorgang ersichtlich.

Test 3: Analyse einer Anlagedokumentation CLI

Beschreibung	Eine Anlagedokumentation soll analysiert werden. Dabei soll sichergestellt werden, dass dieser Prozess erfolgreich abgeschlossen wird und das Analyseresultat verfügbar ist.
Voraussetzungen	- Im Ordner «Interlocking.Service.Test\Data» befindet sich eine verkleinerte Anlagedokumentation. (HergiswilMini.pdf) - Eine Konsole ist geöffnet und befindet sich im Verzeichnis, in dem sich die Konsolenapplikation befindet.
Input / Interaktion	1. Exe mit folgenden Argumenten ausführen: .\\Interlocking.Service.Test\\Data\\HergiswilMini.pdf -s 5 6 2. Warten bis Analyse abgeschlossen
Erwarteter Output	Analyseprozess wird gestartet und Fortschritt wird angezeigt. Sobald die Analyse erfolgreich abgeschlossen ist, kann die GUI Applikation geöffnet werden und der analysierte Dateiname wird mit dem Datum und der Urzeit in der Resultatauflistung angezeigt.

Test 4: Anzeigen lassen des Analyseresultates

Beschreibung	Ein Analyseresultat soll angezeigt werden. Dabei soll sichergestellt werden, dass die Informationen zu allen analysierten Seiten vorhanden sind.
Voraussetzungen	- Das Dokument HergiswilMini.pdf wurde erfolgreich analysiert - Das Resultat dieser Analyse ist auf dem FileSelector Fenster aufgelistet
Input / Interaktion	1. Gewünschtes Resultat in Auflistung auswählen 2. Auf «Open File» klicken
Erwarteter Output	Es werden zwei Einträge in der Auflistung angezeigt. Diese können beliebig sortiert werden mit dem Klicken auf die Spalteninformationen. Die Übersicht der Seiten sieht folgendermassen aus:


Current File: HergiswilMini2021.12.17T15-59-32 Page Count: 2 Symbol Count: 50

Page number	Schema number	Symbol count	Unassigned text count
1	600/0/1	1	81
2	501/101/1	49	20

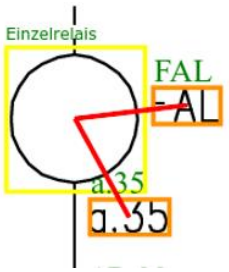
Test 5: Visualisieren einer analysierten Seite

Beschreibung	Das Analyseresultat einer Seite soll visualisiert werden. Dabei soll überprüft werden, ob die detektierten Informationen am richtigen Ort dargestellt werden und die Visualisierung geöffnet werden kann.
Voraussetzungen	- Das Dokument HergiswilMini.pdf wurde erfolgreich analysiert - Das Resultat dieser Analyse ist auf dem FileSelector Fenster aufgelistet
Input / Interaktion	1. Gewünschtes Resultat in Auflistung auswählen 2. Auf «Open File» klicken 3. Seite 2 in Auflistung auswählen 4. Auf «Edit selected Page» klicken 5. «Visualize» anklicken
Erwarteter Output	Ein Ladeindikator erscheint und die Interaktionsmöglichkeiten mit der Seite werden ausgeschaltet. Sobald das Resultat visualisiert worden ist, wird diese mit dem Standard-Bildviewer angezeigt. Alle Symbole sind dabei richtig erkannt worden und alle TextBoxen wurden dem richtigen Symbol zugewiesen. Visualisiertes Bild kann im %Temp% Ordner gefunden werden.

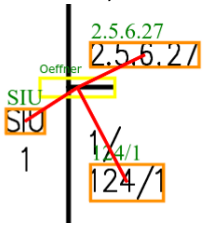
Test 6: Bearbeiten einer TextBox in Resultat

Beschreibung	Es soll geprüft werden, ob erkannte TextBoxen bearbeitet und die Änderungen gespeichert werden können. Alle Änderungen können zudem mit der zuvor getesteten Visualisierungsfunktionalität veranschaulicht werden.
Voraussetzungen	- Das Dokument HergiswilMini.pdf wurde erfolgreich analysiert - Das Resultat dieser Analyse ist auf dem FileSelector Fenster aufgelistet
Input / Interaktion	1. Gewünschtes Resultat in Auflistung auswählen 2. Auf «Open File» klicken 3. Seite 2 in Auflistung auswählen 4. Auf «Edit selected Page» klicken 5. Folgendes Symbol auswählen und auf «Edit selected Symbol» klicken: 6. Text 0.35 zu a.35 ändern und Fenster schliessen 7. Auf FileContent Fenster auf «Save» klicken
Erwarteter Output	Die Änderung wurde abgespeichert und ist somit auch nach einem Neustart der Applikation verfügbar. Wenn auf dem PageContent Fenster «Visualize» gedrückt wird, ist der Text beim Relais oben links korrekt abgeändert:

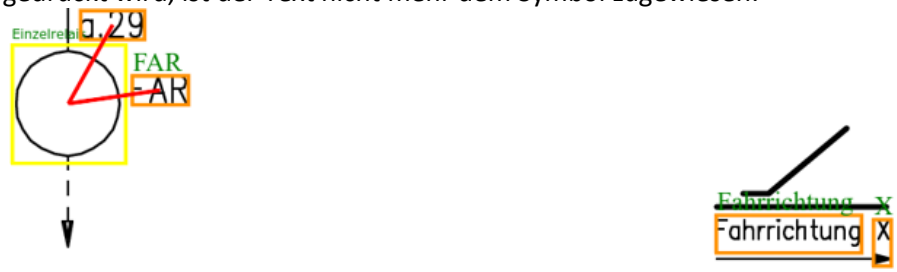
Connector	0.81	1035	722	30	50	3.8
Einzelrelais	0.94	527	205	103	107	0.35, FAL
Connector	0.80	1035	2321	30	50	1.8



Test 7: Zuweisen von «Unassigned Text» zu detektiertem Symbol

Beschreibung	Es soll überprüft werden, ob fälschlicherweise nicht zugewiesener Text einem Symbol zugewiesen werden kann und diese Änderung abgespeichert werden kann. Alle Änderungen sollen zudem mit der zuvor getesteten Visualisierungsfunktionalität veranschaulicht werden können.
Voraussetzungen	- Das Dokument HergiswilMini.pdf wurde erfolgreich analysiert - Das Resultat dieser Analyse ist auf dem FileSelector Fenster aufgelistet
Input / Interaktion	1. Gewünschtes Resultat in Auflistung auswählen 2. Auf «Open File» klicken 3. Seite 2 in Auflistung auswählen 4. Auf «Edit selected Page» klicken 5. Folgende Auswahl machen: Symbols: Type | Confidence | Position X | Position Y | Size Width | Size Height | Texts Oeffner | 0.89 | 1491 | 675 | 80 | 20 | 27, 1/10, SIU, 2.1.6. Oeffner | 0.89 | 1491 | 911 | 80 | 20 | SIU, 2.5.6.27 Unassigned texts: Text | Confidence | Position X | Position Y | Size Width | Size Height b | 0.56 | 852 | 1200 | 17 | 70 124/1 | 0.95 | 1546 | 1004 | 76 | 37 6. Auf «Assign selected text to symbol» klicken 7. Auf FileContent Fenster auf «Save» klicken
Erwarteter Output	Die Änderung wurde abgespeichert und ist somit auch nach einem Neustart der Applikation verfügbar. Wenn auf dem PageContent Fenster «Visualize» gedrückt wird, ist der Text dem richtigen Symbol zugewiesen: 

Test 8: Entfernen von fälschlich zugewiesenem Text

Beschreibung	Es soll überprüft werden, ob fälschlicherweise zugewiesener Text von einem Symbol entfernt werden kann und diese Änderung abgespeichert werden kann. Alle Änderungen sollen zudem mit der zuvor getesteten Visualisierungsfunktionalität veranschaulicht werden können.
Voraussetzungen	- Das Dokument HergiswilMini.pdf wurde erfolgreich analysiert - Das Resultat dieser Analyse ist auf dem FileSelector Fenster aufgelistet - Die Fahrrichtung TextBox ist dem Relais links unten zugewiesen: 
Input / Interaktion	1. Gewünschtes Resultat in Auflistung auswählen 2. Auf «Open File» klicken 3. Seite 2 in Auflistung auswählen 4. Auf «Edit selected Page» klicken 5. Betroffenes Relais Symbol auswählen und auf «Edit selected Symbol» klicken 6. Fahrrichtung auswählen und auf «Unassign selected text» drücken
Erwarteter Output	Die Änderung wurde abgespeichert und ist somit auch nach einem Neustart der Applikation verfügbar. Wenn auf dem PageContent Fenster «Visualize» gedrückt wird, ist der Text nicht mehr dem Symbol zugewiesen: 

Test 9: Löschen eines fälschlich erkannten Symbols

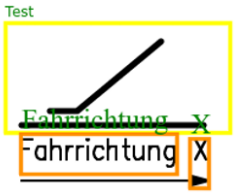
Beschreibung	Es soll geprüft werden, dass fälschlicherweise erkannte Symbole aus dem Resultat gelöscht werden können und diese Änderung abgespeichert werden kann. Alle Änderungen sollen zudem mit der zuvor getesteten Visualisierungsfunktionalität veranschaulicht werden können.
Voraussetzungen	- Das Dokument HergiswilMini.pdf wurde erfolgreich analysiert - Das Resultat dieser Analyse ist auf dem FileSelector Fenster aufgelistet
Input / Interaktion	1. Gewünschtes Resultat in Auflistung auswählen 2. Auf «Open File» klicken 3. Seite 1 in Auflistung auswählen 4. Auf «Edit selected Page» klicken 5. Einziges erkanntes Symbol auswählen und mit «Delete» löschen 6. Auf FileContent Fenster auf «Save» klicken
Erwarteter Output	Die Änderung wurde abgespeichert und ist somit auch nach einem Neustart der Applikation verfügbar. Wenn auf dem PageContent Fenster «Visualize» gedrückt wird, ist das gelöschte Symbol nicht mehr markiert.

Test 10: Manuelles Hinzufügen eines nicht erkannten Symbols

Beschreibung	Es soll geprüft werden, dass fälschlicherweise nicht erkannte Symbole dem Resultat hinzugefügt werden können und diese Änderung abgespeichert werden kann. Alle Änderungen sollen zudem mit der zuvor getesteten Visualisierungsfunktionalität veranschaulicht werden können.
Voraussetzungen	- Das Dokument HergiswilMini.pdf wurde erfolgreich analysiert - Das Resultat dieser Analyse ist auf dem FileSelector Fenster aufgelistet
Input / Interaktion	1. Gewünschtes Resultat in Auflistung auswählen 2. Auf «Open File» klicken 3. Seite 2 in Auflistung auswählen 4. Auf «Edit selected Page» klicken 5. In der Symbol Auflistung einen Neuen Eintrag machen mit den folgenden Daten: 6. Auf FileContent Fenster auf «Save» klicken

Type	Confidence	Position X	Position Y	Size Width	Size Height	Texts
Test	1.00	1154	2818	190	93	

Erwarteter Output	Die Änderung wurde abgespeichert und ist somit auch nach einem Neustart der Applikation verfügbar. Wenn auf dem PageContent Fenster «Visualize» gedrückt wird, ist das neu erstellte Symbol markiert.



Test 11: Löschen einer fälschlich erkannten TextBox

Beschreibung	Es soll geprüft werden, dass fälschlicherweise erkannte Symbole aus dem Resultat gelöscht werden können und diese Änderung abgespeichert werden kann. Alle Änderungen sollen zudem mit der zuvor getesteten Visualisierungsfunktionalität veranschaulicht werden können.
Voraussetzungen	- Das Dokument HergiswilMini.pdf wurde erfolgreich analysiert - Das Resultat dieser Analyse ist auf dem FileSelector Fenster aufgelistet
Input / Interaktion	1. Gewünschtes Resultat in Auflistung auswählen 2. Auf «Open File» klicken 3. Seite 1 in Auflistung auswählen 4. Auf «Edit selected Page» klicken 5. Alle Texte nacheinander auswählen und mit «Delete» löschen 6. Auf FileContent Fenster auf «Save» klicken
Erwarteter Output	Die Änderung wurde abgespeichert und ist somit auch nach einem Neustart der Applikation verfügbar. Wenn auf dem PageContent Fenster «Visualize» gedrückt wird, ist kein markierter Text mehr zu sehen auf der Seite.

Test 12: Manuelles Hinzufügen eines nicht erkannten Textes

Beschreibung	Es soll geprüft werden, dass fälschlicherweise nicht erkannte TextBoxen dem Resultat hinzugefügt werden können und diese Änderung abgespeichert werden kann. Alle Änderungen sollen zudem mit der zuvor getesteten Visualisierungsfunktionalität veranschaulicht werden können.
Voraussetzungen	- Das Dokument HergiswilMini.pdf wurde erfolgreich analysiert - Das Resultat dieser Analyse ist auf dem FileSelector Fenster aufgelistet
Input / Interaktion	1. Gewünschtes Resultat in Auflistung auswählen 2. Auf «Open File» klicken 3. Seite 2 in Auflistung auswählen 4. Auf «Edit selected Page» klicken 5. In der Unassigned Text Auflistung einen Neuen Eintrag machen mit den folgenden Daten: 6. Auf FileContent Fenster auf «Save» klicken
Erwarteter Output	Die Änderung wurde abgespeichert und ist somit auch nach einem Neustart der Applikation verfügbar. Wenn auf dem PageContent Fenster «Visualize» gedrückt wird, ist der neu erstellte Text markiert und beschriftet. (Unten im Dokument)

Text	Confidence	Position X	Position Y	Size Width	Size Height
TEST_TEXT	1.00	1150	3000	40	24

Test 13: Löschen eines Analyseresultates

Beschreibung	Es soll überprüft werden, ob gesamte Analyseresultate gelöscht werden können.
Voraussetzungen	- Das Dokument HergiswilMini.pdf wurde erfolgreich analysiert - Das Resultat dieser Analyse ist auf dem FileSelector Fenster aufgelistet
Input / Interaktion	1. Gewünschtes Resultat in Auflistung auswählen 2. Auf «Delete File» klicken 3. Im Pop Up Dialog bestätigen, dass das Analyseresultat tatsächlich gelöscht werden soll.
Erwarteter Output	Die Änderung wurde abgespeichert und das gelöschte Analyseresultat ist auch nach einem Neustart der Applikation nicht mehr auf der Übersichtsseite aufgelistet.

Test 14: Exportieren eines Analyseresultates in ein PDF

Beschreibung	Es soll geprüft werden, ob ein Analyseresultat als PDF exportiert werden kann. Dabei soll nach den detektierten Texten gesucht werden können. Für den Export können verschiedene Informationslevel ausgesucht werden.
Voraussetzungen	- Das Dokument HergiswilMini.pdf wurde erfolgreich analysiert - Das Resultat dieser Analyse ist auf dem FileSelector Fenster aufgelistet
Input / Interaktion	1. Gewünschtes Resultat in Auflistung auswählen 2. «Export File» anklicken 3. PDF als Zielformat auswählen 4. Folgende Auswahl treffen und anschliessend Export klicken: ☑ Mark symbols ☑ Mark text ☑ Mark assignments ☑ Draw rectangles 5. Zieldatei im Dateiexplorer angeben und bestätigen 6. Warten, bis Prozess abgeschlossen ist
Erwarteter Output	Am angegebenen Zielpfad wurde eine PDF-Datei erstellt. Diese Datei beinhaltet alle Seiten des ursprünglich analysierten Dokumentes. Jede analysierte Seite wurde mit den detektierten Informationen angereichert. Die dabei markierten Texte können mit jedem normalen PDF-Reader gesucht werden.

Test 15: Exportieren eines Analyseresultates in Bilddateien

Beschreibung	Es soll geprüft werden, ob ein Analyseresultat als Bilddateien exportiert werden kann. Für den Export können verschiedene Informationslevel ausgesucht werden.
Voraussetzungen	- Das Dokument HergiswilMini.pdf wurde erfolgreich analysiert - Das Resultat dieser Analyse ist auf dem FileSelector Fenster aufgelistet
Input / Interaktion	1. Gewünschtes Resultat in Auflistung auswählen 2. «Export File» anklicken 3. TIFF als Zielformat auswählen 4. Folgende Auswahl treffen und anschliessend Export klicken: ☑ Mark symbols ☑ Mark text ☑ Mark assignments ☑ Draw rectangles 5. Zielverzeichnis im Dateiexplorer angeben und bestätigen 6. Warten, bis Prozess abgeschlossen ist

Erwarteter Output	Am angegebenen Zielpfad wurde pro Seite eine Bilddatei erstellt. Diese wurde für alle Seiten des ursprünglich analysierten Dokumentes generiert. Jede analysierte Seite wurde mit den detektierten Informationen angereichert.

Test 16: Analyse mit verkleinertem «AssignmentDistanceThreshold»

Beschreibung	Der Einfluss des AssignmentDistanceThreshold sollte geprüft werden. Dieser soll bei einem Symboltypen verkleinert werden und dabei soll geprüft werden, ob die Zuweisungslogik korrekt damit umgehen kann.
Voraussetzungen	- Im Ordner «Interlocking.Service.Test\Data» befindet sich eine verkleinerte Anlagedokumentation. (HergiswilMini.pdf) - GUI ist geöffnet und FileProcessor Fenster ist aktiv. - AssignmentDistanceThreshold der Einzelrelais Konfiguration wurde von 100 auf 10 reduziert
Input / Interaktion	1. Auf «Select File» klicken und zuvor erwähnte Datei im Dateiexplorer suchen 2. Auf «Process File» klicken 3. Warten, bis Prozess abgeschlossen ist
Erwarteter Output	Der AssignmentDistanceThreshold ist jetzt so klein, dass keinem der detektierten Relais auf Seite 2 eine TextBox zugewiesen worden ist. Der Konfigurierbare Parameter hat also Einfluss aufs Resultat und erfüllt seinen Zweck. 

Schema_Analyzer Python Komponente

Test 1: Analyse einer A4 Schemaseite

Beschreibung	Bei diesem Test soll die Möglichkeit geprüft werden, eine A4 Schemaseite zu Analysieren. Dabei soll das Resultat als JSON-Datei im Dateisystem am gleichen Ort abgelegt werden wie das zu analysierende Bild.
Voraussetzungen	- Im Verzeichnis «Interlocking.Service.Test\Data» befindet sich eine Bilddatei «HergiswilMini_page_2.tif» , bei dem es sich um ein Bild einer A4 Seite handelt. - Eine Konsole ist geöffnet und befindet sich im «SchemaAnalyzer_py» Verzeichnis
Input / Interaktion	1. Richtige Python Instanz starten 2. SchemaAnalyzer.py mit den folgenden Argumenten aufrufen: -f C:\\.....\\ HergiswilMini_page_2.tif -s 5,6 3. Warten, bis Analyse abgeschlossen ist.

Erwarteter Output	Im Verzeichnis, an dem das zu analysierende Bild gespeichert ist, wurde eine JSON-Datei mit dem Namen HergiswilMini_page_2.json erstellt. Der Header der erstellten Datei sieht folgendermassen aus:
	"Header": {
	"StationNumber": "26809",
	"SchemaNumber": "501/101/1",
	"File": "HergiswilMini_page_2",
	"Page": "2",
	"Type": "tif",
	"Path": ".\\Data\\HergiswilMini_page_2.tif"
	}
	Zudem beinhaltet das Symbols-Array 49 Einträge und das Texts-Array 109 Einträge.

Test 2: Analyse einer A3 Schemaseite

Beschreibung	Bei diesem Test soll die Möglichkeit geprüft werden, eine A3 Schemaseite zu Analysieren. Dabei soll das Resultat als JSON-Datei im Dateisystem am gleichen Ort abgelegt werden wie das zu analysierende Bild.
Voraussetzungen	- Im Verzeichnis «Interlocking.Service.Test\Data» befindet sich eine Bilddatei «HergiswilMini_page_1.tif», bei dem es sich um ein Bild einer A3 Seite handelt. - Eine Konsole ist geöffnet und befindet sich im «SchemaAnalyzer_py» Verzeichnis
Input / Interaktion	1. Richtige Python Instanz starten 2. SchemaAnalyzer.py mit den folgenden Argumenten aufrufen: -f C:\\.....\\ HergiswilMini_page_1.tif -s 5,6 3. Warten, bis Analyse abgeschlossen ist.

Erwarteter Output	Im Verzeichnis, an dem das zu analysierende Bild gespeichert ist, wurde eine JSON-Datei mit dem Namen HergiswilMini_page_1.json erstellt. Der Header der erstellten Datei sieht folgendermassen aus:
	"Header": {
	"StationNumber": "26809",
	"SchemaNumber": "600/0/1",
	"File": "HergiswilMini_page_1",
	"Page": "1",
	"Type": "tif",
	"Path": ".\\Data\\HergiswilMini_page_1.tif"
	}
	Zudem beinhaltet das Symbols-Array 1 Einträge und das Texts-Array 82 Einträge.

Test 3: Analyse einer handschriftlichen Schemaseite

Beschreibung	Bei diesem Test soll die Möglichkeit geprüft werden, eine handschriftliche Schemaseite zu Analysieren. Dabei soll das Resultat als JSON-Datei im Dateisystem am gleichen Ort abgelegt werden wie das zu analysierende Bild.
Voraussetzungen	- Im Verzeichnis «Interlocking.Service.Test\Data» befindet sich eine Bilddatei «Arth_Goldau_page_1635.tif», bei dem es sich um ein Bild einer handschriftlichen Schemaseite handelt. - Eine Konsole ist geöffnet und befindet sich im «SchemaAnalyzer_py» Verzeichnis
Input / Interaktion	1. Richtige Python Instanz starten 2. SchemaAnalyzer.py mit den folgenden Argumenten aufrufen: -f C:\\.....\\ Arth_Goldau_page_1635.tif -s 5,6 3. Warten, bis Analyse abgeschlossen ist.
Erwarteter Output	Im Verzeichnis, an dem das zu analysierende Bild gespeichert ist, wurde eine JSON-Datei mit dem Namen Arth_Goldau_page_1635.json erstellt. Der Header der erstellten Datei sieht folgendermassen aus:

```
"Header": {
    "StationNumber": "27010",
    "SchemaNumber": "501/101/5",
    "File": "Arth_Goldau_page_1635",
    "Page": "1635",
    "Type": "tif",
    "Path": ".\\Data\\Arth_Goldau_page_1635.tif"
}
```

Zudem beinhaltet das Symbols-Array 26 Einträge und das Texts-Array 93 Einträge.

Test 4: Analyse einer Schemaseite mit Visualisierung des Resultates

Beschreibung	Bei diesem Test soll die Möglichkeit geprüft werden, eine Schemaseite zu Analysieren. Dabei soll das Resultat als JSON-Datei sowie zusätzlich als visualisierte Bilddatei im Dateisystem am gleichen Ort abgelegt werden wie das zu analysierende Bild.
Voraussetzungen	- Im Verzeichnis «Interlocking.Service.Test\Data» befindet sich eine Bilddatei «HergiswilMini_page_2.tif» , bei dem es sich um ein Bild einer A4 Seite handelt. - Eine Konsole ist geöffnet und befindet sich im «SchemaAnalyzer_py» Verzeichnis
Input / Interaktion	1. Richtige Python Instanz starten 2. SchemaAnalyzer.py mit den folgenden Argumenten aufrufen: -f C:\\.....\\ HergiswilMini_page_2.tif -s 5,6 --save_res_img 3. Warten, bis Analyse abgeschlossen ist.
Erwarteter Output	Im Verzeichnis, an dem das zu analysierende Bild gespeichert ist, wurde neben der bereits geprüften JSON-Datei mit dem Namen HergiswilMini_page_2.json eine Bilddatei mit dem Namen HergiswilMini_page_2_result.tif erstellt. Auf diesem Bild sind alle detektierten Symbole und alle erkannten TextBoxen farblich hervorgehoben.

Test 5: Analyse einer Schemaseite mit nicht passender Sektionsnummer

Beschreibung	Bei diesem Test soll das Verhalten geprüft werden, wenn das zu analysierende Schema nicht einer der selektierten Sektionen entspricht.
Voraussetzungen	- Im Verzeichnis «Interlocking.Service.Test\Data» befindet sich eine Bilddatei «HergiswilMini_page_2.tif» , bei dem es sich um ein Schema der Sektion 5xx handelt. - Eine Konsole ist geöffnet und befindet sich im «SchemaAnalyzer_py» Verzeichnis
Input / Interaktion	1. Richtige Python Instanz starten 2. SchemaAnalyzer.py mit den folgenden Argumenten aufrufen: -f C:\\.....\\ HergiswilMini_page_2.tif -s 6 3. Warten, bis Analyse abgeschlossen ist.

Erwarteter Output

Die Schemaseite wird nicht analysiert. Nachdem die Blattnummer aus der Fusszeile mit den zu analysierenden Sektionen verglichen worden ist, wird das Resultat ins Dateisystem geschrieben, ohne den eigentlichen Schemateil zu analysieren. Das Resultat sieht folgendermassen aus:

```
{
    "Header": {
        "StationNumber": "26809",
        "SchemaNumber": "501/101/1",
        "File": "HergiswilMini_page_2",
        "Page": "2",
        "Type": "tif",
        "Path": ".\\Data\\HergiswilMini_page_2.tif"
    },
    "Symbols": null,
    "Texts": null
}
```

Test 6: Konfiguration eins zusätzlichen Symbols

Beschreibung	Bei diesem Test soll das Konfigurieren eines neuen Symboltyps getestet werden. Das neukonfigurierte Symbol sollte vom SymbolDetector auf der zu analysierenden Schemaseite detektiert werden können.
Voraussetzungen	- Im Verzeichnis «Interlocking.Service.Test\Data» befindet sich eine Bilddatei «HergiswilMini_page_2.tif» - Im SymbolTemplates Verzeichnis wurde ein neuer Ordner erstellt mit dem Namen «Fahrrichtungsanzeige» - Die Template-Bilder fahrrichtung1.png und fahrrichtung2.png wurde vom Ordner «Unknown» in den neu erstellten Ordner «Fahrrichtungsanzeige» verschoben. - Eine Konsole ist geöffnet und befindet sich im «SchemaAnalyzer_py» Verzeichnis
Input / Interaktion	1. Richtige Python Instanz starten 2. SchemaAnalyzer.py mit den folgenden Argumenten aufrufen: -f C:\\.....\\ HergiswilMini_page_2.tif -s 6 --save_res_img 3. Warten, bis Analyse abgeschlossen ist.

Erwarteter Output Im Resultat JSON gibt es ein Symbol mit dem Typ Fahrrichtungsanzeige, welches in der Visualisierung am richtigen Ort dargestellt wird.

Systemtest Protokoll

Einführung

In diesem Abschnitt wird das Systemtestprotokoll des Projektes «AI for Relay Interlocking» beschrieben, welches eine Übersicht über die Resultate der manuell ausgeführten Tests bieten soll. Dabei wird auf die Im Kapitel «Systemtest Spezifikation» definierten Testfälle Bezug genommen.

AI Interlocking .NET Komponente

TestCase	Status	Kommentar	Datum	Tester
Test 1	Erfolgreich	Anlagendokumentationsausschnitt konnte erfolgreiche analysiert werden.	20.12.2021	Christian Bisig
Test 2	Erfolgreich	Analyseprozess wurde abgebrochen, wobei alle beteiligten Prozesse beendet wurden.	20.12.2021	Christian Bisig
Test 3	Erfolgreich	Analyseprozess konnte erfolgreich über CLI gestartet werden, wobei der Status angezeigt wurde.	20.12.2021	Christian Bisig
Test 4	Erfolgreich	Analyseresultat wurde vollständig angezeigt und es liess sich durchnavigieren.	20.12.2021	Christian Bisig
Test 5	Erfolgreich	Analyseresultat einer spezifischen Seite konnte erfolgreich angezeigt werden.	20.12.2021	Christian Bisig
Test 6	Erfolgreich	Falsch erkannter Text konnte korrigiert werden und zurück ins Resultat gespeichert werden.	20.12.2021	Christian Bisig
Test 7	Erfolgreich	Fälschlicherweise nicht zugewiesener Text konnte einem Symbol zugewiesen werden.	20.12.2021	Christian Bisig
Test 8	Erfolgreich	Fälschlicherweise zugewiesener Text konnte von einem Symbol entfernt werden.	20.12.2021	Christian Bisig
Test 9	Erfolgreich	Fälschlicherweise erkanntes Symbol konnte erfolgreich aus dem Resultat entfernt werden.	20.12.2021	Christian Bisig
Test 10	Erfolgreich	Fälschlicherweise nicht erkanntes Symbol konnte manuell dem Resultat hinzugefügt werden.	20.12.2021	Christian Bisig
Test 11	Erfolgreich	Fälschlicherweise erkannte TextBox konnte erfolgreich aus dem Resultat entfernt werden.	20.12.2021	Christian Bisig

Test 12	Erfolgreich	Fälschlicherweise nicht erkannte TextBox konnte manuell dem Resultat hinzugefügt werden.	20.12.2021	Christian Bisig
Test 13	Erfolgreich	Das gewählte Analyseresultat konnte erfolgreich von der Datenbank gelöscht werden.	20.12.2021	Christian Bisig
Test 14	Erfolgreich	Das Analyseresultat konnte problemlos als durchsuchbares PDF exportiert werden. Es wurden alle geforderten Informationen angezeigt.	20.12.2021	Christian Bisig
Test 15	Erfolgreich	Das Analyseresultat konnte problemlos als Kollektion von Bildern im TIFF-Format exportiert werden. Es wurden alle geforderten Informationen angezeigt.	20.12.2021	Christian Bisig
Test 16	Erfolgreich	Nachdem der AssignmentDistanceThreshold heruntergesetzt worden ist, wurden den Relais korrekterweise keine TextBoxen mehr zugewiesen.	20.12.2021	Christian Bisig

Table 6 Systemtestprotokoll .NET

Schema_Analyzer Python Komponente

TestCase	Status	Kommentar	Datum	Tester
Test 1	Erfolgreich	A4 Seite konnte erfolgreich analysiert werden und die Resultat-Datei wurde ans richtige Ort geschrieben und wies die geforderten Eigenschaften auf.	20.12.2021	Christian Bisig
Test 2	Erfolgreich	A3 Seite konnte erfolgreich analysiert werden und die Resultat-Datei wurde ans richtige Ort geschrieben und wies die geforderten Eigenschaften auf.	20.12.2021	Christian Bisig
Test 3	Erfolgreich	Handschriftliche Seite konnte erfolgreich analysiert werden und die Resultat-Datei wurde ans richtige Ort geschrieben und wies die geforderten Eigenschaften auf.	20.12.2021	Christian Bisig
Test 4	Erfolgreich	Das visualisierte Resultat wurde als Bild ans gleiche Ort geschrieben wie das JSON File. Auf dem Bild konnten alle detektierten Objekte hervorgehoben erkannt werden.	20.12.2021	Christian Bisig
Test 5	Erfolgreich	Die Seite wurde korrekterweise nicht analysiert. Die Resultat-Datei wurde ans richtige Ort geschrieben und wies die geforderten Eigenschaften auf.	20.12.2021	Christian Bisig

Test 6	Erfolgreich	Der hinzugefügte Symboltyp wurde auf dem Schema detektiert und erfolgreich ins Resultat aufgenommen. Zudem war er auf der Visualisierung des Resultates zu erkennen.	20.12.2021	Christian Bisig

Table 7 Systemtestprotokoll Python

Nicht funktionale Anforderungen (NFR)

TestCase	Status	Kommentar	Datum	Tester
NFR-1	Erfüllt	Logging ist eingebaut, Fehler werden in eine Textdatei geschrieben und können von dort für die Fehlerbehebung verwendet werden. Die Logdatei befindet sich im Ordner «.\\Logs»	21.12.2021	Christian Bisig
NFR-2	Erfüllt	Bei der Eingabe einer korrupten PDF-Datei, stürzt das Programm nicht ab, sondern benachrichtig den User über das Problem und schreibt die genaueren Informationen in das Logfile, das unter «.\\Logs» gefunden werden kann.	21.12.2021	Christian Bisig
NFR-3	Erfüllt	Durch die angestrebte Parallelisierung kann das Ziel im Bereich des Zeitverhalten eingehalten werden. Die Analyse der Anlagedokumentation von Hergiswil mit 1500 Seiten konnte in 90 Minuten abgeschlossen werden. Hochgerechnet auf 5000 Seiten sind das ca. 5 Stunden	21.12.2021	Dominic Klinger
NFR-4	Erfüllt	Alle Funktionalitäten können einfach am erwarteten Ort gefunden werden. Zudem wurde eine Bedienungsanleitung für die Einführung in die Applikation erstellt.	21.12.2021	Christian Bisig
NFR-5	Erfüllt	Die einzelnen Komponenten sind durch die gewählte Architektur sehr lose gekoppelt. Es ist somit unproblematisch möglich, dies auszutauschen ohne Änderungen am Rest der Applikation vornehmen zu müssen. Alle definierten Code Metriken konnten eingehalten werden.	21.12.2021	Christian Bisig
NFR-6	Erfüllt	Es stehen 54 automatisierte Unit- und Integration Tests zur Verfügung. Diese können verwendet werden, um zu überprüfen, ob eine Änderung, die Funktionalität der Applikation beeinträchtigt hat. Diese 54 Testfälle führen zu einer Testabdeckung von 89.75% und somit über die definierte 80% Grenze.	21.12.2021	Christian Bisig

Table 8 NFR Testprotokoll

Entscheidungen

In diesem Abschnitt werden die wichtigsten Entscheidungen noch einmal kurz zusammengeführt und erläutert. Genauere Informationen zu den jeweiligen Entscheidungen können den dazugehörigen Abschnitten in der Produktdokumentation entnommen werden.

Verwendung von Python zur Schemaanalyse

In der Elaborationsphase wurde der Entscheid gefällt, dass zwei Technologien verwendet werden sollen, um die geforderte Lösung aufzubauen. Dabei soll die eigentliche Bildanalyse in Python implementiert werden. Neben den zur Schemabildanalyse gehörenden Funktionalitäten zur Symboldetektion sowie der Texterkennung soll auch die gesamte Bildvorverarbeitung in Python durchgeführt werden. Diese Python Komponente soll auch unabhängig vom Rest des Systems funktionieren können und als eigenständige Software über ein CLI einsetzbar sein. Der Entscheid wurde basierend auf den nachfolgenden Punkten gefällt:

- Breite Palette an ausgereiften Bibliotheken im Bereich der Bildverarbeitung
- Gute OpenCV Unterstützung für die Bild Vorverarbeitung
- Bessere Integration von Tesseract in pytesseract im Vergleich zum .NET Wrapper
- Ausführlichere Konfigurationsmöglichkeiten von Tesseract
- Prototyp mit den besten Ergebnissen bei der Symboldetektion (OpenCV Template-Matching)
- Prototyp mit der besten Texterkennungsrate (Tesseract mit vortrainierten Modellen für Latein und Deutsch)

Dagegen gesprochen haben am Schluss lediglich die Einbussen im Bereich der Performanz durch das Erstellen eines neuen Prozesses für jede zu analysierende Seite.

Verwendung von .NET für die Gesamtapplikation

Nachdem der Entscheid gefällt wurde, die eigentliche Schemaanalyse in Python zu implementieren, wurde darüber nachgedacht, die gesamte Software in Python umzusetzen, um die Bandbreite an verschiedenen eingesetzten Technologien im Rahmen zu halten. Dies Variante wurde zum Schluss aber von den Vorteilen der Verwendung von .NET übertroffen. Es wurde entschieden, den restlichen Teil der Applikation im .NET Umfeld umzusetzen. Zum restlichen Teil der Applikation gehört die Vorbereitung der PDF-Anlagedokumentationen zur Analyse von der Python Komponente, die Zuweisung der TextBoxen und Symbole, die Speicherung der Analyseresultate sowie die Exportfunktionalitäten der Resultate in verschiedene Formate. Folgende Punkte haben zum gefällten Entscheid geführt:

- Gute Unterstützung von ER-Mapping zur Abbildung des Domain Modelles
- Aussage vom Kunden, dass bereits sehr viele Applikationen im .NET Bereich im Einsatz sind
- Sauber Architekturpattern bekannt und gut umsetzbar
- Durch die saubere Architektur gute Erweiterbarkeit der Applikation gegeben
- Erfahrung im Entwicklerteam viel grösser als im Python Bereich

Verwendung von Tesseract in bestimmter Konfiguration für die Texterkennung

Für die Texterkennung wurden verschiedene Prototypen aufgebaut und evaluiert. Die mit den Prototypen erzielten Ergebnisse wurden verglichen, wobei sich herauskristallisiert hat, dass sich mit Tesseract in bestimmten Konfigurationen die besten Ergebnisse erzielen lassen. Die ersten Tesseract Prototypen wurden mit einem .NET Wrapper aufgebaut, der nicht im gleichen Ausmass konfigurierbar ist wie die Python Integration. Aus diesem Grund wurde entschieden, auf pytesseract aufzubauen. Bei weiteren Evaluationsschritten für die bestmögliche Konfiguration hat sich gezeigt, dass mit dem OCR Engine Mode (OEM) 1, der auf LSMT neuronalen Netzen basiert in Kombination mit dem Page Segmentation Mode (PSM) 12, der möglichst viel Text sucht unter Verwendung von Orientation and Script Detection (OSD), die besten Ergebnisse erzielt werden können. Zudem wurden noch weiter Parameter gesetzt, zu welchen die genaue Erklärung im Kapitel «Optical Character Recognition» gefunden werden kann. Zum Entscheid zur Verwendung von Tesseract haben folgende Punkte geführt:

- Sehr gute Python Integration
- Umfangreiche Konfigurationsmöglichkeiten
- Möglichkeit zum Trainieren eigener Modelle
- Möglichkeit zur Kombination von eigenen Modellen mit Vortrainierten Modellen
- Gute Dokumentation und viele Beispielprojekte
- Beste Ergebnisse in der Prototypisierungsphase

Verwendung von OpenCV Template-Matching für die Symboldetektion

Für die Symboldetektion wurden verschiedenste Methoden evaluiert. Angefangen wurde bei eigens trainierten neuronalen Netzen, die für die Object Detection eingesetzt werden konnten. Es wurden Versuche mit TensorFlow sowie ML.NET trainierten Netzen durchgeführt, wobei keine überzeugenden Resultate erzielt werden konnten. Eingesetzt wurden die Netze im ONNX Format im .NET Umfeld. Es hat sich rasch herausgestellt, dass für eine zuverlässige Funktionsweise dieser Methode, sehr viele getaggte Trainingsdaten notwendig wären. Als nächstes wurden Template-Matching Algorithmen evaluiert dabei wurde unter anderem ein Prototyp mit QATM (Quality-Aware Template Matching For Deep Learning) umgesetzt, mit dem gute Ergebnisse erzielt werden konnten. Die Patter-Matching Funktionalität von OpenCV, die zur Evaluation ebenfalls in einem Prototyp angewendet worden ist, hat zum Schluss jedoch die besten Ergebnisse geliefert, was zum Entscheid geführt hat, diese Technologie für die Symboldetektion zu verwenden. Genauere Informationen zur Funktionsweise von OpenCV Template-Matching können dem Kapitel «Symbol Detektion» entnommen werden. Folgende Punkte waren ausschlaggebend bei der Entscheidung:

- Nur wenige Daten benötigt für gute Ergebnisse
- Wertvolle Konfigurationsmöglichkeit für den Vergleichsoperator gegeben
- Hohe Matching Geschwindigkeit (sehr gute Performanz)
- Einfach aber sehr effektiv einsetzbar
- Hervorragende Ergebnisse in der Prototypisierungsphase

Verwendung von SQLite als Datenbankprodukt

Die Analyseresultate sollen in einer Datenbank abgespeichert werden können, um weitere, auf den gewonnenen Daten basierende, Analysemöglichkeiten zu schaffen. Da die aktuelle Version der Analyseapplikation eine Desktopapplikation ist, musste ein Datenbankprodukt gefunden werden, das möglichst ohne Installation einfach einsetzbar ist. Neben dem Einsatz von (SQL / NoSQL) Datenbankprodukten wurde auch die Variante von einfachen Dateien im Dateisystem geprüft. In der Elaborationsphase wurde entschieden, auf SQLite Datenbanken zu setzen, da diese Filebasiert funktionieren und trotzdem viele Möglichkeiten von relationalen Datenbanken zur Verfügung stellen. Durch den losen gekoppelten Aufbau der Applikation ist ein Austausch der Datenbanktechnologie jederzeit problemlos und ohne Änderungen am bestehenden Code möglich. Hauptargumente, welche für die Verwendung von SQLite gesprochen haben können, der unten ersichtlichen Auflistung entnommen werden.

- Keine Installation notwendig
- Gute Weiterverwendbarkeit der Daten gegeben
- Gute Integration mit EF Core als Entity Mapper möglich
- Leichtgewichtigste Variante, die alle geforderten Funktionalitäten bietet

Verwendung von WPF für das User Interface

Für das User Interface wurde entschieden WPF zu verwenden. Das UI sollte möglichst einfach die gesamte Funktionalität der Analyseapplikation zur Verfügung stellen. Im .NET Umfeld handelt es sich bei WPF um die ausgereifteste Technologie zur Entwicklung von Desktop Apps. Daneben wurde auch das mit .NET 6 neu erschienene MAUI als Möglichkeit in Betracht gezogen. Mit der Verwendung von MAUI wären modernere cross-plattform User Interfaces möglich gewesen. Da während der Elaborationsphase aber .NET 6 erst als Release Kandidat verfügbar war, musste sicherheitshalber auf .NET 5 und WPF zurückgegriffen werden. Es wurde entschieden, dass Mitte November kein Upgrade auf, das neu erschienene, .NET 6 Framework durchgeführt wird, sondern die Applikation weiter auf der .NET 5 Basis entwickelt wird. Durch den modularen Aufbau der Applikation ist jedoch jederzeit eine Änderung der Frontend Technologie möglich, ohne dabei den existierenden Code anpassen zu müssen. Es wäre sogar denkbar, dass in Zukunft die Logik in eine Webapplikation ausgelagert wird und so die Analyseergebnisse einfache geteilt werden könnten. Folgende Punkte waren ausschlaggebend für den Entscheid zur Verwendung von WPF:

- Ausgereifteste Technologie für Desktopapplikationen im .NET Umfeld
- Bekannte und gut einsetzbare Pattern zur Datenanbindung (MVVM)
- .NET 6 und MAUI noch nicht verfügbar zu Beginn des Projektes
- Open Source Komponenten Pakete frei verfügbar (z.B. MahApps.Metro)

Schlussbericht

Einführung

In diesem Abschnitt kann der Schlussbericht des Projektes «AI for Relay Interlocking» gefunden werden. Dieser enthält abschliessende und zusammenfassende Worte zum gesamten Projektverlauf. Die Zielerreichung des Projektes, als Team gesammelte und persönliche Erfahrung jedes Teammitgliedes werden in diesem Abschnitt erläutert.

Zielerreichung

Zum Schluss der Projektdauer kann ein Produkt vorgewiesen werden, das wie gefordert Informationen aus Anlagedokumentationen extrahieren kann. Die gewonnenen Informationen können zurück in die Anlagedokumentation geschrieben werden, was die Suche nach bestimmten Elementen erlaubt. Daneben können die Resultate auch als Bilddateien exportiert werden, was eine gute Veranschaulichung der Funktionsweise der Applikation bietet. Analyseresultate können über das User Interface korrigiert und angepasst werden. Über das CLI kann der Analyseprozess aus anderen Programmen oder Scheduling Tools angestossen und somit automatisiert werden. Durch die umfangreich gegebenen Konfigurationsmöglichkeiten ist die Applikation auch für zukünftige und möglicherweise zurzeit noch unbekannten Schemavarianten geeignet. Mit einer Symbolerkennungsrate von über 90% und einer korrekten Texterkennungsrate von über 85% auf maschinell erstellten Plänen werden die Anforderungen gut erfüllt. Die zusätzliche Assoziation von Texten und Symbolen erlaubt weiterführende zukünftige Analysen zur Validierung der Anlagedokumentationen.

Weiterführende Ideen und Verbesserungsmöglichkeiten

Um die Analyseresultate einfacher teilbar zu machen und die Zusammenarbeit zwischen verschiedenen Personen zu ermöglichen, wäre die Integration der Analysesoftware in eine Webapplikation denkbar. Die Analysen könnten zentral auf einem Server durchgeführt werden und die dabei entstehenden Resultate könnten auf einem Datenbankserver abgelegt und somit für mehrere Anwender gleichzeitig verfügbar gemacht werden. Den Nutzern würde mit dieser Variante die Installation erspart bleiben und die Applikation wäre sofort einsetzbar.
Einen weiteren Punkt, den man sich überlegen müsste, wäre das partielle Neuanalysieren von Dokumenten und das Zusammenführen von Resultaten basierend auf den Blattnummern. Im Moment können Dokumentationen nur als Ganzes analysiert werden, was zu einem zeitlichen Mehraufwand führt.

Zeitauswertung

Die detaillierte Zeitauswertung kann in einer separaten Datei im Abgabeordner gefunden werden. Veranschlagt für dieses Projekt waren zwischen 448 und 480 Gesamtstunden. Die Zeitauswertung hat einen Aufwand von 58 Tagen und 7 Stunden, was 471 Stunden entspricht, ergeben. Damit liegen wir gut in dem gesetzten Rahmen der Studienarbeit.

Allgemeiner Erfahrungsbericht

Team / Organisation / Kommunikation

Die Zusammenarbeit im Team funktionierte problemlos. Die bei der Projektdurchführung benutzten Kollaborationswerkzeuge wie Gitlab, Microsoft Teams oder OneCloud haben das Arbeiten am Projekt von unterschiedlichen Orten aus, sehr gut unterstütz. Die Kommunikation mit den Ansprechpersonen bei Siemens Mobility AG hat stets sehr gut funktioniert und die in Rapperswil durchgeführte Zwischenpräsentation hat gute Inputs für die Weiterentwicklung des Produktes mit sich gebracht. Leider konnte die Schlusspräsentation, wegen der aktuellen COVID-19 Situation, nicht wie geplant in Wallisellen durchgeführt werden, sondern musste online gehalten werden.

Meetings

Während dem Projektverlauf wurden regelmässige Meetings mit dem Projektbetreuer durchgeführt. Dabei wurde jeweils der aktuelle Stand der Arbeit, sowie allfällige Probleme besprochen. Daneben wurden innerhalb des Projektteams wöchentliche Besprechungen durchgeführt, bei welchen der aktuelle Stand der Workitems abgeglichen worden ist. Zudem wurde, vor allem in der Elaborationsphase, bei der Ausarbeitung der verschiedenen Ansätze, häufig aktiv gemeinsam gearbeitet. Neben den Projektinternen Sitzungen wurde auch eine Zwischen- sowie Abschlusspräsentation mit den Ansprechpartnern bei Siemens Mobility durchgeführt. Dabei konnte auf Fragen und Anliegen von Seite Kunden eingegangen werden.

Herausforderungen

Im Verlauf des Projektes ist es immer wieder zu technisch herausfordernden Situationen gekommen. Hauptsächlich in der Elaborationsphase, beim Erstellen der Prototypen haben sich gewisse Varianten als Sackgassen herausgestellt. Dank der durchgeführten Risikoanalyse und Mitigation, hat aber keiner dieser Rückschläge den Projektverlauf in Gefahr bringen können. Die grössten technischen Herausforderungen waren die korrekte Konfiguration von Tesseract, um eine optimale Texterkennung erreichen zu können, das Trainieren von neuronalen Netzen für die Symboldetektion sowie das Finden einer passenden Bibliothek zur Verarbeitung von PDF-Dokumenten.
Herausforderungen im Bereich des Projektmanagement hat es, dank einer guten Planung und der engen Zusammenarbeit im Team trotz der erschwerenden COVID-19 Situation, keine nennenswerten gegeben.

Fazit

Trotz vielen Herausforderungen konnte am Schluss des Projektes ein Produkt vorgewiesen werden, das die funktionalen und nicht funktionalen Anforderungen erfüllt. Die bei diesem Projekt erstellte Applikation bildet eine gute Grundlage zum Aufbauen weiterer Funktionalitäten. Die weiteren Möglichkeiten wurden bei der Schlusspräsentation besprochen und auf deren Machbarkeit geprüft. Das Produkt kann als gute Unterstützung in der Verwendung von Anlagedokumentation und als Basis für Weiterentwicklungen in viele Richtungen verwendet werden.

Persönliche Erfahrungen

Dominic Klinger

Dieses Projekt war eine enorme Herausforderung, doch auch sehr spannend zugleich. Da ich keine Vorausbildung vor dem Studium hatte und als Quereinsteiger gelte und erst seit knapp über einem Jahr fest in der Informatik-Branche arbeite, konnte ich hier sehr viel Erfahrung sammeln. Viele Technologien festigen und neue kennenlernen. Das Themengebiet der Bildverarbeitung ist sehr gross und komplex. Daher sind die anfänglichen Entscheidungen nicht gerade einfach gefallen. Durch den Ansatz: «Was eignet sich am besten?» konnte man mehrere Technologien ausprobieren und testen. Das öffnet das breite Wissen, was ich immer einen sehr interessanten Aspekt finde. Im Laufe konnten die Erfahrungen gesammelt werden und zu einem Endresultat geeint werden. Gerade aber auch die organisatorischen Aspekte waren nicht zu vernachlässigen. Man hat den Fokus also nicht nur auf die technische Sicht gerichtet, sondern muss alle Aspekte abdecken. Ich finde, am Ende kam ein sehr gelungenes Produkt heraus, was eine gute Basis zum weiteren Ausbau bietet.

Christian Bisig

Dieses, im Rahmen der SA durchgeführte, Projekt war zugleich sehr spannend aber auch fordernd, da sehr viele für mich neue Technologien zum Einsatz gekommen sind. Die sehr breite Palette der verwendeten Technologien und Methoden zum Lösen einer Aufgabenstellung in einem sehr interessanten Themengebiet, haben dieses Projekt ausgemacht. Im Studium und im bisherigen Berufsalltag angeeignete Fähigkeiten mussten kombiniert werden, um eine optimale Lösung zu finden. Zum Schluss des Projektes konnte eine gute Lösung präsentiert werden, welche die zu Beginn definierten Anforderungen erfüllen kann. Die Projektdurchführung von Anfang bis Schluss ist immer wieder eine gute Gelegenheit, seine Projektmanagementfähigkeiten zu verbessern und neue Ansatzpunkte und Sichtweisen kennenzulernen.

Anhang

Literatur und Quellenverzeichnis

Herkunft der Vorlage	Das Dokument wurde auf der Basis einer Vorlage für Technische Berichte erstellt. Die Vorlage ist ein Element des „Werkzeugkastens Technische Berichte“ der Ostschweizer Fachhochschule. Sie orientiert sich an Prinzipien des Strukturierten Schreibens.
Kapitelaufteilung	Das Dokument wurde basierend auf der im Rahmen des EPJ erstellten Dokumentation strukturiert. Namentlich die Kapitelaufteilung wurde stark an diese Vorlage angelehnt.
Anlage-dokumentationen	Die bei der Durchführung dieses Projektes verwendeten Anlagedokumentationen wurden von Siemens Mobility AG zur Verfügung gestellt. Die Anlagedokumentationsdokumente sind vertraulich zu behandeln, da es sich um Dokumentationen zu aktiven Bahninfrastrukturanlagen handelt. Folgende Anlagedokumentation wurden zur Verfügung gestellt: Samedan.pdf Erstfeld.pdf Poschiavo.pdf Arth_Goldau.pdf Hergiswil.pdf Spiez.pdf
Schemaseiten / Schemaausschnitte	Die in diesem Dokument und für das Testing sowie die Konfiguration verwendeten Schemaseiten respektive Schemaausschnitte stammen aus den oben genannten Anlagedokumentationen, welche von Siemens Mobility AG zur Verfügung gestellt worden sind.
Schema Normen	Bei der Analyse der einzelnen Symbole wird Logik verwendet, welche auf den Siemens Mobility AG Normen für Anlagedokumentationen basieren. Die Normendokumente und dazugehörigen Schulungsunterlagen wurden im Rahmen dieser Arbeit von Siemens Mobility AG zur Verfügung gestellt.
ML.NET Object Detection	Der auf ML.NET basierende Prototyp zur Symboldetektion wurde basierend auf einem Microsoft Tutorial, das unter dem untenstehenden Link gefunden werden kann, aufgebaut. https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/object-detection-model-builder
ML.NET ONNX Object Detection	Der ML.NET Prototyp, der mit einem in TensorFlow trainierten neuronalen Netz im ONNX Format Symbole detektieren sollte, wurde basierend auf einer Anleitung, die unter dem untenstehenden Link gefunden werden kann, erstellt. https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/object-detection-onnx

QATM	Der Prototyp zur Symboldetektion, der auf «Quality Aware Template Matching For Deep Learning» basierte, wurde basierend auf einer nicht offiziellen Pytorch Implementation des Algorithmus entwickelt. Die Bibliothek kann unter dem untenstehenden Link gefunden werden. https://github.com/kamata1729/QATM_pytorch
OpenCV Template Matching	Die auf OpenCV basierende Implementation des SymbolDetectors wurde basierend auf der offiziellen Erklärung, welche unter dem untenstehenden Link gefunden werden kann, implementiert. https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html
OpenCV TemplateMatch-Modes	Die in der Erklärung zum SymbolDetector verwendete Formel stammt aus der OpenCV Dokumentation zum Thema Object Detection, die unter folgendem Link gefunden werden kann: https://docs.opencv.org/4.x/df/dfb/group__imgproc__object.html#gga3a7850640f1fe1f58fe91a2d7583695da5382c8f9df87e87cf1e9f9927dc3bc31
Preprocessing Strategie	Bei der Bildvorverarbeitung, die notwendig ist, um ein optimales Texterkennungsergebnis erzielen zu können, wurde auf der zusammengefassten Vorgehensweise von Susmith Reddy, die unter folgender Adresse gefunden werden kann, aufgebaut. https://towardsdatascience.com/pre-processing-in-ocr-fc231c6035a7
PdfSharpCore	Für die PDF-Bearbeitung wird die PdfSharpCore Bibliothek gesetzt. Dies kann unter der MIT Lizenz frei gebraucht werden. https://github.com/ststeiger/PdfSharpCore
Ghostscript	Für die Umwandlung von einzelnen PDF-Seiten zu Bilddateien, die analysiert werden können, wird Ghostscript als einzige Bibliothek verwendet, welche diese Funktionalität anbietet. Diese kann entweder mit der Veröffentlichung des Codes unter der GNU GPL Affero Lizenz oder unter einer Kommerziellen Lizenz verwendet werden. https://www.ghostscript.com/
Tesseract	Für die Texterkennung wird auf Tesseract 4 gesetzt. Diese OCR Engine ist unter der Apache 2.0 Lizenz frei verfügbar. Verwendet wird Tesseract über den pytesseract Python wrapper. https://github.com/tesseract-ocr/tesseract https://pypi.org/project/pytesseract/

Vortrainierte Tesseract Modelle	In der aktuellen Version der Applikation wird auf die beiden vortrainierten LSTM Modelle für Deutsch und Latein gesetzt.
	https://github.com/tesseract-ocr/tessdata_best
Magick.NET	Die Magick.NET Bibliothek wird für die Bildbearbeitung und als Wrapper um Ghostscript im .NET Teil der Applikation verwendet. Magick.NET ist unter der Apache 2.0 Lizenz frei verfügbar.
	https://github.com/dlemstra/Magick.NET
FuzzySharp	Bei FuzzySharp handelt es sich um die .NET Implementation des FuzzyWuzzy Algorithmus, der für das partielle Vergleichen der erkannten Texte mit den Möglichen Relais Bezeichnungen verwendet wird. FuzzySharp ist unter der MIT Lizenz frei verfügbar.
	https://github.com/JakeBayer/FuzzySharp
Clean Architecture .NET	Die gewählte Clean Architecture ist lose an die, unter dem folgenden Link verfügbare, Referenzarchitektur angelehnt.
	https://www.c-sharpcorner.com/article/introduction-to-clean-architecture-and-implementation-with-asp-net-core/
Repository Pattern	Das zur Entkopplung des Datenzugriffes gewählte Repository Pattern ist lose an die von Microsoft vorgeschlagenen Referenz-Implementation angelehnt.
	Implementing the Repository and Unit of Work Patterns in an ASP.NET MVC Application (9 of 10) | Microsoft Docs
StyleCop	Um die Umsetzung der festgelegten Codestyle Guidelines sicherzustellen werden StyleCopAnalyzers verwendet.
	https://github.com/DotNetAnalyzers/StyleCopAnalyzers
Dofactory	Die definierten Codestyle Guidelines basieren grösstenteils auf den dofactory Guidelines.
	https://www.dofactory.com/csharp-coding-standards
SonarQube	Zur Analyse der Code Metriken wird eine SonarQube Instanz verwendet, die auf dem Buildserver läuft.
	https://www.sonarqube.org/
ISO 25010	Norm, auf welcher die Analyse der nicht funktionalen Anforderungen durchgeführt worden ist.
	https://www.iso.org/standard/35733.html

C4 Modell	Das C4 Modell wurde als Grundlage für die Dokumentation der Architektur verwendet. https://c4model.com/

Formeln

Tabellenverzeichnis

Abbildungsverzeichnis

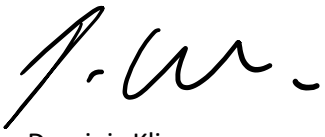
Erklärung zur Urheberschaft

Dominic Klinger

Erklärung Ich erkläre hiermit, dass ich die vorliegende Arbeit ohne Hilfe Dritter angefertigt habe. Ich habe nur die Hilfsmittel benutzt, die ich angegeben habe. Gedanken, die ich aus fremden Quellen direkt oder indirekt übernommen habe, sind kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort/Datum Arosa, 24.12.2021

Unterschrift



Dominic Klinger

Christian Bisig

Erklärung Ich erkläre hiermit, dass ich die vorliegende Arbeit ohne Hilfe Dritter angefertigt habe. Ich habe nur die Hilfsmittel benutzt, die ich angegeben habe. Gedanken, die ich aus fremden Quellen direkt oder indirekt übernommen habe, sind kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort/Datum Eschenbach, 24.12.2021

Unterschrift

Christian Bisig