# INKING AUF PDF

Bachelor Thesis


**Betreuung**
Prof. Dr. Wolfgang Weck

**Experte**
Romano Roth, Zühlke

**Projektteam**
Simon Beck
Matthias Keller

**Hochschule**
FHNW
Hochschule für Technik
Studiengang Informatik

**Auftraggeber**
PDF Tools AG



**Abgabe**
Windisch, 18.08.2017



Fachhochschule Nordwestschweiz
Hochschule für Technik

# 1 Abstract

Diese Arbeit hat zum Ziel eine bestehende Applikation zum Darstellen von PDF-Dokumenten so zu erweitern, dass handschriftliche Notizen erstellt, bearbeitet und gelöscht werden können.
Formen wie Polylines oder Rechtecke sollen zusätzlich erstellt werden können. Ebenfalls enthalten ist die Funktionalität zur Verwendung von Texterkennungsalgorithmen zum Verarbeiten von handschriftlichen Eingaben zu Text. Die Formen sowie die Texterkennung sind basierend auf dem Managed Extensibility Framework umgesetzt. Die führt dazu, dass eigene Formen entwickelt und Texterkennungsalgorithmen eingebunden werden können, ohne dass Änderungen am Programmcode der Komponente nötig sind. Das Resultat der Arbeit ist die durch die Annotationsfunktionalität erweiterte WPF-Komponente. Es wird keine komplette GUI-Applikation erstellt, sondern die Komponente erweitert, welche von Entwicklern in ihre Applikationen eingebunden werden kann. Die Analyse der bestehenden Komponente, sowie die Vorgehensweise zur Implementation der Annotations- und Erweiterungsfunktionalität wird vorgestellt und zusätzliche Erweiterungsmöglichkeiten werden aufgezeigt.

# 2 Inhaltsverzeichnis

# 3 Einleitung

Das Dateiformat PDF verfolgt das Ziel, Dokumente unabhängig vom Ursprungsprogramm oder Betriebssystem originalgetreu wiederzugeben. Dieses Format wird schon seit längerer Zeit in der Industrie und im Privatgebrauch aktiv benutzt.
In den letzten Jahren haben sich Tablets, Convertibles und Laptops mit Touchscreen immer mehr durchgesetzt. Mehr und mehr Endbenutzer finden Gefallen daran, handschriftliche Notizen direkt auf ihrem Arbeitsgerät zu erstellen, oder ihren Computer direkt mittels Fingerbewegungen zu steuern.
Diese Entwicklung hat auch unser Kunde, die PDF Tools AG, erkannt.
Momentan betreibt die PDF Tools AG eine WPF-Komponente, welche PDF-Dateien anzeigen kann. Diese Komponente kann von Entwicklern in ihre Applikationen eingebaut werden. Das Ziel dieser Arbeit ist es, diese Komponente um die Funktionalität der Freihandannotationen zu erweitern.
Diese Annotationen dienen dazu, direkt auf einer PDF-Datei mittels Maus- oder Stifteingabe Notizen zu machen. Die Komponente soll die Möglichkeit bieten, solche Annotationen anzuzeigen, zu erstellen, markieren, bearbeiten und löschen. Ebenfalls sollen verschiedene Formen direkt auf die Datei gezeichnet werden können. Zusätzlich soll es möglich sein, handschriftliche Annotationen in einen Text umzuwandeln.
In dieser Arbeit wurden diese Anforderungen umgesetzt, zusätzlich wurde ein Fokus auf die Analyse der bisherigen Architektur, sowie der Umsetzung der Annotationen im Kontext einer WPF-Komponente gelegt. Weiterhin wurde ermöglicht, dass Entwickler ihre eigenen Formen und Texterkennungsalgorithmen ohne Änderungen an der WPF-Komponente einbauen können.

Das Dokument befasst sich zuerst mit der Analyse der bestehenden Komponente, danach wird die Annotationsfunktionalität erklärt und zum Schluss wird die Umsetzung der Formen und Texterkennung erläutert.
Die einzelnen Unterthemen sind immer nach demselben Prinzip aufgebaut. Zuerst wird die Vision erklärt, danach die Umsetzung und die dabei entstandenen Herausforderungen beschrieben und zum Schluss wird überprüft, ob die Idee den Anforderungen gerecht wird.

## 3.1 Auftraggeber


*Abbildung 1: Logo der PDF Tools AG*

Die PDF Tools AG beschäftigt sich seit 15 Jahren mit dem PDF-Dateiformat. Sie ist der Schweizer Vertreter im ISO-Komitee zum PDF-Format. Ihre Produkte richten sich an Entwickler, welche PDF-Dateien in ihren Programmen verwenden. Sie entwickeln Lösungen zur Darstellung, Konvertierung, Manipulation, Optimierung und Sicherheit von PDF-Dateien.

## 3.2 Vision

Mit dieser Arbeit wird die Möglichkeit geboten, in PDF-Dateien handschriftliche Notizen zu verfassen, sei es nun durch eine Mausbewegung oder Stifteingabe auf dem Touchscreen. Diese direkt in der PDF-Datei gespeicherten Eingaben können bearbeitet und wieder gelöscht werden.
Als Referenz für die Entwicklung und die Benutzung der Annotationen diente der weitverbreitete Adobe Acrobat Viewer.
Zusätzlich zu handschriftlichen Notizen soll es möglich sein, verschiedene Formen zu erstellen. Diese sollen ebenfalls als Notizen dem PDF Dokument hinzugefügt werden.
Handgeschriebene Notizen sollen mit einem Tool zur Handschrifterkennung erkannt und in geeigneter Form in der Datei gespeichert werden.

## 3.3 Anforderungen

Zu Beginn der Arbeit wurden mehrere Anforderungen aufgestellt. Diese wurden in Zusammenarbeit mit dem Kunden, sowie dem Betreuer erarbeitet, verfeinert und in der Projektvereinbarung festgehalten. Dieses Dokument befindet sich im Anhang. Folgende Anforderungen definieren den Fokus dieser Arbeit.

*Tabelle 1: Anforderungen*

| **Name** | **Priorität** |
|---|---|
| Freihand Annotationen erstellen | 1 |
| Freitext Annotationen erstellen | 1 |
| Linienfarbe wählen | 1 |
| Linienbreite wählen | 1 |
| Annotationen löschen | 1 |
| Annotationen auswählen | 1 |
| Schaffen der Erweiterungsmöglichkeiten zu neuen Formen | 1 |
| Rechteck Annotation erstellen | 2 |
| Polyline Annotation erstellen | 2 |
| Annotationen verschieben | 2 |
| Texterkennung von Freihand Annotationen | 2 |
| Implementation der Formen Rechteck Annotation und Polyline Annotation | 2 |
| Auswechseln der Implementation der Texterkennung | 2 |
| Freihand Annotationen in Form konvertieren | 3 |
| Annotationen verziehen (skalieren) | 3 |

Damit Annotationen bearbeitet oder gelöscht werden können, ist es erforderlich, dass diese markiert werden können. Aus diesem Grund bildet das Markieren die Basis für jegliche Änderungen an bestehenden Annotationen.

Damit zusätzliche Formen erstellt werden können, müssen Erweiterungsmöglichkeiten vorhanden sein.

# 4 Ausgangslage

Der erhaltene Code sowie die dazugehörige Dokumentation und PDF-Spezifikation bilden die Ausgangslage dieser Arbeit.

## 4.1 Vision

Die Arbeit baut auf einer bestehenden WPF-Komponente auf und erweitert diese. Die zu erweiternde Komponente besteht aus mehreren Teilbereichen, welche interagieren und für diese Arbeit von unterschiedlicher Wichtigkeit sind.

Aus diesem Grund ist es für die Zielerreichung essentiell, dass die Architektur und die Abläufe der bestehenden Komponente analysiert und verstanden werden.

## 4.2 Aufbau

Bisher besteht eine WPF-Komponente, welche PDF-Dokumente öffnen, darstellen und navigieren kann. Ebenfalls ist es möglich, Textfragmente im Dokument zu markieren. Die Komponente kann von Entwicklern in beliebige WPF-Applikationen eingebunden werden, um diese Funktionalitäten zur Verfügung zu stellen.

Wie in Abbildung 1 ersichtlich, kann das Projekt, welches als Grundlage für unsere Erweiterung dient, in vier Bereiche aufgeteilt werden.

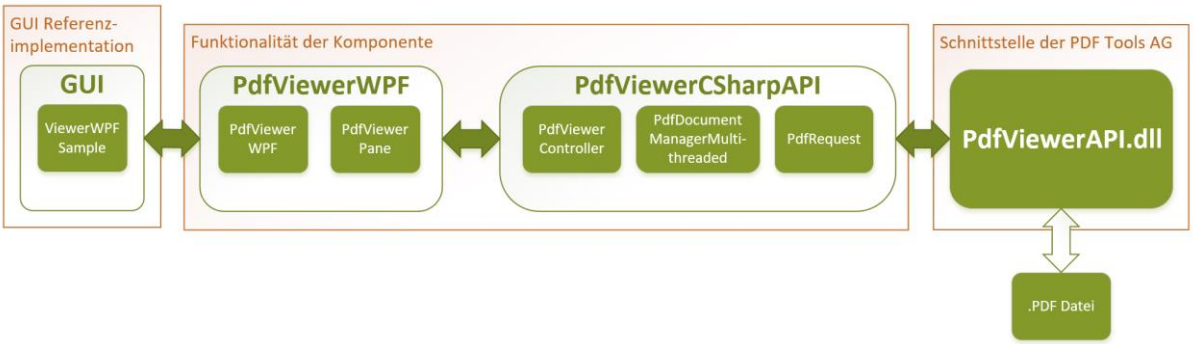


*Abbildung 2: Abgrenzung der Applikationsteile*

### 4.2.1 PdfViewerAPI

Von dieser Bibliothek werden viele Funktionen sowie einige Structs zur Interaktion mit einer PDF-Datei nach aussen zur Verfügung gestellt.

Diese Funktionen umfassen das Öffnen und das Speichern einer PDF-Datei, die Lizenzverwaltung der PDF-Programme unseres Kunden, das Auslesen der Daten in der Datei inklusive deren Darstellung und das Auslesen von Textfragmenten. Ebenfalls wird darin das PDF-Dokument selbst abgebildet und verarbeitet.

Diese API ist das Herzstück mehrerer Applikationen der PDF Tools AG. Sie wird von vielen verschiedenen Applikationen genutzt und stetig weiterentwickelt. Da die API im Zuge dieser Arbeit nur verwendet und nicht verändert werden soll, steht sie den Autoren nur als Bibliothek in Form einer DLL-Datei und nicht als Programmcode zur Verfügung. Dadurch ist der genaue interne Ablauf der Funktionen, welche von den entwickelten Erweiterungen angesprochen werden, nicht bekannt.

### 4.2.2 PdfViewerCSharpAPI

Die Interaktion mit der API wird zentral von einer Managerklasse gesteuert. Die einzelnen Befehle, welche an die API weitergegeben werden, sind als Requests aufgebaut, welche priorisiert von einem separaten Workerthread abgearbeitet werden. Dieses Requestsystem basiert auf dem Design Pattern der Template Methode.

Das Template Method Pattern besteht aus einer abstrakten Basisklasse, welche den Ablauf eines Algorithmus beschreibt. Dieser ist in mehrere Schritte aufgeteilt, die jeweils in einer Methode implementiert werden. Die Kindklassen überschreiben die einzelnen Methoden, welche von der Basisklasse aufgerufen werden. Dieses Pattern garantiert, dass die Teilschritte des Algorithmus in der vorgesehenen Reihenfolge ausgeführt werden und gleichzeitig die Implementation der Teilschritte flexibel gestaltet werden kann.

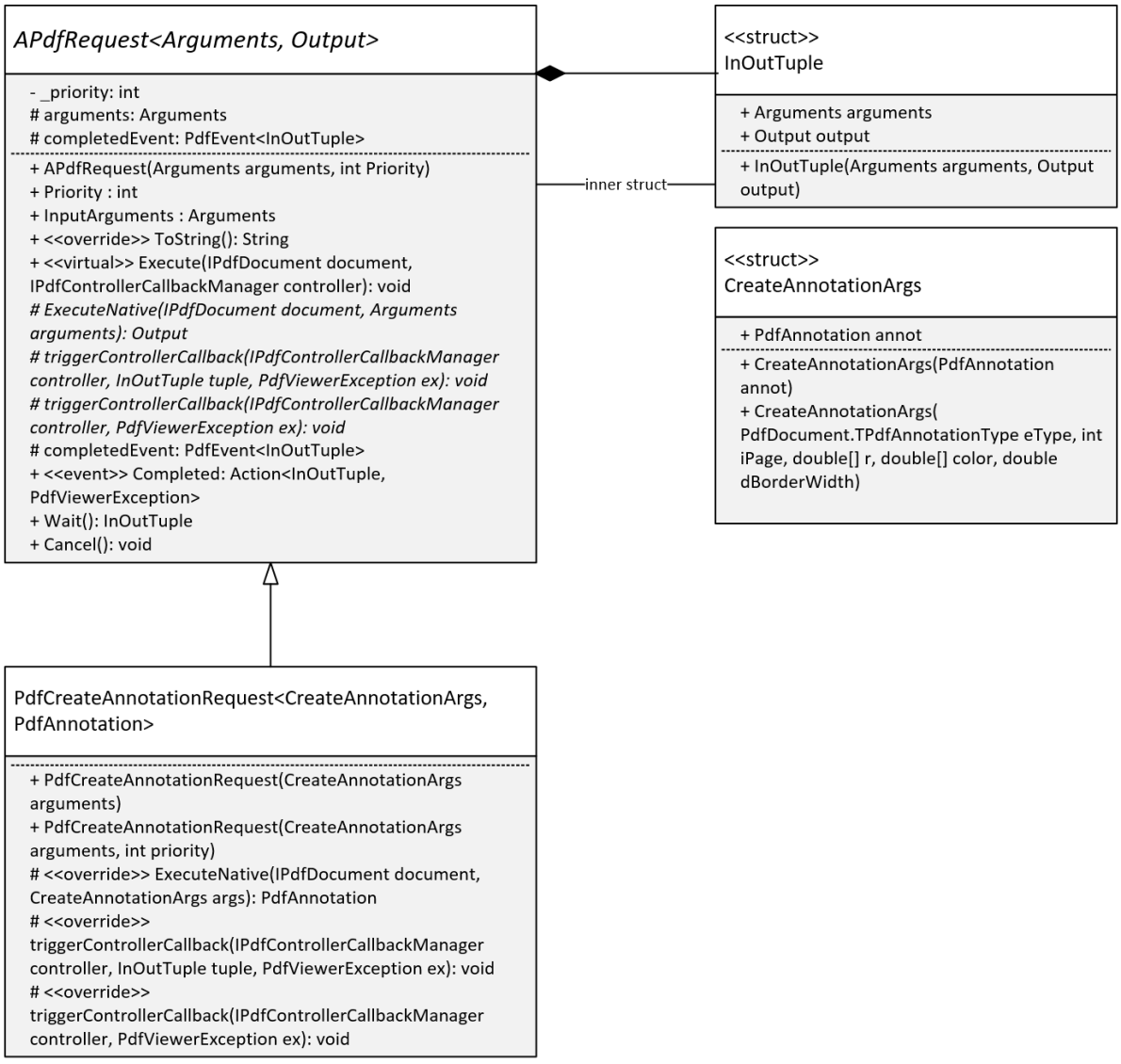


*Abbildung 3: Aufbau der Requestklassen*

In Abbildung 3 ist der Aufbau der Requestklassen anhand des CreateAnnotationRequest ersichtlich.
In der abstrakten Klasse APdfRequest ist die Execute-Methode implementiert. Diese Methode ruft die ExecuteNative, sowie die beiden Callback-Methoden auf, welche durch die Kindklassen überschrieben werden müssen.
Somit ist gewährleistet, dass die Callback-Methoden nach dem Execute ausgeführt werden, was eine einheitliche Abwicklung der Requests sicherstellt.

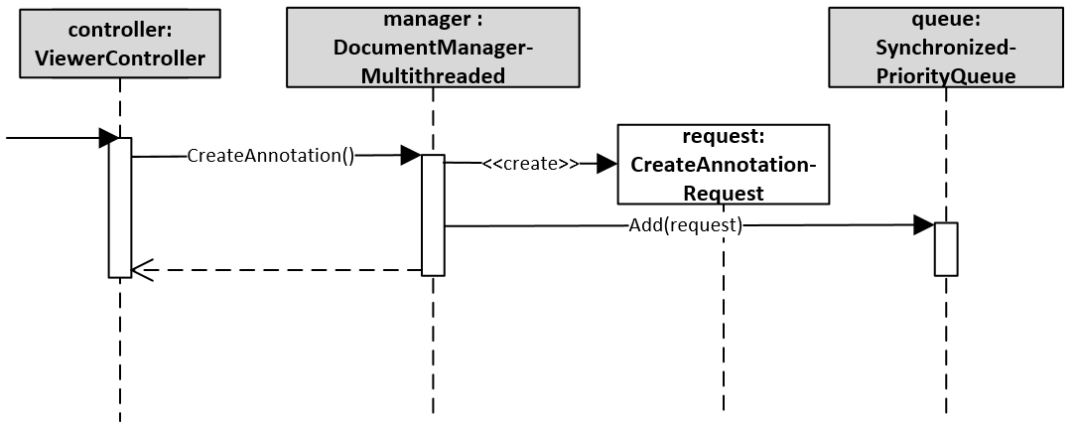


*Abbildung 4: Erstellung eines Requests*

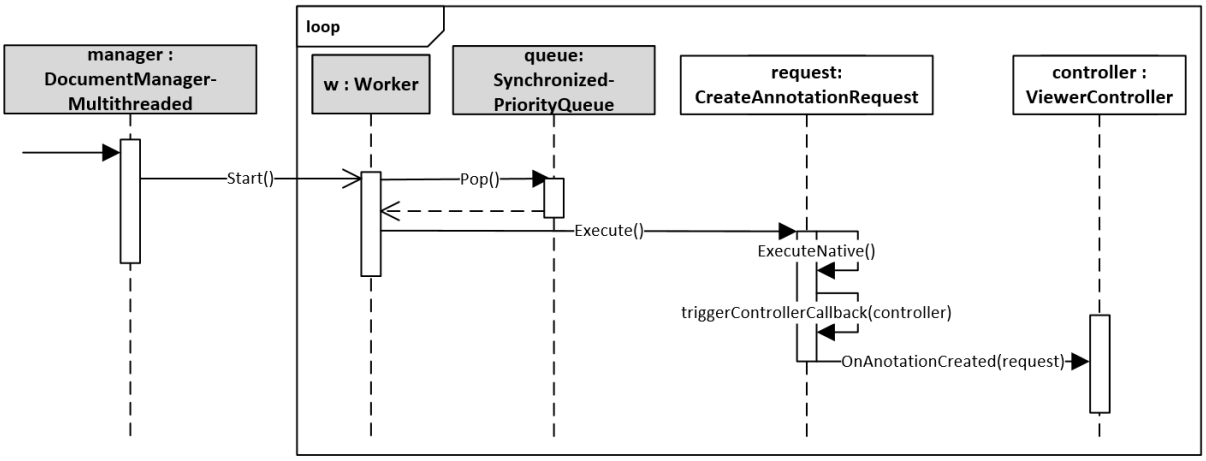


*Abbildung 5: Aufruf eines Requests*

Abbildung 5 4 zeigt das Umfeld des Requestsystems auf. Die Requests werden durch einen Document-Manager in eine synchronisierte und priorisierte Warteschlange abgelegt. Daraus nimmt sich wiederum ein separater Worker-Thread den Request mit der höchsten Priorität und führt diesen aus, was in Abbildung 5 ersichtlich ist.
Die Anfragen, welche zu den Requests führen, stammen aus den entsprechenden Methoden im ViewerController. Diese wiederum werden durch verschiedene Quellen wie den Benutzereingaben oder über eine Aktualisierung der Seite angestossen.

Der Controller organisiert das Erstellen von Requests und bereitet die Daten zur Übergabe an die API vor. Hier wird beispielsweise die Funktionalität der Textsuche, des Scrollens oder des Zooms im Dokument implementiert. Ebenfalls werden viele Callback-Methoden zur Verfügung gestellt, welche von der WPF-Komponente abonniert werden können.
Im PdfDocument werden die nativen Methoden der API referenziert, welche aus dem Controller oder einem Request aufgerufen werden.

*Tabelle 2: Priorisierung der Requests*

| **Request** | **Priorität** |
|---|---|
| Close | 100 |
| Open | 99 |
| Save | 95 |
| GetOpenActionDestination | 88 |
| GetPageLayout | 85 |
| GetPageRect | 80 |
| GetOutlines | 72 |
| CreateAnnotation | 60 |
| DeleteAnnotation | 55 |
| UpdateAnnotation | 50 |
| LoadAnnotationsOnPage | 45 |
| GetTextFragments | 40 |
| Draw | 10 |
| LoadThumbnail | 8 |

In Tabelle 2 werden die vorhandenen Requests dargestellt. Die gelb markierten Einträge sind die Request die im Zuge dieser Arbeit erstellt wurden. Die Einträge, die höher priorisiert sind als die Annotations Request, sind für den regulären Umgang mit der PDF Datei gedacht. Aus diesem Grund sind die interaktiven Requests für die Annotationen gleich anschliessend positioniert worden. Die Prioritäten wurden mit dem Kunden besprochen und von diesem abgesegnet.

#### *4.2.2.1 Koordinatensysteme*

Die Komponente verwaltet drei verschiedene Koordinatensysteme, die im Umgang mit dem PDF-Dokument verwendet werden.

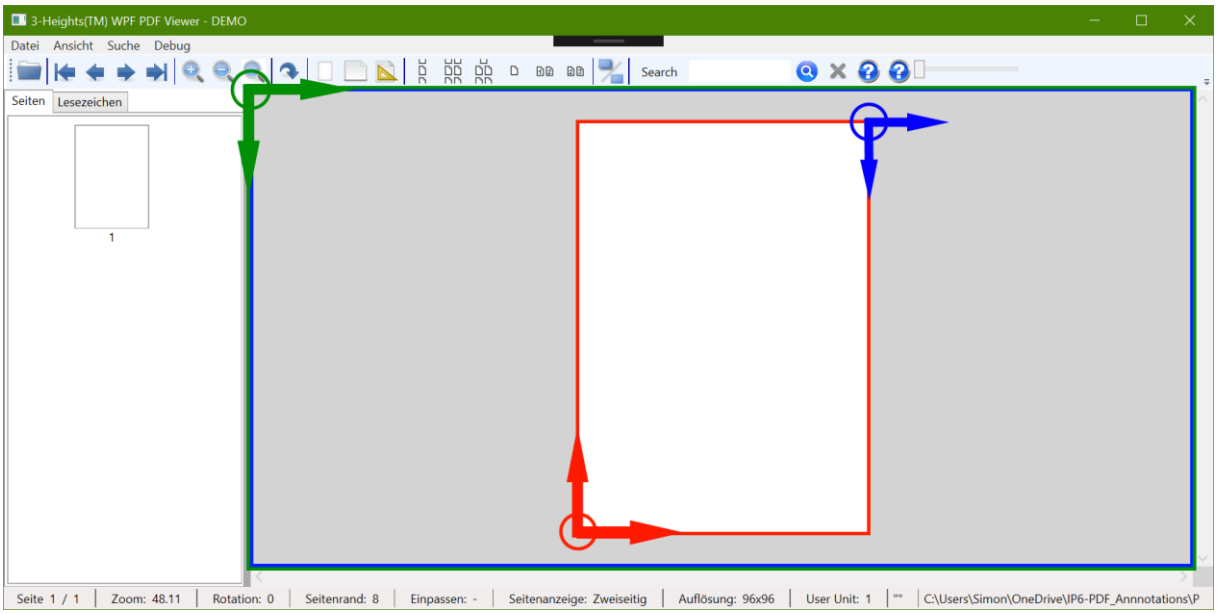


*Abbildung 6: Koordinatensysteme in der Komponente*

In der Abbildung 6 sind die einzelnen Systeme mit Farben gekennzeichnet.

*Tabelle 3: Verschiedene Koordinatensysteme*

| Farbe | Kennzeichnung | Beschreibung |
|---|---|---|
| **Rot** | Seiten-Koordinatensystem | Jede PDF Seite hat ein eigenes Koordinatensystem, welches in der Ecke unten links beginnt und gegen rechts oben wächst. |
| **Blau** | Canvas-Koordinatensystem | Koordinatensystem über das ganze PDF Dokument. Werden mehrere Seiten nebeneinander dargestellt, befindet sich der Ursprung in der Mitte zwischen den ersten beiden Seiten auf gleicher Höhe wie die Oberkante. Werden alle Seiten untereinander aufgelistet, befindet sich der Ursprung in der oberen rechten Ecke. Das Koordinatensystem wächst gegen unten rechts. |
| **Grün** | Bildschirm-Koordinatensystem | Koordinatensystem, welches durch die Komponente aufgezogen wird, die das PDF darstellt. Der Ursprung befindet sich in der oberen linken Ecke und das System wächst gegen rechts unten. |

### 4.2.3 PdfViewerWPF

Dies ist die Komponente, welche von Entwicklern eingebunden werden kann. Die Komponente besteht aus der PdfViewerPane, der OutlinesView und der ThumbnailView.
Der Inhalt der PDF-Datei wird in der PdfViewerPane dargestellt. Ebenfalls werden darin Benutzereingaben wie Mausklicks oder Bewegungen des Scrollrads entgegengenommen und verarbeitet.

In der PdfViewerPane sind mehrere Mausmodi implementiert, welche dazu führen, dass Mausinteraktionen mit der Komponente je nach Modus zu anderen Resultaten führen. Navigiert man beispielsweise mit der Maus durch ein PDF-Dokument, ist es durch den Wechsel des Mausmodus möglich, anstatt zu navigieren, eine bestimmte Textstelle zu markieren. Ebenfalls gibt es weitere Modi für das Zoomen oder die Markierung mittels eines Rechtecks.

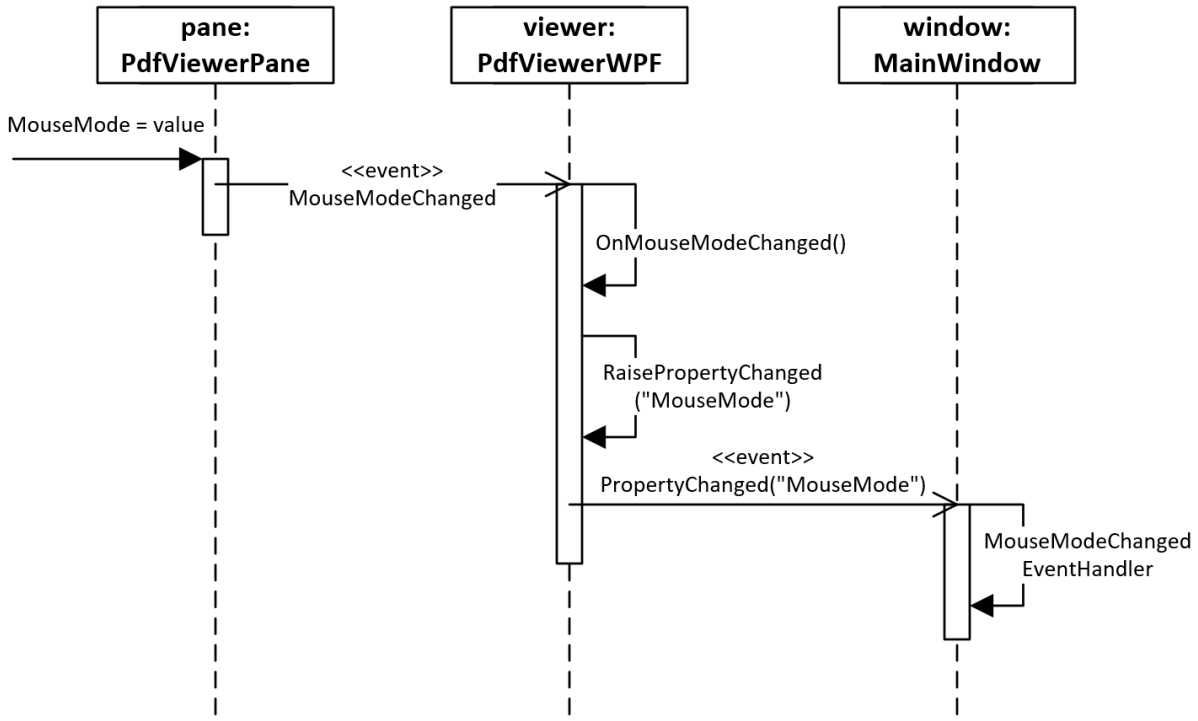


*Abbildung 7: Ablauf beim Wechsel des Mausmodus*

In Abbildung 7 wird am Beispiel des Mausmodus aufgezeigt, wie die Änderung eines Properties der PdfViewerPane an eine User Interface Implementation der Komponente weitergegeben wird. Hier wird das Eventsystem verwendet. In der PdfViewerPane wird bei einer Änderung des Properties das Event MouseModeChanged ausgeführt, welches vom PdfViewerWPF abonniert wurde. Dieser wiederum führt dadurch das Event PropertyChanged aus, welches vom MainWindow abonniert ist.

### 4.2.4 ViewerWPFSample

Dieser Teil der Applikation ist ein User Interface, welches aber nicht als eigenständiger PDF-Viewer gedacht ist, sondern als Referenz für Entwickler, welche die Komponente in ihre eigene Applikation einbinden wollen.

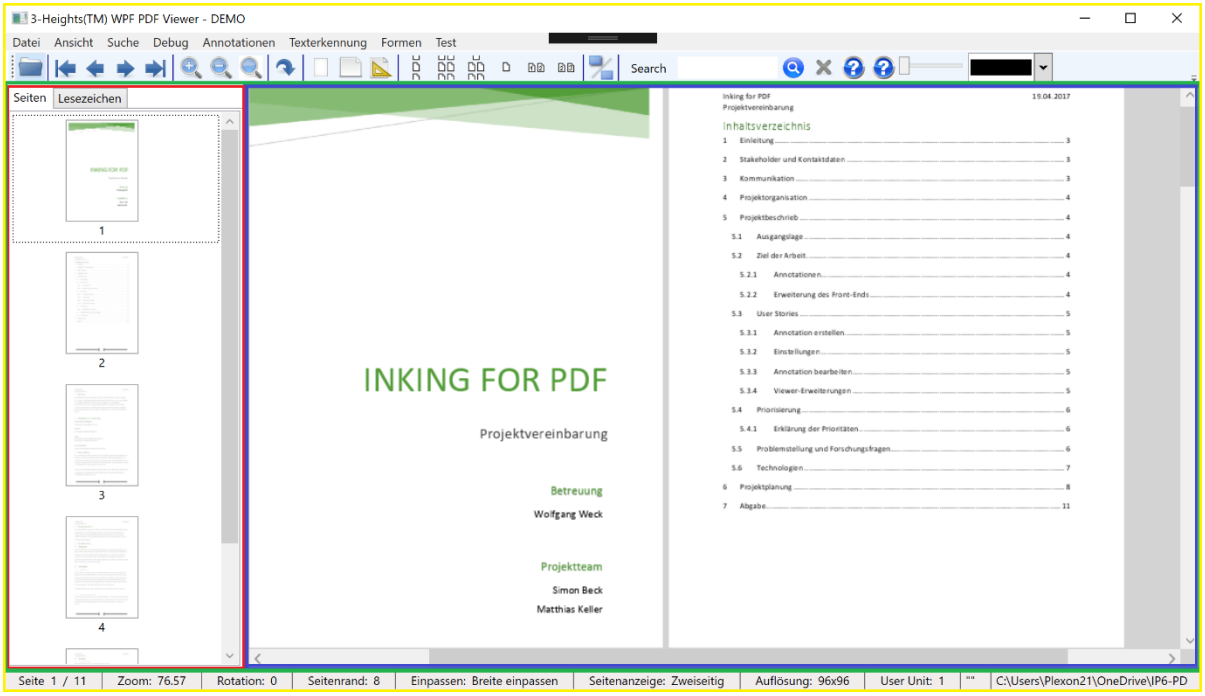


*Abbildung 8: Übersicht der GUI-Teile*

*Tabelle 4: Beschreibung der GUI-Teile*

| Farbe | Kennzeichnung | Beschreibung |
|---|---|---|
| **Rot** | ThumbnailView | Dient zur Darstellung der Übersicht des gesamten Dokuments |
| **Blau** | PdfViewerPane | Stellt das PDF-Dokument dar<br>Enthält die gesamte Annotationsfunktionalität |
| **Grün** | PdfViewerWPF | Gesamte Komponente, welche von Entwicklern in eine Applikation eingebunden werden kann |
| **Gelb** | MainWindow | Referenzimplementation, welche die Komponente verwendet<br>Entspricht dem ViewerWPFSample |

### 4.2.5 Logger

Die Komponente enthält einen statischen Logger, welcher die vier Funktionen LogInfo, LogWarning, LogError und LogException zur Verfügung stellt. Der Pfad des Loggers ist als Property in der Komponente abgelegt. Standardmässig wird eine Datei mit dem Namen log.log verwendet und im BaseDirectory der Applikation abgelegt. Der Logger kann aus der gesamten Komponente verwendet werden.

## 4.3 Herausforderungen

Im Verlauf der Arbeit sind mehrere Herausforderungen in Bezug auf die bestehende Code-Struktur aufgetreten. Diese liessen sich in zwei Kategorien aufteilen.

### 4.3.1 API-Funktionalität

Als Dokumentation der PdfViewerAPI steht den Autoren die Headerdatei der DLL zur Verfügung, welche die Funktionsdefinitionen und Structs beinhaltet. Die internen Abläufe und Abhängigkeiten der API-Funktionalität sind nicht bekannt. Dies führte bei der Verwendung der API zu Unsicherheiten und unerwarteten Fehlfunktionen. Um diese Probleme zu lösen, haben die Autoren auf die Unterstützung unseres Kunden zurückgegriffen. Durch regelmässige Kommunikation sowie Proof-of-Concepts der Verwendung konnten die Unklarheiten beseitigt werden. Ebenfalls lag den Autoren die Java Version des Viewers als Referenz zur Verfügung.
Im Zuge dieser Arbeit wurden mehrfach Änderungen an der API nötig, die die Annotationen betreffen. Die Änderungen wurden auf Anfrage der Autoren vom Kunden überprüft und entsprechend von Ihm umgesetzt.
Zum Zeitpunkt des Erhalts der vierten Version war die Arbeit an der Komponente abgeschlossen, wodurch deren Änderungen nicht mehr in das Produkt einfliessen konnten.

*Tabelle 5: Versionen der API*

| Version | Änderungen | Datum Erhalt |
|---|---|---|
| 1 | Initiale Version | 17.03.2017 |
| 2 | Die API-Aufrufe für das Erstellen und Bearbeiten wurden mit Argumenten erweitert um die Farbe und die Strichbreite festlegen zu können. | 03.05.2017 |
| 3 | Die API-Interne Berechnung der umschliessenden Rechtecke wurde angepasst. | 13.07.2017 |
| 4 | Der Fehler, dass Änderungen der Strichbreite nur von und zu 1.0 möglich waren wurde behoben. Das Struct der Annotationen wurde mit einem Attribut für die Strichbreite erweitert. | 09.08.2017 |

### 4.3.2 Architekturbeschreibung

Der bisherige Aufbau der bestehenden Teile der Komponente ist vor allem im Programmcode festgehalten. Der Aufbau des Requestsystems stand den Autoren als Diagramm zur Verfügung. Dieses Diagramm ist im Anhang auffindbar. Um die verschiedenen Teile und deren Interaktionen zu verstehen, musste der Code manuell analysiert werden.

Um die Funktionalität zu verstehen, wurden nicht einzelne Klassen, sondern spezifische Abläufe analysiert. Dies führte zu einem tieferen Verständnis der Abhängigkeiten der einzelnen Klassen.

Dieses Vorgehen kann am Beispiel des Kontextmenüs erklärt werden. Es soll herausgefunden werden, wie Text von einem Benutzer markiert wird. Dafür werden die Aufrufe ausgehend vom Anzeigen des Kontextmenüs in der Viewerimplementation über die Änderung des Mausmodus bis in die API-Aufrufe verfolgt.

## 4.4 Überprüfung

Die regelmässige und offene Kommunikation mit dem Kunden sowie die Analyse der Schnittstelle durch Proof-of-Concepts versichert den Autoren, dass die Komponente korrekt verstanden und verwendet wird. Wichtige Bestandteile wie das Requestsystem wurden schematisch dargestellt, um einen Überblick über die Abläufe zu erhalten.

# 5 Annotationen

Das PDF-Dateiformat bietet die Möglichkeit, über so genannte Annotationen ein Objekt mit einer Position in einer PDF-Datei zu assoziieren und dem Benutzer dadurch eine Schnittstelle zu bieten, um mit dem PDF zu interagieren. Dies umfasst Objekte wie Notizen, Videos, Zeichnungen und vieles mehr.

Annotationen existieren getrennt vom eigentlichen Inhalt des PDF in einem Array pro Seite. Dabei sind diese, abhängig von der Art der Annotation, mehr oder weniger mit dem Inhalt des PDF verknüpft. Beispiele für Annotationen, welche mit bestimmten Teilen des Textes verlinkt sind wären Highlight- oder Underlineannotationen.
Weitere Informationen zu Annotationen und dem PDF-Format sind in der PDF-Referenz von Adobe auffindbar.

## 5.1 Textannotationen

Der am zahlreichsten verwendete Annotationstyp ist die Textannotation[1], welche meist für Kommentare zu einer Textpassage verwendet wird. Im Gegensatz zu einer Freitextannotation ist bei der Textannotation der Inhalt nicht direkt auf dem PDF ersichtlich, es wird stattdessen ein Symbol angezeigt. Dem Benutzer dient das Symbol als Einstiegspunkt, um mit der Annotation zu interagieren. Eine Textannotation hat jeweils eine Überschrift und einen Inhalt in Textform.
Abbildung 5 zeigt ein Beispiel einer Textannotation, wie sie im Adobe Acrobat Reader dargestellt wird.

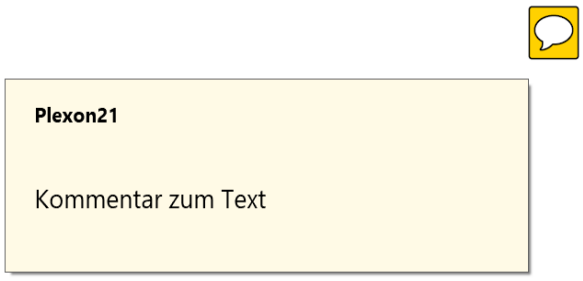


*Abbildung 9: Sticky Annotation in Adobe Acrobat Reader*

[1] Auch «Sticky Note» genannt.

## 5.2 Inkannotationen

Im Fokus dieser Arbeit stehen die Inkannotationen[2]. Diese repräsentieren Linien oder Geraden, die direkt auf dem Dokument angezeigt werden. Die Inkannotation ermöglicht dem Benutzer mithilfe der


*Abbildung 10: gezeichnete Inkannotatioen*

Maus oder eines Stiftes, direkt auf das PDF zu schreiben oder zu zeichnen. Dabei ist jedoch zu beachten, dass eine Inkannotation als eine Liste von Punkten abgespeichert wird. Diese Punkte werden beim Darstellen der Annotation je nach Implementation mit einer geraden Linie oder einer Kurve verbunden.

Die Linienbreite wird für die gesamte Annotation bestimmt, das bedeutet, dass alle Punkt-zu-Punkt-Linien einer Annotation dieselbe Breite haben.
Für die Auswahl der Farbe gibt es drei verschiedene Systeme zur Auswahl.
Welches dieser Systeme zum Tragen kommt, ist abhängig von der Anzahl Zahlen die der API übergeben werden.

*Tabelle 6: Übersicht Farbsysteme*

| **Farbsystem** | **Anzahl übergebene Zahlen** |
|---|---|
| Graustufen | 1 |
| RGB (Rot, Grün, Blau) | 3 |
| CMYK (Cyan, Magenta, Gelb, Schwarz) | 4 |

[2] Auch «Freihandannotation» genannt.

## 5.3 Speicherung im PDF

Die Speicherung von Annotationen im PDF wird durch die PDF-Referenz spezifiziert.
Die Seiten eines PDF sind im «Pagetree» gespeichert. Ein Suchbaum der die Struktur der Seiten einer Datei spezifiziert. Die Blattknoten dieses Baums entsprechen den Seiten des PDF. Jeder Blattknoten enthält eine Liste mit allen Daten dieser Seite, wie zum Beispiel der Inhalt, die Rotation der Seite oder der Name der Seite. Einer dieser Einträge ist das Array der Annotationen.
Eine Annotation wird, wie eine Seite, mittels einer Liste von Attributen dargestellt. Dabei gibt es diverse Einträge, die alle Annotationsarten gemeinsam haben, wie zum Beispiel Subtyp (definiert die Art der Annotation), Rechteck (das umschliessende Rechteck) oder die Farbe.
Zusätzlich hat jeder Annotationstyp eigene Einträge, die für diese Annotationsart notwendig sind.
Für die Inkannotation ist dies zum Beispiel die Inklist, welche die einzelnen Punkte der Annotation beinhaltet.

## 5.4 Annotationsklasse

Es wurde eine Klasse erstellt, welche eine Annotation repräsentiert. Dies dient dazu, dass die Informationen der Annotationen, welche von der API geladen werden, der Applikation zugänglich gemacht werden. Da die Klasse nur eine Kopie der eigentlichen Daten aus der API ist, ist es nicht möglich, über die Klassenattribute die Werte der eigentlichen Annotation in der PDF-Datei selbst zu ändern. Zusätzlich zu den Informationen zur Annotation, enthält die Klasse diverse Methoden, um mit Annotationen zu interagieren.

### 5.4.1 Aufbau

Die Annotationsklasse ist analog zum TPdfAnnotation Struct aufgebaut. Dieses Struct repräsentiert die Darstellung der Annotationen in der API.

Damit klargestellt ist, dass die Klasse nur dazu dient Informationen auszulesen und jegliche Änderungen an den Attributen keine Auswirkungen hätten, sind alle Attribute als Readonly deklariert.

### 5.4.2 Marshalling

Die API liefert beim Laden der Annotationen einen Pointer auf eine Liste von Annotationen sowie deren Anzahl. Um nun die Annotationen und deren Attribute in die Komponente einzulesen, muss jede Annotation mittels Marshalling in ein C# Struct geladen werden.

Ein wichtiger Punkt bei der Konvertierung sind die Datentypen, welche in der API verwendet werden und im Header-File[3] definiert sind. Dabei ist vor allem das Alias BOOL zu beachten, welches einen Wert des Typs int darstellt. Somit ist es wichtig, die Daten eines BOOL auch entsprechend zu behandeln. Zur weiteren internen Verwendung werden die als Struct repräsentierten Annotationen in Objekte der eigenen Klasse umgewandelt.

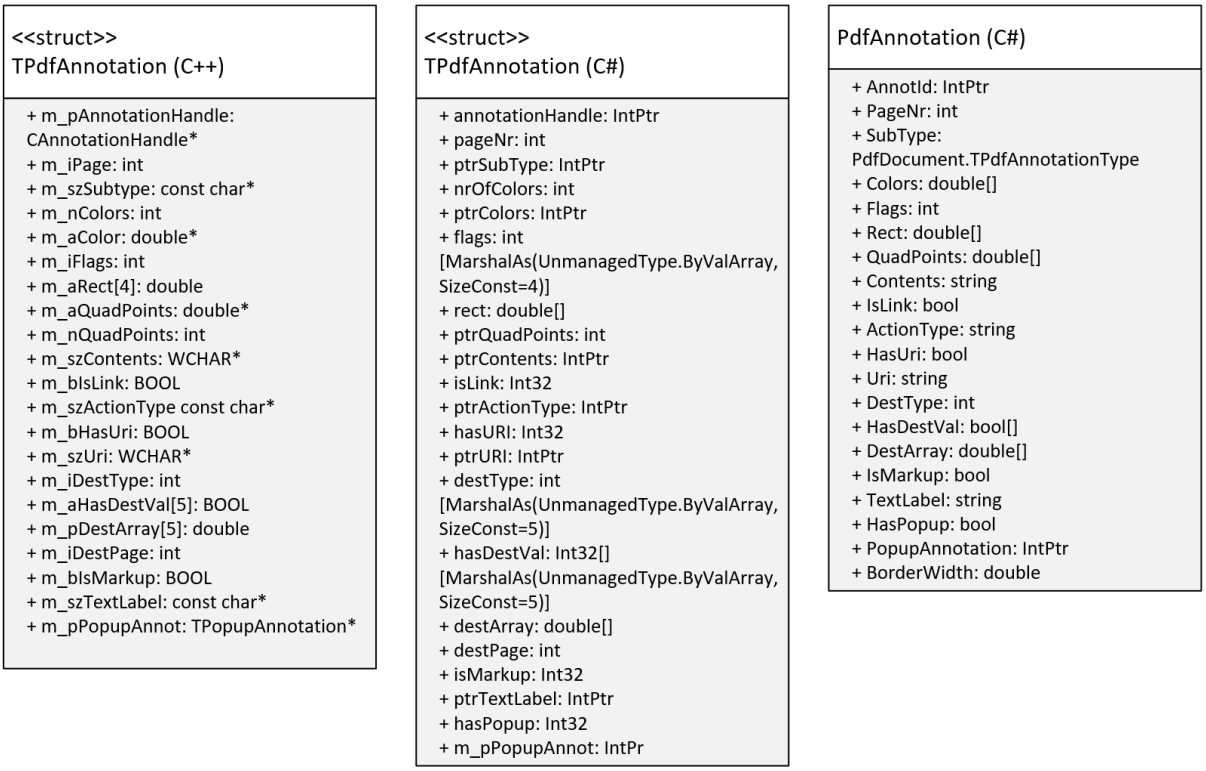


*Abbildung 11: C++ Struct, C# Struct, C# Klasse (v.l.)*

Angestossen durch den Erhalt der vierten Version der API wurde das Marshalling des Structs überprüft. Dabei wurde festgestellt, dass das Marshalling nicht korrekt implementiert ist. Die Werte unterhalb des Contents werden nicht korrekt eingelesen. Da diese Werte für die Funktionalität dieser Arbeit nicht benötigt werden, hat dieses Problem keine Auswirkung auf die Funktionalität der Komponente und wurde nicht eher bemerkt. Die Arbeit an der Komponente war zum Zeitpunkt des Erhalts der neuen Version bereits abgeschlossen. Deshalb wurde dies nicht mehr angepasst.

[3] PdfViewerAPI.h im Anhang

# 6 Annotationen Laden

Mit dem Laden der Annotationen aus der API ist es möglich, innerhalb der Applikation auf die Werte der einzelnen Annotationen zuzugreifen. Dies ist notwendig, da ein bestehendes Dokument bereits diverse Annotationen enthalten kann. Es soll also auch möglich sein, mit den bestehenden Annotationen zu interagieren. Während die API die Darstellung übernimmt, wird jegliche Interaktion von der Komponente gesteuert.

## 6.1 Vision

Das Ziel des Ladens ist es, die momentan aktuellen Annotationen (alle Annotationen auf der Seite, mit welcher der Benutzer zurzeit interagiert) aus der API auszulesen und der Komponente bereitzustellen. Zusätzlich soll die Applikation mit den neu geladenen Annotationen aktualisiert werden.

## 6.2 Ablauf

Das Laden der Annotation lässt sich grob in drei Schritte unterteilen. Zuerst erfolgt der Aufruf zum Laden, anschliessend werden die Annotationen von der API geladen und aufbereitet und zum Schluss wird die Komponente aktualisiert.

### 6.2.1 Integration im Requestsystem

Um stets die Verwendung der aktuellen Annotationsdaten zu garantieren, lädt die Komponente vor jeder Interaktion mit bestehenden Annotationen alle Annotationen auf der Seite neu. Dies ist der Fall, wenn bestehende Annotationen markiert, bearbeitet oder gelöscht werden sollen. Das Erstellen von neuen Annotationen hat keinen Einfluss auf die Bestehenden und erfordert daher kein erneutes Laden.

Basierend auf dem Aufruf wird ein Request zum Laden erstellt. Dieser Request enthält die Nummer der Seite, deren Annotationen geladen werden sollen. Der Request wird der Queue mit der Priorität 45 hinzugefügt. Dies ist die tiefste Priorität aller Requests die Annotationen betreffen und garantiert dadurch, dass alle Erstellungen, Änderungen und Löschungen vor dem Laden durchgeführt werden.

### 6.2.2 API-Aufruf

Wird der Request vom Worker abgearbeitet, werden zuerst die Annotationen von der API mittels eines API-Aufruf geladen.

*Listing 1: PdfViewerGetAnnotationsOnPage*

```
[DllImport("PdfViewerAPI.dll",
        CharSet = System.Runtime.InteropServices.CharSet.Unicode,
        CallingConvention = CallingConvention.StdCall)]
static extern int PdfViewerGetAnnotationsOnPage(
        IntPtr documentHandle,
        int pageNr,
        out IntPtr annotsPointer,
        out int annotsLength);
```

Der Aufruf benötigt vier Argumente:

*Tabelle 7: Argumente der Funktion PdfViewerGetAnnotationsOnPage*

| | |
|---|---|
| `IntPtr documentHandle` | Pointer auf das PDF Dokument |
| `int pageNr` | Nummer der Seite, deren Annotationen geladen werden |
| `out IntPtr annotsPointer` | Pointer, welcher von der API überschrieben wird und auf den Start des Annotations Array zeigt |
| `out int annotsLength` | Anzahl Annotationen auf der Seite |

Mittels Marshalling werden die Annotationen anschliessend aufbereitet.

### 6.2.3 Update

Falls sich unter den geladenen Annotationen eine oder mehrere Annotationen befinden, die markiert werden müssen, werden diese der Liste der markierten Annotationen hinzugefügt. Danach wird der Komponente der Befehl zum Neuzeichnen übergeben.

## 6.3 Eingliederung in die bestehende Architektur

Das Laden ist der einzige API-Aufruf, der Daten zurückliefert, welche von der Komponente weiterverwendet werden. Dies stellte eine Herausforderung dar. Durch den Aufbau des Requestsystems ist es nicht möglich, einen Request abzuschicken und danach auf die Antwort zu warten, ohne den aufrufenden Thread zu blockieren.

### 6.3.1 Lösungsmöglichkeiten

In den nächsten Abschnitten werden die erarbeiteten Lösungsmöglichkeiten vorgestellt.

#### *6.3.1.1 Direktes Laden*

Die einfachste Lösung ist das Laden der Annotationen ausserhalb des Requestsystems zu positionieren. Dies ermöglicht einen direkten API-Aufruf und eine Verarbeitung der Daten, ohne warten zu müssen. Die Vorteile sind, dass es einfach zu benutzen ist, da ein simpler Methodenaufruf reicht. Des Weiteren ist diese Lösung durch den direkten Aufruf sehr schnell.
Durch das Umgehen des Requestsystems beim Kommunizieren mit der API entstehen jedoch auch Nachteile. Ein solcher ist, dass ungewollte Nebenwirkungen eintreten könnten, wenn die API von mehreren Threads gleichzeitig angesprochen wird, da der Aufbau der API den Autoren nicht bekannt ist. Ebenfalls ist es möglich, dass die geladenen Daten nicht aktuell sind, weil sich noch Änderungen in der Requestqueue befinden.

#### *6.3.1.2 Warten auf ein Resultat*

Das Requestsystem bietet an, dass auf das Resultat des Requests gewartet werden kann. Dies hat ebenfalls den Vorteil, dass der Aufruf einfach zu gestalten ist und zudem wird das Requestsystem nicht umgangen. Jedoch wird während der Wartezeit der aufrufende Thread blockiert. Da dieser Thread in den verwendeten Fällen der GUI Thread ist, wird die Anzeige blockiert, was die Benutzung des Programmes behindert.

#### *6.3.1.3 Event auslösen*

Die dritte Lösungsmöglichkeit, ist das Auslösen eines Events im Controller. Dadurch ist es möglich, den Request für das Laden abzuschicken, ohne danach warten zu müssen. Nach dem Laden und Aufbereiten der Annotationen wird der Callback des Requests ausgelöst, worin der angesprochene Event ausgelöst wird. Dieser Aufbau ermöglicht es, dass verschiedene Klassen auf das Laden der Annotationen reagieren können. Der Nachteil ist jedoch, dass der Kontext, aus welchem der Aufruf zum Laden ausgelöst wurde, nicht bekannt ist.

### 6.3.2 Entscheid

Der Entscheid fiel auf das Auslösen eines Events, weil damit die unerwünschten Nachteile der beiden anderen Möglichkeiten umgangen werden konnten. Hinzu kommt, dass der Nachteil der Event-Methode leicht ausgeglichen werden kann, indem mittels einer kurzen Überprüfung des aktuellen Stands der Attribute darauf geschlossen werden kann, was zu aktualisieren ist.

## 6.4 Überprüfung

Um zu testen, ob alle Annotationen bei einem Ladevorgang aus der API zurückgegeben werden, werden zuerst 5000 Annotationen erstellt. Danach werden alle Annotationen markiert, was zum Laden aller Annotationen auf der Seite führt. An dieser Stelle zeigt sich, dass alle Annotationen geladen wurden. Weitere Informationen zur Überprüfung der Ladefunktionalität befindet sich im Testdokument im Anhang.

# 7 Annotationen Erstellen

Neue Annotationen zu erstellen, ist eines der Kernstücke dieser Arbeit. Das Erstellen umfasst nebst dem eigentlichen API-Aufruf auch die temporäre Darstellung der zu erstellenden Annotation. Der Fokus dieser Arbeit liegt auf dem Erstellen neuer Inkannotationen.

## 7.1 Vision

Das Ziel ist es, dem Benutzer die Möglichkeit zu bieten, eine neue Inkannotation hinzuzufügen. Ebenfalls soll er die Farbe, sowie die Strichbreite auswählen können. Zur Unterstützung des Verständnisses des Benutzers wird eine temporäre Approximation der zu zeichnenden Annotation dargestellt.

## 7.2 Auswahl der Farbe und Strichbreite

Die Gestaltung der Farbauswahl und Strichbreite wird der GUI-Implementation überlassen. Die Farbe wird in RGB aufgenommen, da der API diese Werte direkt übergeben werden können. Es ist möglich, der Komponente eine Farbe mit Alpha-Anteil zu übergeben, dieser wird jedoch ignoriert, da die API das Erstellen einer Annotation mit Alpha-Anteil nicht unterstützt.

## 7.3 Aufbau

Das Erstellen einer Annotation findet in drei Teilen statt. Zuerst werden die zu zeichnenden Punkte aufgenommen und zwischenzeitlich dem Benutzer angezeigt. Anschliessend werden die Punkte transformiert und in die ausgewählte Form gebracht. Als letztes findet der API-Aufruf statt, der die Annotation erstellt und dem Dokument hinzufügt.

### 7.3.1 Aufnahme der Punkte

Um eine Annotation zu erstellen, muss zuerst die Applikation informiert werden. Dies geschieht über den Mausmodus. Wie der Mausmodus gewechselt wird, ist der GUI-Implementation überlassen. Im ViewerWPFSample geschieht dies über das Kontextmenü.
Um Punkte aufzunehmen, sind zwei verschiedene Arten möglich:

#### *7.3.1.1 Klicken*

Bei dieser Art wird ein Punkt aufgenommen, wenn die linke Maustaste losgelassen wird. Das Klicken eignet sich für die Erstellung von Polylines und einzelnen geraden Linien.

#### *7.3.1.2 Ziehen*

Das Ziehen entspricht dem Zeichnen von Hand. Dabei werden alle Punkte aufgenommen, während die linke Maustaste gedrückt gehalten wird. Das Ziehen eignet sich für die Erstellung von Freihandlinien und handschriftlichen Notizen.

Abhängig vom Mausmodus werden die einzelnen Punkte entsprechend aufgenommen und danach in einer Liste gespeichert.

### 7.3.2 Temporäre Repräsentation

Wenn eine neue Inkannotation erstellt wird, dann wird dem Benutzer eine Approximation der bisher aufgenommenen Linie dargestellt. Es handelt sich dabei nicht um die exakte Vorschau der finalen Annotation, weil die Punkte, die schlussendlich übergeben werden, innerhalb der API noch mittels einer kubischen Bézierkurve geglättet werden.
Die Strichdicke und Farbe jener Repräsentation entsprechen den aktuell ausgewählten Werten und somit auch der Annotation die zum Schluss erstellt wird.

### 7.3.3 Konvertierung der Punkte

Das Aufnehmen der Punkte wird abhängig vom Mausmodus beendet. Im Zieh-Modus wird die Aufnahme mit dem Loslassen der linken Maustaste beendet. Da beim Klick-Modus mehrmals die linke Maustaste gedrückt werden kann, muss die Aufnahme explizit beendet werden. Im ViewerWPFSample geschieht dies durch das Drücken der Leertaste.
Die Punkte, welche sich im Bildschirm-Koordinatensystem befinden, werden anschliessend in das Seiten-Koordinatensystem transformiert. Zu welcher Seite die Annotation hinzugefügt wird, ist abhängig von der Seite, die am nächsten beim ersten Punkt liegt.
Zum Schluss werden die Punkte mittels der aktuell ausgewählten Form verändert, falls dies nötig ist.

### 7.3.4 Integration im Requestsystem

Das Erstellen einer Annotation hat mit Priorität 60 die höchste Priorität aller Requests, welche Annotationen betreffen. Dies garantiert, dass alle Manipulationen von Annotationen (Bearbeiten, Löschen, Laden) nach dem Erstellen geschehen.

### 7.3.5 API-Aufruf

*Listing 2: PdfViewerCreateAnnotation*

```
[DllImport("PdfViewerAPI.dll",
       CharSet = System.Runtime.InteropServices.CharSet.Unicode,
       CallingConvention = CallingConvention.StdCall)]
static extern IntPtr PdfViewerCreateAnnotation(
       IntPtr documentHandle,
       TPdfAnnotationType annotType,
       int pageNr,
       double[] annotPoints,
       int annotPointsLength,
       double[] color,
       int colorLength,
       double borderWidth);
```

Der Aufruf benötigt acht Argumente:

*Tabelle 8: Argumente der Funktion PdfViewerCreateAnnotation*

| | |
|---|---|
| `IntPtr documentHandle` | Pointer auf das PDF-Dokument |
| `TPdfAnnotationType annotType` | Typ der Annotation |
| `int pageNr` | Nummer der Seite, auf der die Annotation erstellt wird |
| `double[] annotPoints` | Array der Punkte im Seiten-Koordinatensystem, abwechslungsweise X- und Y-Koordinaten (x1,y1,x2,y2,x3,y3) |
| `int annotPointsLength` | Die Länge des annotPoints Array |
| `double[] color` | Die Farbe der Annotation. Die Länge des Arrays bestimmt das Farbsystem (2 für Graustufen, 3 für RGB,4 für CMYK). Die Werte sind im Bereich [0 - 1] zu übergeben. |
| `int colorLength` | Die Länge des color Array |
| `double borderWidth` | Die Breite des Striches. Wird 0 übergeben entspricht die Breite immer einem Pixel, unabhängig des Zoomlevels |

## 7.4 Microsoft Digital Ink

Als Möglichkeit die Punkte für eine Inkannotation aufzunehmen und eine temporäre Repräsentation zu erstellen, wurde überprüft, ob Microsoft Digital Ink verwendet werden kann. Diese Bibliothek bietet eine Vielzahl von Funktionen zum Erstellen von Notizen in einer WPF-Komponente an. Digital Ink ist darauf ausgelegt, dass freie Notizen oder Zeichnungen innerhalb einer WPF-Komponente erstellt werden können. Auf die Verwendung dieser Bibliothek ist für die Erstellung von Annotationen aus folgenden Gründen verzichtet worden.

### 7.4.1 Einschränkung

Digital Ink müsste eingeschränkt werden, da es Funktionen anbietet, die nicht als Annotation in einem PDF gespeichert werden können. Zum Beispiel wird die Stärke des Stiftdrucks berücksichtigt bei der Erstellung einer Linie. Dadurch ist es möglich eine Linie zu erstellen, deren Dicke sich verändert.

### 7.4.2 API-Funktionalität

Das Darstellen von bestehenden Annotationen übernimmt die API. Damit eine Annotation mithilfe von Digital Ink bearbeitet werden kann, muss sie zuerst geladen und mit den Punkten eine entsprechende StrokeCollection erstellt werden. Eine StrokeCollection ist eine Sammlung mehrerer Linien die von der Digital Ink Bibliothek verwendet wird.

Dadurch, dass das Laden dieser Punkte zum aktuellen Stand der API nicht unterstützt wird, ist es nicht möglich die StrokeCollections für die Bibliothek wiederherzustellen. Dadurch können die Funktionen von Digital Ink für die Bearbeitung von Inkannotationen nicht verwendet werden. Für die Aufnahme der Punkte ist es möglich, Digital Ink zu verwenden. Damit die Repräsentation der zu erstellenden Inkannotation entspricht, muss die Verwendung der Bibliothek jedoch entsprechend eingeschränkt werden, wodurch der Mehrwert von Digital Ink wegfällt.

## 7.5 Überprüfung

Die Applikation läuft auch beim programmatischen gleichzeitigen Erstellen vieler Annotationen flüssig. 5000 Annotationen, welche jeweils aus zwei Punkten bestehen werden nach knapp zwei Sekunden erstellt und dargestellt. Die Applikation konnte auch mit 500'000 Annotationen nicht zum Absturz gebracht werden. Um die Komponente in diesem Zustand noch zu benutzen, muss man sich aber bei Scroll- oder Zoomoperationen mit Antwortzeiten von mehreren Sekunden abfinden.

Eine einzelne gezeichnete Annotation kann bis zu 550 Punkte enthalten, bevor merkbare Verzögerungen zwischen der Mausbewegung und der Zeichnung auf dem Bildschirm auftreten.

Weitere Überprüfungen zum Erstellen von Annotationen sind im Anhang im Testdokument zu finden. Diese Grenzwerte sind für Anwendungsfall nicht sonderlich relevant, da bei handschriftlichen Annotationen nur einzelne Annotationen erstellt werden. Relevant ist das gleichzeitige Erstellen von Annotationen bei Formen, welche aus mehreren Annotationen bestehen können.

# 8 Annotationen Markieren

Das Bearbeiten von Annotationen ist gezielt auf einzelne Exemplare anzuwenden. Dies erfordert, dass Annotationen ausgewählt und markiert werden können, um dem Benutzer dies zu symbolisieren. Bis auf das Erstellen erfordert jegliche Interaktion eine vorhergegangene Markierung.

## 8.1 Vision

Das Ziel des Markierens ist es, mehrere bestehende Annotationen auf einer Seite zu markieren. Dies beinhaltet das Zwischenspeichern der betroffenen Annotationen sowie die visuelle Kennzeichnung.

## 8.2 Ablauf

Das Markieren von Annotationen ist in drei Teile unterteilt. Zuerst wird das Markierungsrechteck aufgezogen, anschliessend werden die Annotationen neu geladen und schliesslich werden die geladenen Annotationen, welche sich innerhalb des Markierungsrechtecks befinden markiert.

### 8.2.1 Wechseln des Mausmodus

Die Absicht etwas zu Markieren wird durch das Wechseln des Mausmodus in den Markiermodus gezeigt. Der Markiermodus war bereits Teil der Ausgangslage und wurde mit der Funktionalität erweitert um Annotationen zu markieren.

### 8.2.2 Markieren

Mit dem Drücken der linken Maustaste wird das Markieren gestartet. Während die Taste gedrückt gehalten wird, wird das Markierungsrechteck aufgezogen und dem Benutzer dargestellt. Wird die Maustaste losgelassen, wird das Rechteck in das Canvas-Koordinatensystem transformiert und anschliessend zwischengespeichert. Auf welcher Seite des Dokuments die Annotationen markiert werden, ist abhängig davon, welche Seite sich am nächsten zum Mittelpunkt des Markierungsrechtecks befindet.

Danach wird ein Request zum Laden der Annotationen abgeschickt.

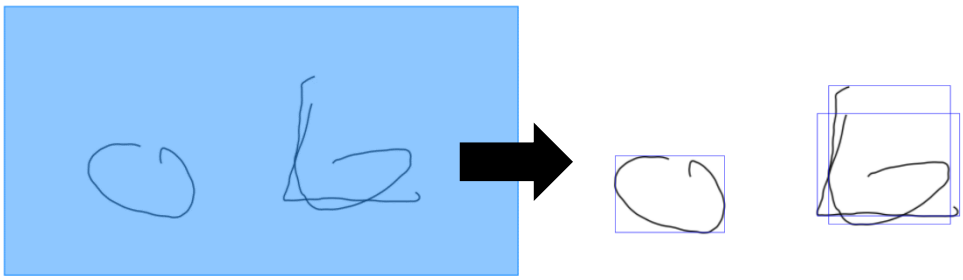

*Abbildung 12: Markierungsrechteck mit markierten Annotationen*

### 8.2.3 Laden und Update

Die umschliessenden Rechtecke der neu geladenen Annotationen werden anschliessend mit dem Markierungsrechteck verglichen. Es sind zwei verschiedene Möglichkeiten implementiert um zu bestimmen, ob eine Annotation markiert wird oder nicht.

#### *8.2.3.1 Umschliessung*

Eine Annotation wird nur markiert, wenn ihr umschliessendes Rechteck komplett vom Markierungsrechteck umschlossen wird.

#### *8.2.3.2 Überschneidung*

Eine Annotation wird markiert, sobald ihr umschliessendes Rechteck sich mit dem Markierungsrechteck schneidet.

Die Auswahl, welche Methode angewendet werden soll, ist der GUI-Implementierung überlassen. Im ViewerWPFSample ist standardmässig die Umschliessungsmethode ausgewählt, sie kann jedoch im Annotationen-Tab umgestellt werden.

Die markierten Annotationen werden dann in einer Liste gespeichert. Um zu visualisieren, dass die Annotationen markiert sind, werden die umschliessenden Rechtecke der Annotationen dargestellt. Der Adobe Acrobat Reader diente als Referenz für die Farbwahl. Auf Wunsch des Kunden wird ist die Farbe beim Markieren auf blau gesetzt.

Wenn der Mausmodus gewechselt oder ein neues Markierungsrechteck aufgezogen wird, dann wird die Liste der aktuell markierten Annotationen geleert.

Bis und mit Version vier der API werden für die Berechnung der umschliessenden Rechtecke in der API nur die Punkte der Annotation beachtet. Durch die Krümmung der Linien durch die kubischen Bézierkurven ist es möglich, dass Teile ausserhalb des umschliessenden Rechtecks liegen. Dieser Fehler ist dem Kunden kommuniziert worden.

## 8.3 Überprüfung

Es können keine Annotationen auf mehreren Seiten gleichzeitig markiert werden. Annotationen, welche sich ausserhalb einer Seite befinden, können markiert werden, wenn Sie vom Markierungsrechteck umschlossen werden. Es können bis zu 1500 Annotationen, welche aus zwei Punkten bestehen, gleichzeitig markiert werden, ohne dass grosse Verzögerungen auftreten. Diese Zahl stimmt auch für Annotationen mit mehr als zwei Punkten, da bei einer Markierung nur das umschliessende Rechteck der Annotation beachtet wird.

Weitere Überprüfungen zur Markierung sind im Anhang im Testdokument auffindbar.

# 9 Annotationen Bearbeiten

Das Bearbeiten ist die Änderung der Attribute von Annotationen. Das Löschen und Erstellen sind keine Bearbeitungen, da die Attribute nicht verändert, sondern gelöscht oder hinzugefügt werden.

## 9.1 Vision

Das Ziel des Bearbeitens ist es, dem Benutzer eine Interaktionsmöglichkeit zu bieten, um die bereits vorhandenen Annotationen zu verändern. Dabei sollen die Änderungen sowohl im GUI als auch in der API zum Tragen kommen.

## 9.2 Arten

Das Bearbeiten umfasst mehrere verschiedene Veränderungen. Dabei wird jeweils mindestens ein Attribut der Annotation verändert. Eine Änderung eines Attributes wirkt sich auf alle momentan ausgewählten und alle zukünftig erstellten Annotationen aus. Dabei wird nicht für jede Annotation ein Request, sondern für alle ausgewählten Annotationen ein Request erstellt. Beim Abarbeiten des Requests wird für jede Annotation ein eigener API-Aufruf durchgeführt.

### 9.2.1 Farbe wechseln

Die Farbe einer Annotation kann zu jeder Farbe im RGB-Raum (ohne Alpha-Anteil) gewechselt werden. Die Auswahl der Farbe ist der GUI-Implementierung überlassen. Im ViewerWPFSample ist eine Komponente zur Farbauswahl eingebaut.

### 9.2.2 Strichbreite wechseln

Die Strichbreite einer Annotation kann theoretisch auf einen beliebigen double-Wert gestellt werden. Die Obergrenze ist auf Kundenwunsch jedoch auf zwölf beschränkt. Negative Zahlen werden ignoriert, da eine negative Strichbreite keinen Sinn ergibt. Die Breite 0 hat eine Sonderfunktion. Dabei wird die Strichbreite unabhängig vom Zoomlevel immer so angepasst, dass sie ein Pixel breit ist. Wie bei der Farbe, ist auch die Auswahl der Strichbreite der GUI-Implementation überlassen. Im ViewerWPFSample ist ein Slider eingebaut, über den man die Strichbreite verändern kann.
In Version drei der API ist die Strichbreite nur von und zu einem Wert von 1.0 veränderbar. Dieser Fehler ist in der darauffolgenden vierten Version behoben worden. Zum Zeitpunkt des Erhalts der vierten Version der API, waren die Arbeiten an der Komponente bereits abgeschlossen, weshalb diese Änderung nicht mehr eingebaut wurde.

### 9.2.3 Verschieben

Das Verschieben von Annotationen bezeichnet die Änderung der Koordinaten der Annotation auf derselben Seite. Da der API-Aufruf die Seite einer Annotation nicht ändern kann, ist es nicht möglich die Annotation auf eine andere Seite zu schieben.

Befindet sich die Maus im Markiermodus und ist innerhalb des umschliessenden Rechtecks einer der markierten Annotationen, ist es möglich jene zu verschieben. Dies wird durch einen Wechsel des Mauszeigers dargestellt. Durch Drücken und Halten der linken Maustaste wird der Verschiebevorgang gestartet. Dabei werden die umschliessenden Rechtecke aller markierten Annotationen an der Stelle, an die sie verschoben werden sollen, mit roter Farbe gezeichnet. Die Farbe Rot wurde gewählt, um einen Kontrast mit der blauen Einfärbung der markierten Annotationen zu bilden.

Die Wahl, die umschliessenden Rechtecke beim Verschieben darzustellen, basiert darauf, dass dem Benutzer eine visuelle Hilfe geboten werden kann. Idealerweise wird direkt die Form der kompletten Annotation und nicht nur das umschliessende Rechteck dargestellt. Bis und mit Version vier der API ist es jedoch nicht möglich, die Punkte einer bestehenden Inkannotation auszulesen, womit diese Variante nicht möglich ist.

Sobald der Benutzer die Maustaste loslässt, wird ein Vektor zwischen dem Startpunkt und dem Endpunkt gebildet. Anschliessend wird ein Request erstellt, der alle markierten Annotationen um diesen Vektor verschiebt.

## 9.3 Erweiterung Skalieren

Das Skalieren konnte aus zeitlichen Gründen nicht implementiert werden. Es umfasst die Änderung der Grösse einer Annotation sowie das Verzerren des Verhältnisses. Folgende Punkte sollten bei der Implementation des Skalierens beachtet werden.

Die API übernimmt das Skalieren. Die Update Methode unterstützt das Ändern des umschliessenden Rechtecks. Wird dieses angepasst, werden die einzelnen Punkte der Annotation von der API verhältnismässig verschoben.

Die PDFViewerPane muss erweitert werden, so dass dem Benutzer angezeigt wird, wann und wie eine Annotation skaliert werden kann.

## 9.4 API-Aufruf

*Listing 3: PdfViewerUpdateAnnotation*

```
[DllImport("PdfViewerAPI.dll",
       CharSet = System.Runtime.InteropServices.CharSet.Unicode,
       CallingConvention = CallingConvention.StdCall)]
static extern int PdfViewerUpdateAnnotation(
       IntPtr documentHandle,
       IntPtr annotId,
       int pageNr,
       double[] boundingBox,
       string content,
       string label,
       double[] color,
       int colorLength,
       double borderWidth);
```

Der Aufruf benötigt neun Argumente:

*Tabelle 9: Argumente der Funktion PdfViewerUpdateAnnotation*

| | |
|---|---|
| `IntPtr documentHandle` | Pointer auf das PDF Dokument |
| `IntPtr annotId` | Id der Annotation |
| `int pageNr` | Nummer der Seite, auf der die Annotation erstellt wird |
| `double[] boundingBox` | Array der Punkte des neuen umschliessenden Rechtecks |
| `string content` | Notizen oder Informationen zur Annotation |
| `string label` | Überschrift der Annotation |
| `double[] color` | Die Farbe der Annotation<br>Die Länge des Arrays bestimmt das Farbsystem (Graustufen, RGB, CMYK)<br>Die Werte sind im Bereich [0 - 1] zu übergeben |
| `int colorLength` | Die Länge des color Array |
| `double borderWidth` | Die Breite des Striches<br>Wird 0 übergeben entspricht die Breite immer einem Pixel, unabhängig des Zoomlevels |

Soll bei einem API-Aufruf ein Attribut nicht geändert werden, ist als Wert «null» oder für das Argument borderWidth «-1» zu übergeben.

## 9.5 Überprüfung

Die Performance der gleichzeitigen Bearbeitung mehrerer Annotationen hängt von der Anzahl der Annotationen, sowie der Anzahl der Punkte der einzelnen Annotationen ab. 500 Annotationen, welche aus 2 Punkten bestehen können ohne Verzögerung bearbeitet werden. Wird dieselbe Anzahl an Annotationen mit mehr Punkten gezeichnet, sind Verzögerungen erkennbar. Handgeschriebene Annotationen, z.B. ein Satz mit 20 Wörtern, können ohne Verzögerung gleichzeitig bearbeitet werden.

Die Autoren nehmen an, dass die bessere Performanz von handgeschriebenen Annotationen damit zusammenhängt, dass die Punkte der generierten Annotationen weiter auseinanderliegen, was zu mehr Aufwand bei der Berechnung der Kurven und beim Rendering der Linien führt.
Weitere Überprüfungen der Bearbeitungsfunktionalität sind im Testdokument im Anhang auffindbar.

# 10 Annotationen Löschen

Das Gegenstück zum Erstellen von Annotationen bildet das Löschen. Dabei wird eine bestehende Annotation aus dem Dokument entfernt.

## 10.1 Vision

Das Löschen hat das Ziel eine oder mehrere bestehende Annotationen, die markiert sind, aus dem Dokument zu entfernen. Danach soll das GUI entsprechend aktualisiert werden.

## 10.2 Löschen

Wenn der Befehl zum Löschen gegeben und sind eine oder mehrere Annotationen markiert wird, dann wird ein Request abgeschickt, der diese Annotationen aus dem Dokument entfernt. Es wird durch die Implementation des GUI definiert, wie der Befehl zum Löschen ausgelöst wird. Im ViewerWPFSample ist es möglich mithilfe der Delete-Taste oder durch einen Kontextmenüeintrag den Vorgang auszulösen.

## 10.3 API-Aufruf

*Listing 4: PdfViewerDeleteAnnotation*

```
[DllImport("PdfViewerAPI.dll",
       CharSet = System.Runtime.InteropServices.CharSet.Unicode,
       CallingConvention = CallingConvention.StdCall)]
static extern void PdfViewerDeleteAnnotation(IntPtr annotId);
```

Der Aufruf benötigt ein Argument:

*Tabelle 10: Argument der Funktion PdfViewerDeleteAnnotation*

| | |
|---|---|
| `IntPtr annotId` | Id der Annotation |

## 10.4 Erweiterung Radieren

Die Erweiterung, dass einzelne Teile einer Annotation gelöscht werden können, konnte im Zuge dieser Arbeit nicht umgesetzt werden. Damit hätten, wie mit einem Radiergummi, einzelne Punkte oder Teile aus einer Annotation gelöscht werden können.

#### *10.4.1.1 Liniensystem*

Eine Inkannotation besteht aus einer Liste von Punkten. Wird die Annotation dargestellt, wird von einem Punkt aus der Liste zum nächsten eine Linie in der definierten Strichdicke gezogen. Dadurch ist ein freies Radieren, wie in einem Zeichnungsprogramm nicht möglich. Die einzelnen Punkte können beliebig weit auseinander stehen. Wird beim Radieren ein Punkt getroffen, kann dieser entfernt und die Annotation an dieser Stelle in zwei neue aufgeteilt werden. Der Fall, in dem kein Punkt mit dem Radierer getroffen wird, ist jedoch komplexer.

Eine Möglichkeit ist, diesen Fall zu ignorieren und nur das Radieren von Punkten zu erlauben. Dadurch können mit dieser Methode nur ganze Abschnitte aus der Annotation gelöscht werden.

Diese Variante ist im Adobe Acrobat Reader implementiert.

Eine weitere Möglichkeit ist es, die Schnittpunkte des Radierbereichs mit der Linie zu berechnen und diese mit den bestehenden Punkten der Linie zu verbinden. Dabei ist zu beachten, dass die Linien zwischen den Punkten je nach Implementation gekrümmt sein können. Diese Variante ist jedoch mit Zeitaufwand verbunden, da für jede Linie einer Annotation, deren umschliessendes Rechteck den Radierbereich tangiert, geprüft werden muss, ob sie den Bereich schneidet und entsprechend betroffen ist. Mit dieser Methode können kleinere Abschnitte aus der Linie entfernt werden, da zusätzliche Punkte erstellt werden.

#### *10.4.1.2 Inklist nicht vorhanden*

Die einzelnen Punkte einer Inkannotation sind in der Inklist gespeichert, dies ist ein Array von double Werten, welche die Koordinaten der einzelnen Punkte repräsentieren. Bis und mit Version vier der API ist es nicht möglich die Inklist einer bestehenden Annotation auszulesen. Ohne diese Liste ist es nicht möglich zu berechnen welche Punkte entfernt werden müssen, wodurch das Radieren für bestehende Annotationen nicht umgesetzt werden kann. Es ist möglich die Komponente so zu gestalten, dass diejenigen Annotationen, welche innerhalb der aktuellen Session selbst erstellt werden, radiert werden können. Dafür müssten diese Annotationen in einer zusätzlichen Liste gespeichert werden. Ebenfalls müsste beim Radieren jeweils überprüft werden, ob es sich um eine Annotation aus dieser Liste handelt. Für den Benutzer könnte diese Version jedoch zu Verwirrung führen, da nicht alle Annotationen bearbeitet werden können.

Da durch den Aufbau kein eigentliches Radieren möglich ist und die Inklist nicht geladen werden kann, wurde in Absprache mit dem Kunden entschieden, das Radieren nicht zu implementieren.

## 10.5 Überprüfung

Beim gleichzeitigen Löschen von 5000 Annotationen welche aus 2 Punkten bestehen, ist keine Verzögerung ersichtlich. Bestehen diese Annotationen aus 10 Punkten, treten hier Verzögerungen von bis zu mehreren Sekunden auf.

# 11 Formen

Im Kontext dieser Arbeit wurden Formen implementiert. Formen wandeln die Punkte einer gezeichneten Annotation in eine bestimmte Anordnung um. Dabei werden zum Beispiel der erste und letzte Punkt einer Annotation verwendet um ein Rechteck zu erstellen.

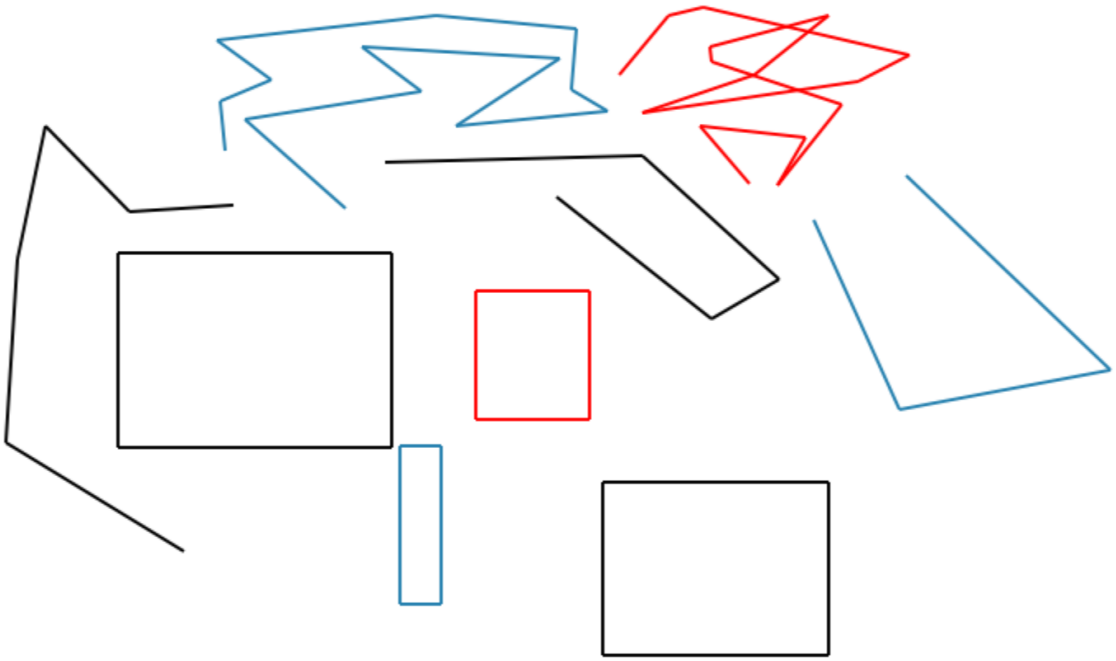

*Abbildung 13: Verschiedene gezeichnete Inkannotationen (Formen)*

## 11.1 Vision

Die Komponente soll die Möglichkeit bieten, nicht nur beliebige Striche, sondern auch spezifische Formen zu zeichnen. In dieser Arbeit wurden die Formen Rechteck und Polyline implementiert. Da die Komponente schlussendlich von Entwicklern in Applikationen eingebunden wird, möchten die Autoren diesen Entwicklern die Möglichkeit bieten, neue Formen zu implementieren, ohne den Quellcode der Komponente anpassen zu müssen. Diese Idee basiert auf dem Dependency Inversion Principle, welches Teil der bewährten SOLID-Softwaredesignprinzipien ist. Um die Funktionalität der Erweiterungen zu demonstrieren, wurden die beiden Formen bereits als Erweiterung implementiert.

## 11.2 Möglichkeiten der Erweiterungsfunktionalität

Es bestehen mehrere Möglichkeiten zur Erweiterung der Komponente durch verschiedene Formen. Eine Möglichkeit ist es, einen Erweiterungspunkt, zum Beispiel ein Interface oder eine abstrakte Klasse, zur Verfügung zu stellen. Dieser soll von Erweiterungen implementiert werden können. Da die Komponente aber selber keine Möglichkeit hat, neue Erweiterungen aufzufinden, muss dieser mitgeteilt werden, beispielsweise durch eine Konfigurationsdatei, welche Erweiterungen zur Verfügung stehen.

Einen Schritt weiter zieht diese Idee das Managed Extensibility Framework (MEF), welches seit der Version 4 im .NET-Framework enthalten ist. Dieses Framework bietet die Möglichkeit, Klassen, welche bestimmte Funktionalitäten exportieren, aus einer externen Bibliothek einzulesen.
Des Weiteren ist es möglich, Metadaten zu den einzelnen Klassen und Funktionen zu definieren, welche der Komponente zur Identifikation der Erweiterungen dienen.

Da die Autoren die Erweiterungen ohne Änderungen an der Komponente implementieren möchten, wurde die Entscheidung für die Implementation der Erweiterungen mithilfe des MEF getroffen.

## 11.3 Umsetzung der Erweiterungsfunktionalität

Um Formen anstelle von Strichen zu zeichnen, müssen die aus Benutzerinteraktionen stammenden Punkte so verarbeitet werden, dass aus den Punkten die gewünschte Form erstellt wird. Eine Form soll also eine Liste von Punkten entgegennehmen, diese verarbeiten und daraus eine Annotation in gewünschter Form erstellen. Es soll jedoch auch möglich sein, bereits während dem Zeichnungsvorgang und nicht erst nach Erhalt aller Punkte dem Benutzer die gewünschte Form anzuzeigen.

Aus diesen Überlegungen ergibt sich folgender Aufbau für die Erweiterungen und deren Schnittstelle:

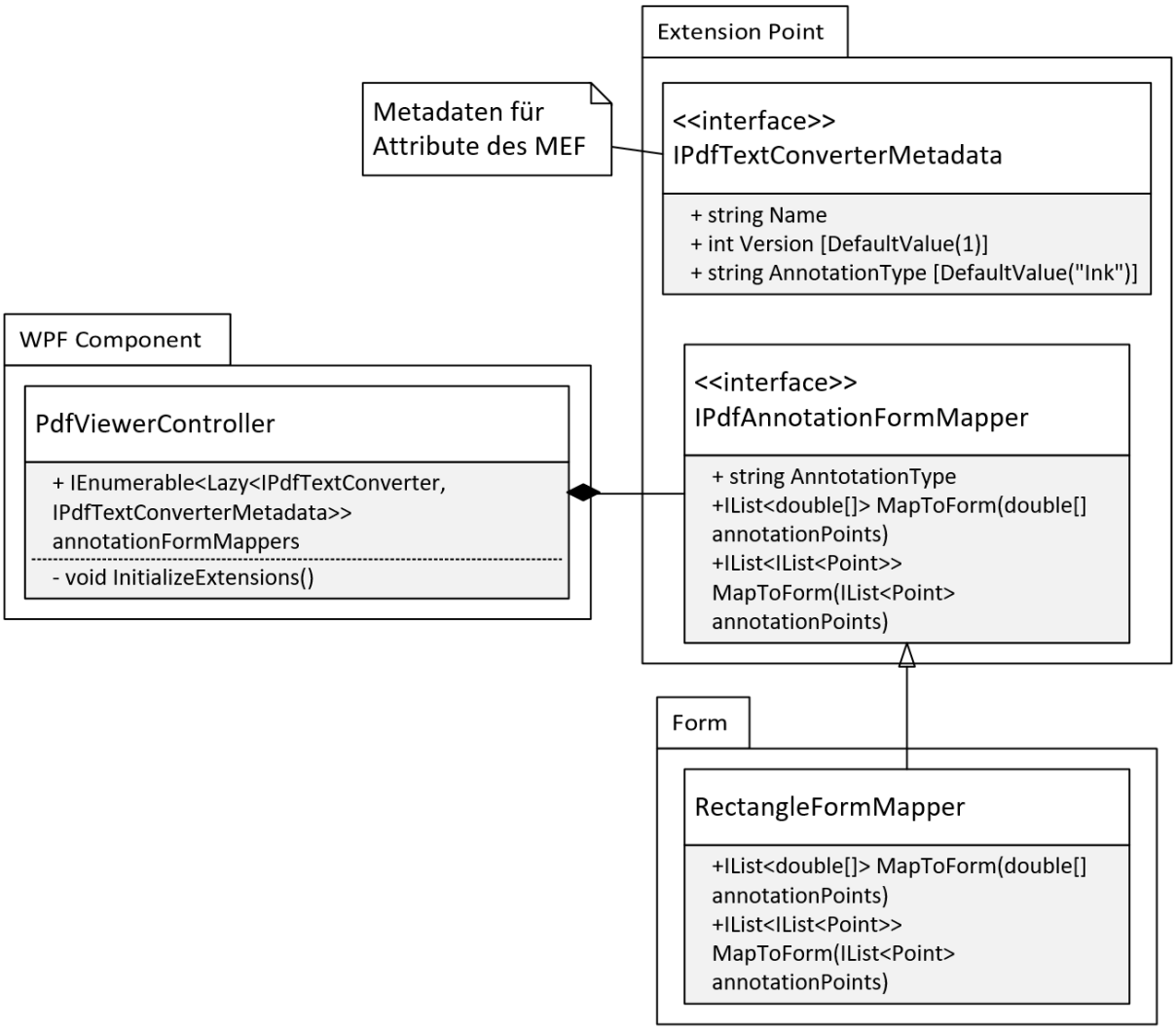


*Abbildung 14: Aufbau der Erweiterung am Beispiel der Formen*

Die beiden MapToForm Funktionen im IPdfAnnotationFormMapper-Interface auf Abbildung 14 geben eine Liste von Punktlisten zurück, da je nach Form statt nur einer einzelnen auch mehrere Annotationen erstellt werden müssen. Es wird also aus jeder Punktliste eine eigene Annotation erstellt. Der Annotationstyp dient der Erstellung von Annotationen eines spezifischen Typs, wie diese in der PDF-Spezifikation vorhanden sind.

Die Funktion, welche mit double-Werten arbeitet, wird zur endgültigen Erstellung der Annotation verwendet, die zweite Funktion mit Points wird zur Darstellung während des Zeichnens benötigt.

Zum Zeitpunkt des Abschlusses der Arbeit bietet die API noch keine Möglichkeit zur Erstellung von spezifischen Annotationsformtypen, dadurch sind die Formen als mehrere Inkannotationen implementiert. Nach Erweiterungen der API wäre es möglich, Annotationstypen wie Kreis, Rechteck oder Ähnliche direkt als einzelne Annotationen des entsprechenden Typs und nicht als Menge von Inkannotationen zu erstellen.

## 11.4 Ablauf

Alle Formen, welche das korrekte Interface implementieren und sich am richtigen Ort befinden und sich korrekt exportieren, werden vom MEF im Konstruktor der Komponente in eine Collection geladen. Es werden zuerst nur die Metadaten aus den Formen geladen, welche zur Identifizierung der Klassen benutzt werden. Stimmen die Metadaten einer Klasse mit der gewählten Form überein, wird diese geladen und als Form verwendet.

*Listing 5: Attribute um die Klasse mit dem korrekten Export zu versehen*

```
[Export(typeof(IPdfAnnotationFormMapper)),
ExportMetadata("Name", "RectangleFormMapper"),
ExportMetadata("Version", 1)]
```

In der Komponente kann von aussen über Attribute, welche an den Controller weitergeleitet werden, die aktuell zu verwendende Form ausgewählt werden. Die OnRender-Methode der PdfViewerPane, welche die aktuell gezeichnete Annotation auf dem Bildschirm darstellt, ruft die MapToForm-Methode der aktuellen Form auf, um die zu erstellende Annotation in der korrekten Form darzustellen.
Nach Abschluss des Zeichnungsvorgangs durch den Benutzer werden vor dem Aufruf der API-Funktion zum Erstellen der Annotation die Punkte durch die MapToForm-Funktion überarbeitet.

Die Punkte werden immer durch die aktuelle Form überarbeitet. Ist keine Form ausgewählt, wird standardmässig der NoChangeFormMapper verwendet, welcher die Punkte wieder unverändert zurückgibt. Der NoChangeFormMapper ist auch als Form implementiert.

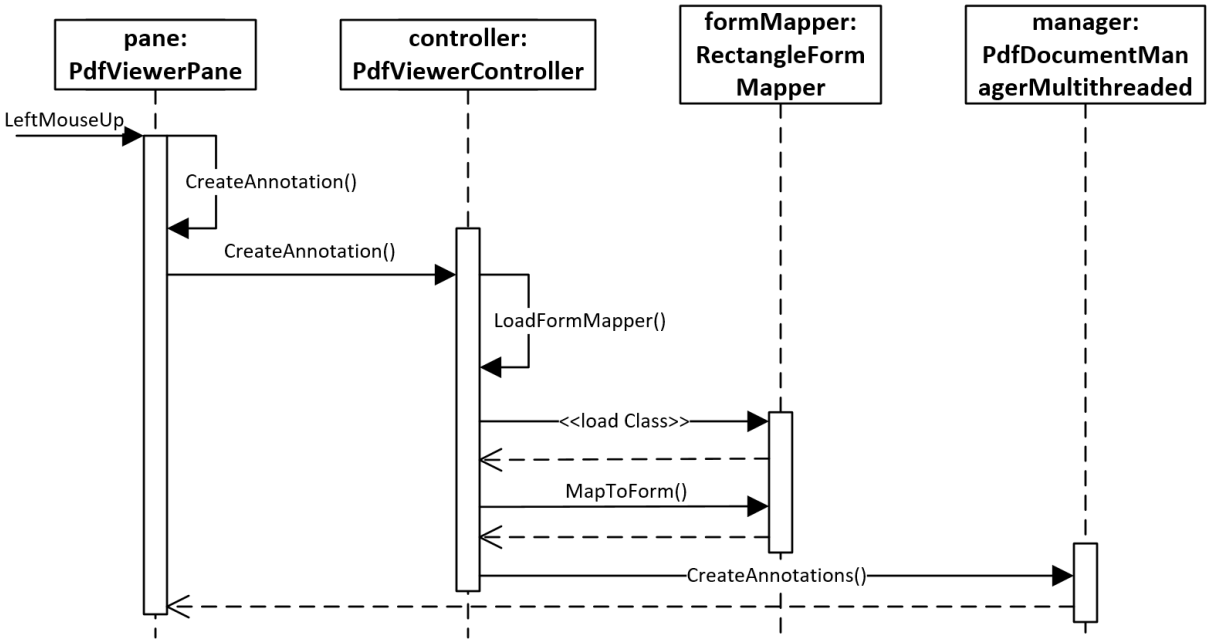


*Abbildung 15: Ablauf des FormMappers*

## 11.5 Implementierte Formen

Die beiden von den Autoren gewählten Formen Rechteck und Polyline unterscheiden sich aber nicht nur in der Verarbeitung der eingegebenen Punkte, sondern auch in der erwarteten Vorgehensweise zum Zeichnen einer solchen Annotation.

Aus diesem Grund werden beide implementierten Mausmodi in den Erweiterungen verwendet. Für den Klickmodus gibt es zwei Funktionen, welche das Abschliessen sowie das Abbrechen einer Annotation auslösen.

Zur einfachen Verwendung wurden die beiden Funktionsaufrufe in der Referenzimplementation mit der Leertaste, respektive mit der Escapetaste verknüpft.

Mithilfe dieser beiden Mausmodi ist es möglich, viele verschiedene neue Formen zu implementieren.

### 11.5.1 Polyline

Die Form Polyline wurde basierend auf dem Klickmodus entwickelt. Um eine Polyline zu zeichnen, wählt ein Benutzer die gewünschten Punkte mittels Mausklick. Zwischen den Punkten werden gerade Linien gezeichnet.

Damit die Polyline aus einzelnen Linien besteht, wird für jede Linie eine einzelne Inkannotation erstellt. Dies ist nötig, da die API bei Annotationen vom Typ Inkannotation automatisch die Ecken abrundet, was bei einer einzelnen Annotation mit mehreren Punkten nicht zur gewünschten Form führt. Dies geschieht, da die Punkte einer Inkannotation in der API mittels kubischen Bézierkurven verbunden werden. Wird in der API der Annotationstyp Polyline implementiert, kann die Erweiterung so umgebaut werden, dass dieser Annotationstyp verwendet wird.

### 11.5.2 Rechteck

Die Form Rechteck beachtet nur den ersten und letzten Punkt, welche sie von aussen erhält. Alle anderen Punkte werden ignoriert. Danach wird mittels vier einzelner Inkannotationen ein Rechteck zwischen Start- und Zielpunkt erstellt. Dies ist ebenfalls nötig, da bei einer einzelnen Annotation die Ecken abgerundet werden. Auch hier kann die Erweiterung auf den Annotationstyp Rechteck umgebaut werden, sobald dieser in der API eingebaut ist.

### 11.5.3 Weitere Formen

Mit den beiden Mausmodi können noch viele weitere Formen implementiert werden. Beispiele dafür wären Ellipsen oder Bézierkurven. Hier muss aber wiederum beachtet werden, dass auch andere Annotationstypen momentan nur begrenzt von der API unterstützt werden. Somit müssten die meisten anderen Formen wie beim Rechteck oder der Polyline durch einzelne Inkannotationen dargestellt werden.

Die Darstellung durch einzelne Inkannotationen bringt aber auch den Vorteil, dass Formen erstellt werden können, welche vom PDF-Format selbst nicht als Annotationstyp unterstützt werden. Dies könnten beispielsweise Zeichen wie Pfeile oder eine vorgefertigte geometrische Form sein.

## 11.6 Fehlerbehandlung

Bei der Verwendung von Erweiterungen können an verschiedenen Stellen Fehler auftreten. An folgenden Stellen werden diese abgefangen und mit einer spezifischen Meldung in die Logdatei geschrieben. Da diese Fehler in den Erweiterungen und nicht in höheren Ebenen auftreten, wird der Benutzer nicht direkt über ein Fehlverhalten informiert.

- Fehler bei der Komposition der Erweiterungen in den beiden Erweiterungsordnern.
- Keine Erweiterung mit dem gesuchten Namen gefunden.
- Fehler beim Laden der gefundenen Erweiterung.
- Ausgeführte Methode auf der gewählten Erweiterung gibt null zurück.

Diese und weiter Fehler sind im Testdokument im Anhang genauer beschrieben.

Eine Möglichkeit, den Benutzer auch direkt von den Fehlern zu benachrichtigen, wäre die jeweilige Rückgabe eines Objektes, welches den eigentlichen Rückgabewert und eine Exception enthält. So können aufrufende Klassen überprüfen, ob in einem der darunterliegenden Schritte eine Exception aufgetreten ist und dies dem Benutzer entsprechend kommunizieren.
Diese Erweiterung konnte aus zeitlichen Gründen nicht mehr in die Komponente eingebaut werden.

## 11.7 Überprüfung

Zur Überprüfung der Erweiterungsschnittstelle wurde nach Abschluss der Entwicklungsarbeiten testweise eine weitere Form erstellt. Diese Form ähnelt dem Rechteck, besteht jedoch nur aus der oberen und unteren Seitenlinie. Die Viewerimplementation wurde um die neue Form erweitert, ebenfalls wurde die Bibliothek im korrekten Ordner abgelegt. Dies hat gezeigt, dass diese Form ohne Änderung an der Komponente eingebaut werden konnte. Somit wurde gezeigt, dass die Erweiterungsfunktionalität mittels MEF die Zeile erreicht.

# 12 Texterkennung

Das maschinelle Erkennen von handschriftlichen Notizen und deren Umwandlung in Textform wird in diesem Kapitel thematisiert.

## 12.1 Vision

Die Komponente soll die Möglichkeit bieten handschriftliche Eingaben, welche vom Benutzer getätigt werden, in Text umzuwandeln und diesen in einer Annotation zu speichern. Auch hier soll beachtet werden, dass Entwickler ihre eigenen Texterkennungsalgorithmen verwenden können, ohne dass die Funktionalität der Komponente angepasst werden muss.

## 12.2 Ablauf

Laut PDF-Spezifikation enthält eine Inkannotation eine so genannte InkList, welche die einzelnen Punkte einer Annotation enthält. Die API stellt momentan noch keine Möglichkeit zur Verfügung, diese InkList auszulesen. Dadurch ist es nicht möglich, bereits erstellte Annotationen in Text umzuwandeln. Aus diesem Grund haben sich die Autoren entschieden, dass im Voraus definiert werden muss, wenn eine handgeschriebene Eingabe in Text umgewandelt werden soll.
Dies haben die Autoren mithilfe eines zusätzlichen Mausmodus, dem TextRecognitionMode umgesetzt. Sobald dieser Mausmodus aktiv ist, werden alle Striche nicht direkt in Annotationen umgewandelt, sondern nur auf die Komponente gezeichnet und in einer StrokeCollection gespeichert. Sobald der Befehl zum Beenden der Texterkennung eingeht, welcher durch einen Funktionsaufruf aus der Komponente ausgeführt wird, werden die Striche an den aktiven TextRecognizer übergeben und in Text übersetzt.

Momentan kann die API noch keine Freitextannotationen erstellen. Dieser Annotationstyp müsste für die Darstellung des übersetzten Textes direkt auf dem PDF vorhanden sein.
Nach Absprache mit dem Kunden wurde entschieden eine Textannotation zu erstellen, welche den übersetzten Text enthält. Sobald das Erstellen von Freitextannotationen möglich ist, kann diese Funktionalität problemlos umgebaut werden, da nur der Annotationstyp der zu erstellenden Annotation angepasst werden muss.

## 12.3 Erweiterungsfunktionalität

Die gesamte Erweiterungsfunktionalität der Texterkennung ist mit demselben Hintergedanken aufgebaut wie die Funktionalität der Formen. Somit wird auch das MEF zur Implementation verwendet. Auch hier sind die TextConverter bereits als Erweiterungen implementiert.

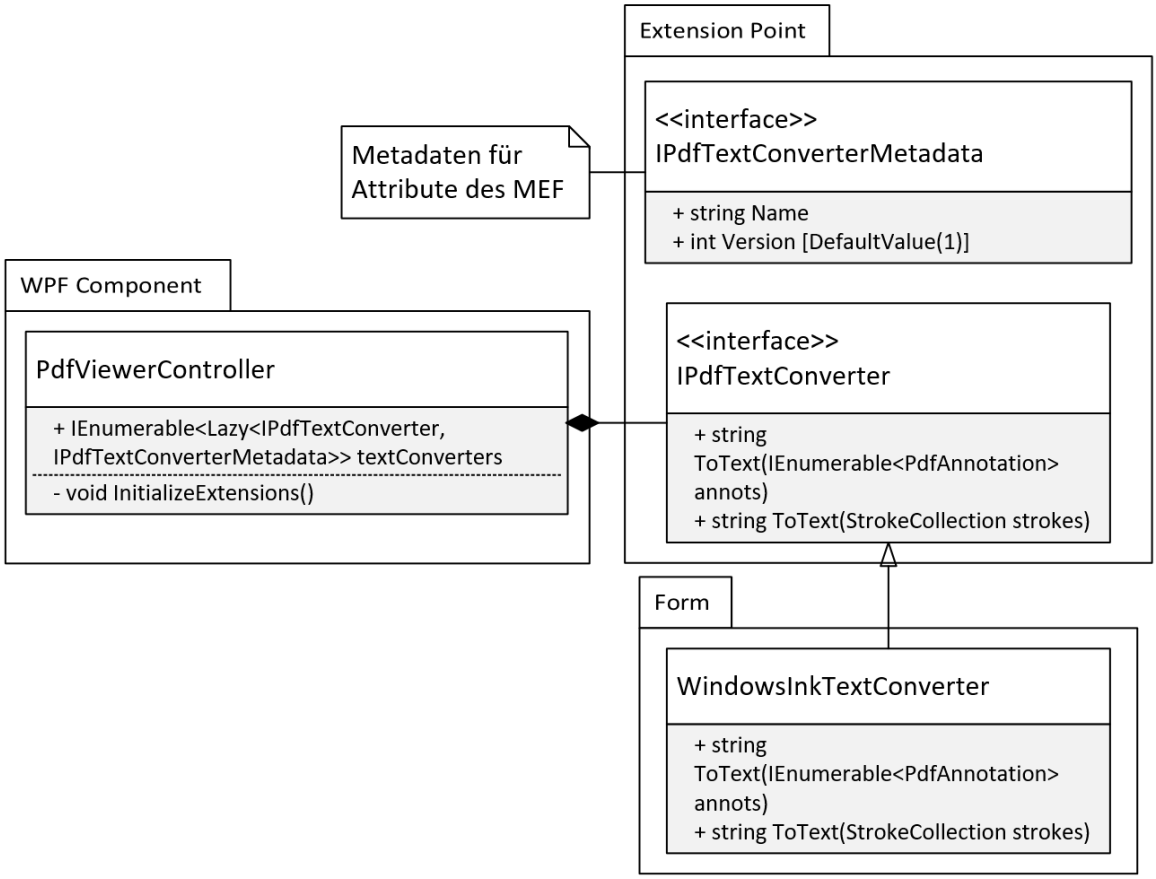


*Abbildung 16: Aufbau der Erweiterungsfunktionalität der Texterkennung*

Das Interface IPdfTextConverter in Abbildung 16 enthält zwei Funktionen. Die ToText-Funktion, welche ein IEnumerable vom Typ PdfAnnotation erhält ist momentan nicht in Gebrauch. Diese Funktion soll aber zur Erkennung von bestehenden Annotationen dienen, sobald von diesen die InkList ausgelesen werden kann.

Die zweite Funktion ist die aktuelle Alternative und verarbeitet Strokes, welche aus der WindowsInk-Bibliothek stammen.

## 12.4 Windows Digital Ink

Als Standard wird zur Konvertierung der Striche in Text die Digital Ink Bibliothek von Microsoft verwendet. Die Funktion erhält eine StrokeCollection, welche von der Bibliothek in einen string umgewandelt werden kann.

## 12.5 Strichzähler

Zusätzlich zur Texterkennung mittels WindowsInk wurde ein weiterer TextConverter implementiert. Dieser hat aber keine Funktionalität zur Übersetzung von Strichen in Text, sondern zählt nur die Striche, welche übergeben werden. Damit wird gezeigt, dass auch die Verwendung eines anderen TextConverters möglich ist.

## 12.6 Problembehandlung

Die Problembehandlung wird bei den TextConvertern auf dieselbe Art und Weise umgesetzt, wie bei den Formen, da sich die auftretenden Probleme in den Erweiterungen sehr ähnlich sind.
Es wird also ein Logger verwendet, mit welchem die Probleme aufgezeichnet werden. Der Benutzer wird wiederum nicht direkt über eine Nachricht informiert.

# 13 Code Analyse

Um zu überprüfen, ob die Erweiterungen den Programmierrichtlinien von Microsoft entsprechen, wurde der Programmcode der Erweiterungen durch das Code Analyse Tool von Visual Studio analysiert.

Diese Analysen wurden auf die neuen Projekte der Erweiterungen beschränkt. Die anderen Projektteile wurden nicht genau analysiert, da der grösste Teil davon bereits vor dieser Arbeit bestehend war und es nicht ohne grossen Aufwand möglich war, nur die in dieser Arbeit erstellten Funktionen zu analysieren.

## 13.1 Code Metrics

Zusätzlich wurden die Code Metrics der gesamten Komponente gemessen. Dabei hat sich herausgestellt, dass die Funktionen, welche die Mausinteraktionen behandeln, sehr komplex sind.

Als Gegenmassnahme schlagen die Autoren vor, zu überprüfen, ob eine Umstellung der Mausmodusfunktionalität in ein Strategy Pattern lohnenswert ist. Diese Umstellung wurde im Rahmen dieser Arbeit aber nicht genauer ausgewertet.

Im Testdokument, welches sich im Anhang befindet, sind die Code Analyse und die Code Metriken genauer beschrieben und analysiert.

# 14 Schluss

Die bestehende WPF-Komponente wurde vollständig analysiert und in diesem Dokument beschrieben. Die Analyse der bisherigen Komponente war aufwändiger als zu Beginn gedacht, da keine bestehende Architekturdokumentation vorhanden war. Dank der ausführlichen Analyse konnte die Annotationsfunktionalität basierend auf der bestehenden Architektur implementiert werden. Das Requestsystem wurde mit den neuen API-Abfragen erweitert. Annotationen auf der PDF-Datei können erstellt, markiert, bearbeitet und gelöscht werden.
Es wurden die Formen Rechteck und Polyline als Inkannotation implementiert. Zur Implementation der Formen wurde das MEF verwendet, was dazu führt, dass ohne grossen Aufwand und ohne Änderungen an der Komponente neue Formen zur Komponente hinzugefügt werden können.
Zur Implementation der Texterkennung wurde ebenfalls das MEF verwendet. Die von den Autoren erstellte Texterkennung nutzt die Windows Ink API und wandelt Linien in eine Textannotation um.

Die erweiterte Komponente kann nun vom Kunden übernommen und weiterentwickelt werden. Weitere Funktionen wären beispielsweise das Skalieren und Verziehen von bestehenden Annotationen. Eine nächste mögliche Erweiterung ist die Implementation der Funktionalität des Radierens einzelner Teile einer Annotation. Diese Erweiterung setzt aber voraus, dass die Inklist der einzelnen Annotationen aus der API ausgelesen werden kann.
Die bestehenden Formen können so umgebaut werden, dass die Formen als Annotation vom entsprechenden Typ (Rechteck, Polyline) und nicht als mehrere Inkannotationen erstellt werden. Eine grössere mögliche Erweiterung ist die Implementation der Funktionalität der Konvertierung von Inkannotationen in Formen. Dabei könnte ein Benutzer eine Form, zum Beispiel ein Rechteck, zeichnen und dies in die Form Rechteck konvertieren lassen. Somit können sich beispielsweise handgezeichnete Diagramme in korrekte Formen konvertieren lassen.
Ebenfalls kann die Texterkennung so angepasst werden, dass Freitextannotationen anstelle von Textannnotationen erstellt werden, damit der Text direkt auf dem Dokument ersichtlich ist. Dies setzt voraus, dass das Erstellen von Annotationen im entsprechenden Typ von der darunterliegenden API unterstützt wird.

# 15 Glossar

| Begriff | Erklärung |
|---|---|
| GUI | Graphical User Interface<br>Benutzeroberfläche |
| Annotation | Text, Zeichnung oder anderes Objekt, welches an einer bestimmten Position mit einer PDF-Datei verknüpft ist. |
| Inkannotation | Auch Freihandannotation. Eine Annotation, welche handschriftliche Eingaben darstellt. |
| API | Application Programming Interface, Programmierschnittstelle. Schnittstelle, an welchem eine Bibliothek oder ein Programm von einem anderen Programm angesprochen werden kann. |
| Convertibles | Ein Laptop, welcher sich über einen Mechanismus in ein Tablet «verwandeln» kann |
| FHNW | Fachhochschule Nordwestschweiz |
| MEF | Managed Extensibility Framework |
| WPF | Windows Presentation Framework |
| Struct | Ein Datentyp, welcher aus einem oder mehreren Datentypen zusammengesetzt wurde |
| Exception | Ausnahmefall, welcher in einem Programm auftritt. |
| DLL | Dynamic Link Library, Dateiformat von Programmbibliotheken |
| Umschliessendes Rechteck | Auch Boundingbox. Das Rechteck, welches die Annotation umschliesst. |
| Textannotation | Auch Sticky Note. Eine Annotation, welche einen Kommentar auf einer PDF-Datei darstellt. |
| Design Pattern | Entwurfsmuster<br>Bewährte Lösungsmethode eines Softwareproblems |
| Template Methode | Entwurfsmuster zur Definition eines Programmablaufs, bei welchem die einzelnen Teileschritte von Kindklassen überschrieben werden |

# 16 Literaturverzeichnis

International Data Corporation Research (2016): Convertible and Detachable Devices Winning Over Consumers in Western Europe, Says IDC. http://www.idc.com/getdoc.jsp?containerId=prEMEA41310416 (abgerufen am 08.07.2017)

Petrusha, Ron; Latham, Luke; tompratt-AQ; yishengjin1413; (2017): Managed Extensibility Framework (MEF). https://docs.microsoft.com/en-us/dotnet/framework/mef/ (abgerufen am 08.07.2017)

Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995): Design Patterns – Elements of Reusable Object-Oriented Software. 43. Aufl, Indianapolis: Addison-Wesley

Martin, Robert C. (2009): Clean Code – A Handbook of Agile Software Craftsmanship. 1. Aufl, Crawfordsville: Pearson Education Inc.

Albahari, Joseph; Albahari Ben (2016): C# 6.0 in a Nutshell. 2. Aufl, Sebastopol: O'Reilly Media Inc.

Seeman, Mark (2011): Dependency Injection in .NET. 1. Aufl, Greenwich: Manning Publications.

Adobe Systems Incorporated (2006): PDF reference – Adobe Portable Document Format, 6. Edition, San José: Adobe Systems Inc.

Weins, Conor (2015): Digital Ink - Ink Interaction in Windows 10. https://msdn.microsoft.com/en-us/magazine/mt590975.aspx (abgerufen am 25.07.2017)

Latham, Luke; tompratt-AQ; yishengjin1413; (2017): Digital Ink. https://docs.microsoft.com/de-ch/dotnet/framework/wpf/advanced/digital-ink (abgerufen am 25.07.2017)

Qiao, Kang; Koren, Alexander; Satran, Michael (2017): Inputs and Devices. https://docs.microsoft.com/en-us/windows/uwp/input-and-devices/ (abgerufen am 03.08.2017)

Matrin, Robert C. (2009): Getting a SOLID start. https://sites.google.com/site/unclebobconsulting-llc/getting-a-solid-start (abgerufen am 13.06.2017)

Martin, Robert C. (2009): The Dependency Inversion Principle. https://web.archive.org/web/20150905081103/http://www.objectmentor.com/resources/articles/dip.pdf (abgerufen am 13.06.2017)

# 17 Abbildungsverzeichnis



# 18 Tabellenverzeichnis



# 19 Codeausschnittverzeichnis

# 20 Ehrlichkeitserklärung

Hiermit erklären wir, die vorliegende Bachelorarbeit selbstständig, ohne Hilfe Dritter und nur unter Benutzung der angegebenen Quellen verfasst zu haben.

Windisch, 18. August 2017

Simon Beck

Matthias Keller

# 21 Anhang

Folgende Dokumente befinden sich im Anhang dieser Arbeit:

- Aufgabenstellung
- Projektvereinbarung
- Header-Datei der API
- Diagramm des Requestsystems
- Testdokument
- Codeübersicht