



universität
uulm

**Fakultät für
Ingenieurwissenschaften,
Informatik und
Psychologie**
Institut für Datenbanken
und Informationssysteme

Entwicklung einer Anwendung zur Definition und Durchführung von Experimenten mit dem SMI iView X Hi-Speed Eye Tracker

Abschlussarbeit an der Universität Ulm

Vorgelegt von:

Sandro Allgaier
sandro.allgaier@uni-ulm.de
987194

Gutachter:

Prof. Dr. Manfred Reichert

Betreuer:

Michael Winter

2021

Fassung 30. Dezember 2021

© 2021 Sandro Allgaier

Satz: PDF- \LaTeX 2 _{ϵ}

Abstract

Auf Grund von Inkompatibilität mit modernen Betriebssystemen ist eine Weiterverwendung der Software SMI Experiment Center zur Steuerung älterer iView X Hi-Speed Eye Tracking-Tower nicht mehr problemlos möglich. Diese Bachelorarbeit behandelt die Entwicklung einer Desktop-Applikation, welche die Definition und Durchführung von Eye Tracking-Experimenten mitunter durch Fernsteuerung des SMI Experiment Centers auch auf aktuellen Betriebssystemen ermöglicht. Die Anwendung wird primär für das Betriebssystem Windows 10 unter Verwendung der C#-Programmiersprache entwickelt. Die Applikation erlaubt allerdings eine einfache Portierung auf alle gängigen mobilen Plattformen und Mac-Systeme. Die Arbeit umfasst eine Analyse des notwendigen Funktionsumfangs der Applikation sowie eine Konzeption der Grundfunktionsweise und der Benutzeroberfläche. Auf den Anforderungen und Konzeptionen wird die Architektur und schließlich die Implementierung basiert. Es wird außerdem ein Einblick in einige wichtige Aspekte dieser Implementierung und in die grundlegende Softwarearchitektur gegeben. Weiter wird auch die fertiggestellte Applikation vorgestellt. Schließlich werden Beschränkungen und Erweiterbarkeit diskutiert.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.2	Zielsetzung	2
1.3	Struktur der Arbeit	2
2	Grundlagen	4
2.1	Eye Tracking	4
2.2	iView X Hi-Speed	6
2.3	.NET 5.0	7
2.4	Visual Studio	7
2.5	Model-View-Viewmodel	9
2.6	Frameworks und Bibliotheken	10
2.6.1	Windows Presentation Foundation	10
2.6.2	MvvmCross	12
	Datenbindung	13
	Navigation	15
2.6.3	Ghostscript	16
2.6.4	MediaToolkit	17
2.7	JavaScript Object Notation	17
3	Anforderungsanalyse	18
3.1	Anforderungen	18
3.1.1	Funktionale Anforderungen	18
3.1.2	Nicht-funktionale Anforderungen	22
3.2	Nutzerrollen	23
3.2.1	Versuchsleiter	23
3.2.2	Versuchsperson	23

3.3	Anwendungsfälle	23
4	Konzept	26
4.1	Sequenzen	26
4.2	Experimentablauf	28
4.3	Benutzeroberfläche	28
4.3.1	Farbschema	28
4.3.2	Positionierung der Elemente	29
4.3.3	Prototyping	30
5	Implementierung	34
5.1	Architektur	34
5.2	Snapshot-Generierung	38
5.3	Speichern und Laden von Experimenten	40
5.4	Logging	42
5.5	Remote-Control-Command-Kommunikation	43
6	Präsentation der Anwendung	48
6.1	Hauptfenster	48
6.2	Neues Experiment	51
6.3	Experiment laden	51
6.4	Experiment bearbeiten	51
6.5	Sequenzeditor	53
6.6	Videoeditor	55
7	Anforderungsabgleich	56
7.1	Funktionale Anforderungen	56
7.2	Nicht-funktionale Anforderungen	60
8	Zusammenfassung	62
8.1	Ausblick	62
	Literatur	64

1 Einleitung

In diesem Kapitel wird die Motivation der Arbeit dargelegt und im Anschluss auf die gesetzten Ziele und den Aufbau der Arbeit eingegangen.

1.1 Motivation

Eye Tracking-Experimente werden nunmehr seit dem 19. Jahrhundert durchgeführt, zu dieser Zeit noch unter direkter Beobachtung der Versuchsperson durch den Versuchsleiter und wenig mehr technologischer Unterstützung zur Beobachtung der Blickrichtung von Probanden als einem Spiegel [34]. Obgleich ungenau, wurden mit dieser Methode bereits wichtige Erkenntnisse über die Psychologie des Menschen durch dessen Blickverhalten errungen. Da rund 80% aller Sinneseindrücke visueller Natur sind, ist der Stellenwert des Eye Tracking in der Erforschung des Menschen hoch.

Technischer und methodischer Fortschritt brachten Eye Tracking seither weit voran. Heutzutage wird Eye Tracking sehr viel präziser über Kamerasysteme oder Elektroden vollführt, deren Rohdaten dann über Computerprogramme ausgewertet werden können. Ein solches computerunterstütztes Kamerasystem ist der Eye Tracking-Tower iView X Hi-Speed von SMI, welcher gestützt durch die Software SMI Experiment Center präzise und verlässliche Eye Tracking-Messdaten liefert [22].

Der iView X Hi-Speed Eye Tracking-Tower an der Universität Ulm ist bereits seit vielen Jahren in Betrieb. Die aktuelle Version des SMI Experiment Centers ist jedoch nicht länger mit dem Eye Tracking-Tower kompatibel und die letzte kompatible Version dieser Software kann nicht auf Windows 10 ausgeführt werden. Aus diesem Grund kann das SMI Experiment Center nur noch auf einem Rechner ausgeführt werden, auf welchem eine veraltete Windows-Version installiert ist.

Das SMI Experiment Center erlaubt Fernsteuerung über eine Netzwerkschnittstelle mit einer spezialisierten Remote Control Language. Diesen Umstand machte man sich bisher zu Nutze und verwendete rudimentäre Software, um Kalibrierung und Stimulus-Präsentation auch von Windows 10 aus durchzuführen.

Die momentan dazu verwendete Software erfordert jedoch bei jeder Änderung der Stimulus-Präsentation Eingriffe in den Programmcode und erlaubt dadurch nur sehr umständliche Durchführungen von Eye Tracking-Experimenten. Dies verlangt Forschern und Studenten nicht-informatischer Bereiche unzumutbare Programmierkenntnisse ab und ist selbst nach Erlangen dieser Kenntnisse mit großem Zeitaufwand verbunden. Die Notwendigkeit einer einfacher bedienbaren und flexibleren Anwendung ist deutlich erkennbar.

1.2 Zielsetzung

Das Ziel dieser Arbeit ist es, eine Applikation zu erstellen, mit welcher Eye Tracking-Experimente definiert und am iView X Hi-Speed in Verbindung mit dem SMI Experiment Center durchgeführt werden können. Die Applikation soll das Abspielen von Reizen in Bild, Video und PDF-Format für gewünschte Zeitperioden erlauben. Des Weiteren soll das Pausieren von Reizen möglich sein, um auf Eingaben einer Versuchsperson zu warten. Diese Eingaben sollen in auswertbarer Form vom Programm ausgegeben werden. Die Applikation soll außerdem die Funktion besitzen, eine Kalibrierung des iView X Hi-Speed mit der Versuchsperson durchzuführen. Erstellte Experimente müssen dann speicher- und ladbar sein.

1.3 Struktur der Arbeit

In Kapitel 2 werden die im Rahmen der Arbeit verwendeten Technologien und weitere relevante Grundlagen behandelt. Darauf folgt die Analyse der an die Applikation notwendigen Anforderungen und möglicher Nutzerrollen in Kapitel 3. Diese werden dort aus der Zielsetzung erhoben. Aus dieser Analyse wird in Kapitel 4 ein Konzept zur Funktionsweise der Anwendung abgeleitet. Hier wird vor allem auf das Konzept

der für die Anwendung erdachten Sequenzen in Kapitel 4.1 und den Ablauf eines Experiments in der Anwendung in Kapitel 4.2 eingegangen. In Kapitel 4.3 wird ein Konzept der Benutzeroberfläche erstellt. Es werden zunächst in Kapitel 4.3.1 wichtige Aspekte der Farbwahl und in Kapitel 4.3.2 Positionierung der UI-Elemente behandelt. Anschließend wird in Kapitel 4.3.3 ein Papierprototyp der Anwendung erstellt.

In Kapitel 5 wird auf wichtige Aspekte der Implementierung und der Softwarearchitektur eingegangen. Zunächst wird dort in Kapitel 5.1 ein genereller Überblick über den Aufbau der Software gegeben. Danach wird die Generierung der Snapshots aus Sequenzen in Kapitel 5.2 behandelt, die Funktionsweise des Ladens und Speicherns innerhalb der Applikation in Kapitel 5.3. Die Erstellung der Programmlogs und Versuchslogs wird in Kapitel 5.4 erklärt. Schließlich wird in Kapitel 5.5 die Kommunikation der Anwendung mit dem SMI Experiment Center mittels Remote Control Commands beschrieben.

In Kapitel 6 wird ein Überblick über die fertige Anwendung gegeben. Es werden alle Fenster, die Navigation zwischen diesen und deren Funktionen besprochen. Anschließend wird die fertige Arbeit in Kapitel 7 mit den zuvor gesetzten Anforderungen abgeglichen und in Kapitel 8 zusammengefasst. In Kapitel 8.1 wird mit einem Ausblick zur möglichen Erweiterbarkeit der Anwendung geschlossen.

2 Grundlagen

Dieses Kapitel dient der Erläuterung der wichtigsten mit dieser Arbeit zusammenhängenden Fachbegriffe und Konzepte. In Kapitel 2.1 wird ein kleiner Überblick über Eye Tracking und dessen Bedeutung in der Forschung gegeben. Im folgenden Kapitel 2.2 wird der Eye Tracking-Tower besprochen, für welchen die im Rahmen dieser Arbeit erstellte Anwendung entwickelt wurde. Danach, in Kapitel 2.3, wird die .NET-Entwicklungsplattform behandelt, auf welcher die Software entwickelt wurde. Kapitel 2.5 dient anschließend der Erklärung des genutzten Entwurfsmusters. In Kapitel 2.6 werden dann das MVVM-Framework MvvmCross, das Grafik-Framework WPF sowie andere wichtige verwendete Technologien erklärt. Zuletzt wird in Kapitel 2.7 die gewählte Serialisierungsmethode ausgeführt.

2.1 Eye Tracking

Eye Tracking ist eine Methode, mit welcher Fixationspunkt, Verweildauer, Sakkaden und andere Augenbewegungen und Parameter bestimmt werden können [11]. Bei Eye Tracking-Experimenten wird dieses Blickverhalten anhand spezifischer visueller Reize getestet, um daraus Erkenntnisse zu ziehen. Diese Reize können beispielsweise auf Computerbildschirmen dargestellt werden.

Eye Tracking-Verfahren werden in den Forschungsfeldern der Mensch-Computer-Interaktion [24, 11], der Psychologie [35, 11] sowie auch der Medizin [19, 11] verwendet. Darüber hinaus findet Eye Tracking auch oft Anwendung im Ingenieurwesen und Marketing, als auch bei der Human-Factors-Forschung [11].

Der Begriff fasst verschiedene Grundmethoden zusammen, mit denen dies bewerkstelligt werden kann [11].

- **Elektrookulografie (EOG)** ist eine Methode, mit der Augenbewegungen durch elektrische Potenzialunterschiede an der das Auge umgebenden Haut gemessen werden können. Die elektrischen Potenziale werden durch das Anbringen von Elektroden gemessen [11].
- **Sklerallinsen** sind Kontaktlinsen, an welchen ein mechanisches oder optisches Referenzobjekt befestigt ist. Diese Methode ist sehr präzise, aber auch sehr invasiv und daher oft unpraktisch [11].
- **Foto- (POG) und Videookulografie (VOG)** sind Methoden, bei denen durch eine Kamera aufgenommene Daten zur Bestimmung der Augenbewegungen genutzt werden. Die Kamera muss entweder am Kopf fixiert sein oder der Kopf selbst wird in einer Halterung angebracht. Um Augenbewegungen erfassen zu können, wird hier die corneale Reflexion, welche die Reflexion von Licht auf der Hornhaut (Cornea) bezeichnet (in Abbildung 2.1 sichtbar), oder die Pupillenbewegung ausgewertet [11, 6].

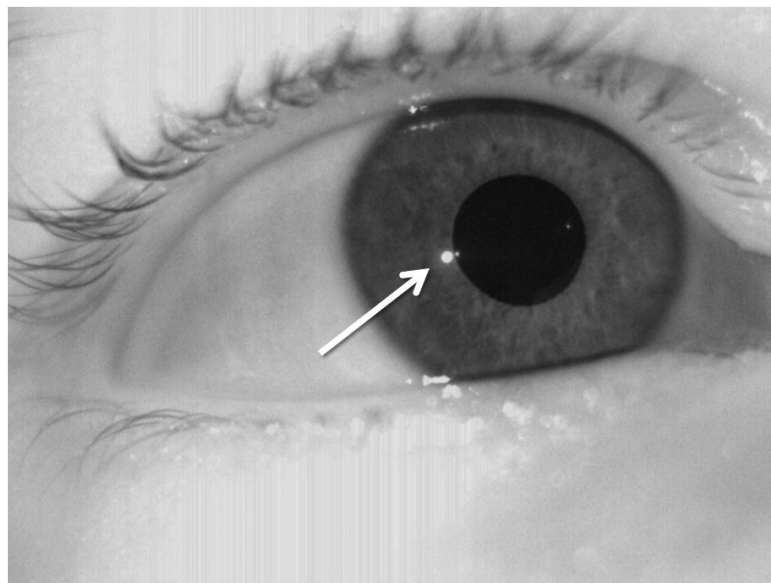


Abbildung 2.1: Die corneale Reflexion (durch einen weißen Pfeil markiert) [20].

Im Rahmen dieser Arbeit ist vor allem die Videookulografie mit der Cornea-Reflexions-Methode betrachtenswert, da diese beim iView X Hi-Speed eingesetzt wird [22].

2.2 iView X Hi-Speed



Abbildung 2.2: Eine Versuchsperson in der iView X Hi-Speed-Arbeitsstation, die einen Text auf dem Stimulus-PC liest [22].

Die iView X Hi-Speed-Arbeitsstation (Abbildung 2.2) ist ein stationärer Eye Tracking-Tower, bei der mit Videookulografie und der Cornea-Reflexions-Methode Daten über die Augenbewegungen von Versuchspersonen erhoben werden [22]. Die iView X Hi-Speed-Arbeitsstation wird in Verbindung mit einem Stimulus-PC verwendet, auf welchem die Software SMI Experiment Center ausgeführt wird, um Reizdarbietung und Kalibrierung zu handhaben.

Der Ablauf eines normalen Eye Tracking-Experimentes sieht an der Arbeitsstation wie folgt aus:

1. Der Kopf der Versuchsperson wird komfortabel in der Arbeitsstation befestigt, wie im Handbuch beschrieben. Die Kamera wird korrekt positioniert, so dass diese das Auge horizontal und vertikal zentriert erfasst.
2. Im Experiment Center wird nun die Kalibrierung durchgeführt, um Punkte auf dem Bildschirm mit den Ausrichtungen des Auges zu verbinden. Dies erfolgt durch eine zufällige Präsentation der Kalibrierungspunkte auf dem Stimulus-PC.

3. Anschließend kann die Darbietung der Stimuli erfolgen. Auf dem Stimulus-PC werden Videos, Bilder oder Dokumente gezeigt, welche von der Versuchsperson betrachtet werden. Die Augenbewegungsdaten werden im Zusammenhang mit dem dargebotenen Reizmaterial in einer .idf-Datei abgespeichert. Diese können für weitere Analysen dann mit dem Programm SMI BeGaze ausgewertet werden.

Die direkte Kontrolle über die iView X Hi-Speed-Arbeitsstation ist mit der Software SMI Experiment Center möglich. Diese Software kann über ein Remote-Control-Command-System über eine Netzwerkverbindung ferngesteuert werden, was die Erstellung von Drittsoftware erlaubt.

2.3 .NET 5.0

.NET 5.0 ist eine von Microsoft entwickelte quelloffene Entwicklungsplattform, welche (unter anderem) die Programmiersprachen C#, F# und Visual Basic mit einer gemeinsamen Laufzeitumgebung und Klassenbibliotheken verknüpft, wie in Abbildung 2.3 gezeigt wird [42]. Dort zu sehen ist, dass alle .NET-Programmiersprachen zunächst in eine gemeinsame Zwischensprache (CIL) mit einer gemeinsamen Laufzeitbibliothek übersetzt werden, bevor diese wiederum in Maschinencode übersetzt wird. .NET 5.0 unterstützt unter anderem die Desktop-Betriebssysteme Windows, Linux und macOS sowie die mobilen Betriebssysteme Android und iOS [9, 45].

Die Implementierung dieser Arbeit wurde in C# durchgeführt. C# ist eine objektorientierte und typsichere Programmiersprache [9]. C# unterstützt Funktionen wie nullable primitive Datentypen, anonyme Funktionen, automatische Garbage-Collection und eine SQL-ähnliche integrierte Query (LINQ) für Listen und Arrays [1].

2.4 Visual Studio

Visual Studio 2019 Community ist eine kostenlose von Microsoft entwickelte und vertriebene integrierte Entwicklungsumgebung (IDE) [41]. Diese kann für das Arbeiten mit einer Vielzahl von Sprachen verwendet werden [38]. Im Rahmen dieser Arbeit ist die gebotene Unterstützung für C#, XAML und JSON relevant.

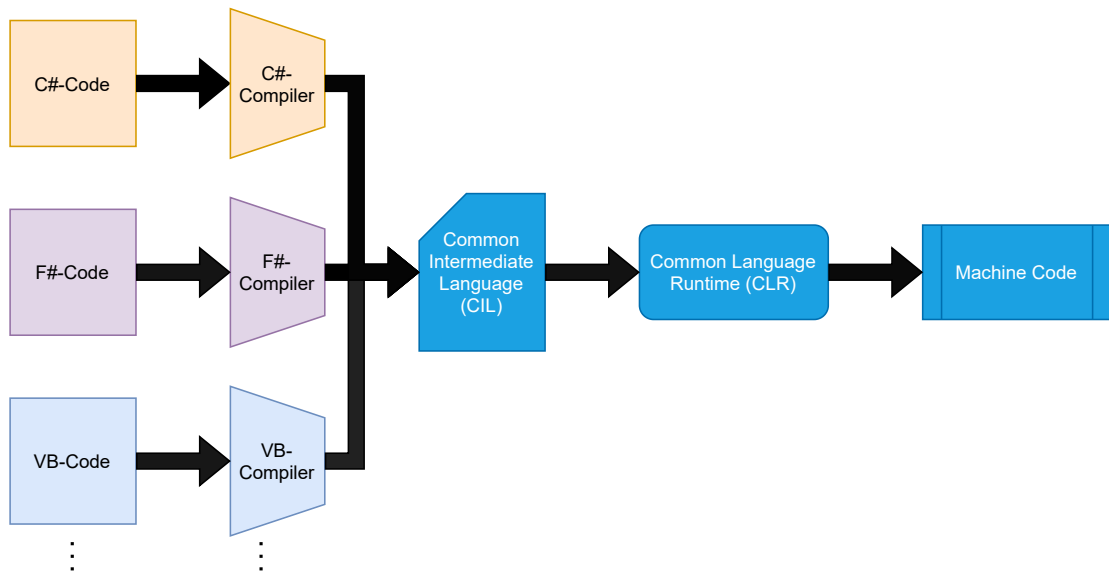


Abbildung 2.3: Ein Überblick über die Architektur von .NET [43].

Visual Studio bietet Debuggingfunktionen wie beispielsweise das Setzen von Breakpoints oder die Echtzeiterfassung von Speicher- und Prozessornutzung. Unter Verwendung des Klassen-Designer-Werkzeugs kann das Projekt in UML-Klassendiagramm-Form bearbeitet werden (siehe Abbildung 2.4) [40]. Das Programmieren selbst wird in Visual Studio mit automatischem Refactoring, Code-Vervollständigung, Highlighting und anderen Optionen vereinfacht.

Das Design von Benutzeroberflächen unter Verwendung des in Kapitel 2.6.1 besprochenen Grafikframeworks WPF ist ebenfalls in Visual Studio möglich, da WPF die Auszeichnungssprache XAML für die Definition der Benutzeroberfläche verwendet. Um Designs auszutesten, können Elemente durch einen integrierten UI-Designer auch durch Drag-and-Drop grob platziert und skaliert werden [7]. Visual Studio bietet zudem durch eine Git-Integration eine Möglichkeit, Versionsmanagement zu handhaben [39].

Bei Bedarf kann die Nutzererfahrung durch das Hinzufügen von Erweiterungen angepasst werden [13]. Zusätzliche Klassenbibliotheken können über den .NET-Paketmanager NuGet hinzugefügt werden [33]. Visual Studio verfügt über Such- und Verwaltungsfunktionen für NuGet und vereinfacht dadurch das Finden, Hinzufügen, Aktualisieren und Löschen externer Klassenbibliotheken (siehe Abbildung 2.5) [21].

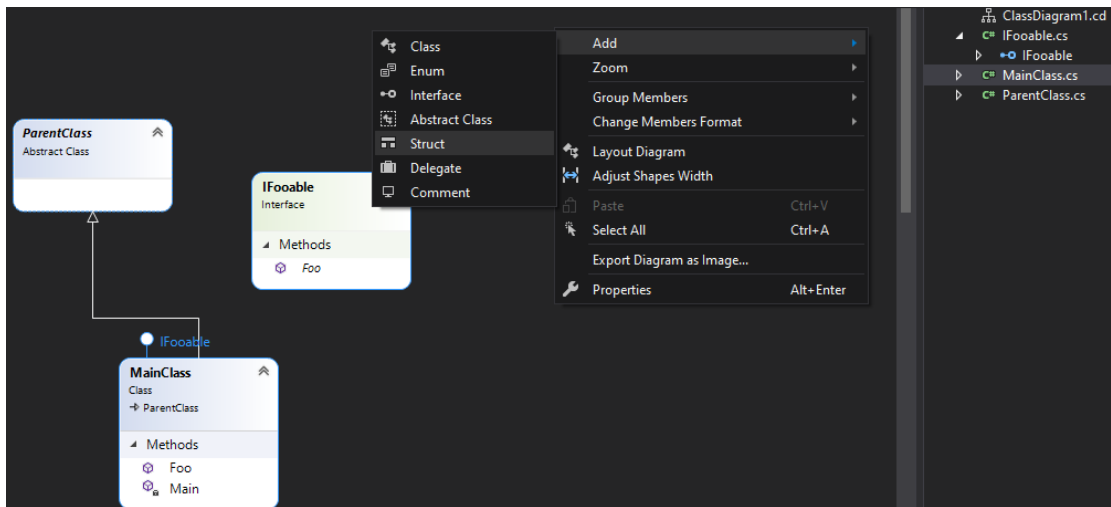


Abbildung 2.4: Im Klassen-Designer kann das Projekt in der Form eines UML-Klassendiagramms erstellt und bearbeitet werden.

2.5 Model-View-Viewmodel

Das gewählte Entwurfsmuster für die Implementierung ist das Model-View-Viewmodel-Muster, welches Entkopplung der Logik vom User-Interface (View genannt) bietet [37]. Ein View wird durch Datenbindung an sein entsprechendes Viewmodel gebunden (siehe Abbildung 2.6) und kann so auf dessen Eigenschaften zugreifen. Dies erfordert in der Implementierung aber lediglich eine einseitige Bindung vom View zum Viewmodel, auch wenn die Bindung in der Ausführung beidseitig besteht. Da Viewmodels und Models also in der Implementierung selbst in keiner Form vom View Kenntnis haben, ist es leichter, Views auch für andere Plattformen zu entwickeln, ohne etwas an Models oder Viewmodels zu ändern.

Viewmodels repräsentieren eine abstrahierte Version der Views, welche Benutzeroberflächenlogik und Nutzern zugängliche Daten enthält. Models hingegen enthalten die der Applikation zu Grunde liegenden Daten und führen die Geschäftslogik aus. Um Interaktion zwischen Nutzer und Datenschicht zu erlauben, stehen Viewmodels und Models in regem Austausch.

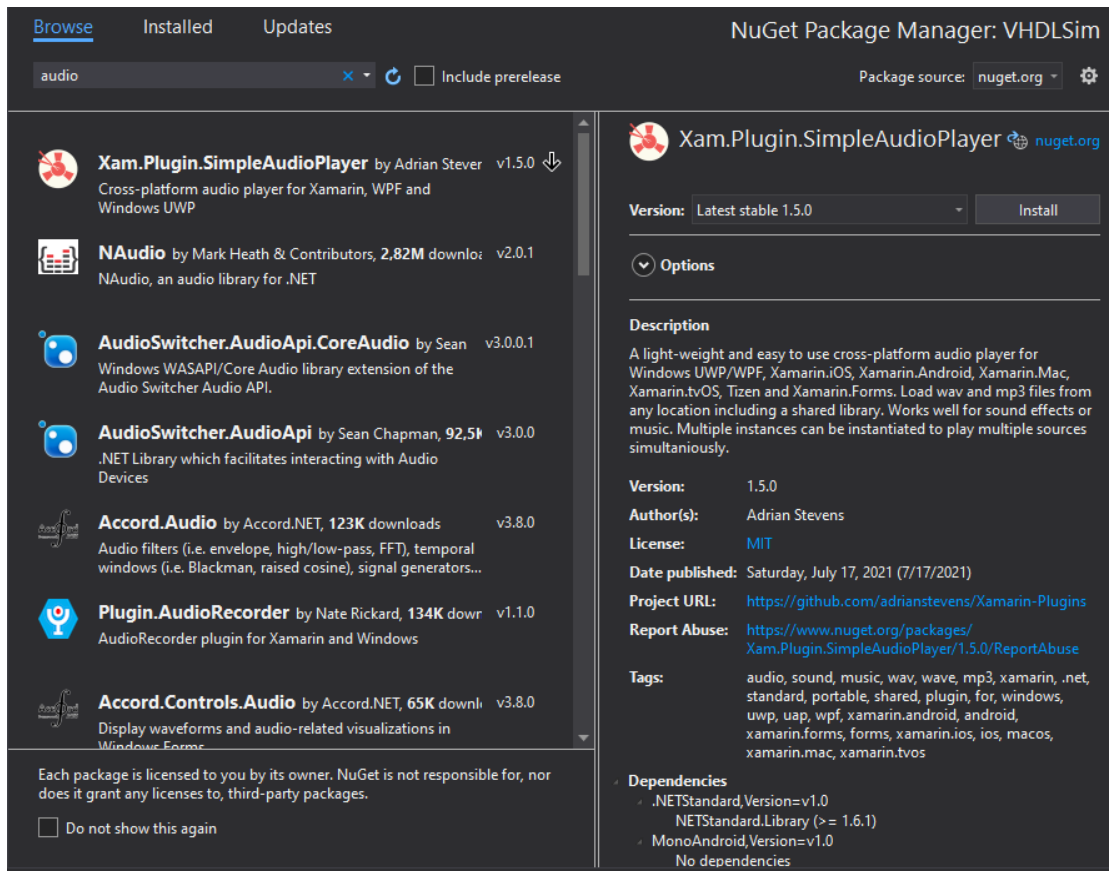


Abbildung 2.5: Die Suchansicht des NuGet-Paketmanagers. Hier können Klassenbibliotheken gesucht und in der erwünschten Version installiert werden.

2.6 Frameworks und Bibliotheken

Dieses Kapitel behandelt das grafische Framework, sowie das Framework, welches zur Umsetzung des MVVM-Entwurfsmusters verwendet wurde. Außerdem werden andere wichtige verwendete Bibliotheken besprochen.

2.6.1 Windows Presentation Foundation

Windows Presentation Foundation, kurz WPF, ist ein C#-GUI-Framework für Windows [44]. In WPF werden Elemente der Benutzeroberfläche hierarchisch angeordnet. Das bedeutet, dass transformierende Änderungen (Position, Sichtbarkeit, etc.)

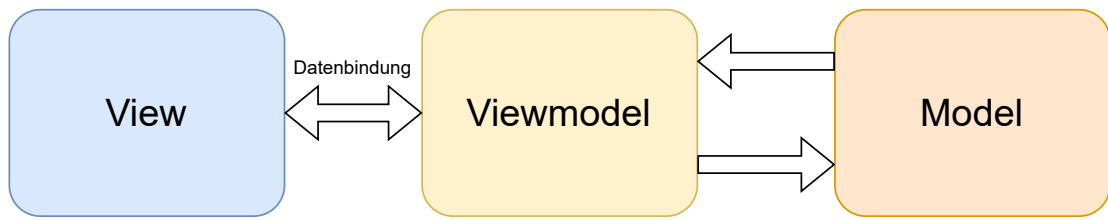


Abbildung 2.6: Die Interaktionen von View, Viewmodel und Model im MVVM-Entwurfsmuster.

eines Elements auch dessen untergeordnete Elemente betreffen. Benutzeroberflächen werden in WPF mit der Extensible Application Markup Language (XAML) erstellt, welche auf XML basiert.

Eine simples `Button`-Element der Höhe und Breite von 20 Pixeln, welches seinerseits hierarchisch einem `Grid`-Element untergeordnet ist, kann beispielsweise wie in Listing 2.1 definiert werden.

Listing 2.1: Beispiel einer hierarchischen Ordnung.

```
1 <Grid>
2   <Button Height="20"
3       Width="20"/>
4 </Grid>
```

Eigenschaften einzelner Elemente können also als Attribute in den Elementen deklariert werden. Attribute können auch auf Eigenschaften aus dem Datenkontext zugreifen. In Listing 2.2 bindet das `Button`-Element seine Höhe und Breite an Werte, welche im Datenkontext gehalten werden.

Listing 2.2: Height und Width binden an den Datenkontext.

```
1 <Button Height="{Binding Height}"
2       Width="{Binding Width}"/>
```

Ohne weitere Spezifikationen ist der Datenkontext der Code-Behind. Eine Benutzeroberfläche in WPF ist immer zweiteilig: Eine partielle C#-Klasse und ihre zugehörige xaml-Datei. Diese bilden zusammen eine vollständige WPF-Klasse. Der angesprochene Code-Behind ist also die zugehörige partielle C#-Klasse. Eine Veränderung des Datenkontextes kann in der xaml-Klasse deklariert werden. Dies führt aber

nicht zu einer vollständigen Entkopplung vom Code-Behind, sondern verändert lediglich das Ziel von Binding-Operationen, welche nur öffentliche Eigenschaften betreffen. Es kann immer noch auf beispielsweise Event-Handler oder Methoden des Code-Behinds verwiesen werden.

2.6.2 MvvmCross

MvvmCross ist ein C#-Framework, welches eine Implementierung des MVVM-Entwurfsmusters erlaubt, ohne dass man sich um die Details der Implementierung der Datenbindung oder Seitennavigation kümmern muss [30, 32, 8]. Mit MvvmCross können Views für

- Xamarin.iOS
- Xamarin.Android
- Xamarin.Mac
- Xamarin.Forms
- Universal Windows Platform (UWP)
- Windows Presentation Foundation (WPF)

erstellt werden [30]. MvvmCross wurde für die Entwicklung mobiler Anwendungen ausgelegt, bietet aber auch die Möglichkeit, Desktop-Applikationen für Windows und Mac zu erstellen.

Das MVVM-Entwurfsmuster wird in MvvmCross so umgesetzt, dass eigene Projekte für die Programmlogik und die jeweiligen View-Projekte innerhalb derselben Projektmappe erstellt werden [31]. Das Projekt, welches Models und Viewmodels enthält, wird Core-Projekt genannt. View-Projekte werden entsprechend ihrer Plattform benannt.

Das Framework wurde für das Projekt gewählt, da es ein Ziel dieser Arbeit ist, eine Applikation zu schreiben, welche die Beschränkungen durch inkompatible/veraltete Software umgeht. Um diesem Problem auch in Zukunft vorzubeugen, ist es notwendig, eine portierbare Anwendung zu entwickeln. Das MVVM-Entwurfsmuster bietet diese Möglichkeit, indem die Geschäftslogik des Programms von der GUI getrennt

und auf eine möglichst lose Kopplung bedacht ist. MvvmCross unterstützt unterschiedliche Plattformen, auf welche unter weiterer Verwendung des Frameworks portiert werden kann. Sollte die Notwendigkeit bestehen, die Software auf einer anderen Plattform zu nutzen, so ist lediglich die Erstellung eines neuen Views für diese Plattform notwendig. MvvmCross erleichtert die Nutzung des MVVM-Entwurfsmusters, da es den meisten Boilerplate-Code implementiert und auch darüber hinaus viele Zusatzfunktionen bietet.

Datenbindung

View-Projekt und Core-Projekt werden durch die in Listing 2.3 erkennbare Anweisung im Code des Views miteinander verbunden [8]. Das Core-Projekt wird mit der in Listing 2.3 gezeigten Anweisung im View-Projekt registriert. Die Funktion `RegisterSetup()` wird durch das Framework selbst beim Start der Applikation aufgerufen. Das Framework erkennt zu Views zugehörige Viewmodels anhand des Klassennamens und verändert entsprechend deren Datenkontexte. Klassen für Views und Viewmodels müssen denselben Namen tragen, welcher dann durch den Suffix `View` oder `ViewModel` appendiert wird. `BeispielView.xaml` (für WPF) wird im View-Projekt automatisch mit `BeispielViewModel.cs` im Core-Projekt verbunden und kann somit alle öffentlichen Eigenschaften dieses Viewmodels verwenden.

Listing 2.3: Der Datenkontext der Views wird über diese Anweisung in der App-Klasse eines View-Projekts zu den Viewmodels im Core-Projekt verändert.

```
1 public partial class App : MvxApplication
2 {
3     protected override void RegisterSetup()
4     {
5         this.RegisterSetupType<MvxWpfSetup<Core.App>>();
6     }
7 }
```

Methoden des Viewmodels können dort in sogenannte Command-Eigenschaften gewrappt werden, welche vom View aus ausführbar sind. Diese können beispielsweise im xaml-Markup eines WPF-Views wie normale Eigenschaften gebunden

werden. Eine ansprechbare Command-Eigenschaft im Viewmodel ist in Listing 2.4 gegeben. Bei ihrer Initialisierung wird dieser eine Methode übergeben, für die sie als Wrapper dient. Dies wird in Listing 2.5 anhand des Commands `BeispielCommand` dargestellt.

Listing 2.4: Commands sind normale Eigenschaften.

```
1 public IMvxCommand BeispielCommand { get; private set; }
```

Listing 2.5: Commands sind Wrapper für Methoden.

```
1 BeispielCommand = new MvxCommand( BeispielFunktion );
```

`BeispielCommand` führt, wenn aus dem View aufgerufen, dann die übergebene `BeispielFunktion()` aus. Der Aufruf aus dem WPF-View durch ein `Button`-Element findet dann über eine Datenbindung an die `BeispielCommand`-Eigenschaft wie in Listing 2.6 statt.

Listing 2.6: Durch `MvvmCross` werden Commands automatisch ausgeführt, wenn diese im Command-Parameter des UI-Elements gebunden werden.

```
1 <Button Content="Beispiel-Button"  
2       Command="{Binding BeispielCommand}" />
```

Änderungen einer Eigenschaft im Viewmodel können durch die in Listing 2.7 gezeigte Methode deklariert werden und teilen so dem View mit, den Wert erneut abzufragen. Wenn eine Eigenschaft im Viewmodel gesetzt wird und ihr zugrundeliegendes Feld ändern soll und gleichzeitig den View über die Änderung benachrichtigen soll, dann kann dies mittels des in Listing 2.8 gezeigten Aufbaus einer Eigenschaft-Setters erreicht werden.

Listing 2.7: Durch diese Methode wird der View über Veränderungen im Viewmodel informiert.

```
1 RaisePropertyChanged( "BeispielEigenschaft" );
```

Listing 2.8: Dieser Aufbau einer Eigenschaft führt neben der Änderung des Feldes auch zur Information des Views über die Wertveränderung.

```
1 private BeispielTyp _beispielFeld;  
2 public BeispielTyp BeispielEigenschaft
```

```
3 {  
4     get => _beispielFeld;  
5     set => SetProperty(ref _beispielFeld, value);  
6 }
```

`SetProperty` setzt also das im ersten Parameter mit dem Schlüsselwort `ref` als direkte Referenz übergebene Feld auf den im zweiten Parameter übergebenen Wert und benachrichtigt dann den View über die Änderung.

Datenbindungen gehen, sofern nicht genauer spezifiziert, in beide Richtungen. Veränderungen im View rufen die `set`-Methode der veränderten Eigenschaft auf und notifizierende Veränderungen einer Eigenschaft im Viewmodel lösen eine Veränderung im View aus. Bei Bedarf ist es jedoch möglich, die Richtung der Datenbindung zu beschränken.

Wichtig zu erwähnen und oft in diesem Projekt verwendet ist außerdem der Datentyp `MvxObservableCollection`. Dieser kann für alle Zwecke wie eine normale `ObservableCollection` verwendet werden und erbt auch von dieser. Normale `ObservableCollections` sind jedoch nicht mit projektübergreifender Datenbindung im Sinne des `MvvmCross`-Frameworks kompatibel, da diese, anders als ihr `MvvmCross`-Gegenstück, nicht ohne Weiteres Thread-übergreifend funktionieren. Alternativ wird also an jeder Stelle, an welcher `ObservableCollections` verwendet würden, `MvxObservableCollections` verwendet.

Navigation

Die Fensternavigation findet mit `MvvmCross` nicht von View zu View, sondern von Viewmodel zu Viewmodel statt, wie in Abbildung 2.7 dargestellt [32]. Ein geöffnetes Viewmodel öffnet dann eine Instanz der verknüpften View-Klasse. Dies erlaubt eine Navigation, die komplett im Core-Projekt untergebracht werden kann.

Das Framework verwendet dafür einen `NavigationService`, der durch eine asynchrone Operation eine simple Navigation von Viewmodel zu Viewmodel ermöglicht. Es genügt die in Listing 2.9 gezeigte Anweisung des `NavigationService`s zu dem zu öffnenden Viewmodel. Bei Bedarf ist es über den `NavigationService` auch möglich Parameter zu übergeben, welche dann vom geöffneten Viewmodel entgegengenommen werden können.

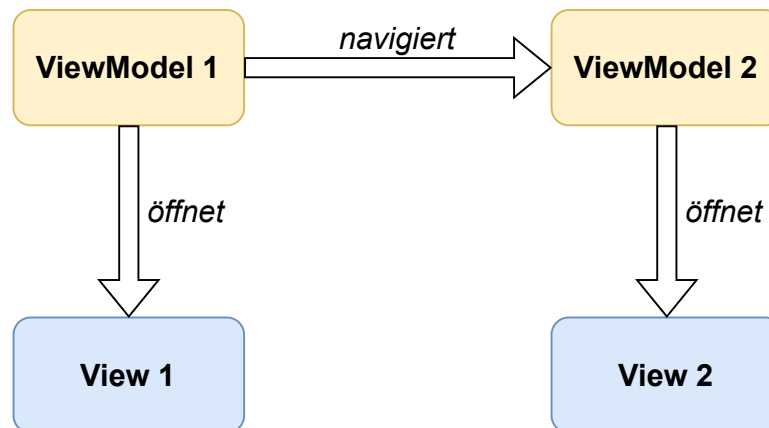


Abbildung 2.7: Die Navigation von Viewmodel zu Viewmodel in MvvmCross.

Listing 2.9: Durch diese asynchrone Anweisung navigiert der Mvvm-Cross-Navigationsservice zum angegebenen BeispielViewModel.

```
1 await _navigationService.Navigate<BeispielViewModel>();
```

2.6.3 Ghostscript

Ghostscript ist ein Programm, mit welchem PDF-Dateien interpretiert und konvertiert werden können [17]. Ghostscript wird für quelloffene Software unter der GNU Affero General Public License veröffentlicht und ist damit kostenlos [17, 15]. Aufgrund der Art, wie in dieser Applikation Snapshots erstellt werden (siehe Kapitel 5.2), ist es für in PDF-Format vorliegende Reize zunächst notwendig, diese in ein verwendbares Bildformat zu konvertieren.

Um Ghostscript in C# nutzen zu können, wird in dieser Implementierung eine C#-Wrapper-Bibliothek namens Ghostscript.NET verwendet [49]. Da Ghostscript auf dem System existieren muss, um von Ghostscript.NET genutzt werden zu können, wurden die erforderlichen Dynamic Link Libraries (DLLs) als Build-Ressource hinzugefügt. Dadurch kann auf eine separate Installation Ghostscripts auf dem System verzichtet werden.

2.6.4 MediaToolkit

Da die Applikation im Core-Projekt aus rohen Videodaten benutzerdefinierte Clips extrahieren und speichern soll, wird die MediaToolkit-Bibliothek verwendet. Diese wurde unter MIT-Lizenz veröffentlicht und ist damit kostenlos und quelloffen [27]. MediaToolkit ist eine Wrapper-Bibliothek für das plattformunabhängige Videobearbeitungsprogramm FFmpeg, welches unter GNU Lesser General Public License, Version 2.1 veröffentlicht wurde und damit ebenfalls kostenlose Nutzung erlaubt [16, 14, 3]. Videoschnitt und -konversion erfolgen in dieser Applikation ausschließlich unter Verwendung dieser Bibliothek.

2.7 JavaScript Object Notation

JSON ist eine textbasierte und plattformunabhängige Syntax, um Daten zwischen oder innerhalb von Applikationen zu teilen [12]. Es bietet eine Möglichkeit, Experimentdaten zu serialisieren, um diese als Textdatei abzuspeichern. JSON erlaubt auch die Deserialisierung und Reinstanzierung einer solchen Textdatei als Klasse, so dass Daten über mehrere Sessions hinweg abgespeichert und wieder geladen werden können.

Ein JSON-String wird aus Datenstrukturen aus Name-/Wert-Paaren und Listen aus geordneten Werten zusammengesetzt [23]. Diese Datenstrukturen können in Objekten gespeichert werden. Objekte selbst sind ebenfalls Werte, wodurch sich eine hierarchische Struktur entwickelt, in welcher Objekte in anderen Objekten verschachtelt sein können.

Für die Implementierung wurde das Newtonsoft Json.NET-Nuget-Paket gewählt [25].

3 Anforderungsanalyse

In diesem Kapitel werden Anforderungen an die Applikation analysiert. Dies beinhaltet, welche Funktionen umgesetzt sein müssen (Kapitel 3.1.1), welche Qualitätsstandards die Software erfüllen muss (Kapitel 3.1.2) und welchen Handlungsspielraum einzelne Nutzerrollen im System besitzen sollen (Kapitel 3.3).

3.1 Anforderungen

In diesem Kapitel wird ausgelegt, welche Kriterien die Software erfüllen muss. Dies beinhaltet sowohl quantifizierbare Merkmale, wie beispielsweise implementierte Funktionen, als auch qualitative Merkmale, welche die Nutzererfahrung mit der Applikation betreffen.

3.1.1 Funktionale Anforderungen

ID	Name	Beschreibung
1	Experiment erstellen	Der Nutzer muss ein neues Experiment erstellen können. Dies muss die Auswahl der Auflösung der erstellten Snapshots und die Anzahl der Kalibrierungspunkte ermöglichen.

3 Anforderungsanalyse

ID	Name	Beschreibung
2	Experiment bearbeiten	Der Nutzer muss das Experiment bearbeiten können. Dabei sollen Hintergrundfarbe der generierten Snapshots, Auflösung der Snapshots, Name sowie Anzahl der Kalibrierungspunkte gewählt werden können.

ID	Name	Beschreibung
3	Experiment speichern	Das Experiment muss speicherbar sein. Vorhandene Komponenten und Einstellungen sollen serialisiert und ab-speicherbar sein.

ID	Name	Beschreibung
4	Experiment laden	Das Experiment muss ladbar sein. Die Anwendung muss imstande sein, die serialisierten Daten auszulesen und wieder als vollständiges Experiment zu erstellen.

ID	Name	Beschreibung
5	Reize laden	Reize in den Dateiformaten .pdf, .mp4, .jpg, .bmp, .png und .gif müssen ladbar sein und in der Applikation angezeigt werden können.

ID	Name	Beschreibung
6	Experiment-komponenten	Experimentkomponenten der Typen Sequenz und Video müssen erstellt werden können.

ID	Name	Beschreibung
7	Sequenzen bearbeiten	Sequenzen müssen Folgen aus Bildreizen enthalten können, welche individuelle und bearbeitbare Skalierungen, Positionen, Start- und Endzeiten und Anzeigeebenen besitzen. Bildreize können frei hinzugefügt oder gelöscht werden.

3 Anforderungsanalyse

ID	Name	Beschreibung
8	Videos bearbeiten	Für Videos muss Start- und Endzeit sowie der verwendete Reiz wählbar sein.

ID	Name	Beschreibung
9	Experimentkomponenten duplizieren	Experimentkomponenten können inklusive ihrer enthaltenen Reize dupliziert werden.

ID	Name	Beschreibung
10	Experimentkomponenten anordnen	Experimentkomponenten können innerhalb des Experiments nach Belieben angeordnet werden und werden in dieser Reihenfolge abgespielt.

ID	Name	Beschreibung
11	Experimentkomponenten aktivieren/-deaktivieren	Experimentkomponenten müssen deaktivierbar und wieder aktivierbar sein. Deaktivierte Komponenten sollen nicht weiter in der Präsentation berücksichtigt werden.

ID	Name	Beschreibung
12	Experimentkomponenten löschen	Experimentkomponenten können bei Bedarf gelöscht werden.

ID	Name	Beschreibung
13	Snapshots erstellen	Snapshots sollen aus der Auswahl an aktiven Slots in einer Sequenz zusammengesetzt werden. Snapshots sind Bilder im .png-Dateiformat in der im Experiment ausgewählten Auflösung.

3 Anforderungsanalyse

ID	Name	Beschreibung
14	Snapshots speichern	Beim Durchführen eines Experiments sollen die erstellten Snapshots in einem eigenen Unterordner des Experimentordners gespeichert werden, damit diese für die Auswertung der Eye Tracking-Daten verfügbar sind.

ID	Name	Beschreibung
15	Snapshots anzeigen	Snapshots müssen in der Applikation einsehbar sein.

ID	Name	Beschreibung
16	Netzwerkverbindung	Es muss möglich sein, sich über das Netzwerk mit dem Computer, auf welchem das SMI Experiment Center läuft, zu verbinden. Verbindungsinformationen müssen vom Nutzer eingebbar sein.

ID	Name	Beschreibung
17	Kalibrierung	Die Applikation muss die Kalibrierung am SMI Experiment Center einleiten können. Sie muss die Kalibrierungsbefehle, welche vom SMI Experiment Center geschickt werden, richtig ausführen.

ID	Name	Beschreibung
18	Experiment durchführen	Das Experiment muss korrekt durchgeführt werden können. Relevante Remote-Commands müssen mit korrektem und vollständigen Informationsgehalt an das SMI Experiment Center geschickt werden.

ID	Name	Beschreibung
19	Versuchspersonen-Input	Versuchspersonen müssen mittels der Pfeiltasten Eingaben tätigen können. Diese Eingaben müssen geloggt werden.

3.1.2 Nicht-funktionale Anforderungen

ID	Name	Beschreibung
1	Modifizierbarkeit	Der Aufwand, die Applikation abzuändern und Fehler zu beseitigen, soll möglichst gering gehalten werden.

ID	Name	Beschreibung
2	Analysierbarkeit	Der Aufwand, Fehler in der Applikation auf ihre Ursachen zu überprüfen, soll möglichst gering gehalten werden.

ID	Name	Beschreibung
3	Portierbarkeit	Es soll möglichst einfach sein, die Applikation auf andere Plattformen zu portieren.

ID	Name	Beschreibung
4	Stabilität	Unerwartete fehlerhafte Zustände sollen möglichst selten auftreten.

ID	Name	Beschreibung
5	Verständlichkeit	Benutzer sollen das Konzept der Applikation schnell verstehen.

ID	Name	Beschreibung
6	Erlernbarkeit	Benutzer sollen die Bedienung der Applikation schnell verstehen.

ID	Name	Beschreibung
7	Bedienbarkeit	Der Aufwand, die Applikation in ausreichendem Umfang richtig zu bedienen, soll gering gehalten werden.

3.2 Nutzerrollen

Die zu berücksichtigenden Akteure im System. Der vollständige Handlungsraum der jeweiligen Rollen kann im Kapitel 3.3 eingesehen werden.

3.2.1 Versuchsleiter

Versuchsleiter erstellen und laden Experimente und deren Komponenten. Außerdem führen sie Kalibrierung und Experimentstart durch. Ihre Rolle im System beinhaltet alles, außer Eingaben während des Experiments selbst.

3.2.2 Versuchsperson

Nachdem ein Experiment gestartet wurde, können Versuchspersonen auf Reize mit Eingaben auf den Pfeiltasten reagieren und so beispielsweise Antwortmöglichkeiten auswählen oder Reflextests absolvieren.

3.3 Anwendungsfälle

Nachdem die Anwendung gestartet wurde, kann der Versuchsleiter entweder ein Experiment erstellen oder ein bereits bestehendes Experiment laden. Anschließend werden weitere Optionen verfügbar. Ab dann kann das derzeitige Experiment bearbeitet werden. Es können Komponenten hinzugefügt, bearbeitet, dupliziert, gelöscht oder in ihrer Reihenfolge verändert werden. Komponenten können entweder Videos oder Sequenzen sein. Videos können bearbeitet werden. Sie enthalten einen Videoreiz, einen Startzeitpunkt und ihre Dauer. Sequenzen enthalten eine

Liste an Slots, welche jeweils wieder hinzugefügt, bearbeitet und gelöscht werden können. Komponenten können generell zusätzlich aktiviert und deaktiviert werden. Nach der Eingabe von Verbindungsinformationen kann der Versuchsleiter dann die Kalibrierung starten. Diese wird von der Versuchsperson absolviert. Danach ist es möglich, das Experiment durchzuführen. Dieses wird ebenfalls von der Versuchsperson absolviert. Sie kann dabei Tasteneingaben auf den Pfeiltasten durchführen. Diese Tasteneingaben sind die einzigen direkten Interaktionen mit der Applikation, welche von einer Versuchsperson durchgeführt werden können. Durch das Schließen des Fensters wird die Applikation vom Versuchsleiter beendet. Eine komplette Übersicht über alle Anwendungsfälle ist in Abbildung 3.1 gegeben.

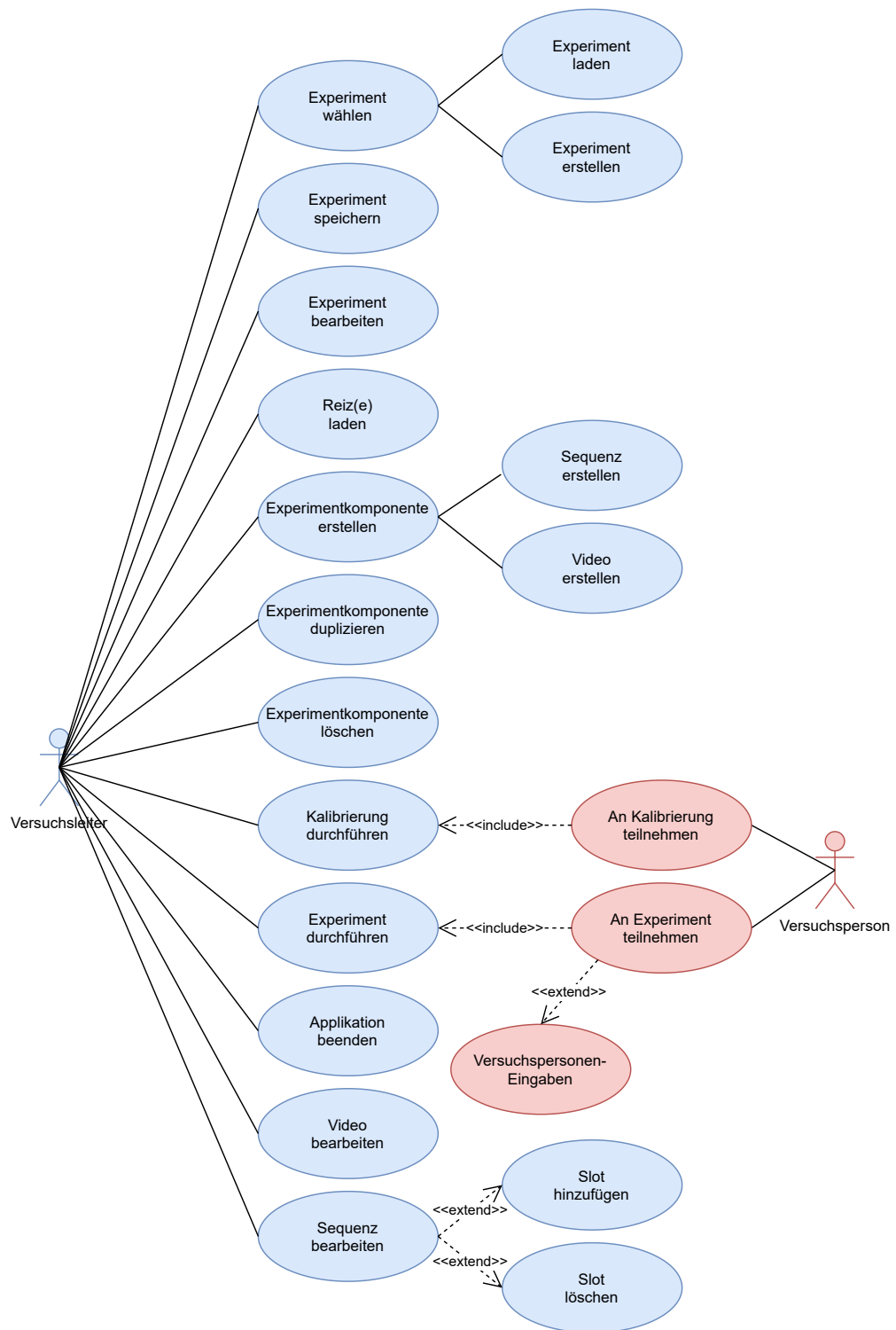


Abbildung 3.1: Das UML-Anwendungsfalldiagramm, welches die verschiedenen Anwendungsfälle für die Nutzerrollen zeigt.

4 Konzept

Die Applikation soll es ermöglichen, Eye Tracking-Experimente zu definieren und durchzuführen. Experimente sollen speicher- und ladbar sein. Das Einfügen neuer Reize muss ebenfalls möglich sein. Experimente sollen über einzelne Komponenten definiert werden, welche entweder Videos oder Reizsequenzen sind.

Videos sind Komponenten, bei welchen eine Datei in einem unterstützten Videoformat aus der Reizliste gewählt und beschnitten werden kann. Sequenzen werden im Kapitel 4.1 behandelt. In Kapitel 4.2 wird die Durchführung eines Experiments näher behandelt. Ferner wird in Kapitel 4.3 das Konzept der Benutzeroberfläche beschrieben.

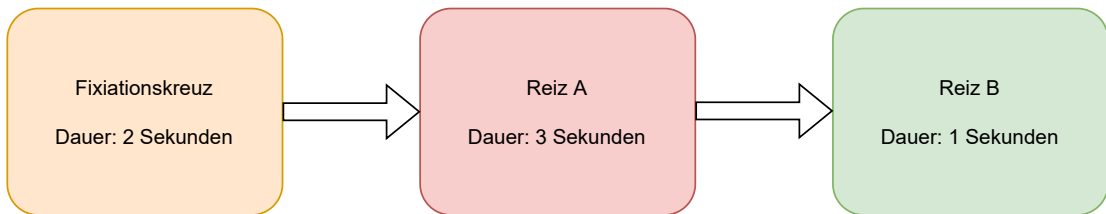
4.1 Sequenzen

Ein Problem beim Definieren eines Eye Tracking-Experiments kann sein, dass man Reizfolgen erstellen muss, welche sich wie in Abbildung 4.1 nur marginal unterscheiden. Trotzdem erfordert dies die Definition einer komplett neuen Abfolge und erzeugt dadurch repetitive Arbeit.

Eine mögliche Lösung für dieses Problem ist es, Sequenzen zu definieren, in welcher Slots mit vom Reiz unabhängigen Positions-, Dauer- und Skalierungseinstellungen hinzugefügt werden können. Wenn eine neue Sequenz erforderlich ist, die sich lediglich in den präsentierten Reizen von einer existierenden Sequenz unterscheidet, kann die existierende Sequenz dupliziert und mit den neuen Reizen gefüllt werden.

Ein weiteres Problem tritt auf, wenn mehrere Bildreize gleichzeitig angezeigt werden sollen. Für jede benötigte Bildreiz-Permutation muss dann eine Datei in einem

Abfolge 1:



Abfolge 2:

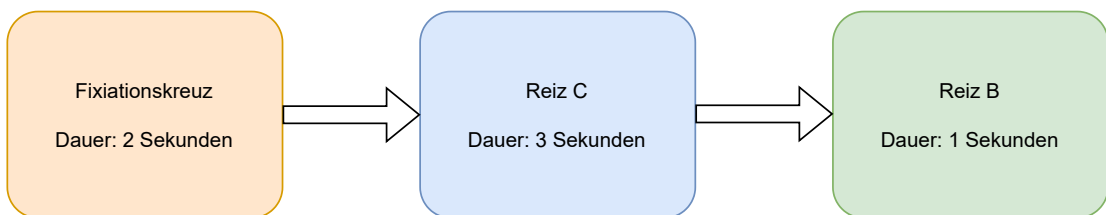


Abbildung 4.1: In den beiden Abfolgen unterscheidet sich lediglich das zweite Element, obwohl auch dort die Dauer gleich bleibt.

Bildbearbeitungsprogramm erstellt werden. In der in Abbildung 4.2 gegebenen Situation müsste zum Beispiel eine separate Datei für die Überschneidungsperiode von Reiz A und Reiz B, welche im zweiten Schritt zu sehen ist, erstellt werden.

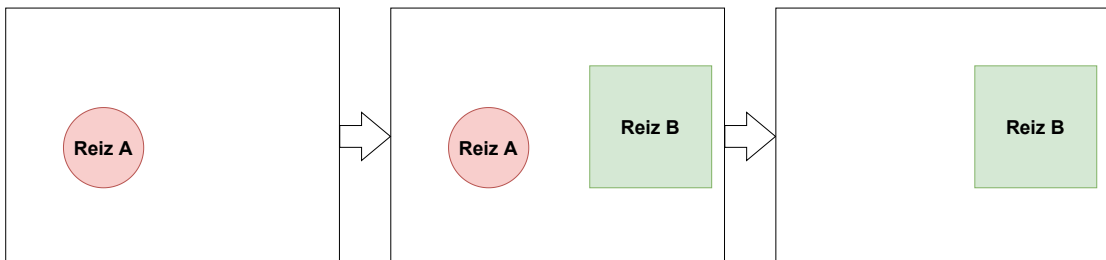


Abbildung 4.2: In dieser Abfolge überschneiden sich Reiz A und Reiz B.

Das vorliegende Problem kann gelöst werden, indem mehrere Slots gleichzeitig aktiv sein können und diese dann in eine Bilddatei zusammengefügt werden. Es wird für jede Änderung in der Konstellation aktiver Slots automatisch eine Bilddatei erstellt. Diese zusammengefügt Bilddateien werden in der Applikation Snapshots genannt.

4.2 Experimentablauf

Nach dem Erstellen oder Laden eines Experiments werden die Verbindungsinformationen des Computers, auf welchem das SMI Experiment Center ausgeführt wird, eingetragen. Diese Verbindungsdaten bestehen aus IP-Adresse und Port.

Dann wird eine Kalibrierung an der Versuchsperson durchgeführt. Bei dieser wird der Applikation vom SMI Experiment Center mittels der Remote Control Commands eine vom Versuchsleiter vorher in der Applikation eingestellte Anzahl an Kalibrierungspunkten geschickt [22]. Diese werden dann auf Anfrage des SMI Experiment Centers durch weitere Remote Control Commands in zufälliger Reihenfolge abgespielt.

Nachdem die Kalibrierung erfolgreich abgeschlossen wurde, beginnt die Präsentation der Snapshots und Videos. Wann diese starten, enden und wie diese benannt wurden wird mittels Remote Control Commands an das SMI Experiment Center übermittelt. Der Versuchsperson ist an dieser Stelle, falls dies erwünscht ist, die Möglichkeit gegeben, mit Tasteneingaben der Pfeiltasten auf Reize zu reagieren. Abschließend können Eingaben in einer Versuchslog-Datei ausgewertet werden.

4.3 Benutzeroberfläche

Da ein Ziel des Projektes ist, eine Applikation zu entwickeln, durch welche das Definieren und Durchführen von Eye Tracking-Experimenten einfacher wird, ist eine funktionale, intuitive und einfach bedienbare Benutzeroberfläche ein integraler Bestandteil der Aufgabe.

4.3.1 Farbschema

Durch das Nutzen eines Darkmode-Farbschemas (Hell-auf-Dunkel-Farbschema) kann Augenermüdung vermindert werden, obgleich viele Nutzer Dunkel-auf-Hell-Farbschemata bevorzugen [46]. Diese Nutzerpräferenz deckt sich nicht damit, dass Nutzer generell weichere Farben im Hintergrund bevorzugen und dunklere Farben generell weicher sind [48]. Die Bevorzugung von Dunkel-auf-Hell-Farbschemata

kann aber durch Nutzungsgewohnheiten erklärt werden, da diese Farbschemata verbreiteter sind [46].

Aufgrund des Vorteils durch verminderte Augenermüdung und der unklaren Situation der Nutzerpräferenz in der Zukunft wird für das Projekt ein Darkmode-Farbschema mit dunkelgrauen Hintergrundtönen und hellblauem und weißem Text gewählt (vgl. Tabelle 4.1).





Farbe	Hex-Code	RGB	Nutzung
	#FFFFFF	(255, 255, 255)	Nicht-interagierbarer Text
	#87CEEB	(135, 206, 235)	Interagierbarer Text; selektierte Elemente; Log
	#28353B	(40, 53, 59)	Hintergrund von interaktiven Kontrollelementen
	#17191A	(23, 25, 26)	Hintergrund (generell)

Tabelle 4.1: Das Farbschema der Applikation.

Die Entscheidung, Hellblau für interagierbaren Text und selektierte Elemente zu wählen, wurde getroffen, da dieser Farbton gut dafür geeignet ist, die periphere Aufmerksamkeit von Nutzern auf sich zu ziehen [48]. Änderungen in der Log oder den Experimentkomponenten können so leichter von Nutzern erkannt werden.

4.3.2 Positionierung der Elemente

Elemente der Benutzeroberfläche sollten bestenfalls so angeordnet sein, dass diese vom Nutzer bestmöglich verarbeitet werden können und dessen Aufmerksamkeit auf die wichtigen Regionen des Bildschirms lenken.

Es gibt eine funktionale Asymmetrie der visuellen Informationsverarbeitung in den beiden Gehirnhälften des Menschen [26, 29], welche auch bei der Entwicklung von Benutzeroberflächen beachtet werden muss. So ist das visuelle Feld des Menschen dahingehend in zwei Felder unterteilt, wie in Abbildung 4.3 zu sehen ist.

Das linke visuelle Feld eignet sich besonders gut zur Verarbeitung von nicht-alphabetischem Material wie geometrischen Formen [5] - also auch Bildern. Aus diesem

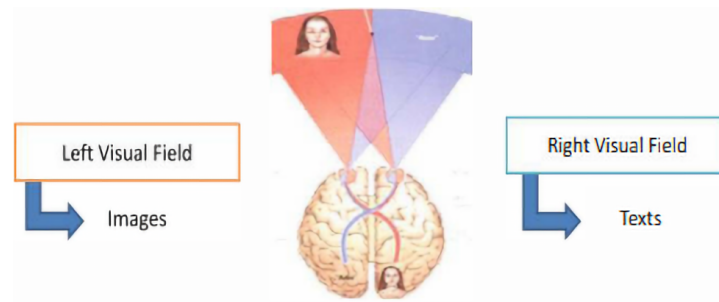


Abbildung 4.3: Das linke und rechte visuelle Feld mit den jeweiligen funktionalen Schwerpunkten [47].

Grund werden Bilder hauptsächlich links angezeigt und Kontrollelemente, welche sich im linken Bereich des visuellen Felds befinden, in Bildform dargestellt.

Im rechten visuellen Feld werden alphabetische Reize, also Buchstaben und Texte, besser verarbeitet [28]. Aus diesem Grund wird Text bevorzugt rechts in der Applikation dargestellt.

Des Weiteren wird bei der Platzierung von Elementen auf die kulturell vermutlich primär westlich geprägte Nutzergruppe der Mitglieder der Universität Ulm eingegangen. Information in Textform wird in dieser Nutzergruppe eher von links nach rechts und von oben nach unten abgearbeitet [2]. Deshalb werden Kontrollelemente so platziert, dass diese ihrer Nutzungsreihenfolge oder Nutzungshäufigkeit entsprechend so angeordnet sind, dass häufiger oder früher zu nutzende Elemente links/oben in ihrer jeweiligen Kontrollengruppe sind.

4.3.3 Prototyping

Um frühzeitig eine Vorstellung von der generellen Form der Benutzeroberfläche zu erhalten wird ein Papierprototyp der Anwendung erstellt. Die Entscheidung einen Papierprototypen anstatt eines digitalen Prototypen oder Wireframes zu erstellen wurde getroffen, weil in Visual Studio, wie in Kapitel 2.4 erwähnt wird, ein UI-Designer enthalten ist, welcher grobe Previews von Design-Entscheidungen durch Drag-and-Drop-Platzierung von GUI-Elementen erlaubt. Dadurch kann in Visual Studio während der Entwicklung schnell und effektiv prototypisiert werden. Zeitkonsumierend sind in WPF lediglich die spezifischeren Anpassungen der GUI-Elemente,

wie Verankerung, Datenbindung, Skalierung bei Größenveränderung, etc. Diese Anpassungen sind für Prototypen aber nicht notwendig.

Für alle Listen in den Papierprototypen gilt, dass Elemente per Rechtsklick über ein Kontextmenü entfernt, dupliziert und bearbeitet werden können. Es folgen simplifizierte Beschreibungen der Papierprototypen. Eine ausführlichere Beschreibung der Funktionalität der fertigen Benutzeroberfläche wird in Kapitel 6 gegeben.

In Abbildung 4.4 ist der Entwurf des Hauptfensters sichtbar. Von hier aus soll eine Navigation zu den Fenstern zum Erstellen, Laden und Bearbeiten von Experimenten und zu Sequenz- und Videoeditorfenstern möglich sein.

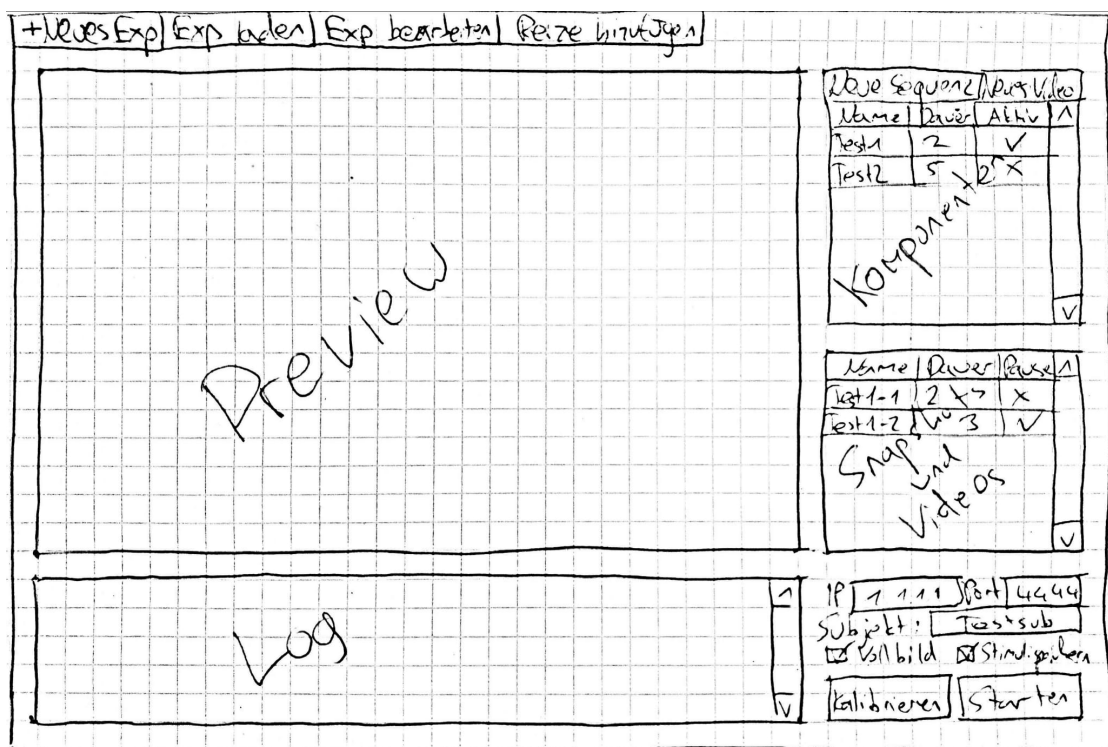


Abbildung 4.4: Ein Papierprototyp des Hauptfensters.

Über das in Abbildung 6.3 zu sehende Fenster können Name, Auflösung und Anzahl der Kalibrierungspunkte eingestellt werden. Das Experiment-bearbeiten-Fenster besitzt den selben Aufbau, jedoch mit einer anderen Beschriftung des Kontrollelements am unteren Rand.

Über das Experiment-laden-Fenster, welches in Abbildung 4.6 zu sehen ist, können Experimente über eine Combobox ausgewählt werden. Diese Combobox soll alle

A hand-drawn paper prototype of a window titled 'Neues-Experiment-Fenster'. It features a grid background. The form includes the following fields and controls:

- 'Experiment name:' followed by a text input field containing 'Test'.
- 'Auflösung:' followed by two stacked input fields: 'Höhe:' with '1920' and 'Breite:' with '510'.
- 'Kalibrierungspunkte:' followed by an input field containing '13' and a checked checkbox.
- A button labeled 'Erstellen' at the bottom right.

Abbildung 4.5: Ein Papierprototyp des Neues-Experiment-Fensters.

Experimente enthalten, die bisher erstellt wurden.

A hand-drawn paper prototype of a window titled 'Experiment-laden-Fenster'. It features a grid background. The form includes the following fields and controls:

- 'Experiment:' followed by a text input field containing 'Test' and a checked checkbox.
- A button labeled 'Laden' at the bottom right.

Abbildung 4.6: Ein Papierprototyp des Experiment-laden-Fensters.

Der Sequenzeditor besteht aus einer Slotsliste und einer Liste aller verfügbaren Bildreize. Von hier aus können neue Slots und neue Reize hinzugefügt werden. Hinzugefügte Reize sind in allen Sequenzen verfügbar, hinzugefügte Slots zählen exklusiv für die bearbeitete Sequenz. Slots können direkt in dieser Liste bearbeitet werden. Über Drag-and-Drop sollen Slots auf dem Preview-Feld des Hauptfensters (siehe Abbildung 4.4) abgelegt werden können. Dies soll dann die Position, auf welcher sie abgelegt wurden, in den Positionsfeldern übernehmen. Reize aus der Reizliste sollen ebenfalls per Drag-and-Drop in Slots hinzugefügt werden können. Es kann zusätzlich über die Aktiv-Checkbox ausgewählt werden, ob die Komponente im Experiment berücksichtigt werden soll.

Zuletzt ist in Abbildung 4.8 das Videoeditor-Fenster sichtbar. Dieses erlaubt eine Benennung und Bearbeitung des Videos sowie einige Editieroptionen. Außerdem kann hier ebenfalls durch eine Aktiv-Checkbox die Berücksichtigung der Komponente im Versuch eingestellt werden.

Sequenzname: <input type="text" value="Testsequenz"/>		Reize hinzufügen	
<div> <div>o</div> <div>o</div> <div>↓</div> </div>	Startzeit: <input type="text" value="1"/> Dauer: <input type="text" value="2"/> 1 Position x: <input type="text" value="100"/> y: <input type="text" value="500"/> Pausiert: <input checked="" type="checkbox"/> Ebene: <input type="text" value="1"/>	<div> <div>↓</div> <div>Smile</div> </div>	<div> <div>↓</div> <div>Frown</div> </div>
+ Slot hinzufügen ↗ ↘		Reize ↗ ↘	
		Aktiv <input checked="" type="checkbox"/>	

Abbildung 4.7: Ein Papierprototyp des Sequenzeditor-Fensters.

Videoname:	<input type="text" value="Testvideo"/>
Stimulus:	<input type="text" value="Test.mp4"/> 14
Startzeit:	<input type="text" value="1"/> Dauer: <input type="text" value="5"/>
Pausiert:	<input checked="" type="checkbox"/> Aktiv: <input checked="" type="checkbox"/>

Abbildung 4.8: Ein Papierprototyp des Videoeditor-Fensters.

5 Implementierung

Dieses Kapitel beschäftigt sich mit einigen Aspekten der Implementierung, die als wichtig erachtet werden. Zunächst wird in Kapitel 5.1 auf den Aufbau der Anwendung eingegangen. Danach werden einige spezifische Implementierungen abgehandelt: Wie die Generierung von Snapshots implementiert wurde, wird in Kapitel 5.2 erklärt, das Speichern und Laden von Experimenten wird in Kapitel 5.3 besprochen, das Logging-System anschließend in Kapitel 5.4 vorgestellt und zuletzt wird die Kommunikation der Anwendung mit dem SMI Experiment Center über die Remote-Control-Command-Language in Kapitel 5.5 veranschaulicht.

5.1 Architektur

In Abbildung 5.1 ist der Aufbau der Implementierung des Core-Projekts als simplifiziertes UML-Klassendiagramm dargestellt. Dies soll nur die Relationen der im Core-Projekt enthaltenen Klassen darstellen, es wurde deshalb auf Darstellung fast aller Felder, Eigenschaften und Methoden, als auch der Verdeutlichung von Relationen zu nativen oder externen Interfaces und Vererbungen verzichtet.

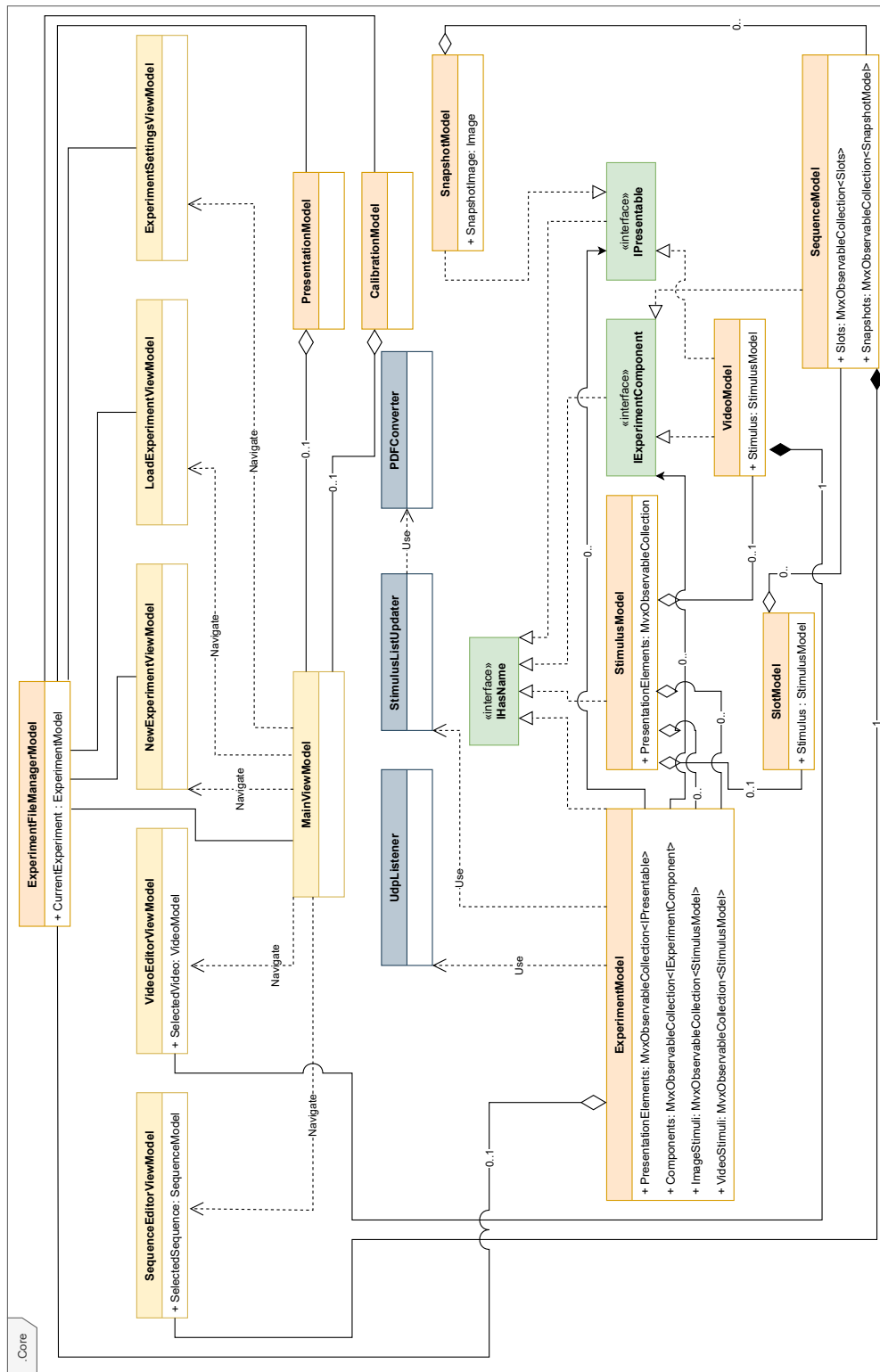


Abbildung 5.1: Eine simplifizierte Version der Core-Projekt-Architektur als UML-Klassendiagramm. Viewmodels werden hier gelb, Models orange, Interfaces grün und sonstige Hilfsklassen blau dargestellt.

Als Einstiegspunkt dient das `MainViewModel`, von welchem nur eine Instanz gleichzeitig existieren kann, welche über eine statische Eigenschaft `Instance` zugänglich ist. Dieses erlaubt die Navigation zu allen anderen ViewModels. `SequenceEditorViewModel` und `VideoEditorViewModel` halten Referenzen zu respektiv einem `SequenceModel` oder `VideoModel`, welche diese über den `MvvmCross-Navigationsservice` als Übergabeparameter erhalten. `ExperimentFileManagerModel` ist eine statische Klasse, welche das Laden und Speichern des derzeitig aktiven Experiments handhabt. Dieses Experiment existiert in Form eines `ExperimentModels`. Das derzeit geladene `ExperimentModel` ist über die Eigenschaft `CurrentExperiment` im `ExperimentFileManagerModel` abruf- und änderbar. Ein `ExperimentModel` hält vier `MvxObservableCollections`:

- **PresentationElements** ist eine Liste aus `IPresentables`. Wie in Abbildung 5.1 verdeutlicht, wird das Interface `IPresentable` von `VideoModel` und `SnapshotModel` implementiert. `IPresentable` wird Klassen implementiert, welche in der Versuchsdurchführung als Präsentationsobjekte dienen. `VideoModels` enthalten Videodaten und `SnapshotModels` enthalten Bilddaten. Die Zusammenfassung dieser Klassen unter einem Interface erlaubt das Abspielen der Daten in Reihenfolge durch das Verwenden einer Schleife, welche `PresentationElements` durchläuft. Dies findet in `PresentationModel` statt.
- **Components** ist eine Liste aus `IExperimentComponents`. Das Interface `IExperimentComponents` wird von `VideoModel` und `SequenceModel` implementiert. Dies vereinfacht die Population der `PresentationElements`-Liste in korrekter Reihenfolge. Wenn `Components` zu diesem Zweck durchlaufen wird, werden `VideoModels` direkt in `PresentationElements` übernommen, da diese sowohl `IExperimentComponent` als auch `IPresentable` implementieren. `SequenceModels` generieren `SnapshotModels`, wie in Kapitel 5.2 erklärt wird, welche an ihrer Stelle in die Liste hinzugefügt werden.
- **ImageStimuli** und **VideoStimuli** sind Listen aus `StimulusModels`. Diese Listen enthalten alle Reize, welche dem Experiment hinzugefügt wurden und sich im Stimuli-Ordner innerhalb des Experiment-Ordners befinden. Über das `ExperimentFileManagerModel` erhalten die zwei Editor-ViewModels Zugriff auf die Stimuli-Listen, von wo aus diese dann dem Nutzer über die GUI zugänglich gemacht werden. Die beiden Listen werden über einen `StimulusListUpdater` populiert. Innerhalb der `StimulusListUpdater`-Klasse wird eine `FileSystem-`

Watcher-Coroutine gestartet, welche den Stimuli-Ordner des Experiments überwacht. Jede in diesem Ordner hinzugefügte Datei feuert ein OnCreated-Event. Durch dieses werden vor der Population unerlaubte Stimuli aussortiert und PDF-Dateien über die PDFConverter-Klasse mit Hilfe von Ghostscript (siehe Kapitel 2.6.3) auf einzelne Seiten aufgeteilt und zu PNG-Bilddateien konvertiert.

Die Kalibrierung findet in der Klasse CalibrationModel statt. In gleicher Weise findet auch die Präsentation in der Klasse PresentationModel statt. Instanzen beider Klassen werden durch das MainViewModel erstellt und mit einer UDP-Verbindung zum SMI Experiment Center ausgestattet. Die Kommunikation zum SMI Experiment Center wird in Kapitel 5.5 behandelt.

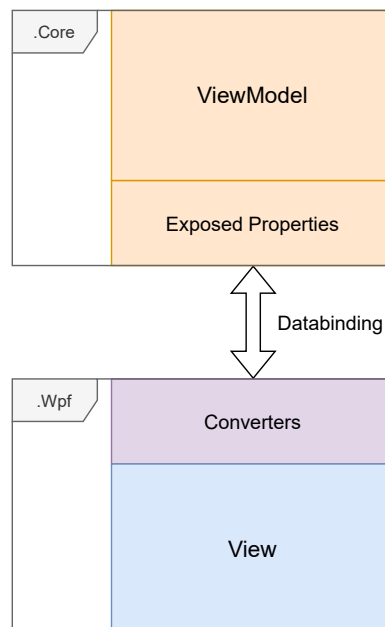


Abbildung 5.2: Öffentliche Eigenschaften eines ViewModels, welche an den verknüpften View gebunden sind, werden zuerst konvertiert.

Durch die Verwendung von MvvmCross ist es möglich, Updates der an den View gebundenen Eigenschaften durch das Erben der MvxNotifyPropertyChanged-Klasse zu verwirklichen. Dies kann entweder durch direkte Vererbung oder durch transitive Vererbung über MvxViewModel geschehen. Alle Viewmodels erben von MvxViewModel. Bei manchen Models ist ein Update des Views bei Veränderung einer Eigenschaft aber ebenfalls notwendig. In diesen Fällen wird von MvxNotifyPropertyChan-

ged geerbt. Dies erlaubt die Verwendung der Methoden `RaisePropertyChanged` und `SetProperty`, welche in Kapitel 2.6.2 behandelt werden.

Wenn einem View die Veränderung einer Eigenschaft mitgeteilt wird oder er neu an diese gebunden wird, dann kann es sein, dass diese zuerst durch einen Converter vorverarbeitet werden muss. Dies kann aus vielen Gründen notwendig sein, wie zum Beispiel der Konvertierung eines plattformunabhängigen Formats wie `System.Drawing.Imaging.Image` zum Windows-spezifischen Format `System.Windows.Media.Imaging.Image`, da dieses sonst nicht an WPF gebunden werden kann. Es ist aber auch möglich, dass es beispielsweise nur notwendig ist, einen Boolean invertiert an den View zu binden oder sicherzustellen, dass ein Wert nicht null ist. Generell sind Converter in diesem Projekt als eine weitere Schicht zwischen View und Viewmodel anzusehen, wie in Abbildung 5.2 verdeutlicht wird. Alle Converter erben vom nativen C#-Interface `IValueConverter`, welches die Methoden `Convert` und `ConvertBack` enthält. WPF erfordert, dass Converter dieses Interface implementieren und führt durch die Verwendung der dadurch garantierten Methoden Konvertierungen durch.

5.2 Snapshot-Generierung

Das Generieren von Snapshots aus Sequenzen erfolgt immer dann, wenn sich innerhalb eines `SequenceModel` eine Eigenschaft ändert, in der Regel durch eine Veränderung der Experimentdefinition durch den Versuchsleiter. Dieses Verhalten wurde so implementiert, dass es auch rekursiv für enthaltene `SlotModels` und in `SlotModels` enthaltene `StimulusModels` gilt. Hierfür wird von den besagten Models die Klasse `MvxNotifyPropertyChanged` geerbt.

`MvxNotifyPropertyChanged` implementiert das `INotifyPropertyChanged`-Interface. `INotifyPropertyChanged` ist ein in .NET bereits enthaltenes Interface, welches verwendet werden kann, um bei einer sich ändernden Eigenschaft ein `Property-Changed`-Event zu feuern, das alle zur Änderung relevanten Daten trägt. `MvxNotifyPropertyChanged` ist, wie in Kapitel 5.1 erklärt, auch für das projektübergreifende Updaten des Views zuständig und wird deshalb an dieser Stelle dem nativen Interface vorgezogen.

Wann immer ein PropertyChanged-Event gefeuert wird, wird eine Regenerierung der SnapshotModels veranlasst. Zu diesem Zweck werden alle SnapshotModels des betroffenen SequenceModels gelöscht und dann unter Verwendung der in Listing 5.1 ausgeführten Methode CreateSnapshotPlaylist() neu erstellt.

In dieser Methode werden zunächst durch die Methode GetRelevantTimestampsInSlotList() die Zeitstempel berechnet, in welchen Slots entweder beginnen oder enden. Nur diese Zeitstempel werden berücksichtigt, da sich zwischen zwei so kalkulierter Zeitstempel nie eine Veränderung an einem Snapshot ergeben kann.

Nachfolgend werden alle so gefundenen Zeitstempel in einer Schleife durchlaufen, wo dann überprüft wird, welche Slots an den gegebenen Zeitstempel aktiv sind. Außerdem wird überprüft, ob ein Slot hier pausiert, damit diese Information an den generierten Snapshot weitergeleitet werden kann. CreateSnapshot() bereitet dann die Daten für den Snapshot weiter vor und erstellt ein neues SnapshotModel-Objekt.

Listing 5.1: Diese Methode erstellt eine Liste aus SnapshotModels.

```
1 private List<SnapshotModel> CreateSnapshotPlaylist()  
2 {  
3     List<float> relevantTimestamps =  
4     GetRelevantTimestampsInSlotList();  
5     List<SnapshotModel> snapshots = new();  
6     for (int i = 0; i < relevantTimestamps.Count; i++)  
7     {  
8         bool PausesAtStart(SlotModel sl)  
9             => relevantTimestamps[i] == sl.StartTime && sl.Pauses;  
10        bool pauses = Slots.Any(sl => PausesAtStart(sl));  
11        List<SlotModel> activeSlots = Slots.Where(  
12            sl => relevantTimestamps[i] >= sl.StartTime &&  
13                relevantTimestamps[i] < sl.EndTime)  
14            .ToList();  
15        snapshots.Add(  
16            CreateSnapshot(activeSlots, i, relevantTimestamps, pauses));  
17    }  
18    return snapshots;  
19 }
```

Wenn ein SnapshotModel-Objekt angesprochen wird, generiert es ein Image-Objekt auf Basis der Slots, die in seiner Erstellung verwendet wurden. Image ist eine .NET-native Klasse und repräsentiert ein Bild, in diesem Fall ein Bitmap.

Um einen Snapshot zu generieren wird in der in Listing 5.2 gezeigten Schleife zunächst ein leeres Bitmap mit der im Experiment festgelegten Auflösung und Hintergrundfarbe erstellt. Dann werden alle Slots nach ihrem Layer sortiert durchiteriert, wobei deren verknüpfte Stimuli jeweils an die gewählte Stelle mit der gewählten Skalierung im Bild eingefügt werden.

Listing 5.2: In dieser Schleife werden die Bilder der Slots zu einem Snapshot zusammengefügt.

```
1 foreach (SlotModel sl in Slots.OrderBy(sl => sl.Layer).ToList())
2 {
3     Image stimulus = Image.FromFile(sl.StimulusPath);
4
5     Point location =
6         new(sl.XCoordinate - (int)(stimulus.Width * sl.Scale / 2),
7            sl.YCoordinate - (int)(stimulus.Height * sl.Scale / 2));
8     SizeF size =
9         new(stimulus.Width * sl.Scale, stimulus.Height * sl.Scale);
10    RectangleF slotParams = new(location, size);
11
12    graphic.DrawImage(stimulus, slotParams);
13    stimulus.Dispose();
14 }
```

Der fertige Snapshot wird im SnapshotModel selbst hinterlegt. Die Speicherung auf der Festplatte erfolgt erst bei der Ausführung des Experiments, falls dies vom Versuchsleiter erwünscht ist.

5.3 Speichern und Laden von Experimenten

Das derzeitige Experiment wird automatisch gespeichert, wenn ein anderes Experiment geladen oder die Applikation geschlossen wird. Experimenteinstellungen

werden gespeichert, indem das das Experiment repräsentierende `ExperimentModel` in der in Listing 5.3 gezeigten Methode mit Hilfe von JSON serialisiert und als JSON-Strings in eine Textdatei geschrieben wird. Hierfür wurde `Newtonsoft.Json` verwendet. Für jedes Experiment wird ein eigener Ordner erstellt, der das serialisierte `ExperimentModel` enthält.

Listing 5.3: Das Experiment wird serialisiert und als JSON-String abgespeichert.

```
1 public void SaveExperiment(string path)
2 {
3     if (CurrentExperiment == null) return;
4     string expJson =
5         JsonConvert.SerializeObject(CurrentExperiment, JsonSerializerSettings);
6     File.WriteAllText(path + @"\ExperimentSettings.json", expJson);
7 }
```

Die Serialisierung findet mit den in Listing 5.4 aufgeführten Einstellungen statt. Es ist erforderlich, dass Referenzen zwischen Objekten auch nach der Deserialisierung weiterhin bestehen. So würde ohne die Einstellung `PreserveReferencesHandling.All` nach einer Deserialisierung jede Referenz eines Objekts zu einer eigenständigen Kopie des referenzierten Objekts werden. Die Einstellung `TypeNameHandling.Auto` wurde gewählt, damit `Newtonsoft.Json` die Typzuweisung bei der Deserialisierung von als Interface oder abstrakter Klasse deklarierter Objekte automatisch handhabt. Zuletzt wird die Formatierung der erstellten Textdatei durch `Formatting.Indented` leserlicher gemacht. Dies bewirkt, dass der JSON-String nicht komplett in eine Zeile geschrieben wird, hat aber keinen funktionalen Einfluss.

Listing 5.4: Die für die Serialisierung verwendeten JSON-Einstellungen.

```
1 public JsonSerializerSettings JsonSerializerSettings = new()
2 {
3     PreserveReferencesHandling = PreserveReferencesHandling.All,
4     TypeNameHandling = TypeNameHandling.Auto,
5     Formatting = Formatting.Indented
6 };
```

Das Speichern von Snapshots und Videosnippets erfolgt, wenn ein Experiment durchgeführt wird, bei welchem der Versuchsleiter das Speichern angeordnet hat

und das Experiment seit der letzten Änderung in solcher Weise verändert wurde, dass die generierten Snapshots oder Videosnippets ebenfalls verändert würden.

Versuchslogs werden immer dann angelegt, wenn ein Versuch durchgeführt wird. Sie werden unter dem Namen des Versuchssubjekts in einem Unterordner des Experimentordners gespeichert und enthalten einen Verweis auf die Version der Snapshots und Videosnippets, die zu diesem Zeitpunkt verwendet wurde, so wie weitere Informationen über das Experiment.

Beim Laden eines Experiments wird ein serialisiertes ExperimentModel deserialisiert. Da sämtliche komplexe Objekte wie Bilder und Videos nicht mehr zwischengespeichert sind, müssen diese dann neu aus dem Stimuli-Ordner geladen werden. Dies erfordert die Regenerierung der Snapshots, wie in Kapitel 5.2 beschrieben wird, was eine kurze Ladezeit zur Folge hat.

5.4 Logging

Die Applikation muss für zwei verschiedene Zwecke Log-Dateien erstellen. Als Klassenbibliothek wurde dafür Serilog verwendet. Serilog wurde unter der Lizenz Apache 2.0 veröffentlicht und ist quelloffen [36]. Serilog bietet einfach zu editierende Log-Formatierung und verschiedene Levels von Log-Einträgen. In dieser Implementierung wurden die drei in Tabelle 5.1 beschriebenen Levels verwendet. Mit Serilog werden die Programmlog und die Versuchslog erstellt.

Level	Sichtbarkeit	Verwendung
Information	Build	Gibt Informationen an den Nutzer weiter und dient der Nutzererfahrung und Bedienbarkeit.
Debug	Debug	Gibt Informationen an Entwickler weiter und kann zum Debugging verwendet werden.
Error	Build	Gibt behandelte aber dennoch unerwartete Exceptions mit deren Stacktrace und einer kurzen Erklärung aus.

Tabelle 5.1: Die verschiedenen Logging-Levels, welche in der Applikation verwendet werden.

Programmlogs enthalten alle wichtigen Informationen und Ereignisse, die während der Ausführung des Programmes auftreten. Sie können jedes der in Tabelle 5.1 genannten Levels enthalten. Debug-Einträge werden aber lediglich in einer Debug-Version des Programmes geschrieben, da diese für normale Nutzer nicht von Bedeutung sind.

Die erstellten Programmlogs werden in einem Unterordner im Hauptverzeichnis der Applikation gespeichert und dort gehalten, bis der Nutzer diese manuell löscht. Ein automatisches Löschen der Programmlogs ist nicht implementiert und auch nicht erwünscht. Die Namenskonvention der Programmlog-Dateien ist log-TT-MM-JJJJ-SS-mm-ss.log, wobei T = Tag, M = Monat, J = Jahr, S = Stunde, m = Minute und s = Sekunde des Erstellzeitpunkts sind. Eine Programmlog wird immer dann neu erstellt, wenn das Programm gestartet wird.

Versuchslogs enthalten lediglich Nachrichten des Informations-Levels und einen Datenblock zu Beginn der Datei, der Start- und Endzeitpunkt des Versuchs, die verwendete Experimentversion, sowie den Experimentnamen und Subjektnamen beinhaltet. In Versuchslogs werden Eingaben von Versuchspersonen, die festgelegte Bedeutung dieser Eingaben, Zeitpunkte von Pausen und Start- und Endzeitpunkte von Stimuluspräsentationen festgehalten. Subjekt- und Experimentname sind hier redundant angegeben, denn Versuchslogs werden in einem Unterordner im Verzeichnis des Experiments für welches sie erstellt wurden gespeichert. Versuchslogs werden nach dem Subjekt, an welchem diese ausgeführt wurden, benannt. Die Redundanz ergibt allerdings Sinn, sollten die Dateien umbenannt oder verschoben werden. Auch Versuchslogs werden nicht automatisch gelöscht.

Log-Einträge werden direkt in die Log-Datei geschrieben. Dadurch kann auch bei einem unerwarteten Absturz des Programmes nachvollzogen werden, was vor dem Absturz geschah. Zur Anzeige der Log in der Benutzeroberfläche wird diese Textdatei dann ausgelesen.

5.5 Remote-Control-Command-Kommunikation

Die Kommunikation mit dem SMI Experiment Center erfolgt durch RCCs, sogenannte Remote Control Commands [22]. Ein RCC ist ein String bestehend aus

einem Identifikator und gegebenenfalls mehreren Parametern, die durch Leerstellen getrennt werden. Die von der Applikation versandten RCCs mit ihren jeweiligen Beschreibungen sind in Tabelle 5.2 aufgeführt. Die vom SMI Experiment Center als Antwort oder Anweisung an die Applikation versandten RCCs sind in Tabelle 5.3 aufgelistet.

Die Kommunikation der Applikation mit dem SMI Experiment Center zur Kalibrierung der Eye Tracking-Hardware wird in Abbildung 5.3 beschrieben. Durch das Versenden der RCCs ET_CSZ, ET_DEF, ET_EST und ET_CAL wird das SMI Experiment Center dazu aufgefordert, die Kalibrierung zu beginnen. Hierfür werden zunächst einzelne ET_PNT-RCCs mit jeweils x,y -Koordinaten und einem Index i versendet. Diese speichert die Applikation in einem Array aus Integer-Tupeln. Anschließend erfolgt die tatsächliche Kalibrierung. Das SMI Experiment Center sendet nach jedem erfolgreich kalibrierten Punkt ein ET_CHG-RCC, welches den nächsten zu kalibrierenden Punkt durch dessen Index angibt. Die Präsentation der Punkte erfolgt randomisiert, um prädiktives Verhalten der Versuchspersonen zu verhindern. Der Abschluss der Kalibrierung wird vom SMI Experiment Center durch ein ET_FIN-RCC signalisiert. Die Kalibrierung kann jederzeit frühzeitig abgebrochen werden. Dafür wird von der Applikation aus ein ET_BRK-RCC geschickt.

Nach einer erfolgreichen Kalibrierung können durch Reizpräsentationen Eye Tracking-Daten der Versuchsperson gesammelt werden. Die Präsentation von Reizen geschieht in einseitiger Kommunikation der Applikation zum SMI Experiment Center.

Die Speicherung der Eye Tracking-Daten selbst wird vom SMI Experiment Center vorgenommen. Mit dem RCC ET_FRM wird dafür ein Ausgabeformat der Blickpunkt-Daten festgelegt ("%PX %PY", beispielsweise "200 560"). Anschließend wird durch den RCC ET_STR der Datenstream der Eye Tracking-Hardware zum SMI Experiment Center aktiviert.

Anschließend werden in einer Schleife alle zu präsentierenden Reize abgespielt. Dafür wird in jeder Schleifeniteration zunächst eine neue Aufnahme der Eye Tracking-Daten durch den RCC ET_REC gestartet. Nach abgeschlossener Präsentation eines Snapshots oder Videosnippets wird die Aufnahme dann durch den RCC ET_STP angehalten und mit ET_SAV abgespeichert. Die Routine wird dann mit ET_CLR abgeschlossen, um den Bufferinhalt zu löschen und eine neue Iteration zu erlauben.

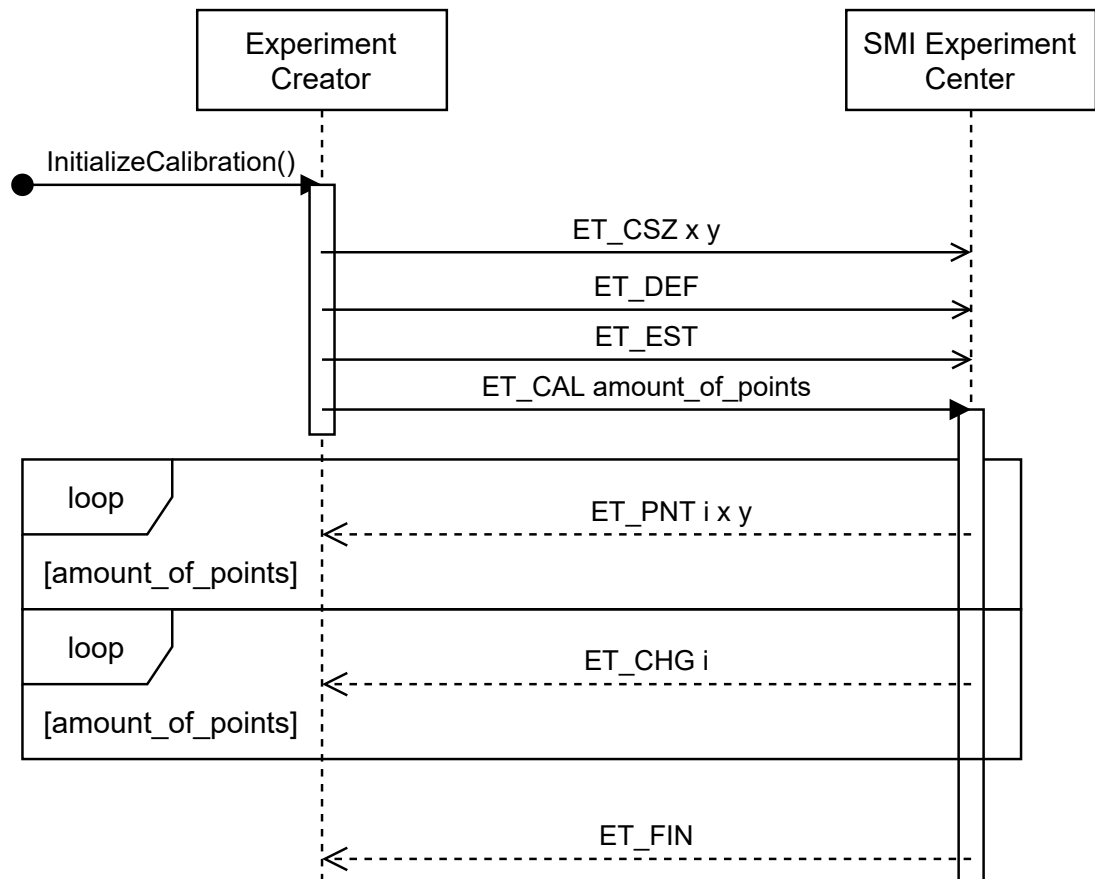


Abbildung 5.3: Der Ablauf der Kommunikation zwischen Applikation und SMI Experiment Center während der Kalibrierung.

Identifikator	Parameter	Verwendung
ET_CSZ	Integer, Integer	Setzt die Auflösung der Kalibrierungsfläche beim Start der Kalibrierung. Die Parameter beschreiben Höhe und Breite.
ET_DEF	-	Setzt die Positionen der Kalibrierungspunkte im SMI Experiment Center wieder auf ihre Standardpositionen zurück. Wird vorsichtshalber beim Start der Kalibrierung gesendet.
ET_EST	-	Beendet den Datenstream des SMI Experiment Centers, falls dieser offen ist. Wird vorsichtshalber beim Start der Kalibrierung gesendet.
ET_CAL	Integer	Ordnet eine neue Kalibrierung an. Der Parameter beschreibt die Anzahl der Kalibrierungspunkte.
ET_BRK	-	Bricht die Kalibrierung vorzeitig ab. Wird gesendet, wenn während der Kalibrierung der Escape-Knopf gedrückt wird.
ET_FRM	String	Legt das Format der Ausgabe des SMI Experiment Centers fest.
ET_STR	Integer	Startet den Datenstream der Eye Tracking-Hardware zum SMI Experiment Center. Der übergebene Parameter beschreibt die Bildwiederholungsfrequenz.
ET_REC	-	Startet die Aufnahme der vom Eye Tracking-Tower übermittelten Daten.
ET_STP	-	Bricht die Aufnahme der vom Eye Tracking-Tower übermittelten Daten ab.
ET_SAV	String	Veranlasst die Speicherung der erhaltenen Eye Tracking-Daten. Der im Parameter übergebene String beinhaltet den Pfad, unter welchem die zu speichernde Datei erstellt werden soll.
ET_CLR	-	Löscht den internen Datenbuffer. Wird nach ET_SAV gesendet, um einen klaren Abschluss des Experiments zu garantieren.

Tabelle 5.2: Eine List der RCCs, welche von der Applikation an das SMI Experiment Center versandt werden [22].

Identifikator	Parameter	Verwendung
ET_PNT	Integer, Integer, Integer	Gibt als Parameter Index, X-Koordinate und Y-Koordinate eines Kalibrierungspunktes weiter. Es werden nacheinander so viele ET_PNT-RCCs gesendet, wie in der Anfrage ET_CAL spezifiziert wurde.
ET_CHG	Integer	Ordnet die Änderung des Kalibrierungspunktes an. Der Parameter bezieht sich auf den Index, welcher im ET_PNT-RCC festgelegt wurde.
ET_FIN	-	Gibt das Ende der Kalibrierung an.

Tabelle 5.3: Eine Liste der RCCs, welche vom SMI Experiment Center an die Applikation versandt werden [22].

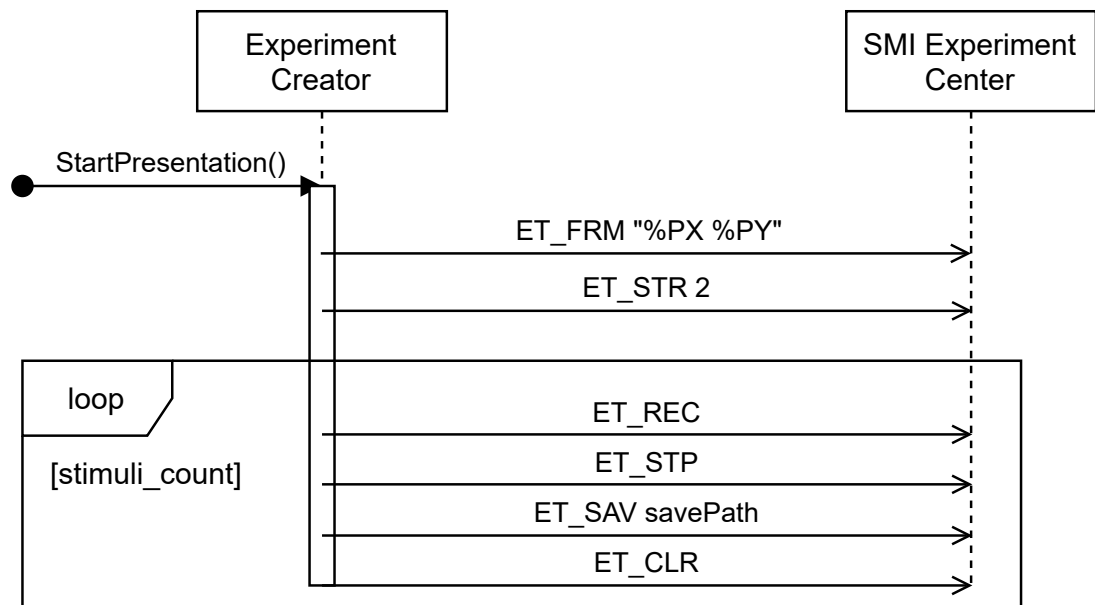


Abbildung 5.4: Der Ablauf der Kommunikation zwischen Applikation und SMI Experiment Center während der Kalibrierung.

6 Präsentation der Anwendung

In diesem Kapitel wird die fertiggestellte Anwendung vorgestellt und ihre Funktionsweise und Bedienung näher erklärt.

6.1 Hauptfenster

Nach dem Start der Anwendung wird dem Nutzer das Hauptfenster präsentiert. An diesem Punkt sind alle Kontrollelemente, die ohne ein geladenes Experiment nicht nutzbar sind, ausgegraut (vgl. Abbildung 6.1). Dies soll bewirken, dass der Nutzer organisch am linken oberen Bildschirmrand die Buttons zur Erstellung oder zum Laden eines Experiments findet. Sicherheitshalber wird diese Information auch noch in der Log, welche sich am unteren Bildschirmrand befindet, an den Nutzer übermittelt.

Nachdem über das in Kapitel 6.2 beschriebene Fenster ein Experiment erstellt oder über das in Kapitel 6.3 beschriebene Fenster ein Experiment geladen wurde, werden die restlichen Kontrollelemente freigegeben.

Vom Hauptfenster ausgehend kann der Nutzer Fenster zur Erstellung (Kapitel 6.2), zum Laden (Kapitel 6.3) oder zur Bearbeitung (Kapitel 6.4) von Experimenten öffnen. Neue Sequenzen und Videos können ebenfalls erstellt werden, welche dann respektiv das Sequenzeditor-Fenster (Kapitel 6.5) und das Videoeditor-Fenster (Kapitel 6.6) öffnen, damit diese sofort definiert werden können. Von hier aus können auch Reize in Bild-, Video- und PDF-Form importiert werden, die dann in allen Sequenzen und Videos des derzeitigen Experiments genutzt werden können.

Im rechten Bereich sind in Abbildung 6.2 zwei Listen vertikal zueinander orientiert erkennbar. Die obere Liste enthält alle Experimentkomponenten, also Videos

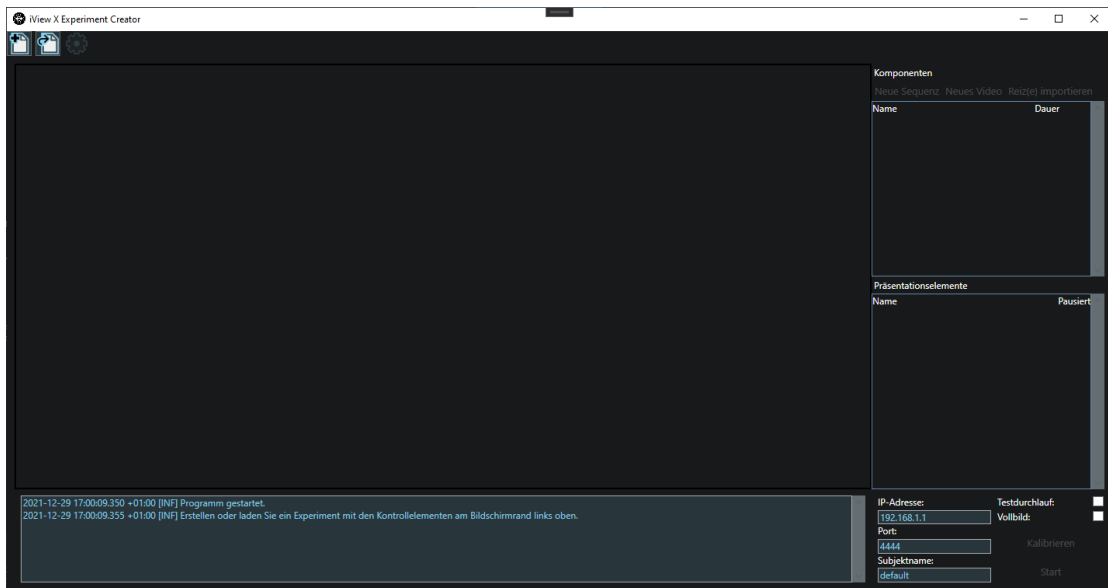


Abbildung 6.1: Das Hauptfenster unmittelbar nach dem Start der Anwendung: Kontrollelemente sind größtenteils noch deaktiviert.

und Sequenzen. Diese können dort durch das Bedienen der Pfeilelemente in ihrer Reihenfolge verändert werden, was eine Modifizierung ihrer Abspielreihenfolge zur Konsequenz hat. Durch einen Rechtsklick auf eine Komponente kann ein Kontextmenü aufgeklappt werden, welches das Duplizieren, Löschen und Bearbeiten der Komponente erlaubt. Die Bearbeitung kann überdies auch durch einen Doppelklick auf die Komponente erreicht werden. Beide Wege führen zur Öffnung eines Sequenz-/Videoeditor-Fensters.

In Abbildung 6.2 ist eine große Fläche mit dem Logo der Universität Ulm auf blauem Hintergrund sichtbar. Diese Vorschaufläche wird zur Präsentation eines Nicht-Vollbild-Experiments oder zur Einsicht von Snapshots und Videosnippets außerhalb der Präsentation verwendet.

Die Textfläche direkt unterhalb der Vorschaufläche ist die Anzeige der Programmlog, die dem Nutzer wichtige Ereignisse und Informationen anzeigt. Sollte ein behandelter Fehler auftreten, so wird dieser an gleicher Stelle ausgegeben.

Im rechten unteren Eck befindet sich eine Gruppe aus Kontrollen, welche zur Durchführung des Experiments mit einer Versuchsperson relevant sind. Hier können Verbindungsinformationen zum SMI Experiment Center und der Subjektname eingegeben werden. Weiter kann dort eingestellt werden, ob das Experiment im Vollbild an-

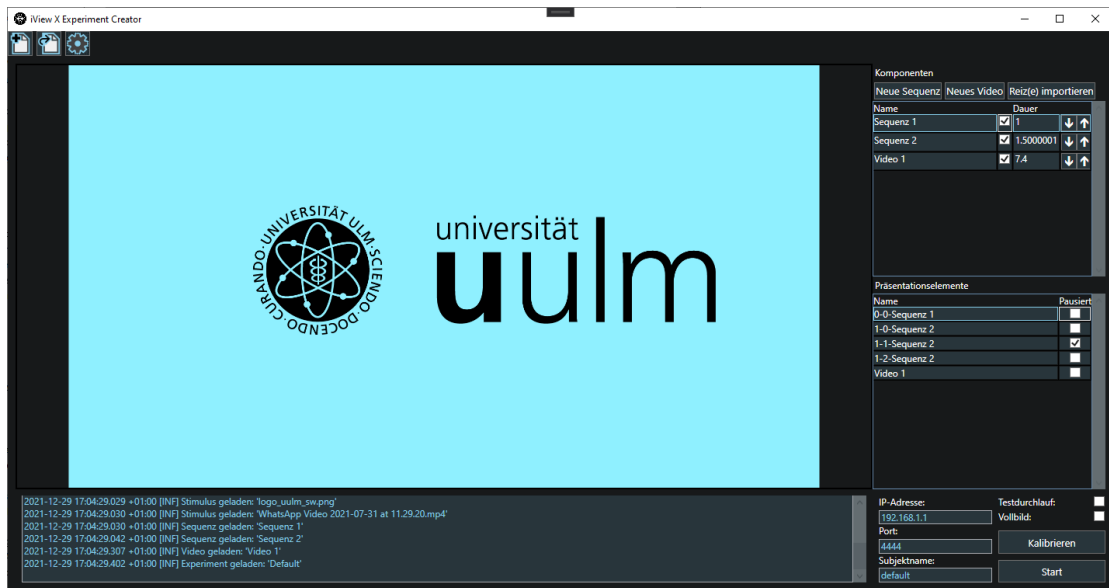


Abbildung 6.2: Das Hauptfenster nachdem ein Experiment mit einigen bereits definierten Komponenten geladen wurde.

gezeigt werden soll und ob generierte Stimuli, also Snapshots und Videosnippets, gespeichert werden sollen. Nachdem die Einstellungen nach Wunsch gewählt wurden, kann der Versuchsleiter eine Kalibrierung und nachfolgend das Experiment selbst durch die jeweiligen Kontrollen starten. Während der Kalibrierung wird, je nach Auswahl des Versuchsleiters, entweder im Vollbildmodus oder auf der Vorschaufläche eine Abfolge an Fixationskreuzen abgespielt.

Wenn das Experiment gestartet wird, werden die Präsentationselemente entweder im Vollbildmodus oder im Vorschaufenster ihrer Reihenfolge nach abgespielt. Der Rahmen des Vorschaufensters ändert sich zu rot. Er wird grün, wenn eine Nutzereingabe erfordert wird. Das Abspielen im Vorschaufenster wird für richtige Experimentdurchläufe nicht empfohlen und soll vor allem der Testung durch Versuchsleiter oder als Debughilfe für Entwickler dienen.

Um während eines laufenden Experiments Änderungen an der Experimentdefinition zu verhindern, wird eine Lock-out-Maßnahme [4] angewandt. Es werden alle Kontrollelemente deaktiviert und alle anderen Fenster der Anwendung geschlossen, bis der Durchlauf durch Betätigung der Escape-Taste abgebrochen oder erfolgreich abgeschlossen wurde.

6.2 Neues Experiment

Das Neues-Experiment-Fenster erlaubt die Erzeugung eines neuen Experiments. Hier kann der Name, die Auflösung der generierten Reize und die Anzahl der Kalibrierungspunkte eingestellt werden. Ein Kontrollelement, das in Abbildung 6.3 am unteren Fensterrand zentriert sichtbar ist, erlaubt die Erstellung des Experiments mit den eingestellten Parametern. Das neu erstellte Experiment wird dann als aktives Experiment gesetzt und alle Fenster des vorherigen Experiments werden geschlossen.

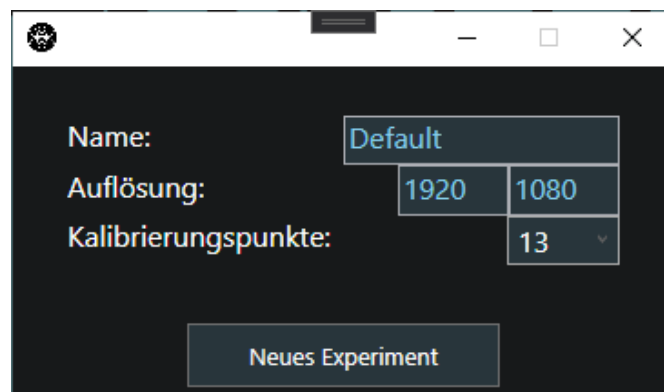


Abbildung 6.3: Das Fenster, das die Erstellung eines neuen Experiments erlaubt.

6.3 Experiment laden

Durch das Experiment-laden-Fenster (sichtbar in Abbildung 6.4) können Experimente geladen und gelöscht werden. Experimente können entweder durch einen Doppelklick, über das Kontextmenü mit einem Rechtsklick oder durch das zugehörige Kontrollelement geladen werden. Gelöscht werden Experimente entweder über das Kontextmenü oder das zugehörige Kontrollelement.

6.4 Experiment bearbeiten

Mit dem Experiment-bearbeiten-Fenster können die bei der Erstellung des Experiments angegebenen Parameter nachträglich verändert werden. So kann das Ex-



Abbildung 6.4: Durch dieses Fenster können Experimente geladen und gelöscht werden.

periment hier umbenannt werden und die Auflösung sowie die Anzahl der Kalibrierungspunkte wieder geändert werden.

Zusätzlich bietet dieses Fenster die Möglichkeit, die Hintergrundfarbe der generierten Snapshots zu setzen. Dies kann, wie in Abbildung 6.5 sichtbar, entweder durch das Angeben eines Hex-Codes der gewünschten Farbe oder der Auswahl der Farbe mit dem Farbwahl-Element geschehen. Diese Komponente wurde durch das quelloffene und unter der MIT-Lizenz veröffentlichte Nuget-Paket `PixiEditor.ColorPicker` hinzugefügt und ist nicht WPF-nativ [18].

Weiter ist hier auch die Aktivierung und Deaktivierung von Eingabemöglichkeiten möglich. Links, Rechts, Hoch und Runter beziehen sich hierbei auf die Pfeiltasten der Tastatur. Die Eingaben können mit einer Notiz versehen werden, die in der Log zusätzlich zur betätigten Eingabetaste notiert wird und für bessere Verständlichkeit beim Auswerten der Experimentdaten sorgen soll.



Abbildung 6.5: Durch dieses Fenster können Experimente bearbeitet werden.

6.5 Sequenzeditor

Die primären Komponenten des Sequenzeditor-Fensters sind eine Liste aus Slots (in Abbildung 6.6 links) und eine Liste der im Experiment hinzugefügten Bildreizen (in Abbildung 6.6 rechts). Unter Sequenzname kann der Sequenz ein eindeutiger Name zugewiesen werden. Weiter können durch Kontrollelemente am unteren Fensterrand neue Slots erstellt und Reize über den Windows-File-Dialog importiert werden.

Reize können durch Drag-and-Drop von der Reizliste auf die Bildflächen der Slots den Slots hinzugefügt werden. In den einzelnen Slots können Dauer und Startzeit mittels der Pfeil-Kontrollelemente an den Textboxen oder durch direkte Eingabe der gewünschten Werte in den Textboxen eingestellt werden. Die Position des Slots auf dem generierten Snapshot kann durch Angabe der Pixelkoordinaten bestimmt werden. Außerdem kann ein Slot durch Drag-and-Drop vom Sequenzeditor auf die Vorschaufläche im Hauptfenster gezogen werden, um die Position zu be-

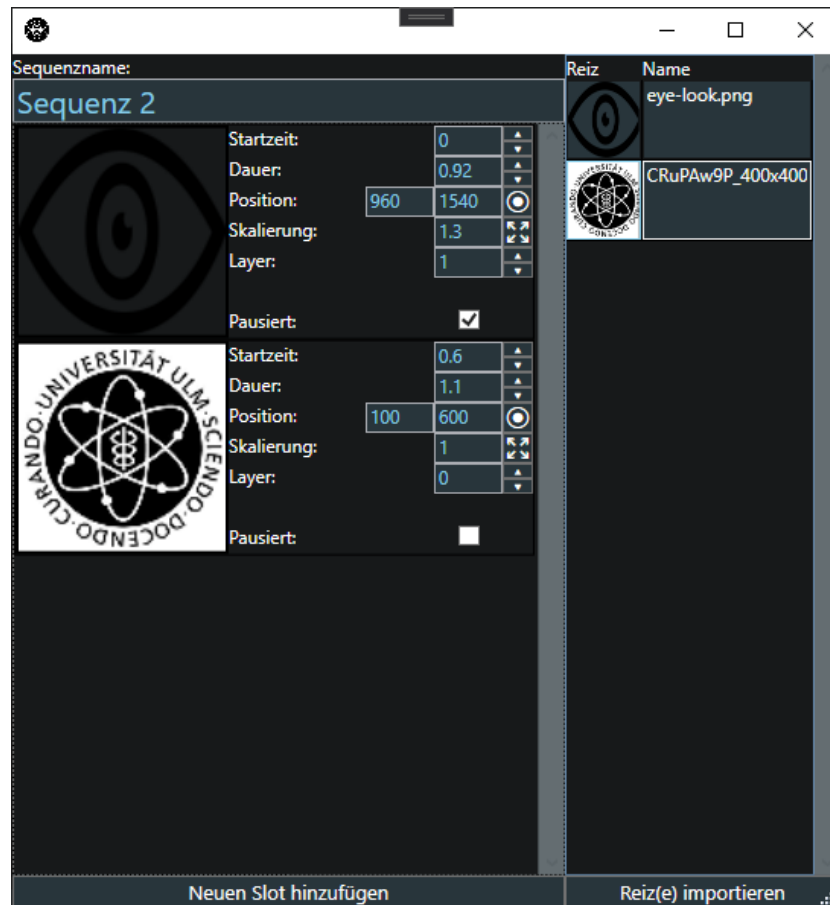


Abbildung 6.6: Durch dieses Fenster können Sequenzen bearbeitet werden.

stimmen. Zudem kann die Position des Slots durch das Drücken des Zentrieren-Kontrollelements zentriert werden. Die Skalierung des Slots bestimmt, mit welchem Koeffizienten die Größe des Slots multipliziert wird. Eine Skalierung von 1 ist die Originalgröße des enthaltenen Reizes. Die Skalierung kann durch das Drücken des Strecken-Kontrollelements an die Auflösung des Experiments angepasst werden. Durch die Checkbox zur Pausierung kann eine Pausierung der Präsentation zur Eingabeaufforderung einer Versuchsperson bei Start oder Ende des Slots erwirkt werden. Der Layer beschreibt die Ebene, auf welcher der Slot gezeichnet wird. Die Ebene beeinflusst, welcher Slot bei einer Überlappung zuerst gezeichnet wird. Niedrigere Layer werden zuerst gezeichnet. Das bedeutet, dass niedrigere Layer von höheren Layern überzeichnet werden können.

6.6 Videoeditor

Im Videoeditor-Fenster kann ein Videoreiz gewählt werden. Durch Angabe von Startzeitstempel und Dauer kann ein Ausschnitt des Videos in der Präsentation gezeigt werden. Diese können, wie in Abbildung 6.7 zu sehen ist, entweder durch direkte Eingabe oder die Pfeil-Kontrollelemente eingestellt werden. Außerdem kann auch hier eine Pausierung des Videos verlangt werden, um eine Reaktion der Versuchsperson zu erlangen. Diese Pausierung erfolgt am Anfang des Videos. Bei Bedarf ist es möglich, die Lautstärke des Videos zu regulieren.

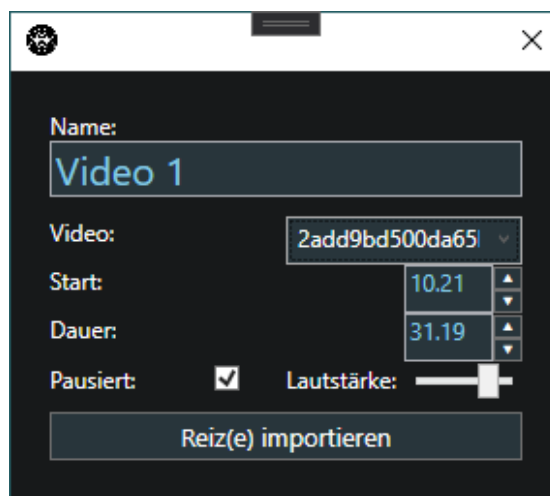


Abbildung 6.7: Im Videoeditor-Fenster können Anpassungen an Dauer und Startzeitpunkt des Videos vorgenommen werden.

7 Anforderungsabgleich

In diesem Kapitel wird beschrieben, inwiefern die implementierte Anwendung den im Kapitel 3.1.1 analysierten Anforderungen gerecht wird.

7.1 Funktionale Anforderungen

ID	Name	Beschreibung
1	Experiment erstellen	Erfüllt. Der Versuchsleiter kann über die Benutzeroberfläche ein Fenster zum Erstellen eines neuen Experiments öffnen. Näher beschrieben wird dies in Kapitel 6.2.

ID	Name	Beschreibung
2	Experiment bearbeiten	Erfüllt. Der Versuchsleiter kann über die Benutzeroberfläche ein Fenster zur Bearbeitung des Experiments öffnen. Näher beschrieben wird dies in Kapitel 6.4.

ID	Name	Beschreibung
3	Experiment speichern	Erfüllt. Das Experiment wird automatisch gespeichert, wenn ein neues Experiment erstellt, ein vorhandenes Experiment geladen oder die Anwendung geschlossen wird.

ID	Name	Beschreibung
4	Experiment laden	Erfüllt. Der Versuchsleiter kann über die Benutzeroberfläche ein Fenster zum Laden eines Experiments öffnen. Näher beschrieben wird dies in Kapitel 6.3.

7 Anforderungsabgleich

ID	Name	Beschreibung
5	Reize laden	Teilweise erfüllt. Reize können in der Applikation an zwei Stellen geladen werden: Im Sequenzeditor ist es möglich, Bildreize und PDFs zu laden und im Hauptfenster können zusätzlich Videoreize geladen werden. Bildreize können angezeigt werden, indem sie in Snapshots hinzugefügt werden. Videoreize können außerhalb der Präsentation noch nicht angezeigt werden. Es gibt noch kein dediziertes Preview für rohes Reizmaterial.

ID	Name	Beschreibung
6	Experimentkomponenten	Erfüllt. Wie in Kapitel 6.1 beschrieben, können im Hauptfenster Sequenzen und Videos als Experimentkomponenten erstellt werden.

ID	Name	Beschreibung
7	Sequenzen bearbeiten	Erfüllt. Sequenzen bestehen aus Slots, für welche Größenskalierung, Position, Start- und Endzeit sowie Reihenfolge im Bild bestimmt werden kann. Diese können nach Belieben hinzugefügt, gelöscht und dupliziert werden.

ID	Name	Beschreibung
8	Experimentkomponenten	Erfüllt. Im Hauptfenster können wie in Kapitel 6.1 sowohl Komponenten der Art Video als auch der Art Sequenz erstellt werden.

ID	Name	Beschreibung
9	Experimentkomponenten duplizieren	Erfüllt. Über das Kontextmenü können Komponenten dupliziert werden.

7 Anforderungsabgleich

ID	Name	Beschreibung
10	Experiment-komponenten anordnen	Erfüllt. Komponenten können mit den Pfeil-Steuerungselementen in ihrer Reihenfolge verändert werden.

ID	Name	Beschreibung
11	Experiment-komponenten aktivieren/-deaktivieren	Erfüllt. Sowohl Sequenzen als auch Videos können in ihren respektiven Editoren aktiviert/deaktiviert werden.

ID	Name	Beschreibung
12	Experiment-komponenten löschen	Erfüllt. Experimentkomponenten können über das Kontextmenü gelöscht werden.

ID	Name	Beschreibung
13	Snapshots erstellen	Erfüllt. Snapshots werden wie in Kapitel 5.2 beschrieben korrekt generiert.

ID	Name	Beschreibung
14	Snapshots speichern	Erfüllt. Der Versuchsleiter kann vor der Durchführung eines Experiments auswählen, ob Snapshots und Videosnippets gespeichert werden sollen. Sie werden nur dann gespeichert, wenn das Experiment seit der letzten Speicherung in solcher Weise verändert wurde, dass sich die Snapshots/Videosnippets ebenfalls geändert haben könnten.

7 Anforderungsabgleich

ID	Name	Beschreibung
15	Snapshots anzeigen	Erfüllt. Snapshots können im Hauptfenster aus der Liste der generierten Snapshots ausgewählt und im Previewfenster angezeigt werden.

ID	Name	Beschreibung
16	Netzwerkverbindung	Erfüllt. Der Versuchsleiter kann Verbindungsinformationen angeben und sich erfolgreich mit dem SMI Experiment Center verbinden.

ID	Name	Beschreibung
17	Kalibrierung	Erfüllt. Die Anzeige der Kalibrierungspunkte und die Kommunikation mit dem SMI Experiment Center während der Kalibrierung wurden korrekt implementiert.

ID	Name	Beschreibung
18	Experiment durchführen	Erfüllt. Die Durchführung von Experimenten wurde einige Male erfolgreich an der Eye Tracking-Hardware getestet. Alle RCCs werden korrekt versandt.

ID	Name	Beschreibung
19	Versuchspersonen-Input	Erfüllt. In den Experimenteinstellungen können Eingabeoptionen für Versuchspersonen aktiviert und deaktiviert werden. Diesen kann dort auch eine Bedeutung zugewiesen werden, die dann später zur besseren Interpretation der Daten genutzt werden kann. Eingabe, Bedeutung und Snapshots/Videosnippet, zu denen eine Eingabe getätigt wurde, werden in der Versuchsdurchführungslog im Experimentenordner gespeichert.

7.2 Nicht-funktionale Anforderungen

ID	Name	Beschreibung
1	Modifizierbarkeit	Erfüllt. Die Anwendung wurde dokumentiert und die meisten Funktionen wurden so einfach wie möglich gehalten. Zur Erstellung der Dokumentation wurde das Programm Doxygen verwendet [10].

ID	Name	Beschreibung
2	Analysierbarkeit	Erfüllt. Fehler werden in der Log informativ ausgegeben und Debug-Ausgaben sind über die Log ebenfalls möglich. Diese können Dank Speicherung auf der Festplatte auch nach Beenden des Programms noch ausgewertet und auch mit anderen Programmdurchläufen verglichen werden.

ID	Name	Beschreibung
3	Portierbarkeit	Erfüllt. Durch die Verwendung des MVVM-Entwurfsmusters und des MVVMCross-Frameworks muss für die Portierung auf eine der in Kapitel 2.6.2 genannten Plattformen lediglich ein View entwickelt werden. Das Core-Projekt kann so übernommen werden.

ID	Name	Beschreibung
4	Stabilität	Teilweise erfüllt. Es wurden keine Unit-Tests geschrieben. Allerdings wurde die Applikation ausgiebig getestet und es wurde bei der Implementierung achtgegeben, auch scheinbar unmögliche Fehlerzustände abzufangen.

7 Anforderungsabgleich

ID	Name	Beschreibung
5	Verständlichkeit	Nicht erfüllt. Es haben keine größeren Tests mit projektfremden Personen stattgefunden. Dies kann also nicht beurteilt werden.

ID	Name	Beschreibung
6	Erlernbarkeit	Teilweise erfüllt. Es haben keine größeren Tests mit projektfremden Personen stattgefunden. Dies kann also nicht beurteilt werden. Allerdings werden alle Elemente der Anwendung durch Tooltips erklärt.

ID	Name	Beschreibung
7	Bedienbarkeit	Erfüllt. Es wurde darauf geachtet, dass die Steuerung mehrere Wege zulässt, um zum selben Ergebnis zu gelangen. Die Navigationstiefe wurde sehr gering gehalten, da alle Fenster vom Hauptfenster aus angesprochen werden. Diese beiden Faktoren tragen dazu bei, dass alle Funktionen der Anwendung schnell und einfach erreicht werden können.

8 Zusammenfassung

Es wurde eine kurze Einführung in die Thematik gegeben, die die Notwendigkeit dieser Arbeit verdeutlicht und mit einigen grob definierten Anforderungen an die zu entwickelnde Applikation abgeschlossen wurde. Danach wurden die in dieser Arbeit verwendeten Techniken und Technologien erklärt. Anschließend wurden die funktionalen und qualitativen Anforderungen formal definiert sowie das Konzept der Funktionsweise und Benutzeroberfläche vorgestellt. Im Anschluss wurden die Architektur und einige wichtige Aspekte der Implementierung behandelt. Zuletzt wurde dann die fertiggestellte Anwendung präsentiert sowie ein Abgleich mit den gesetzten Anforderungen vollzogen.

8.1 Ausblick

Die Applikation wurde mit Portierbarkeit im Sinn entwickelt. Es können Dank des verwendeten Entwurfsmusters und des Verzichts auf Windows-spezifische Funktionen im Core-Projekt problemlos neue Views für die in Kapitel 2.6.2 angesprochenen Plattformen entwickelt werden. Dies würde einer größeren Anzahl an Forschern und Studenten die Nutzung der Applikation ermöglichen. Eine Android- oder iOS-Applikation könnte sogar das Definieren von Experimenten am Smartphone oder Tablet erlauben, wozu dann lediglich ein Bildschirm an der Universität zur Durchführung des Experiments gefunden werden müsste.

Die Erweiterbarkeit wurde während der Entwicklung ebenfalls beachtet. Das Core-Projekt kann um viele Funktionen erweitert werden.

So würden mehr Optionen in Slots zu mehr Kontrolle über die generierten Snapshots führen und weniger Vorverarbeitung der Reize erfordern. Solche Optionen

könnten zum Beispiel Colour-Tint, Rotation oder Cropping-Tools sein. Eine erweiterte Funktionsvielfalt könnte im selben Sinne auch im Videoeditor umgesetzt werden, da dieser bisweilen nur das Schneiden von Videos erlaubt. Des Weiteren wäre es vorteilhaft, wenn die Position der Pause bei Videos näher bestimmt werden könnte oder diese sogar mehrfach pausiert werden könnten.

Die Art und Weise wie Experimente momentan gespeichert werden erlaubt den Austausch von Experimenten unter verschiedenen Systemen leider noch nicht ohne direkten Eingriff in die JSON-Datei. Dies kann ebenfalls zukünftig noch behoben werden, so dass Experimentdaten immer mit zum Anwendungsverzeichnis relativem Pfad geladen werden.

Abschließend wäre eine willkommene Addition auch noch eine Möglichkeit, die Versuchsdaten direkt in der Anwendung auswerten zu können. Es existiert ein RCC, mit dem diese Daten vom SMI Experiment Center nach einem Versuch abgerufen werden können. Eine computergesteuerte Analyse und Visualisierung der gewonnenen Daten wäre für Versuchsleiter sehr hilfreich.

Literatur

- [1] *A Tour of C# - C# Guide | Microsoft Docs*. <https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>. (Accessed on 10/14/2021).
- [2] Gabriel Aberg und Jessica Chang. „Applying Cognitive Science Research in Graphical User Interface (GUI)“. In: *Umea Institute of Design* (2005), S. 23–28.
- [3] *AydinAdn/MediaToolkit: A .NET library to convert and process all your video & audio files*. <https://github.com/AydinAdn/MediaToolkit>. (Accessed on 11/27/2021).
- [4] Martin Baumann. *Skript: Arbeits- und Organisationspsychologie B - Menschliche Fehler und Zuverlässigkeit II*. Universität Ulm, Juni 2020.
- [5] MP Bryden. „Tachistoscopic recognition of non-alphabetical material.“ In: *Canadian Journal of Psychology/Revue canadienne de psychologie* 14.2 (1960), S. 78.
- [6] Cathy Buquet u. a. „Photo-oculography: A new method for eye movements study, interest in ophthalmological and extra pyramidal neurological diseases“. In: *1992 14th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. Bd. 4. 1992, S. 1555–1556. DOI: 10.1109/IEMBS.1992.5761922.
- [7] *Create UIs with Visual Studio XAML Designer - Visual Studio (Windows) | Microsoft Docs*. <https://docs.microsoft.com/en-us/visualstudio/xaml-tools/creating-a-ui-by-using-xaml-designer-in-visual-studio?view=vs-2022>. (Accessed on 12/01/2021).
- [8] *Data binding | MvvmCross*. <https://www.mvvmcross.com/documentation/fundamentals/data-binding>. (Accessed on 12/23/2021).

- [9] *Download .NET 5.0 (Linux, macOS, and Windows)*. <https://dotnet.microsoft.com/download/dotnet/5.0>. (Accessed on 10/14/2021).
- [10] *Doxygen: Doxygen*. <https://www.doxygen.nl/index.html>. (Accessed on 12/23/2021).
- [11] Andrew T Duchowski und Andrew T Duchowski. *Eye tracking methodology: Theory and practice*. Springer, 2017.
- [12] *ECMA-404 - Ecma International*. <https://www.ecma-international.org/publications-and-standards/standards/ecma-404/>. (Accessed on 10/29/2021).
- [13] *Extensions for Visual Studio family of products | Visual Studio Marketplace*. <https://marketplace.visualstudio.com/vs>. (Accessed on 12/01/2021).
- [14] *FFmpeg*. <http://ffmpeg.org/>. (Accessed on 11/27/2021).
- [15] *GNU Affero General Public License, Version 3 - GNU-Projekt - Free Software Foundation*. <https://www.gnu.org/licenses/agpl-3.0.html>. (Accessed on 11/20/2021).
- [16] *GNU Lesser General Public License, Version 2.1 - GNU-Projekt - Free Software Foundation*. <https://www.gnu.org/licenses/old-licenses/lgpl-2.1.html>. (Accessed on 11/27/2021).
- [17] *Ghostscript*. <https://www.ghostscript.com/>. (Accessed on 11/20/2021).
- [18] *GitHub - PixiEditor/ColorPicker: Customizable Color Picker control for WPF*. <https://github.com/PixiEditor/ColorPicker>. (Accessed on 12/23/2021).
- [19] Katarzyna Harezlak und Pawel Kasprowski. „Application of eye tracking in medicine: A survey, research issues and challenges“. In: *Computerized Medical Imaging and Graphics* 65 (2018).
- [20] Ignace Hooze, Kenneth Holmqvist und Marcus Nyström. „The pupil is faster than the corneal reflection (CR): Are video based pupil-CR eye trackers suitable for studying detailed dynamics of eye movements?“ In: *Vision research* 128 (2016), S. 6–18.
- [21] *Install and manage NuGet packages in Visual Studio | Microsoft Docs*. <https://docs.microsoft.com/en-us/nuget/consume-packages/install-use-packages-visual-studio>. (Accessed on 12/05/2021).

- [22] SensoMotoric Instruments. *iView X System Manual*. SMI, 2014.
- [23] JSON. <https://www.json.org/json-de.html>. (Accessed on 10/29/2021).
- [24] Robert Jacob und Keith Karn. „Eye Tracking in Human-Computer Interaction and Usability Research: Ready to Deliver the Promises“. In: Bd. 2. Jan. 2003. ISBN: 9780444510204. DOI: 10.1016/B978-044451020-4/50031-1.
- [25] *Json.NET - Newtonsoft*. <https://www.newtonsoft.com/json>. (Accessed on 12/23/2021).
- [26] Doreen Kimura. „Dual functional asymmetry of the brain in visual perception“. In: *Neuropsychologia* 4.3 (1966), S. 275–285.
- [27] *MediaToolkit/LICENSE.md at master · AydinAdn/MediaToolkit*. <https://github.com/AydinAdn/MediaToolkit/blob/master/LICENSE.md>. (Accessed on 11/27/2021).
- [28] Mortimer Mishkin und Donald G Forgays. „Word recognition as a function of retinal locus.“ In: *Journal of experimental psychology* 43.1 (1952), S. 43.
- [29] Morris Moscovitch. „Information processing and the cerebral hemispheres“. In: *Neuropsychology*. Springer, 1979, S. 379–446.
- [30] *MvvmCross | MvvmCross is a convention based MVVM framework for Xamarin and Windows, with strong community support, filled to the brim with useful features*. <https://www.mvvmcross.com/>. (Accessed on 12/23/2021).
- [31] *MvvmCross Overview | MvvmCross*. <https://www.mvvmcross.com/documentation/getting-started/mvvmcross-overview>. (Accessed on 12/23/2021).
- [32] *Navigation | MvvmCross*. <https://www.mvvmcross.com/documentation/fundamentals/navigation>. (Accessed on 12/23/2021).
- [33] *NuGet Gallery | Home*. <https://www.nuget.org/>. (Accessed on 12/05/2021).
- [34] Monika Płużyczka. „The first hundred years: A history of eye tracking as a research method“. In: *Applied Linguistics Papers* 25/4 (2018), S. 101–116.
- [35] Keith Rayner. „Eye movements in reading and information processing: 20 years of research.“ In: *Psychological bulletin* 124.3 (1998).
- [36] *Serilog — simple .NET logging with fully-structured events*. <https://serilog.net/>. (Accessed on 11/12/2021).

- [37] Erik Sorensen und M Mikaillesc. „Model-view-ViewModel (MVVM) design pattern using Windows Presentation Foundation (WPF) technology“. In: *Mega-Byte Journal* 9.4 (2010), S. 1–19.
- [38] *Sprachen - Visual Studio*. <https://visualstudio.microsoft.com/de/vs/features/web/languages/>. (Accessed on 12/01/2021).
- [39] *The Git experience in Visual Studio | Microsoft Docs*. <https://docs.microsoft.com/en-us/visualstudio/version-control/git-with-visual-studio?view=vs-2022>. (Accessed on 12/01/2021).
- [40] *Use Class Designer - Visual Studio (Windows) | Microsoft Docs*. <https://docs.microsoft.com/en-us/visualstudio/ide/class-designer/designing-and-viewing-classes-and-types?view=vs-2022>. (Accessed on 12/01/2021).
- [41] *Visual Studio: IDE und Code-Editor für Softwareentwickler und -teams*. <https://visualstudio.microsoft.com/de/>. (Accessed on 12/01/2021).
- [42] *What is .NET? An open-source developer platform*. <https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet>. (Accessed on 10/14/2021).
- [43] *What is .NET Framework? A software development framework*. <https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet-framework>. (Accessed on 12/23/2021).
- [44] *What is Windows Presentation Foundation - WPF .NET | Microsoft Docs*. <https://docs.microsoft.com/en-us/dotnet/desktop/wpf/overview/?view=netdesktop-6.0>. (Accessed on 12/21/2021).
- [45] *Xamarin | Open-source mobile app platform for .NET*. <https://dotnet.microsoft.com/en-us/apps/xamarin>. (Accessed on 12/23/2021).
- [46] Xiaojiao Xie u. a. „Study on the effects of display color mode and luminance contrast on visual fatigue“. In: *IEEE Access* 9 (2021), S. 35915–35923.
- [47] Chee Kit Yee u. a. „GUI design based on cognitive psychology: theoretical, empirical and practical approaches“. In: *2012 8th International Conference on Computing Technology and Information Management (NCM and ICNIT)*. Bd. 2. IEEE. 2012, S. 836–841.

- [48] Chee Yee u. a. „GUI design based on cognitive psychology: Theoretical, empirical and practical approaches“. In: Bd. 2. Jan. 2012, S. 836–841. ISBN: 978-1-4673-0893-9.
- [49] *jhabjan/Ghostscript.NET: Ghostscript.NET - managed wrapper around the Ghostscript library (32-bit & 64-bit)*. <https://github.com/jhabjan/Ghostscript.NET>. (Accessed on 11/20/2021).

Name: Sandro Allgaier

Matrikelnummer: 987194

Erklärung

Ich erkläre, dass ich die Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den 30.12.2021



Sandro Allgaier