Hochschule
für Technik
Stuttgart

Bachelorarbeit

Konzeption und Realisierung einer Webanwendung zur Visualisierung von Positionsdaten

im Studiengang Wirtschaftsinformatik
der Fakultät Vermessung, Informatik und Mathematik
Sommersemester 2018

Name: Denise Müller
Matrikelnummer: 355097

Zeitraum: 03.04.2018 - 03.07.2018
Prüfer: Prof. Dr. Jan Seedorf
Zweitprüfer: M. Sc. Kevin Erath

Firma: IT-Designers GmbH
Betreuer: M. Sc. Kevin Erath

Abstract

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

MVVM Model-View-ViewModel

MVC Model-View-Controller

MVP Model-View-Presenter

HTML Hypertext Markup Language

CSS Cascading Style Sheets

DOM Document Object Model

CLI Call Level Interface

SPA Single-page Application

UI User Interface

JSON JavaScript Object Notation

MASTeR Mustergültige Anforderungen - die SOPHIST Templates für Requirements

C.A.M. Ceiling Automotiv Monitoring

1

1 Einführung

Dieses Kapitel dient dazu, die Arbeit in ihrem thematischen Umfeld einordnen zu können. Dazu wird zunächst ein Einblick in das Aufkommen der ersten Ideen, die Entwicklung und dem aktuellen Stand des autonomen Fahrens gegeben. Anschließend wird auf den Inhalt des Projektes und den Aufbau der Arbeit eingegangen.

1.1 Motivation

Moderne Fahrerassistenzsysteme unterstützen heutzutage den Fahrer in jeder erdenklichen Situation. Ob die Einhaltung der Fahrspur, Abstandsregler zum vorderen Auto oder die Einparkhilfe, alle Assistenzsysteme tragen zum autonomen Fahren bei. Durch die immer größer werdende Digitalisierung[1], werden auch die Assistenzsysteme immer weiter optimiert bis hin zum fahrerlosen Fahren, das bis 2025 realisierbar sein wird[2].

1.2 Ziel der Arbeit

Durch vorhergehende Studentenprojekte wurden Hot Wheels A.I. Modellautos im Maßstab von 1 zu 32 mit eigene Hardware und Firmware modifiziert. Diese können sich bereits auf der dazugehörigen Fahrbahn anhand eines Graustufenverlaufs autonom bewegen. Zusätzlich ist eine manuelle Ansteuerung über Bluetooth möglich. Zu entwickelndes Gesamtsystem.

Im Rahmen der Bachelorarbeit soll ein System entwickelt werden, welches mit Hilfe einer Deckenkamera die im Grundsystem Ceiling Automotiv Monitoring (C.A.M.) vorgestellten Autos erfasst und über eine individuell zusammengestellte Fahrbahn visuell darstellt. Dazu wird das Gesamtsystem in zwei Teilsysteme aufgeteilt. Jedes Teilsystem wird jeweils von einem Bacheloranten bearbeitet. Dabei sind die Verantwortlichkeiten für die Systemaufgaben entsprechend aufgeteilt. Da das System gemeinsam entwickelt wird, sind Teile dieser Arbeit gemeinschaftlich erarbeitet und von anderen Gruppenmitgliedern niedergeschrieben worden. Die Diagramme in Kapitel 5 für das Gesamtsystem wurde von Felix Grammling erstellt.

Teilsystem

Das Teilsystem der vorliegenden Bachelorarbeit umfasst die Visualisierung einer individuell zusammengestellten Fahrbahn mit Position von autonom fahrenden Modellautos und zugehörige Live Daten des getrackten Fahrzeugs. Die Informationen, wie zum Beispiel Koordinationspunkte der Fahrbahn oder die aktuelle Position der Fahrzeuge, werden über das erste Teilsystem ermittelt und zur Verfügung gestellt. Da die Aktualisierung der getrackten Position des Fahrzeuges in einem abgestimmten Zyklus geschieht, soll die Visualisierung optimiert werden und die Bewegungen des Fahrzeuges so reibungslos wie möglich sein. Es werden Algorithmen zur Fahrspurvisualisierung und neue Technologietrends und Werkzeuge wie Vue.js genutzt.

Des Weiteren soll eine Bluetooth Verbindung zu den Fahrzeugen hergestellt werden, um vorerst manuelle Änderungen wie zum Beispiel an der Geschwindigkeit vorzunehmen. Optional sollte die Visualisierung mehrere Fahrzeuge und Hindernisse auf der Fahrbahn darstellen.

Aufbau der Bachelorarbeit

In Kapitel 2.1 wird auf das Framework Vue.js eingegangen, mit dem die Webanwendung erstellt wird und das dazugehörige Entwurfsmuster Model-View-ViewModel (MVVM) in Kapitel 2.2 beschrieben.

Kapitel 3 werden die Anforderungen an das Gesamtsystem erläutert. Zu den funktionalen und nicht funktionalen Anforderungen an das Teilsystem wird in Kapitel 4 eingegangen.

Die Anforderungsanalyse in Kapitel 5 beinhaltet das Gesamtsystem mit Kontextdiagramm und die dazugehörigen Anwendungsfälle von Kapitel 3 in Kurz- und Langbeschreibung.

Kapitel 6 befasst sich mit der technischen Umsetzung des Teilsystems, dabei werden Mockups und die Architektur beschrieben.

Das Implementierungskapitel 7 gibt die Weise der Realisierung der Webanwendung wieder sowie Codebeispiele zur Verdeutlichung der verwendeten Technologien.

Zum Schluss in Kapitel 8 wird das Ergebnis und die Erfahrung dargelegt und bietet die darauf folgende Schlüsse an.

2 Theoretische Grundlagen

Dieses Kapitel befasst sich mit dem Framework Vue.js, das clientseitige Frontend Anwendungen unterstützt und dem Entwurfsmuster MVVM, welches von Vue.js angewandt wird. Vue.js wird für die praktische Ausarbeitung der Bachelorarbeit genutzt und die Grundlagen sollen zum Verständnis der vorliegenden Bachelorarbeit dienen.

2.1 VueJS

Durch zunehmende Digitalisierung und die Nutzung sozialer Netzwerke kommen Sprachen und Technologien wie Hypertext Markup Language (HTML)[1]5 , Cascading Style Sheets (CSS)[2] oder JavaScript universell zum Einsatz. Egal welches Gerät gerade genutzt wird, jede Art von Anwendung sollte auf den Endgeräten lauffähig sein[3]. Für Anwendungen im Web wird ein Client als Browser und ein Webserver benötigt. Um die Daten, die durch HTMLangezeigt werden, zu manipulieren, interpretieren und zu aktualisieren, wird die Skriptsprache JavaScript benötigt[4]. Ebenso Anwenderinteraktionen wie Scrollen oder Klicken wird durch JavaScript interpretiert und verarbeitet.

2.1.1 Framework

Evan You ist der Gründer eines JavaScript Frameworks **Vue.js**[5]. Vue.js ist ein clientseitiges Framework, mit dem Webanwendungen in JavaScript komponentenorientiert entwickelt werden können. Dies bedeutet, dass die Anwendung in kleine Teile unterteilt wird, die miteinander kommunizieren und als eigenständige Module laufen. Das Framework setzt die Implementierung einer Webanwendung mit dem Architekturmuster MVVM durch und bietet hierdurch eine schlanke, universelle, anpassungsfähige und performante Implementierung[3]. Das Interesse an Vue.js ist in den letzten Jahren immer mehr gestiegen, was sich auch in den Google Trends zeigt[6].

1 Sprache, die es ermöglicht, Informationen im Internet zu präsentieren
2 Stilsprache, die das Aussehen von HTML-Dokumenten definiert

Aufbau

Der Aufbau gleicht dem von Angular.js oder React.js. Diese Frameworks werden in weiteren Abschnitten etwas genauer erläutert werden. Eine Anwendung besitzt mehrere Komponenten, die an sich einheitlich sind, aber von den anderen Komponenten zu trennen sind und über eine bestimmte Schnittstelle kommunizieren. Für die Benutzeroberfläche wird HTML benutzt, welche durch Attribute erweitert werden kann. Die Schnittstelle zur Kommunikation der Komponente wird an diese Attribute durch Datenbindung und Events realisiert. In diesem HTML Dokument stellt Vue.js viele Hilfsmittel für Styling und Formulare bereit. Um weitere Funktionen hinzuzufügen, werden Plug-ins in die Anwendung eingebaut[5]. Beispiele hierfür sind vue-router oder vue-custom-element.

```
<!DOCTYPE HTML>
...
 <BODY>
   <DIV ID="APP">
     <H1> {{MSG}}</H1>
  < DIV>
   <SCRIPT>
      NEW VUE({
         EL: "#APP",
         DATA() {
            RETURN {
              MSG: "HELLO WORD"
            }
         }
      });
  </SCRIPT>
</BODY>
```

Abb. 2.1: Codebeispiel Initialisierung Vue

Zuerst muss ein Standard HTML Dokument erstellt werden. Es enthält ein `<head>` und ein `<body>` Tag. Innerhalb des `body` Tags wird ein `div` mit einer Identifikation erstellt, beispielsweise wie in Abbildung 2.1 mit `id="app"`. Die Anwendung wird dort integriert. Um Daten wie einen String anzuzeigen, wird ein beliebiger Tag erstellt (zum Beispiel `h1`), der innerhalb einen Identifier besitzt, der in einer geschweiften Klammer (`<h1> {{msg}} </h1>`) steht. Um die `msg` mit einem String zu befüllen, wird ebenfalls innerhalb des `body` Tags ein `<script>` Tag definiert. Hier wird eine Instanz der Vue erstellt. Die Instanz besitzt das Attribut `el`, welches für die Identifikation zuständig ist, heißt, hier sollte `#app`

stehen. Des Weiteren hat die Vue Instanz eine `data()` Funktion, die Objekte, wie die `msg`, zurück gibt.

Model-View-ViewModel

Das Ausführen des Architekturmuster MVVM in Vue.js hat eine deutliche und klare Abgrenzung der Komponenten und kann durch das einfache Beispiel in Abbildung 2.2 veranschaulicht werden.

Abb. 2.2: MVVM in VueJS[1]

Die Benutzerschnittstelle, die dem Anwender dargestellt wird, wird in HTML erfasst. In Vue.js wird dabei ein *div* mit einer Identifikation, im Beispiel der Abbildung 2.2 `"app"`, erstellt. In dem div soll ein Text stehen, der durch das ViewModel übergeben werden soll. Dieser ist, wie auch in Angular oder ähnlichen Templating Frameworks, mit zwei geschweiften Klammern geschrieben. Der Text, der an diese Stelle eingefügt werden soll, wird im Model deklariert. Es wird ein Objekt, mit einem Attribut `Text`, in dem Beispiel 2.2 ein String, erstellt. Das Attribut muss genauso bezeichnet werden wie das Wort innerhalb der geschweiften Klammer im HTML Dokument.

Das Objekt wird an das ViewModel übergeben und dort in die Vue Instanz übertragen. In der Vue Instanz muss die Identifikation des `div` stehen und das Objekt, welches an die Oberfläche gegeben werden soll. Wenn das ViewModel die Instanz an das View übergibt, erkennt die View die Identifikation und sucht im entsprechenden HTML Dokument nach der passenden Identifikation. Ist das `div` gefunden, werden die Daten durchsucht und sobald dann das Attribut `Text` gefunden wurde, wird dies an der Stelle mit dem gleichen Attributsnamen eingesetzt.

2.1.2 Funktionsweise/Anwendung

Routing

In Vue.js können Seiten durch Routing im HTML Dokument verlinkt werden. Die Seiten werden hierzu vorher definiert. Dies kann die Reaktionsfähigkeit der Webanwendung deut-

1 in Anlehnung an: [7]

lich verbessern, da die Seite dynamisch auf dem Kontext der Seite aufgebaut ist, bedeutet, dass die Seiten einmal geladen werden und sonst nur angezeigt werden müssen und aktualisiert werden, wenn sich beispielsweise Daten ändern. Dabei müssen die verschiedenen Views oder Seiten unterschieden werden. Vue.js unterstützt hierfür eine Router Bibliothek, **vue-router**[8].
Um die verschiedenen Seiten zu definieren und zu verlinken, wird eine Instanz des Routers erstellt, in der ein oder mehrere Routen übergeben werden. Die Definition eines Routers ist ein Array, das mehrere Routen enthält, mit dem Attribut `path`, das die Verlinkung zur passenden Seite enthält. Diese Instanziierung wird an die Vue-Instanz übergeben. Um die Implementierung zu vollenden, müssen im HTML Dokument die Routen aufgeführt werden um sie im Web wiederzugeben. Das geschieht wie folgt: `<router-view></router-view>`[9].

Templating

Templating ist die Vordefinierung eines Designs oder eines Formats für ein Dokument. Dies können Vorlagen in Word oder Vorlagen für HTML Dokumente und deren Design und Funktionen von einzelnen Komponenten sein. Diese Vordefinierung ist universell und kann mit beliebigen Daten gefüllt werden[10]. Templating erlaubt, Werte bzw. Daten vom Model in die View zu binden. Seit Version 2.0 wird auch JavaScript Templating mit HTML Templating in Vue.js unterstützt.
Das meist benutzte Symbol ist die doppelte geschweifte Klammer. Durch diese Art von Templating wird eine One-Way-Bindung vom Model zum Template aufgebaut. Mit einer One-Way-Verbindung können Daten von dem Model zum Template gesendet werden, aber nicht von dem Template zum Model[11]. Reaktive Datenbindung ist eine der Haupteigenschaften von Vue.js, sie speichert die Daten, wie Arrays oder JavaScript Variablen, in Verbindung mit dem HTML Dokument. Die One-Way Bindung, wie die Abbildung 2.2 zeigt, aktualisiert das HTML Dokument automatisch, wenn in JavaScript die `msg` manipuliert wird.

Um eine Two-Way Bindung zu erzeugen und beispielsweise in einem Input Feld die Daten zu ändern, muss an das `input` Tag ein **v-model** integriert und an die Identifikation des zu ändernden Wertes gebunden werden. In Abbildung 2.1 müsste unterhalb der Zeile `<h1> {{msg}} </h1>` ein Input Feld mit der Bindung angefügt werden (`<input v-model="msg">`)[12]. Wird ein neuer Text in das Input Feld geschrieben, um somit den Wert zu ändern, wird der Wert durch die reaktive Datenbindung automatisch und mit einer geringen Reaktionszeit aktualisiert.

Events

Die Autoren Etzion und Niblett erläutern in ihrem Buch „Event processing in action“ ein Event folgendermaßen:

> „An event is an occurrence within a particular system or domain; it is something that has happened, or is contemplated as having happened in that domain. The word event is also used to mean a programming entity that represents such an occurrence in a computing system[13].“

Darunter wird verstanden, dass ein Event dann auftritt, wenn beispielsweise ein Mausklick, ein Druck auf einen Bildschirm oder das Scrollen geschieht. Um ein Event in unser HTML Dokument etwa an einen Klick-Button zu binden, wird in Vue.js ein `v-on` verwendet. Hierbei können auf verschiedene Art und Weise ein Button oder Methoden, die ausgeführt werden sollen, gebunden werden.

Für einen Button ist die `click`-Methode die standardgemäße Weise des Events. Für das folgende Beispiel wird ein Button zum hochzählen einer Zahl verwendet: `<button v-on:click="counter += 1">counter</button>`[14]. Das Klick-Event wird an den Button gebunden, sodass die Variable `counter` beim Eintreten des Events verändert wird.

Statt das Event an ein Objekt zu hängen, kann `v-on` auch an Methoden gebunden werden, um eventuell komplexere Ausführungen durchzuführen. Dabei kann ebenfalls eine `click`-Methode verwendet werden, die auf den Namen der Methode, die ausgeführt werden soll, hinweist (`v-on: click="Methodennamen"`)[14]. Zusätzlich können Parameter in dem Methodennamen angegeben werden, die in der Methode interpretiert werden. Ebenso werden sogenannte *Modifier* von Vue.js unterstützt, um die immer wiederkehrenden Aufrufe während den Events handzuhaben. Modifiers sind Schlüsselwörter, die den Grad des Zugriffsrechts und die Sichtbarkeit auf Variablen, Funktionen oder Klassen regeln. Ein Beispiel ist das `preventDefault`, das typischerweise aufgerufen wird, wenn das standardgemäse Verhalten des Browsers verhindert werden soll [14].

Validation

Eine Validierung durchzuführen ist vor allem bei Formularen und Registrierungen wichtig. Die Validierung überprüft die Richtigkeit der Daten und die Erfüllung der gegebenen Anforderungen. Browser besitzten üblicherweise nativ die Validierung einer Form. Da jedoch jeder Browser Objekte unterschiedlich handhabt, ist die Vue.js basierte Validierung eine gute Lösung, um einheitlich zu arbeiten. Zu Beginn sollte in dem HTML Dokument ein `form` Tag vorhanden sein, indem die Felder eines Formulars hinzugefügt werden. Durch das schon bekannte `v-model` können Bedingungen an ein Attribut gebunden werden und in JavaScript anhand dessen verarbeitet werden. Standardgemäß kann bei Eingabe einer Zahl, das über das `type` Attribut überprüft werden kann, ein Minimum und Maximum angegeben, welches dann in Vue.js mit einem `v-if` überprüft wird und die passende Nachricht an den Anwender weiter gibt. Diese Validierung erfolgt hauptsächlich in JavaScript, clientseitig oder serverseitig. Dabei gibt es Plugins, wie Vee-Validate und Vuelidate, die schon bestimmte Regeln mit sich bringen und das Validieren in Vue.js vereinfachen[15].

Mit Vee-Validate wird clientseitig validiert und durch das Hinzufügen von einem `v-validate` Attribute, das auf das Model, das überprüft werden soll, verweist. Vee-Validate bringt eigene Regeln mit, die mit `data-vv-rules` auf das jeweilige Model angepasst werden können[16].

Vuelidate ist eine modelbasierte Validierung, was eine flexiblere, auf das Minimum reduzierte Möglichkeit der Validierung darstellt. In dem von Vuelidate eigenen $v Schema werden die Validierungsmöglichkeiten gespeichert (`this.$v[propertyName]`). Das kann in dem HTML Dokument durch `v-if="!$v.emailValue.required"` auf Vorhandensein überprüft werden. Ebenso kann eine eigene Validierung erstellt werden, wenn die gewünschte Überprüfung nicht vorhanden ist, indem sie als Funktion dargestellt wird[17].

Komponente

Komponenten sind im Allgemeinen gesprochen Bereiche bzw. Teile eines Systems, die zusammen arbeiten können. In Vue.js helfen Komponenten, das Standard HTML Dokument zu erweitern. Um ein Komponent zu erstellen, muss dies erstmals mit `Vue.component(tag, constructor)` registriert werden[18]. In dem Konstruktor wird die Funktion definiert, die beim Verwenden des Komponenten ausgeführt wird, bzw. die Optionen mit den beinhaltenden Daten für das HTML Dokument. Eine zwingende Option ist `data`. `data` sollte für die Wiederverwendbarkeit eine Funktion sein, die das unabhängige Objekt zurück gibt. Die Wiederverwendung kann per Name, der als `tag` übergeben wird, definiert werden[18].

2.1.3 Reactive Programming

Reactive Programmierung wurde in den letzten Jahren immer mehr zum Trend[19]. Reactive Programmierung ist ein Programmierstil, der vor allem in event gesteuerten und interaktive Anwendungen gut genutzt wird. Bekannte Unternehmen wie Amazon[1] und Netflix[2] benutzen bereits reaktive Programmierung. Dabei werden Zustandsänderungen für das System bekannt gemacht. Wenn zum Beispiel eine Funktion aufgerufen wird, die zwei Zahlen addiert, würde in dem üblichem Programmierstil (*imperativ*) die Funktion die Variable der Summe ändern. Wenn nach der Ausführung der Funktion die Variablen sich ändern würden, hätte das keinerlei Auswirkungen auf die Summenvariable. Bei einem reaktiven Programmierstil nimmt die Summenvariable den aktuellen Wert der beiden addierten Variablen an[20].

Reactive Programming ist ein Teil aus objektorientierter und ein Teil aus funktionaler Programmierung, das asynchrone und immutable Streams von Events beinhaltet. Die Streams werden miteinander kombiniert und werden von `Observables` „abgehört“. Jedoch muss

1 https://www.amazon.de/p/feature/j5d6uh4r8uhg8ep
2 https://help.netflix.com/de/node/68708

bei den `Observable` zwischen `Hot` und `Cold Observables` unterschieden werden. Cold `Observables` sind mit Listen vergleichbar, da wie gewohnt bei der Erzeugung die Größe fest steht und der Inhalt dementsprechend nicht verändert werden kann. Hot `Observables` sind das genaue Gegenteil, hierbei ist nicht bekannt, wie der Inhalt aussehen könnte oder wie groß das Objekt wird. Hierbei wird ein Zeitintervall erstellt, welches innerhalb der angegebenen Zeit ein Event sendet, an das sich das Objekt abbonieren (engl. *subscriben*) und somit Veränderungen der Daten registrieren kann.[21]. Diese ganze Verarbeitung läuft in der reaktiven Programmierung asynchon ab und macht die Programme effizienter. Bei einer asynchronen Verarbeitung wird im Gegensatz zur synchronen Verarbeitung nicht gewartet , bis das Ergebnis einer Methode zurück gegeben wird, sondern läuft im Code weiter. Das Ergebnis wird sobald es vorhanden ist zurück gegeben.

Reaktives System

Um Anforderungen, Ziele und den Aufbau eines Reaktives System zu definieren, wird ein Manifest benötigt. Beim Reactive Programmingwird dies als „*Reactive Manifesto*“ bezeichnet[22]. Oftmals werden reaktive Systeme als schneller und robuster bezeichnet, dass sich mit den vier Eigenschaften beschreiben lässt:

- **Responsive:** Die Antwortbereitschaft des Systems und die Erkennung von Bugs erfolgt unmittelbar[21]. Die Fehler können jedoch nur durch Abwesenheit einer Antwort erkannt werden, außerdem sollten hierfür die Zeit bis zur erwarteten Antwort eingegeben werden[22].
- **Resilient:** Die Widerstandsfähigkeit des Systems wird bewiesen durch die immer noch vorhandene Antwortbereitschaft während eines Systemfehlers oder von Ausfällen der Hard- oder Software. Erreicht werden kann dies durch Isolation von Komponenten, Eindämmung von Fehlern, Replikation von Funktionalitätem und das Delegieren der Verantwortungen[22]. Anfragen warten eine bestimmten Zeit auf eine Antwort, wenn diese nicht kommt, weiß der Service Bescheid und kann ihn zeitnah behandeln[21].
- **Elastic:** Elastisch bedeutet, dass das System während verschiedener Lastbedingungen einen konstanten Service liefert. Für die Erkennung von Veränderungen und die Reaktion hierauf, muss auch hier die Replikation von Funktionalitäten gegeben sein. Bei erhöhter Last werden die Replizierungsfaktoren darauf angepasst. Dabei sollten keine Einschränkungen vorhanden sein[22]. Cloud-Dienste sind eine gute Lösung, um die Problematik einzugrenzen und das System kosteneffektiv zu betreiben[21].
- **Message Driven:** Nachrichtenorientierte Systeme benutzen eine asynchrone Nachrichtenkommunikation. Dabei werden die Komponente, die miteinander kommunizieren, entkoppelt und isoliert. Fehler können an andere eventuell übergeordnete Komponente gesendet werden. Die Systeme lassen sich an die Nachrichten hängen,

was die Überwachung von Veränderungen verinfacht[21]. Die Nachrichtenüberwachung veranlasst einen Überblick über das Laufzeitverhalten des Systems und der übermittelten Nachrichtenflüsse[22].

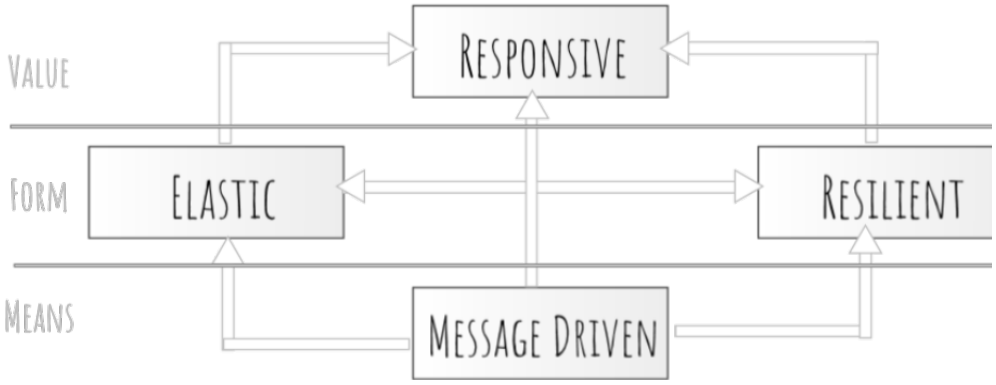

Abb. 2.3: Zusammenspiel reaktives System [1]

In Abbildung 2.3 wird das Zusammenspiel der vier reaktiven Qualitäten gezeigt. In größeren Anwendungen existieren mehrere Komponenten, die voneinander abhängig sind. Deshalb müssen die vier Qualitäten in jeder Ebene des Gesamtsystems berücksichtigt werden und sind somit innerhalb der Schichten kombinierbar[22]. Frameworks helfen bei der Umsetzung eines reaktiven Programms bei den Anforderungen, Eigenschaften und Ziele. Beispiele hierfür sind Vue.js und React.js.

2.1.4 Andere Frameworks

Analog zu Vue.js existieren weitere Bibliotheken bzw. Frameworks, die die Anforderungen ebenfalls als Reactive Programming betreiben und mit dem Entwurfsmuster MVVM oder Model-View-Controller (MVC) arbeiten. Ein Beispiel ist **React.js**. React.js ist eine von Facebook erstellte JavaScript Bibliothek, die dem Entwickler beim Erstellen von Oberflächen hilft. Whatsapp, Instagram und Facebook nutzen bereits React.js. Das Ziel, das React.js verfolgt, ist es einen einfacheren Code zu schreiben, um die einzelnen Bestandteile besser zu verstehen und weniger komplex zu halten. Die wesentlichen Bestandteile von React.js sind die Komponentenarchitektur, der virtuelle Document Object Model (DOM) und die Browserkompatibilität.
Die Komponente von React.js sind äquivalent zu den Web Komponenten, die mit `React.createClass` erstellt wurden. Innerhalb gibt es Funktionen wie etwa `render`, die HTML Dokumente für das Web präsentierbar machen und bei jeder Änderung aufgerufen werden. Dazu können eigene Funktionen definiert werden. Bei React.js wird die Standard `onClick` Methode als Attribut auf den Button gesetzt und durch eine geschweifte Klammer kann auf die

1 In Anlehnung an: [22]

Funktionen verwiesen werden (`<button onClick=this.add>`). Ein besonderes Attribut von React.js ist das `State`. Dieses Attribut beinhaltet die zu verändernden Daten und kann die Aktualisierung im HTML Dokument durchfüren. Somit muss auf die Anpassung des DOM keine Rücksicht genommen werden. Die Komponente werden in React.js innerhalb der `render` Methode definiert, da React.js JSX eine schlanke Syntaxerweiterung zum Schreiben von Markups verwendet. Änderungen am Stil der Seite oder des Buttons werden innerhalb der `createClass`, in der auch die `render` Methode für die Definition des HTML Dokuments implementiert wird, erstellt (`return backgroundColor: #fff;`). Hierbei wird erkannt, dass die übliche bekannte Trennung der Bereiche HTML, CSS und JavaScript nicht stattfindet[23].
Die Bearbeitung durch JSX in React.js wird in der Regel nicht direkt in dem DOM des Browsers stattfinden, sondern in einem virtuellen DOM, das zur Bearbeitung genutzt wird. Bei einer Veränderung wird jeweils ein neues Objekt, also ein neuer virtueller DOM, erstellt. Dabei wird der virtuelle DOM mit dem Browser DOM verglichen und aufgelistet. Die Änderungen werden erst zum Zeitpunkt des Batch an den Browser geschickt und aktualisiert[24]. Vue.js ist React.js in vielen Bereichen ähnlich. Beiden nutzen einen virtuellen DOM, eine reaktive, zusammensetzbare Benutzeroberfläche und detektieren Änderungen unmittelbar. Die folgende Auflistung zeigt die Unterschiede in Vue.js und React.js in der Leistung, im Templateing und JSX und der Skalierbarkeit des Systems.

- **Leistung:** Vue.js und React.js sind in hinsicht der Leistungin etwa gleich schnell. Damit ist dieser Punkt für die Entscheidung nicht relevant.[25].
- **Templating** Im Gegensatz zu Vue.js ist in React.js HTML und CSS zusammen mit JavaScript mit Hilfe von JSX geschrieben. Ebenso werden Werkzeuge wie die Typenüberprüfung in React.js besser unterstützt[25]. Die konkrete Trennung zwischen JavaScript, HTML und CSS ist in Vue.js deutlich und übersichtlicher[26]. Für viele Entwickler, die in HTML Erfahrung haben, ist es einfacher, den Code zu lesen und zu interpretieren[25].
- **Skalierbarkeit:** Für die Skalierung größerer Anwendungen gibt es in Vue.js sowie in React.js Routing Lösungen. In React.js gibt es die Frameworks Fluex oder Redux und in Vue.js das Vuex. Auch Redux kann in Vue.js integriert werden[27]. Vue.js beinhaltet einen Call Level Interface (CLI), das die Einbindung neuer Projekte mit Werkzeugen zum Bauen von Projekte wie webpack oder Browserify ermöglicht. In React.js müssen die Build Systeme, zum Bauen des Projekts, vom Entwickler angeeignet werden[25].

Ein weiteres Framework, das bei der Frontend Entwicklung hilft, **Angular.js**. Angular.js ist bekannter als Vue.js oder React.js. Viele Komponenten oder Syntaxen von Angular.js waren für Vue.js ein Vorbild, weshalb es vielen Angular.js Entwicklern einfacher fällt sich in Vue.js einzuarbeiten. Angular.js ist deutlich komplexer und besitzt mehr Vorgabe in Hinsicht der Softwarearchitektur als Vue.js. Ebenso ist die Leistung von Vue.js deutlich besser und leichter zu optimieren.

Die Entscheidung für Vue.js basierte auf der immer größerer werdenden Popularität[6]. Für die Größe des zu implementierenden Projekts ist Angular.js zu überladen. Vue.js oder React sind kompakt genug, für Webanwendungen zur Darstellung von Positionsdaten. Dabei ist die Leistung sehr entscheidend.
Eine Studie aus Schweden hat die Performance von Angular 2, Aurelia, Ember, Vue und weiteren verglichen. Dabei wurden die Befehle `Create`, `Delete` und `Update` mit jeweils 1000 Reihen getestet.[28]

- **Angular.js 2:** Im Ganzen hatte Angular 2 in jedem Testszenario die beste Leistung, dabei bbietet Angular 2 eine deutliche Verbesserung zu seinem Vorgänger 1.5[28].
- **Aurelia:** Aurelia hat in dem Szenario das zweitbeste Ergebnis erziehlt, jedoch schnitt Aurelia bei dem Befehl Update mit unter Anderem als schlechtestestes Framework ab[28].
- **Ember:** Eins der schlechtesten Frameworks ist Ember. Nur in dem Befehl Delete erreichte Ember einen durchschnittliches Ergebnis[28].
- **Vue.js:** Vue.js ist eins der schnellsten Frameworks. Vor allem bei den Befehlen Create und Update[28].

2.2 Architekturmuster Model-View-ViewModel

2.2.1 Motivation

Naveen Pete, ein Online-Blogger, bezeichnete ein Architekturmuster, auch Entwurfsmuster genannt, als ein „gut strukturiertes Dokument“, dass als Lösung für wiederkehrende Probleme dienen soll[29].

> „A design pattern is a well-documented solution to a recurring problem.“

Architekturmuster werden zur sauberen Trennung der Anwendungslogik und der Benutzeroberfläche genutzt. Das vereinfacht das Testen, die Instandhaltung und das Weiterentwickeln der Software. Außerdem verbessern Architekturmuster die Wiedernutzung des Codes für Andere Projekte, da lediglich der View Part plattformspezifisch angepasst werden muss[30]. Architekturmuster werden vor allem in großen Projekten genutzt, um eine Abgrenzung zwischen „Ansicht“ und „Modell“ zu schaffen. In dem Architekturmuster MVVM ist eine klare Abgrenzung zwischen der grafischen Oberfläche (View) und der Daten (Model) durch eine Schnittstelle (ViewModel) gegeben.

Model

In den üblichen Architekturmustern wird das Model als Abbildung der Datenquelle gesehen. Für das Architekturmodell MVVM ist das Model eine Abbildung der Daten für die Visualisierung. Diese Daten werden vom Benutzer manipuliert und zur Verfügung gestellt. Das Model stellt dabei folgende Funktionalitäten bereit:

- Validierung
- Benachrichtigungen bei Änderungen
- Verarbeiten nach vorgegebenen Regeln

Was hier zum Einsatz kommt, hängt von den Anforderungen an das Model ab. Werden die Daten beispielsweise von einem OR-Mapper oder einem Service zurück geschickt, könnten diese Funktionalitäten ebenso in das ViewModel implementiert werden[31].

View

Die View ist die strukturierte Benutzeroberfläche, in der Daten, Videos sowie Bilder dargestellt werden und die keinerlei Logik enthält. Benutzereingaben, beispielsweise über die Tastatur werden in der View abgefangen und an das ViewModel weiter gegeben. Die View ist ausschließlich mit dem ViewModel verbunden und das nur, wenn Daten an die View überreicht werden. Im Code sollte in der View so wenig wie möglich geschrieben werden und unabhängig sein. Das heißt, der Code sollte sich ausschließlich auf die View beziehen. Dies sollte das einfache Austauschen der View, ohne große Änderungen am Code zu leisten ermöglichen[32].

ViewModel

Einfach gesagt, stellt das ViewModel ein Model für die View dar und gibt es nach außen. Das heißt das ViewModel bearbeitet die Logik der View. Das ViewModel kommuniziert mit dem Model durch Methodenaufrufe und stellt die Daten, das das ViewModel vom Model geliefert bekommt, dar. Die zur Verfügung stehenden Funktionalitäten werden durch die View gebunden, wodurch für die View keinerlei Code anfällt. Referenzen auf Elemente der View dürfen nicht erstellt und darauf auch nicht zugegriffen werden, da, wie in dem Abschnitt der View bereits erwähnt wurde, keine Abhängigkeiten erstellt werden sollten. Durch Funktionen über die View zu testen ist unpraktisch und entfällt hier, da das ViewModel selbst die Abstraktion der View die Möglichkeit besitzt. Das ViewModel bezieht sich niemals auf die View und kann somit auf jede View bezogen werden und macht sich dafür wiederverwendbar[33].

2.2.2 Model-View-ViewModel

Durch die Entwicklung des Windows Presentation Foundation Framework (WPF) durch Microsoft[1] entstand 2005 das Architekturmodell MVVM. Es ist ein fester Bestandteil von Silverlight[2]. Ebenso werden existierende Frameworks um MVVM erweitert[34].

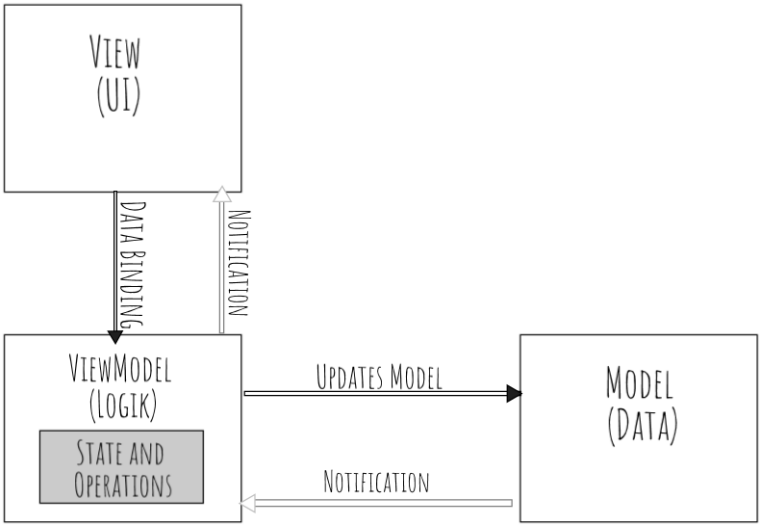

Abb. 2.4: Komponente des MVVM Architekturmusters[3]

Wie die drei Komponente View, ViewModel und Models miteinander zusammenhängen und kommunizieren, wird im folgenden erläutert und durch Abbildung 2.4 abgebildet.

Data-Binding

Wie vorher schon erwähnt wurde, kennt die View das ViewModel und das ViewModel kennt das Model, jedoch nicht vice versa, was die Pfeile in Abbildung 2.4 illustriert. Damit die View mit dem ViewModel interagieren kann, wird in WPF ein sogenanntes Data-Binding verwendet. Durch die Datenbindung wird eine Verbindung zwischen der View (User Interface (UI), die Benutzerschnittstelle) und dem ViewModel (Logik) hergestellt. Wenn das Model (Daten) die korrekten Benachrichtigung bereitstellt, können die eingebundenen Daten automatisch die Änderung des Wertes annehmen und in der View wiedergeben werden[35]. Eine Verbindung besteht zumeist aus vier Komponenten: einem Zielobjekt, einer Zieleigenschaft, die eine Abhängigkeitseigenschaft sein muss, einer Quelle und einem

1 https://www.microsoft.com/de-de
2 Silverlight is a cross-browser, cross-platform plug-in for delivering media and rich interactive applications for the Web
3 in Anlehnung an: [30]

Pfad zum Wert, der verwendet wird. Wenn eine Verbindung aufgebaut wird, ist die Richtung des Datenflusses entscheidend. Es gibt drei verschiedene Wege die Daten zu senden bzw. zu empfangen.

- One-Way Bindung
- Two-Way Bindung
- One-Way To Source Bindung

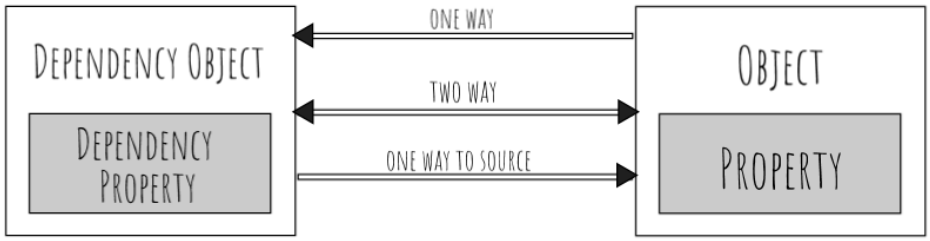

Abb. 2.5: Datenbindung zwischen View und ViewModel[1]

Fußnoten
Die View, besitzt ein Dependency(engl. für abhängig) Objekt mit einer Dependency Property, einer Eigenschaft, wie zum Beispiel einem String. Diese sind abhängig voneinander. Das ViewModel, enthält ein Objekt mit einer Property. Eine One Way Bindung lässt Änderungen an der Quelle zu und aktualisiert die Zieleigenschaften automatisch. Diese Verbindung ist dann nützlich, wenn schreibgeschützte Elemente in der UI vorhanden sind, denn Änderungen an den Zieleigenschaften sind bei einer One Way Bindung nicht möglich. Eine Two Way Bindung ermöglicht dem User, Änderungen vorzunehmen. Hierbei werden die Zieleigenschaften verändert und automatisch die Quelleigenschaften aktualisiert. Andersherum geschieht dasselbe, analog zur One Way Bindung. Die konträre Bindung zur One Way Bindung ist die One Way To Source Bidnung, diese aktualisiert automatisch die Quelleigenschaften wenn die Zieleigenschaft geändert wurde.

Notifications

Um die Änderung zu erkennen, muss die Quelle einen Benachrichtgungsmechanismus beinhalten. In der Regel ist dies die `INotifyPropertyChanged` Implementierung. Bei dieser Implementierung wird ein Objekt an die UI gebunden. Hier ist das Ereignis `PropertyChanged` enthalten, dies wird auslöst, sobald eine Änderungen an der Eigenschaft des Objektes stattfindet. Diese Änderung wird von dem Model an das ViewModel gesendet, das dann die Aktualisierung an der Quelleigenschaft vornehmen kann und die Änderung in der View angezeigt. Die Quellaktualisierungen werden anhand `UpdateSourceTrigger`, die Eigenschaften enthalten, die bestimmen, wann und warum die Benachrichtigung ausgelöst wird, ausgelöst.

Fazit

Das Ziel des Architekturmuster ist, wie auch bei dem Architekturmuster MVC, dem Vorreiter von MVVM, das Trennen der Logik und der Präsentationsschicht bzw. der View. Die Vorteile des Entwurfsmuster MVVM sind:

- Durch die Trennung müssen Änderungen, die an dem Model vorgenommen werden, nicht an der View geändert werden[36] .
- Während der Entwicklung können die Entwickler und die Designer unabhängig voneinander an den Komponenten arbeiten[29].
- Modultests für das ViewModel und das Model können ohne die View erstellt werden[29].
- Die View kann beliebig ausgetauscht und wiederbenutzt werden, da die View unabhängig von den anderen Komponenten ist[29].
- Die Weiterentwicklung sowie die Instandhaltung kann durch die Trennung detaillierter vorgenommen werden, ohne dass die Änderungen negative Auswirkungen auf das System haben[36].

Die Nachteile des Architekturmuster sind nicht von Relevanz. Für viele Entwickler ist MVVM für einfache Oberflächen zu mächtig und ebenso bei größere Fälle kann es zu Problemen bei der Konstruktion des ViewModel kommen. Wenn die Datenbindung komplexer wird, wird das Debugging problematischer[36].

2.2.3 Verwandte Architekturmuster

2.2.3.1 Model-View Controller

Das Hauptmerkmal des Architekturmusters MVC ist die Trennung der View und des Controllers. Das macht das MVC komfortabel für Web Anwendungen und weniger geeignet für Desktop Anwendungen[37]. Das Model ist beim Architekturmuster MVC der Ort, an dem die Daten und Objekte sowie der Netzwerkcode gespeichert und implementiert werden. Das heißt, der Model Part beinhaltet die Informationen, um die View zu veranschaulichen und die Logik, die die Änderungen des Users verarbeitet und die View damit benachrichtigt[38]. In der View ist ebenso wie in dem MVVM Architekturmuster keinerlei Logik enthalten, was die Wiederverwendbarkeit für andere Projekte erhöht[39]. Mit dem Controller zusammen definieren sie das UI (Benutzerschnittstelle)[38].
Der Controller vermittelt vorwiegend über „delegation pattern“ zwischen der View und dem Model. Delegation Pattern ist eine Technik, bei der Objekte Verhalten nach außen darlegen und das Verhaltens an ein anderes Objekt übertragen[40]. Idealerweiße kennt der Controller die View nicht und kommuniziert über ein bestimmtes abstraktes Protokoll[39]. Der Controller verarbeitet die Events der View und leitet sie an das Model weiter, das dann die View über die Änderung benachrichtigen kann und die View die Änderung dem Anwender darstellt. Vergleichbar mit dem ActionListener in Swing[41].

Model-View-Controller

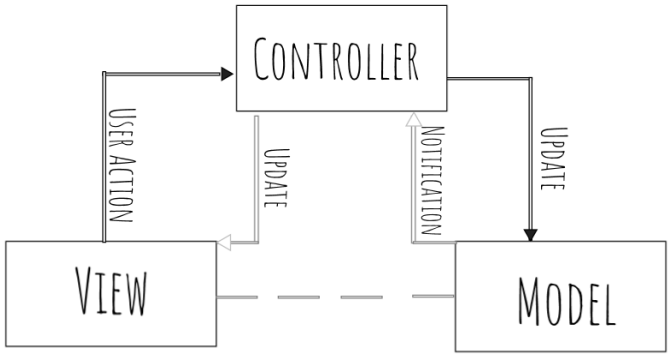

Abb. 2.6: Komponente des MVC Architekturmusters

In Abbildung 2.6 erkennt man die Zusammenhänge zwischen den Komponenten des Architekturmodells MVC. Dieses Modell lässt mehrere Views und Controller zu, die unabhängig von dem Model erstellt und modifiziert werden können[42]. Wenn der Anwender eine Aktion in der View auslöst, wird diese an den Controller geschickt, der anhand der Manipulation das Model aktualisiert. Sobald die Aktualisierung abgeschlossen ist, wird eine Benachrichtigung an den Controller gesendet, ob diese erfolgreich war oder nicht. War die Aktualisierung erfolgreich, aktualisiert der Controller anhand der Benachrichtigung die View.

Kommunikation

Obwohl das Model unabhängig von den Views ist, sollten die Komponente miteinander kommunizieren. Da die View jedoch das Model kennt, kann die View mit dem Model in Kontakt treten. Die Kommunikation mit den Nachrichten findet über **Events** (Aktionen) statt. Events stellen Mechanismen bereit, die mit wenigen Abhängigkeiten eine Kommunikation zustande kommen können.

Grafische Komponente, wie Listen, Textfelder oder ähnliche können Benachrichtigungen empfangen, beispielsweise wenn ein Klick stattfindet. Diese Benachrichtigungen kommen in der Regel vom Controller. Die View registrieren die Events an Objekte, die sie bearbeiten möchten. Wenn die Nachricht zum Model gesendet wurde, wird das Application Model auf diese Nachricht antworten, die Daten aktualisieren und eine Nachricht zurück senden. Das Application Model kann ebenso events auslösen und somit die abhängigen Views überwachen[43].

2.2.3.2 Model-View Presenter

Das Architekturmuster Model-View-Presenter (MVP) basiert auf dem MVC Muster. Die Komponente wurden anhand hohen Flexibilität angepasst und die Mängel des vorherigen Musters (MVC) werden besser gehandhabt. Das Hauptmerkmal des MVP Musters ist der Presenter, der direkten Zugriff auf die View und das Model besitzt und deren Zusammenspiel regelt. Die View und das Model können kleine Situationen selbst handhaben, was das ganze die Komplexität des Presenters reduziert, da dieser nur noch für die komplexen Anforderungen verantwortlich ist. Somit ist das Architektumuster MVP das flexibelste der MV* Familie, da es dem Entwickler viel Freiraum und Kontrolle gibt. Zum Beispiel kann die Datenbindung an dem Punkt an dem der Entwickler es wünscht genutzt werden.[37].

Die Views sind wie auch beim MVC Muster für die Anschauung der Daten zuständig und das Model für die Datenhaltung verantwortlich. Martin Fowler verglich 2006 das Architekturmuster MVC mit MVP so, dass der Controller in MVC bei MVP ein Teil der View ist[44]. Die Reaktion auf die Aktivitäten des Users sind Presenter enthalten. Er kann entscheiden wie das Model manipuliert und verändert werden kann, heißt er übernimmt bzw. integriert die Rolle des Application Model, das man von MVC kennt. Zusammenfassend zu sagen ist, dass der Presenter die Business Logik zur Verfügung stellt. In der Regel verhält sich die Kommunikation gleich wie beim MVC. Durch Aktionen des Anwenders, wie eine Mausbewegung, werden `Interactor Events` ausgelöst. Diese Interaktionen werden vom Presenter interpretiert[45].

2.2.4 Fazit

Bevor Software-Projekte mit Architekturmuster realisiert wurden, war die grafische Oberfläche in die Mitte der Anwendung implementiert. Dieser einfache Einsatz wird als Smart UI anti-pattern bezeichnet, das wie jedes Muster Vorteile hat, aber bei größeren Projekten zu Problemen führt. Das Gestalten der verschiedenen Architekturmustern bewältigen die Probleme des Smart UI anti-pattern. Jedes Architketurmuster ist ähnlich aufgebaut, jedes trennt die Benutzerschnittstelle mit den Daten und deren Verarbeitung. Die Kommunikation zwischen den drei Komponenten hängt von der grundlegenden Umgebung ab. Beispielsweise ist die Synchonisation mehrerer Views durch einen „`observer`", der die Änderungen weitergibt und somit die View aktualisiert, nützlich, aber in vielen Situationen nicht zugänglich. Somit sind auch andere Mechanismen brauchbar, wie die Datenbindung die in dem Architekturmuster MVVM zwischen dem ViewModel und der View. Die Anwendung des Mechanismus zur Kommunikation hängt von der Anwendung selbst, der Programmiersprache, der Frameworks, des angewendeten Musters und der persönlichen Preferenz ab[44]. In der folgenden Tabelle[37] werden die Vorteile und Nachteile des jeweiligen Entwurfsmusters. Dies zeigt, dass kein Gewinner ausgewählt werden kann, jeder hat seine eigenen individuellen Vorzüge.

Tab. 2.1: Vorteile und Nachteile der Architekturmuster

Muster	Vorteile	Nachteile
MVVM	Unterstützt mehrere Views für das gleiche Model und aktualisiert View und ViewModel automatisch	Das Benutzten von „Beobachter“ schwächt die Leistung. Beruht auf der zugrunde liegenden Technologie.
MVC	Gut geeignet für Web Anwendungen, da View und Controller hier getrennt gehandhabt wird.	Die Abhängigkeit von View und Controller, was hier nicht gegeben ist, sind in manchen Steuerelementen von nöten.
MVP	Flexibilität und Freiheit zum Verwenden der verschiedenen Mechanismen und kann an viele Anwendungsszenarien verwendet werden.	Keine strikte Trennung der Komponenten, das bei komplexere Codestellen zu Problemen führen kann.

3 Requirements Gesamtsystem

Im Folgenden werden die funktionalen und nicht-funktionalen Anforderungen an das Gesamtsystem beschrieben. Diese werden aus der Sicht des Auftragsgebers geschrieben und dienen dazu, die Anwendung genau zu definieren. Dadurch sollen Unstimmigkeiten vermieden werden, um ein Ergebnis zu erzielen, welches für alle beteiligten Parteien zufriedenstellend ist.

Zur Erstellung der Requirements wurde eine Schablone zur Unterstützung verwendet. Diese wurde von den SOPHISTen erstellt, welche sich mit dem Requirements Engineering auseinandersetzen. Die SOPHISTen haben ein Innovationsprojekt unter dem Namen „Mustergültige Anforderungen - die SOPHIST Templates für Requirements (MASTeR)“ durchgeführt, welches sich als Ziel die Verbesserung der natürlichsprachlichen Dokumentation von Anforderungen gesetzt hat. Daraus gingen Schablonen für funktionale und nicht-funktionale Anforderungen hervor, welche zur Erstellung der folgenden Anforderungen verwendet wurden[46].

Die SOPHISTen definieren ebenfalls eine rechtliche Verbindlichkeit für Anforderungen. Hierfür wurden von ihnen drei Schlüsselwörter festgelegt. Die wichtigsten Anforderungen werden mit dem Schlüsselwort *„muss“* definiert. Diese sind verpflichtend umzusetzen. Sind diese Anforderungen nicht umgesetzt, ist es möglich, die Abnahme des Produkts zu verweigern. Das zweite Schlüsselwort *„sollte“* wird verwendet, um Wünsche des Auftraggebers darzustellen. Diese müssen nicht erfüllt werden, erhöhen jedoch die Zufriedenheit des Auftraggebers. Das dritte Schlüsselwort *„wird“* beschreibt Vorbereitungen für zukünftige Funktionen der Anwendung. Sie dienen zur Dokumentation der Absichten des Auftraggebers.

3.1 Top-Level Requirement

- **Top Requirement 1000:**
  Das System C.A.M. soll die Lage eines Fahrzeuges über eine Deckenkamera auf der Fahrbahn ermitteln und zur Visualisierung bereitstellen.

3.2 Geforderte Anwendungfälle

- **Anwendungsfall Tracking:**
  Das System soll das Fahrzeug auf der Fahrbahn erfassen.
- **Anwendungsfall Darstellung:**
  Das System soll die Fahrzeugposition auf der Fahrbahn visuell darstellen.

Anforderungen an den Anwendungsfall "Tracking"

- **Requirement 2100:**
  Das System muss das Fahrzeug erkennen.
- **Requirement 2200:**
  Das System muss die Position des Fahrzeugs erkennen.
- **Requirement 2300:**
  Das System muss die Richtung des Fahrzeugs erkennen.
- **Requirement 2400:**
  Das System soll die Geschwindigkeit des Fahrzeugs erkennen.
- **Requirement 2500:**
  Das System muss die ermittelten Daten über eine Schnittstelle ausliefern.

Anforderungen an den Anwendungsfall "Darstellung"

- **Requirement 3100:**
  Das System muss als Webanwendung implementiert werden.
- **Requirement 3200:**
  Das System muss die Fahrbahn visualisieren.
- **Requirement 3300:**
  Das System muss anhand der übermittelten Daten das Fahrzeug auf der Fahrbahn visualisieren.

3.2.1 Anforderungen an die technischen Funktionen

Anforderungen an Start-up und Shut-down

- **Requirement 1100:**
  Nach dem Starten des Systems müssen alle Funktionen zur Verfügung stehen.

Anforderungen an die Fehlererkennung, -behandlung und -ausgabe

- **Requirement 1200:**
  Alle Fehler die während des Betriebs des Systems entstehen, sollten protokolliert und gespeichert werden.
- **Requirement 1210:**
  Geht das System in einen undefinierten, unsicheren Zustand über, sollte es automatisch in einen sicheren Zustand gebracht werden.

Anforderungen an die Kommunikation

- **Requirement 1300:**
  Es sollen die ermittelten Fahrzeuginformationen von dem Tracking zu der Darstellung mittels eines Websockets übertragen werden.
- **Requirement 1400:**
  Die externe Deckenkamera soll die Bilder über eine USB-Schnittstelle an das Systemübertragen.

Anforderungen an die Qualitäten

- **Requirement 1520:**
  Das System soll im Bereich Mehrfachdetektion von Fahrzeugen sowie von Hindernissen erweiterbar sein.
- **Requirement 1530:**
  Das System soll die Mehrfachdetektion von Fahrzeugen sowie von Hindernissen visuell erweiterbar sein.
- **Requirement 1540:**
  Das System soll die Fahrzeuge steuern können.

4 Teilsystem Visualisierung

4.1 Anforderungen an das Teilsystem

Requirements, in deutsch „Anforderungen“, beschreiben die Ziele eines Kunden, die angeforderten Funktionalitäten und Eigenschaften des Systems, sowie in welcher Qualität dies umgesetzt werden muss[47]. Die Beschreibung, das Prüfen, das Verwalten sowie die Analyse solcher Requirements wird als des Requirements Engineering beschrieben[48].

Dieses Kapitel definiert die funktionalen und nicht-funktionalen Anforderung an das Teilsystem. Die Anforderungen wurden nach dem selben Prinzip, welches zu Beginn in Kapitel 3 beschrieben wurde, erstellt.

Funktionale Requirements

Funktionale Requirements charakterisieren die Leistungen an das System von Anwendungsfällen. Zusätzlich definieren die funktionalen Requirements die technischen Funktionen eines Systems, die als Lösung für die Problematik der Anwendungsfälle fungieren[49].

Funktionales Top Level Requirement

Requirement 1000:
Das System soll Anwendern eine Fahrbahn visualisieren und die darauf vorhandene Position eines Fahrzeugs aufzeigen.

Requirements an die Anwendungsfälle

- **Requirement 1110:**
  Der Anwender kann das Fahrzeug vom System aus ein- und ausschalten.
- **Requirement 1120:**
  Optional: Der Anwender kann durch Eingabe einer Zahl die Geschwindigkeit des Fahrzeugs anpassen.

Nicht funktionale Requirements

Nicht funktionale Requirements definieren Forderungen an den Lösungsbereich wie die Architektur, Technologien oder an die Qualität wie zum Beispiel die Performance. Beispiele hierfür sind sie Bedienbarkeit, das eingesetzte Betriebssystem, die verwendete Hardware und die genutzte Programmiersprache[49].

Anforderungen an den Nutzungskontext

Darstellung:

- **Requirement 1210:**
  Durch Intervalländerungen der Positionen, sollen die Änderungen zeitlich schnell aktualisiert werden.
- **Requirement 1220:**
  Die Darstellung des Fahrzeugs in der Webanwendung soll in einer reibungslose Bewegung dargestellt werden

Anforderungen an die Implementierung

Architektur:

- **Requirement 1310:**
  Die Anwendung muss eine Single-page Application (SPA)-Architektur aufzeigen.

Technologien:

- **Requirement 1320:**
  Das System muss mit der Frontend Bibliothek Vue.js umgesetzt werden.
- **Requirement 1330:**
  Durch die gegebene Zeit und der Strecke muss die Geschwindigkeit durch Berechnungen dargestellt werden.
- **Requirement 1340:**
  Die Verbindung zu den Fahrzeugen soll durch Web-Bluetooth verbunden werden.

4.2 Anwendungsfall Teilsystem

In diesem Kapitel wird das Teilsystem und der dazugehörige Anwendungsfall Visualisierung genauer mit Hilfe von Abbildung 4.1 beschrieben.

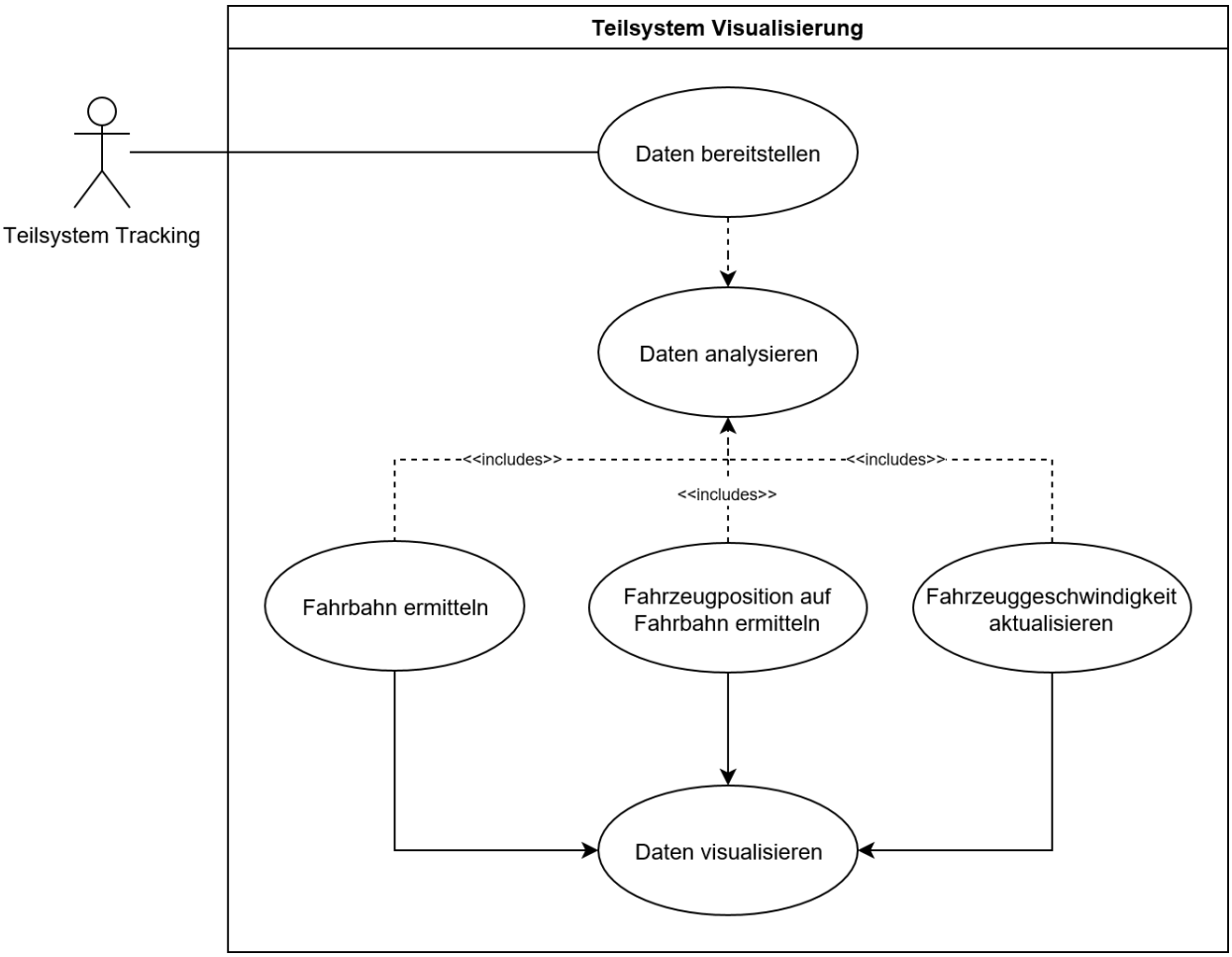

Abb. 4.1: Anwendungsfall Visualisierung

Das Teilsystem Visualisierung wird in einer Webanwendung mit Vue.js realisiert und umgesetzt. Dabei wird ein Koordinatensystem für die Positionsdaten verwendet. Ein **2D-Koordinatensystem**, das seinen Namen durch die zwei vorhandenen Dimensionen hat, dient zur subtilen Bezeichnung von Positionen anhand von Punkten in einem geometrischen Raum. Der Zahlenstrahl von links nach rechts ist die `x-Richtung` und von unten nach oben wird dies als `y-Richtung` bezeichnet. Das Koordinatensystem hat, obwohl der Zahlenstrahl bis in das Negative geht, seinen Ursprung in dem Punkt `(0/0)`[50]. Um die Visualisierung zu implementieren müssen folgende Schritte erfolgen.

Daten bereitstellen

Durch das Teilsystem Tracking werden Positionsdaten sowie weitere Informationen des Fahrzeuges über eine Kommunikationsschnittstelle gesendet. Die Daten werden als JavaScript Object Notation (JSON) Objekte übergeben und zur weiteren Verarbeitung bereitgestellt.

Daten analysieren

Bevor die Daten visualisiert werden können, müssen diese verarbeitet und gegebenfalls aufgerüstet, beispielsweise konvertiert werden.

Fahrbahn ermitteln

Die Fahrbahn wird anhand weiterer Positionsdaten oder eines Bildes analysiert und mit der Bibliothek `Canvas`[1] in einem Koordinatensystem mit `x` und `y` Koordinaten dargestellt.

Fahrzeugposition auf Fahrbahn ermitteln

Auf dem Koordinatensystem der Fahrbahn, wird die Position des Fahrzeuges anhand der bereitgestellten Positionsdaten ermittelt. Dabei zu beachten ist, dass die Werte nicht einfach zu übernehmen sind. Es muss darauf geachtet werden, dass die Daten der Fahrbahn und des Fahrzeuges auf dem gleichen Ursprung basieren, um eine genaue und richtige Visualisierung vorzunehmen.

Fahrzeuggeschwindigkeit aktualisieren

Die Geschwindigkeit (`v` für `velocity`) des Fahrzeuges wird ermittelt, durch das Dividieren von Weg (`s` für `Strecke`) und Zeit (`t` für `time`).

$$v = \frac{s}{t} \tag{4.1}$$

Die Berechnung der Geschwindigkeit wie in der Funktion 4.1 wird in einem kurzen Abstand wiederholt und wird in Bezug auf Vue.js und dessen Reaktivität in Echtzeit angepasst und dargestellt.

1 https://canvasjs.com/

Daten visualisieren

Nachdem die Daten analysiert und aufbereitet werden, werden diese in einer Webanwendung visualisiert und bei Änderungen angepasst. Die Visualisierung soll reibungslos und relativ in Echtzeit angezeigt und aktualisiert werden.

5 Anforderungsanalyse

5.1 Anwendungsfalldiagramme

5.1.1 Kurzbeschreibung der Anwendungsfälle

Anwendungsfall: „Darstellung“
Das System ermittelt anhand der bereitgestellten Daten aus dem Teilsystem „Tracking“, wo sich das Fahrzeug auf der Fahrbahn befindet und stellt dies in einer Webanwendung dar.

5.1.2 Langbeschreibung der Anwendungsfälle

Anwendungsfall: „Darstellung“
Initiator: Teilsystem Darstellung
Beteiligte Akteure: Teilsystem „Tracking“, Browser
Basisablauf: Das System erhält durch Anfragen an das Teilsystem „Tracking“ die zu analysierenden Daten. Anhand dieser wird die Fahrbahn in der Webanwendung visualisiert und die Position des Fahrzeugs darauf abgebildet. Die Webanwendung wird reaktiv mit einem dafür vorgesehenen Frontend Framework in einer SPA Architektur erstellt. Des Weiteren wird die Geschwindigkeit ermittelt und angezeigt.

6 Systementwurf

Nachdem die Anforderungen und Anwendungsfälle in den Kapiteln 3 und 4 beschrieben wurden, befasst sich der Systementwurf mit dem konkreten Plan für die technische Umsetzung des Systemes. Daher werden in diesem Kapitel der Entwurf des Systems mit Hilfe von Mockups, der Systemarchitektur und Beschreibungen der verwendeten Technologien veranschaulicht.

6.1 Mockups

6.2 Architektur

6.3 Verwendete Technologien

7 Implementierung

7.1 Analysieren und Auswerten der Tracking Daten

1. auswerten

7.2 Darstellung Fahrbahn

1. fahrbahn darstellen

7.3 WebSockets Kommunikation

1. webSockets

7.4 Bluetooth Verbindung zu Modellfahrzeugen

1. bluetooth

8 Zusammenfassung/Ausblick

Ergebnis-Bewertung, Erfahrung, Zusammenfassung und Ausblick

Literaturverzeichnis

[1] Gero Hesse. Digitalisierung der arbeitgeberkommunikation: Individualisierung und erlebnis sind gefragt, 2015. URL https://www.saatkorn.com/digitalisierung-der-arbeitgeberkommunikation-individualisierung-und-erlebnis-sin

[2] Don Dahlmann. Die fünf level des autonomen fahrens, 2018. URL http://www.dondahlmann.de/?p=24974.

[3] Marc Teufel. Vue.js: Der siegeszug eines frameworks. *entwicklerspezial, entwickler.de*, 2018. URL https://entwickler.de/leseproben/vue-framework-579832700.html.

[4] Benjamin Wallenborn Dirk Frischalowski. Einführung in java-webanwendungen. *Uni Hagen*. URL http://www.isdb.fernuni-hagen.de/wbt/files/demo/jsp/JSP/Kursseite_35130.htm.

[5] Sebastian Springer. Vue.js: Das framework im überblick. *PHP Magazin, entwickler.de*, 2017. URL https://entwickler.de/leseproben/framework-ueberblick-vuejs-579760880.html.

[6] Google Trends. Trendsvue, 2018. URL https://trends.google.de/trends/explore?date=today%205-y&q=vue.js,react.js,angular.js.

[7] Packt Video. Web development with vue.js 2, 2017.

[8] Ed Zynda. Getting started with vue router. *scotch.io*, 2017. URL https://scotch.io/tutorials/getting-started-with-vue-router.

[9] Joshua Bemenderfer. Introduction to routing in vue.js with vue-router. *Alligator.io*, 2017. URL https://alligator.io/vuejs/intro-to-routing/.

[10] Unknown, . URL https://www.thefreedictionary.com/Templating.

[11] Unknown. Vue.js templating. *Alligator.io*, 2016-2017. URL https://alligator.io/vuejs/templating/.

[12] Anthony Gore. Exploring vue.js: Reactive two-way data binding. *medium.com*, 2016. URL https://medium.com/js-dojo/exploring-vue-js-reactive-two-way-data-binding-da533d0c4554.

[13] Opher Etzion and Peter Niblett. *Event processing in action*. Manning, Greenwich, Conn., 2011. ISBN 1935182218. URL http://proquest.tech.safaribooksonline.de/9781935182214.

[14] Unknown. Event handling, 2018. URL https://vuejs.org/v2/guide/events.html.

[15] Unknown. Form validation, . URL https://vuejs.org/v2/cookbook/form-validation.html.

[16] Joshua Bemenderfer. Template-based form validation with vue.js and vee-validate, 2017. URL https://alligator.io/vuejs/template-form-validation-vee-validate/.

[17] Joshua Bemenderfer. Model-based form validation with vue.js and vuelidate, 2017. URL https://alligator.io/vuejs/model-form-validation-vuelidate/.

[18] Unknown. Components, . URL https://v1.vuejs.org/guide/components.html#What-are-Components.

[19] Engineer Bainomugisha, Andoni Lombide Carreton, Tom van Cutsem, Stijn Mostinckx, and Wolfgang de Meuter. A survey on reactive programming. *ACM Computing Surveys*, 45(4):1–34, 2013. ISSN 03600300. doi: 10.1145/2501654.2501666.

[20] Christian Lambert and Matthias Ebbing. Reaktiv in java mit rxjava: Reaktiv in die praxis: Reaktive programmierung mit rxjava, 2016. URL https://jaxenter.de/reaktiv-in-die-praxis-reaktive-programmierung-mit-rxjava-39088.

[21] Jan Carsten Lohmüller. Reactive programming – mehr als nur streams und lambdas, 2016. URL https://www.informatik-aktuell.de/entwicklung/programmiersprachen/reactive-programming-mehr-als-nur-streams-und-lambdas.html.

[22] Jonas Bonér, Dave Farley, Roland Kuhn, and Martin Thompson. Das reaktive manifest, 2014. URL https://www.reactivemanifesto.org/de.

[23] Paul Kögel. React einsteiger tutorial, 2015. URL http://reactjs.de/posts/react-tutorial.

[24] Marcin Skirzynski. Virtuelles dom mit react.js, 2015. URL http://reactjs.de/posts/virtuelles-dom-mit-react-js.

[25] Unknown. Comparison with other frameworks, 2018. URL https://vuejs.org/v2/guide/comparison.html.

[26] Toni Haupt. Vue.js – it's simple until you make it complicated, 2018. URL https://blog.codecentric.de/2018/02/vue-js-simple-make-complicated/.

[27] Kevin Peters. Large-scale vuex application structures, 2017. URL https://medium.com/3yourmind/large-scale-vuex-application-structures-651e44863e2f.

[28] Svensson Alexander. Speed performance comparison of javascript mvc frameworks, 2015. URL http://www.diva-portal.org/smash/get/diva2:998701/FULLTEXT03.pdf.

[29] Naveen Pete. Mvc, mvvm and angular. 2016. URL https://naveenpete.wordpress.com/2016/09/07/mvc-mvvm-and-angular/.

[30] Unknown. The mvvm pattern. *Microsoft*, 2012. URL https://msdn.microsoft.com/en-in/library/hh848246.aspx?irgwc=1&OCID=AID681541_aff_7593_1211691&tduid=(ir_VqY0n%3AR2F0Iazt90ZWSkDWZQUkjwTGy5NQ1xQg0)(7593)(1211691)()(ir)&clickid=VqY0n%3AR2F0Iazt90ZWSkDWZQUkjwTGy5NQ1xQg0&ircid=7593.

[31] Norbert Eder. Mvvm das model. 2017. URL https://www.norberteder.com/mvvm-das-model/.

[32] Norbert Eder. Mvvm die view. 2017. URL https://www.norberteder.com/MVVM-Die-View/.

[33] Norbert Eder. Mvvm das viewmodel. 2017. URL https://www.norberteder.com/MVVM-Das-ViewModel/.

[34] Lukas Jaeckle, Joachim Goll, Manfred Dausmann. Das architekturmuster model-view-viewmodel. 2015.

[35] Saisang Cai. Datenbindung. *Microsoft*, 2017. URL https://docs.microsoft.com/de-de/dotnet/framework/wpf/data/data-binding-wpf.

[36] Unknown. Mvvm tutorial. *tutorialspoint*, . URL https://www.tutorialspoint.com/mvvm/mvvm_advantages.htm.

[37] Artem Syromiatnikov and Danny Weyns. A journey through the land of model-view-design patterns. In *2014 IEEE/IFIP Conference on Software Architecture*. IEEE, 2014. doi: 10.1109/wicsa.2014.13.

[38] A. Leff & J.T. Rayfield, editor. *Web-application development using the Model/View-/Controller design pattern*, 2001. IEEE Comput. Soc.

[39] Rui Peres. Model-view-controller (mvc) in ios: A modern approach. 2016. URL https://www.raywenderlich.com/132662/mvc-in-ios-a-modern-approach.

[40] Alexander Schatten, Stefan Biffl, Markus Demolsky, Erik Gostischa-Franta, Thomas Östreicher, Dietmar Winkler. Delegation pattern. *Best-Practise Software Engineering*, 2013. URL http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/delegation.html.

[41] Leif Singer. Model-view-controller. *Uni Hannover*, 2004. URL http://www.se.uni-hannover.de/priv/lehre_2004winter_seminar_software_entwurf/04_mvc.pdf.

[42] Edward Curry and Paul Grace. Flexible self-management using the model-view-controller pattern. *IEEE Software*, 25(3):84–90, 2008. doi: 10.1109/ms.2008.60.

[43] John Deacon. Model-view-controller (mvc) architecture. *JOHN DEACON Computer Systems Development, Consulting & Training*, 1995, 2000, 2005.

[44] Matti Bragge. Model-view-controller architectural pattern and its evolution in graphical user interface frameworks, 2013. URL http://www.doria.fi/bitstream/handle/10024/92156/Model-View-Controller%20architectural%20pattern%20and%20its%20evolution%20in%20graphical%20user%20interface%20frameworks.pdf?sequence=2&isAllowed=y.

[45] Mike Potel. Mvp: Model-view-presenter. the taligent programming model for c++ and java. *Taligent, Inc*, 1996.

[46] SOPHIST GmbH. Sophisten, master - schablonen für alle fälle, 2013. URL https://www.sophist.de/fileadmin/SOPHIST/Puplikationen/Broschueren/SOPHIST_Broschuere_MASTeR.pdf.

[47] R.Heini. Ziele und anforderungen: Funktionale, nicht funktionale anforderungen, 2010. URL http://www.anforderungsmanagement.ch/in_depth_vertiefung/ziele_und_anforderungen/index.html.

[48] Sophist. Anforderungen, 1998-2011. URL http://www.sophist.de/anforderungen/requirements-engineering/faq-requirements-engineering/.

[49] Joachim Goll. *Methoden und Architekturen der Softwaretechnik*. Studium. Vieweg+Teubner Verlag, Wiesbaden, 1. aufl. edition, 2011. ISBN 978-3-8348-1578-1. doi: 10.1007/978-3-8348-8164-9. URL http://dx.doi.org/10.1007/978-3-8348-8164-9.

[50] Dennis Rudolph. x-y-koordinatensystem mit punkte, 2017. URL https://www.gut-erklaert.de/mathematik/x-y-koordinatensystem-mit-punkte.html.