Institut für Visualisierung und Interaktive Systeme
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart
Germany

Studienarbeit Nr. 2452

Entwicklung einer Präsentation-Software für große, hochauflösende Displays

Eugen Mannweiler

Studiengang:	Diplom Informatik
Prüfer:	Prof. Dr. Thomas Ertl
Betreuer:	Dipl.-Inf. Christoph Müller
begonnen am:	13.01.2014
beendet am:	15.07.2014
CR-Klassifikation:	C.2.4, H.5.2, I.3.4, I.7.1, I.7.2

Kurzfassung

In dieser Studienarbeit wird eine Präsentationssoftware für sogenannte Powerwalls entwickelt. Die erste Komponente ist ein Editor, der speziell auf hochauflösende Powerwalls zugeschnitten ist und auf den herkömmlichen Desktop-Rechnern laufen soll. Die von Microsoft angebotene Präsentationssoftware PowerPoint soll bei der Entwicklung des Editors als Vorbild dienen. Die zweite Aufgabe dieser Arbeit ist die Entwicklung einer Kommunikationskomponente, weil die Powerwall mit mehreren Rechnern betrieben wird. Diese Komponente wird auf einem Client-Server-Modell aufbauen. Der Server soll in der Lage sein, Anfragen mehrerer Clients gleichzeitig bearbeiten zu können.

Inhaltverzeichnis

Abbildungsverzeichnis

Listingverzeichnis

1 Einleitung

1.1 Hintergrund

Mit dem technologischen Fortschritt finden die großen hochauflösenden Displays, die sogenannten Powerwalls, in immer mehr Branchen praktischen Anwendung. Sie werden unter anderem in der Autoindustrie zur Entwicklung neuer Autodesigns oder in Kontrollräumen zur Überwachung von Kraftwerken eingesetzt. Die höhere Auflösung und die größeren physischen Ausmaße sind zwei gemeinsame Eigenschaften solcher hochauflösender Displays. Heutzutage wird eine Reihe von vernetzten Computern dazu verwendet, um das Gesamtbild zu rendern und via mehrerer Projektoren, die jeweils mit einem Computer verbunden sind und mit Bilddaten versorgt werden, anzuzeigen [1] [2]. Dadurch ist es meist nicht möglich, vorhandene Anwendungen, wie PowerPoint, direkt auf diesen Displays zu verwenden. Die benötigte Anwendungssoftware wird meist speziell für eine Anlage entwickelt, um von der hohen Auflösung zu profitieren. Die erzeugten Ausgaben werden von vorhandener Software bestenfalls auf diese Auflösung hochskaliert. Dies verhindert jedoch die effektive Nutzung der zur Verfügung stehenden Pixel und resultiert in einem unschönen, pixeligen Bild.

In Abbildung 1.1.1 ist eine Powerwall zusehen. Sie wird mit Hilfe von 5 Projektoren, die jeweils mit einem Rechner verbunden sind, dargestellt.

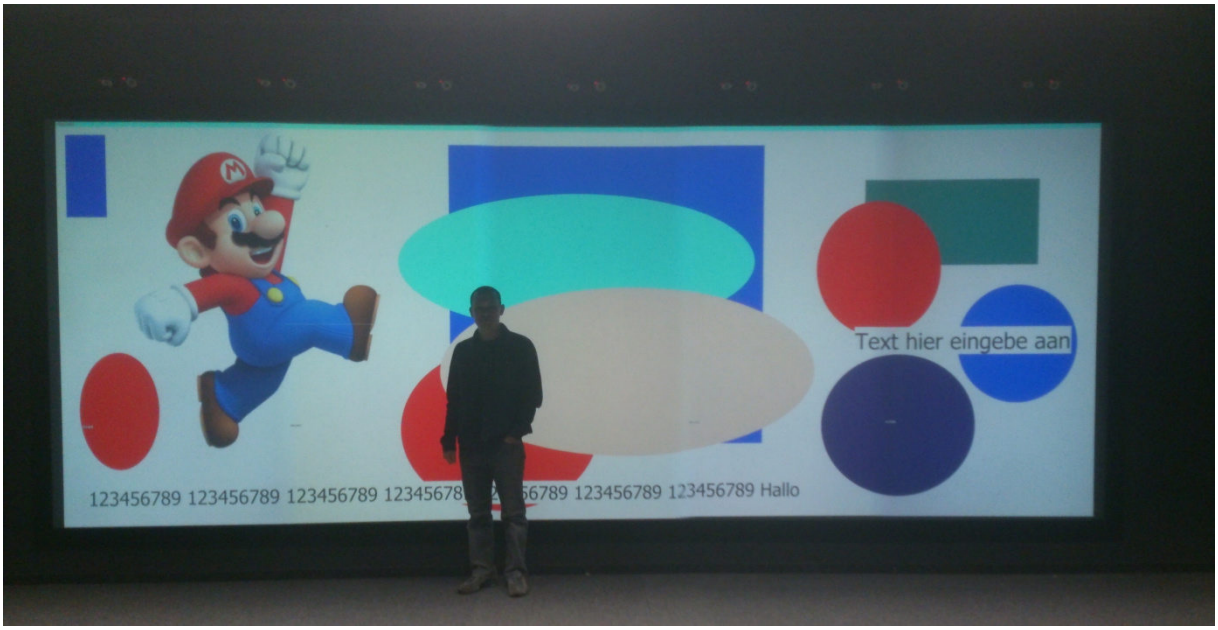

Abbildung 1.1.1 Eine Powerwall

Aufgrund der hohen Auflösung können sehr viele Informationen auf einmal dargestellt werden. Hier läuft man Gefahr die Benutzer mit Informationen regelrecht zu überfluten, sodass der Einsatz der Powerwall keinen Nutzen bringt. Hierfür müssen wichtige Informationen leichter zu identifizieren sein. Dafür können verschiedene Ansichten einer Darstellung gleichzeitig angezeigt und müssen nicht einzeln aufgerufen werden, wie es beim Desktop-Rechner der Fall ist [2]. Außerdem erleichtert die Powerwall die Arbeit im Team, indem sie die Präsentation für alle Teammitglieder gleichzeitig zugänglich macht. So kann ein Team ihren Blick gemeinsam auf die Powerwall richten und nicht auf einen normalen Desktop Display, dessen Größe nur eine beschränkte Anzahl an Leuten, die effektiv mitarbeiten können, zulässt.

1.2 Aufgabenstellung

Die Aufgabe dieser Arbeit ist die Entwicklung einer Präsentationssoftware, nach dem Vorbild von *PowerPoint.* Mit dieser Software soll der Benutzer in der Lage sein Präsentationen für hochauflösende Powerwalls, die mit mehreren Rechnern betrieben werden, zu erstellen. Die zwei wichtigsten Bestandteile der Präsentationsoftware sind die Anzeigekomponente und der Editor. Die Anzeigekomponente soll dem Benutzer das Anzeigen der Präsentationen auf mehreren Rechnern(Cluster) ermöglichen. Mit Hilfe des Editors sollen diese Präsentationen vorher auf einem Desktop-Rechner unter Berücksichtigung der Usability angefertigt werden.

Die Anzeigekomponente soll, die von Clustern gestellte Gesamtauflösung, komplett verwenden. Die Darstellung soll anhand von Vektorgrafiken der *Windows Presentation Foundation* (WPF) ablaufen. Außerdem soll die Komponente in der Lage sein, Text, Bilder und einfache Vektorgrafiken darstellen zu können.

Die zweite wichtige Komponente ist der Editor. Mit ihm soll der Benutzer in der Lage sein, hochauflösende Folien auf einem Desktop-Rechner erstellen zu können. Zur Verbesserung der Usability sollen Konzepte untersucht werden, mit deren Hilfe die geringe Auflösung auf einem Desktop-Rechner besser bewältigt werden kann. Außerdem soll ein Dateiformat entwickelt werden, mit dessen Hilfe die Inhalte der Präsentation gespeichert werden können.

2 Grundlagen

2.1 Usability auf kleinen Displays

2.1.1 Einführung und Eingliederung

Da der Powerwall-Editor bei der Erstellung einer Präsentation auf den herkömmlichen Desktop-Rechnern zum Einsatz kommt und erst die fertige Präsentation auf die Powerwall projiziert wird, entsteht das folgende Problem. Wie erstellt man Präsentationen für eine riesige Powerwall mit Hilfe herkömmlicher Desktop-Rechnern. Die großen Ausmaße der Powerwall zwingen den vorhandenen Platz auf dem Desktop-Bildschirm optimal ausnutzen zu wollen. Aber zuerst muss die Präsentation erstellt werden. Aufgrund der kleinen Arbeitsfläche muss das Design des Editors so platzsparend wie möglich sein. Folglich muss der Editor so konstruiert werden, dass die Usability aufgrund relativ geringer Größe so wenig wie möglich leidet. Eine ähnliche Ausgangslage hat man auch bei den sogenannten „small devices“, wie zum Beispiel Smartphones. Dazu gibt es verschiedene Möglichkeiten, die man auf den Editor anwenden kann, um Usability zu verbessern.

2.1.2 Möglichkeiten zur Verbesserung der Usability

In diesen Unterabschnitt behandeln wir einige Methoden zur Verbesserung der Usability auf kleinen Displays, wie Smartphones.

Ein Verfahren namens Scrolling ermöglicht eine bessere Übersicht bei großen Dokumenten. Wenn der Inhalt eines Dokuments, einer Datei oder Darstellung nicht auf das Display passt, kann man einen Teil der Information sichtbar darstellen. Die restlichen Informationen sind demzufolge nicht sichtbar, aber sofort erreichbar und einsehbar, indem man nach unten oder oben scrollt. Da man aber Scrolling generell vermeiden möchte, wurden bessere Methoden entwickelt [3].

Hierarchisches Menü mit Suboptionen ist bei mehreren Optionsmöglichkeiten üblich. Man kann diese Suboptionen gruppieren und unter einem Oberbegriff zusammenzufassen. Als Obergriff nimmt man zum Beispiel „Einstellungen“. Wenn man auf Einstellungen klickt, erscheinen drei weitere Optionsmöglichkeiten, wie Sound, Video und Allgemein [3].

Einstellungen → Sound

→Video

→Allgemein

Dieser Ansatz eignet sich besonders gut bei der Erstellung von Menüleisten, da die vorhandene Fläche begrenzt ist. Auf diese Weise gruppiert man alle wichtige Funktionen, wie „Datei speichern“, „Datei löschen“ oder „neue Datei erstellen“, unter dem Oberbegriff „Datei“. Bei Bedarf klickt der Benutzer die „Schaltfläche Datei“ an und die Optionen erscheinen. Diese sind in der Zwischenzeit nicht sichtbar.

Page-Methode kann angewandt werden, falls eine Seite zu viele Informationen enthält, um sie vollständig anzuzeigen. Man kann die Information, ähnlich wie in einem Buch, auf mehrere Seiten verteilen,. Bei der Suche nach den gesuchten Daten oder Informationen blättern man durch die vorhandenen Seiten durch, bis die benötigten Informationen gefunden sind [3]. So eine Methode kann bei Informationslandschaften angewandt werden. Während dieser Arbeit

gab es Überlegungen die hohe Auflösung der Powerwall auszunutzen, indem man zusätzlich zur der Präsentation auch die Folienhistorie anzeigt. Dieser Ansatz wurde aufgrund geringer Wichtigkeit nicht weiter verfolgt.

Shortcuts werden des Öfteren bei Adressbüchern angewandt. Wie bei einem Adressbuch auf einem Smartphone, werden Kontakte mit Hilfe einer Scrollleiste dargestellt, falls die Anzahl der Kontakte zu groß ist, um sie alle anzuzeigen. Für eine schnellere Navigation benutzt man zusätzlich Shortcuts, d.h. wenn man den Buchstaben „P“ anklickt, springt man direkt zu den Kontakten, die mit dem „P“ anfangen. Auf diesem Weg erübrigt sich das langwierige Durchsuchen des ganzen Adressbuches [3]. Die folgende Abbildung zeigt, wie man mit Hilfe von Shortcuts nach Informationen sucht.

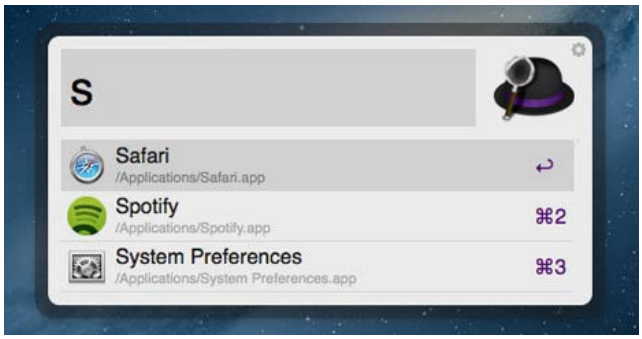

Abbildung 2.1.2.1 Suche nach einer App mit Hilfe von Shortcuts [4]

Mit Hilfe einer Baumstruktur stellt WebTwig die Präsentation einer Web-Site dar. Dem Benutzer werden am Anfang nur die Top-Knoten angezeigt. Durch das Anklicken eines dieser Knoten werden die dazugehörigen Unterknoten sichtbar [3].

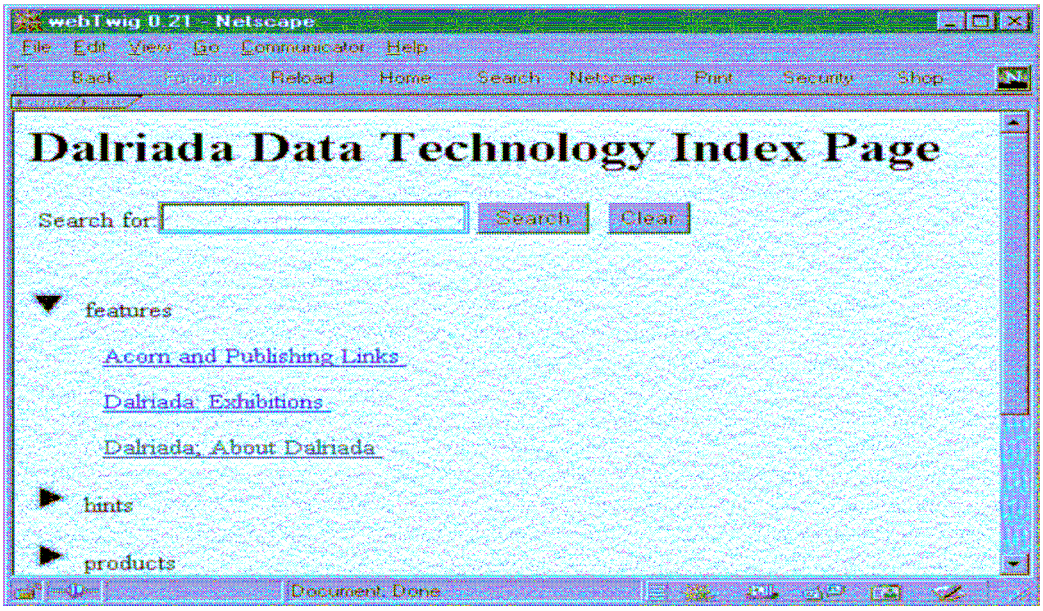

Abbildung 2.1.2.2 WebTwig Browser [3]

Eine weitere Möglichkeit ist Hashing. Wenn der Benutzer eine bestimmte Funktion aufrufen will, wie „Call XYZ“, wird durch das Anklicken von „C“ und „A“ eine Liste von Ergebnissen erstellt und angezeigt, wie „Call ABC“, „Call Doctor“ und „Call XYZ“ , aber auch „ABC

Call“. Man kann eine der vorgeschlagenen Optionen wählen oder weiter mit der Eingabe verfahren, bis die gesuchte Lösung angezeigt wird [3].

Zooming ist eine einfache Lösung, um große Mengen an Informationen auf einem begrenzten Display anzuzeigen. Durch das Zoomen verkleinert oder vergrößert man die dargestellten Informationen. Der Nachteil ist, dass beim Zoomen die Schriftgröße so klein werden kann, das sie nicht mehr lesbar ist [3]. Um dieses Problem zu vermeiden, kann man eine Methode namens Fisheye View anwenden. Dabei wird ein ausgewählter Abschnitt detaillierter dargestellt [5].

2.2 XAML

XAML *(Extended Application Markup Language)* ist eine deklarative Programmiersprache, die auf XML basiert und somit auch an die gleichen strikten Regeln gebunden ist:

- Elementenbeschreibung erfolgt durch Tags. Jeder Tag ist eine Objekt-Instanziierung und ergibt eine XML/Quellcode-Äquivalenz.
- Auf jedes Starttag muss zwingend ein Endtag folgen. Das ist wurde von XML geerbt.
- Die Groß-/Kleinschreibung muss berücksichtigt werden.
- Durch die Einschränkung komplexer Typen können neue Typen definiert werden [6].

In dieser Arbeit wird XAML, um eine Benutzeroberfläche zu beschreiben, benutzt. Und so könnte ein XAML-Code aussehen. In diesem Fall wird ein neues Fenster erstellen.

```
<Window x:Class="WpfApplication1.mainWindow"
        xmlns="httpt://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x=" httpt://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="370" Width"545">
  <Grid>

        // Hier kann man weitere Steuerelemente einfügen
        //  wie Buttons, Textboxen usw.

  </Grid>
</Window>
```

Listing 2.2.1 Struktur einer XAML-Datei

Jede XAML-Datei enthält eine Wurzel. Das heißt, dass diese Wurzel alle restlichen Elemente einer XAML-Datei einschließt. Im oberen Beispiel heißt dieses Wurzelelement *Window*. Um eine bestimmte Darstellung sicherzustellen, können beim Wurzelelement zusätzliche Attribute vordefiniert werden [7].

- „*X:Class* , gibt die Code-Behind-Datei an, die den C#-Code des aktuellen XAML-Dokuments enthält [7].“
- „Mit *xmlns* werden zwei XML-Namespaces bekannt gegeben, damit die Elemente im XAML-Code einwandfrei identifiziert werden können. Dieses Feature wurde von XML übernommen [7].“

- „Mit dem Attribut *Title* wird anschließend die Zeichenfolge beschrieben, die in der Titelleiste des Fensters angezeigt wird, und *Height* und *Width* legen die Gesamthöhe bzw. -breite des Fensters fest“ [7] .

In der Wurzel sind alle Elemente aufgelistet, aus denen das Fenster besteht. Dazu gehören Schaltflächen, Menüs, Textboxen und viele andere Elemente. Das Element *Window* kann nur ein direkt untergeordnetes Element haben. In diesem Fall ist es das Element *Grid*. Dieses Element ist ein Containersteuerelement und kann seinerseits mehrere Steuerelemente aufnehmen, wie Buttons oder auch andere Elemente [7].

2.3 WPF

Unter WPF, kurz für *Windows Presentation Foundation*, versteht man eine Bibliothek für Entwicklung grafischer Benutzeroberflächen. Diese Bibliothek ist ein Teil des .NET Frameworks. Eine Vorgängerversion von WPF ist Windows Forms. Windows Forms ist eine Programmierschnittstelle zur Erstellung grafischer Benutzeroberflächen. In den folgenden Abschnitten werden die Vorteile von WPF gegenüber Windows Forms aufgezählt und näher erläutert [7].

Die Benutzeroberfläche wird mit XAML, kurz für *eXtensible Application Markup Language,* beschrieben. Damit bewirkt man eine strikte Trennung zwischen dem Quellcode für Prozeduren und Funktionen auf einer Seite und der Beschreibung der Benutzeroberfläche auf der anderen Seite. Da XAML ziemlich mächtig ist, hat der Anwender zahlreiche Möglichkeiten, wie Objektdeklarierung, Beschreibung der Benutzeroberfläche, Veränderung der Benutzeroberfläche zur Laufzeit, Eventhandling, Binding von selbsterstellten Elementen [8]. Das kann zu einem umfangreichen XAML-Code führen. Dadurch ist der reine Quellcode, zum Beispiel C#-Code, übersichtlicher, da mit ihm kompaktere Funktionen und Operationen beschrieben werden. Die Benutzeroberfläche kann mit XAML erstellt werden. So erreicht man eine Trennung zwischen der Oberflächenbeschreibung und Funktionalität [7].

Um bessere Performance zu erreichen, unterstützen WPF-Anwendungen die 2D- und 3D-Grafiken. Durch die Nutzung von DirectX wird „die Grafikkarte zur Berechnung der grafischen Elemente herangezogen“. Dadurch wird die CPU entlastet und bessere Performance bis zur einer gewissen Anzahl an visuellen Elementen erreicht [7].

Eine der stärksten Seiten der WPF sind die umfangreichen Datenbindungsmöglichkeiten. Ein weiterer Vorteil ist, dass die WPF-Ausgabe vektorbasiert ist. Das heißt, dass Grafiken mit Hilfen von Vektoren dargestellt werden. Beim Anwenden von Vektorgrafik werden die geometrischen Informationen über Objekte gespeichert, im Gegensatz zur Pixelgrafik, bei dem die Objekte aus einzelnen Bildpunkten bestehen. Das führt dazu, dass beim Skalieren von Objekten, die mit Vektorgrafik dargestellt werden, kein Qualitätsverlust entsteht [7].

Außerdem wurde mit Hilfe von WPF ein altes Problem der Windows Forms gelöst. Früher hat sich das Ausgabefenster nicht automatisch an die Monitorauflösung angepasst, was im schlimmsten Fall dazu führte, dass einige Elemente der Benutzeroberfläche nicht mehr sichtbar waren und so unerreichbar wurden [7]. „Das WPF-Grafiksystem verwendet geräteunabhängige Einheiten, um Auflösungs- und Geräteunabhängigkeit zu ermöglichen. Jedes geräteunabhängige Pixel wird automatisch mit der Punkte pro Zoll (dots per inch, dpi)-Einstellung des Systems skaliert. Dies ermöglicht WPF-Anwendungen eine ordnungsgemäße Skalierung für unterschiedliche dpi-Einstellungen und bewirkt, dass die Anwendung automatisch dpi-Einstellungen berücksichtigt“ [9].

WPF-Anwendungen unterstützen die Darstellung verschiedener Steuerelemente:

- ***Schaltfläche:*** Buttons
- ***Menüs:*** Menu und Toolbar
- ***Auswahl:*** Combobox, Checkbox
- ***Layouts:*** Canvas, Window, Viewbox
- ***Eingabe*****:** Textbox, RichTextbox
- ***Medien*****:** Image
- ***Dialogfelder*****:** OpenFileDialog, SaveFileDialog

3 Editor

3.1 Editoroberfläche

Ein zu entwickelnder Editor benötigt am Anfang eine Editieroberfläche, auf der man verschiedene Objekte, wie Rechtecke, Ellipsen oder Texte, frei positionieren kann. Dafür eignen sich die sogenannten Layout-Container, in denen jeweils mehrere Steuerelemente und Objekte positioniert werden können. In der folgenden Abbildung sind einige Layout-Container, die in WPF verfügbar sind, abgebildet.

Layout-Container	Kurzbeschreibung
Canvas	Die Steuerelemente werden an einer angegebenen Position in einer festgelegten Größe angezeigt.
DockPanel	Die Steuerelemente können an den Rändern angedockt werden.
Grid	Dieser Container stellt eine tabellenartige Struktur zur Verfügung, in deren Zellen die einzelnen Controls positioniert werden können.
StackPanel	Die Steuerelemente werden vertikal oder horizontal angeordnet (gestapelt).
UniformGrid	Dieser Container stellt ein Raster aus gleich großen Zellen dar.
WrapPanel	Mit diesem Container werden die Controls vertikal oder horizontal angeordnet. Falls die Breite oder die Höhe nicht ausreichen, werden die enthaltenen Steuerelemente in die nächste Zeile umbrochen.

Abbildung 3.1.1 Überblick der Layout-Container [7]

Aufgrund der Anforderungen an eine Editieroberfläche, wie das Einfügen, Löschen undVerschieben von Objekten, aber auch Möglichkeiten zur Veränderung der Größe zur Laufzeit, ist das „Canvas“ die bessere Option im Vergleich zur der anderen Möglichkeiten aus der Abbildung 3.1.1. Mit Hilfe von „Canvas“ kann man Objekte einfügen, bewegen und editieren [7]. Und der wichtigste Vorteil des Canvas gegenüber den anderen Containern, ist die Möglichkeit der absoluten Positionierung. Absolute Positionierung erlaubt untergeordnete Objekte relativ zum übergeordnetem Element, in dem Fall Canvas, anordnen zu können [10].

Mit dem Aufruf *Canvas.Children.Add(Objekt)* fügt man das Objekt in den Canvas-Container ein. Die Objekte in Canvas haben vier Eigenschaften, die ihre Position innerhalb des Canvas bestimmen, nämlich *Canvas.Left, Canvas.Right, Canvas.Top und Canvas.Bottom.* Um unerwünschte Nebeneffekte zu vermeiden, ist es nötig nur eins aus den Pärchen *Canvas.Left/Canvas.Right,* und *Canvas.Top/Canvas.Bottom* zu setzen, wie zum Beispiel die Kombination *Canvas.Left/Canvas.Top* [7]. Mit den Aufrufen *Canvas.SetLeft(Objekt,X)* und *Canvas.SetTop(Objekt,Y)* positionieren wir das Objekt mit den (X,Y)- Koordinaten in Canvas.

Als ein Layout-Container bietet das Canvas keine Möglichkeiten zur Verschiebung, Vergrößerung oder Verkleinerung von Objekten zur Laufzeit, was bei einem Editor Grundvoraussetzung ist. Um das zu ermöglichen, benötigt man eine Hilfsklasse. Im Laufe der Studienarbeit wurden zwei mögliche Lösungen miteinander verglichen, ResizingAdorner-Klasse [11] und DragCanvas-Klasse [12]. Beide Klassen ermöglichen das Verschieben von Objekten innerhalb des Canvas zur Laufzeit, aber mit der DragCanvas-Klasse ist es nicht möglich ein bestimmtes Objekt zu vergrößern oder zu verkleinern. Bei einer Präsentationsoftware wäre die fehlende Fähigkeit zur Größenveränderung von Objekten ziemlich gravierend. Aus diesem Grund verwendet dieser Editor die ResizingAdorner-Klasse, welche sowohl das Verschieben von Objekten, als auch die Manipulation im Bezug auf die Größe des Objekts ermöglicht.

Präsentationen für die Powerwall sind viel größer, als bei den herkömmlichen Desktop-Rechner. Da der Benutzer diese aber dennoch auf dem Desktop-Rechner erstellt, würde das Canvas die vorhandene Bildschirmgröße sprengen. Alle Informationen, die nicht auf dem angezeigten Bildschirm passen, sind für den Benutzer automatisch nicht sichtbar und somit nicht zugänglich. Als Lösung für dieses Problemplatziert man das Canvas in einem Steuerelement namens „ScrollViewer“. Dieser „ScrollViewer“ erlaubt durch den Canvas zu scrollen, wenn das Canvas zu groß für den Bildschirm ist. Diese Vorgehensweise ähnelt Photoshop. Auch hier hat der Benutzer die Möglichkeit nach oben oder unten zu scrollen, falls das Bild zu groß für den Bildschirm ist

3.2 Objekte erzeugen und editieren

Die Anforderung ist, dass der Editor in der Lage sein muss, unterschiedlich formatierte Texte, Pixelgrafiken und einfache Vektorgrafiken (Linien, Rechtecke, usw.) verarbeiten zu können.

Für eine Texteingabe und Textformatierung ist die einfachste Lösung eine „TextBox“. Die „TextBox“ erfüllt die Anforderungen und ist einfach zu implementieren. Mit den Eigenschaften *FontSize, FontFamily* und *Foreground* lassen sich die Schriftgröße, die Schriftart und die Farbe des Textes verändern. Die einfachen Vektorgrafiken sind mit der Shape-Klasse darstellbar. Die Shape-Klasse enthält weitere Unterklassen wie Ellipse, Rectangel oder Line, mit deren Hilfe Linien, Rechtecke und Ellipsen gezeichnet werden können. Außerdem besitzt die Shape-Klasse, also folglich auch die Unterklassen, verschiedene Eigenschaften, wie Farbe. Durch diese Bearbeitungsmöglichkeiten können die Objekte an die Wünsche des Benutzers angepasst werden. Um einfache Pixelgrafiken darzustellen, bietet sich die Image-Klasse an. Mit dieser Klasse ist das Laden von folgenden Bildtypen möglich: .bmp, .gif, .ico, .jpg, .png, .wdp und .tiff [13]. Durch das Öffnen eines Open-Image-Fensters und der Auswahl des Bildes wird das Bild als Quelle des Image-Controls verwendet und im „Canvas“ platziert.

Die Symbole für die jeweiligen Objekte werden in einer „ToolBar“, einem Container, gespeichert. Durch das Anklicken werden dazugehörige Objekte im „Canvas“ erstellt und positioniert. Die Auswahl an Schriftgrößen, Schriftarten und Farben wird jeweils durch eine „ComboBox“ ermöglicht. So werden viele Möglichkeiten zur Manipulation von Objekte auf einem kleinen Raum gebündelt. Durch das Anklicken der bestimmten „ComboBox“ werden die Optionen in einem Scrollfenster präsentiert. Dieses Design orientiert sich an bekannte Editoren wie PowerPoint.

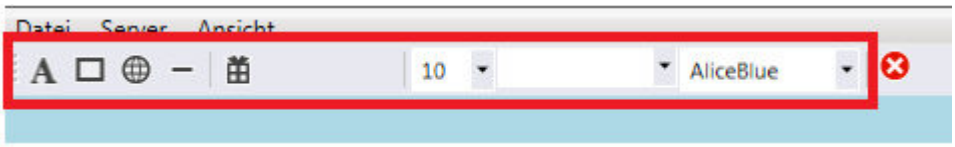

Abbildung 3.2.1 ToolBar mit den Unterelementen

3.3 Präsentation erstellen

Nachdem der Benutzer jetzt in der Lage ist, Objekte erstellen und auf die Editieroberfläche platzieren zu können, ist der nächste Schritt die Erstellung einer kompletten Präsentation. Bis zu diesem Zeitpunkt hat man nur die Seite, die in „Canvas“ dargestellt wird. Das Ziel ist aber, das eine Präsentation aus mehreren Seiten bestellt und daraus folgt, dass der Benutzer in der Lage sein muss, neue Seiten erstellen, vor –und zurückblättern aber auch unerwünschte Seiten löschen zu können.

Dafür werden hier drei Möglichkeiten vorgestellt:

- Array
- ArrayListe
- Generische Liste

Ein Array kann eine bestimmte Menge an Elementen speichern. Die Elementen und die Größe des Array müssen beim Initialisieren angegeben werden. Die Elemente können vordefinierte Typen sein, wie String, Integer oder selbstdefinierte Typen, wie Person. Das Element kann Canvas sein, wobei ein Canvas eine Seite der Präsentation darstellt. Der große Nachteil ist, dass die Größe vordefiniert sein muss und im Nachhinein nicht mehr verändern kann. Deswegen eignet sich ein Array in diesem Fall nicht, da es bessere Möglichkeiten gibt [14].

Die zweite Möglichkeit ist die Verwendung einer ArrayListe. Es ist ein Array, dessen Größe der Benutzer dynamisch variieren kann. Somit ist eine ArrayListe einem Array überlegen, da sie die gleichen Eigenschaften besitzt und den Nachteil festgeschriebener Größe nicht mit sich zieht. In einer ArrayListe können verschiedene Elemente gespeichert werden und sie müssen nicht vom gleichen Typ sein [15].

Die letzte Methode ist das Benutzen einer generischen Liste, bei der aufgrund ihrer dynamischen Größe, die Möglichkeit besteht, zur Laufzeit neue Elemente hinzuzufügen oder zu entfernen. Sie unterstützt die Methoden „Durchsuchen“ und „Bearbeiten“. Im Vergleich zur einer ArrayListe kann diese nur einen Typen enthalten. Da in diesem Fall man nur ein Typ Canvas speichern will, ist es kein Nachteil. Die Vorteile einer generischen Liste gegenüber der ArrayListe liegen in der Performance und Typsicherheit. Wenn man als Element einen Wertetyp verwendet, verschlechtern das Boxing und Unboxing die Performance der ArrayListe [16].

Aus diesen Grund wird in diesem Fall die generische Liste, bei der das Element Canvas ist, verwendet. Jedes Canvas entspricht einer Seite der Präsentation.

3.4 Speichern und Laden

Nachdem der Erstellung einer Präsentation ist der nächste Schritt diese zu speichern. Wie speichert man jedoch den Inhalt einer Seite. Eine Möglichkeit wäre eine List<Array[]> zu machen, wobei in einem Array alle wichtigen Eigenschaften eines Objekts gespeichert wären, wie Typ, Größe, Position usw. Somit würde ein Array genau einem Objekt entsprechen und die Liste wäre eine Kette aus Objekten, in der Reihenfolge, in der sie erstellt und eingefügt wurden. Der Vorteil dieser Vorgehensweise ist es, dass die beiden Typen List<> und Array[] mit der DataContractSerializer-Klasse sofort in einen XML-Stream serialisiert und wieder deserialisiert werden können [17]. Das würde bei einer möglichen Datenübertragung als XML-Stream die manuelle Implementierung eines Datenvertrags(engl. Data Contract) ersparen. Das Problem besteht hiermit, dass man diesen Vorgang eigenständig implementieren muss. Man müsste viele Vorgänge manuell erstellen, wie zum Beispiel jede wichtige Eigenschaft in ein vorgesehenes Feld eintragen und nachher auch auslesen. Diese List<Array[]> kann man als eine Text-Datei auf einem Speichermedium abspeichern. Hier müsste man jedoch alles einzeln und manuell implementieren. Ein bessere Variante wäre das Abspeichern des Inhalt einer Seite als eine XAML-Datei. Dafür gibt es schon implementierte

Methoden, wie *XamlWriter.Save(Canvas,FileStream)* [18]. So wird der Inhalt eines Canvas in einen FileStream gespeichert. Ein FileStream, dem vorher ein Pfad übergeben wird, entspricht einer Datei. Das Laden ist ebenfalls schon vordefiniert. Dafür wird ein XML-Reader benutzt, weil XAML-Reader keine Methode hat, mit der man direkt eine Datei laden kann. Aber XAML kann mit der Methode „Load „ aus einem XML-Reader einen Canvas füllen [19]. Weil XAML auf XML basiert, entstehen syntaktisch keine Problem. So kann man auf die vordefinierten Methode zurückgreifen, was eine Zeitersparnis beim Implementieren von Laden und Speichern einer Seite als eine XAML-Datei, bedeutet.

```
XmlReader rdr = XmlReader.Create(Dateipfad)
Canvas myCanvas = XamlReader.Load(rdr) as Canvas;
```

Listing 3.4.1 Rekonstruktion des Canvas mit Hilfe einer XAML-Datei

Doch wie kann man nicht nur eine Seite der Präsentation, sondern gleich mehrere Seiten abspeichern? Außerdem kann man die Pixelgrafiken nicht in einer XAML-Datei abspeichern, da wird nur ein Verweis auf das Bild abgelegt. Das heißt, dass jedes Bild gesondert abspeichert wird. Es biete sich an, alle XAML-Dateien und Bilddateien in eine ZIP-Datei zusammenzufassen. Dafür erstellt man eine neue ZIP-Datei und fügt alle nötigen Datei, wie XAML-Datei und Bilder ein und speichern diese ab, siehe Listing 3.4.3. Beim Laden werden die Daten entpackt, die Präsentation rekonstruiert und die entpackten Dateien wieder gelöscht.

```
ZipFile zip = new ZipFile();
zip.AddFile(Pfad der einzelnen Dateien oder Bilder, Unterordner);
zip.Save(Pfad der Zip-Datei);
```

Listing 3.4.2 Alle Dateien in eine ZIP-Datei zusammenfassen

Nachdem die Mechanismen geklärt sind, muss nur noch der Pfad und der Namen der ZIP-Datei bestimmt werden. Dafür eignet sich *Microsoft.Win32.SaveFileDialog,* welches besonders benutzerfreundlich und leicht nachvollziehbar ist.

```
Microsoft.Win32.SaveFileDialog dlg = new Microsoft.Win32.SaveFileDialog();
dlg.FileName = "result"; // Name der Datei
dlg.DefaultExt = ".zip"; // Format der Datei
 dlg.Filter = "ZIP documents (.zip)|*.zip"; //  Filter für Formate
```

Listing 3.4.3 Fenster zum Speichern einer Datei

Die folgende Abbildung zeigt ein Fenster, welches sich dem Benutzer zum Speichern öffnet.

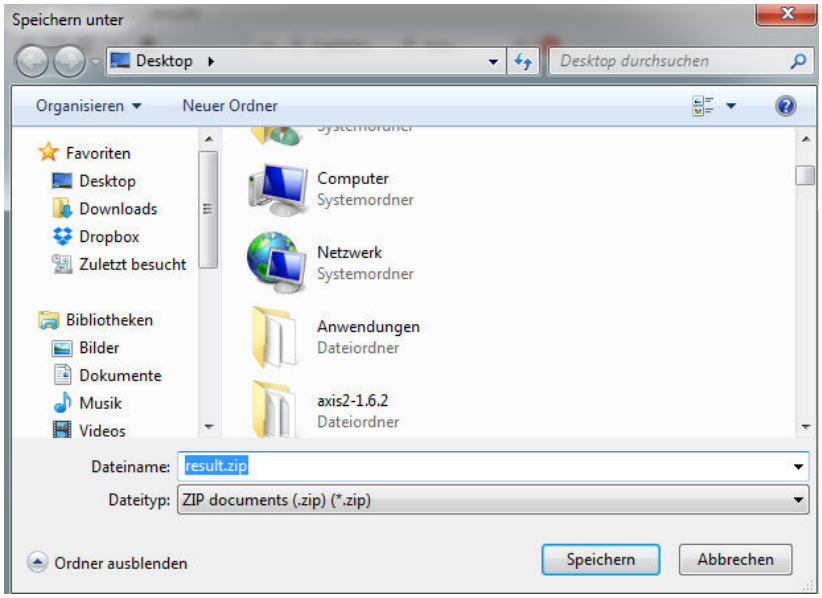

Abbildung 3.4.1 Name und Speicherort bestimmen

3.5 Usability

3.5.1 Panning und Zoom

Da mit dem Editor Präsentation meistens so groß werden können, dass sie nicht auf einen Bildschirm passen, ist es notwendig, gewisse Techniken anzuwenden. Um trotz Platzmangel effektiv arbeiten zu können

Mit dem Zoomen kann der Benutzer die Darstellung der aktuellen Seite der Präsentation vergrößern oder verkleinern. Die Objekte behalten ihre Eigenschaften, wie Höhe oder Breite, damit sich der Benutzer, ohne mühseliges Scrollen, einen Gesamtüberblick verschaffen kann. Die Zoomreichweite liegt zwischen 1% und 100 % der Originalgröße.

Unter Panning versteht man in diesem Fall, das automatische Verschieben der Scrollleisten, wenn man den Mauszeiger in die gewünschte Richtung bewegt. Bei einer sehr großen Seite auf einem relativ kleinen Bildschirm ist es sehr mühselig ein Objekt über die gesamte Seite zu verschieben. Wenn man das Ende des Bildschirms erreicht hat, muss man zuerst in die gewünschte Richtung scrollen, das Objekt wieder verschieben, solange bis man die richtige Stelle erreicht hat. Mit Panning wird man automatisch in die Richtung gescrollt, in die man das Objekt verschiebt oder den Mauszeiger bewegt. Der Editor kann das Panning mit gedrückter „Leertaste“ ausführen.

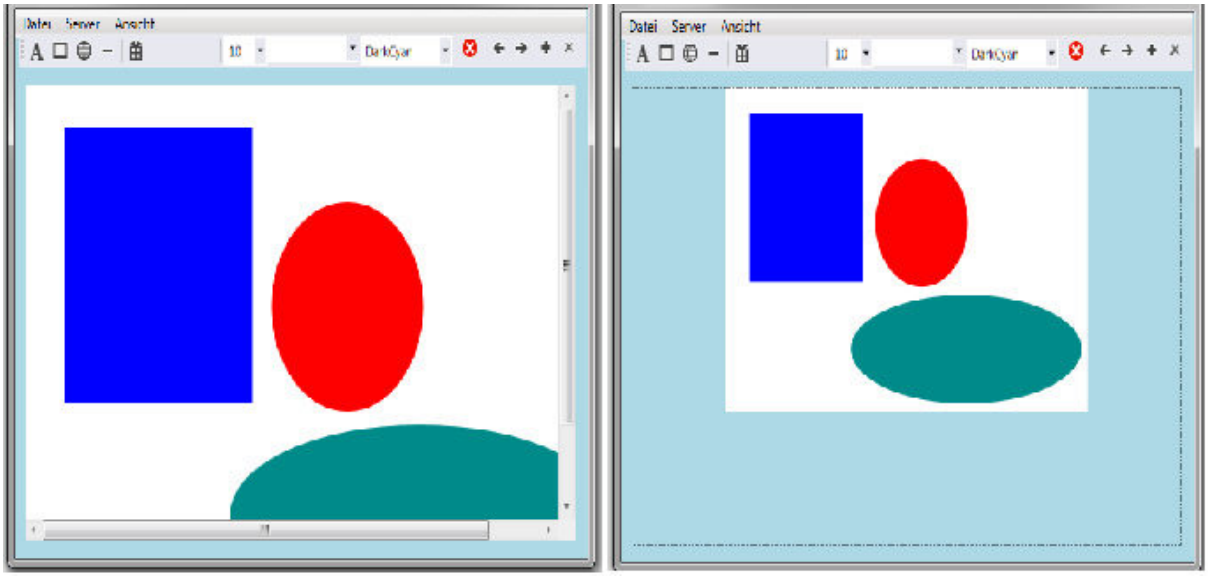

Abbildung 3.5.1.1 Zoomen

3.5.2 Verschachtelte Menüs

Aufgrund des begrenzten Platzes auf der Benutzeroberfläche, müssen die Bedienungsmöglichkeiten, wie Datei speichern, Datei laden usw., unter einem Oberbegriff zusammengefasst werden. Beim Aufruf einer Schaltfläche, die als Oberbegriff für die untergeordneten Funktionen fungiert, öffnet sich eine Schaltfläche mit Unterfunktionen. Solange der Aufruf nicht erfolgt ist diese Schaltfläche nicht sichtbar. So wird der Platz für wichtigere Elemente freigehalten, solange diese Optionsmöglichkeiten nicht aufgerufen werden. Wenn man dann diese Optionen letztendlich einsehen und aufrufen will, klickt man zum Beispiel auf die Schaltfläche *Datei* und es öffnet sich das Fenster mit mehreren Optionen, wie Datei speichern, Datei öffnen oder Datei drucken, die aufgerufen werden können. Solange jedoch die Präsentation erstellt wird, sind diese Optionen nicht sichtbar und verbrauchen den wertvollen Platz auf einem Bildschirm nicht.

3.5.3 Anpassung der Bedienungsfläche der Anwendung an Fenstergröße

Der Editor enthält viele Elemente wie Buttons, Menüs, Editierfläche(Canvas) oder Texteingabefelder. Wenn man das Anwendungsfenster maximiert, möchte der Benutzer, dass nur bestimmte Objekte mit vergrößert werden. Elemente wie Buttons oder Menüleisten sollen ihre Originalgröße behalten und die Editierfläche, in der die Präsentation erstellt werden soll, soll automatisch an die Fenstergröße angepasst werden. Dadurch erhält man bei der Maximierung des Anwendungsfensters mehr Editierfläche, was die Erstellung der Präsentation einfacher macht.

4 Kommunikationskomponenten

4.1 Aufbaustruktur einer Powerwall

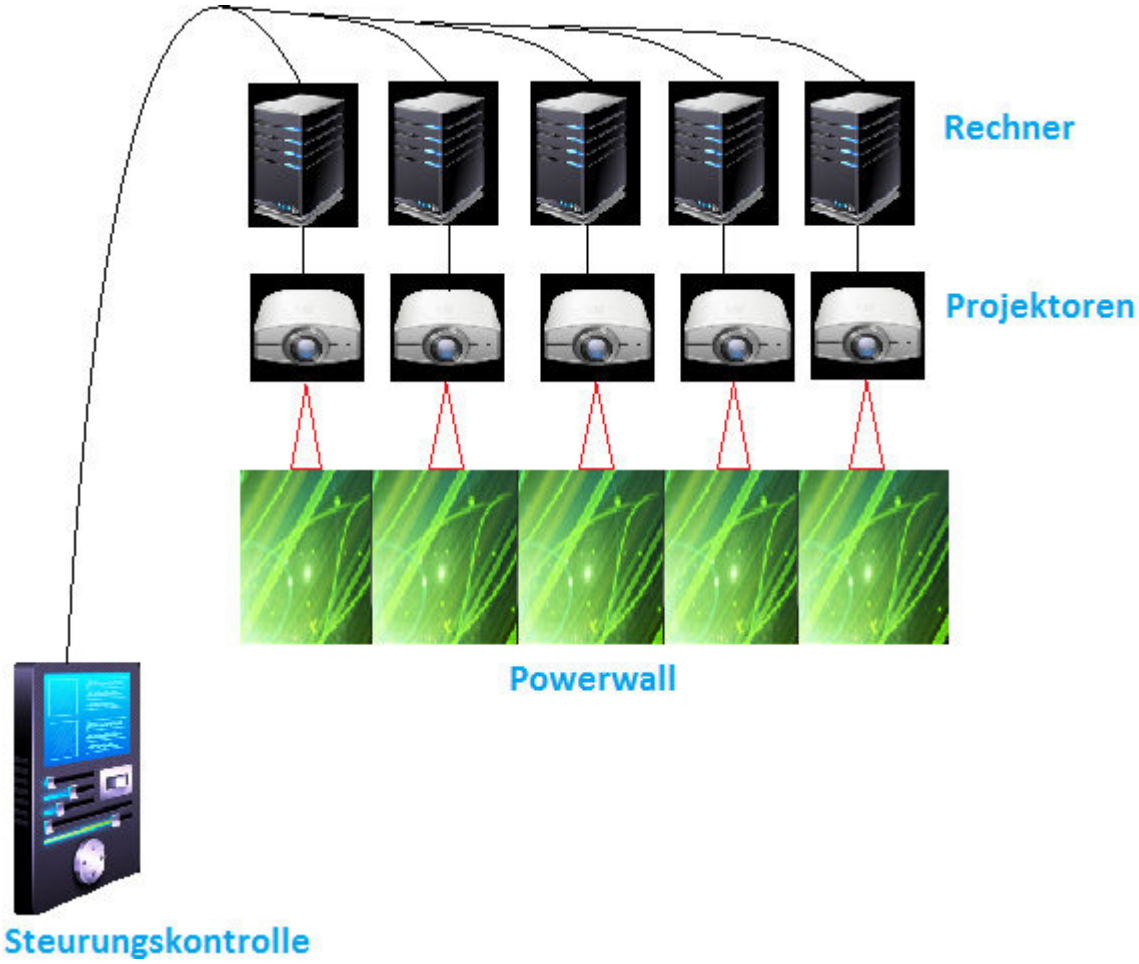

Abbildung 4.1.1 Aufbaustruktur einer Powerwall mit dazugehörigen Komponenten

In Abbildung 4.1.1 sieht man die Aufbaustruktur einer Powerwall mit dazugehörigen Komponenten. Die Steuerungskontrolle sendet die Informationen an die Rechner. Diese Informationen werden auf den Rechnern gerendert und mit Hilfe von Projektoren auf die Powerwall projiziert. In dieser Arbeit läuft auf der Steuerungskontrolle der Server, mit dem sich die Client, die auf den Rechnern laufen, verbinden können. Nachdem Informationen gesendet sind, werden diese nach dem Rendern von den Projektoren auf die Powerwall projiziert.

4.2 Server-Client-Model

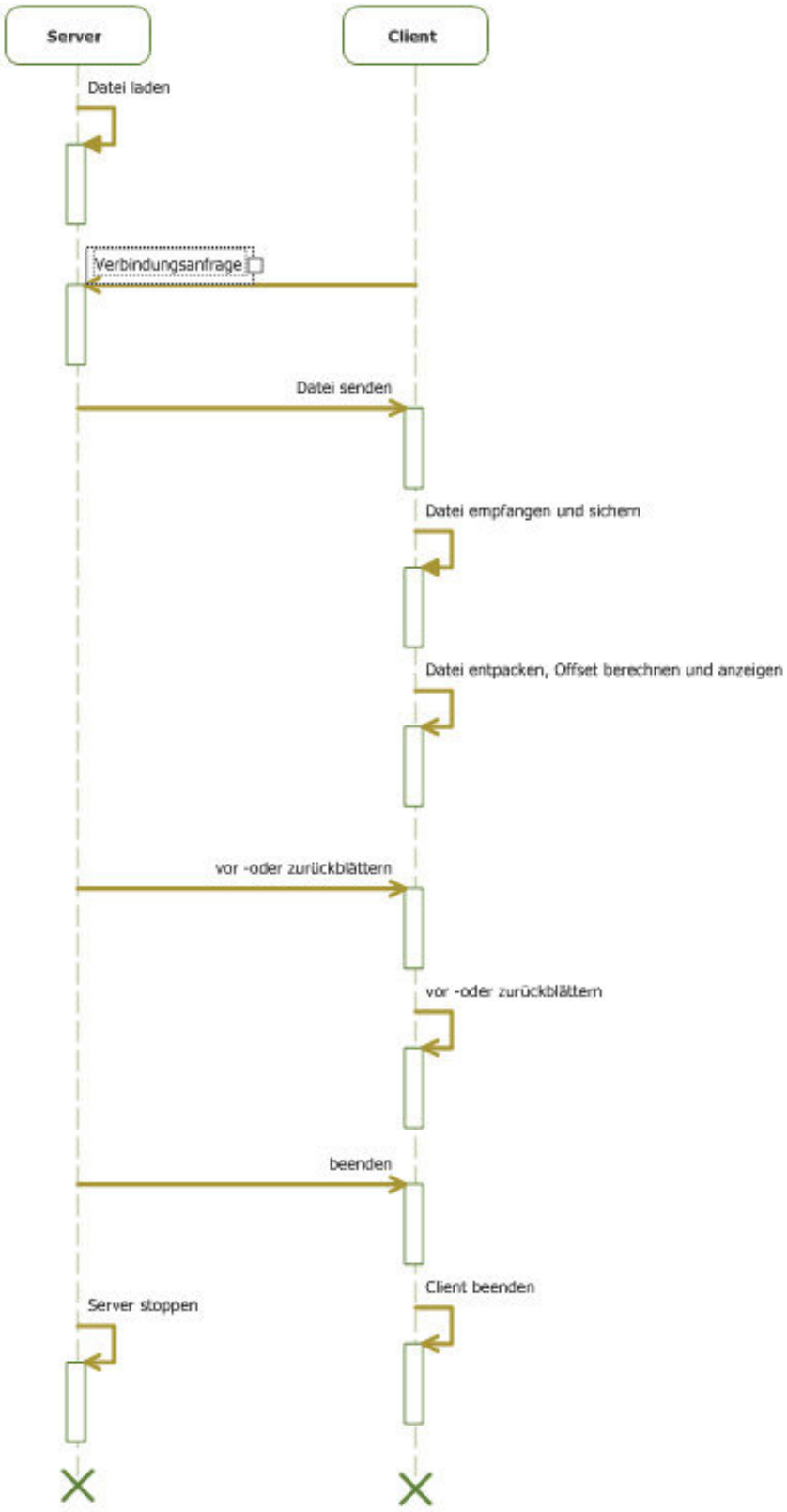

Abbildung 4.2.1 Übersicht des Server-Client-Modells

Der Server kann mit mehreren Clients kommunizieren [20]. Da jedoch jede einzelne Server-Client-Verbindung gleich ist, zeigt die Abbildung 4.1.1 nur die Kommunikation mit einem Client.

Wenn der Server gestartet wird, sucht der Benutzer die Präsentation aus, die zum Client übertragen werden soll. Falls sich ein oder mehrere Clients eine Verbindungsanfrage an den Server schicken, antwortet er, indem er die Datei mit der Präsentation überträgt. Auf der Clientseite wird die Datei empfangen und im eigenen Verzeichnis gespeichert. Danach entpackt der Client die Präsentation, berechnet den Offset und zeigt die Präsentation an. Jetzt kann man über den Server Befehle an den Client senden. Diese Befehle dienen dazu, die Präsentation auf der Clientseite durchzublättern. Falls erwünscht, kann der Befehl zur Beendigung der Kommunikation gesendet werden. Nachdem Erhalt dieses Befehls wird die Verbindung geschlossen und die Clientanwendung beendet. Auf der Serverseite beendet der TcpListener den Lauschvorgang, entfernt alle Clientinformationen aus der Liste und ist dann bereit erneut gestartet zu werden.

4.3 Server

Nachdem eine Präsentation erstellt wurde, müsste diese an einen Client übertragen werden, sodass sie dann an einer Powerwall angezeigt werden kann. Zuerst sind jedoch die Möglichkeiten zur Netzwerkkommunikation zu betrachten. Es gibt zahlreiche Optionen, wie TcpListener, Socket, WCF-Webservices, RCP usw. In dieser Arbeit hat man sich aufgrund einfacher Netzwerkarchitektur auf zwei Optionen beschränkt. Auf der einen Seite hat man die Klasse System.Net.Sockets.TcpListener, kurz TcpListener, auf der anderen Seite System.Net.Sockets.Socket oder abgekürzt Socket. Die beiden Varianten stellen eine Anzahl an Methoden und Eigenschaften zur Netzwerkkommunikation bereit [21].

Die TcpListener-Klasse überwacht eingehende Verbindungsanfragen und nimmt diese an, falls es erwünscht ist. Man erstellt einen TcpListener, indem man ihm einen IPEndPoint übergibt. Dieser IPEndPoint besteht aus einer IP-Adresse und Portnummer. Man kann statt einer bestimmten IP-Adresse eine Konstante *Any* übergeben, dann hört der TcpListener nicht eine bestimmte IP-Adresse ab, sondern beliebig viele. Mit der Methode *Start()* beginnt der TcpListener nach den eingehenden Verbindungen zu lauschen. Diese Verbindungen werden in eine Warteschleife verwiesen. Mit der Methode *Stop()* hört der TcpListener auf die eingehenden Verbindungen zu überwachen. Mit *BeginAcceptTcpClient()* wird eine Verbindung aufgerufen und dann aus der Warteschlange entfernt [22].

```
TcpListener lis = new TcpListener(IPAddress.Any, 2509);
lis.Start();
lis.BeginAcceptTcpClient( Parameter);

//mache was

lis.Stop()
```

Listing 4.3.1 TcpListener

Die Socket-Klasse kann entweder ein verbindungsorientiertes Protokoll, wie TCP benutzen oder ein verbindungsloses Protokoll, wie UDP, verwenden. Mit einem verbindungsorientierten Protokoll stellt die Socket-Klasse die Listen-Methode bereit, mit der sich die eingehenden Verbindungen überwachen lassen können. Mit den Methoden *Send* oder *Receive* können sie Daten senden. Beim verbindungslosen Protokoll ist keine

Überwachungsmethode für eingehende Verbindungen nötig. Mit der Methode *SendTo* können Daten gesendet werden [23].

```
Socket soc = new Socket(AddressFamily.InterNetwork,
                                    SocketType.Stream, ProtocolType.Tcp );
soc.Listen();
```

Listing 4.3.2 EinTCP/IP-Socket

Beide Klassen eignen sich für die Konfiguration eines Servers. Der TcpListener ist jedoch bei einfacher Netzwerkkommunikation, gemäß zahlreicher Erfahrungsberichte leichter im Umgang.

4.4 Client

Auch beim Client hat man eine Wahl zwischen der Klasse Socket und der Klasse TcpClient. Da die Wahl beim Server für TcpListener gefallen ist ist die Wahl der Klasse TcpClient eine logische Konsequenz. Dadurch hat man eine bessere Anpassung zwischen Server und Client, die beide auf dem TCP-Protokoll laufen.

Nachdem eine Verbindung zwischen Server und Client hergestellt wurde, stellt sich die Frage, ob man die ganze Präsentation auf einmal überträgt oder nur die aktuelle Folie der Präsentation. Die Möglichkeit, nur die aktuelle Folie zu übertragen, hat den Vorteil, dass man am Anfang der Präsentation nur aktuelle Folie überträgt und so die Sendelast relativ gering hält. Außerdem lassen sich die Folien während der Präsentation verändern. Hier wird einem die Option freigehalten, während der Präsentation wichtige Anmerkungen hinzuzufügen oder Fehler beseitigen zu können. Der Nachteil ist, dass jede Folie einzeln gesendet werden muss. Bei größeren Folien mit mehreren Bildern dauert es länger. Dadurch muss man im schlimmsten Fall auf die nächste Folie warten, was den Fluss der Präsentation stört und die Zuhörer unnötig ablenkt. Die andere Möglichkeit ist es ist die gesamte Präsentation auf einmal zu senden, wenn sich der Client mit dem Server verbindet. Der Nachteil besteht darin, dass am Anfang eine größere Sendelast anfällt, weil die gesamte Präsentation auf einmal gesendet wird. Auf der anderen Seite hat man den Vorteil, dass während der Präsentation keine Folien gesendet werden müssen, sondern nur ein Befehl zum vor -oder zurückblättern. Das erlaubt die Sendelast während der Präsentation gering zu halten. Dadurch wirkt die Präsentation flüssiger. Die Möglichkeit, die gesamte Präsentation auf einmal zu senden, ist in diesem Fall besser, da die Powerwall nur zum Vorführen einer Präsentation benutzt wird und nicht zum Erstellen einer. Die anfängliche größere Sendelast wird durch die niedrigere Sendelast während der Präsentation ausgeglichen.

Weil die Powerwall aus mehreren Displays besteht und jedes Display genau einem Rechner beziehungsweise Client entspricht, muss man dafür sorgen, dass alle Displays zusammen ein gemeinsames Bild zeigen und nicht den gleichen Ausschnitt. Dafür wird ein Offset für jeden Client definiert. Der Offset wird größer, je weiter sich der Display rechts im Gesamtverbund der Displays befindet. Wenn sich die Clients in einer bestimmten Reihenfolge verbinden, kann man anhand dieser Reihenfolge den Offset jeweils so berechnen, dass am Ende das Gesamtbild korrekt wiedergegeben wird. Zum Beispiel würden sich die Clients von links nach rechts mit dem Server verbinden, würde dem ersten Client der Offset = 0 und den weiteren Client immer größere Offsets zugewiesen. In diesem Fall ist die Reihenfolge der Verbindungen zufällig. Somit fällt diese Möglichkeit aus. Die bessere Option wäre, die wichtigen Daten in einer Datei abzuspeichern. Bei der Darstellung der Präsentation würde der Client auf diese Datei zugreifen und anhand bestimmter Kriterien, wie IP-Adresse oder Name,

den Offset für diesen Client bestimmen. Der Vorteil dieser Vorgehensweise ist, dass man nur die Datei austauschen muss, um sich auf die neuen Konfigurationen einzustellen. Die Notwendigkeit zur Quellcodemanipulation fällt aus.

5 Zusammenfassung

Das Ziel dieser Studienarbeit war die Entwicklung einer Präsentationssoftware, die speziell auf hochauflösende Powerwalls zugeschnitten ist. Der erste Teil der Aufgabe bestand aus der Entwicklung eines Editors, mit dessen Hilfe eine Präsentation erstellt werden kann. Diese Präsentation kann unterschiedlich formatierten Text, Pixelgrafiken und einfache Vektorgrafiken (Linien, Quadrate, …) enthalten. Die Objekte der Präsentation können eingefügt, positioniert und gelöscht werden. Das verwendete Dateiformat erlaubt die Speicherung aller Inhalte der Präsentation in einer ZIP-Datei. Für die Kommunikation mit einer Powerwall enthält der Editor eine Serverkomponente. Diese ermöglicht, dass sich mehrere Clients, mit deren Hilfe man das Gesamtbild auf die Powerwall projiziert, gleichzeitig verbinden können. Der Offset, der zur korrekten Darstellung des Gesamtbildes für jeden einzelnen Client explizit berechnet werden soll, wird in einer XML-Datei gespeichert. Beim Zeichnen der Präsentation greift jeder Client auf die Datei zu und bestimmt anhand seines Rechnernamens seinen Offset. Das Bild wird dann um diesen Offset versetzt, was zum korrekten Gesamtbild auf der Powerwall führt.

Anhang

Im Anhang werden der Powerwalleditor und seinen Funktionen beschrieben. Es werden alle Umgangsmöglichkeiten beschrieben und die Funktionsweise des Editors dem User erklärt.

Übersicht

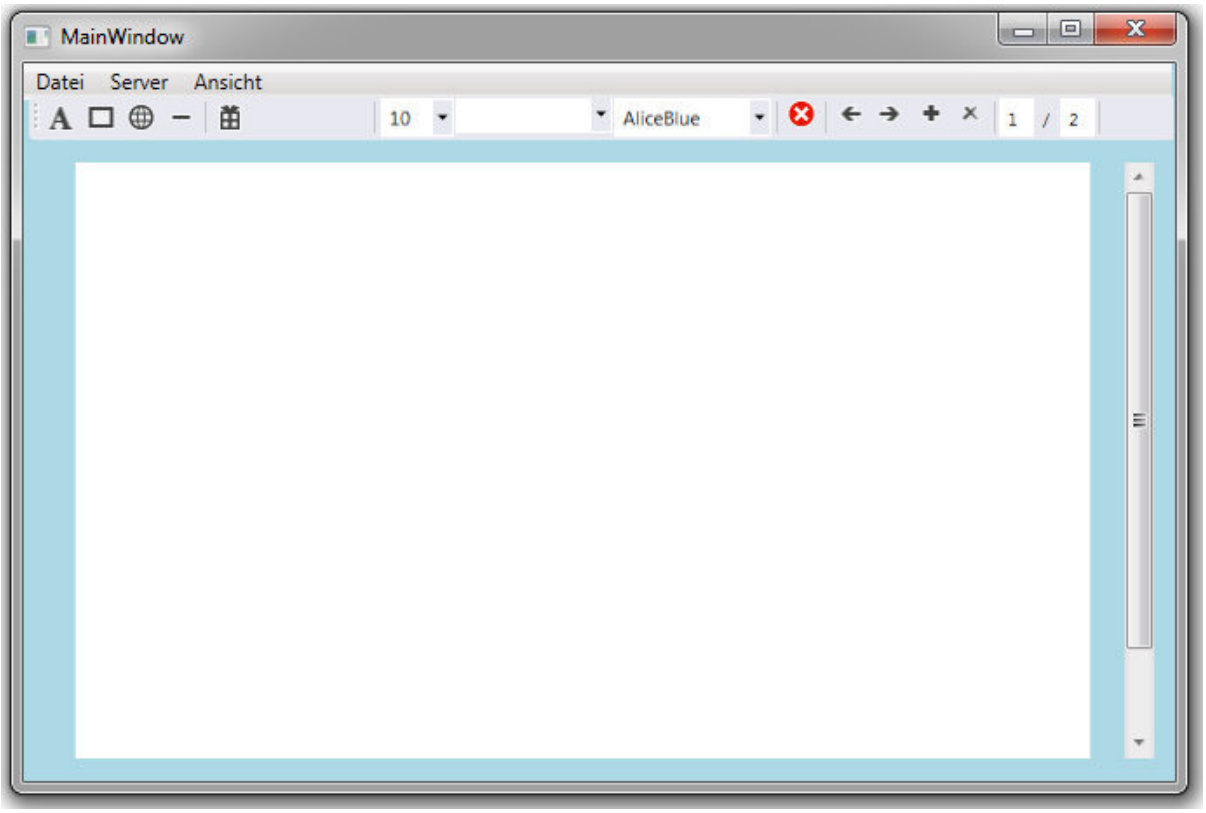

Abbildung 6.1 Editorübersicht

In Abbildung 6.1 sieht man das Design des Editors. Mit Hilfe des Editors kann man verschiedene Objekte, wie Ellipsen, Rechtecke erzeugen oder Linien zeichnen. Außerdem ist es möglich beliebig viele TextBoxen zu platzieren, um Texteingabe zu ermöglichen. Dieser Text kann formatiert werden, indem man die Textgrößen oder Schriftart verändert. Desweiteren ist es möglich von der Festplatte ein oder mehrere Bilder in PNG-Format zu laden. Alle diese Objekte können dann auf der Folie an der aus gewählten Position platziert werden. Zuerst klickt man auf das gewünschte Objekt und dann an die Position, an der das Objekt positioniert werden soll. Falls nötig kann man mit der Schaltfläche *Neue Seite* weitere Folien erzeugen und so eine Präsentation aufbauen. Unnötige Seiten können mit der Schaltfläche *Seite löschen* entfernt werden. Weiterhin kann man mit den Schaltflächen *Zurück* und *Vor* durch die Präsentation blättern. Unter dem Menü *Menü → Datei → Speichern* kann die Präsentation persistent gespeichert werden und bei Bedarf *im Menü → Datei → Öffnen* wieder laden.

Objekte erzeugen

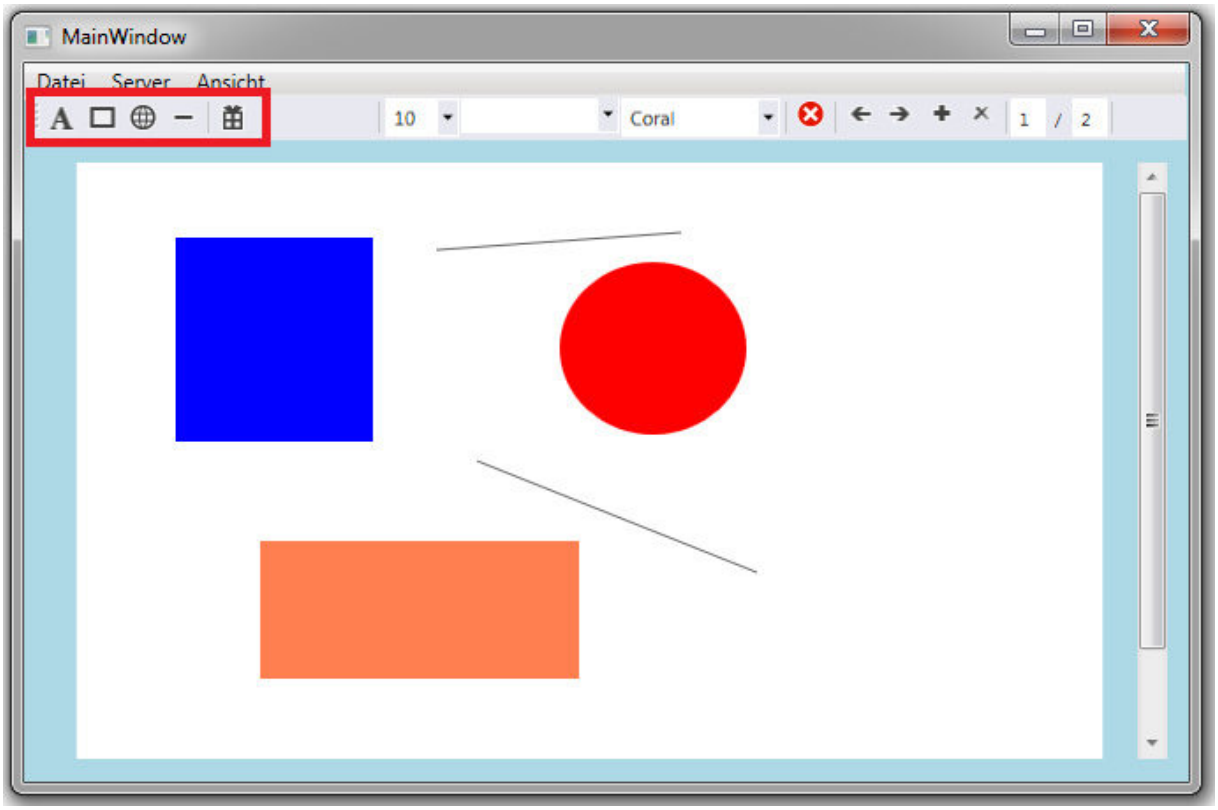

Abbildung 6.2 Objekte erzeugen und platzieren

In Abbildung 6.2 sind Optionen zum Auswählen und Platzieren von Objekten rot markiert. Von links nach rechts hat man folgende Optionen:

- ***TextBox***: In einer TextBox kann man Text eingeben und formatieren
- ***Rechteck***: Einen Rechteck erzeugen, platzieren und Größe verändern
- ***Ellipse***: Eine Ellipse erzeugen, platzieren und Größe verändern
- ***Linie***: Linie zeichnen
- ***PNG-Bild***: Ein Pixelbild in PNG-Format von einem Speichermedium laden und platzieren

Diese Objekte können auf der Folie positioniert und danach an die Bedürfnisse des Users angepasst werden, indem man sie an die richtige Stelle verschiebt und die Größe ändert.

In Abbildung 6.3 sieht man im blau markierten Bereich weitere Möglichkeiten zur Formatierung. Von links nach rechts hat man folgende Optionen:

- Textgröße verändern, gilt nur für TextBox
- Schriftart verändern, gilt nur für TextBox
- Farbe verändern, das gilt für TextBox und Objekte wie Rechteck oder Ellipse

Wie in Abbildung 6.3 zu sehen, kann man den Text formatieren, indem man Farbe, Schriftgröße und Schriftart ändert. Aber auch andere Objekte können modifiziert werden, wie Abbildung anhand eines Rechtecks verdeutlicht.

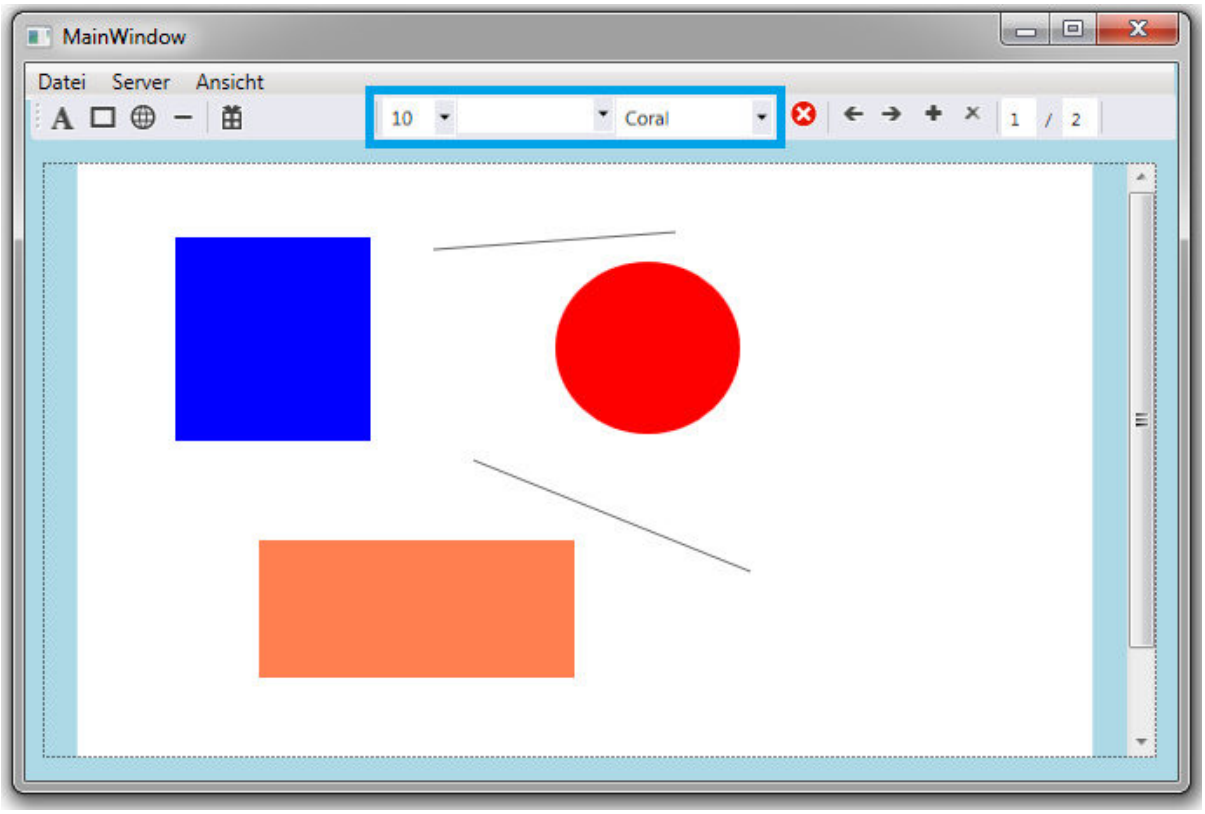

Abbildung 6.3 Text und Objekte formatieren

Präsentation erstellen

Mit dem Powerwall-Editor hat man natürlich auch die Möglichkeiten mehrere Seiten innerhalb einer Präsentation zu erstellen. Wie man in Abbildungen 6.4 und 6.5 sehen kann, hat man folgende Optionen und Bearbeitungsmöglichkeiten:

- ***Zurück***: durch die Präsentation nach hinten blättern
- ***Vor***: durch die Präsentation nach vorne blättern
- ***Neue Seite*:** eine neue leere Seite wird am Ende der Präsentation eingefügt
- ***Seite löschen***: aktuelle Seite wird aus der Präsentation komplett entfernt, die gesamte Seitenanzahl verringert sich um 1, Außer man hat die erste Seite ausgewählt, dann wird nur der Inhalt gelöscht und die Seitenzahl bleibt gleich.
- ***Seitenanzahl***: aktuelle Seite und Gesamtanzahl an Seiten

Die Folien werden in einer Liste, die aus mehreren Canvas besteht, abgespeichert. Wenn man durch diese Liste blättert, wird der Inhalt ausgewählter Seiten in einen Display-Canvas geladen. Sobald man weiter blättert, werden alle Veränderungen abgespeichert und eine neue Seite geladen. Mit der Schaltfläche *Neue Seite* wird ein neues Element am Ende der Liste eingefügt. Dieses neue Element wird als neue leere Seite am Ende der Präsentation

dargestellt. Bei *Seite löschen* wird die Seite und das dazugehörige Element aus der Liste komplett entfernt.

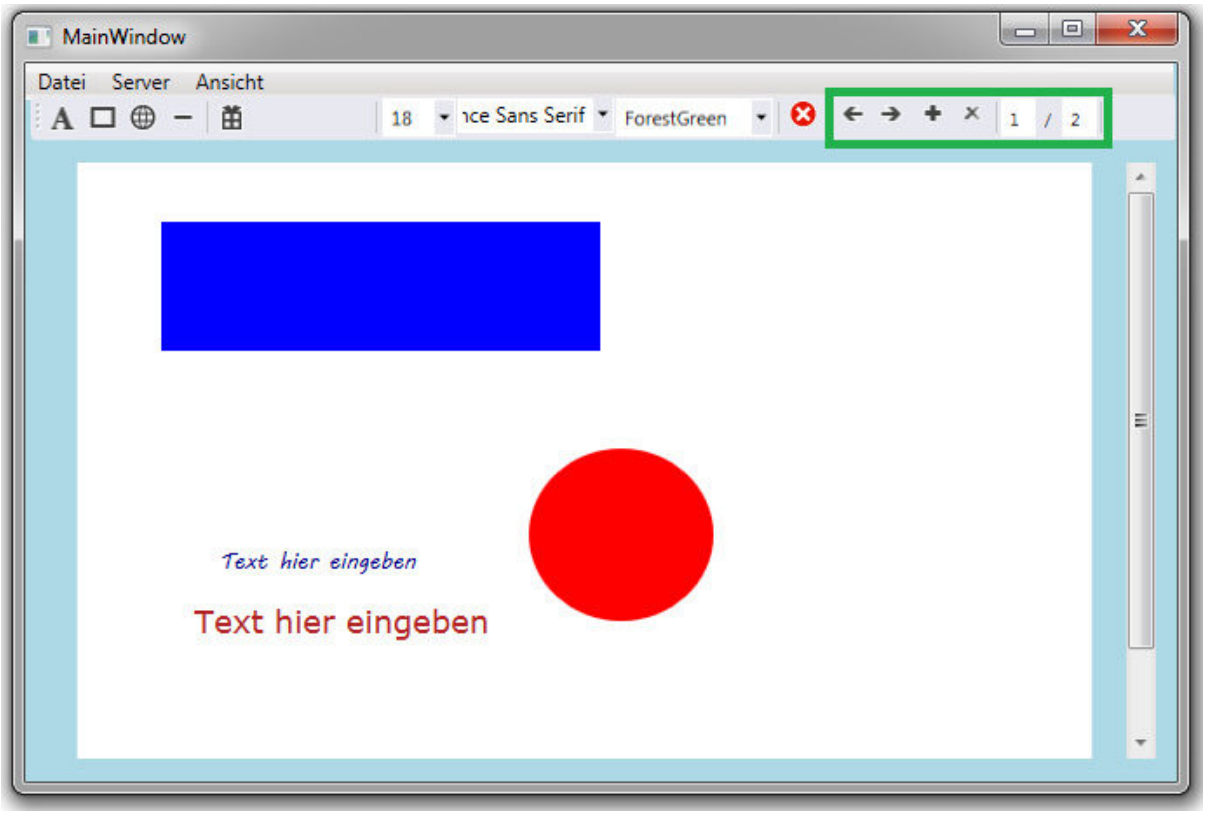

Abbildung 6.4 Präsentation Seite 1/2

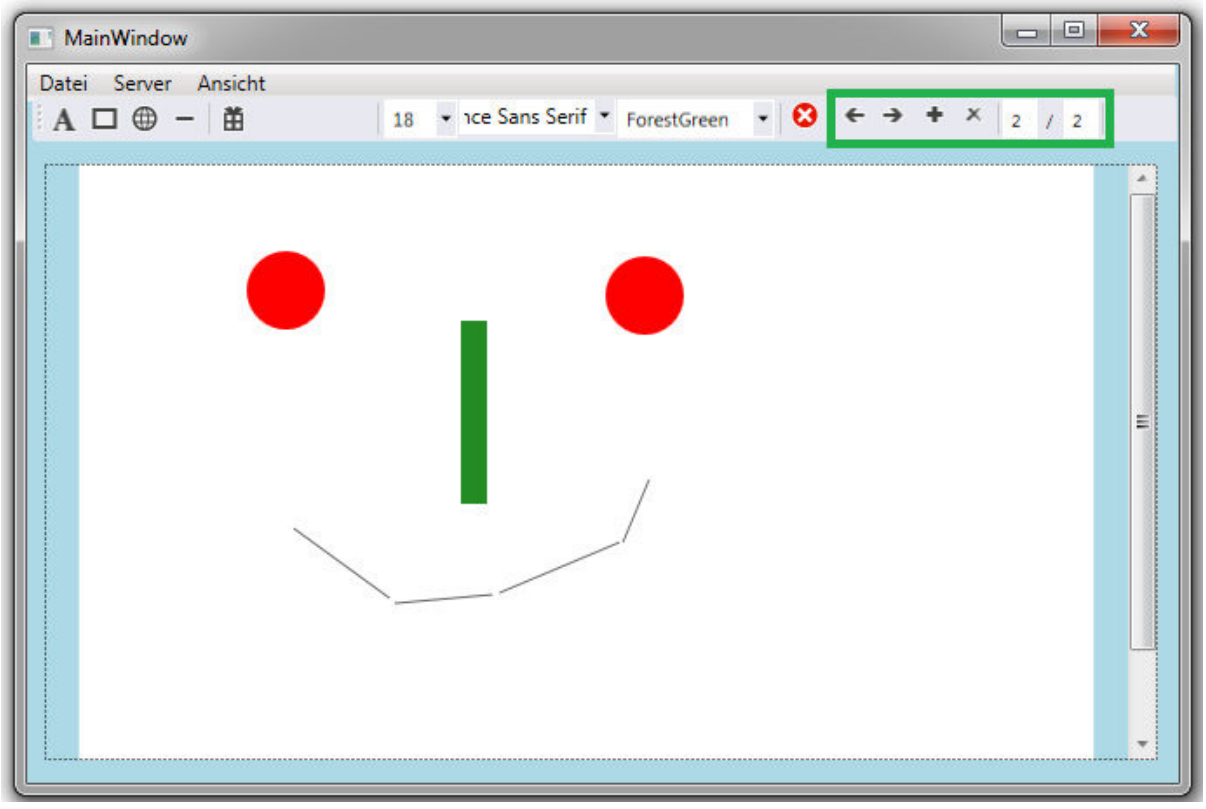

Abbildung 6.5 Präsentation Seite 2/2

Speichern und Laden

Zum persistenten Abspeichern der Präsentation geht man in der Menüleiste auf *Datei* → *Speichern* , siehe Abbildung 6.6. Danach öffnet sich ein Fenster, indem man den Namen und den Speicherort bestimmen kann, siehe Abbildung 6.7.

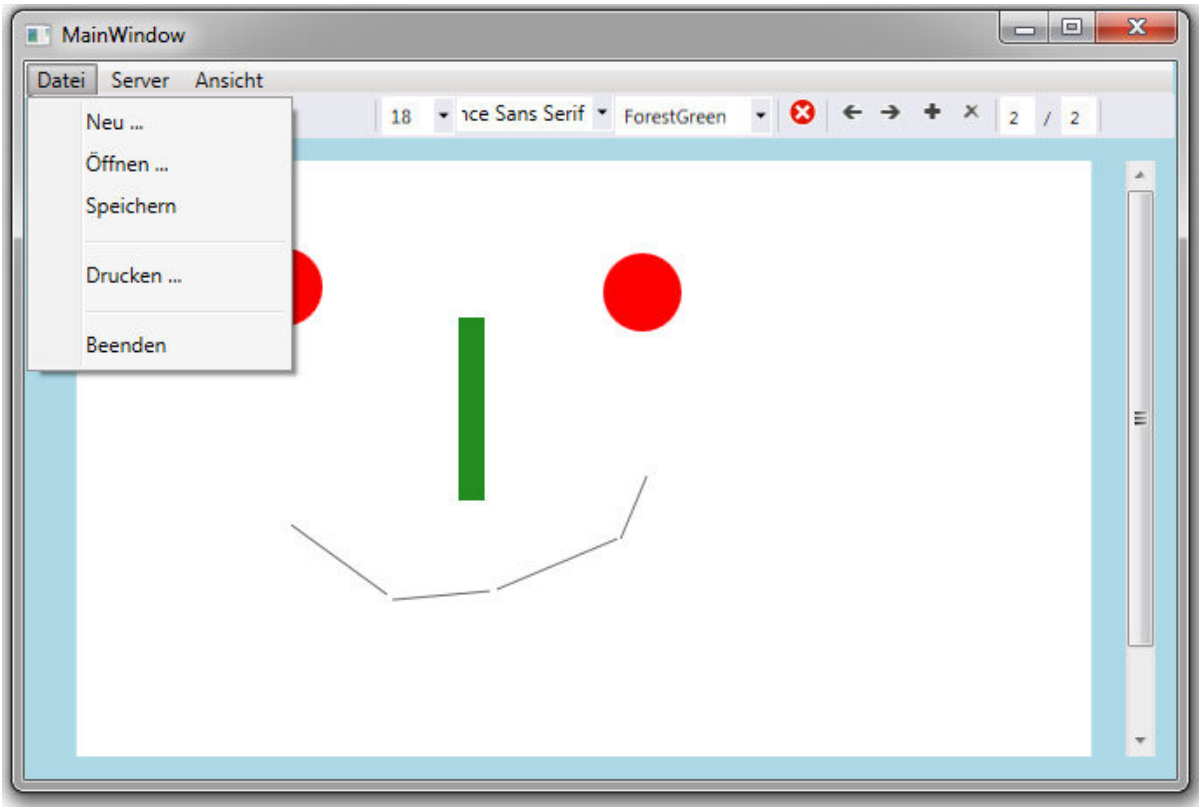

Abbildung 6.6 Speichern und laden

Die Präsentation besteht aus einer Liste und jedes Element dieser Liste entspricht einer Seite. Beim Speichern wird jedes Element(beziehungsweise Seite in der Präsentation) in eine XAML-Datei umgewandelt. Diese XAML-Dateien werden alle in eine ZIP-Datei verpackt. Alle Objekte wie Rechtecke oder Ellipsen sind in diesen XAML-Dateien enthalten. Die einzige Ausnahme bilden die PNG-Bilder, bei welchen der Verweis auf das Bild gespeichert wird, aber nicht das Bild selbst. Außer dem Verweis werden auch die Position und Größe abgespeichert. Damit werden die PNG-Bilder auch auf anderen PCs zur Verfügung gestellt und explizit in die ZIP-Datei mitverpackt. Beim Öffnen einer ZIP-Datei, die eine Präsentation enthält, werden die XAML-Dateien auf die Seiten gematched, die sie darstellen und die Bilder werden dann neu verlinkt. Außerdem kann der Server im Menüpunkt *Server* gestartet und wieder gestoppt werden. Im Menüpunkt Ansicht kann der Anwender Seitenansicht aufrufen und Foliengröße bestimmen.

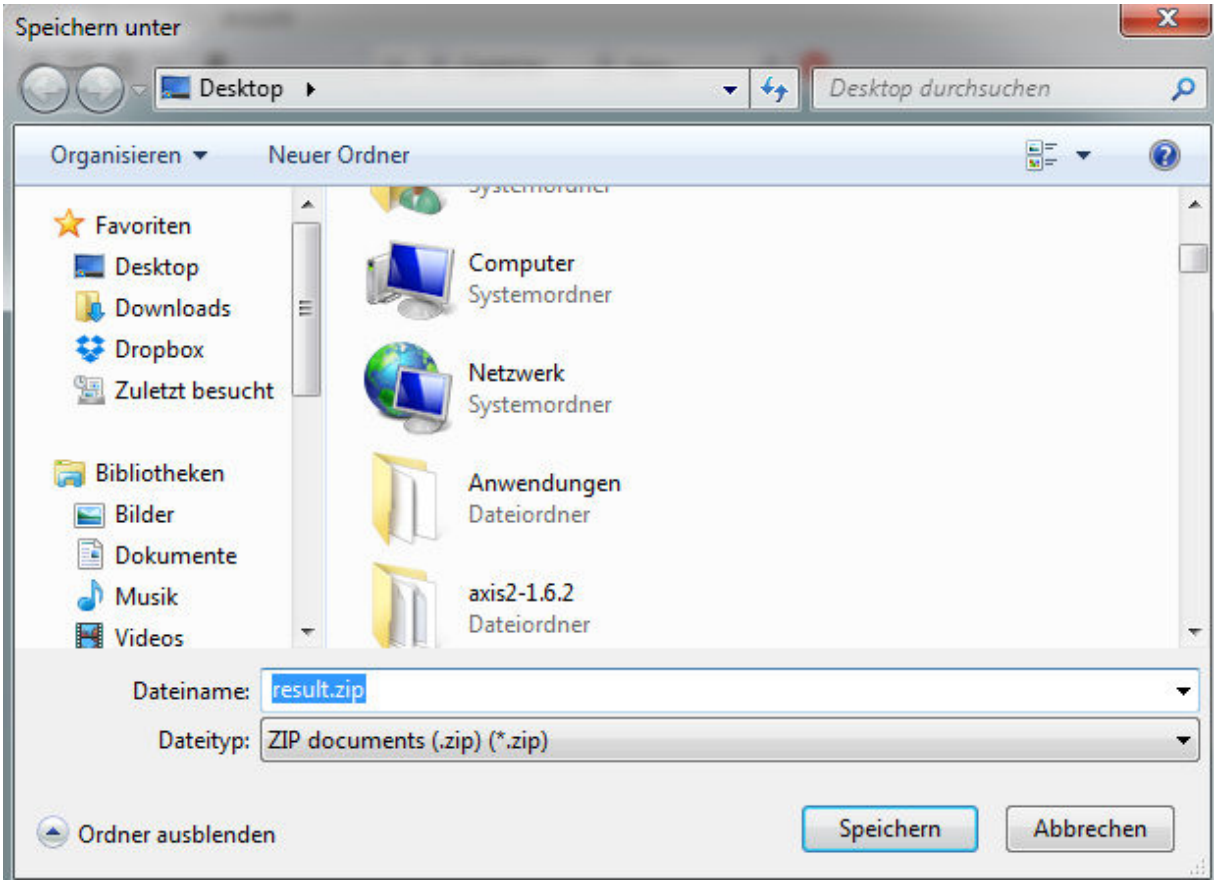

Abbildung 6.7 Name und Speicherort bestimmen

Literaturverzeichnis

[1] T. Ni, G. S. Schmidt, O. G. Staadt, M. A. Livingston, R. Ball und R. May, „A Survey of Large High-Resolution Display Technologies," in s *IEEE Virtual Reality Conference*, Alexandria, 2006.

[2] R. A. Ruddle, W. Fateen, D. Treanor, P. Sondergeld und P. Quirke, „Leveraging Wall-sized High-Resolution Displays for Comparative Genomics Analyses of Copy Number Variation," in s *IEEE Symposium on Biological Data Visualization*, Atlanta, Georgia, USA, 2013.

[3] G. Buchanan, S. Farrant, M. Jones, H. Thimbleby, G. Marsden und M. Pazzani, „Improving Mobile Internet Usability," in s *WWW10*, Hong Kong, China, 2001.

[4] Chris, „ifun," ifun Apple-News seit 2001, 14 März 2013. [Online]. Available: http://www.ifun.de/apps-starten-suchen-und-vieles-mehr-der-mac-produktivitats-boosteralfred-. [Zugriff am 13 Juni 2014].

[5] G. W. Furnas, „Generalized Fisheye Views," in s *ACM Conference on Human Factors in Computing Systems (CHI)*, Boston, Massachusetts, 1986.

[6] H. S. Thompson, D. Beech, M. Maloney und N. Mendelsohn, „XML Schema Part 1: Structures Second Edition," W3C, 2004 Oktober 28. [Online]. Available: http://www.w3.org/TR/xmlschema-1/. [Zugriff am 13 Juli 2014].

[7] A. Kühnle, Visual C# 2012. Das umfassende Handbuch, Bonn: Galileo Press, 2012.

[8] W. Schmidt, „XAML Overview," Microsoft Developer Network, 23 April 2010. [Online]. Available: http://msdn.microsoft.com/en-us/library/cc189036%28VS.95%29.aspx. [Zugriff am 13 Juni 2014].

[9] „Pixelausrichtung in WPF-Anwendungen," Microsoft Developer Network, November 2007. [Online]. Available: http://msdn.microsoft.com/de-DE/dede/library/vstudio/aa970908%28v=vs.90%29.aspx. [Zugriff am 13 Juni 2014].

[10] „Exemplarische Vorgehensweise: Erstellen eines Layouts auf Grundlage der absoluten Positionierung," Microsoft Developer Network, November 2007. [Online]. Available: http://msdn.microsoft.com/de-de/library/bb514508%28v=vs.90%29.aspx. [Zugriff am 13 Juni 2014].

[11] D. Vuyka, „WPF. Simple adorner usage with drag and resize operations," Denis Vuyka Research and Development, 15 Oktober 2007. [Online]. Available: http://denisvuyka.wordpress.com/2007/10/15/wpf-simple-adorner-usage-with-drag-and-resize-operations/. [Zugriff am 13 Juni 2014].

[12] Igkutikov, „Dragging Elements in a Canvas,“ Code Project, 8 Juni 2012. [Online]. Available: http://www.codeproject.com/Articles/387977/Dragging-Elements-in-a-Canvas. [Zugriff am 13 Juni 2014].

[13] „Image-Klasse,“ Microsoft Developer Network, [Online]. Available: http://msdn.microsoft.com/de-de/library/system.windows.controls.image%28v=vs.110%29.aspx. [Zugriff am 13 Juni 2014].

[14] „Arrays,“ Microsoft Developer Network, [Online]. Available: http://msdn.microsoft.com/de-de/library/9b9dty7d.aspx. [Zugriff am 13 Juni 2014].

[15] „ArrayList-Klasse,“ Microsoft Developer Network, [Online]. Available: http://msdn.microsoft.com/de-de/library/system.collections.arraylist%28v=vs.110%29.aspx. [Zugriff am 13 Juni 2014].

[16] „List<T>-Klasse,“ Microsoft Developer Network, [Online]. Available: http://msdn.microsoft.com/de-de/library/6sh2ey19%28v=vs.110%29.aspx. [Zugriff am 13 Juni 2014].

[17] Microsoft Developer Network, [Online]. Available: http://msdn.microsoft.com/de-de/library/system.runtime.serialization.datacontractserializer%28v=vs.110%29.aspx. [Zugriff am 13 Juni 2014].

[18] „XamlWriter-Klasse,“ Microsoft Developer Network, [Online]. Available: http://msdn.microsoft.com/de-de/library/system.windows.markup.xamlwriter%28v=vs.110%29.aspx. [Zugriff am 13 Juni 2014].

[19] „XmlReader-Klasse,“ Microsoft Developer Network, [Online]. Available: http://msdn.microsoft.com/de-de/library/system.xml.xmlreader%28v=vs.110%29.aspx. [Zugriff am 13 Juni 2014].

[20] S. Arouje, „Multi client Asynchronous TCP Server,“ Sony Arouje Blog, [Online]. Available: http://sonyarouje.com/2011/11/25/multi-client-asynchronous-tcp-server/. [Zugriff am 13 Juni 2014].

[21] „System.Net.Sockets-Namespace,“ Microsoft Developer Network, [Online]. Available: http://msdn.microsoft.com/de-de/library/vstudio/system.net.sockets. [Zugriff am 13 Juni 2014].

[22] „TcpListener-Klasse,“ Microsoft Developer Network, [Online]. Available: http://msdn.microsoft.com/de-de/library/vstudio/system.net.sockets.tcplistener. [Zugriff am 13 Juni 2014].

[23] „Socket-Klasse,“ Microsoft Developer Network, [Online]. Available: http://msdn.microsoft.com/de-de/library/vstudio/system.net.sockets.socket. [Zugriff am

13 Juni 2014].

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.
Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.
Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.
Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.
Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Stuttgart, den 15. Juli 2014 _____