Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Alexander Sawadski und Wlad Timotin

Entwicklung einer kollaborativen Enterprise Architecture Management Webapplikation mit Full Stack Javascript

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Alexander Sawadski und Wlad Timotin

Entwicklung einer kollaborativen Enterprise Architecture Management Webapplikation mit Full Stack Javascript

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Ulrike Steffens
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 15. April 2016

Alexander Sawadski und Wlad Timotin

Thema der Arbeit
Entwicklung einer kollaborativen Enterprise Architecture Management Webapplikation mit Full Stack Javascript

Stichworte
Full Stack, Javascript, Kollaboration, Enterprise Architecture Management, Webapplikation, ReactJS, FLUX, Mongoose, MongoDB, ArchiMate, Virtueller DOM

Kurzzusammenfassung
In dieser Bachelorarbeit wurde ein kollaboratives Modellierungstool entwickelt, das die Erstellung von Archimate Diagramme ermöglicht. Der Schwerpunkt lag dabei in der Realisierung der Webapplikation durch Anwendung neuer Technologien, wie die ReactJS, WebSockets, NodeJS und MongoDB. Dabei wurde ReactJS mit dem Architekturstil FLUX kombiniert.
Alexander Sawadski und Wlad Timotin

Title of the paper
Development of a collaborative enterprise architecture management web application with full stack javascript

Keywords
Full Stack, Javascript, Collaboration, Enterprise Architecture Management, Webapplication, ReactJS, FLUX, Mongoose, MongoDB, ArchiMate, Virtual DOM

Abstract
In this bachelor thesis a collaborative modelling tool has been developed, which allows to create archimate diagrams. The main focus was the development of a web application with help of new technologies like ReactJS, WebSockets, NodeJS and MongoDB. The usage of ReactJS was also combined with Flux architecture style.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Unternehmen wachsen täglich. Dementsprechend müssen deren Prozesse schneller an neue Ziele angepasst werden. Früher haben sich Unternehmen auf operative und taktische Lösungen orientiert. Probleme wurden kurzfristig gelöst, ohne das komplette System zu betrachten. Dadurch entstanden viele Redundanzen, Spaghetti-Code und nach längerer Zeit ein komplexes System. Das System eines global operierenden Unternehmens besteht aus Tausend IT-Applikationen, die Hundert verschiedene Software- und Hardware Plattformen beinhalten [Sch14, S. 186], die nicht mehr zu überblicken sind.

Die Einführung von EA in einem Unternehmen soll die Vorgehensweise ändern. Durch Visualisierungen der IT-Landschaften wird ein besseres Verständnis über das ganze System und ein erweiterter Blickwinkel gegeben. Viele Unternehmen haben sich für EA entschieden, aber viele Erwartungen wurden nicht erfüllt [Sch14, S. 186].

Eine Ursache könnte daran liegen, dass das System bis ins Detail gesteuert wird. Außerdem kann Silodenken die Arbeit erheblich beeinträchtigen [Sch14, S. 186]. Auch bei der entfernten Arbeit kommen Synchronisationsprobleme zum Vorschein.

Ein möglicher Lösungansatz für die genannten Probleme könnte kollaborative EA sein. Hier liegt die Kernidee darin, die Anzahl an Beteiligten auf die IT-Entscheidungen zu vergrößern. Dadurch Entstehen neue Probleme. Eines davon wäre den aktuellen Zustand des Diagramms bei allen Anwendern gleich zu halten. Ein anderes Problem ist die geografische Trennung. Es soll ermöglicht werden dass Anwendern mobil auf die Webapplikation zugreifen können. Daher ist die Einführung einer EA in einem Unternehmen komplex. Um diesen Prozess zu erleichtern wurden EA Frameworks und dazu passende Tools entwickelt. Nicht jedes Framework passt zu einem Unternehmen, da jedes Unternehmen unterschiedliche Ziele verfolgt. Sowie nicht jedes Tool zu einem Unternehmen passt, zum Beispiel auf Grund der Kosten, Benutzbarkeit oder Funktionalität. Zudem gibt es kein Tool das alle Frameworks anbietet. Wenn ein Unternehmen ein passendes Framework zu seinen Zielen gefunden hat, zu dem es kein Tool gibt, muss ein eigenes Tool entwickelt werden. Diese Arbeit befasst sich mit der Entwicklung eines neuen

EA Tools auf Grundlage der ArchiMate Spezifikationen welches die gestellten Anforderungen bearbeiten soll.

1.2 Zielsetzung

Das erste Ziel dieser Arbeit ist die Entwicklung einer Webapplikation für kollaborative Enterprise Architecture. Um dieses Ziel zu erreichen, soll zunächst eine Marktanalyse an EA Tools durchgeführt werden. Dabei wird die Auswahl der Tools auf die Modellierungssprache Archimate abgegrenzt. Die Tools werden nach Bedienbarkeit, Design und Funktionalität verglichen. Die Ergebnisse sollen die üblichen Funktionalitäten und Strukturen aufdecken. Daraus werden die Anforderung an die neuentwickelnde Webapplikation formuliert. Anhand der Anforderungen werden Mockups und daraufhin UML Diagramme erstellt. Diese werden bei der Realisierung verwendet. Anschließend wird ein Systemtest der Webapplikation von den Entwicklern durchgeführt.

Das zweite Ziel ist aufzuzeigen, wie viel Potenzial die neuen JavaScript Technologien bei der Entwicklung von Webapplikationen bringen. Hierbei wird sowohl das Frontend, wie auch das Backend in JavaScript geschrieben. Es werden mindestens zwei Prototypen mit verschiedenen Ansätzen entwickelt. Zudem werden bei der Evaluierung mindestens fünf Diagramme für jeden Prototyp erstellt. Die Zeiten der Funktionalitäten wird bei jedem Prototyp für jedes Diagramm gemessen und miteinander verglichen. Dadurch werden die Grenzen der Prototypen bestimmt. Anschließend wird der Prototyp mit dem besten Ergebnis mit mehreren Clients getestet.

1.3 Aufbau der Arbeit

Diese Arbeit ist in sieben Kapitel gegliedert, die folgendermaßen aufgebaut sind:

Im Kapitel 1, Einleitung, führt in die Thematik ein und gibt einen Überblick über diese Arbeit.

Das zweite Kapitel, Grundlagen, gilt als Einführung in die Grundlagen und Grundbegriffe von Enterprise Architecturen. Weiterhin wird auf die Modellierungssprache Archimate zugegriffen. Außerdem wird die Bibliothek ReactJS und das Architekturstil FLUX vorgestellt. Zum Schluss wird die dokumentenorientierte Datenbank MongoDB beschrieben.

Das dritte Kapitel, Analyse, zeigt zunächst eine Marktanalyse von vorhandenen EA Tools, die die Archimate Modellierungssprache verwenden. Dannach werden diese Tools miteinander Verglichen. Abschließend werden die Anforderungen an das System beschrieben. Diese stellt die Basis der Erwartungen an die Webapplikation und seine Funktionsweise dar.

Anschließend wird in Kapitel 4, Design und Entwurf, ein Mockup, für die Darstellung gezeigt. Daraufhin wird anhand von UML Diagrammen die Architektur modelliert.

Das fünfte Kapitel, Implementierung, zeigt zunächst wie die Visualisierung der Webapplikation aufgebaut ist. Weiterhin wird auf die Client-Server Kommunikation eingegangen. Danach wird beschrieben wie die FLUX Architektur in React integriert wurde. Nach der Beschreibung wird erklärt wie die Verifizierung der Archimate Elemente implementiert wurde. Außerdem wird die Datenbank Implementierung erläutert. Als letztes werden die zwei Prototypen vorgestellt.

Das sechste Kapitel, Evaluierung. fängt mit der Installationsbeschreibung an. Daraufhin wird die Durchführung des Systemtests beschrieben. Zum Schluss werden die Prototypentest mit dem Versuchsaufbau gezeigt.

Das letzte Kapitel, Fazit und Ausblick, gibt eine abschließende Zusammenfassung der Arbeit mit Bewertung des Ergebnisses. Der Ausblick erläutert offene Fragestellungen und schließt die Arbeit ab.

2 Grundlagen

2.1 Enterprise Architecture

Die Ausdehnung eines Unternehmens führt zur Entwicklung neuer Prozessketten. Dabei werden die Anforderungen und die technischen Umsetzung an einem Unternehmen komplexer [HGG13, S. 7]. Anhand der Fähigkeiten einer zügigen und effizienten Umsetzung der Geschäftsanforderungen, kann die Qualität der IT eines Unternehmens gemessen werden [Bö08, S. 8]. Jedoch genügt die IT den Fachbereichen in vielen Fällen nicht. Bei der dezentralen Struktur ist die Komplexität eines Unternehmens besonders hoch [Mat11, S. 21], was den Überblick über das gesamte Unternehmen erschwert. Einer der Gründe ist, dass eine Barriere zwischen fachlichen und technischen Abteilungen existiert [Sch09, S. 4]. Beide Abteilungen unterscheiden sich im Grad der Abstraktion. Die technische Abteilung spezialisiert sich in der Tiefe des Problems, wobei die fachliche Abteilung die Breite des Problems betrachtet.

Für einen Überblick auf ein Unternehmen eignen sich Unternehmensarchitekturen (englisch EA, Enterprise Architecture). Der Begriff Enterprise Architecture wird in der Literatur im unterschiedlichen Detaillierungsgrad definiert. Somit gibt es keine eindeutige Definition des Begriffs. Eine Definition von Scott A. Bernard [Ber12, S. 31] ist:

> *„ The analysis and documentation of an enterprise in its current and future states from an integrated strategy, business, and technology perspective. “*

In dieser Definition wurden die wesentlichen Architekturebenen von Unternehmen aufgelistet: Strategie, Business und Technologie. Diese werden für den aktuellen und zukünftigen Zustand anhand von Diagrammen dokumentiert und analysiert.

Das Problem bei der Dokumentierung der EA ist, dass die Diagramme schnell veraltet werden können. Die Einführung eines Enterprise Architecture Management, soll dabei unter anderen für eine kontinuierliche und aktuelle Dokumentation der Architektur sorgen. Zudem ist mit dem Enterprise Architecture Management (EAM) die Steuerung und Planung von Unternehmensarchitekturen gemeint [Bö08, S. 8].

Es gibt über fünfzig Frameworks für EA, von den nur zehn in der Literatur populär sind [Mat11, S. 5]. Grund dafür ist, dass einige Frameworks voneinander abgeleitet sind oder durch Praxiserkenntnisse entstanden sind [Mat11, S. 17].

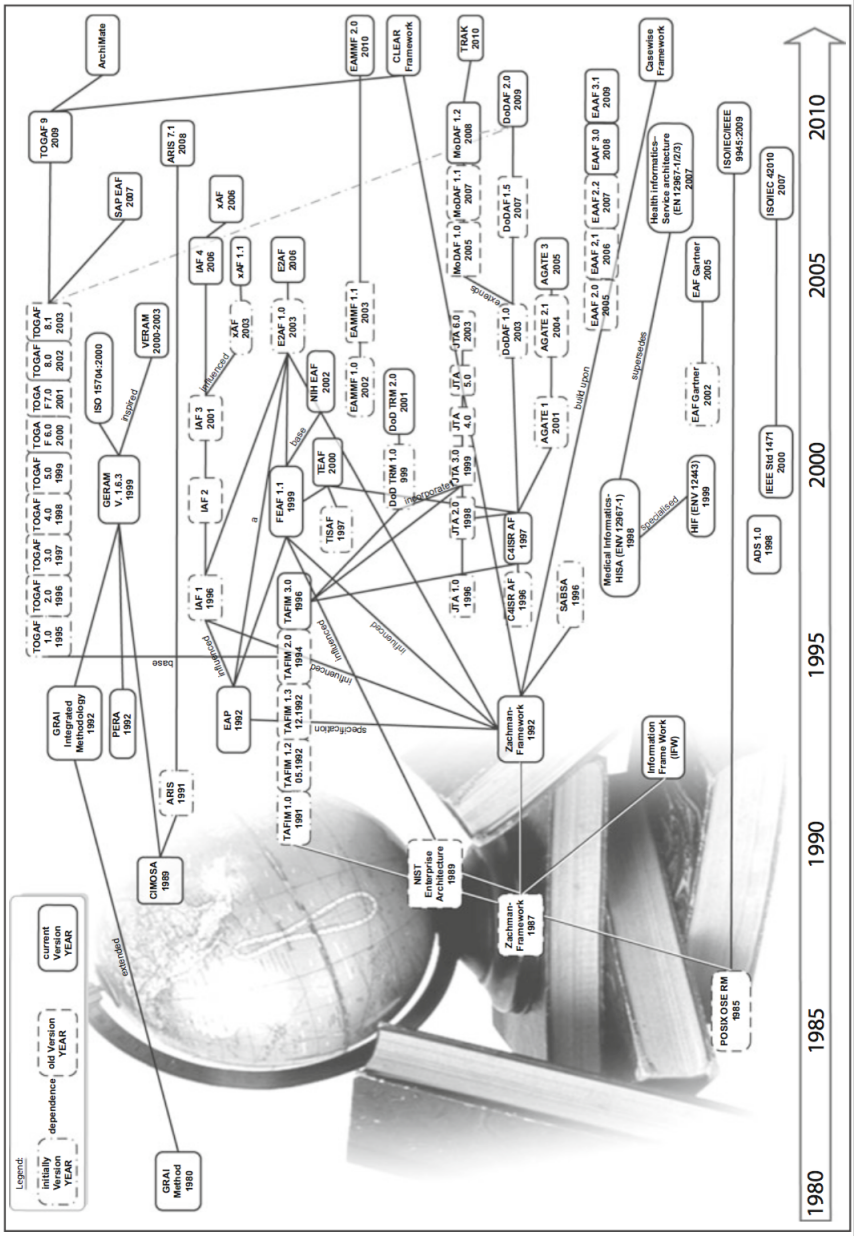

Abbildung 2.1: Über 50 verschiedene EA Frameworks und deren Beziehungen untereinander [Mat11, S. 57].

Die Abbildung 2.1 gibt einen Überblick über die verschiedenen Frameworks. Die von Sven Feurer durchgeführte Umfrage [Feu07] zeigt, dass das meist verwendete Framework mit 33% *TOGAF* ist. Den zweiten Platz mit 22% besetzt das *Zachmann* Framework.

2.1.1 Archimate

Archimate wurde von *The OpenGroup* entwickelt und ist vollständig auf das *TOGAF* Framework abgestimmt [Gro]. Diese Arbeit grenzt sich auf diese Modellierungssprache ab. Die Modellierungssprache Archimate wurde für die einheitliche Darstellung von Diagrammen, die eine EA beschreiben, zur Verfügung gestellt [Gro13a]. Archimate bietet die drei Schichten (Layer): *business layer*, *application layer*, *technology layer* an. Zusätzlich gibt es noch den *motivation layer* und *implementation and migration extension layer*.

business layer bietet Produkte und Dienstleistungen für externe Kunden an.

application layer unterstützt das business layer mit Anwendungsservices, die von Softwareanwendungen realisiert werden.

technology layer bietet IT-Infrastruktur Dienste an. Es wird zur Ausführung von Anwendungen benötigt.

motivation layer beinhaltet die Motivationskonzepte, wie das Ziel, die Prinzipien und die Anforderungen.

implementation and migration extension layer umfasst die Konzepte der Modellierung der Umsetzung von Programme.

2.2 Web Applikation

Webapplikation lassen sich in drei unterschiedliche Bereiche gliedern: Inhalt(HTML), Stil(CSS) und Logik(JavaScript). Frühere Webapplikationen wurden als Sammlung von Webseiten betrachtet. Mehrere Programmierer arbeiteten an unterschiedlichen Teilen der Webseite. Am Inhalt, Stil oder an der Logik. Heutzutage haben Webapplikationen nur eine einzelne Seite. Solche Applikationen werden Single Page Application (SPA) genannt. Anstatt als Layout für den Inhalt einer Seite, werden stattdessen diese Applikationen als Behälter für die Webapplikationen betrachtet. Die Arbeitsweise beim Aufrufen der Webseite über ein Webbrowser geschieht bei beiden Methoden gleich. Der Browser erstellt dabei ein Document Object Model (DOM). Jedoch wird bei der traditionellen Methode das Layout der Webseite komplett beschrieben.

Bei der SPA werden zusätzlich Elemente erstellt, die mittels JavaScript dazu dienen den DOM zu manipulieren. Dazu bietet der Browser die JavaScript DOM API an. Das Manipulieren des DOMs mittels JavaScript bringt zwei Probleme mit sich. Zu einem wird beim direkten verwenden der JavaScript DOM API der Programmierstil imperativ, welcher schwer zu pflegen ist. Zu Anderen ist der DOM nie für die Erstellung dynamischer Benutzerschnittstellen optimiert worden, sodass die Performanz darunter leidet. Diese Probleme werden bei der JavaScript Bibliothek React gelöst [Fed15, S. 17-19]. Zusätzlich wurde als Server der Webapplikation die Plattform NodeJS verwendet.

2.2.1 NodeJS

NodeJS [nod] ist eine Plattform die erlaubt serverseitige Anwendungen mit der clientseitigen Programmiersprache JavaScript zu schreiben. Zudem verwendet NodeJS Ereignisbasierende nichtblockierende I/O Module, die perfekt für datenintensive Echtzeitanwendungen geeignet sind. In dieser Arbeit soll es mit vielen Änderungen an Elementen in einem Diagramm gut umzugehen sein. Daher bietet sich die Plattform an. Weiterhin wird mit der NodeJS Version *v5.4.1* gearbeitet. Zusätzlich bietet NodeJS ein Menge von Modulen an, die frei zu Verfügung stehen. Modulen sind Anwendungen, die mit NodeJS implementiert sind. Diese können wiederverwendet werden. Da die Anzahl der Module so groß ist, bietet NodeJS zusätzlich einen Package Manager an, der bei der Installation mitgeliefert wird. [Fed15, S. 4-5]

2.2.2 ReactJS

ReactJS ist eine JavaScript Bibliothek von Facebook und Instagram, die für die Erstellung von Benutzerschnittstellen dient. Dazu gibt es eine Online verfügbare Dokumentation [fac]. In dieser Arbeit wird die Bibliothek für das Rendern der Webapplikation genutzt. Die Performanz erhält React dadurch, weil es niemals direkt mit dem DOM arbeitet. Stattdessen arbeitet die Bibliothek mit dem Virtuellen DOM. Der Virtuelle DOM ist eine Schicht zwischen den echten DOM und den React States.

[Fed15], [fac].

2.2.2.1 React Komponente

React repräsentiert die Benutzeroberfläche mittels React Komponenten, die in einer Baumstruktur angeordnet sind. Dabei können die Komponenten Elternknoten oder Kinderknoten mit oder ohne einem Zustand beinhalten. Dabei soll nur ein geringer Teil der React Komponenten zustandsbehaften sein. Wenn der oberste Knoten einen Zustands besitzt, kann dieser seinen

Kinderknoten per Eigenschaftenübergabe (siehe Listing 2.1) diesen Weitergeben. Der Großteil der React Komponeten sollte daher zustandslos bleiben und durch die Eigenschaftenübergabe des Elternknoten wird dessen Renderfunktion aufrufen. Die Renderfunktion bewirkt das neumalen des React Elements im virtuellen DOM. Weiterhin kann eine React Komponente nur über eine Eigenschaftenübergabe (siehe Listing 2.1) oder durch ändern des Zustands (siehe Listing 2.2), zum Rendern gebracht werden. Die Eigenschaften einer React Komponente können wie in Listing 2.1 zugegriffen werden. Zudem sind in den Eigenschaften Daten enthalten, die nur gelesen werden können. Diese werden der Komponente vom Elternknoten überreicht.

```
1 this.props
```

Listing 2.1: Die Eigenschaften einer React Komponente.

Der Zugriff auf den Zustand einer React Komponente kann wie in Listing 2.2 erfolgen. Außerdem kann nur innerhalb der Komponente selbst auf den Zustand zugegriffen werden. Dabei kann die Komponente diesen ändern. Nach jeder Zustandsänderung wird die Komponente neu gerendert.

```
1 this.state
```

Listing 2.2: Der Zustand einer React Komponente.

[Fed15], [fac].

2.2.2.2 Virtual DOM

Da Webanwendungen heutzutage nicht mehr statisch sind, muss der DOM häufiger manipuliert werden. Dies bringt einen enormen Performanzverlust mit sich, da der DOM nicht dafür optimiert wurde. Die Webapplikation besitzt einen Zustand für die Benutzeroberfläche. Dieser Zustand kann verändert werden, sobald ein Ereignis eintritt. Ein Ereignis kann dabei entweder vom Anwender oder vom Server kommen. Die Anwenderereignisse sind zum Beispiel das tippen, klicken, die Größe verändern usw.. Serverereignisse können Nachrichten sein, die die Anwendung empfängt, oder Fehler, die auf Serverseite passieren. Beim Eintreten solcher Ereignisse können die Daten, von der die Applikation abhängig ist, verändert werden. Diese Daten repräsentieren den Zustand vom Datenmodell der Webapplikation. Wenn sich dieser Zustand ändert, soll auch der Zustand von der Benutzeroberfläche geändert werden. Hierbei sollen zwei verschiedene Zustände synchronisiert werden. Der Zustand von der Benutzeroberfläche, sowie der Zustand des Datenmodells. Beide sollen auf die Änderung des Anderen reagieren können. Für dieses Problem gibt es unterschiedliche Lösungsansätze. Eines ist das *2-way-binding*. Dies

lässt sich zudem in zwei weitere Typen unterscheiden: Key Value observing (KVO) und Dirty Checking. Jedoch verwendet React einen anderen Lösungsansatz, den Virtuellen DOM.

Die Abbildungen 2.2 und 2.3 geben ein Beispiel für das Rendern beim virtuellen DOM. In Abbildung 2.2 ist ein DOM in einer Baumstruktur zu sehen. Dieser besteht aus einer Wurzelknoten *"body"*, darunter liegen weitere *div* Knoten. Wenn ein Ereignis den Knoten mit der *id=2* verändert, dann werden nur die zu ändernde Knoten und dessen Kinderknoten mit den ids 4 und 5 gerendert. Die Abbildung 2.3 kennzeichnet diese mit einer blauen Farbe. Der Vorteil liegt darin das nicht der komplette DOM, sondern nur ein Teil gerendert wird.

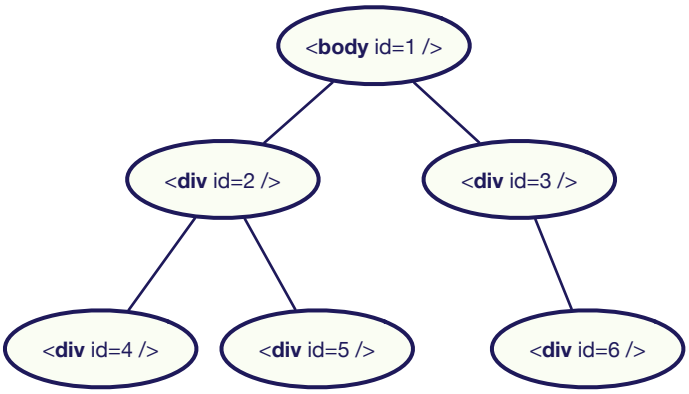

Abbildung 2.2: Beispiel eines virtuellen DOM Baums.

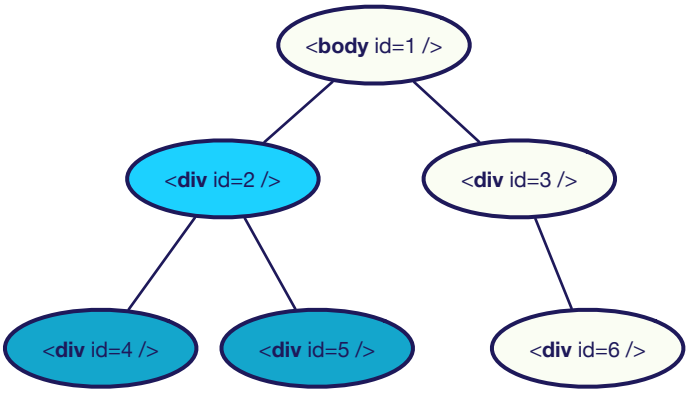

Abbildung 2.3: Beispiel eines virtuellen DOM Baums der gerendert wird.
Die in Blau gefärbten Knoten zeigen die Komponenten an, die gerendert werden.

[Fed15], [fac].

2.2.2.3 Lebenszyklus

Der Lebenszyklus einer React Komponente kann in drei Bereiche eingeteilt werden.

- Montieren (Mounting): Fügt die React Komponente in das DOM ein.
- Aktualisieren (Updating): Erneutes Rendern der React Komponente.
- Abmontieren (Unmounting): Entfernen der React Komponente aus dem DOM.

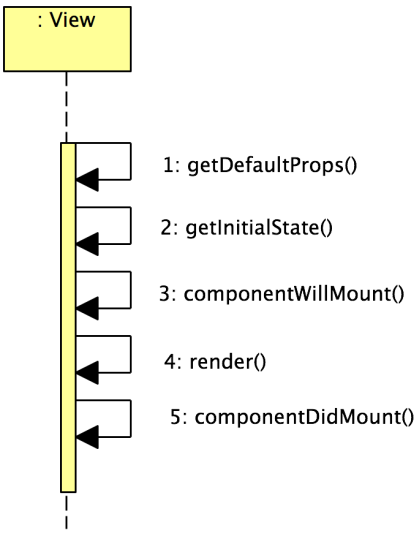

Abbildung 2.4: Sequenzdiagramm für das erste Rendern der React Komponente.

Beim Montieren wird die Komponente in das DOM eingefügt und das erste Rendern der Komponente wird durchgeführt [fac]. Das Montieren wird nur einmal im Lebenszyklus einer ReactKomponente verwendet. In Abbildung 2.4 sind die Funktionen in chronologischer Reihenfolge dargestellt.

Die erste Funktion, die aufgerufen wird, ist *getDefaultProps*. Hierbei werden Standardwerte für die Eigenschaften der Komponente gesetzt, falls diese nicht von den Elternkomponenten gesetzt wurden. Als nächstes wird *getInitialState* ausgeführt. Diese Funktion initialisiert den Zustand der Komponente. Vor der Render-Funktion wird *componentWillMount* ausgeführt. Falls innerhalb dieser Funktion der Zustand geändert wird, wird die Komponente trotzdem nur einmal gerendert. Anschließen wird die Renderfunktion aufgerufen. Nachdem die Komponente gerendert wurde, wird das Montieren mit der Funktion *componentDidMount* abgeschlossen. Hierbei kann auf die unterliegenden KinderKomponenten zugegriffen werden.

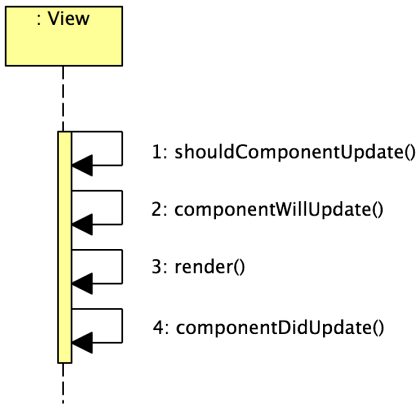

Abbildung 2.5: Sequenzdiagramm zeigt React Komponente bei Zustandsänderung.

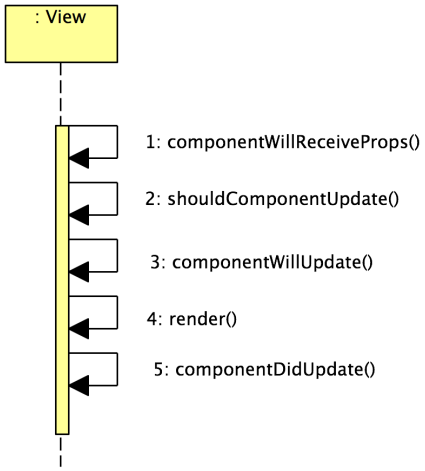

Abbildung 2.6: Sequenzdiagramm zeigt React Komponente bei Eigenschaftenänderung.

Ein weiterer Bereich ist das Aktualisieren. Hierbei wird die React Komponente neu gerendert und bestimmt, ob das DOM neugeladen werden soll. Zudem wird unterschieden, ob der Zustand oder die Eigenschaften gerendert werden.

In Abbildung 2.5 wird das Rendern der Komponente bei einer Änderung des Zustandes gezeigt. Zuerst kommt eine Abfrage, ob die Komponente wirklich gerendert werden soll (*shouldComponentUpdate*). Hierbei können Bedingungen eingetragen werden. Der Rückgabewert dieser Funktion ist entweder Wahr oder Falsch. Wenn Wahr zurückgeliefert wird, wird das Rendern fortgesetzt. Bei Falsch wird die Komponente nicht gerendert. Falls vor dem Rendern der Zustandsänderung noch etwas durchzuführen sein sollte, kann dies bei der Funktion

componentWillUpdate eingetragen werden. Dannach wird die Komponente gerendert. Nach dem Rendern wird die Funktion *componentDidUpdate* aufgerufen.

In Abbildung 2.6 wird das Rendern der Komponente bei einer Änderung der Eigenschaften gezeigt. Zuerst wird die Funktion *componentWillReceiveProp* mit den neuen Eigenschaften als Parameter aufgerufen. Die alten Eigenschaft können über *this.prop* erreicht werden. Diese Funktion bietet die Möglichkeit die alten mit den neuen Eigenschaften in Vergleich zu setzen. Zudem wird bei Verwendung von *setState* die Render-Funktion nicht erneut aufgerufen.

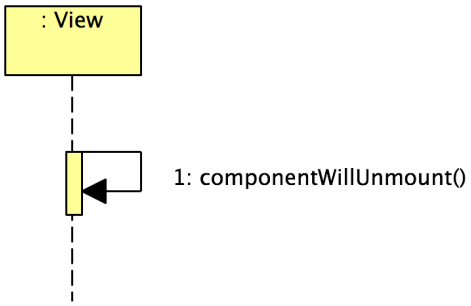

Abbildung 2.7: Sequenzdiagramm React unmount

Zuletzt gibt es das Abmontieren. Dabei wird die React Komponente vom DOM getrennt. Dazu wird lediglich die Funktion *componentWillUnmount* aufgerufen. Diese bietet die Möglichkeit noch eine Bereinigung des System durchzuführen, bevor die Komponente vom DOM entfernt wird.

[Fed15], [fac].

2.2.3 Flux

FLUX ist ein Architekturstil von Facebook und erweitert die Bibliothek ReactJS bei der Benutzung von Unidirektionalen Datenfluss (nur in eine Richtung). Dieser besteht aus den vier Komponenten (siehe Abbildung 2.8): View, Action, Dispatcher und Store.

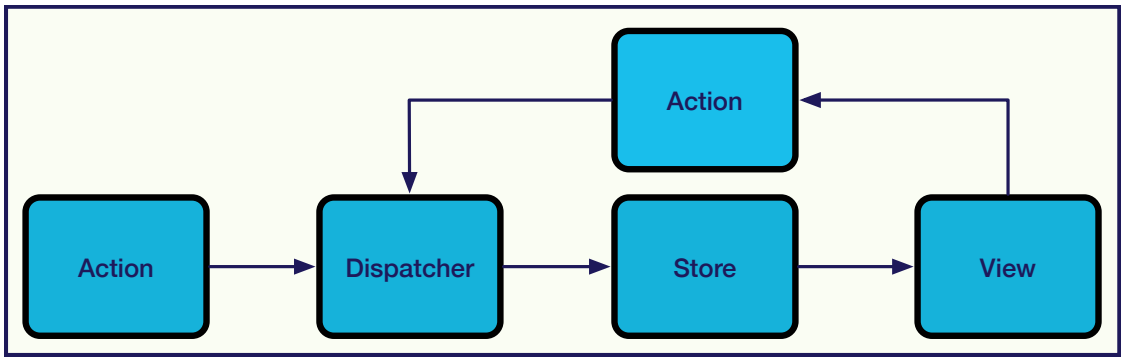

Abbildung 2.8: Flux Architektur

Die Abbildung 2.8 stellt den Architekturstil FLUX dar. Dieser zeigt den Datenfluss einer Anwendung an. Wenn die Webapplikation ihren Zustand ändert will, wird zunächst ein Action Objekt erstellt. Dieses Objekt hat zwei Eigenschaften: Eine ist der Aktionstyp, der die Action identifiziert, der Andere ist die Information die für den neuen Zustand gebraucht wird. Eine Action wird über den Dispatcher an den passenden Store geliefert. Für die Verwaltung der Daten ist die Store Komponente verantwortlich. Zusätzlich empfängt das Store das Action Object vom Dispatcher und aktualisiert alle passenden Views. Eine View ist eine React Komponente die ihre Ereignisse an die Action senden und Stores abonnieren.

[Fed15], [fac].

3 Analyse

Im folgenden Kapitel wird eine Marktanalyse durchgeführt. Darauf aufbauend werden die Anforderungen analysiert und dargestellt.

3.1 Tools mit Archimate im Vergleich

Vor der Entwicklung einer Webapplikation muss zuerst überprüft werden, welche Produkte auf dem Markt existieren. Dabei beschränkt sich die Suche auf Desktop- und Webapplikationen mit ArchiMate. Auf dem Markt gibt es dazu verschiedene Modellierungstools, aus den Simply Archimate, Visual Paradigm und Archi in Vergleich gesetzt werden. Die Analyse konzentriert sich auf die Visualisierungsmöglichkeiten dieser EA-Werkzeuge. Weiterhin werden auch die Navigierung, Erzeugung und Bearbeitung von EA sowie die kollaborativen Eigenschaften analysiert.

3.1.1 Simply Archimate

Das erste Werkzeug ist eine Webapplikation Simple Archimate, die sich selbst als die weltweit erste webbasierte Archimate2.1 Applikation bezeichnet [Sim16]. Veröffentlicht wurde es im Jahr 2014. Zudem existiert sowohl eine freie, als auch eine kostenpflichtige Lizenz für die Applikation. Die Letztere bietet den Vorteil der unbeschränkten Speicherung von Modellen. Es wurde die freie Version getestet. Nach der Anmeldung erscheint dem Anwender ein Arbeitsbereich (siehe Abbildung 3.1).

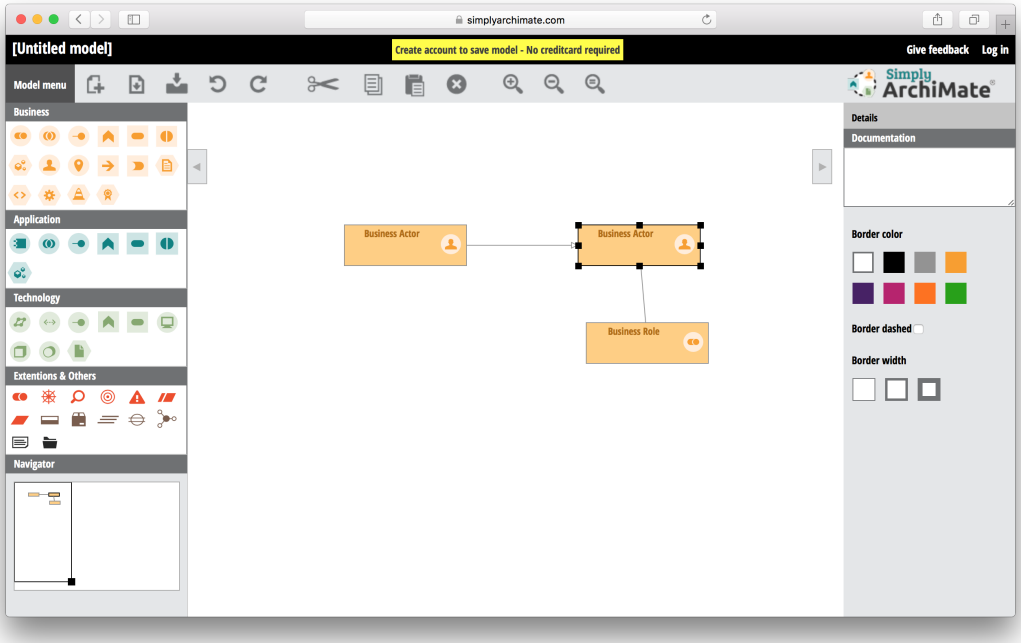

Abbildung 3.1: Ausschnitt der Webapplikation Simply Archimate.

Die am linken Rand dargestellte Palette zeigt alle ArchiMate Elemente in einer Miniaturansicht. Hier sind diese in den passenden Bereichen eingeordnet. Beim Mausübergang eines Elements wird dieser in einem Tooltip vergrößert angezeigt. Für die Erstellung eines Elements wird die Drag and Drop Funktion benutzt. Dazu wird das Element in den mittig eingeordnete Hauptbereich gezogen. Zusätzlich kann ein Element gruppiert, bewegt und die Größe verändert werden. Die Randfarbe, -style und -breite kann im rechten Bereich geändert werden. Die Beziehungen werden über die Elemente erstellt und passen sich automatisch an. Bei der Erstellung wird ein Popupfenster mit allen möglichen Beziehungen zwischen den beiden Elementen angezeigt. In der oberen Leiste befinden sich die Wiederherstellung von Aktionen, das Ausschneiden, Kopieren und Einfügen von Elementen sowie das Vergrößern und Verkleinern des Hauptbereichs und die Speicherung des Diagramms. Der Navigator unter der Palette, dient zur Übersicht. Das Bearbeiten eines Diagramms ist nur mit einem User möglich.

3.1.2 Archi

Als zweites Tool wurde die Open Source Desktopapplikation Archi mit der Version 3.3.1 betrachtet, das von hunderten Enterprise Architekten benutzt wird [arc16]. Das Open Group ArchiMate® Model entwickelt Plugins für dieses Werkzeug. Beim Starten erscheint ein Fenster wie in Abbildung 3.2 zu sehen.

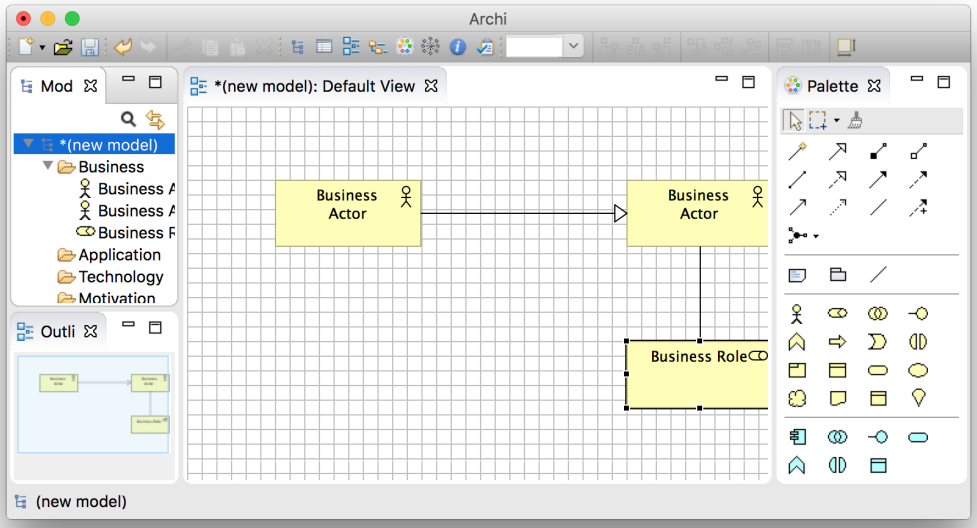

Abbildung 3.2: Ausschnitt der Desktopapplikation Archi.

Anders als bei Simply Archimate können bei Archi die Bereiche wie die Palette (am rechten Rand), das Hauptbereich (Zentral), das Modellverzeichnis (am linken oberen Rand) und das Outline (am linken unteren Rand) vom Anwender verschoben, minimiert, maximiert oder geschlossen werden. Zudem befinden sich in der oberen Leiste Hilfsfunktionen zur Steuerung der Applikation. Das Erstellen eines Elements wird, wie in Simply Archimate, über die Drag and Drop Funktion von der Palette in dem Hauptbereich ermöglicht. Die Palette beinhaltet neben den Archimate Elementen zusätzlich die Beziehungselemente. Mit Doppelklick auf ein Element im Hauptbereich öffnet sich das Eigenschaftenbereich. Dort können die Werte des Elements verändert werden. Der Hauptbereich ist mit einem Raster versehen. Das Vergrößern und Verschieben ist direkt am Element möglich. Bei der Erstellung einer Beziehung muss zuerst eine Beziehungsart in der Palette ausgewählt werden. Danach folgt die Markierung zweier

Elemente die in Beziehung stehen sollen. Die Beziehungsverläufe können im Nachhinein verändert werden. Zudem können nur valide Beziehungen erstellt werden. Zum Vergrößern oder Verkleinern der Sicht im Hauptbereich gibt es in der oberen Menuleiste eine Auswahlliste in der die Größe der Sicht ausgewählt werden kann. Das Outline dient zur Navigation. Der Anwender kann dadurch das Diagramm bewegen. Archimate Model Elemete, die im Hauptbereich liegen, werden in einer Baumstruktur im Modelverzeichnis dargestellt. Die Applikation wurde in Java geschrieben, was es Plattform unabhängig macht. Das Bearbeiten eines Diagramms ist nur mit einem User möglich.

3.1.3 Visual Paradigm

Als drittes wird die Desktopapplikation Visual Padardigm betrachtet. Laut der offiziellen Seite von Paradigm wird die Software weltweit von mehreren Firmen benutzt, unter anderem wird es Apple Inc., Airbus, Deutsche Bank AG, SAP und NASA verwendet [par16]. Zudem ist die Applikation kostenpflichtig. Nach dem Programmstart erscheint dem Anwender ein Arbeitsbereich (siehe Abbildung 3.3)

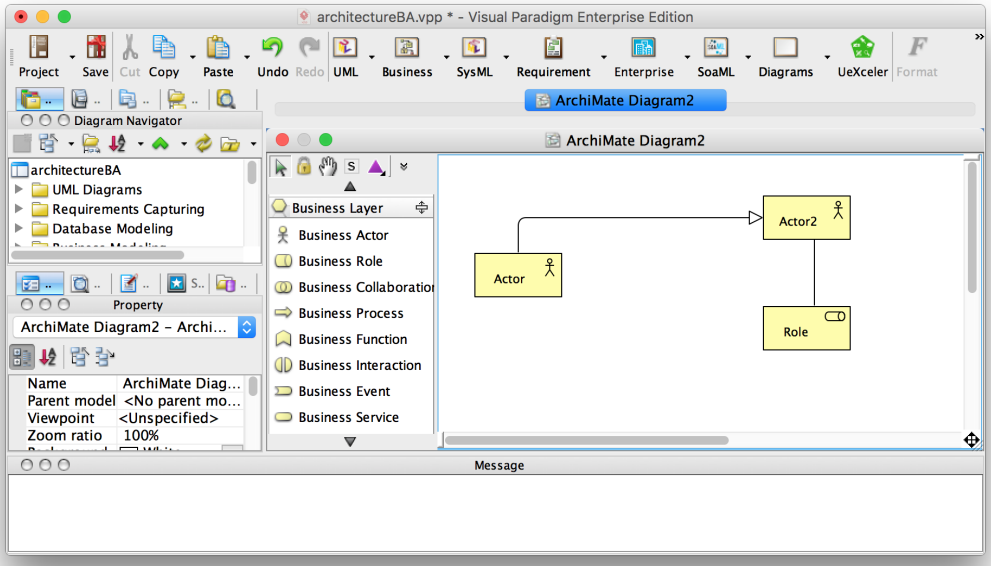

Abbildung 3.3: Ausschnitt der Desktopapplikation Visual Paradigm

Das Hauptfenster wird in mehrere Unterfenster aufgeteilt. Die obere Leiste dient zur Steuerung der Applikation, darin kann unter anderem die Diagrammart Archimate ausgewählt werden. Nach der Erstellung von Archimate Diagramm, wird eine Palette mit allen Archimate Elemente angezeigt. Die Elemente können durch Drag and Drop Funktion in das Hauptbereich erzeugt werden. Das Werkzeug bietet kollaborative Modellierung mit cloudbassierten Projektrepositorie an. Am rechten unteren Rand befindet sich ein gekreuztes Pfeilsymbol, wodurch das Outline geöffnet wird. Im Outline kann die Sicht des Hauptbereichs navigiert werden.

3.1.4 Tools Analyse

Wir haben festgestellt, dass die ausgewählten Tools Visualisierungsähnlichkeiten besitzen. Grundsätzlich kann das Design der Werkzeuge in die Bereiche Hauptbereich (H), Palette (P), Menu (M), Outline (O) und Eigenschaftenbereich (E) geteilt werden. Zudem wurden mehrere Funktionalitäten beobachtet, die Teilweise in unterschiedlichen Bereichen der Tools liegen.

Die Tabelle 3.1 fasst einzelne Merkmale aus den detaillierten Beschreibungen zusammen. Es werden die Funktionalitäten zu den Bereichen zugeordnet und farblich markiert. Die farblichen Markierungen zeigen wie oft eine Funktionalität in einem Bereich benutzt werden. Die Werkzeuge werden in zwei Kategorien eingeteilt: Webapplikation und Desktopapplikation.

Im Folgenden wird die Wahl der Funktionalitäten erläutert. EA-Diagramme ändern sich täglich und werden nach einiger Zeit so groß, dass die nicht mehr auf das Bild passen. Damit wird die Bewegung der Sicht des Hauptbereichs und das Vergrößern/Verkleinern der Sicht nötig gefragt. Aufgrund der Abgrenzung auf die Archimate Modellierungssprache, werden die Tools auf die Darstellung der Archimate Elemente/Beziehungen überprüft. Unter den Grundfunktionen bei der Bearbeitung eines EA-Diagramms zählt die Erstellung der Elemente. Zusätzlich werden Elemente aus dem Arbeitsbereich entfernt. Das Zusammenfassen von Elementen, sowie die Änderung der Elementenposition gehören auch zu den Grundfunktionalitäten. Um Elemente in den Diagrammen zu verdeutlichen wird das Design des Diagramms verändert. Dazu wird die Größe, die Hintergrundsfarbe, Textfarbe, Textgröße und Textform eines Elements angepasst. Es können nur die Elemente in Beziehung stehen, die die Archimate Spezifikation (siehe Anhang) vorgibt. Zudem können bestimmte Bereiche angezeigt oder ausgeblendet werden. Zur Verbesserung des Verständnis über das Element, wird dieses mit einem eigenen Namen versehen. Da die Beziehungen zwischen zwei Elemente die Hauptteile eines Diagramms sind, wird die Funktion Erstellung der Beziehung auch als Vergleichskriterien genommen. Damit die Beziehungen nicht das Verständnis verkomplizieren, wird die Funktion Änderung Beziehungsverhalten gefragt. Die Beziehungen können damit sauber positioniert werden. Die Darstellung der Raster wird für die Erleichterung der Positionierung von Elemente benutzt. Abschließend kommt die

kollaborative Eigenschaft, das heißt die Möglichkeit zusammen mit mehreren Anwender eine Diagramm gemeinsam bearbeiten.

Werkzeuge → /Funktionalität ↓	Webapplikation					Desktopapplikation									
	Simple Archimate					Archi					Visual Paradigm				
	H	P	M	O	E	H	P	M	O	E	H	P	M	O	E
Bewegung der Sicht des Hauptbereichs	X	-	-	X	-	X	-	-	X	-	X	-	-	X	-
Vergrößern/Verkleinern der Sicht	-	-	X	X	-	X	-	X	-	-	X	-	X	X	-
Darstellung der Archimate Elemente	X	X	-	X	-	X	X	-	X	-	X	X	-	X	-
Darstellung der Archimate Beziehungen	X	-	-	X	-	X	X	-	X	-	X	X	-	X	-
Änderung Beziehungsverhalten	-	-	-	-	-	X	-	-	-	-	X	-	-	-	-
Erstellung der Elemente durch Drag'n'Drop	X	X	-	-	-	X	X	-	-	-	X	X	-	-	-
Erstellung der Beziehung	X	-	-	-	-	X	X	-	-	-	X	X	-	-	-
Löschen der Elemente	-	-	X	-	-	X	-	X	-	-	X	-	-	-	-
Gruppierung der Elemente	X	-	-	-	-	X	-	-	-	-	X	-	-	-	-
Bewegung der Elemente	X	-	-	-	-	X	-	-	-	-	X	-	-	-	-
Änderung der Größe	X	-	-	-	-	X	-	-	-	-	X	-	-	-	-
Änderung des Styles	-	-	-	-	X	-	-	-	-	X	-	-	-	-	X
Änderunger der Elementenname	X	-	-	-	-	X	-	-	-	X	-	-	-	-	X
Darstellung der Raster	-	-	-	-	-	X	-	X	-	-	X	-	X	-	-
Validierung	X	-	-	-	-	X	-	-	-	-	X	-	-	-	-
Ein-/Ausblenden	-	X	-	X	X	X	X	-	X	X	X	X	-	X	X
Kollaboration	-	-	-	-	-	-	-	-	-	-	X	-	-	-	-

Tabelle 3.1: Überblick von Funktionalitäten der EA Tools.

__H__: Hauptbereich; __P__: Palette; __M__: Menu; __O__: Outline; __E__: Eigenschaftenbereich; __X__: Vorhanden; -: Nicht Vorhanden; __FARBEN__:[ROT : Es gibt genau 1 Werkzeug, der diese Funktionalität hat; GELB : Es gibt genau 2 Werkzeuge, die diese Funktionalitäten im gleichen Bereich haben; GRÜN : Es gibt genau 3 Werkzeuge, die diese Funktionalitäten im gleichen Bereich haben].

Als Haupttendenz lässt sich feststellen, dass die meisten Funktionalitäten im Hauptbereich geschehen. Im Gegensatz zur Desktopapplikation haben Webapplikation weniger Funktionen. Auffällig ist, dass die Kollaboration nicht die Hauptanforderung an kostenfreien EA-Tool ist. Zudem ist zu erkennen, dass die Funktionalitäten der Tools größtenteils identisch sind. Abschließend kann festgestellt werden, dass die Kosten für Visual Paradigm durch die Kollaboration Funktionalität gerechtfertigt wird.

3.2 Anforderungsanalyse

Durch Anforderungen lassen sich Lösungswege eines Problems oder das Erreichen eines Zieles detailliert beschreiben[Poh08, S. 13-18]. Die Analyse der Anforderungen lässt sich dazu in zwei Bereiche klassifizieren, die funktionalen und nichtfunktionalen Anforderungen. Wobei sich die nichtfunktionalen Anforderungen auf die Qualitätsanforderungen beziehen. Gemeinsam

mit externen Einflussfaktoren können dadurch Softwarearchitekturen entstehen.[Sta15, S. 29] In diesem Abschnitt werden die funktionalen und nichtfunktionalen Anforderungen zu der Webapplikation beschrieben, die sich an die Funktionen vom Markt verfügbaren EA-Tools orientieren.

3.2.1 Funktionale Anforderungen

Die funktionalen Anforderungen definieren die von der Webappliaktion bereitzustellenden Funktionen oder Services für den Nutzer[Poh08, S. 15]. Zunächst wird eine Auflistung der funktionalen Anforderungen an die Webapplikation gezeigt.

FA1 **Anwender sollen den Hauptbereich navigieren können.** Die Navigation soll das Vergrößern, Verkleinern und Bewegen der Sicht des Hauptbereichs beinhalten. Um andere Anwender nicht zu stören, soll die Navigierung lokal abgearbeitet werden.

FA2 **Alle möglichen Elemente und Beziehungen sollen in einer Palette aufgelistet werden.** Um eine Auswahl zur Erstellung der Elemente zu ermöglichen, wird eine Palette benötigt.

FA3 **Anwender sollen Elemente durch Drag and Drop erstellen können.** Das Erstellen eines Elements soll durch die Drag and Drop Funktion ermöglicht werden. Dabei soll das ausgewählte Element aus der Palette in den Hauptbereich gezogen werden.

FA4 **Anwender sollen Elemente/Beziehungen löschen können.** Beim Löschen eines Elements soll dessen Beziehungen automatisch mitgelöscht werden.

FA5 **Anwender sollen Elemente bearbeiten können.** Zum Bearbeiten eines Elements gehören die Veränderung der Position, Größe, Titel oder Farbe.

FA6 **Anwender sollen Elemente gruppieren können.** Die Gruppierung soll durch das Einfügen eines Elements in ein anderes erfolgen.

FA7 **Anwender sollen Elemente in Beziehung setzen können** Bei der Setzung einer Beziehung zweier Elemente soll zunächst eine Beziehungsart ausgewählt werden. Eine Beziehung soll ein Ziel-Element und ein Quell-Element beinhalten.

FA8 **Anwender sollen Beziehungsverlauf verändern können.** Um die Beziehungen übersichtlicht zu halten, soll es möglich sein den Beziehungsverlauf zu verändern. Nachdem eine Beziehung existiert können weitere Knotenpunkte darauf erstellt oder gelöscht

werden. Die Knotenpunkte lassen sich einzeln Positionieren, wobei sich dadurch der Verlauf der Beziehung verändert.

FA9 Die Webapplikation soll eine Unterstützung von der Modellierungssprache ArchiMate bieten. Der Anwender soll mit der Webapplikation eine Unternehmensarchitektur modellieren und visualisieren können. Die Beziehungen sollen bei der Erstellung verifiziert werden (siehe Anhang ArchiMate Spezifikation).

FA10 Die Webapplikation soll in die Bereiche Hauptbereich, Palette, Menu, Outline und Eigenschaftenbereich geteilt werden. Die Webapplikation soll aus den Bereichen Hauptbereich, Palette, Menu, Outline und Eigenschaften bestehen. Der Hauptbereich ist für die Darstellung eines Diagramms verantwortlich. Die Palette stellt Elemente zu Verfügung. Das Menu soll Steuerelemente für die Webapplikation beinhalten. Das Outline ist für die Navigierung zuständig. Durch das Eigenschaftenbereich können Elemente manipuliert werden.

FA11 Die Webapplikation soll einen Raster besitzen. Das Raster soll zur Orientierung im Diagramm helfen.

FA12 Bereiche können an und ausgeblendet werden. Zur besseren Übersicht der Webapplikation, soll der Anwender die Möglichkeit haben einzelne Bereiche an- und auszublenden.

3.2.2 Nicht-Funktionale Anforderungen

Die nichtfunktionalen Anforderungen sollen die funktionalen Anforderungen mittels Qualitätseigenschaften und Einschränkungen der Webapplikation ergänzen [Ebe08, S.131]. Diese werden als Oberbegriff für Qualitätsanforderungen verwendet [Poh08, S.15-16]. Zudem lassen sich Qualitätsanforderungen durch Richtlinien identifizieren. Dazu wurde der Standard ISO/IEC 25010 (siehe Abbildung 3.4) ausgewählt.

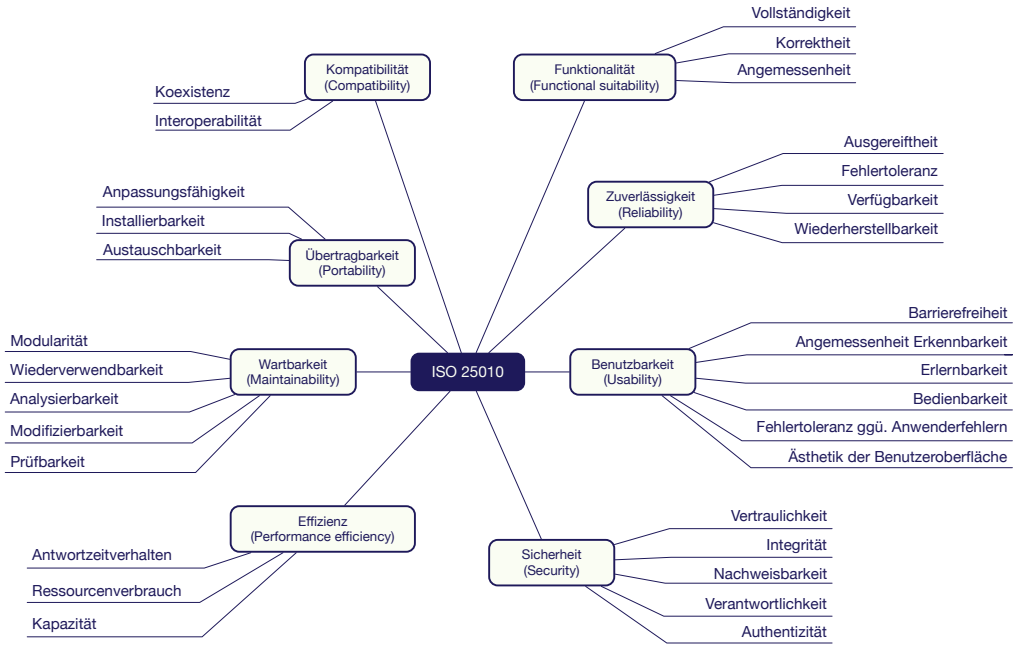

Abbildung 3.4: Qualitätsmerkmale nach ISO/IEC 25010:2011 [iso11].

NFA1 Anpassungsfähigkeit Die Webapplikation soll mindestens auf den folgenden Browsern laufen:

- Chrome Version: 49.0.2623.87 (64-bit)

NFA2 Installierbarkeit Der Aufwand zum Installieren der Webappliaktion soll gering sein.

NFA3 Modularität Die Webapplikation soll in Module aufgeteilt werden.

NFA4 Analysierbarkeit Die Webapplikation soll komplett in der Programmiersprache JavaScript geschrieben werden.

NFA5 Antwortzeitverhalten Die Webapplikation soll einen konsistenten Zustand innerhalb von 1 Sekunde erreichen. Zum Beispiel wenn ein Client ein neues Element erzeugt, sollen alle verbundenen Clients innerhalb von einer 1 Sekunde das neue Element darstellen.

NFA6 Wiederherstellbarkeit Die Webapplikation soll nach einem Verbindungsabbruch den aktuellesten Zustand wiedererlangen.

NFA7 Bedienbarkeit Die Bedienung der Applikation soll sich nach den vorhanden Werkzeugen auf dem Markt richten.

3.3 Anwendungsfälle

Das nachfolgend dargestellte Umweltdiagramm (siehe Abbildung 3.5) stellt dar, welche Akteure mit dem System arbeiten. Es gibt insgesamt einen Akteur, der mit der Webapplikation kommuniziert. Der Benutzer greift auf das System zu und geht damit um.

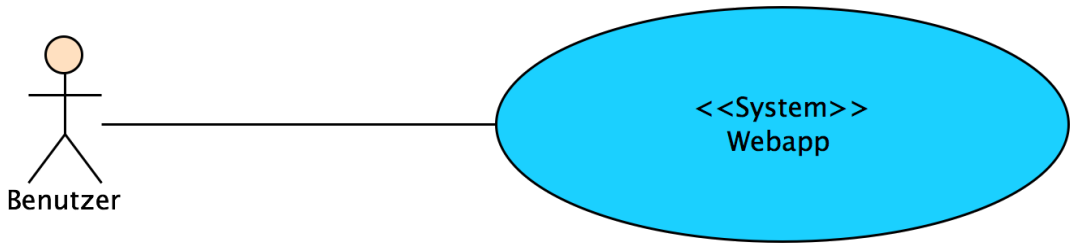

Abbildung 3.5: Diagramm einer Kontextabgrenzung der Webapplikation.

In der Abbildung 3.6 wird der Akteur „Benutzer“ mit seinen zugehörigen Use-Cases sowie dessen Beziehungen dargestellt. Aus den Anforderungen ergeben sich insgesamt sieben Anwendungsfälle: Hauptbereich navigieren, Element erstellen, Beziehung erstellen, Element bearbeiten, Beziehungsverlauf bearbeiten, Gruppierung bearbeiten und Element/Beziehung löschen.

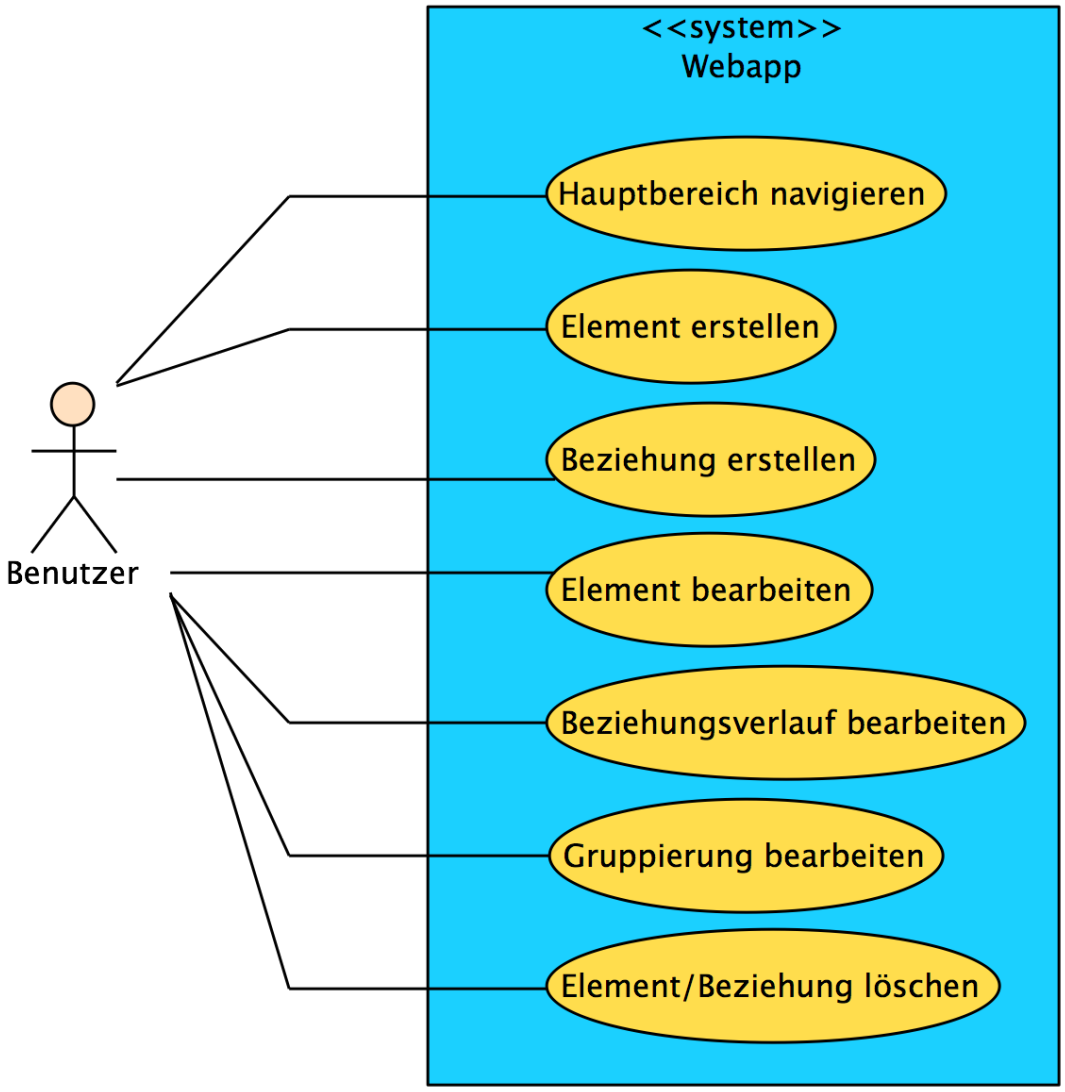

Abbildung 3.6: Alle Anwendungsfälle eines Benutzers.

3.3.1 Hauptbereich navigieren

Die EA-Diagramme sind komplex und können über den Bildschirm hinausragen. Aus diesem Grund wird eine Navigation des Hauptbereichs benötigt. In der Tabelle 3.2 wird der Anwendungsfall zur Navigierung des Hauptbereichs beschrieben. Zur Navigation gehören das Vergrößern, Verkleinern und die Bewegung des Hauptbereichs. Bei der Vergrößerung und der

Verkleinerung soll das Größenverhältnis der Elemente beibehalten werden. Bei der Bewegung soll der Hauptbereich in alle Himmelsrichtung verschoben werden.

ID	UC01
Name	Hauptbereich navigieren
Kurzbeschreibung	Der Anwender soll der Hauptbereich vergrößern, verkleinern oder bewegen können.
Akteure	Anwender
Vorbedingung	Die Anwendung ist gestartet.
Auslöser	Der Anwender will den Hauptbereich navigieren.
Ablauf	
1. Der Anwender wählt eine der Navigationsoption aus. **2.** Das System verändert die Sicht im Hauptbereich des Anwenders. **3.** Anwendungsfall ist beendet.	
Alternativer Ablauf	
2.a Der Anwender hat Zoom out ausgewählt: Das System vergrößert den Hauptbereich. **2.b** Der Anwender hat Zoom in ausgewählt: Das System verkleinert den Hauptbereich. **2.c** Der Anwender hat Bewegung in alle Richtungen ausgewählt: Das System verschiebt den Hauptbereich.	
Nachbedingung	Der Hauptbereich wurde erfolgreich navigiert.
Anforderungen	FA1, FA10

Tabelle 3.2: Anwendungsfall UC01 Hauptbereich navigieren.

3.3.2 Element erstellen

Für das Erstellen eines neuen Element werden die Bereiche Palette und Hauptbereich benötigt. Dazu muss die Palette eingeblendet sein. Zunächst wählt der Anwender einer der Archimate Element aus der Palette aus und zieht diesen per Drag and Drop in das Hauptbereich. Im Hauptbereich soll das Element genau an der losgelassenen Mausposition erstellt werden. Für das kollaborative Arbeiten sorgt das System für eine Aktualisierung des Zustands bei allen verbundenen Clients. In der Tabelle 3.3 wird der Anwendungsfall UC02 beschrieben.

ID	UC02
Name	Element erstellen
Kurzbeschreibung	Es soll ein Element aus der Palette gewählt und im Hauptbereich dargestellt werden.
Akteure	Anwender
Vorbedingung	- Die Anwendung ist gestartet. - Verbindung mit dem Server ist hergestellt.
Auslöser	Der Anwender will ein Element erstellen.
Ablauf	
1. Der Anwender zieht ein Element aus der Palette in den Hauptbereich. **2.** Das System erstellt das gewünschte Element. **3.** Das System aktualisiert den Hauptbereich aller Anwender. **4.** Anwendungsfall ist beendet.	
Alternativer Ablauf	
-	
Nachbedingung	Ein Element ist erstellt.
Anforderungen	FA2, FA3, FA9, FA10, FA12

Tabelle 3.3: Anwendungsfall UC02 Element erstellen.

3.3.3 Beziehung erstellen

Vor der Erstellen einer Beziehung zwischen zwei Elementen soll der Benutzer die Art der Beziehung auswählen. Dazu muss die Palette eingeblendet sein. Als nächstes wird im Hauptbereich ein erstes Element ausgewählt von der die Beziehung startet und ein zweites Element bei der die Beziehung endet. Nach der Auswahl des zweiten Element validiert das System die Beziehung. Für das kollaborative Arbeiten aktualisiert das System den Zustand bei allen verbundenen Clients. Die Tabelle 3.4 beschreibt den Anwendungsfall UC03.

ID	UC03
Name	Beziehung erstellen
Kurzbeschreibung	Es soll eine Beziehung zwischen zwei Elementen erstellt werden.
Akteure	Anwender
Vorbedingung	- Die Anwendung ist gestartet. - Verbindung mit dem Server ist hergestellt. - Es existiern mindestens zwei Elemente.
Auslöser	Der Anwender will zwischen zwei Elementen eine Beziehung erstellen.
Ablauf	
1. Der Anwender wählt ein Beziehungsart aus der Palette aus. **2.** Der Anwender wählt ein erstes Element im Hauptbereich aus. **3.** Der Anwender wählt ein anderes Element im Hauptbereich aus. **4.** Das System überprüft ob die ausgwählte Beziehungsart zu den beiden Elementen passt. **5.** Die ausgewählte Beziehungsart passt zu den Elementen. **5.1** Das System erstellt die ausgewählte Beziehungsart vom ersten Element bis zum zweiten Element. **6.** Das System aktualisiert den Hauptbereich aller Anwender. **7.** Anwendungsfall ist beendet.	
Alternativer Ablauf	
5.a Die ausgewählte Beziehungsart passt nicht zu den Elementen. **5.1.a** Anwendungsfall ist beendet.	
Nachbedingung	Eine Beziehung wurde erfolgreich zwischen zwei Elementen erstellt.
Anforderungen	FA2, FA7, FA9, FA10, FA11

Tabelle 3.4: Anwendungsfall UC03 Beziehungen erstellen.

3.3.4 Element bearbeiten

Ein Elemente das schon im Hauptbereich existiert kann bearbeitet werden. Das heißt die Position, die Größe, der Titel oder die Farbe des Elements kann verändert werden. Die Position und die Größe kann der Anwender über das Hauptbereich bearbeiten. Im Eigenschaftenbereich kann der Anwender zudem die Größe, den Titel und die Farbe des Elements verändern. Dazu

muss der Eigenschaftenbereich zunächst eingeblendet sein. Die Tabelle 3.5 beschreibt den Anwendungsfall UC04.

ID	UC04
Name	Element bearbeiten
Kurzbeschreibung	Es sollen die Eigenschaften eines Elements verändert werden.
Akteure	Anwender
Vorbedingung	- Die Anwendung ist gestartet. - Verbindung mit dem Server ist hergestellt. - Es existiert mindestens ein Element.
Auslöser	Der Anwender will ein Element bearbeiten.
Ablauf	
1. Der Anwender wählt ein Element aus. **2.** Der Anwender verändert den Wert des Elements. **3.** Das System übernimmt die Veränderung. **4.** Das System verändert die Sicht im Hauptbereich aller Anwender. **5.** Anwendungsfall ist beendet.	
Alternativer Ablauf	
-	
Nachbedingung	Die Eigenschaft eines Elements wurde erfolgreich verändert.
Anforderungen	FA5, FA10, FA11, FA12

Tabelle 3.5: Anwendungsfall UC04 Element bearbeiten.

3.3.5 Beziehungsverlauf bearbeiten

Um die Beziehungen übersichtlicht zu halten, soll es möglich sein den Beziehungsverlauf zu verändern. Unter den Verlauf einer Beziehung ist zu verstehen, wie der Pfad vom Ziel bis zum Quellelement führt. Dazwischen können mehrere Knotenpunkte erstellt werden, die frei positioniert oder wieder gelöscht werden können. Die Position der Knotenpunkte zu verändern bedeutet auch den Verlauf der Beziehung zu verändern. Die Tabelle 3.6 beschreibt den Anwendungsfall UC05.

ID	UC05
Name	Beziehungsverlauf bearbeiten
Kurzbeschreibung	Es soll die Bearbeitung eines Beziehungsverlaufs möglich sein.
Akteure	Anwender
Vorbedingung	- Die Anwendung ist gestartet. - Verbindung mit dem Server ist hergestellt. - Es existieren mindestens zwei Elemente mit einer Beziehung.
Auslöser	Der Anwender will den Verlauf einer Beziehung verändern.
Ablauf	
1. Der Anwender wählt eine Beziehung im Hauptbereich aus. **2.** Der Anwender verändert den Verlauf der Beziehung. **3.** Das System übernimmt die Veränderung. **4.** Das System verändert die Sicht im Hauptbereich aller Anwender. **5.** Anwendungsfall ist beendet.	
Alternativer Ablauf	
2.a Der Anwender hat einen neuen Knoten erstellt. **2.b** Der Anwender hat einen Knoten gelöscht. **2.c** Der Anwender hat die Position eines Knotens verändert.	
Nachbedingung	Der Beziehungsverlauf wurde erfolgreich verändert.
Anforderungen	FA8, FA11

Tabelle 3.6: Anwendungsfall UC05 Beziehungsverlauf bearbeiten.

3.3.6 Gruppierung bearbeiten

Elemente werden zusammen in eine Gruppe gefasst. Um dies zu ermöglichen kann der Benutzer ein Element in ein Anderes rein- oder rausziehen. Die Tabelle 3.7 beschreibt den Anwendungsfall UC06.

ID	UC06
Name	Gruppierung bearbeiten
Kurzbeschreibung	Elemente werden gruppiert oder die Gruppierung wird aufgehoben.
Akteure	Anwender
Vorbedingung	- Die Anwendung ist gestartet. - Verbindung mit dem Server ist hergestellt. - Es existieren mindestens zwei Elemente.
Auslöser	Der Anwender will eine Gruppierung bearbeiten.
Ablauf	
1. Der Anwender wählt ein Element aus. **2.** Der Anwender zieht das ausgewählte Element aus/in einer Gruppe. **3.** Die Gruppe ist erstellt/aufgehoben. **4.** Anwendungsfall ist beendet.	
Alternativer Ablauf	
-	
Nachbedingung	Es wurde erfolgreich eine Gruppe erstellt/aufgehoben.
Anforderungen	FA6

Tabelle 3.7: Anwendungsfall UC06 Gruppierung bearbeiten.

3.3.7 Element/Beziehung löschen

Der Benutzer hat die Möglichkeit sowohl Elemente als auch Beziehungen aus dem Hauptbereich zu löschen. Nachdem eine Element oder eine Beziehung ausgewählt wurde, kann durch einen Button im Menu diese/dieser gelöscht werden. Im Fall bei dem ein Element in Beziehung zu einem anderen Elementen steht, sollen die Abhängigkeiten gelöscht werden. Zum Beispiel: Element A hat eine Beziehung X zu Element B. Zusammen mit Element A wird auch Beziehung X gelöscht. Ein Anderes Beispiel: Eine Gruppe A besteht aus Element B. Beim Löschen der Gruppe A wird auch das Element B gelöscht. Die Tabelle 3.8 beschreibt den Anwendungsfall UC07.

ID	UC07
Name	Element/Beziehung löschen
Kurzbeschreibung	Es soll ein Element oder eine Beziehung vom Hauptbereich gelöscht werden.
Akteure	Anwender
Vorbedingung	- Die Anwendung ist gestartet. - Verbindung mit dem Server ist hergestellt. - Es existiert mindestens ein Element/Beziehung im Hauptbereich.
Auslöser	Der Anwender will ein Element/Beziehung löschen.
Ablauf	
1. Der Anwender wählt ein Element/Beziehung aus. **2.** Der Anwender klickt auf den Button Löschen. **3.** Das System überprüft das zu löschende Element/Beziehung. **4.** Das System löscht das Element/Beziehung. **5.** Das System aktualisiert den Hauptbereich aller Anwender. **6.** Anwendungsfall ist beendet.	
Alternativer Ablauf	
4.a Der Anwender hat ein Element ausgewählt: Das System löscht alle Beziehungen des zu löschenden Elements.	
Nachbedingung	Ein Element/Beziehung wurden vom Hauptbereich gelöscht.
Anforderungen	FA4

Tabelle 3.8: Anwendungsfall UC07 Element/Beziehung löschen.

4 Design und Entwurf

Im folgenden Kapitel werden die Anforderungen aus der Analyse weiter spezifiziert. Das Kapitel beginnt mit einer Vorstellung des Systems anhand eines Mockups. Danach werden die Funktionen der Komponenten beschrieben. Zudem wird eine Architekturübersicht dargestellt.

4.1 Mockups

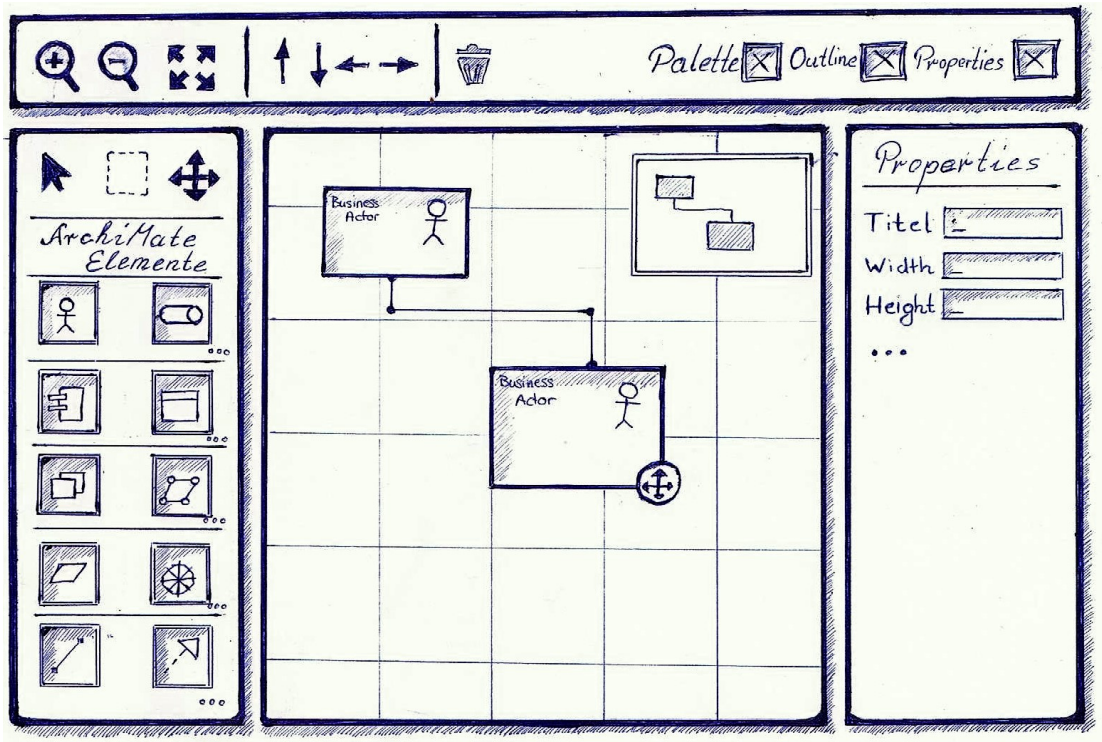

Abbildung 4.1: Das Mockup der Webapplikation, bestehend aus Hauptbereich, Menu, Palette, Eigenschaftenbereich und Outline.

Um eine klarere Vorstellung des Konzepts zu ermöglichen, wird das Mockup für die Applikation erstellt. Beim Abrufen der Internetseite im Browser soll die Applikation wie auf dem Mockup (siehe Abbildung 4.1) angezeigt werden. Aufgrund der Anforderung FA10 besteht die Applikation aus den fünf Bereichen: Menu, Palette, Hauptbereich, Outline und Eigenschaftenbereich.

4.1.1 Menu

Das Menu ist am oberen Rand positioniert. Die Steuerelemente für das Vergrößern und Verkleinern der Sicht vom Hauptbereich (FA1) sind als Lupen dargestellt. Durch Betätigen des Buttons mit den vier Pfeilen wird die Sicht auf die Standardgröße zurückgesetzt. Die vier Buttons mit den Richtungssymbolen sind für die Bewegung der Sicht im Hauptbereich zuständig. Zudem zeigen die abgebildeten Symbole jeweils die Richtung an. Durch Klicken des Buttons mit dem Mülleimersymbol sollen ausgewählte Elemente gelöscht werden (FA4). Zusätzlich hat der Benutzer die Möglichkeit die Palette, das Outline oder die Eigenschaften mit den drei Checkboxen an- oder auszublenden (FA12).

4.1.2 Palette

Durch die Aktivierung der Paletten-Checkbox wird unter dem Menu auf der linken Seite die Palette angezeigt. Darin sollen alle Archimate Elemente aufgelistet werden (FA2, FA9), wobei jedes Element durch ein Icon und ein Tooltip zu identifizieren ist. Die Archimate Schichten (Applikation Layer, Business Layer, Technology Layer, Extensions Layer) sollen visuell getrennt werden. Zudem sind die drei Steuerelemente Auswahl, Gruppenauswahl und Bewegung enthalten. Durch das Auswahl- bzw. Gruppenauswahl-Steuerelement, bekommt der Benutzer die Möglichkeit ein bzw. mehrere Elemente im Hauptbereich auswählen zu können. Das Betätigen des Bewegungs-Steuerelement ermöglicht dem Benutzer das Verschieben der Sicht im Hauptbereich in allen Richtungen (FA1). Das ausgewählte Steuerelement soll durch eine farbige Markierung zu erkennen sein.

4.1.3 Hauptbereich

Der Hauptbereich liegt mittig unter dem Menu. In diesem Bereich wird kollaborativ an der Enterprise Architektur gearbeitet. Für die bessere Positionseinschätzung befindet sich im Hintergrund ein Gittermuster (FA11).

4.1.4 Outline

Durch die Aktivierung der Outline-Checkbox im Menu wird am oberen rechten Rand des Hauptbereichs das Outline angezeigt. Dessen Aufgabe liegt darin die Enterprise Architektur zu navigieren (FA1).

4.1.5 Eigenschaften

Durch die Aktivierung der Eigenschaften-Checkbox im Menu werden unter dem Menu auf der rechten Seite der Eigenschaftenbereich (in der Abbildung 4.1 mit Properties gekennzeichnet) angezeigt. Es sollen zum Beispiel die Eigenschaften wie Text, Farbe oder Größe der ausgewählten Elemente dargestellt werden (FA5).

4.2 Architektur

4.2.1 Architekturübersicht

Die Architektur der Applikation richtet sich nach der Flux Architektur. Zudem wird diese mit dem Client-Server Modell erweitert.

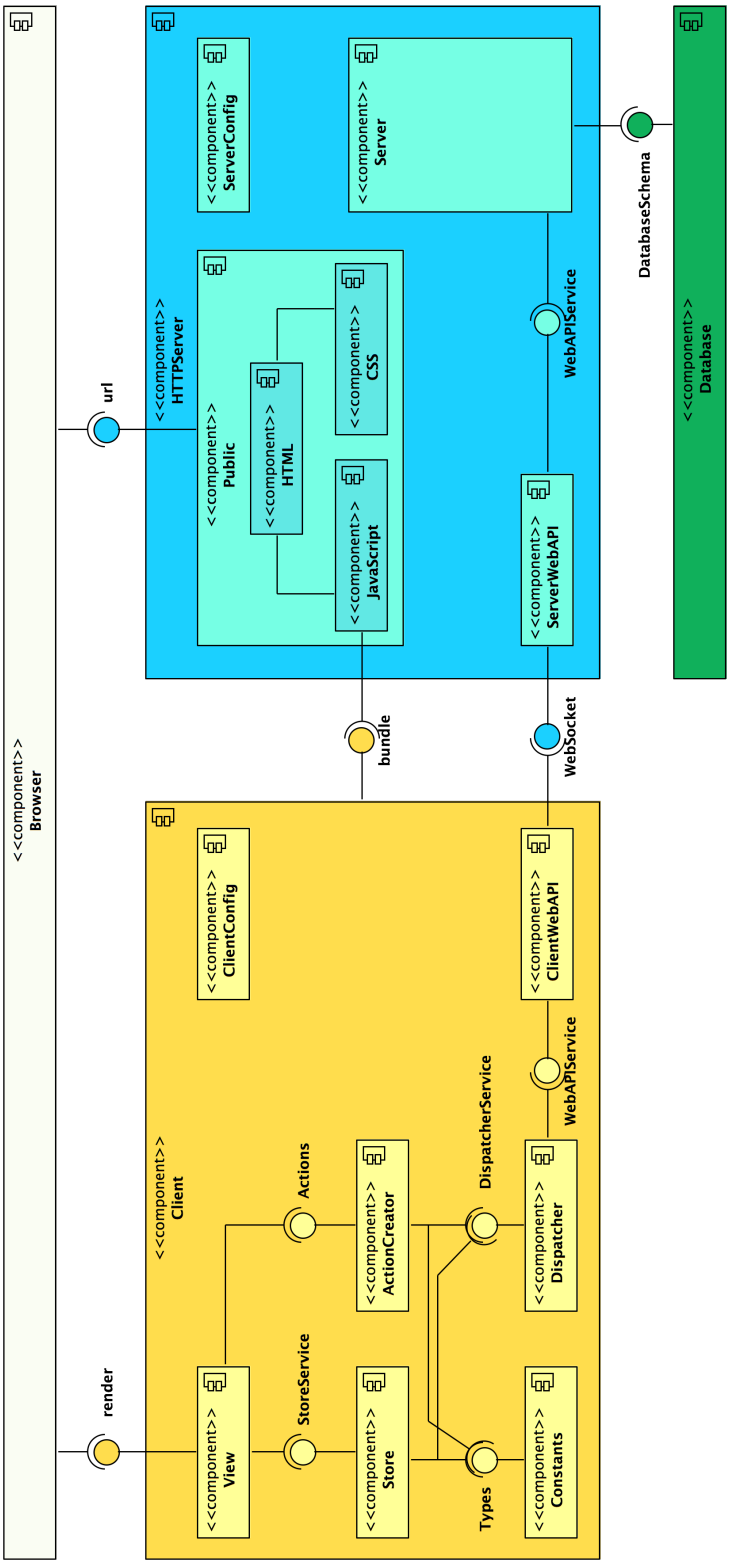

Abbildung 4.2: Das Komponentendiagramm zeigt die Webapplikation in der Blackbox Sicht.

In Abbildung 4.2 wird ein Komponentendiagramm als Blackbox Sicht mit den Hauptkomponenten der Applikation dargestellt. Diese bestehen aus Browser (Weiß), Client (Gelb), HTTPServer (Blau) und Database (Grün). Im Nachfolgenden werden die einzelnen Komponenten beschrieben.

Browser Für die Benutzeroberfläche soll ein Webbrowser verwendet werden. Der Browser kann auf die offene Schnittstelle (url) des HTTPServers zugreifen. Diese Schnittstelle ist für die Auslieferung der HTML, CSS und JavaScript Dateien verantwortlich. Durch die gelieferte Javascript Datei, wird dem Browser die Render-Schnittstelle des Clients zur Verfügung gestellt.

Client Die Client Komponente soll als umgewandelte Javascript-Datei (bundle) zum HTTPServer zur Verfügung gestellt werden. Zudem wird die FLUX-Architektur in der Client-Komponente umgesetzt. Dazu zählen die Komponenten View, ActionCreator, Store, Dispatcher und Constants. Wobei die Constants-Komponente als Hilfskomponente für die Kommunikation zwischen ActionCreator und Store dienen soll. Da das System keine reine FLUX Applikation und zudem das Client-Server Modell mit umgesetzt werden soll, dient die ClientWebAPI als Client. Diese Komponente soll sich mit dem Server verbinden. Zudem sollen in der ClientConfig Komponente Konfigurationsinformationen gehalten werden.

View Die View beschreibt den virtuellen DOMs. Dazu sind in Abbildung 4.3 die Unterkomponenten detaillierter dargestellt. Außerdem bietet die View Komponente die Render-Schnittstelle nach außen an. Die Aktionen soll die View von der ActionCreator Komponent (Actions) holen. Um die Aktionen zu erhalten, steht der gewünschte Store über die StoreService Schnittstelle (StoreService) zur Verfügung.

ActionCreator In der ActionCreator-Komponente sind alle Aktionen definiert die von der View aufgerufen werden können. Die Aktionen werden dabei jeweils mit einem Typ (Types) versehen und an den Dispatcher (DispatcherService) weitergereicht.

Dispatcher Der Dispatcher soll die Aktionen auf die passenden Stores verteilen. Dazu wird die Schnittstelle DispatcherService bereitgestellt. Zudem soll die Komponente globale Aktionen an die WebAPI (WebAPIService) weiterleiten.

Store Im Store soll der Zustand gespeichert werden. Dazu bietet jeder Store die Möglichkeit diesen zu abbonieren oder zu stornieren (StoreService). Der Store wird bei einem Dispatcher registriert. Bei einer Aktion soll der Store über die DispatcherService Schnittstelle

die Information erhalten. Zudem soll anhand des Types entsprechend auf die Aktion reagiert werden.

Constants Die Constants Komponente dient als Hilfskomponente zur Realisierung der Fluxarchitektur. Es werden Typen von Aktionen definiert und angeboten.

ClientWebAPI Die ClientWebAPI dient zur Kommunikation zwischen dem Server und Client. Diese Komponente befindet sich auf der Clientseite und bietet diesem die Schnittstellen für globale Aktionen zu Verfügung. Die Komponente soll auf die Schnittstelle der ServerWebAPI zugreifen.

HTTPServer Die Komponenten Public, ServerConfig, Server und ServerWebAPI bilden zusammen die HTTPServer Komponente. Zudem sollen in der ServerConfig Komponente Konfigurationsinformationen gehalten werden.

Public (HTML, CSS, JavaScript) Die Public Komponente beinhaltet die drei benötigten Dateien für den Browser HTML, CSS und JavaScript. Diese werden auch als Komponenten dargestellt. In der HTML Komponente wird der DOM in einer Baumstruktur definiert. Die CSS Komponente wird für das Design benötigt. Zudem gibt es die JavaScript Komponente, diese ist eine Zusammengefasste Form der Client Komponente (bundle). Die Public Komponente stellt die drei Komponenten dem Browser zu Verfügung (url).

ServerWebAPI Die Komponente ServerWebAPI dient als Server im Client Server Model. Es soll Anfragen von Clients entgegen nehmen. Daher ist Komponente für die Kommunikation zu den Clients zuständig und bietet die WebSocket Schnittstelle an. Die Nachrichten, die über die WebSocket Schnittstelle laufen, sollen an den Server weitergeleitet werden (WebAPIService).

Server Die Server Komponente ist für die Logik verantwortlich. Diese ist mit der WebAPI-Service Schnittstelle von ServerWebAPI verbunden. Für das Speichern der Daten greift die Komponente auf die Schnittstelle (DatabaseSchema) der Datenbank (Database) zu.

Database Die Komponete Database ist für die Persistierung zuständig. Dazu wird die Schnittstelle DatabaseSchema angeboten.

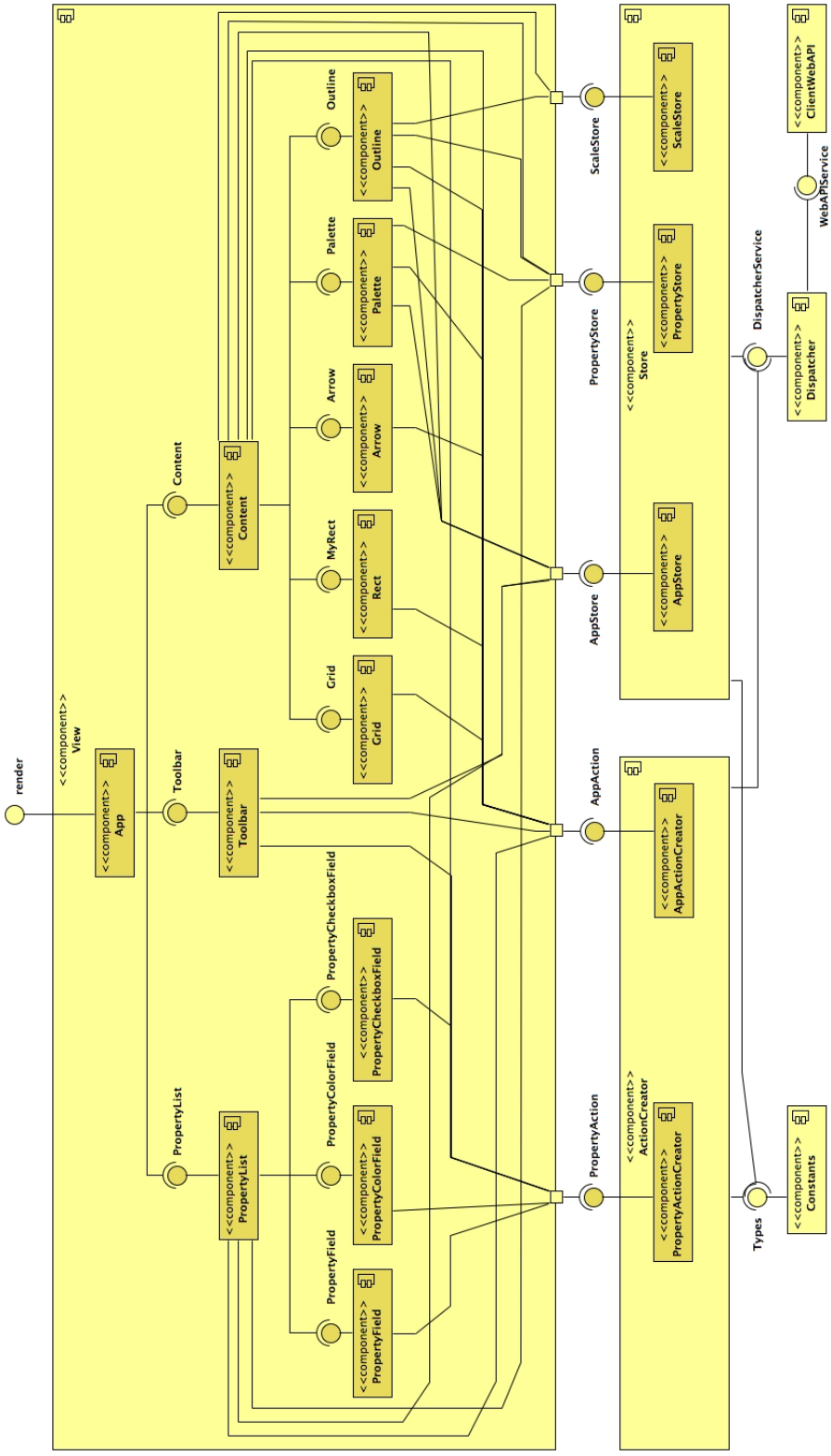

Abbildung 4.3: Das Komponentendiagramm zeigt die Clientkomponente in der Whitebox Sicht.

Das zweite Komponentendiagramm (siehe Abbildung 4.3) stellt die Innensicht der Clientkomponente als Whitebox dar. Im Folgenden werden die einzelnen Komponenten beschrieben.

View Die View Komponente beinhaltet mehrere React Komponenten die in einer Baumstruktur dargestellt sind. Diese Struktur spiegelt den Virualen DOM wieder. Zudem wird nur eine Render Funktion nach außen angeboten.

App Die App Komponente ist der Wurzelknoten des Virtualen DOMs. Diese bietet eine Schnittstelle nach außen an. Von der App Komponente werden die Kinderknoten PropertyList, Toolbar und Content erstellt und gerendert.

PropertyList Die PropertyList Komponente stellt das Eigenschaftenbereich dar. Die Darstellung wird als Liste beschrieben. Dabei werden die Kinderknoten PropertyField, PropertyColorField und PropertyCheckboxField als Listenelemente erstellt und gerendert. Listelemente stellen jeweils eine Eigenschaft dar. Die Komponente selbst hat als Elternknoten die App Komponente.

PropertyField Die PropertyField Komponente dient als Listenelement der PropertyList. Zudem wird die Komponente bei Eigenschaften benötigt, die eine Text oder Zahleneingabe fordern. Daher beinhaltet die Komponente einen Label und ein Textfeld. Weiterhin hat die Komponente keine React Komponente als Kinderknoten.

PropertyColorField Die PropertyColorField Komponente dient als Listenelement der PropertyList. Zudem wird die Komponente bei Eigenschaften benötigt, die eine Farbauswahl fordern. Daher beinhaltet die Komponente eine Farbpalette. Weiterhin hat die Komponente keine React Komponente als Kinderknoten.

PropertyCheckboxField Die PropertyColorField Komponente dient als Listenelement der PropertyList. Zudem wird die Komponente bei Eigenschaften benötigt, die entweder an- oder ausgeschaltet werden können. Daher beinhaltet die Komponente einen Label und eine Checkbox. Weiterhin hat die Komponente keine React Komponente als Kinderknoten.

Toolbar Die Toolbar Komponente ist für die Darstellung des Menu verantwortlich. Die Komponente selbst hat als Elternknoten die App Komponente. Weiterhin hat die Komponente keine React Komponente als Kinderknoten.

Content Die Content Komponente ist eine der drei Kinderknoten von der App Komponente. Diese dient als Wurzelknoten des Hauptbereichs. Zudem werden in ihr die Komponente Grid, Rect, Arrow, Palette und Outline erstellt und gerendert.

Grid Die Grid Komponente ist für die Darstellung des Raster zuständig. Zudem ist die Content Komponente der Elternknoten davon. Weiterhin hat die Grid keine React Komponente als Kinderknoten.

Rect Die Rect Komponente ist für die Darstellung der Elemente im Hauptbereich zuständig. Zudem ist die Content Komponente der Elternknoten davon. Weiterhin hat die Grid keine React Komponente als Kinderknoten.

Arrow Die Arrow Komponente ist für die Darstellung der Beziehungen verantwortlich. Zudem ist die Content Komponente der Elternknoten von Arrow. Weiterhin hat die Komponente keine React Komponente als Kinderknoten.

Palette Die Palette Komponente ist für die Darstellung des Palette verantwortlich. Zudem ist die Content Komponente der Elternknoten davon. Weiterhin hat die Komponente keine React Komponente als Kinderknoten.

Outline Die Outline Komponente ist für die Darstellung des Outlines verantwortlich. Zudem ist die Content Komponente der Elternknoten davon.

ActionCreator Innerhalb der ActionCreator Komponente befinden sich zwei spezifische ActionCreator Komponenten. Zu einem die PropertyActionCreator, welcher alle Aktionen die für das Eigenschaftenbereich gelten beinhaltet. Die Andere ist die AppActionCreator, in der sich die Restlichen Aktionen die für die Webapplikation relevant sind befinden.

Store Innerhalb der Store Komponente befinden sich drei spezifische Store Komponenten. Zum einem der PropertyStore, welcher den Zustand für das Eigenschaftenbereich beinhaltet. Der zweite Store ist der AppStore, in der sich der Zustand aller Archimate Elemente befinden. Der letzte Store ist ScaleStore. Dieser hat den Sichtzustand für den Hauptbereich. Darin wird die Vergrößerung oder Verschiebung der Sicht gespeichert.

4.2.2 Datenbankschema

Damit die Wiederherstellbarkeit von State nach dem Serverneustart möglich wird, ist die Speicherung vom aktuellen Zustand in einer Datenbank notwendig. Im Vergleich zu rationalen Datenbanken, hat die MongoDB ein flexibles Schema. Da ein Schema vom Zustand ständig durch Einfügen oder Löschen von neuen Attributen ändern kann, wird für die Verwendung von einer MongoDB NoSQL Datenbank entschieden. (literatur: https://docs.mongodb.org/manual/core/data-modeling-introduction/) In dem Listing 5.10 ist das entwickelte Schema dargestellt. Dieses besteht aus einen Zeitstempel (timestamp), den zuletzt vergebenen Schlüssel eines Elements

(lastElementId) und eine Menge von erstellten Elementen (elements). Wenn eine Zeit nicht eingegeben wurde, wird die Systemzeit vom Server genommen.

```
STATE_ITEMS =
    {
        timestamp:
        {
            type:       Date,
            default:    Date.now
        },
        lastElementId:  Number,
        elements:       Elements
    }
```

Listing 4.1: Das Datenbankschema bestehend aus STATE_ITEMS.

4.2.3 Benutzerinteraktionen

Die Benutzerinteraktionen teilen sich in lokalen und globalen Interaktionen ein. Bei der lokalen Interaktion laufen die Prozesse clientseitig ab, sodass die anderen Anweder davon nichts mitkriegen. Zu dieser Gruppe gehören folgende Interaktionen: Hauptbereich vergrößern, Hauptbereich verkleinern, Bewegung des Haupbereichs, Element auswählen das Ein- und Ausblenden von der Palette, Eigenschaftenbereich und Outline.

Bei der zweiten Gruppe handelt es sich um die Aktionen wie zum Beispiel: Element erstellen, bewegen, löschen und die Elementeneigenschaft ändern.

Hierbei wird jede Interaktion zum Server geschickt und an alle Clients verteilt.

4.2.4 Schnittstellenbeschreibung

Für die Webapplikation wird das Client Server Modell angewendet. Der Server bietet Dienste an, die der Client daraufhin anfordert. Um die Kommunikation zwischen Client und Server zu gewährleisten, müssen klare Schnittstellen beschrieben werden. Die Anwendung besteht aus mehreren Schnittstellen.

4.2.4.1 Zustand übermitteln

Bei Änderungen des Zustands informiert der Server alle verbundenen Clients. Dazu sendet der Server ein Broadcast mit dem aktuellen Zustand. Zusätzlich hat der Client die Möglichkeit den aktuellen Zustand selbst abzurufen.

```
1 Client.send("updateState")
2 Server.send("currentState", state)
```

Listing 4.2: Schnittstellenbeschreibung Zustand übermitteln

updateState Der Client ruft den aktuellen Zustand des Servers ab. Der Server sendet dem Client über die Schnittstelle currentState den Zustand.

currentState Der Server kann entweder allen verbundenen oder einen einzelnen Client den aktuellen Zustand zusenden. Der Zustand wird im Beispiel mit „state“ benannt.

4.2.4.2 Element erstellen

Das Erstellen eines neuen Elements soll jeder angemeldete Client mitbekommen. Daher informiert ein Client bei der Erstellung zuvor den Server. Dieser sendet die Information als Broadcast weiter an alle angemeldeten Clients.

```
1 Client.send("addElement", element)
2 Server.send("addElement", element)
```

Listing 4.3: Schnittstellenbeschreibung Element erstellen

addElement Die Schnittstelle wird sowohl vom Client als auch vom Server verwendet. Der Client sendet zum Erstellen eines Elements die Nachricht mit dem zu erstellen Element an den Server. Der Server leitet beim Empfangen der Nachricht diese weiter an alle verbundenen Clients.

4.2.4.3 Element bearbeiten

Jede Änderung eines Elements sollen alle angemeldeten Clients mitbekommen. Vor einer Änderung wird daher eine Nachricht zum Server gesendet.

```
1 Client.send("changeElement", element)
2 Client.send("changeElements", elements)
3 Server.send("changeElement", element)
4 Server.send("changeElements", elements)
```

Listing 4.4: Schnittstellenbeschreibung Element bearbeiten

changeElement Die Schnittstelle wird sowohl vom Client als auch vom Server verwendet. Bei der Veränderung eines Elements wird diese Schnittstelle verwendet.

changeElements Die Schnittstelle wird sowohl vom Client als auch vom Server verwendet. Bei der Veränderung mehrerer Elemente wird diese Schnittstelle verwendet.

4.2.4.4 Element löschen

Das Löschen eines Elements sollen alle Clients mitbekommen.

```
1 Client.send("deleteElement", element)
2 Server.send("deleteElement", element)
```

Listing 4.5: Schnittstellenbeschreibung Element löschen

deleteElement Die Schnittstelle wird sowohl vom Client als auch vom Server verwendet. Das zu löschende Element wird zusammen in einer Nachricht geliefert.

4.2.5 Programmablauf

Der Programmablauf der Webapplikation ist in den folgenden Sequenzdiagrammen beschrieben. Die Abbildung 4.4 und 4.5 dokumentieren wie sich die Applikation beim Starten verhalten soll.

Das Starten der Applikation ist der Übersicht Halber in zwei Diagramme aufgeteilt. Die Abbildung 4.4 beschreibt den ersten Teil. Zuerst öffnet der Anwender den Browser seiner Wahl und gibt dort die URL der Webapplikation ein. Nach Bestätigung der Eingabe wird eine Anfrage an den HTTPServer gesendet. Dieser sendet daraufhin die HTML, CSS und JavaScript Dateien an den Browser. Mit den gelieferten Dateien kann der Browser die Webseite aufrufen. Dabei wird die Render Funktion der React Rootkomponente gestartet. Die React Komponente wird über die *React.createClass* erstellt. Bei der Erstellung wird der React Lebenszyklus befolgt. Zuerst wird die *getDefaultProps* Funktion gestartet. Dannach kommt die *getInitialState* Funktion. Darin kann der Zustand einer Komponente initialisiert werden. Als drittes wird die Funktion *componentWillMount* aufgerufen. Hier soll der gewünschte Store abonniert werden. Es ist sinnvoll das Abonnieren in dieser Funktion zu implementieren, da diese Funktion nur einmal im Lebenszyklus einer React Komponente aufgerufen wird und zwar bei der Erstellung. Nachdem die Funktion vollendet ist, wird das Rendern durchgeführt. Zum Schluss wird die *componentDidMount* Funktion aufgerufen. In dieser Funktion soll der aktuelle Zustand vom Server abgerufen werden. Dazu wird die *updateState* Funktion vom *AppActionCreator* aufgerufen. Dieser holt sich den passenden Aktionstypen von *Constants*. Da der Store auf die Aktionstypen matched, kann die Aktion somit identifiziert werden. Im Szenario heißt der Aktionstyp *UPDATE_STATE*. Die *AppActionCreator* Komponente ruft zudem die Funktion *updateState* auf und übergibt dieser den Aktionstypen. Der *Dispatcher* deligiert die Funktion an die *ClientWebAPI* weiter. Dieser soll die Aktion zum Server senden. Jedoch wird zuvor ein Verbindungsaufbau gestartet.

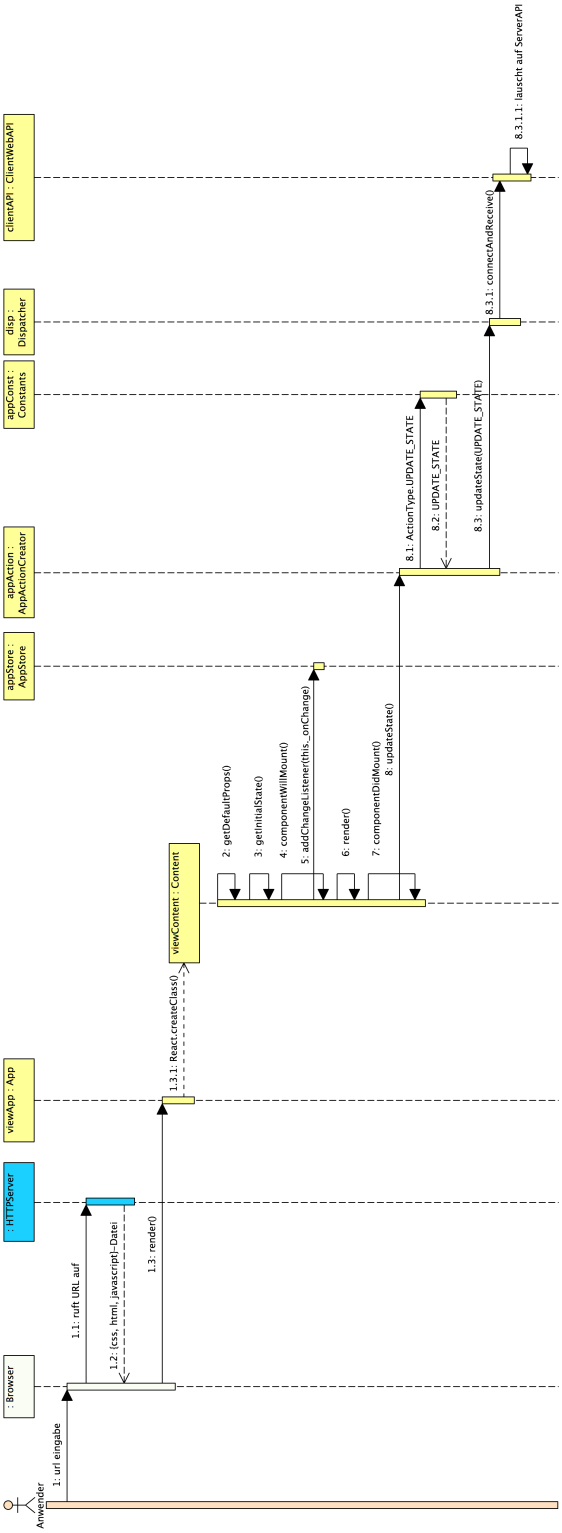

Abbildung 4.4: Sequenzdiagramm zeigt die Anmeldung des Clients (Teil 1).

Der zweite Teil des Startens der Applikation ist in Abbildung 4.5 beschrieben. Für den Verbindungsaufbau sendet der Server der ClientWebAPI die Nachricht *"connect"* zu. Nachdem die ClientWebAPI mit dem Server verbunden ist, stellt dieser die Anfrage *üpdateState"* auf den aktuellen Zustand. Daraufhin sendet im der Server dann den Zustand über *{currentState, state}*. Nach dem Empfangen des Zustands, baut die ClientWebAPI ein Aktionpaket aus dem zugesendeten Zustand und einen Aktiontypen. Deie Aktionstypen werden von Constants über *ActionType.CURRENT_STATE* geholt. Dannach wird vom Dispatcher die *dispatch* Funktion mit dem Aktionspaket aufgerufen. Dadurch wird die Nachricht an den Store weiterdeligiert. Der Store erhält *{CURRENT_STATE, state}* und matched diese Nachricht, indem der gleiche Aktionstyp von Constants geholt wird. In diesem Fall ist es *CURRENT_STATE*. Beim erfolgreichen Matchen ruft der Store seine interne Funktion *updateState* auf. Diese aktualisiert den alten Zustand des Stores mit dem zugelieferten Zustand. Zusätzlich werden alle Abonnenten über die Zustandsänderung informiert, indem die Funktion *_onChange* bei allen Abonnenten aufgerufen wird. Daraufhin rufen die Abonnenten die Storefunktion *getCurrentState* auf und erhalten so den aktuellen Zustand. Dieser wird in der View mit *setState* gesetzt. Die *setState* Funktion bewirkt zusätzlich das erneute Rendern der View.

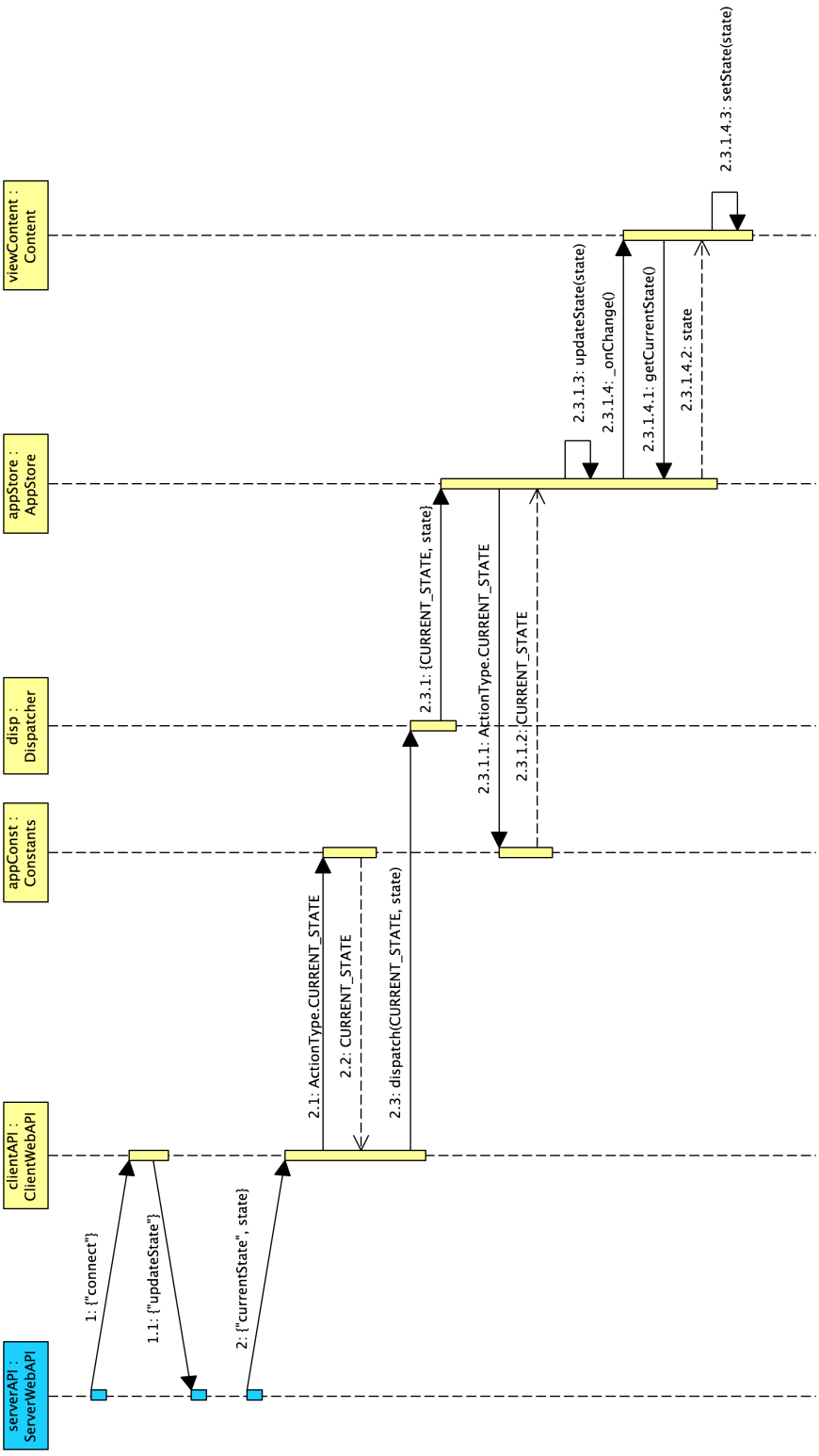

Abbildung 4.5: Sequenzdiagramm zeigt die Anmeldung des Clients (Teil 2).

Die Sequenzdiagramme stellen dar das Starten der Webapplikation als Szenario. Dabei wird zunächst eine Verbindung zwischen Server und Client aufgebaut. Nach den Verbindungsaufbau sendet der Client dem Server eine Anfrage auf dem aktuellen Zustand. Dies ist notwendig, da die React Komponenten Zustandslos an den Browser gesendet werden und der Zustand so nachgereicht wird. Zudem ist der Vorgang bei allen Aktionen stets der gleiche. Sobald ein Ereignis geschieht sendet die View der Action eine Nachricht. Je nachdem ob das ein globales oder lokales Ereignis ist, wird die Nachricht entweder vom Dispatcher an die WebAPI oder direkt zum Store delegiert. Trotzallem wird die Nachricht vorher am Dispatcher vorbeilaufen. Fall die Nachricht an den WebAPI weitergeleitet wird, erhält sie der Server, damit dieser die Nachricht an allen verbunden Clients senden kann. Am Ende erhält der Store die Nachricht. Dieser verarbeitet die Nachricht und gibt den entsprechenden Views Bescheid.

5 Implementierung

5.1 Visualisierung

5.1.1 Struktur der Visualisierung

Die EAM Diagramme können aus mehreren Elementen bestehen, die gemeinsam sehr komplexe Grafiken repräsentieren. Das Hauptelement in der Webapplikation soll Canvas sein. Die Nutzung von Canvas kann viele Schwierigkeiten mit sich bringen. Dies ist der Grund, warum die Entscheidung auf ein Framework fiel, dass die Arbeit erleichtern soll.

Konva.js ist ein HTML5 Canvas JavaScript Framework für Desktop- und Mobile-App. [Kon16b] Konva unterstützt die Arbeit mit dem zweidimensionalen Zeichnen und die Ereignisbehandlungen.

Zudem besteht Konva aus den Komponenten State, Layer, Group und Shape. Die Stage Komponente wird nur einmal erzeugt und dient als Rootknoten, der die Benutzerlayers (Layer) beinhaltet. Jedes Layer ist ein Canvas-Element. Zusätzlich kann ein Layer aus Shapes, Gruppen (Group) von Shapes oder aus einer Gruppe von Gruppe bestehen. Konva.js unterstützt viele Basisshapes unter anderen sind es Kreise, Rectangle, Ellipse, Linie, Bild und Text. Die Abbildung 5.1 stellt die Hierarchie von Konva dar. Stage, Layers, Groups, und Shapes sind virtuelle Knoten. Vergleichbar mit dem DOM Knoten in einer HTML-Seite. Bei jedem Knoten kann der Stil geändert werden. [Kon16b]

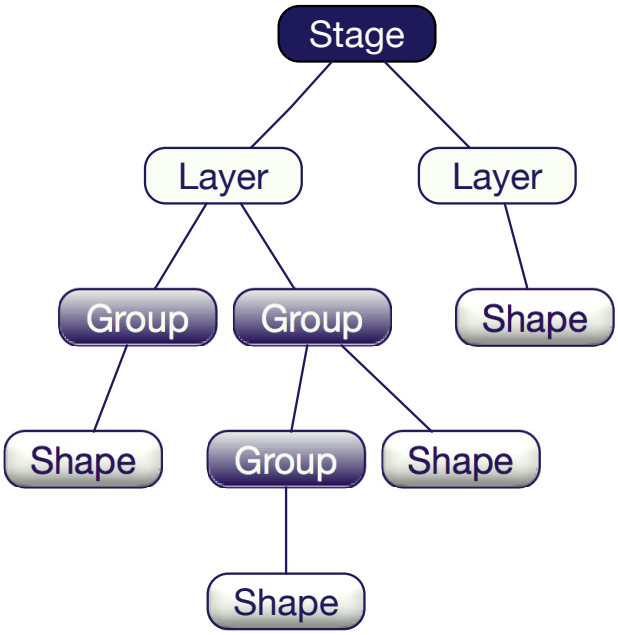

Abbildung 5.1: Beispiel Struktur von Konva.

Für die Nutzung des Konva Frameworks in React, wird React-Konva JavaScript Bibliothek verwendet. Die Bibliothek untestützt Mobile und Desktoperreignisse wie Mouseenter, Click, Dblclick, Dragstart, Dragmove, Dragend, Tap, Dbltap usw.[Kon16a]

In der Webapplikation wurde mit zwei Stages gearbeitet. Eine Stage ist für das Outline verantwortlich und besteht aus zwei Layer. Sie befindet sich in *client/components/parts/outline*. In einem Layer sind Elemente aus dem Hauptbereich in einer Miniatur Ansicht dargestellt. Das andere Layer wird für die Darstellung des Blickfeldes vom Anwender benötigt.

Die zweite Stage wird in *client/components/parts/content* erstellt und enthält die Palette und das Hauptbereich. Damit können die Elemente aus der Palette in das Hauptbereich per Drag and Drop eingefügt werden. Zudem sind in dieser Stage fünf Layer enthalten. Im ersten Layer ist das Grid dargestellt. Alle Elemente eines Diagramms befinden sich im zweiten Layer. Das Navigieren der Sicht im Hauptbereich benötigt auch ein Layer, genau wie die Aufhebung der Selektierung eines Elements. Das letzte Layer ist für die Palette verantwortlich.

5.1.2 Attribute eines Elements

Ein Element hat folgende Attribute:

key Die Elemente eines Diagramms sind dynamisch. Mit eindeutigen Schlüssel kann von React eine Hash-Tabelle benutzt werden. Die Erstellung, Bearbeitung und Löschung eines Elements liegt somit in der Komplexitätsklasse O(n) [fac16].

id Die Elemente werden von Anwender erstellt, bearbeitet und gelöscht. Damit auf ein Element zugegriffen werden kann, wird dieses über ein Schlüssel eindeutig identifiziert.

type Dieses Attribut wird beim Rendern von Elementen gebraucht, damit ein richtiges View ausgewählt werden kann. Ein Element hat den Type 'rect.

name Hier wird der Name eines Elements gespeichert.

x, y Die Position eines Elements im Hauptbereich wird durch X,Y-Werte der oberen linken Ecke bestimmt.

width, height, minWidth, minHeight - Auch die Größe eines Elements wird festgehalten. Es ist möglich die Breite und Höhe zu verändern, so dass nicht die minimale Werten überschritten werden.

draggable Damit ein Element nicht mehr bewegen werden kann, wird draggable auf false gesetzt, sonst ist es immer true.

textName Der Title eines Elements.

textX, textY, textFontSize - Die Position eines Titels in einem Element wird durch textX,textY-Werte bestimmt. Die Größe des Titels ist durch textFontSize bestimmt.

color, textFill Hier wird die Farbe eines Elements und des Titels gespeichert.

groupElements Wenn ein Element zur einer Gruppe wird, dann werden die ID's von gruppierten Elemente in diesem Attribute als Array gespeichert.

groupId Wenn ein Element zu einer Gruppe hinzugefügt wird, dann ist die ID des Elements, in dem festgehalten.

5.1.3 Darstellung von ArchiMate Elemente

Im Ordner *client/components/diagrams/archiMate/* ist der Quellcode zur grafischen Darstellung von Archimate Elemente zu finden. Die Archimate Elemente sind vielfältig und unterscheiden sich in Form, Farbe und Icons. Die einfachste Formen haben Elemente aus den Technology

Layer. Die Form von den Elementen ist ein Rechteck. In der Abbildung 5.2 sind alle Elemente der Technology Layers dokumentiert.

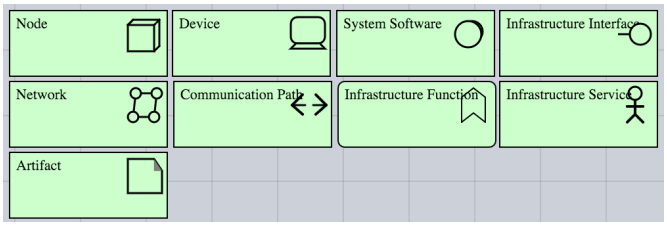

Abbildung 5.2: Ein Ausschnitt der Webapplikation mit allen Elementen vom Technology Layer.

Das Application Layer hat Rechtecke mit abgerundeten sowie mit normalen Ecken. Die abgerundene Ecken werden durch die Eigenschaft *cornerRadius={15}* ermöglicht. Die Abbildung 5.3 stellt das Application Layer dar.

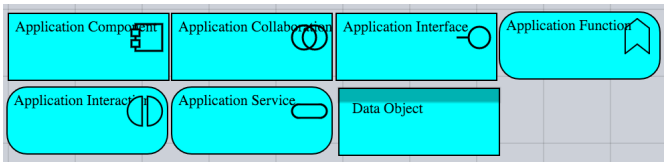

Abbildung 5.3: Ein Ausschnitt der Webapplikation mit allen Elementen von Application Layer.

Beim Business Layer sind zwei weitere Formen für die Elemente Meaning und Representation verfügbar. Die Abbildung 5.4 zeigt alle Elemente des Business Layers.

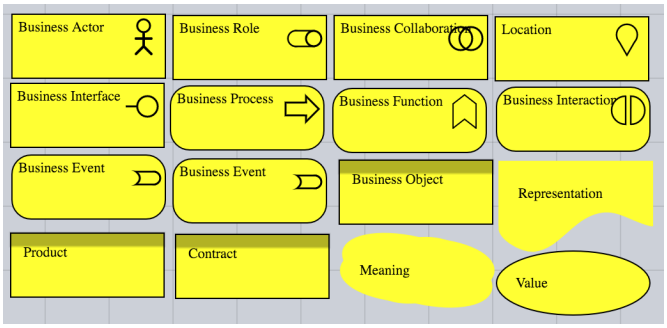

Abbildung 5.4: Ein Ausschnitt der Webapplikation mit allen Elementen von Business Layer.

Meaning wird mit Hilfe von drei Elipse erzeugt. Wobei Represenation durch eine Kombination von zwei Linien erstellt wird. Eine Linie mit den Eigenschaften *closed = true* und

tension=0.4, damit es die Runde Formen erhält. In der Abbildung 5.5 sind Komponenten eines Representationelements.

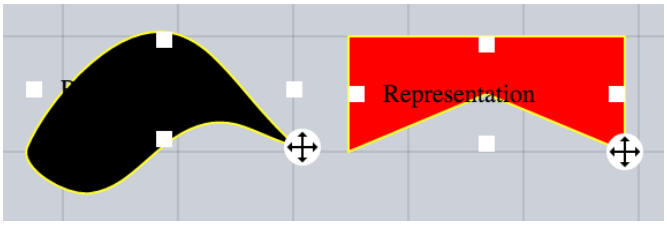

Abbildung 5.5: Komponenten des Representation Elements.

In Extension Layer ist die Form, ein Rechteck mit abgeschnittenen Ecken. Dies wird auch mit Hilfe einer Linie erstellt. Die Attribute sind dabei *closed=true* und *tension=0*. In der Abbildung 5.6 werden alle Elemente des Extensions Layers dargestellt.

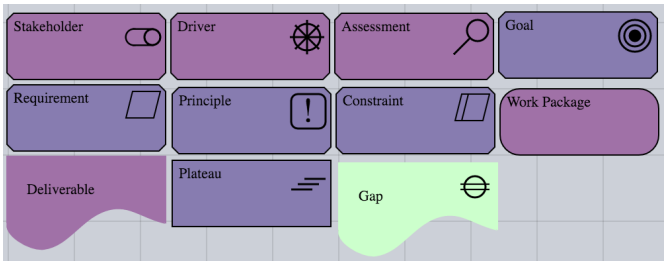

Abbildung 5.6: Ein Ausschnitt der Webapplikation mit allen Elemente von Extension Layer.

Außerdem besitzt jedes Archimate-Element vier Ports: oben und unten, links und rechts. Die ein- und ausgehende Beziehungen werden an diesen Ports automatisch zugeordnet. Die Ports werden so ausgewählt, dass dadurch ein kürzester Weg der entsprechenden Beziehung entsteht.

5.1.4 Qualitäten der Visualisierung eines ArchiMate Elementes

Um zu Testen wie stark die Qualität der Visualisierung auf die Performanz wirkt, werden Elemente mit niedriger Qualität erstellt. Somit wird eine undetaillierte Sicht erzeugt. Diese befinden sich im Ordner *client/components/diagrams/archiMate/business/lowQuality*.

Eine detaillierte Form beinhaltet einen Name und als Icon wird eine Gruppe von Konva Elemente verwendet. Die Gruppe besteht aus Kreis und Linien. Bei der undetaillierter Form wird auf den Namen verzichtet und an der Stelle eines Icons, ein Bild mit 26x26 Pixel aus

den Ordner *public/images/archiMate/businessLayer/* verwendet. Die Abbildung 5.7 stellt zwei Business Actor Elemente in detaillierter und undetaillierter Form dar.

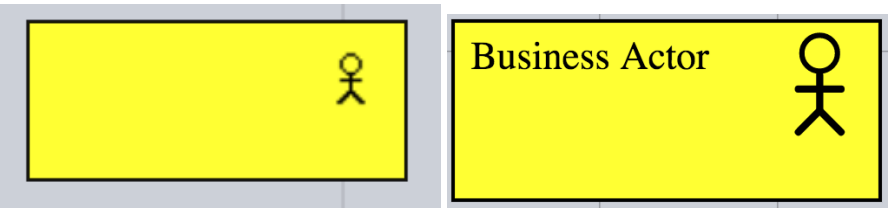

Abbildung 5.7: Beispiel für undetaillierte (links) und detaillierte (rechts) Darstellung von Element Business Actor.

In *client/components/shapes/rect* wird zwischen den Qualitäten der Darstellung eines Elements entschieden. Wenn das Verkleiner des Hauptsicht auf weniger als 100 Prozent kommt, werden die Elemente in undetaillierter Sicht dargestellt, ansonsten detailliert.

5.1.5 Erzeugung von ArchiMate Elemente

Bei der Erstellung eines Elements in *client/components/parts/content*, bekommt das Element alle für die Darstellung wichtigen Attribute wie Position, Farbe, Größe, oder Name. Zusätzlich werden auch die Werte zum Navigieren des Hauptbereichs mitgeteilt. Die Werte werden bei der Bearbeitung eines Elements berücksichtigt. Zum Beispiel, bei der Bewegung eines Elements wird die Position eines Elements wie folgt berechnet:

Die Berechnung von X-Wert: $X = (Position.X - Verschiebung.X)/Skalierung.X$.

Die Berechnung von Y-Wert: $Y = (Position.Y - Verschiebung.Y)/Skalierung.Y$.

Die Werte *Position.X* und *Postion.Y* entsprechen den aktuellen Koordinaten der Mouseposition. Die Werte der Bewegung des Hauptbereichs sind *Verschiebung.X* und *Verschiebung.Y*. Für das Vergrößern oder Verkleiner sind *Skalierung.X* und *Skalierung.Y* zuständig.

5.1.6 Attribute einer Beziehung

Eine Beziehung hat folgende Attribute:

key - Die Beziehungen einer Diagramm sind dynamisch. Mit einem eindeutigen Schlüssel kann von React eine Hash-Tabelle benutzt werden. Die Erstellung, Bearbeitung und Löschung eines Elements liegt somit in der Komplexitätsklasse O(n) [fac16].

id - Die Beziehungen werden von Anwender erstellt, bearbeitet und gelöscht. Damit auf ein Element zugegriffen werden kann, wird dieses über ein Schlüssel eindeutig identifiziert.

type - Dieses Attribute wird beim Rendern von Beziehungen gebraucht, damit ein richtiges View ausgewählt werden kann. Eine Beziehung hat den Type 'arrow'.

name - Der Name einer Beziehung ist in diesem Attribute zu finden.

fromElement - Hier ist die ID des Start-Elements hinterlegt.

toElement - Hier ist die ID des Quelle-Elements hinterlegt.

points - Hier werden die Positionen von Knoten einer Beziehung abgespeichert.

5.1.7 Darstellung von ArchiMate Beziehungen

Die Beziehungen von Archimate unterscheiden sich in der Darstellung. Konva bietet die Pfeile an, welche aber nicht alle Beziehungen von Archimate abdecken können. Der Grund dafür ist, dass die Beziehungen von Archimate unterschiedlicher Zeiger und Style der Linien besitzen. Es werden die neue Beziehungtypen erstellt. In dem Ordner *client/components/diagrams/archiMate/relationships* sind alle erstellte Beziehungen zu finden. In Abbildung 5.8 werden alle Beziehungen grafisch aufgezeigt.

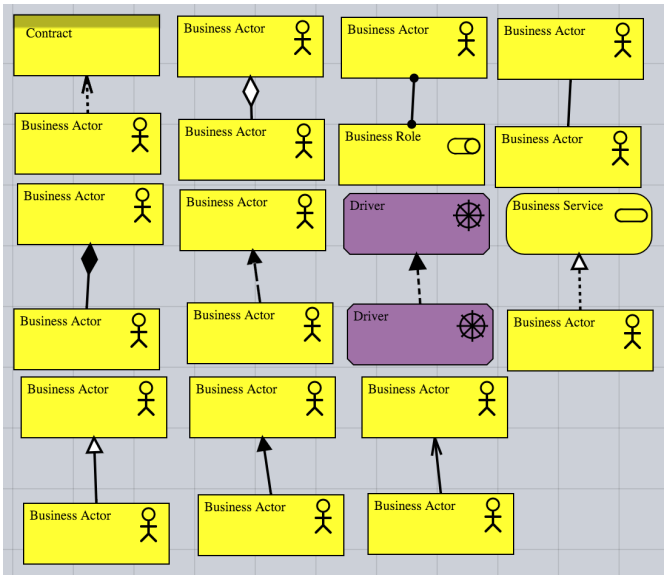

Abbildung 5.8: Ein Ausschnit der Webapplikation mti allen Elemente von Business Layer.

Bei der gerichtete Beziehungen muss der Zeiger eines Pfeils richtig positioniert werden. Wenn ein Element oder Knoten einer Beziehung sich in eine andere Position verschoben wird,

soll der Zeiger einer Beziehung an den Beziehungsverlauf angepasst werden. Der Zeiger soll um einen bestimmten Winkel α (Referenz Abbildung 5.9) gedreht werden.

Die Abbildung 5.9 zeigt ein Beispiel mit zwei Elementen mit einer Beziehung.

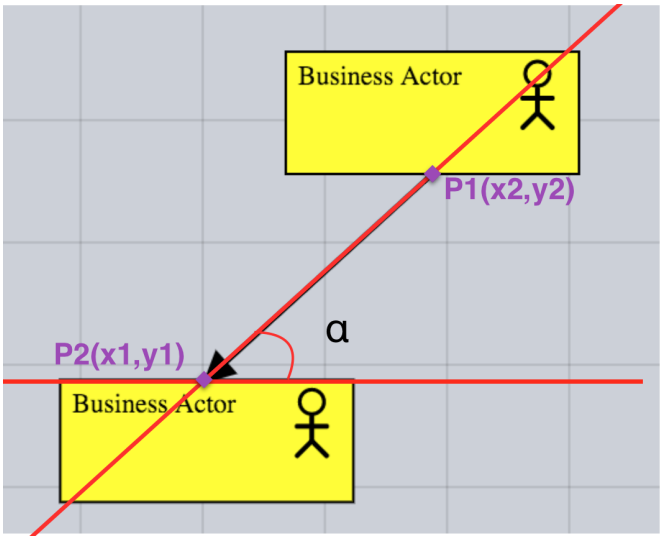

Abbildung 5.9: Ein Ausschnit der Webapplikation mti allen Elemente von Business Layer.

Die Beziehung besteht aus zwei Koordinaten. Die erste Koordinate ist vom Zeiger P2(x1,y1) und die zweite Koordinate ist der Startpunkt einer Beziehung. Für ein besseres Verständnis werden zwei Hilfsgeraden in roter Farbe angezeigt. Eine Gerade verläuft durch die Elementen Beziehung. Die andere Geraden wird als X-Achse betrachtet. Der Winkel zwischen den beiden Geraden ist der gesuchte Winkel. Die Aufgabe reduziert sich Schnittwinkelberechnung einer Gerade und der X Achse. Zuerst wird den Anstieg der Geraden bestimmt. Da die Gerade über P1 und P2 verläuft, kann die Steigung durch Formel berechnet werden:

$m = (P2_{y_2} - P1_{y_1})/(P2_{x_2} - P1_{x_1})$

Die Berechnung des Steigungswinkels lautet:

$\tan(\alpha) = m$

Jetzt kann den Arkustangens gebildet werden, dass der eine Zahl im Radiantenmaß liefert. Um den Grad zu bekommen, wird nach folgender Formel den zugehörigen Winkel berechnet:

$\alpha = \arctan(m) * 180/\pi$

Die Steigung kann positive oder negative Werte annehmen. Im Fall eines negativen Wertes und wenn P1x1 größer als P2x2 ist, muss zusätzlich 180 Grad addiert werden. Im Fall eines positiven Wertes und wenn P1x1 größer als P2x2 ist, muss 180 Grad subtrahiert werden.

Der Zeiger wird somit um den Winkel α (Referenz Abbildung 5.9) durch das Attribute *rotation=alpha* routiert.

5.1.8 Palette

Die Komponente *client/components/parts/palette.jsx* ist für die Palette zuständig. Die Palette befindet sich mit Hauptbereich auf einer Stage. Hier ist es notwendig die Palette im Vordergrund zu positionieren, damit die Elemente nicht auf der Palette liegen. Das wird durch Attribute *zindex* für die ganze Gruppe ermöglicht. Die Icons für die Elemente sind in der Datei *public/images/icons.gif* gespeichert. Die Behandlung von Ereignissen wird in *client/components/parts/paletteItem.jsx* implementert. Dazu zählt das Drag and Drop und die Selektierung eines Elements.

5.1.9 Resize

Die Komponente *client/components/shapes/anchor.jsx* ist für die Veränderung der Größe eines Elements verantwortlich. Die grafische Darstellung besteht aus einer Gruppe von vier Pfeilen und einen Kreis. Bei der Selektierung eines Archimate Elements wird die Gruppe durch ein Attribute *visible* eingeschaltet. Die Selektierung wird durch Props *visibleAnchor* mitgeteilt.

5.1.10 Grid

Um ein Raster im Hauptbereich zu erhalten, wird ein neuer Layer erstellt. In diesem Layer befindet sich eine Gruppe mit horizontalen und vertikalen Linien. Der Abstand zwischen den Linien ist von der Vergrößerung oder Verkleinerung des Hauptbereichs abhängig. Das Grid wird bei der Vergrößerung, Verkleinerung oder Verschiebung des Hauptbereichs neu gerendert, weil das Hauptbereich eine unbestimmte Breite und Höhe hat. Der Quellcode für das Grid ist in der Datei *client/components/shapes/grid.js* gespeichert.

5.1.11 Eigenschaftenbereich

Für den Eigenschaftenbereich wurden zwei Bibliotheken verwendet: Material-UI und React-Color-Picker. Die Material-UI [mu] Bibliothek bietet eine Menge von React Komponenten an, die direkt übernommen werden können. Zudem ist es an das von Google entwickelte Material Design [Goo] gerichtet. Diese Bibliothek wurde ausschließlich für das Design der Webapplikation verwendet. Im Eigenschaftenbereich wurden dabei die List Komponenten eingefügt. Diese dienten zur Auflistung von den Eigenschaften eines Archimate-Elements im Diagramm. Zudem sollte die Auswahl der Farbe im Eigenschaftenbereich ermöglicht werden. Jedoch

bietet Material-UI keine passende Komponente dazu an. Daher wurde die React-Color-Picker [zip] Bibliothek zusätzlich implementiert. Diese ermöglicht das Auswählen der Farbe durch Mausklick in einer Farbpalette. Der Quellcode für das Eigenschaftenbereich liegt im Ordner *client/components/parts/* und beinhaltet die Dateien *propertyListMD.jsx*, *propertyFieldMD.jsx*, *propertyColorFieldMD.jsx*, *propertyCheckboxFieldMD.jsx*.

5.1.12 Menu

Das Menu dient zur Steuerung der Webapplikation. Für die Erstellung wurde wie auch in Eigenschaftenbereich die Bibliothek Material-UI [mu] verwendet. Zudem wurde die Toolbar Komponente übernommen. Der Quellcode für das Menu ist in der Datei *client/components/menu/toolbar.jsx* gespeichert.

5.2 Client-Server Kommunikation

Die meisten Verfahren für das Erreichen einer höheren Interaktivitäten und Echtzeitfähigkeit von Webanwendungen haben Nachteile im kontinuierlichen Anfragen von Daten, was zu hohen Datenmengen führen kann. [PLG15] Die geringere Latenzen werden durch Websockets ermöglicht. Sockets bauen eine persistente Verbindung zwischen Client und Server. Dabei können beide Seiten jederzeit mit dem Senden von Daten beginnen.

Die Kommunikation zwischen Client und Server wird in dieser Arbeit mit der JavaScirpt-Bibliothek Socket.io Version 1.4.5 realisiert. Socket.io bietet die Möglichkeit eigene Events zu definieren. Damit der Server weiß, dass die Verbindung noch nicht beendet wurde, schickt jede verbundene Socket in Intervallen ein Hearbeat an den Server. [PLG15] Bei der inkorrekter Verbindungsabbruch trennt der Socket.io die zugehörige Verbindung automatisch ab. [PLG15] Somit kennt der Server alle verbundene Clients und kann mit Broadcast an alle Clients die Nachrichten mit neuen Inhalte abschicken.

5.2.1 WebAPI Client

In dem Listing 5.1 ist ein kleiner Ausschnitt aus der Datei *client/webapi/socketIO.js* zu finden. Damit *Socket.IO* auf der Client Seite benutzt werden kann, wird das Modul *socket.io-client* eingebunden.

Um mit dem Server eine Verbindung aufzubauen, wird die Methode *io.connect* mit Serveradresse und Port aufgerufen. Die Adresse und Port vom Server lassen sich in der Datei *client/config.js* konfigurieren.

Nach der Verbindung mit dem Server wird ein selbst definiertes Event *updateState* ausgelöst. Damit wird der aktuellste Zustand vom Server abgefragt. Alle Events werden mit *emit* Methode bestehend aus zwei Parameter aktiviert. Der erste Parameter dieser Methode ist der Name des Events. Der zweite Parameter ist die Nachricht selbst. Die Zeile 11 des Listing 5.1 zeigt ein Beispiel zur Erstellung eines Elements.

Die Antwort vom Server wird mit *socketIO.on* abgefangen. In der Zeile 14 des Listing 5.1 wird auf Event *currentState* gewartet. Bei dem Empfang einer Nachricht wird zu der Nachricht ein ActionType hinzugefügt und weiter an den Store transportiert.

```
1  import io from 'socket.io-client';
2  var socketIO = io.connect(config.HOST + ':' + config.API_PORT);
3
4  var Socket = {
5      connectAndReceive() {
6          socketIO.on('connect', (socket) => {
7              socketIO.emit('updateState',{});
8          });
9
10         addElement(action) {
11             socketIO.emit('addElement', action);
12         },
13
14         socketIO.on('currentState', function(payload) {
15             AppDispatcher.dispatch({
16                 actionType: ActionTypes.CURRENT_STATE,
17                 state: payload
18             });
19         });
20     },
21 }
```

Listing 5.1: Ein Ausschnitt aus der ClientWebAPI.

5.2.2 WebAPI Server

Der Server wird in der Datei *server/config/config.js* konfiguriert. Es wird ein Standard Namensraum / benutzt, da der Server eine volle Kontrolle über alle Ereignisse und Nachrichten hat. Nachdem der Server gestartet wurde, fängt dieser an Server (Siehe Zeile 10 des Listing 5.2) auf dem Port zu lauschen.

In *connections* werden alle verbundene Clients mit Sesion-ID verwaltet. Bei den Ereignis *addElement* wird eine generierte ID zum neuen Element hinzugefüt und der *State* aktualisiert. Anschließend informiert der Server per Broadcast Zeile 25 des Listing 5.2 alle verbunden Clients über die Änderungen und speichert den State in der Datenbank. Mit *io.sockets* werden alle verbundene Clients ausgewählt.

```
1  var socketIO = require('socket.io');
2  var connections = [];
3  var Socket =  {
4      startListening : function(server){
5          var io = socketIO.listen(server);
6          io.sockets.on('connection', function (socket) {
7              socket.once('disconnect', function() {
8                  connections.splice(connections.indexOf(socket), 1);
9                  socket.disconnect();
10             });
11             socket.on('addElement', function(payload){
12                 payload.data.id = id;
13                 payload.data.key = id;
14                 state.elements[id]= payload.data;
15                 state.lastElementId = id;
16                 id++;
17                 io.sockets.emit('addElement', payload);
18                 var stateItem = new StateItem(state);
19                 stateItem.save(function (err) {
20                     if (err) return console.log(err);
21                 });
22             });
23             connections.push(socket);
24         }
25     }
26 }
```

Listing 5.2: Ein Ausschnitt aus ServerWebAPI.

5.3 React mit Flux

Die Webapplikation ist mit der Bibliothek ReactJS implementiert. Dazu sind die Architekturrichtlinien von FLUX realisiert. Im Folgenden ist beschrieben wie FLUX in der React-

Webapplikation integriert ist. Dabei sind ein lokales und ein globales Szenario als Beispiel gegeben. Das lokale Ereignis ist das Hineinzoomen (scaleIn) in das Hauptbereich. Das globale Ereignis ist das Erstellen eines Elements. Die Szenarien werden an den Komponenten View, Action, Dispatcher, Store und Constants gezeigt.

5.3.1 View

Die View ist eine React Komponente und nutzt deren Funktionalitäten (siehe Listing 5.3). Zudem besteht die View aus mehreren JSX Dateien die sich im Ordner *client/components/* befinden. Zunächst wird das Abonnieren des Stores für eine View in der Funktion *componentWillMount* implementiert, da diese Funktion nur einmal bei der Erstellung einer React Komponente aufgerufen wird. Für das Abonnieren ruft die View die Funktion *addChangeListener* vom Store auf. Dieser Merkt sich die View solange diese nicht mittels *removeChangeListener* vom Store entfernt wurde. Das Stornieren des Stores wird in der Funktion *componentWillUnmount* ausgeführt. Dies passiert nach dem React Lebenszyklus bevor die Komponente vom Virtualen DOM entfernt wurde. *_onChange* ist eine private Funktion der Komponente, die wird vom Store aufgerufen sobald eine Änderung passiert. Dabei wird der State mittels *setState* neu gesetzt und die Komponente wird neu gerendert.

```
componentWillMount() {
    AppStore.addChangeListener(this._onChange);
}
componentWillUnmount() {
    AppStore.removeChangeListener(this._onChange);
}
_onChange() {
    setState(
        //...
    );
}
render() {
}
```

Listing 5.3: FLUX View Realisierung in der React Komponente.

5.3.2 Action

Die Action ist ein eigenständiges JavaScript Modul, dass von der View verwendet wird, sobald ein Ereignis stattfindet. Ein Ereignis könnte eine Benutzerinteraktion mit der Webapplikation

oder eine Servernachricht sein. Zudem gibt es in der Webapplikation zwei Action Dateien, die unter *client/actions/* gespeichert sind. Die Actions beinhaltet alle Aktionen der Webapplikation. Zum Beispiel wird die lokale Aktion für das Vergrößern der Sicht im Hauptbereich in der Action aufgelistet (siehe Listing 5.4). Hierbei wird die *dispatch* Funktion des Dispatchers verwendet. Dadurch wird die Aktion direkt an den passenden Store geliefert. Ein anderes Beispiel ist die globale Aktion für das Erstellen eines Elements (siehe Listing 5.5). Hierbei wird die Funktion *addElement* vom Dispatcher verwendet. Dadurch wird die Aktion zunächst an den Server gesendet, statt direkt an einen Store. Zudem unterscheiden sich die lokale und globale Aktion nur anhand des Aufrufes des Dispatchers. Die Aktion Packete sind gleich aufgebaut und bestehen aus einen Aktionstypen und Daten. Die Aktionstypen werden in der Constants Komponente definiert.

```
1 var ActionTypes = AppConstants.ActionTypes;
2 scaleIn(){
3     AppDispatcher.dispatch({
4         actionType:     ActionTypes.SCALE_IN,
5         data:           {}
6     })
7 }
```

Listing 5.4: Flux Action Realisierung für das Vergrößern der Sicht des Hauptbereichs.

```
1 var ActionTypes = AppConstants.ActionTypes;
2 addElement(elem) {
3    AppDispatcher.addElement({
4         actionType:     ActionTypes.ADD_ELEMENT,
5         data:           elem
6    });
7 }
```

Listing 5.5: Flux Action Realisierung für das Erstellen einer Elements.

5.3.3 Dispatcher

Der Dispatcher ist ein eigenständiges JavaScript Modul, dass von Action, Store und der Client-WebAPI verwendet wird. Dieser ist unter den Pfad *client/dispatcher/dispatcher.js* gespeichert. Der Dispatcher nimmt Aktionen von der Action Komponente auf und verteilt diese entweder an Stores oder sendet die weiter an die ClientWebAPI, die dann zum Server gesendet werden. Zudem werden solche globalen Anfragen als neue Funktion zum Dispatcher angehängt, wie

im Beispiel Listing 5.6 zu sehen. Lokale Anfragen an den Dispatcher werden über die *dispatch* Funktion getätigt. Diese wird vom FLUX Dispatcher mitgeliefert (siehe Listing 5.6).

```
import {Dispatcher} from 'flux';
var AppDispatcher = new Dispatcher();
AppDispatcher.addElement = (action) => {
    SocketIO.addElement(action);
};
```

Listing 5.6: Flux Dispatcher Realisierung.

5.3.4 Store

Der Store beinhaltet den Zustand einer React Komponente. Zudem besteht die Webapplikation aus drei Stores *AppStore*, *ScaleStore* und *PropertyStore* die sich im Ordner *client/stores/* befinden. Das Listing 5.7 zeigt einen Ausschnitt aus der *AppStore* Datei. Dabei sind alle Stores vom Aufbau gleich. Der Zustand *_state* wird zudem im Store manipuliert und anschließend an alle Abonnenten Bescheid gegeben ,dass etwas geändert wurde. Die Abonnenten haben dann die Möglichkeit sich den aktuellen Zustand mit der *getState* Funktion zu holen. Zudem sind Abonnenten Views, die die *addChangeListener* zum abonnieren und *removeChangeListener* zum stornieren des Stores verwenden. Dabei benutzt jeder Store für das Launchen und Versenden von Ereignissen den *EventEmitter* aus NodeJS. Dies Hilft dabei, dass die Views auf den aktuellsten Zustand bleiben. Zudem registriert sich jeder Store beim Dispatcher über *AppDispatcher.register*. Somit kennt der Dispatcher jeden Store und sendet die Nachrichten von der Aktion an diese weiter. Wenn die Nachricht *payload* beim Store ankommt, werden diese zunächst auf den Aktiontype gematched. Bei dem Fall das eine Nachricht gematched wurde, wird die passende Funktion im Store aufgerufen.

```
import assign from 'object-assign';
import EventEmitter from 'events';
const CHANGE_EVENT = 'change';
var ActionTypes = AppConstants.ActionTypes;
var _state = {}
var AppStore = assign({}, EventEmitter.prototype, {
    getState() { return _state; },
    emitChange() { this.emit(CHANGE_EVENT); },
    addChangeListener(callback) {
        this.on(CHANGE_EVENT, callback);
    },
```

```
12      removeChangeListener(callback) {
13          this.removeChangeListener(CHANGE_EVENT, callback);
14      },
15      addElement(elem) {
16          var tmpState = this.getState();
17          var tmpElements = tmpState.elements || {}
18          tmpElements[elem.id] = elem;
19          tmpState.elements = tmpElements;
20          this.setState(tmpState);
21          this.emitChange();
22                  this.dropElement(elem);
23      },
24  };
25  AppDispatcher.register(function(payload){
26      switch(payload.actionType){
27          case ActionTypes.ADD_ELEMENT:
28              AppStore.addElement(payload.data);
29              break;
30      };
31  });
```

Listing 5.7: Flux Store Realisierung im Beispiel des AppStores.

5.3.5 Constants

Die Constants Komponente definiert Aktionstypen, die von den Actions und Stores verwendet werden. Zudem befinden sich alle Constants Dateien im Ordner *client/constants/*. Für die Definition der Aktionstypen wird die Bibliothek *keyMirror* verwendet. In Listing 5.8 ist ein Beispiel für die Anwendung von *keyMirror* gegeben, wie es auch in der Webapplikation verwendet wird.

```
1  import keyMirror from 'keymirror';
2  var AppConstants = {
3      ActionTypes: keyMirror({
4          ADD_ELEMENT: null,
5          SCALE_IN: null,
6      }),
7  };
```

Listing 5.8: Constants Realisierung mit keyMirror.

5.4 Verifizierung

Da nicht jede Beziehung zwischen zwei Archimate-Elemente möglicht ist, soll bei der Erstellung einer Beziehung zuerst überprüft werden, ob die Beziehung valide ist. Im Anhang sind die Tabelle 1, 2 und 3 für die Bestimmung der Validität einer Beziehung zwischen zwei Elemente dargestellt. Zum Beispiel, zwischen ein Business Actor als Start-Element und Business Role als Ziel steht 'fiotu' Beziehungen möglich. In der Archimate Spezifikationen sind die Beziehungstypen durch einen Buchstaben definiert. Im Folgenden sind diese aufgelistet:

a - access

c - composition

f - flow

g - aggregation

i - assignment

n - influence

o - association

r - realization

s - specialization

t - triggering

u - used by

D.h. zwischen Business Actor und Business Role existieren fünf valide Beziehungen: Flow, Assignment, Association, Triggering und Used by.

Die Abkürzung für die Beziehungstypen wurde ber der Implementierung beibehalten. In der validateRelationship Methode (Siehe das Listing 5.9) wird überprüft, ob die *relationship*-Beziehung zwischen *from*-Element und *to*-Element möglicht ist. Zuerst wird ein Beziehungsschlüssel erstellt, der aus den Namen von beiden Elementen mit Minus als Trennzeichen besteht. Die Überprüfung wird an *getRelationshipCode*-Methode weiter delegiert. Hier wird die Abkürzung für die Beziehungstypen anhand der Beziehungsschlüssel bestimmt und zurückgeliefert, in der dann die Position von *relationship* mit Hilfe von der *indexOf* Methode bestimmt werden. Die Beziehung ist nicht valide, wenn die Position auf -1 gesetzt ist.

```
validateRelationship(from, to, relationship) {
    var key = from.name + '-' + to.name;
    var relCode = this.getRelationshipCode(key);
    if (relCode !== undefined) {
        return relCode.indexOf(relationship.code) > -1;
    }
    return true;
},
getRelationshipCode(key) {
    switch(key){
        case 'Business Actor-Business Actor': return 'cfgostu';
        case 'Business Actor-Business Role': return 'fiotu';
        ...
    }
}
```

Listing 5.9: Ein Ausschnitt aus dem AppStore.

5.5 Datenbank

Die URL einer Datenbank wird in der Datei *server/config/config.js* konfiguriert. Es wurde für die Umsetzung Mongoose gewählt, da es ein asynchrones Arbeiten ermöglicht. In dem Listing 5.10 wird das Schema, wie im 4.2.2 beschrieben, definiert.

```
var mongoose = require('mongoose');
var stateItemSchema = new mongoose.Schema({
        timestamp: { type: Date, default: Date.now },
        lastElementId: Number,
        elements: mongoose.Schema.Types.Mixed
    });
var StateItem = mongoose.model('stateItem', stateItemSchema);
module.exports = StateItem;
```

Listing 5.10: Ein Ausschnitt aus der Datei stateItem.js.

In der Datei *server/db/database.js* wird eine Verbindung mit der Datenbank aufgebaut. In dem Listing 5.11 ist die Einbindung von Mongoose und die Verbindung mit der MongoDB-Datenbank dargestellt.

```
var mongoose = require('mongoose');
```

```
var config = require('./../config/config');
var StateItem = require('./models/StateItem');
mongoose.connect(config.DB_URL, function(){
     mongoose.connection.db.dropDatabase();
     console.log("connect");
});
```

Listing 5.11: Ein Ausschnitt aus der Datei database.js.

Die Abbildung 5.10 zeigt ein Beispiel mit drei Elementen von Typ "Business Actor", ein Element von Typ „Business Role“ und einer Beziehung von Typ „Used by“ zwischen Business-1 und Role-1. Dabei sind Business-2 und Business-3 zusammen in dem Business-1 gruppiert. Bei der Beziehung wurde der Verlauf geändert, worauf hin zwei neu Knoten entstanden.

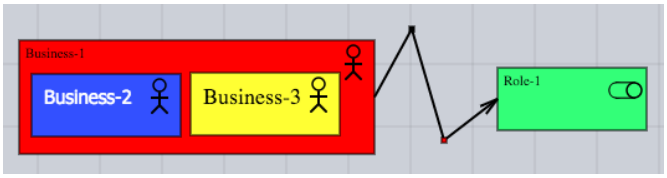

Abbildung 5.10: Beispiel für die Datenbank

In dem Listing 5.12 ist ein Ergebnis einer MongoDB-Abfrage nach dem letzten Zustand dargestellt. Die ID's für die Elemente oder die Beziehungen werden vom Server angefangen mit 0 um den Wert 1 inkrementiert. Außerdem vergibt Mongoose ein Schlüssel von Typ ObjectId (Zeile 2 des Listing 5.12) an jeden Datensatz.

Der Beziehungsverlauf (Zeile 25-26) wird durch die Attribute *points* beschrieben.In dieser sind die Koordinaten von Knoten in einer Liste gespeichert. Die Reihenfolge der Knoten beschreibt den Beziehungsverlauf vom Start zur Quelle.

Die Gruppierung ist durch *groupElements* und *groupId* organisiert (Siehe Zeile 4, 10 und 29 des Listing 5.12). Das Elternelement der Gruppe hat ein Attribute *groupElements*, in dem eine Liste von Elemente, die zu einer Gruppe gehören, aufgelistet ist. Das Attribute *groupId* hat die ID von Elternelement. Dies wird bei jedem Element, das zu einer Gruppe gehört, hinzugefügt.

```
> db.stateitems.find().limit(1).sort({timestamp:-1})
{ "_id" : ObjectId("570bb25d5a61a9bf1b6652a6"), "lastElementId" : 4,
"elements" : { "0" :
{ "minHeight":77, "minWidth":258, "groupElements":[ 1, 4 ],
"textFill":"#000", "textFontFamily":"Calibri", "textFontSize":11,
"textName":"Business-1", "name":"Business Actor", "textY":5,
```

```
"textX":5, "draggable":true, "color":"#ff0000", "height":91,
"width":286, "y":131, "x":185, "type":"rect", "id":0, "key":0 },

"1":{"minHeight":50, "minWidth":120, "groupId":0,
"textFill":"#ffffff", "textFontFamily":"Tahoma",
"textFontSize":"15", "textName":"Business-2",
"name":"Business Actor", "textY":"10", "textX":"10",
"draggable":true, "color":"#3350ff", "height":50, "width":120,
"y" : 158, "x" : 195, "type" : "rect", "id" : 1, "key" : 1 },

"2" : { "minHeight" : 50, "minWidth" : 120, "textFill" : "#000",
"textFontFamily" : "Calibri", "textFontSize" : 11,
"textName" : "Role-1", "name" : "Business Role", "textY" : 5,
"textX" : 5, "draggable" : true, "color" : "#33ff78",
"height" : 50, "width" : 120, "y" : 153, "x" : 570,
"type" : "rect", "id" : 2, "key" : 2 },

"3" : { "name" : "Used by", "toPort" : null, "fromPort" : null,
"toElement" : 2, "fromElement" : 0, "points" : [ { "y" : 122,
"x" : 501 }, { "y" : 211, "x" : 527 } ],"type" : "arrow",
"id" : 3, "key" : 3 },

"4" : { "minHeight" : 50, "minWidth" : 120, "groupId" : 0,
"textFill" : "#000","textFontFamily" : "Calibri",
"textFontSize" : "18", "textName" : "Business-3",
"name" : "Business Actor", "textY" : "10", "textX" : "10",
"draggable" : true, "color" : "#FFFF33", "height" : 50,
"width" : 120, "y" : 157, "x" : 323, "type" : "rect",
"id" : 4, "key" : 4 } },

"timestamp" : ISODate("2016-04-11T14:19:09.983Z"), "__v" : 0 }
```

Listing 5.12: Konsolenausgabe einer MongoDB Abfrage.

Beim Starten des Servers wird der letzte Zustand aus der Datenbank geholt. Dies geschieht in der Datei *server/webAPI/SocketIO.js*. In dem Listing 5.13 wird der Quellcode für die Wiederherstellung der State dokumentiert. Dabei wird der Server die ID's an den zuletzt vergebenen Schlüssel vergeben. Wenn der State noch nicht existiert, wird die Vergabe von Schlüsseln mit 0 angefangen.

```
var StateItem = require('./../db/models/StateItem');
```

```
StateItem.findOne().sort('-timestamp').exec(function(err, post){
    if (post === null){
        id = 0;
        state = {
            lastElementId: id,
            elements: {}
        };
    }else{
        state.lastElementId = post.lastElementId;
        state.elements = post.elements;
        id = state.lastElementId+1;
}});
```

Listing 5.13: Ein Ausschnitt aus der ServerWebAPI.

5.6 Prototypen

Um den Nutzen von React besser zu identifizieren, wurden zwei Prototypen entwickelt. Im Folgend werden beide Prototypen präsentiert.

5.6.1 Prototyp 1

Im ersten Prototypen wird bei jedem Rendern der Webapplikation wird die Funktion *createElement* aus Listing 5.14 für jedes Element oder jede Beziehung im Hauptbereich aufgerufen. Zudem betrachtet die Funktion zwei Fallunterschiede. Zu einem kann ein Element oder eine Beziehung im Sichtfeld des Anwenders liegen. Dabei wird das Element detailliert dargestellt bzw. gerendert. Zum anderen kann sich ein Element oder eine Beziehung außerhalb des Sichtfeldes befinden. Hierbei wird das Element in einer undetaillierten Darstellung gerendert. Die Form dieses Element ist ein einfaches Rechteck, das nur die Farbe und die Größe des Elements beinhaltet.

```
createElement(elem, renderType) {
if (!(elem === undefined || elem === null))  {
    if (elem.x+elem.width <
    -this.state.moveX/this.state.scaleX
    || elem.x >
    ((-this.state.moveX+this.state.windowWidth)/this.state.scaleX)
    || elem.y+elem.height <-this.state.moveY/this.state.scaleY
    || elem.y >
```

```
    ((-this.state.moveY+this.state.windowHeight)/this.state.scaleY)){
        return <Rect
            key={elem.id}
            id={elem.id}
            x={elem.x}
            y={elem.y}
            width={elem.width}
            height={elem.height}
            fill={elem.color}
            />;
    } else if (elem) {
        switch (elem.type) {
            case "rect":
                return <MyRect .../>;
            case "arrow":
                return <MyArrow .../>;
        }
    }
}
return <Group/>
}
```

Listing 5.14: Die *createElement* Funktion aus der content.jsx Datei.

5.6.2 Prototyp 2

Der zweite Prototyp nutzt die von der React Komponente gegebene *shouldComponentUpdate* Funktion. Diese überprüft ob eine React Komponente gerendert werden soll oder nicht. Die Erweiterung soll dafür sorgen, dass unnötiges Rendern vermieden wird. In Listing 5.15 ist die Funktion *shouldComponentUpdate* aus einer React Komponente dargestellt. Diese wird stets vor dem Rendern einer Komponente ausgeführt. Zudem werden in dieser Funktion die alten Attribute einer Komponente mit den neuen Attributen verglichen. Sobald sich mindestens ein Attribut verändert hat, wird die Komponente neu gerendert.

```
shouldComponentUpdate: function(nextProps, nextState) {
    return nextProps.selected !== this.props.selected ||
        nextProps.x !== this.props.x ||
        nextProps.y !== this.props.y ||
        nextProps.name !== this.props.name ||
        nextProps.width !== this.props.width ||
```

```
        nextProps.height !== this.props.height ||
        nextProps.color !== this.props.color ||
        nextProps.draggable !== this.props.draggable ||
        nextProps.textX !== this.props.textX ||
        nextProps.textName !== this.props.textName ||
        nextProps.textFontSize !== this.props.textFontSize ||
        nextProps.textFontFamily !== this.props.textFontFamily ||
        nextProps.textFill !== this.props.textFill ||
        nextProps.properties !== this.props.properties ||
        nextProps.icons !== this.props.icons ||
        nextProps.groupId !== this.props.groupId ||
        nextProps.groupElements !== this.props.groupElements ||
        nextProps.minWidth !== this.props.minWidth ||
        nextProps.minHeight !== this.props.minHeight ||
        ((nextProps.scaleX <1) && (this.props.scaleX>=1)) ||
        ((nextProps.scaleY >=1) && (this.props.scaleX<1));
}
```

Listing 5.15: Die *shouldComponentUpdate* Funktion von einer React Komponente.

6 Evaluierung

In diesem Kapitel wird die Umsetzung der funktionalen und nicht-funktionalen Anforderungen überprüft. Zuerst wird die Installation und der Start der Applikation beschrieben. Bei der Evaluation wird geprüft, ob die Webapplikation die Anforderungen aus Kapitel 3 erfüllt. Zusätzlich werden zwei entwickelte Prototypen aus Kapitel 5 miteinander verglichen.

6.1 Installation

Bevor das System getestet werden kann, muss zunächst die Applikation auf dem Server installiert und gestartet werden. Vor dem Installieren der Applikation, sollen folgende Software vorinstalliert sein:

- nodeJS
- mongoDB
- webpack

Die Webapplikation nutzt lokale MongoDB Datenbank für die Speicherung von Diagramme. Die Adresse vom Server, Port sowie die Adresse einer Datenbank werden in der Datei */server/-config/config.js* konfiguriert. Außerdem soll Dev-Server in der Datei *./dev-server.js* konfiguriert werden. Dev-Server aktualisiert automatisch den Browser bei der Änderung des Codes auf der Clientseite.

Listing 6.1 stellt die Konsoleneingabe für das Starten und Installieren des Servers dar. Im obersten Verzeichnis des Servers sollte ein Ordner mit der Bezeichnung *node_modules* existieren. In dem sind alle Bibliotheken oder Abhängigkeiten enthalten. Falls dieser Ordner nicht exisitiert, sollten diese Bibliotheken mit *npm install* runtergeladen werden. Damit ist die Installation vollendet.

Zum Starten des Servers sollte die Datenbank mongoDB und der Dev-Server im Hintergrund laufen. Diese kann durch die Konsoleneingabe *mongod* und *node dev-server* gestartet werden. Sobald die Datenbank läuft, kann der Server mit *npm start* gestartet werden.

Über die Server-IP und den entsprechenden Port, kann der Client auf den Server zugreifen.
Beispiel: http://localhost:8080/webpack-dev-server/public/

```
// MongoDB starten
> mongod
// Bibliotheken downloaden
> npm install
// Dev-Server starten
> node dev-server
// Server starten
> npm start
```

Listing 6.1: Installation und Start des Servers.

6.2 Systemtest

Die funktionale Anforderungen an die Webapplikation wurden manuell von Entwickler geprüft.

Die Webapplikation wird durch die Eingabe der URL des Servers im Browser gestartet. Die Abbildung 6.1 stellt einen Ausschnitt der Webapplikation dar. Entsprechend der Anforderung FA10 besteht die Applikation aus den Bereichen Hauptbereich, Palette, Menu, Outline und Eigenschaftenbereich. Die Positionierung der Bereiche wurden nach dem Mockup (siehe Abbildung 4.1) umgesetzt. Im Folgenden wird die Umsetzung der einzelnen Bereiche beschrieben.

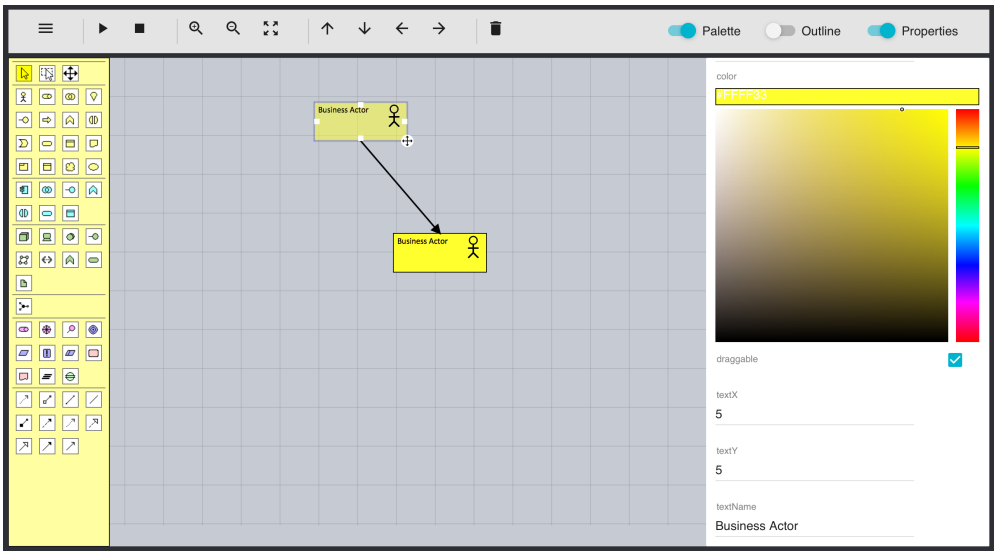

Abbildung 6.1: Die Benutzeroberfläche der Webapplikation im Überblick.

6.2.1 Menu

Das Menu befindet sich am oberen Rand (siehe Abbildung 6.2) und beinhaltet die Steuerelemente der Webapplikation.

Durch Betätigung der Buttons mit dem Lupensymbol vergößert bzw. verkleinert sich die Sicht im Hauptbereich. Dabei bleibt das Größenverhältnis aller Elemente unverändert. Die Sicht kann nur bis zu mindestens 10% verkleinert und maximal bis zu 500% vergrößert werden. Zudem kann die Sicht, durch betätigen des Buttons rechts neben den Lupensymbolen, auf 100% zurückgesetzt werden.

Durch Drücken der Pfeiltasten bewegt sich das Hauptbereich in die dementsprechende Richtung um 10%. Damit ist die Anforderung FA1 vollständig realisiert.

Bei Auswahl eines Elements oder einer Beziehung wird das Mülleimersymbol im Menu angezeigt. Dieses löscht eine Gruppe von Elemente oder ein Element mit allen Beziehungen der zu löschenden Elementen.

Das Ein- oder Auschecken einer der drei Checkboxen, blendet das entsprechende Bereich ein- bzw. aus und damit ist die Anforderung FA12 erreicht.

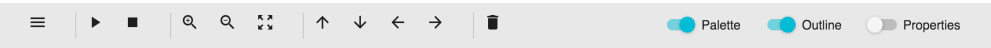

Abbildung 6.2: Die Darstellung der Menus der Webapplikation.

6.2.2 Palette

Die Palette befindet sich auf der linken Seite, wenn die Paletten-Checkbox aktiviert ist. In der Abbildung 6.3 ist ein Ausschnitt der Palette zu sehen.

Die Auflistung der Archimate Elemente in der Palette genügt der Anforderungen FA2 und FA9. Mit Mousover Effekt auf ein Element erscheint ein Tooltip, in dem ein Elementname zu sehen ist. Die Elemente sind in Archimate Schichten durch eine Linie getrennt.

Durch das ziehen eines Elements aus der Palette in das Hauptbereich wird die Anforderung FA3 erfüllt.

Zusätzlich sind die drei oberen Buttons für das Selektieren von Elementen und das Verschieben der Sicht im Hauptbereich enthalten. Das ausgewählte Steuerelement wird mit gelb gekennzeichnet

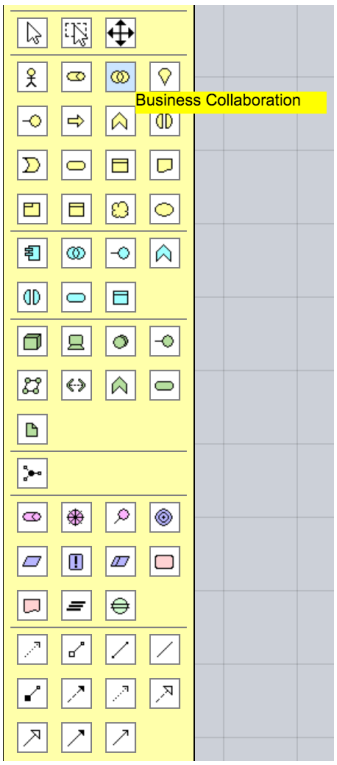

Abbildung 6.3: Beispiel der Palettendarstellung.

6.2.3 Hauptbereich

Das Hauptbereich liegt mittig unter dem Menu.

Durch die Auswahl der Bewegungsymbole in der Palette, kann das Hauptbereich mit gedrückter Maustaste verschoben werden und erfüllt somit die Anforderung FA1. Das Bewegungsymbols ist oben rechts in der Abbildung 6.3 zu sehen.

Gemäß der Anforderung FA11 ist ein Raster im Hauptbereich realisiert.

Die Größe eines Elements lässt sich im Hauptbereich ändern. Dabei wird beim Mouseover-Event auf ein Element ein Button für die Veränderung der Größe angezeigt (siehe Abbildung 6.4). Mit gedrückter Maustaste auf den Button kann die Größe des Elements angepasst werden. Somit ist die Anforderung FA5 teilweise realisiert.

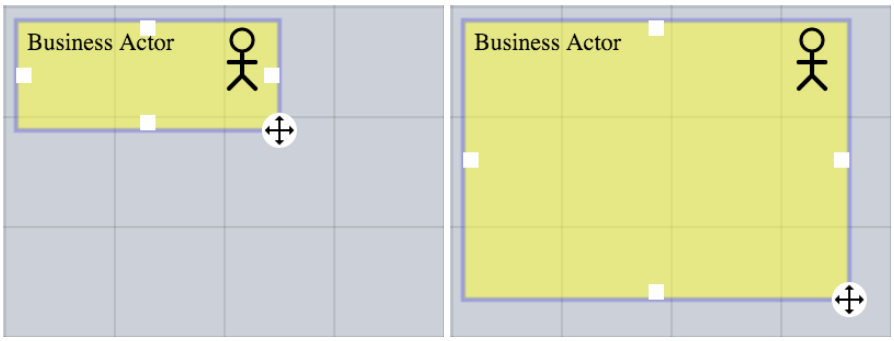

Abbildung 6.4: Beispiel für das Vergrößern eines Business Akteur Elements.

Um die Anforderung FA6 die Gruppierung von Elementen abzudecken, werden Elemente, wie in Abbildung 6.5 zu sehen, ineinander integriert. Die relative Position von gruppierten Elemente bleiben nach Verschiebung einer Gruppe erhalten. Zudem kann durch das Entfernen des Elements die Gruppierung aufgehoben werden.

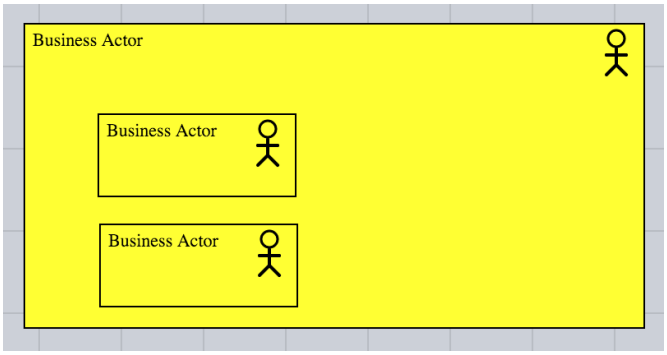

Abbildung 6.5: Beispiel für die Gruppierung mehrerer Business Akteur Elemente.

Für das Entfernen eines Elements oder einer Beziehung, muss zunächst das Element bzw. die Beziehung ausgewählt werden. Das geschieht durch einen Mausklick auf das jeweilige Element bzw. die jeweilige Beziehung. Zudem muss auf das Mülleimersymbol (siehe Abbildung 6.2) geklickt werden. Bei einem Element werden zusätzlich seine Abhänigkeiten mitgelöscht. Zum Beispiel: Beziehungen die am Element hängen werden mitgelöscht oder in Abbildung 6.5 werden bei Löschung der Gruppe die inneren Elemente auch gelöscht. Dadurch wird die Anforderung FA4 realisiert.

Für die Erstellung einer Beziehung zwischen zwei Elemente, soll zuerst einen Beziehungstyp aus der Palette ausgewählt werden. Danach werden die Quelle und das Ziel jeweils per Klick

ausgewählt. In der Abbildung 6.6 sind zwei Business Akteuren mit einer Komposition Beziehung dargestellt. Hiermit ist die Anforderung FA7 erfüllt.

Abbildung 6.6: Beispiel der Erstellung einer Beziehung zwischen zwei Business Akteur Elemente.

Wenn die Beziehung im Hauptbereich zu sehen ist, kann der Beziehungsverlauf geändert werden. Durch einen Doppelklick wird ein Knoten, über den die Beziehung verläuft, zu der Beziehung hinzugefügt. Diese kann auf eine beliebige Position verschoben werden. In der Abbildung 6.7 ist der Beziehungsverlauf durch Einfügen weiterer Knotenpunkte geändert. Die Anforderung FA8 ist damit unterstützt.

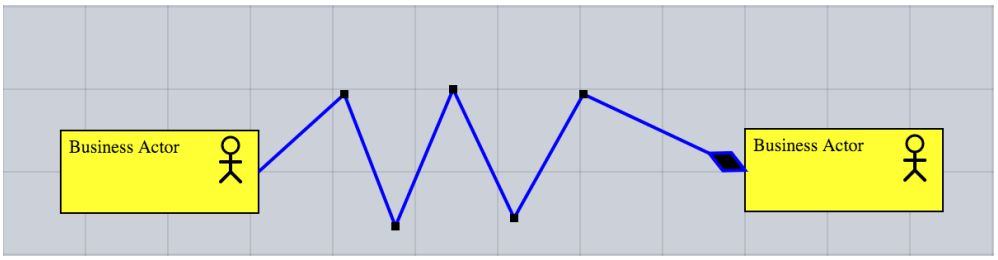

Abbildung 6.7: Beispiel der Änderung eines Beziehungsverlaufes.

6.2.4 Outline

Die Outline wird durch die Aktivierung des Outline-Checkboxes im Menu am oberen rechten Rand des Hauptbereichs positioniert. Die Abbildung 6.8 zeigt ein Ausschnitt der Outlines im Hauptbereich der Webapplikation.

Das Outline bietet die Navigierung des Hauptbereichs an. Das Bewegen sowie die Vergrößerung und die Verkleinerung des Hauptbereichs erfüllen die Anforderung FA1.

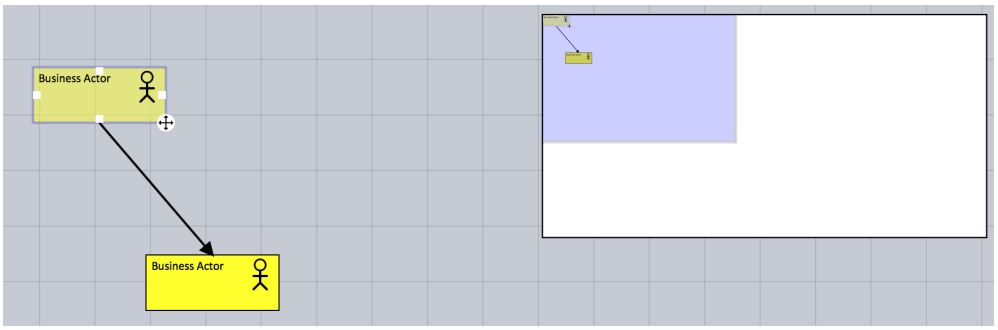

Abbildung 6.8: Beispiel der Outline Anzeige.

6.2.5 Eigenschaften

Der Eigenschaftenbereich gibt die Möglichkeit Attribute eines ausgewählten Elements zu verändern. Durch die Aktivierung der Eigenschaften-Checkbox im Menu werden unter dem Menu auf der rechten Seite der Eigenschaftenbereich angezeigt.

Es ist Möglich die Veränderung der Größe, Titel oder Farbe durchzuführen (FA5). Die Aktualisierung eines Elements passiert zeitgleich mit Eingabe des Attributwerts.

6.3 Prototypentest

In diesem Abschnitt werden zwei Prototypen getestet und miteinander verglichen. Zuerst wird die Testumgebung beschrieben. Danach kommen die durchgeführte Tests. Dabei werden die Testergebnisse mit Hilfe von Tabellen und Diagramme analysiert.

6.3.1 Versuchsaufbau

6.3.1.1 Hardware

In allen Testfällen wurde folgende Hardware verwendet:

als Server: MacBook Pro

- Betriebssystem: OS X El Capitan (Version: 10.11.3)
- Prozessor: 2,6 GHz Intel Core i5
- Speicher: 16 GB 1600 MHz DDR3
- Startvolume: Macintosh HD

- Grafikkarte: Intel Iris 1536 MB

als 1. Client: PC

- Betriebssystem: Windows 10 Education
- Prozessor: 3,4 GHz Intel Core i7-3770
- Speicher: 8 GB
- Grafikkarte: NVIDIA GeForce GTX 660

als 2. Client: MacBook Pro

- Betriebssystem: OS X El Capitan (Version: 10.11.3)
- Prozessor: 2,6 GHz Intel Core i5
- Speicher: 16 GB 1600 MHz DDR3
- Startvolume: Macintosh HD
- Grafikkarte: Intel Iris 1536 MB

Die Stationen sind über die WLAN Basisstation AirPort Time Capsule miteinander verbunden. Wobei der erste Client über ein 7,5 m LAN Kabel verbunden ist. Der Server und der zweite Client sind über das WLAN verbunden und befinden sich in 2m Entfernung zum Router, ohne Hindernisse dazwischen. Die Abbildung 6.9 veranschaulicht den Netzplan.

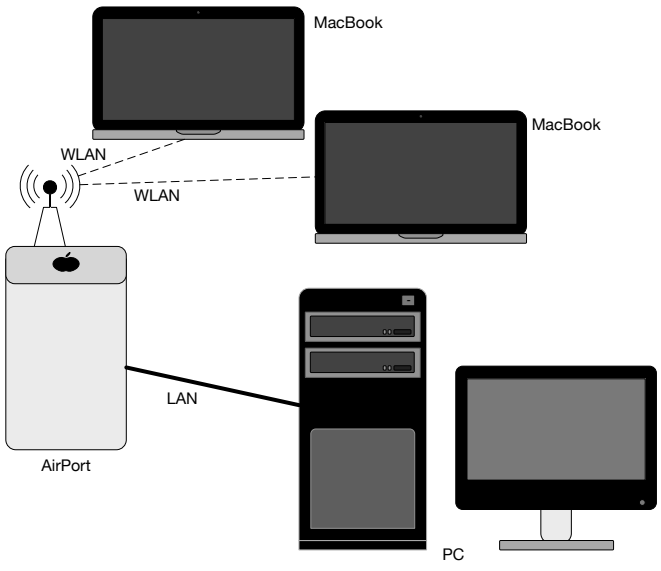

Abbildung 6.9: Der Netzplan für den Testaufbau.

6.3.1.2 Testszenario

Mit der Hilfe des JavaScript Frameworks Mocha, wurden fünf Diagramme jeweils mit 1, 250, 500, 750 und 1000 Elementen vom Typ "Business Actor"generiert. Die Breite und Höhe eines Elements ist jeweils 50 Pixel. Die Elemente wurden so positioniert, dass alle Elemente zu 100 Prozent im Sichtbereich des Hauptbereichs liegen. Jede Messung wurde zehn mal wiederholt und der Durchschnitt gebildet, um möglich zuverlässige Werte zu ermitteln. Die Webapplikation wurde im Chrome Browser getestet.

6.3.2 Testfall 1

Der erste Testfall soll zeigen, wie die Bereiche Palette, Outline und Eigenschaftenbereich die Performanz beeinflussen. Durch die Menu-Navigierung wurde die Sicht des Hauptbereichs mit und ohne Outline, Palette und Eigenschaften verschoben, sodass alle Elemente in Sicht bleiben. Dabei wurde die Zeit der Aktion gemessen.

Zuerst wurden die Diagramme mit einer detaillierten Sicht getestet. Die Messung zeigt, dass Prototyp 1 stark vom angeschaltete Bereiche abhängig ist. Mit angeschalteten Outline, Palette und Eigenschaften braucht Prototyp 1 doppelt so viel Zeit, im Vergleich zu dem Fall, wenn dies ausgeschaltet sind. Außerdem kann die Anforderung NFA5 bei Prototyp 1 mit mehr als 500 Elemente nicht erfüllt werden. Wobei Prototyp 2 bis 1000 Elemente unter 1 Sekunde liegt. In der Tabelle 6.1 sind die Ergebnisse der Menu-Navigierung in der detaillierten Sicht zu sehen. Mit Rot sind die Zeiten mit mehr als 1 Sekunde markiert.

Anzahl Elemente	**1**		**250**		**500**		**750**		**1000**	
Alle Bereich→ /Prototyp ↓	**Aus**	**An**	**Aus**	**An**	**Aus**	**An**	**Aus**	**An**	**Aus**	**An**
Prototyp1 Zeit [ms]	8,6	9,1	233,3	461,9	612,8	1110,6	805,6	1735	1352,2	2466,4
Prototyp2 Zeit [ms]	7,9	9,1	124,2	158,1	327,3	363,2	491,8	534,8	664,4	710,4

Tabelle 6.1: Erster Testfall mit Hauptbereich navigieren in der detaillierten Sicht.

Anschließend wurden Diagramme mit einer undetaillierter Sicht getestet. Im Vergleich zur detaillierten Sicht, verbessern sich die Werte von Prototyp 1 bei der undetaillierten Sicht. Die Zeit bei der Navigierung eines Diagramms mit 1000 Elemente, liegt bei 950 ms. Jedoch hat Prototyp 2 bessere Ergebnisse erzielt. Bei 1000 Elemente liegt die Zeit bei 254 ms. Beide Prototypen erfüllen hiermit die Anforderungen FA1. In der Tabelle 6.2 sind die Ergebnisse der

Navigierung in der undetaillierten Sicht zu sehen. Auch hier ist zu sehen, dass Prototyp 1 mit den angeschaltete Bereiche doppelt so viel Zeit braucht.

Anzahl Elemente	**1**		**250**		**500**		**750**		**1000**	
Alle Bereiche→ /Prototyp↓	**Aus**	**An**	**Aus**	**An**	**Aus**	**An**	**Aus**	**An**	**Aus**	**An**
Prototyp1 Zeit [ms]	2	6	81	163	185	370	295	658	482	950
Prototyp2 Zeit [ms]	4	7	47	71	90	127	160	182	180	254

Tabelle 6.2: Erster Testfall mit Hauptbereich navigieren in der undetaillierten Sicht.

Für eine bessere Übersicht über die gewonnen Daten stellt die Abbildung 6.10, die Abbildung 6.11 die Abbildung 6.12 in die Daten aus der Tabelle 6.1 und Tabelle 6.2 dar. Auf der X-Achse ist die Anzahl der Elemente aufgetragen. Auf der Y-Achse ist die Zeit von 0 bis 3000 ms aufgetragen. Das Diagramm dokumentiert den zeitlichen Vergleich der Navigierung in Prototyp 1 von einer detaillierten und undetaillierten Sicht. Bei der detaillierten Sicht liegt der Unterschied zwischen an- und ausgeschalteten Outline, Palette und Eigenschaftenbereich bei 1 Sekunde. Damit wird deutlich, dass die angeschalteten Bereiche stark die Zeit beim Rendern beeinträchtigen.

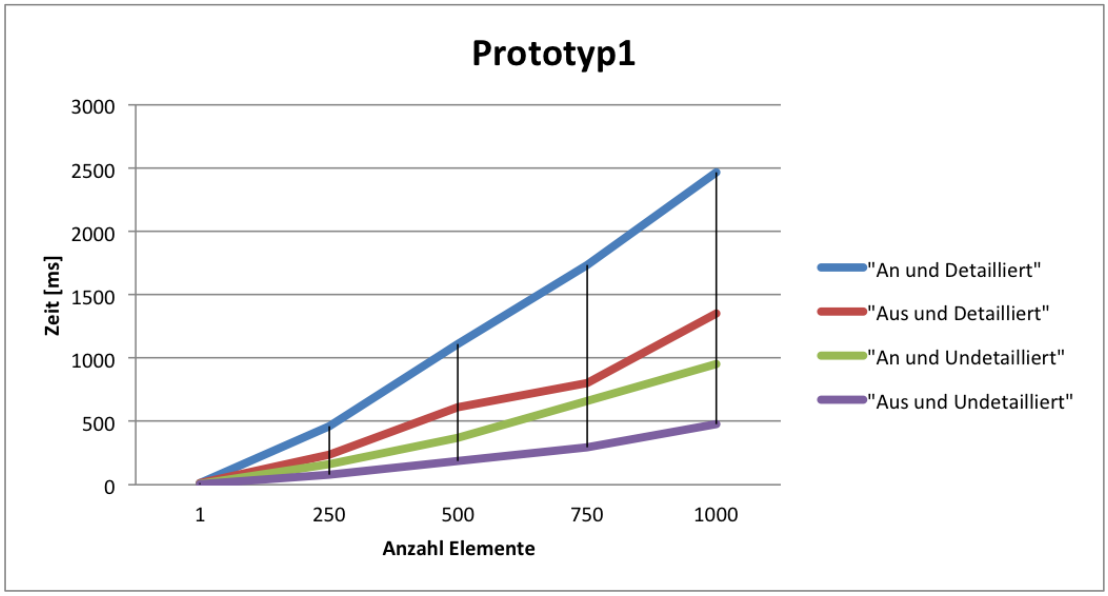

Abbildung 6.10: Prototyp 1: Vergleich der detaillierten und undetaillierten Sicht mit an- und ausgeschalteten Bereichen (zu Tabelle 6.1 und Tabelle 6.2).

In Abbildung 6.11 werden die Testergebnisse für Prototyp 2 dargestellt. Hier ist zu sehen, dass die angeschaltete Bereiche keinen Performanzverlust mit sich bringt. Auffällig ist die Abweichungen zwischen der detaillierten und undetaillierten Sicht.

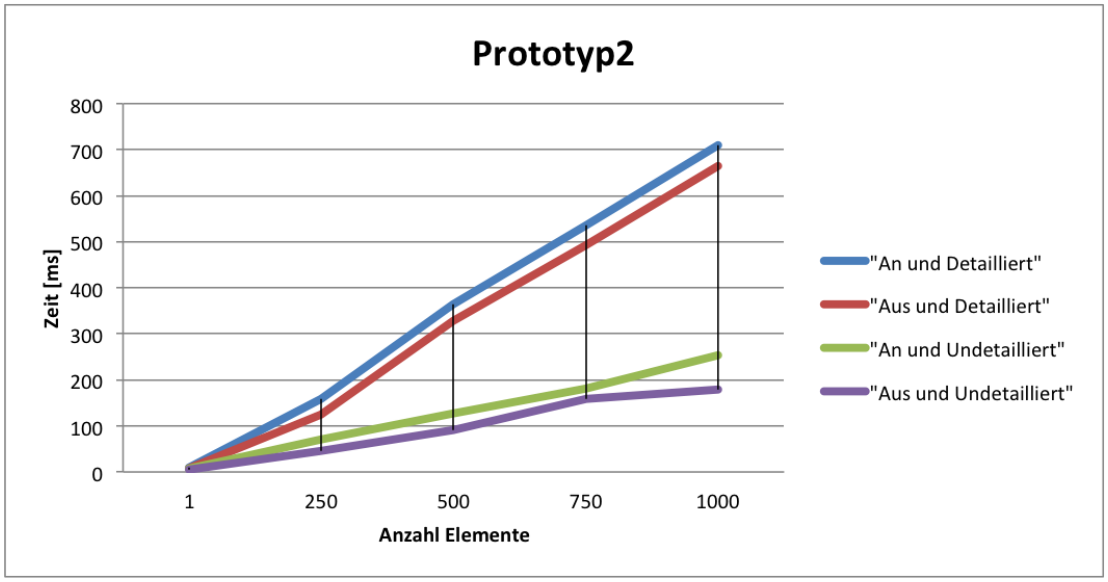

Abbildung 6.11: Prototyp 2: Vergleich der detaillierten und undetaillierten Sicht mit an- und ausgeschalteten Bereichen (zu Tabelle 6.1 und Tabelle 6.2).

Den Unterschied zwischen beiden Prototypen zeigt die Abbildung 6.12 an. Es ist zu erkennen, dass Prototyp2 die kürzere Zeiten besitzt. Jedoch wurde dies nur bei der Funktionalität Hauptbereich verschieben gezeigt. Im nachfolgenden Testfall werden die beiden Prototypen auf weitere Funktionalitäten getestet. Aufgrund des Performanzverlustes durch die eingeschalteten Bereiche bei Prototyp1, wurden die weiteren Tests mit ausgeschalteten Bereiche durchgeführt.

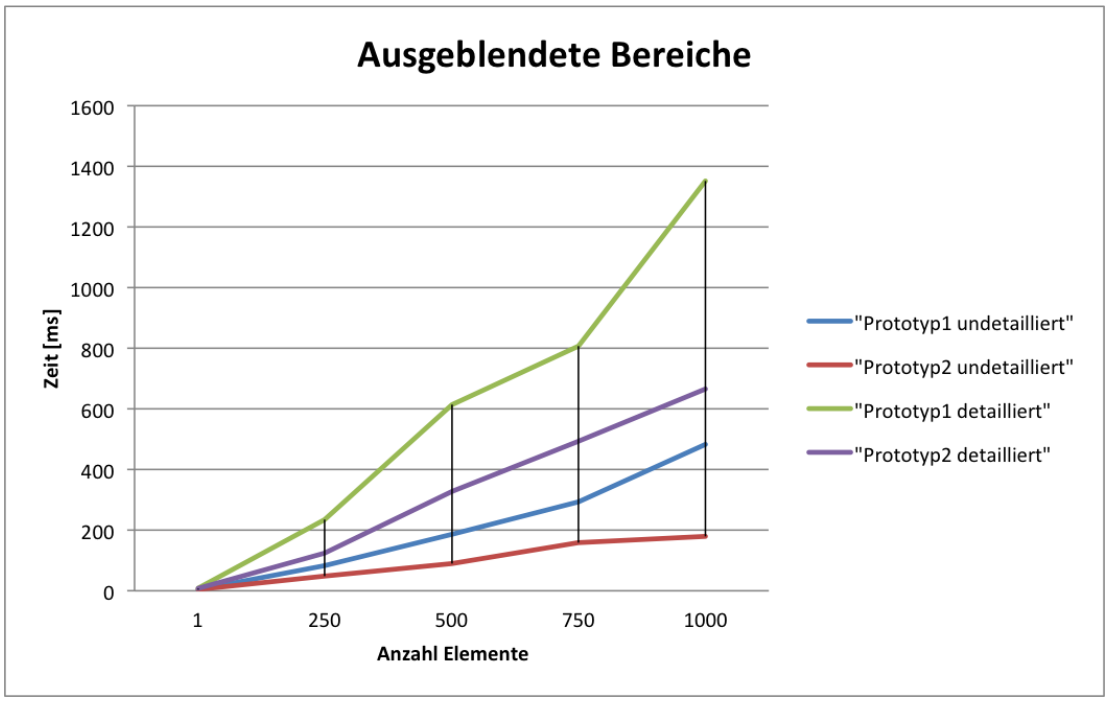

Abbildung 6.12: Prototyp 1 und Prototyp 2 im Vergleich bei ausgeschalteten Bereichen, sowohl Detailliert wie auch undetailliert.(zu Tabelle 6.1 und Tabelle 6.2).

6.3.3 Testfall 2

Im zweiten Testfall wird gezeigt, mit wie vielen Elementen das System umgehen kann. Zudem werden die Funktionalitäten bei einer detaillierten und undetaillierten Sicht verglichen.

Dazu wurden die Funktionalitäten Hauptbereich Größe ändern, Element erstellen, Element Position ändern und Element löschen betrachtet. Bei der Hauptbereichgröße Ändern Funktion wurde die Sicht verkleinert und vergrößert. Die Erstellung eines Elements wurde durch das Reinziehen eines Elements aus der Palette in das Hauptbereich vollzogen. Danach wurde das erstellte Element verschoben und dadurch auch seine Position geändert. Danach wurde die Zeit für das Löschen dieses Elements gemessen. Zudem wurden wie in Testfall1 die Tests jeweils von 1 bis 1000 Elemente in detaillierter und undetaillierter Sicht durchgeführt. Zusätzlich wurde noch unterschieden, ob nur ein Element oder alle Elemente in der Sicht angezeigt werden.

Bei dem Test mit nur einem Element in Sicht, liegen die Zeiten von beiden Prototypen unter 1 Sekunde. In den Tabellen 6.3 und 6.4 werden die Ergebnisse der Messung für die detaillierte und undetaillierte Sicht dokumentiert.

Funktion↓	Anzahl Elemente→ /Prototyp↓	1	250	500	750	1000
Hauptbereich größe ändern	Prototyp1	21	48	60	146	212
	Prototyp2	20	116	220	312	417
Element erstellen	Prototyp1	197	336	332	446	501
	Prototyp2	79	145	247	302	434
Element Position ändern	Prototyp1	128	166	234	234	294
	Prototyp2	54	143	264	454	641
Element löschen	Prototyp1	143	184	239	219	341
	Prototyp2	81	216	360	464	634

Tabelle 6.3: Zweiter Testfall in den nur 1 Element detailliert im Sichtfeld liegt. Die Werte sind in Millisekundenbereich angezeigt.

Funktion↓	Anzahl Elemente→ /Prototyp↓	1	250	500	750	1000
Hauptbereich größe ändern	Prototyp1	18	36	68	123	150
	Prototyp2	18	53	100	140	189
Element erstellen	Prototyp1	155	192	302	337	426
	Prototyp2	62	87	131	187	227
Element Position ändern	Prototyp1	129	152	200	210	221
	Prototyp2	74	92	123	158	194
Element löschen	Prototyp1	172	180	185	188	203
	Prototyp2	70	121	223	231	299

Tabelle 6.4: Zweiter Testfall in den nur 1 Element undetailliert im Sichtfeld liegt. Die Werte sind in Millisekundenbereich angezeigt.

Bei dem Test mit allen Elemente in Sicht vergrößern sich die Zeiten bei beiden Prototypen. Die Messungen haben gezeigt, dass der Prototyp1 mit einem Diagramm von 250 Elementen noch die Anforderungen erfüllen kann. Ab 500 Elemente sind die Werte nicht mehr akzeptabel. Die Tabelle 6.5 enthält die Messungen einer detaillierten Sicht. Die mit Rot markierten Zellen sind die Werte, größer als 1 Sekunde.

Bei der undetaillierten Sicht sind die Zeiten besser geworden, aber auch hier kann der Prototyp 1 nicht alle Diagramme bearbeiten. Ab 750 Elemente sind die Zeiten bei der Erstellung eines Elements über 1 Sekunde. Die Messdaten sind in der Tabelle 6.6 dokumentiert.

Funktion↓	**Anzahl Elemente→ /Prototyp↓**	**1**	**250**	**500**	**750**	**1000**
Hauptbereich größe ändern	**Prototyp1**	21	317	524	1005	1406
	Prototyp2	21	119	359	540	704
Element erstellen	**Prototyp1**	197	843	1597	2539	3288
	Prototyp2	80	206	379	517	662
Element Position ändern	**Prototyp1**	128	503	1004	1611	2594
	Prototyp2	55	323	538	901	1695
Element löschen	**Prototyp1**	143	524	989	1464	1784
	Prototyp2	82	138	602	856	1356

Tabelle 6.5: Zweiter Testfall in der alle Elemente detailliert im Sichtfeld liegt. Die Werte sind in Millisekundenbereich angezeigt.
ROT: sind die Zeiten über einer 1 Sekunde.

Funktion↓	**Anzahl Elemente→ /Prototyp↓**	**1**	**250**	**500**	**750**	**1000**
Hauptbereich größe ändern	**Prototyp1**	18	95	210	322	510
	Prototyp2	19	66	110	163	211
Element erstellen	**Prototyp1**	155	340	561	1012	1438
	Prototyp2	62	114	137	191	252
Element Position ändern	**Prototyp1**	129	184	318	421	695
	Prototyp2	74	113	183	255	280
Element löschen	**Prototyp1**	172	217	327	538	750
	Prototyp2	70	138	259	267	342

Tabelle 6.6: Zweiter Testfall in der alle Elemente undetailliert im Sichtfeld liegt. Die Werte sind in Millisekundenbereich angezeigt.
ROT: sind die Zeiten über einer 1 Sekunde.

Die Abbildung 6.13 stellt die Messungen aller Funktionen mit der detaillierten und undetaillierten Sicht bei Prototyp 1 dar. Aus dem Schaubild geht hervor, dass sich die undetaillierte Sicht nur auf die Performanz einziger Funktionen auswirkt.

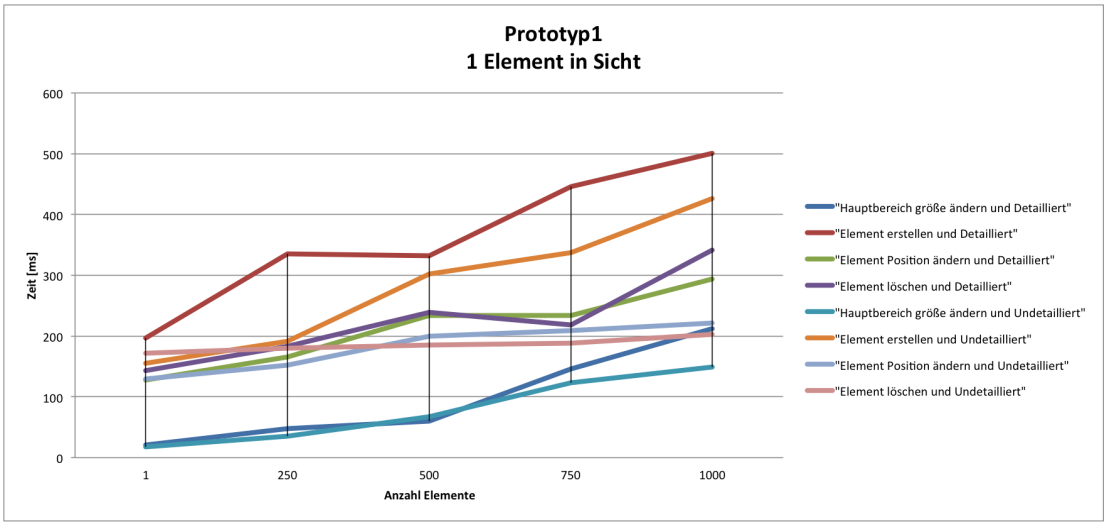

Abbildung 6.13: Zweiter Testfall für Prototyp 1 in der ein Element in detaillierter und undetaillierter Sicht liegt.

Die Abbildung 6.14 zeigt, wie stark die Detaillierung der Elemente auf die Performanz von Prototyp 2 wirkt. Je weniger detailliert die Sicht bei Prototyp2 ist, desto besser wird die Performanz für das ganze System.

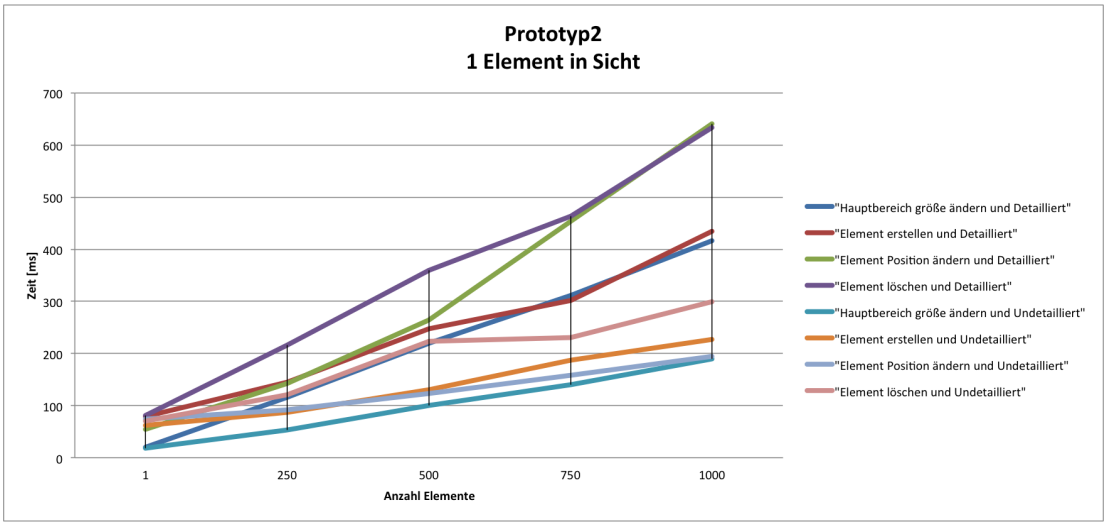

Abbildung 6.14: Zweiter Testfall für Prototyp 2 in der ein Element in detaillierter und undetaillierter Sicht liegt.

Die Abbildung 6.15 stellt den Vergleich von beiden Prototypen bei der detaillierten Sicht dar. Die Performanz von Prototyp 1 sinkt erheblich mit steigender Anzahl von Elementen in der Sicht. Beim Prototyp 2 ist die Anzahl von Elemente in der Sicht nicht bedeutsam.

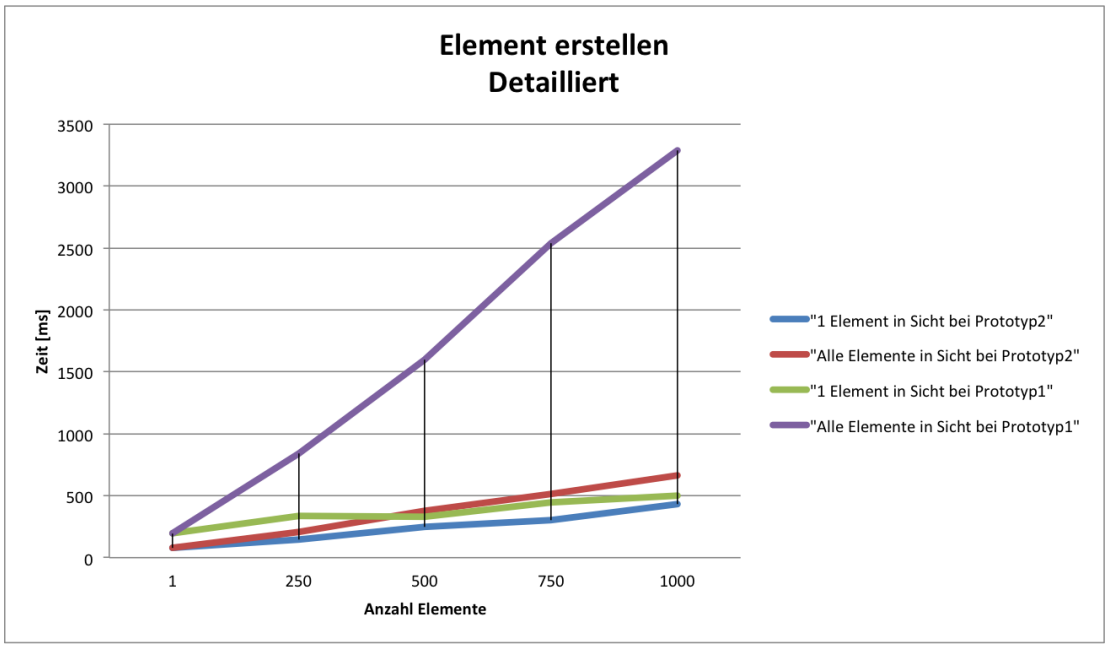

Abbildung 6.15: Zweiter Testfall stellt den Vergleich zwischen Prototyp 1 und 2 dar, bei einer detailierten Sicht für die Funktion Element erstellen

Die Abbildung 6.16 stellt den Vergleich von beiden Prototypen bei der undetaillierten Sicht dar. Im Gegensatz zur detaillierten Sicht ergibt die Messung von Prototyp 1 bessere Zeiten. Jedoch ist die Performanz von der Elementenanzahl in Sicht immer noch abhängig. Bei Prototyp 2 verbessern sich die Zeiten und es gibt kein Unterschied zwischen der Anzahl Elemente in Sicht.

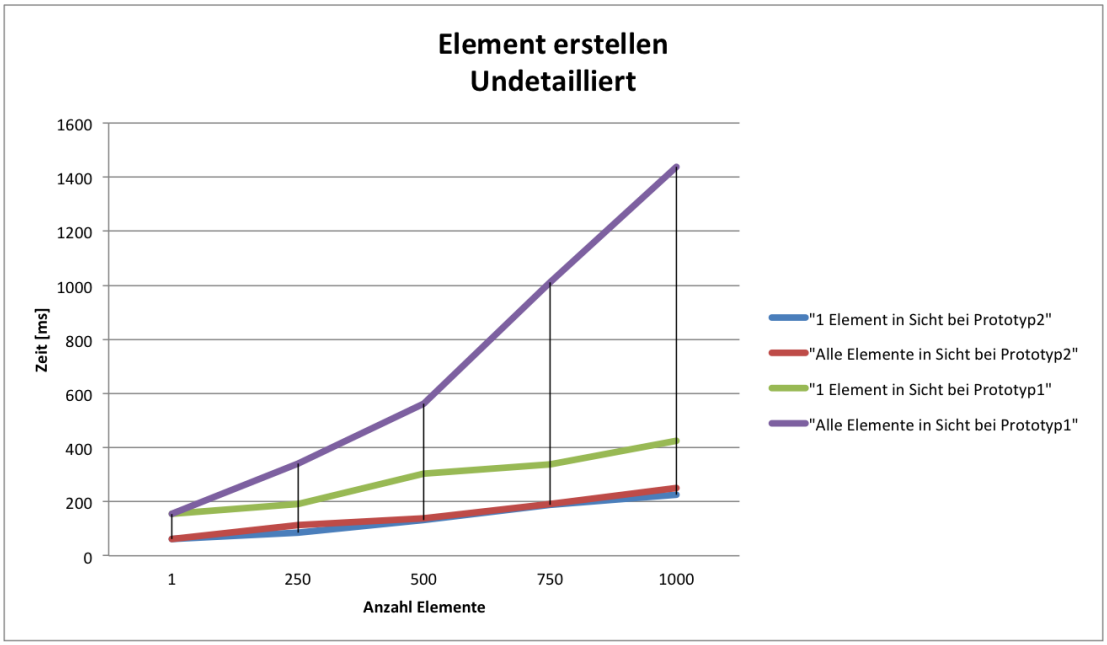

Abbildung 6.16: Zweiter Testfall zeigt den Vergleich zwischen Prototyp 1 und 2, bei einer undetailierten Sicht für die Funktion Element erstellen

6.3.4 Testfall 3

Letzter Testfall zeigt, wie mehrere Clients das System beeinflussen. In diesem Test wurde die Messungen jeweils für 1, 2, 15 und 30 Clients durchgeführt. Die Clients wurden durch weitere Browser Fenster erstellt. Bei beiden vorherigen Tests hat sich Prototyp 2 besser als Prototyp 1 gezeigt, deswegen wurde hier nur der Prototyp 2 getestet. Da der Prototyp 2 eine Diagramm mit maximal 750 Elemente bearbeiten kann, wird nur dieses getestet.

Die Tabelle 6.7 enthält die Messergebnisse für detaillierter Sicht. Alle Zeiten liegen unter 1 Sekunde.

Anzahl Clients→	**1**		**2**		**15**		**30**	
Anzahl Elemente in Sicht→ /Funktionen↓	**1**	**750**	**1**	**750**	**1**	**750**	**1**	**750**
Hauptbereich größe ändern	312	540	291	498	306	524	304	516
Element erstellen	302	517	279	477	280	473	271	476
Element Position ändern	454	901	352	777	350	821	361	835
Element löschen	464	856	431	887	442	773	439	773

Tabelle 6.7: Dritter Testfall stellt Anzahl Clients von 1 bis 30 für 750 Elemente in detaillierter Sicht dar.

Die Tabelle 6.8 enthält die Messergebnisse für die detaillierte Sicht. Auch hier liegen die Zeiten unter 1 Sekunde.

Anzahl Clients→	**1**		**2**		**15**		**30**	
Anzahl Elemente in Sicht→ /Funktionen↓	**1**	**750**	**1**	**750**	**1**	**750**	**1**	**750**
Hauptbereich größe ändern	140	163	127	151	127	153	129	154
Element erstellen	187	191	154	184	156	166	139	166
Element Position ändern	158	255	149	225	137	198	136	199
Element löschen	231	267	199	277	158	225	176	225

Tabelle 6.8: Dritter Testfall stellt Anzahl Clients von 1 bis 30 für 750 Elemente in undetaillierter Sicht dar.

In der Abbildung 6.17 sind alle Funktionen in detaillierter Sicht für 750 Elemente dargestellt. Die Zeiten ändern sich durch erhöhen der Anzahl Clients nicht. Die Funktionen Element Erstellen und Element löschen brauchen am meisten Zeit. Der Abstand zwischen Maximum und Minimum beträgt 630 ms.

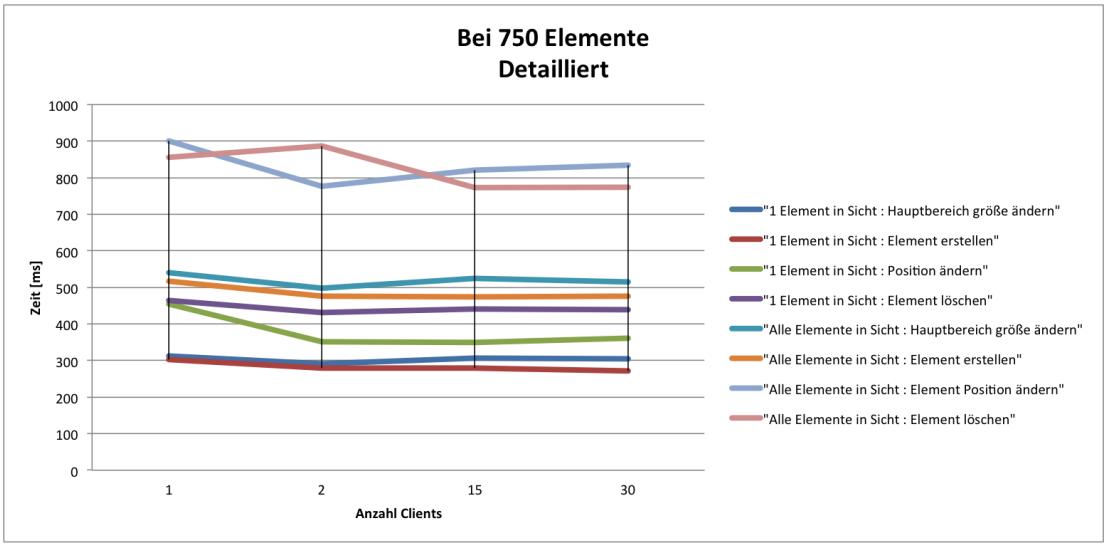

Abbildung 6.17: Dritter Testfall zeigt die Zeiten aller Funktionen in Abhängigkeit von der Anzahl an Clients in detailierter Sicht.

In der Abbildung 6.18 sind alle Funktionen in undetaillierter Sicht dargestellt. Die Zeiten von Funktionen liegen nah aneinander und haben keine Abhängigkeit zur Anzahl der Clients. Auch hier brauchen Funktionen Element Erstellen und Element löschen am meisten Zeit. Der Abstand zwischen Maximum und Minimum wird durch die undetaillierte Sicht um das 4fache verringert und beträgt 150 ms.

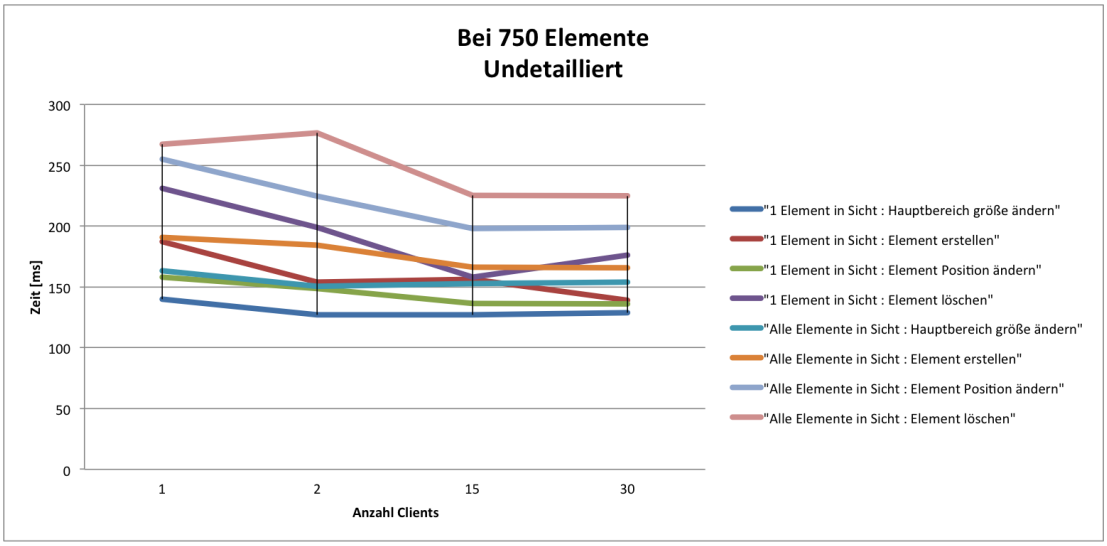

Abbildung 6.18: Dritter Testfall zeigt die Zeiten aller Funktionen in Abhängigkeit von der Anzahl an Clients in undetailierter Sicht.

6.4 Bewertung

Durch die Prototypentest bei der Evaluation wurde deutlich, dass sich Prototyp 2 gegenüber Prototyp 1 in Punkten Performanz besser abgeschnitten hat. Das bedeutet, dass das Rendern einer Komponente mehr Zeit in Anspruch nimmt, als die Überprüfung ob gerendert werden muss.

Bei der detaillierter Sicht ist die Performanz schlechter als bei undetaillierter Sicht, dafür ist die Qualität der Visualisierung der Elemente besser. Bei der Verkleinerung der Sicht wird der Name einer Komponente so klein, dass es an einem bestimmten Größe unlesbar ist. Eine detaillierte Darstellung des Elements ist dabei unnötig.

Die Navigation des Hauptbereiches als lokale Interaktion zu betrachten wirkt sich sowohl positiv, als auch negativ aus. Diese Architekurentscheidung bietet den Vorteil, dass bei der Navigation auf die Kommunikation zwischen Client und Server zu verzichten. Durch weniger Abfragen vom Client, wird die Auslastung am Server verringert. Ein Nachteil ist, dass bei der Programmierung redundante Berechnungen von Koordinaten entstehen. Bei der Navigation müssen diese neuen Werte an jede Komponente mitgeteilt werden, damit die Bearbeitung einer Komponente richtig funktionieren kann. Zum Beispiel muss bei der Erstellung einer Komponente die Berechnung der Anzeigeposition, die Verschiebung und Skalierung des Haupt-

bereichs berücksichtigt werden. Die Berechnung wiederholt sich auch bei Veränderung der Größe einer Komponente. Ein weiterer Nachteil ist die Verschlechterung der Code-Übersicht. Die neuen Werte der Navigation werden durch Eigenschaften an eine Komponente mitgeteilt. Dadurch vergrößert sich die Liste der Eigenschaften jeder Komponente, die von der Navigation abhängen, was sich zusätzlich auf die Lesbarkeit des Codes auswirkt. Einer der schwerwiegendsten Nachteile ist der Performanzverlust bei der Navigation. Bei jeder Interaktion mit der Navigation des Hauptbereichs werden die Komponente neu gerendert, da sich die Eigenschaften einer Komponente geändert haben. Je größer die Anzahl an Komponenten, die gerendert werden müssen, desto mehr Zeit wird für das Rendern bei einer Navigation gebraucht. Dies wird im zweiten Prototyp durch die Funktion *shouldComponentUpdate* verbessert, da hier eine Überprüfung stattfindet ob gerendert werden muss.

Funktionale Anforderungen	Kurzbeschreibung	Realisierung
FA1	Das Hauptbereich navigieren	X
FA2	Elemente und Beziehungen in einer Palette	X
FA3	Elemente durch Drag and Drop erstellen	X
FA4	Elemente/Beziehungen löschen	X
FA5	Elemente bearbeiten	X
FA6	Elemente gruppieren	X
FA7	Elemente in Beziehung setzen	X
FA8	Beziehungsverlauf verändern	X
FA9	Unterstützung von ArchiMate	X
FA10	Hauptbereich, Palette, Menu, Outline und Eigenschaftenbereich	X
FA11	Raster	X
FA12	Bereiche können an und ausgeblendet werden.	X
Nicht-funktionale Anforderungen	Kurzbeschreibung	Realisierung
NFA1	Anpassungsfähigkeit	X
NFA2	Installierbarkeit	X
NFA3	Modularität	X
NFA4	Analysierbarkeit	X
NFA5	Antwortzeitverhalten	X
NFA6	Wiederherstellbarkeit	X
NFA7	Bedienbarkeit	X

Tabelle 6.9: Umsetzung der Anforderungen

Die Tabelle 6.9 stellt alle Anforderungen dar, die in Kapitel 3.2 erarbeitet wurden, an. Mit dem Zusatz, welche realsiert wurden. Die funktionalen Anforderungen wurden alle erreicht. Dies zeigt der Systemtest in Kapitel 6.2. Die Prototypentests konnten erfolgreich im Chrome Browser durchgeführt werden. Dies erfüllt die nichtfunktionale Anforderung NFA1, dass die Webapplikation den Chrome Browser verwenden soll. Die nichtfunktionale Anforderung NFA2 wurde in Kapitel 6.1 eine Installationbeschreibung gezeigt, die diesen Ansprüchen genügt.

Die Verwendung des Architekturstils FLUX hat eine modulare Aufteilung der Webapplikation ermöglicht und somit die nichtfunktionale Anforderung NFA3 erfüllt. Die Anforderung NFA4 konnte erfolgreich erfüllt werden, da die Webapplikation komplett in der Programmiersprach JavaScript implementiert werden konnte. Dabei wurden ausschließlich JavaScript Bibliothek und Frameworks verwendet, wie *reactJS*, *socketIO*, *mongoDB*, *konvaJS*, *expressJS*, sowie *mocha* beim Testen. Die Webapplikation kann die Anforderung NFA5 bis eine Anzahl von 750 Element erfolgreich ausführen. Beim Neustart der Webapplikation kann der letzte Zustand des Diagramms wiederhergestellt werden. Dadurch wird die Anforderung NFA6 erfüllt. Die Tabelle 6.10 zeigt, dass die Webapplikation sich nach den vorhanden Werkzeugen auf dem Markt richtet. Somit ist die Bedienbarkeit (NFA7) gegeben.

Werkzeuge → /Funktionalität ↓	Webapplikation										Desktopapplikation									
	Resultat					Simple Archimate					Archi					Visual Paradigm				
	H	P	M	O	E	H	P	M	O	E	H	P	M	O	E	H	P	M	O	E
Bewegung der Sicht des Hauptbereichs	X	-	X	X	-	X	-	-	X	-	X	-	-	X	-	X	-	-	X	-
Vergrößern/Verkleinern der Sicht	-	-	X	X	-	-	-	X	X	-	X	-	X	-	-	X	-	X	X	-
Darstellung der Archimate Elemente	X	X	-	X	-	X	X	-	X	-	X	X	-	X	-	X	X	-	X	-
Darstellung der Archimate Beziehungen	X	X	-	X	-	X	-	-	X	-	X	X	-	X	-	X	X	-	X	-
Änderung Beziehungsverhalten	X	-	-	-	-	-	-	-	-	-	X	-	-	-	-	X	-	-	-	-
Erstellung der Elemente durch Drag'n'Drop	X	X	-	-	-	X	X	-	-	-	X	X	-	-	-	X	X	-	-	-
Erstellung der Beziehung	X	X	-	-	-	X	-	-	-	-	X	X	-	-	-	X	X	-	-	-
Löschen der Elemente	-	-	X	-	-	-	-	X	-	-	X	-	X	-	-	X	-	-	-	-
Gruppierung der Elemente	X	-	-	-	-	X	-	-	-	-	X	-	-	-	-	X	-	-	-	-
Bewegung der Elemente	X	-	-	-	-	X	-	-	-	-	X	-	-	-	-	X	-	-	-	-
Änderung der Größe	X	-	-	-	X	X	-	-	-	-	X	-	-	-	-	X	-	-	-	-
Änderung des Styles	-	-	-	-	X	-	-	-	-	X	-	-	-	-	X	-	-	-	-	X
Änderunger der Elementenname	X	-	-	-	X	X	-	-	-	-	X	-	-	-	X	-	-	-	-	X
Darstellung der Raster	X	-	-	-	-	-	-	-	-	-	X	-	X	-	-	X	-	X	-	-
Validierung	X	-	-	-	-	X	-	-	-	-	X	-	-	-	-	X	-	-	-	-
Ein-/Ausblenden	-	X	-	X	X	-	X	-	X	X	X	X	-	X	X	X	X	-	X	X
Kollaboration	X	-	-	-	X	-	-	-	-	-	-	-	-	-	-	X	-	-	-	-

Tabelle 6.10: Endergebnis

*(**H**: Hauptbereich; **P**: Palette; **M**: Menu; **O**: Outline; **E**: Eigenschaftenbereich; **X**: Vorhanden; -: Nicht Vorhanden; **FARBEN**:{ ROT : Es gibt genau 1 Werkzeug, der diese Funktionalität hat; GELB : Es gibt genau 2 Werkzeuge, die diese Funktionalitäten im gleichen Bereich haben; GRÜN : Es gibt genau 3 Werkzeuge, die diese Funktionalitäten im gleichen Bereich haben})*

7 Fazit und Ausblick

Abschließend werden in diesem Kapitel die erreichten Ergebnisse zusammengefasst, sowie ein Ausblick über mögliche Weiterentwicklungen und Verbesserungen des Systems der Arbeit gegeben.

7.1 Zusammenfassung

In dieser Arbeit wurde eine Webapplikation entworfen und realisiert, die es ermöglicht, kollaborativ EA Archimate Diagramm zu erstellen. Dabei können Diagramme bis zu 750 Elementen im Mehrbenutzerbetrieb bearbeitet werden. Um dies zu erreichen wurde zuerst eine Marktanalyse von existieren Tools, die die Archimate Modellierungssprache verwenden, durchgeführt. Anhand der Ergebnisse konnte eine Anforderungsanalyse vollzogen werden, die die funktionalen und nichtfunktionalen Anforderungen an das System festlegten.

Im nächsten Schritt wurde das System anhand eines Mockups entworfen. Daraufhin wurden Komponentendiagramme erstellt, um die Architektur festzulegen. Diese basierte auf den Architekturstil FLUX mit der Kombination von den Client-Server Modell. Für die Persistierung der Daten wurde die NoSQL Datenbank MongoDB verwendet.

Entsprechend der Spezifikation des Entwurfs wurde das System dann realisiert. Es wurde hierbei mit der Visualisierung der Webapplikation begonnen. Diese zeigte, wie Konva in der Anwendung integriert wurde. Als Nächstes wurde beschrieben, wie sich Client und Server miteinander mittels WebSockets kommunizieren. Nach dem Client-Server Modell wurden die Stellen beschrieben, an den die FLUX Architektur in den React Komponenten eingesetzt wurde. Daraufhin wurde der Lösungsansatz zur Verifizierung der Archimate Elemente beschrieben. Zum Schluss wurden dann zwei Prototypen vorgestellt.

Durch Entwicklertests wurden die Qualitäten der Anwendung über den ganzen Entwicklungszyklen hinweg sichergestellt. Die Evaluierung beginnt mit einer Einführung der Installation der Software. Darauf Folgend wurde der Systemtest durchgeführt. Am Schluss wurden die Prototypen getestet und die Testergebnisse analysiert.

Zusammenfassend ist zu sagen, dass die Entwicklung einer FullStack JavaScript Vorteile mit sich bringt. Einer der Vorteile ist die Nutzung einer Programmiersprache über alle Ebenen

ein besseres Codeverständnis mit sich bringt. Zudem kann der Code an verschiedenen Stellen wiederverwendet werden. So können dieselben Datenobjekte sowohl auf server- als auch auf clientseite benutzt werden, ohne diese umzuwandeln. Das Schreiben von Tests, ist dazu angenehmer. Ein weiter Vorteil ist die Nutzung von NodeJS als Plattform des Servers. Dieser bietet einen Package Manager mit sich, mit dem es Möglich ist über 70.000 NodeJS Module in sein Projekt einzubinden. Dadurch wird auch die Installierbarkeit erheblich vereinfacht, da alle Abhängigkeiten zusammengefasst aufgelistet werden. Mit Hilfe von Socket.IO kann durch wenig Aufwand eine Asynchrone Client-Server Kommunikation aufgebaut werden. Bei der Recherche von ReactJS wurde festgestellt, dass die Bibliothek recht neu ist und es wenig Literatur dazu zufinden ist. Dafür bietet Facebook auf ihrer Github Seite eine übersichtliche Dokumentation dazu an. Der Architekturstil FLUX bietet einen Unidirektionalen Datenfluss an. Dadurch wird die Übersicht der Daten behalten. Die Kombination von ReactJS mit der FLUX Architektur erleichtert die Kommunikation der React Komponenten. So muss eine React Komponente seinen Zustand nicht über die Eltern oder Kinderknoten senden, sondern schickt einfach eine Action an den passenden Store.

7.2 Ausblick

Die entwickelte Webapplikation ist eine Basis für viele Erweiterungsmöglichkeiten. Jedoch konnten innerhalb dieser Arbeit konnten nicht alle Aspekte berücksichtigt werden.

Obwohl die Zeiten für das Vergrößern, Verkleinern und Verschieben des Hauptbereichs noch in Rahmen der Anforderungen liegen, gibt es ein Spielraum für die Performanzverbesserung. Zudem geht Zeit verloren, wenn das Raster (Grid) beim Navigieren gerendert wird. Ein alternativer Lösungsansatz wäre die horizonatalen und vertikalen Linien des Rasters nur einmal bei der Erstellung rendern zu lassen. Bei der Navigieren soll nur beim Layer mit dem Raster die Props geändert werden.

Als weitere Entwicklungsmöglichkeit ist, die Arbeit an den unterschiedlichen Projekte bereitzustellen. Dazu werden bei Socket.IO Rooms angeboten, die eine Sortierung von Nachrichten für jeweilige Projekte erleichtern soll. Clients können die Rooms betreten und verlassen. Die Nachrichten werden nur innerhalb eines Rooms verteilt. Jedes Projekt kann ein eigenen Room besitzen, damit die Nachrichten von verschiedenen Projekten nicht vermischen werden.

Für die Kollaboration können weitere Module integriert werden, wie zum Beispiel: Kommentare zu einer Komponente einfügen, um das Verständnis des Diagramms zu verbessern. Dazu muss das Datenbankschema erweitert werden.

Ebenfalls können die ViewPoints von ArchiMate implementiert werden, um mehr Zeit beim Rendern zu sparen. Dazu werden nur die Elemente gerendert, die zu einem ViewPoint gehören.

Auch die Übertragung auf mobile Geräte wäre möglich. Dafür bietet Konva Events an, die für die Views erweitert werden können.

Das Löschen und Position brauchen im Vergleich zu anderen Funktionen viel Zeit. Für die Performanzverbesserung werden andere Algorithmen und Datenstrukturen benötigt.

Anschließlich können weitere Diagrammtypen integriert werden.

Durch eine Verfeinerung der Schnittstellen können Konflikte vermindert werden. Zum Beispiel wird bei der Bearbeitung eines Elements wird die Schnittstelle *changeElement* aufgerufen, dabei wird nicht betrachtet welche Art der Änderung vollzogen wird. Bei einer Verfeinerung der Schnittstellen kann eine Element von mehreren Anwendern gleichzeitig bearbeitet werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

[arc16] Archimatetool. http://archimatetool.com/, Online-Abfrage: 24. März 2016 2016.

[Ber12] Scott A. Bernard. *An Introduction to Enterprise Architecture* -. AuthorHouse, Bloomington, 2012.

[Bö08] A. ; König K. ; Krcmar Helmut ; Leimeister Stefanie ; Lohmann Jörg ; Böhmann, Tilo ; Häge. Geschäftsorientiertes architekturmanagement - wertbeitrag und implementierungsstrategien in deutschen unternehmen. Ibm-whitepaper, TUM, Technische Universität Muenchen, 2008.

[Ebe08] Christof Ebert. *Systematisches Requirements Engineering und Management - Anforderungen ermitteln, spezifizieren, analysieren und verwalten.* Dpunkt.Verlag GmbH, Heidelberg, 2. aktualisierte und erweiterte. aufl. edition, 2008.

[fac] facebook. Docs. http://facebook.github.io/react/. Online-Abfrage: 2. April 2016.

[fac16] facebook. https://facebook.github.io/react/docs/reconciliation.html, Online-Abfrage: 11. April 2016 2016.

[Fed15] Artemij Fedosejev. *React.js Essentials* -. Packt Publishing Ltd, Birmingham, 1. aufl. edition, 2015.

[Feu07] Sven Feurer. Enterprise architecture – an overview. Technical report, SAP Deutschland AG & Co. KG, Karlsruhe, 2007.

[Goo] Google. material-design. https://www.google.com/design/spec/material-design/introduction.html. Online-Abfrage: 7. April 2016.

[Gro] The Open Group. http://help.innovator.de/12.0/de_de/Innovator/Content/Ref.MetaM.ArchiMate/ArchiMate.htm. Online-Abfrage: 14. April 2016.

[Gro13a] The Open Group. Archimate® 2.1 specification. http://pubs.opengroup.org/architecture/archimate2-doc/, 2013. Online-Abfrage: 16. März 2016.

[Gro13b] The Open Group. Archimate® 2.1 verification. http://pubs.opengroup.org/architecture/archimate2-doc/appendixB.html#_Toc371945300, 2013. Online-Abfrage: 16. März 2016.

[HGG13] Inge Hanschke, Gunnar Giesinger, and Daniel Goetze. *Business Analyse – einfach und effektiv - Geschäftsanforderungen verstehen und in IT-Lösungen umsetzen.* Carl Hanser Verlag GmbH Co KG, M, 1. aufl. edition, 2013.

[iso11] Iso/iec 25010:2011(en). https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en, March 2011.

[Kon16a] React Konva. React konva. https://github.com/lavrton/react-konva, Online-Abfrage: 6. April 2016 2016.

[Kon16b] KonvaJS. Konva. http://konvajs.github.io/docs/index.html, Online-Abfrage: 6. April 2016 2016.

[Mat11] Dirk Matthes. *Enterprise Architecture Frameworks Kompendium - Über 50 Rahmenwerke für das IT-Management.* Springer-Verlag, Berlin Heidelberg New York, 2011. aufl. edition, 2011.

[mu] material ui. material-ui. http://www.material-ui.com/#/. Online-Abfrage: 7. April 2016.

[nod] nodeJS. node js. https://nodejs.org/en/. Online-Abfrage: 11. April 2016.

[par16] Paradigm userlist. https://www.visual-paradigm.com/aboutus/userlist.jsp, Online-Abfrage: 24. März 2016 2016.

[PLG15] hoai viet Nguyen Peter Leo Gorski, Luigi Lo Iacono. *WebSockets Moderne HTML5-Echtzeit anwendungen entwickeln.* Carl Hanser Verlag GmbH Co KG, München, 2015.

[Poh08] Klaus Pohl. *Requirements Engineering - Grundlagen, Prinzipien, Techniken.* dpunkt.verlag, Heidelberg, (2. aufl.) edition, 2008.

[Sch09] Bettina Schwarzer. *Enterprise Architecture Management - Verstehen - Planen - Umsetzen.* BoD – Books on Demand, Norderstedt, 1. aufl. edition, 2009.

[Sch14] Klaus-Peter Schoeneberg. *Komplexitätsmanagement in Unternehmen - Herausforderungen im Umgang mit Dynamik, Unsicherheit und Komplexität meistern.* Springer-Verlag, Berlin Heidelberg New York, 2014. aufl. edition, 2014.

[Sim16] Simplyarchimate. https://www.simplyarchimate.com/, Online-Abfrage: 23. März 2016 2016.

[Sta15] Gernot Starke. *Effektive Softwarearchitekturen - Ein praktischer Leitfaden.* Carl Hanser Verlag GmbH Co KG, M, 2015.

[zip] zippyui. zippyui/react-color-picker. http://jslog.com/react-color-picker/. Online-Abfrage: 7. April 2016.

Anhang

A. Core Concepts and Relationships

B. ArchiMate Spezifikation

Relationships

1. (a)ccess
2. (c)omposition
3. (f)low
4. a(g)gregation
5. ass(i)gnment
6. i(n)fluence
7. ass(o)ciation
8. (r)ealization
9. (s)pecialization
10. (t)riggering
11. (u)sed by

From ↓ / To →	Business Actor	Business Role	Business Collaboration	Location	Business Interface	Business Process	Business Function	Business Interaction	Business Event	Business Service	Business Object	Representation	Product	Contract	Meaning	Value
Business Actor	cfgostu	fiotu	fiotu	o	cfiotu	fiotu	fiotu	fiotu	ot	ioru	ao	o	o	ao	o	o
Business Role	fotu	cfgostu	cfgostu	o	cfgiotu	fiotu	fiotu	fiotu	ot	ioru	ao	o	o	ao	o	o
Business Collaboration	fgotu	cfgostu	cfgostu	o	cfgotu	fiotu	fiotu	fiotu	ot	ioru	ao	o	o	ao	o	o
Location	fiotu	fiotu	fiotu	cfgostu	fiotu	io	fiotu	io	iot	io	io	io	o	ao	o	o
Business Interface	fotu	fotu	fotu	o	cfgostu	ou	ou	ou	ot	iou	ao	o	o	ao	o	o
Business Process	fotu	fotu	fotu	o	fotu	cfgostu	cfgostu	cfgostu	ot	fortu	ao	o	o	ao	o	o
Business Function	fotu	fotu	fotu	o	fotu	cfgostu	cfgostu	cfgostu	ot	fortu	ao	o	o	ao	o	o
Business Interaction	fotu	fotu	fotu	o	fotu	cfgostu	cfgostu	cfgostu	ot	fortu	ao	o	o	ao	o	o
Business Event	ot	ot	ot	ot	ot	ot	ot	ot	cgost	o	ao	o	o	ao	o	o
Business Service	ou	ou	ou	o	ou	fotu	fotu	fotu	o	cfgostu	ao	o	o	ao	o	o
Business Object	o	o	o	o	o	o	o	o	o	o	cgos	o	o	cgos	o	o
Representation	o	o	o	o	o	o	o	o	o	o	or	cgos	o	or	o	o
Product	ou	ou	ou	o	ou	ou	ou	ou	o	gou	ao	o	cgos	ago	o	o
Contract	o	o	o	o	o	o	o	o	o	o	cgos	o	o	cgos	o	o
Meaning	o	o	o	o	o	o	o	o	o	o	o	o	o	o	cgos	o
Value	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	cgos
Application Component	fotu	fotu	fotu	o	fotu	fiotu	fiotu	fiotu	ot	ioru	ao	o	o	ao	o	o
Application Collaboration	fotu	fotu	fotu	o	fotu	fiotu	fiotu	fiotu	ot	ioru	ao	o	o	ao	o	o
Application Interface	ou	ou	ou	o	fotu	fotu	fotu	ou	o	iou	ao	o	o	ao	o	o
Application Function	ou	ou	ou	o	fotu	fotu	ou	ou	o	ou	ao	o	o	ao	o	o
Application Interaction	ou	ou	ou	o	fotu	fotu	ou	ou	o	ou	ao	o	o	ao	o	o
Application Service	ou	ou	ou	o	ou	ou	ou	ou	o	ou	ao	o	o	ao	o	o
Data Object	o	o	o	o	o	o	o	o	o	o	or	o	o	or	o	o
Node	ou	ou	ou	o	ou	oru	oru	oru	o	oru	aoru	o	o	aoru	o	o
Device	ou	ou	ou	o	ou	oru	oru	oru	o	oru	aoru	o	o	aoru	o	o
System Software	ou	ou	ou	o	ou	oru	oru	oru	o	oru	aoru	o	o	aoru	o	o
Infrastructure Interface	aou	aou	aou	o	aou	aou	aou	aou	o	aou	aou	o	o	aou	o	o
Network	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Communication Path	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o
Infrastructure Function	ou	ou	ou	o	ou	ou	ou	ou	o	ou	aou	o	o	aou	o	o
Infrastructure Service	ou	ou	ou	o	ou	ou	ou	ou	o	ou	aou	o	o	aou	o	o
Artifact	ou	ou	ou	o	ou	oru	oru	oru	o	oru	aor	o	o	aor	o	o
Junction	ft	ft	ft	ft	ft	ft	ft	ft	t	ft						

Tabelle 1: Core Concepts Teil 1 [Gro13b]

From ↓ / To →	Application Component	Application Collaboration	Application Interface	Application Function	Application Interaction	Application Service	Data Object	Node	Device	System Software	Infrastructure Interface	Network	Communication Path	Infrastructure Function	Infrastructure Service	Artifact	Junction
Business Actor	fot	fot	fot	fot	fot	o	o	o	o	o	o	o	o	o	o	o	ft
Business Role	fot	fot	fot	fot	fot	o	o	o	o	o	o	o	o	o	o	o	ft
Business Collaboration	fot	fot	o	fot	fot	o	o	o	o	o	o	o	o	o	o	o	ft
Location	io	io	io	io	io	io	io	io	io	io	io	io	io	io	io	io	ft
Business Interface	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	ft
Business Process	fot	fot	o	o	o	o	o	o	o	o	o	o	o	o	o	o	ft
Business Function	fot	fot	o	o	o	o	o	o	o	o	o	o	o	o	o	o	ft
Business Interaction	fot	fot	o	o	o	o	o	o	o	o	o	o	o	o	o	o	ft
Business Event	ot	ot	o	o	o	o	o	o	o	o	o	o	o	o	o	o	t
Business Service	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	ft
Business Object	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	
Representation	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	
Product	ou	ou	ou	ou	o	gou	ao	ou	ou	ou	ou	o	o	ou	gou	ao	
Contract	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	
Meaning	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	
Value	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	o	
Application Component	cfgostu	cfgostu	cfgotu	iou	iou	ioru	ao	o	o	o	o	o	o	o	o	o	ft
Application Collaboration	cfgostu	cfgostu	cfgotu	iou	iou	ioru	ao	o	o	o	o	o	o	o	o	o	ft
Application Interface	fotu	fotu	cfgostu	ou	ou	iou	ao	o	o	o	o	o	o	o	o	o	ft
Application Function	ou	ou	ou	cfgostu	fot	fortu	ao	o	o	o	o	o	o	o	o	o	ft
Application Interaction	ou	ou	ou	fotu	cfgost	oru	ao	o	o	o	o	o	o	o	o	o	ft
Application Service	ou	ou	ou	fortu	ou	cfgostu	ao	o	o	o	o	o	o	o	o	o	ft
Data Object	o	o	o	o	o	o	cgos	o	o	o	o	o	o	o	o	o	
Node	aoru	aoru	aoru	aoru	aoru	aoru	aoru	cfgostu	cfgostu	cfgostu	cfgotu	o	o	iou	ioru	aiou	ft
Device	aoru	aoru	aoru	aoru	aoru	aoru	aoru	cfgostu	cfgostu	cfgiostu	cfgotu	o	o	iou	ioru	aiou	ft
System Software	aoru	aoru	aoru	aoru	aoru	aoru	aoru	cfgostu	cfgostu	cfgostu	cfgotu	o	o	iou	ioru	aiou	ft
Infrastructure Interface	aou	aou	aou	aou	aou	aou	aou	fotu	fotu	fotu	cfgostu	o	o	ou	iou	aou	ft
Network	o	o	o	o	o	o	o	o	o	o	o	cgos	or		o	o	
Communication Path	o	o	o	o	o	o	o	o	o	o	o	o	cgos	o	o	o	
Infrastructure Function	aou	aou	aou	aou	aou	auo	aou	ou	ou	ou	ou	o	o	cfgostu	oru	aou	ft
Infrastructure Service	ou	ou	ou	ou	ou	ou	aou	ou	ou	ou	ou	o	o	ou	cfgostu	aou	ft
Artifact	oru	oru	oru	oru	oru	oru	aor	o	o	or	or	o	o	or	or	cgors	
Junction	ft	ft	ft	ft	ft	ft		ft	ft	ft	ft			ft	ft		ft

Tabelle 2: Core Concepts Teil 2 [Gro13b]

From ↓ / To →	Core Element[b]	Business Actor	Business Role	Location	Value	Stakeholder	Driver	Assessment	Goal	Requirement	Principle	Constraint	Work Package	Deliverable	Plateau	Gap
Core Element[a]																
Business Actor	go															
Business Role	go															
Location	go															
Value	go															
Stakeholder	go															
Driver	go															
Assessment	go															
Goal	go															
Requirement	go															
Principle	go															
Constraint	go															
Work Package	go															
Deliverable	go															
Plateau	go															
Gap	go															

Tabelle 3: Core Concepts Teil 3 [Gro13b]

[a] Except Value and Meaning.
[b] Except Value and Meaning.

C. CD

Inhalte der beigefügten CD

- Bachelorarbeit als PDF
- Sourcecode

Hiermit versichern wir, dass wir die vorliegende Arbeit im Sinne der Prüfungsordung ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt haben.

Kapitelnr	1.1.	1.2.	1.3.
Timotin		x	
Sawadski	x		x

Kapitelnr	2.1.	2.1.1.	2.2.	2.2.1.	2.2.2.	2.2.2.1.	2.2.2.2.	2.2.2.3.	2.2.3.
Timotin	x		x		x		x		x
Sawadski		x		x		x		x	

Kapitelnr	3.1.	3.1.1.	3.1.2.	3.1.3.	3.1.4.	3.2.	3.2.1.	3.2.2.	3.3.	3.3.1.	3.3.2.	3.3.3.	3.3.4.	3.3.5.	3.3.6.	3.3.7.
Timotin		x		x		x		x		x		x		x		x
Sawadski	x		x		x		x		x		x		x		x	

Kapitelnr	4	4.1.	4.1.1.	4.1.2.	4.1.3.	4.1.4.	4.1.5.	4.2.1.	4.2.2.	4.2.3.	4.2.4.	4.2.4.1.	4.2.4.2.	4.2.4.3.	4.2.4.4.	4.2.5.
Timotin		x		x		x		x		x		x		x		x
Sawadski	x		x		x		x		x		x		x		x	

Kapitelnr	5.1.1.	5.1.2.	5.1.3.	5.1.4.	5.1.6.	5.1.7.	5.1.8.	5.1.9.	5.1.10.	5.1.11.	5.1.12.	5.2.	5.2.1.	5.2.2.	5.3.	5.3.1.
Timotin		x		x		x		x		x		x		x		x
Sawadski	x		x		x		x		x		x		x		x	

Kapitelnr	5.3.2.	5.3.3.	5.3.4.	5.3.5.	5.4.	5.5.	5.6.	5.6.1.	5.6.2.
Timotin		x		x		x		x	
Sawadski	x		x		x		x		x

Kapitelnr	6.1.	6.2.	6.2.1.	6.2.2.	6.2.3.	6.2.4.	6.2.5.	6.3.	6.3.1.1.	6.3.1.2.	6.3.2.	6.3.3.	6.3.4.	6.4.
Timotin	x		x		x		x		x		x		x	
Sawadski		x		x		x		x		x		x		x

Kapitelnr	7.1.	7.2.
Timotin	x	
Sawadski		x

Hamburg, 15. April 2016 Alexander Sawadski

Hamburg, 15. April 2016 Wlad Timotin