Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 346

Bausteinbasierte Entwicklung von Engineering Web Apps

Denis Lehmann

Studiengang:	Informatik
Prüfer/in:	Prof. Dr.-Ing. habil. Bernhard Mitschang
Betreuer/in:	Dipl.-Inf. Eva Hoos
Beginn am:	8. Juni 2016
Beendet am:	8. Dezember 2016
CR-Nummer:	C.2.4, D.2.2, D.4.9

Kurzfassung

Produkte aus dem Engineering–Bereich werden immer komplexer. In jeder Phase des Produktlebenszyklus, angefangen bei der Entwicklung über Konstruktion und Montage, entstehen Daten, die verwaltet werden müssen. Während sich die Anwendungen der einzelnen Bereiche unterscheiden, werden häufig ähnliche oder gleiche Daten in unterschiedlicher Umgebung und auf unterschiedlichen Endgeräten benötigt. Das Ziel dieser Bachelorarbeit ist es ein Konzept zu entwickeln, das es erlaubt verschiedene Daten aus dem Engineering–Bereich auf modulare Art und Weise auf beliebigen Endgeräten darzustellen. Dies erhöht die Wiederverwendbarkeit und senkt die Entwicklungskosten. In der Arbeit werden bereits existierenden Oberflächenkonzepte verglichen und Anforderungen an verschiedene Geschäftsfelder aus dem Engineering–Bereich ausgearbeitet. Auf dieser Basis wird ein Konzept vorgestellt, das es erlaubt sich eigene Oberflächen aus Bausteinen zusammenzusetzen. Dieses wurde mit einer Prototypimplementierung auf Basis von modernen Web–Technologien realisiert. Die Evaluierung erfolgt durch diverse Use–Cases.

Inhaltsverzeichnis

1 Einleitung

Durch immer komplexer werdende Produkte fallen im Engineering–Bereich mehr Daten an. Um diese zu verwalten benutzt man sogenannte Produktdatenmanagementsysteme. Diese erlauben es alle, während des Produktlebenszyklus entstehende, Daten zu verwalten. Die Anforderungen an solche Anwendungen unterscheiden sich je nach Geschäftsgebiet, sind jedoch häufig ähnlich oder überschneiden sich. Zum Beispiel können bei der Teilrecherche die benötigten Teilinformationen variieren, wohingegen in der Konstruktion eher Modelldaten vonnöten sind. Zusätzlich werden in verschiedenen Geschäftsgebieten des Engineering-Bereichs unterschiedliche Endgeräte verwendet, was zusätzliche Anforderungen an die Anwendungen stellt.

Diese Bachelorarbeit stellt ein Konzept vor, welches es erlaubt Daten aus dem Engineering–Bereich auf modulare Art und Weise darzustellen. Durch ein bausteinbasiertes Prinzip kann sich der Nutzer eigene Oberflächen zusammenstellen. Bausteine sind einzelne kleine Anwendungen, die auf unterschiedliche Darstellung von Informationen ausgelegt sind und einen bestimmten taskorientierten Funktionsumfang bieten. Dadurch kann sich der Nutzer, je nach Anwendungsgebiet, genau die Informationen anzeigen lassen, die er benötigt.

Moderne Web–Technologien erlauben es, Informationen plattformunabhängig zu präsentieren. Dies ist im Hinblick auf verschiedene Geschäftsfelder und immer mobiler werdende Endgeräte praktisch. Durch die Plattformunabhängigkeit müssen aber auch zusätzliche Aspekte wie Bildschirmgrößen und Eingabemethoden betrachtet werden.

Es werden verschiedene Anforderungen ausgearbeitet, welche für eine Anwendung im Engineering-Bereich wichtig sind. Weiterhin werden diverse Oberflächenkonzepte aus bereits existierenden Anwendungen verglichen und verwandte Arbeiten vorgestellt.

Im Zuge dieser Bachelorarbeit wurde ein Prototyp entwickelt, der dieses Konzept implementiert. Dieser wurde mit aktuellen Web–Technologien wie Angular 2 und Node.js konstruiert.

Gliederung

Die Arbeit ist in folgender Weise gegliedert: Kapitel 2 beschreibt die Grundlagen dieser Arbeit. Es werden zentrale Begriffe wie Produktdatenmanagement erläutert, moderne Web–Technologien erklärt und existierende Oberflächenkonzepte präsentiert. In Kapitel 3 werden verwandte Arbeiten und Konzepte vorgestellt. Hier werden ähnliche existierende Anwendungen und Oberflächenkonzepte beschrieben und verglichen. In Kapitel 4 werden die Anforderungen an diverse Geschäftsfelder aus dem Engineering–Bereich ausgearbeitet. Zusätzlich werden die Anforderungen an verschiedene Endgeräte hervorgehoben. Kapitel 5 erklärt das in dieser Bachelorarbeit erarbeitete Konzept. Hierbei wird besonders auf die Gestaltung der Benutzeroberfläche und die Darstellung der Produktdaten eingegangen. Die Implementierung des Prototyps unter Verwendung von Web–Technologien wird in Kapitel 6 zusammengefasst. Kapitel 7 evaluiert das Konzept indem auf die Anforderungen eingegangen wird und beschreibt diverse Use–Cases. Die Ergebnisse dieser Arbeit werden in Kapitel Kapitel 8 zusammengefasst.

2 Grundlagen

In diesem Kapitel werden grundlegende Technologien und Systeme vorgestellt, die das Grundwissen für die folgenden Teile der Arbeit bilden.

2.1 Produktdatenmanagement

Heutzutage wird der Produktentwicklungsprozess in der Industrie immer komplexer und teurer. Angefangen mit dem Design und der Konstruktion eines Produktes, über die Herstellung bis hin zur Wartung, müssen alle Prozesse verwaltet und die zugehörigen Daten gespeichert werden. Die Entwicklungszeiten und Produktlebenszeiten werden zudem immer kürzer. Die einzelnen Prozesse, die den kompletten Produktlebenszyklus darstellen, werden unter dem Begriff PDM zusammengefasst [MMP04]. „Unter PDM wird schwerpunktmäßig die Zuordnung von beliebigen IT– oder manuell erzeugten Dokumenten, z.B. 2D–Zeichnungen, 3D–Modelle, textuelle Dokumente, Berechnungsergebnisse zu Produktstamm– und –strukturdaten bzw. Projektdaten verstanden“[ES09].

2.2 PDM–Systeme

Es ist schwierig die anfallende Menge an Daten und Informationen in einer sinnvollen Struktur zu speichern. Alle Daten müssen archiviert werden, sodass diese, falls Änderungen gewünscht sind, leicht wieder auffindbar sind. Dafür benutzt man in der heutigen Industrie sogenannte Produktdatenmanagement–Systeme (PDM–Systeme). Diese erlauben es den gesamten Entwicklungsprozess über eine einheitliche Software zu verwalten.

2.2.1 Hintergründe

Die rechnergestützte Konstruktion wird bereits seit es Rechner gibt vorangetrieben. Dies wird auch als Computer-Aided Design (CAD) bezeichnet. Durch immer komplexer werdende Produkte wurde auch die Fülle an Daten immer größer. Viele Firmen fühlten sich zu Beginn der 1980er mit der Papier–basierten Produktdatenverwaltung überfordert und suchten nach Alternativen zur Daten– und Dokumentverwaltung. Da es noch keine PDM–Systeme gab,

entwickelten sie eigene Lösungen. Ende der 1980er erkannten einige Software–Firmen, welche bereits CAD–Systeme herstellten, ebenfalls diese Marktlücke und entwickelten daraufhin PDM–Systeme. [LX01]

2.2.2 Funktionalität

Bei PDM–Systemen werden die Produktdaten zentral abgelegt. Sie erleichtern somit die Prozessoptimierung und –änderung. Weiterhin ermöglichen moderne Systeme das Einsehen der Daten von mehreren Benutzern gleichzeitig wie man in Abbildung 2.1 sehen kann.

Beispielsweise stellen sie den einzelnen Benutzern Informationen über bestimmte Produktteile bereit. Neben einzelnen Teilen, deren 3D–Modelle und Zeichnungen, wird aber auch die gesamte Produktstruktur von PDM–Systemen zur Verfügung gestellt. Werden zum Beispiel Daten von Konstrukteuren verändert, verwaltet das PDM–System zusätzlich die einzelnen Versionen. Neben den Produktstruktur– und –modell–Daten können zusätzlich noch Spezifikationen, Testergebnisse und diverse Kosten abgerufen werden [MMP04].

PDM–Systeme können als ein bereichübergreifendes Werkzeug gesehen werden, welches die richtigen Informationen, der richtigen Person zur richtigen Zeit zur Verfügung stellt [LX01].

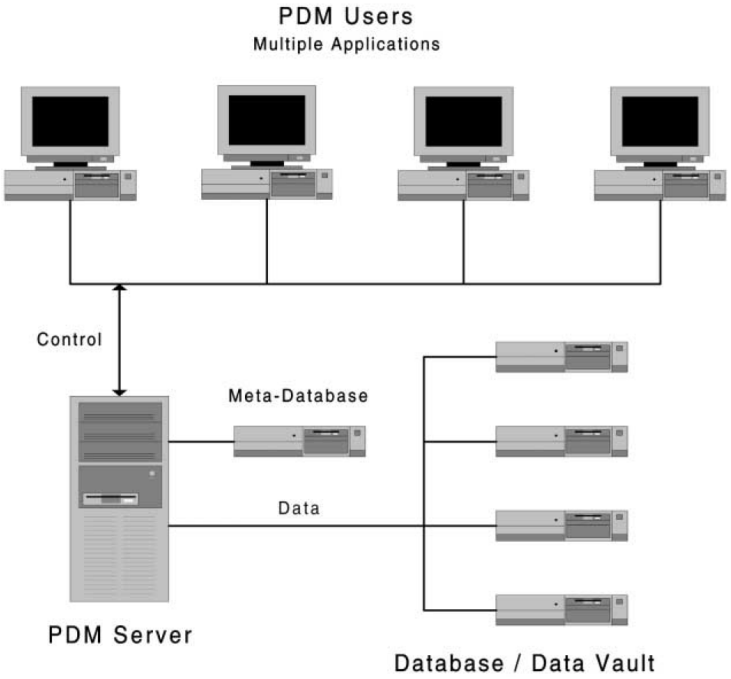

Abbildung 2.1: Architektur eines PDM–Systems [LX01]

2.2.3 Anwendungsbereiche

Es gibt eine Vielzahl unterschiedlicher, auf verschiedene Anwendungsbereiche ausgelegte PDM–Systeme. Große Industriefirmen haben häufig auch Eigenentwicklungen im Einsatz. Der Automotive–Hersteller Daimler setzt beispielsweise auf *Smaragd*. „Es unterstützt den Produktentstehungsprozess von der Erzeugung der ersten Geometrie– und Strukturdaten über den Aufbau digitaler Prototypen bis hin zur Planung für die Serienproduktion“ [ECS]. Bosch hingegen setzt auf das von Siemens entwickelte *Teamcenter*, welches eine domänenübergreifende Konstruktionsdatenverwaltung ermöglicht [TKS].

Moderne PDM–Systeme decken die meisten Anwendungsbereiche des Produktlebenszyklus ab. Im Bereich des Produktdesigns, der Konstruktion und der Produktion werden die Systeme beispielsweise zur Verwaltung der Produktstruktur und einzelnen Versionen / Varianten verwendet. Testergebnisse und anfallende Materialkosten werden hingegen eher in den Bereichen Vertrieb, Prozessplanung und Projektmanagement benötigt [LX01].

2.3 Webanwendungen

In den letzten Jahren wird immer häufiger das Internet verwendet, um Daten und Informationen an Mitarbeiter und Kunden weiterzureichen. Um diese Daten ansprechend darzustellen, verwendet man nicht nur die altbewährten Technologien wie Hypertext Markup Language (HTML) und Cascading Style Sheets (CSS) sondern setzt auch immer häufiger auf Frameworks. Damit kann die Darstellung der Daten programmiert werden. Meist wird hier auf JavaScript als Programmiersprache gesetzt. Die Daten werden dabei von einem Server bereitgestellt. So entstehen komplette Anwendungen, welche im Browser dargestellt werden. Das Model-View-Controller (MVC) Pattern spielt hierbei eine große Rolle. Durch die getrennte Client–Servers Architektur des Internets (siehe Abbildung 2.2) sind diese Anwendungen von überall aus der Welt erreichbar. Somit benötigen Anwender nicht unbedingt ein eigenes Endgerät, von dem sie auf die Daten zugreifen können.

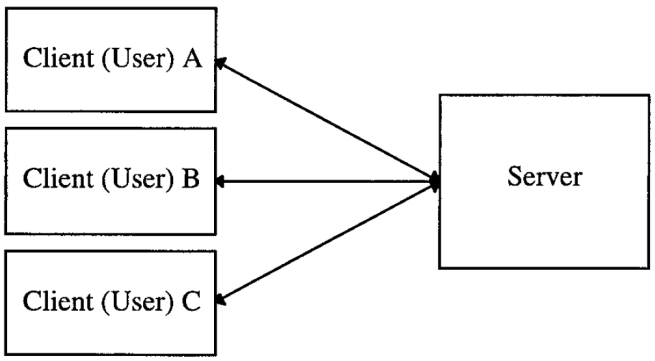

Abbildung 2.2: Client–Server Architektur [Lew98]

2.3.1 Model–View–Controller Pattern

Das MVC Pattern ist ein Modell um interaktive Anwendungen zu gestalten. Die Idee ist, die Datenhaltung (Model), die Datenverarbeitung (Controller) und die Darstellung (View) voneinander zu trennen. Dies erlaubt es die einzelnen Teile wiederzuverwenden oder auszutauschen. Somit können zum Beispiel Benutzerschnittstellen einfach ausgetauscht werden ohne die komplette Anwendungslogik neu aufbauen zu müssen. Die Datenhaltung stellt der Anwendung Daten bereit und speichert diese. Dies kann beispielsweise durch eine Datenbank realisiert werden. Die Datenverarbeitung hat die Aufgabe, Benutzer–Eingaben zu verarbeiten und dabei die entsprechenden Daten zu aktualisieren. Die Darstellung präsentiert dem Benutzer die Daten.

Das MVC Pattern wird häufig bei Webanwendungen eingesetzt. Durch das Client–Server Modell von Webanwendungen kommt es auf die Implementation an, wie und wo die Daten verarbeitet und gespeichert werden. Die Datenverarbeitung kann zum Beispiel auf dem Client mittels JavaScript oder auf dem Server implementiert werden. Häufig werden auch Kombinationen aus Datenverarbeitung auf dem Client und dem Server realisiert [LR01].

2.3.2 HTML

Die Hypertext Markup Language (HTML) ([HTM5]) ist eine Auszeichnungssprache, welche es erlaubt Informationen strukturiert darzustellen. Sie ist der Standard des World–Wide–Webs um statische Webseiten darzustellen. Besonders in der neuen Version HTML5 wurde die Darstellung von Informationen auf den neusten Stand gebracht. Direkt eingebettete Videos und Canvas–Elemente werden durch den Standard unterstützt. Die Offline–Speicherung von Daten ermöglicht es, komplette Webapplikationen auch ohne Internet zu verwenden. Standortdienste erlauben es nun Webanwendungen positionsbezogene Daten darzustellen [Vau10].

2.3.3 CSS

Cascading Style Sheets (CSS) ist eine Stylesheet–Sprache, die es erlaubt einzelne HTML–Elemente anzuordnen und das Aussehen dieser zu verändern [LBLJ05]. Mit der neusten Version CSS3 ist es sogar möglich, HTML–Elemente zu transformieren. Mögliche Transformationen sind

- Verschieben (translate)
- Rotieren (rotate)
- Skalieren (scale)

Es ist aber auch möglich, eigene Transformationsmatrizen zu definieren, mit welchen die Elemente auf dem Bildschirm platziert werden können [CST3]. Weiterhin ist es in der neuen Version möglich CSS–Werte langsam zu verändern [CST] oder sogar Animationen zu erstellen [CSA].

2.3.4 JavaScript

JavaScript ist eine clientseitige Skriptsprache, mit der HTML– und CSS–Elemente dynamisch bearbeitet werden können. Dies bedeutet, dass sie, eingebettet in HTML, Programmcode auf der Seite des Webanwenders ausführen kann. Dadurch lässt sich die Darstellung von Informationen programmieren [Fla06].

In neuerer Zeit wird häufig TypeScript anstelle von JavaScript zur Entwicklung von Webanwendungen eingesetzt. Dies ist ebenfalls eine Programmiersprache, welche eine einfachere Syntax hat und Programmierkonstrukte wie Interfaces und Vererbung zur Verfügung stellt. TypeScript lässt sich direkt in JavaScript–Code kompilieren. [BAT14].

2.4 Webframeworks

Webframeworks werden häufig verwendet um dynamische Webseiten einfach zu halten. Sie stellen Programmcode zur Verfügung, der ansonsten häufig wiederholt werden müsste.

2.4.1 Node.js

Node.js ist eine Event–basierte JavaScript–Laufzeitumgebung. Zu jedem Zeitpunkt an dem ein Event (z.B. eine neue Verbindung wird mit dem Server hergestellt) auftritt, wird JavaScript–Code ausgeführt. Damit können schnelle, skalierbare Netzwerkanwendungen erstellt werden.

Node.js stellt einen eigenen Webserver bereit, der zu Entwicklungszwecken verwendet wird. Der Code wird direkt während der Laufzeit kompiliert. Weiterhin ist es möglich, das eigentlich nur clientseitige JavaScript auf dem Server auszuführen (siehe Abbildung 2.2), wodurch keine zweite Programmiersprache für den Server benötigt wird. [And14]

2.4.2 Angular 2

Angular 2 ist ein clientseitiges JavaScript–Framework, das von Google entwickelt wird. Es wird häufig in TypeScript programmiert und läuft zu Entwicklungszwecken auf Servern wie Node.js oder .NET [AFB].

Das Konzept von Angular ist komponentenbasiert. Einzelne Komponenten bestehen aus einer JavaScript– / TypeScript–Logik, einem HTML–Template und CSS–Direktiven. Die einzelnen Komponenten können beliebig oft in andere Komponenten eingebettet werden, was die gesamte Webanwendung modular gestaltet und ein Wiederverwenden von Komponenten ermöglicht.

HTML–Templates sind HTML–Codestücke, die die Struktur der anzuzeigenden Daten beschreiben. Einzelne HTML–Elemente können mit speziellen Angular–Direktiven bearbeitet werden. Zum Beispiel ist es möglich, ein einzelnes Element mithilfe einer Schleife mehrmals darzustellen. In dem Template können Daten aus der JavaScript– / TypeScript–Logik gelesen und gesetzt werden. Weiterhin können so CSS–Werte mithilfe von Variablen gesetzt werden.

Angular beinhaltet ein weiteres Konzept, die *Angular Services*. Angular Services sind eine Art Schnittstelle, in die Komponenten eingebunden werden können. Sie stellen ihnen Daten und Methoden zur Datenmodifikation bereit. Wo die Daten abgelegt werden, bleibt dem Service überlassen. Häufig werden Daten in Datenbanken auf entfernten Servern gespeichert. Die Services können auch als eigene Komponenten gesehen werden.

Durch die Modularität der einzelnen Komponenten und Services kann Angular 2 als Implementierung des MVC Modells gesehen werden. Die einzelnen Teile wie Logik, Daten–Backend und Anzeige können dabei einfach ausgetauscht oder mehrfach verwendet werden.

2.4.3 RxJS

Die Reactive Extensions for JavaScript (RxJS) sind mehrere JavaScript–Bibliotheken, um asynchrone Daten zu handhaben und Event–basierte Webanwendungen zu erstellen. [RXJ] Mithilfe von ihnen kann man sogenannte Observable–Objekte erstellen. Diese können von Klassen abonniert werden. Wird dem Objekt ein neuer Wert zugewiesen, erhält jede Klasse, welche dieses abonniert hat, diesen. Mit dem neuen Wert wird dann eine in der Klasse definierte Funktion ausgeführt [SF16].

Damit kann man einfache JavaScript Schnittstellen realisieren und Daten durch die gesamte Webanwendung propagieren.

2.4.4 WebGL

Die Web Graphics Library (WebGL) ist eine Browsererweiterung, mit der hardwarebeschleunigte 3D–Inhalte dargestellt werden können. Es basiert auf OpenGL ES 2.0, das ursprünglich

eine OpenGL Adaption für Embedded–Systems wie Smartphones und Tablets ist. Somit ist es plattformunabhängig und steht unter keiner Lizenz. Inzwischen wird WebGL von allen gängigen Browsern, sowohl für den Desktop als auch für mobile Endgeräte, unterstützt.

Die 3D–Inhalte werden in einem HTML–Canvas–Element präsentiert. Programmiert wird die Darstellung von Inhalten komplett in JavaScript. WebGL kennt allerdings keine der üblichen, in der Computergrafik gängigen, Konstrukte wie Gitternetze, Materialien und Lichter. Es kennt nur Dreiecke. Somit gibt es auch kein definiertes Dateiformat für die Speicherung von 3D–Modellen. Viele 3D–Grafiksuites stellen aber inzwischen Exportfunktionen für WebGL bereit [Par14].

2.4.5 Three.js

Three.js ist eine weit verbreitete WebGL–Bibliothek, um 3D–Applikationen im Browser zu erstellen. Es abstrahiert die WebGL–API und präsentiert dem Programmierer 3D–Szenen aus Gitternetzen, Materialien und Lichtern. Dafür stellt es eigene Objekte bereit, die für Animationen und Datenvisualisierung benutzt werden können.

Three.js liest Dateiformate von vielen 3D–Grafiksuites ein. Es können auch Daten in einem eigenen JSON–basierten Format und für die kommerzielle Nutzung einem binären Format gelesen werden.

Three.js stellt im Gegensatz zu WebGL Funktionen bereit, welche die Interaktion mit den 3D–Objekten ermöglichen. Dies erlaubt es, interaktive 3D–Szenarien zu erstellen.

Weiterhin kann Three.js 3D–Szenen nicht nur mittels WebGL darstellen, sondern rendert diese auch in 2D oder direkt in ein HTML–Canvas Element. Zusätzlich ist es möglich CSS–Elemente darzustellen und zu transformieren, was insbesondere im Zusammenspiel mit Vektorgrafiken von Nutzen ist [Par14].

2.5 Grafische Benutzeroberflächen

Grafische Benutzeroberflächen sind ein Bestandteil vieler moderner Anwendungen. Sie sind die Schnittstelle zum Benutzer und erlauben die Interaktion mit der Anwendung.

Dem Benutzer sollten die benötigten Informationen logisch und übersichtlich dargestellt werden. Die Oberfläche darf keine wichtigen Informationen vor dem Nutzer verstecken. Außerdem sollten Oberflächen so gestaltet werden, dass ein konsistentes Gesamtbild entsteht.

Je nach Anwendungsgebiet gibt es verschiedene Möglichkeiten, um Oberflächen zu erstellen. Im Bereich der Webanwendungen liegt der Fokus auf HTML und CSS. Mit Frameworks wie

jQuery[1] oder Bootstrap[2] lassen sich einfach und schnell Oberflächen für Webanwendungen gestalten. Aber auch zum Aufbauen von kompletten Betriebssystemoberflächen gibt es mehrere Möglichkeiten. Hier liegt der Fokus nicht auf der Darstellung des Inhalts einzelner Anwendungen, sondern auf der Positionierung der Fenster, welche die Anwendung beinhalten.

2.5.1 Oberflächenkonzepte

Moderne Oberflächen haben meist einige gemeinsame Patterns. Einige von ihnen werden in diesem Abschnitt vorgestellt.

Ein beliebtes Oberflächenkonzept ist eine statische Menüleiste. In ihr sind Menüs enthalten, mit denen durch die Anwendung navigiert werden kann. Aber auch direkte Einstellungsmöglichkeiten oder Eingabefelder, wie es bei Browsern der Fall ist, werden dort untergebracht (siehe Abbildung 2.3). Meist sind sie horizontal ausgerichtet und befinden sich am oberen oder unteren Rand der Anwendung oder des Bildschirms. Dadurch legt der Benutzer den Fokus nur auf sie, wenn er die dort angezeigten Daten oder Funktionen benötigt. In einigen Fällen wird die Menüleiste auch ausgeblendet und durch Gesten (bei Touchscreens) oder Tastendruck wieder eingeblendet.

Virtuelle Oberflächen sind ein weiterer Bestandteil moderner Anwendungen. Dabei ist nur ein Teil der Gesamtoberfläche auf dem Bildschirm sichtbar. Der Rest geht über die Bildschirmränder hinaus und ist nicht sichtbar.

Bei Desktop–PCs werden Anwendungen häufig in Fenstern dargestellt. Dies sind rechteckige Beschränkungen, in welchen die Grafische Benutzeroberfläche von Anwendungen dargestellt werden. Öffnet man aus der Anwendung heraus ein Untermenü, wird dies in einem neuen Fenster dargestellt. Bei mobilen Anwendungen erscheint das Anwendungsfenster meist im Vollbild. Weitere Fenster, wie zum Beispiel ein Einstellungsmenü, erscheinen beim Öffnen ebenfalls im Vollbild. Schließt man dieses Einstellungsmenü, befindet man sich wieder im eigentlichen Anwendungsfenster. Häufig kann man auch per Gesten zwischen mehreren Anwendungsoberflächen wechseln. Bei vielen Betriebssystemoberflächen lassen sich mehrere Anwendungen auf diese Art darstellen. Eine oder mehrere Anwendungen befindet sich auf einem virtuellen Desktop. Wechselt der Benutzer den virtuellen Desktop erhält er eine leere Oberfläche auf welcher er neue Anwendungen positionieren kann. Als Beispiel dient hier der

Abbildung 2.3: Menüleiste von Google Chrome mit Texteingabe, Buttons und Menüknopf

[1] https://jquery.com/
[2] https://getbootstrap.com/

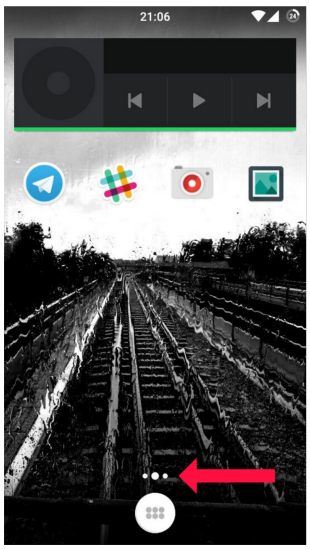

Abbildung 2.4: Android–Homescreen mit drei virtuellen Oberflächen, dargestellt durch die drei Punkte am unteren Bildschirmrand. Die aktuell Ausgewählte wird als etwas größerer Punkt dargestellt.

Android–Homescreen auf welchem kleine Anwendungen und Programmstarter auf beliebig vielen Oberflächen positioniert werden können (siehe Abbildung 2.4).

Auf die Positionierung der Anwendungen soll hier besonderer Augenmerk gelegt werden. Es gibt mehrere Möglichkeiten Fenster auf dem Desktop zu platzieren. Meist geschieht dies per Drag–n–Drop mit der Maus. Man kann Fenster vergrößern und verkleinern. Bei mobilen Endgeräten werden Anwendungen aufgrund der geringen Displaygröße hingegen meist im Vollbild dargestellt.

3 Verwandte Arbeiten

3.1 Online–PDM–Systeme

In den folgenden Abschnitten werden zwei, mit dem Konzept der Bachelorarbeit, verwandte Online–PDM–Systeme vorgestellt. Beide Anwendungen sind für die Desktop–Verwendung optimiert, es existieren aber Erweiterungen, um Daten im Browser darzustellen. Dafür müssen die Daten auf einem Server, zentral bereitgestellt werden.

3.1.1 ImageSite

ImageSite ist ein von eQuorum entwickeltes Dokumentmanagement–System mit dem Fokus auf Versionsverwaltung. Es kann alle möglichen Dokumente, angefangen mit PDF, über Microsoft Office Dokumente, bis hin zu komplexen CAD–Zeichnungen verwalten.

ImageSite unterstützt Datei–Referenzierungen und –Strukturen von CAD Dateien. Diese können auch im Programm selbst als 3D–Modell oder Zeichnung angezeigt werden. Mithilfe der integrierten Suche ist es möglich, nach Teilinformationen suchen.

Weiterhin kann mit ImageSite der komplette Produktlebenszyklus, von der Konstruktion bis hin zur Verwertung, verwaltet werden.

Mit ImageSite–Mobile existiert eine Erweiterung für ImageSite, mit der alle Dokumente und Dateien auch auf mobilen Endgeräten angezeigt werden können. Diese Anwendung ist komplett in HTML5 geschrieben und läuft somit auf allen Geräten, die einen Browser besitzen. Ebenso ist die Offline–Nutzung von Daten möglich [IDM].

3.1.2 SpinFire

Im Gegensatz zu ImageSite ist SpinFire ein von dem Unternehmen Actify entwickelter CAD–Viewer, bei welchem der Fokus nicht auf der Dokumentverwaltung, sondern auf der eigentlichen Darstellung von 3D–Modellen liegt.

Die Modelle können zum Beispiel auch im Querschnitt und in einer Explosionsansicht dargestellt werden. Sie können mit Kommentaren und Anmerkungen direkt in der Zeichnung versehen werden [ASU].

Mit SpinFire Web existiert eine Version des Programms, welches auf einem Server eines Unternehmens installiert werden kann. Mitarbeiter können dann in einem Web Browser die Modelle betrachten und sind somit, wie bei ImageSite, unabhängig von dem Endgerät [ASW].

3.2 Oberflächen

In diesem Abschnitt werden diverse Konzepte von Anwendungen und Desktop–Umgebungen vorgestellt und verglichen.

3.2.1 Anwendungen

Moderne Desktop–Anwendungen bestehen oft aus mehreren Teilansichten, welche auf die Darstellung oder Interaktion von verschiedenen Daten ausgelegt sind. Die einzelnen Programmteile werden dabei auf einer Ebene angezeigt und nebeneinander, überlappungsfrei positioniert. Häufig werden auch Tabs verwendet um ähnliche Daten zu gruppieren. In einem Texteditor wird auf der linken Seite eine Baumstruktur angezeigt, welche die Ordnerstruktur der Dateien visualisiert. Die Größe der Baumstruktur lässt sich per Drag–n–Drop verändern. In der Mitte wird der Inhalt von geöffneten Dateien angezeigt, welcher sich verändern lässt. Zwischen geöffneten Dateien kann man mittels Tabs navigieren (siehe Abbildung 3.1).

Größere Unterprogramme wie zum Beispiel ein Einstellungsmenü werden meist in einem neuen Fenster geöffnet und beinhalten wieder eigene Teilansichten. Das eigentliche Programmfenster ist weiterhin sichtbar.

Apps auf mobilen Endgeräten werden aufgrund der geringen Bildschirmgröße meist im Vollbild dargestellt um den Platz optimal auszunutzen. Wird zum Beispiel ein Einstellungsmenü geöffnet,

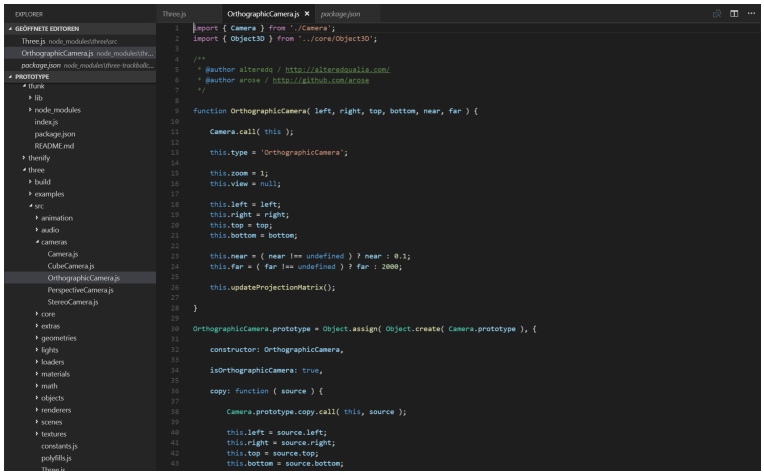

Abbildung 3.1: Visual Studio Code mit mehreren geöffneten Dateien

wird dies im Gegensatz zu Desktop Anwendungen über der App im Vollbild angezeigt. Die eigentliche App ist dann nicht mehr sichtbar.

Menüleiste

Viele Anwendungen haben eine Menüleiste am oberen Rand. Sie beinhaltet meist mehrere Untermenüs durch welche sich Funktionen des Programms aufrufen und ändern lassen. Oft befinden sich auch direkt Knöpfe in der Leiste.

Aufgrund der geringen Größe von mobilen Engeräten wird dort die Menüleiste meist ausgeblendet. Zusätzlich wird die nicht horizontal dargestellt, sondern vertikal und ist mit wischen vom linken oder rechten Bildschirmrand in die Mitte des Bildschirms erreichbar [NAV]. Die Eigentliche Funktion der Leiste ist aber die selbe. Mit ihr lässt sich leicht durch die App navigieren.

3.2.2 Desktop–Umgebungen

In kompletten Desktop–Umgebungen finden sich einige der Konzepte aus den modernen Anwendungen wieder.

Statusleiste

Die Statusleiste ist in vielen Desktop–Umgebungen integriert. Sie befindet sich meist am oberen oder unteren Bildschirmrand und spannt sich über die gesamte Bildschirmbreite auf. Je nach Implementierung stellt sie verschiedene Funktionen zur Verfügung. In den gängigen Desktop–Umgebungen, wie der Windows Desktop oder das K Desktop Environment, ist es möglich, Anwendungsstarter in der Leiste zu platzieren. Weiterhin werden dort offene Anwendungen angezeigt, auf welche man den Fokus legen kann. Informationen wie Batteriestatus bei Laptops, Lautstärke und Uhrzeit werden ebenfalls angezeigt [PLS]. Sie stellen eine Art Menüleiste der Desktop–Umgebung dar, ähnlich wie bei den Anwendungen.

Bei Android existiert ein ähnliches Konzept. Am oberen Bildschirmrand befindet sich eine Leiste, welche aktuelle Benachrichtigungen, Uhrzeit, Akkulaufzeit, Signalstärke und weitere Systeminformationen anzeigt. Durch das Herunterziehen der Leiste werden die kompletten Benachrichtigungen und Systemfunktionen angezeigt [MS].

Fenstermanager

Der Fenstermanager ist ein Programm mit der Aufgabe, die Anwendungsfenster auf dem Desktop zu platzieren. Hierbei gibt es verschiedene Methoden. Die beiden gängigsten sind Floating und Tiling. Beide werden im Folgenden näher beschrieben.

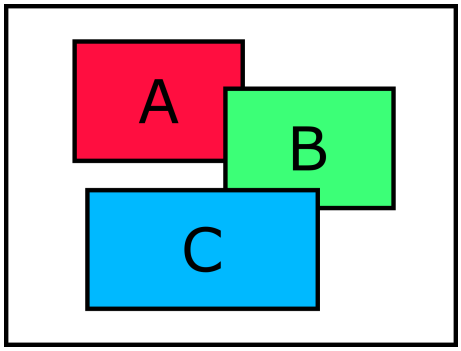

Abbildung 3.2: Darstellung von Fenstern mit einem Floating Fenstermanager

Floating

Bei der Darstellung von Fenstern mit einem Floating (dt. fließend) Fenstermanager kann der Benutzer die Anwendungsfenster frei auf seiner Oberfläche positionieren. Dabei können diese sich überlappen oder sogar komplett verdecken. Der Desktop wird dadurch in verschiedene Ebenen aufgeteilt (siehe Abbildung 3.2).

Die Fenster können, meist per Drag–n–Drop, verschoben werden. Auf die gleiche Weise kann auch ihre Größe verändert werden. Wird der Fokus auf ein Fenster gelegt, das durch ein anderes verdeckt wird, wird dieses automatisch auf die oberste Ebene verschoben. Somit ist der gesamte Inhalt sichtbar. Meist enthält die Fensterleiste Knöpfe, mit denen der Benutzer die Anwendungen minimieren oder das Fenster auf den kompletten Desktop maximieren kann.

Bei vielen Fenstermanagern sind diverse Einrastoptionen implementiert. Wird ein Fenster über ein Anderes geschoben, stoppt das Fenster dabei kurz. So lassen sich Anwendungen überlappungsfrei direkt nebeneinander positionieren. Zusätzlich sind Fensterpositionierungen mit den Bildschirmrändern möglich. Wird ein Fenster an den Bildschirmrand geschoben, kann es auf eine bestimmte Größe gesetzt werden. Die linke obere Ecke setzt die Fenstergröße beispielsweise auf ein Viertel des Bildschirms und platziert es in der linken oberen Ecke. Der linke Rand setzt das Fenster auf die halbe Größe des Bildschirms und positioniert es am linken Rand. Mit Hilfe von diesen Operationen lassen sich Fenster ebenfalls überlappungsfrei darstellen.

Beispielanwendungen für Floating Window Management sind KWin[1] oder Openbox[2].

[1] https://userbase.kde.org/KWin
[2] http://openbox.org/

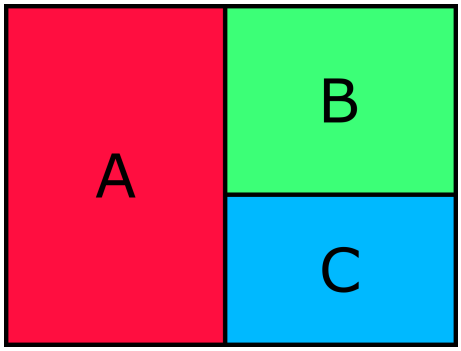

Abbildung 3.3: Darstellung von Fenstern mit einem Tiling Fenstermanager

Tiling

Im Gegensatz zu Floating Fenstermanagern, liegt der Fokus bei den Tiling (dt. Kacheln) Fenstermanagern auf der überlappungsfreien Positionierung von Fenstern. Dadurch sieht der Benutzer immer alle offenen Anwendungen und es werden keine der dargestellten Informationen versteckt oder verdeckt (siehe Abbildung 3.3).

Das Prinzip dabei ist, dass der gesamte Bildschirm genutzt und kein Platz verschwendet wird. Ist nur eine Anwendung offen, wird diese auf die gesamte Bildschirmgröße maximiert dargestellt. Wird eine zweite Anwendung geöffnet, verkleinert sich die bereits offene Anwendung nach einem vordefinierten Teilungsverhältnis und die zweite Anwendung wird direkt neben der ersten dargestellt. Dies lässt sich mit beliebig vielen Anwendungen wiederholen, wobei jeweils der komplette Fensterinhalt dargestellt wird. Die Teilverhältnisse zwischen den Fenstern können, meist per Drag–n–Drop oder Tastaturkürzeln, verändert werden. Das macht die Darstellung von Fenstern flexibel [IUG]. Das Tiling Window Management ist vergleichbar mit dem Aufbau von Desktop–Anwendungen. Die einzelnen Anwendungsfenster sind dabei Teile des Programms, welche auf die Darstellung von bestimmten Daten ausgelegt sind. Diese befinden sich auch hier auf einer Ebene und werden überlappungsfrei dargestellt.

Die Anordnung der Fenster wird häufig als Baum gespeichert. Zum Beispiel bei den Tiling Window Managern i3 [IWM] oder bspwm [BSP]. Hierbei werden die Ausrichtung, die Teilverhältnisse und die Fenster IDs gespeichert. In Listing 3.1 sieht man die Anordnung von Fenstern in dem Tiling Window Manager i3, welcher die Anordnung von Fenstern in JavaScript Object Notation (JSON)–Dateien speichert. Der Bildschirm ist hierbei in zwei Spalten aufgeteilt. Die linke Spalte füllt 40% des Bildschirms und enthält zwei Unterfenster mit *irssi* und *Emacs*. Die rechten 60% werden von *Google Chrome* ausgefüllt, wie man in Abbildung 3.4 sehen kann.

Dadurch, dass immer alle Anwendungen zugleich dargestellt werden, wird der eigentliche Anwendungsinhalt, im Gegensatz zum Floating Prinzip, bei vielen offenen Anwendungen sehr klein angezeigt. Deswegen beinhalten viele Tiling Fenstermanger zusätzliche Konzepte um dies zu vermeiden. Meist wird dabei auf mehrere virtuelle Desktops gesetzt oder es sind zusätzliche Funktionen, wie eine mögliche Floating–Positionierung, der Fenster implementiert.

Listing 3.1 JSON Datei welche das Fenster–Layout des tiling window manager i3 darstellt

```
{
      // splitv split container with 2 children
      "layout": "splitv",
      "percent": 0.4,
      "type": "con",
      "nodes": [{
            "border": "none",
            "name": "irssi",
            "percent": 0.5,
            "type": "con",
            "swallows": [{
                  "class": "^URxvt$",
                  "instance": "^irssi$"
            }]
            },
            {
                  // stacked split container with 2 children
                  "layout": "stacked",
                  "percent": 0.5,
                  "type": "con",
                  "nodes": [{
                  "name": "notmuch",
                  "percent": 0.5,
                  "type": "con",
                  "swallows": [{
                        "class": "^Emacs$",
                        "instance": "^notmuch$"
                  }]
                  },
                  {
                        "name": "midna: ~",
                        "percent": 0.5,
                        "type": "con"
                  }]
            }]
      }

      {
            // stacked split container with 1 children
            "layout": "stacked",
            "percent": 0.6,
            "type": "con",
            "nodes": [{
            "name": "chrome",
            "type": "con",
            "swallows": [{
                  "class": "^Google-chrome$"
            }]
      }]
}
```

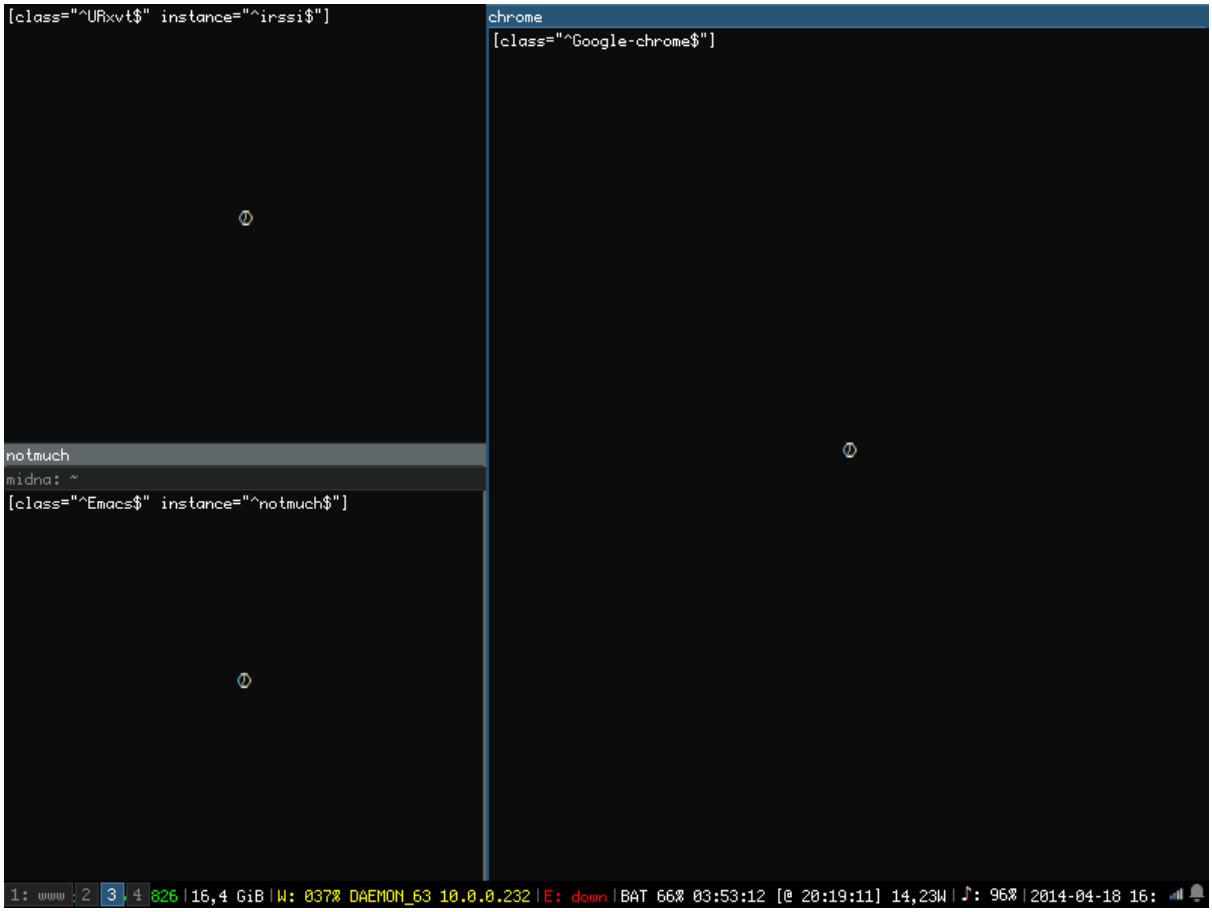

Abbildung 3.4: i3 window manager mit drei Platzhalter–Fenstern

Beispielanwendungen für Tiling Window Management sind i3 window manager [IWM] und binary space partition window manager [BSP].

Virtuelle Oberflächen

Virtuelle Oberflächen sind ein beliebtes Konzept bei Desktop–Umgebungen. Eine virtuelle Oberfläche ist ein Desktop auf welchem Anwendungsfenster platziert werden können. Häufig können beliebig viele dieser virtuellen Desktops erstellt werden. Im Gegensatz zu einem statischen Desktop erlaubt es dieses Konzept Anwendungen zu gruppieren. Beispielsweise kann eine virtuelle Oberfläche mit Web Anwendungen, wie Browser und Mail–Programm, und eine weitere mit Multimedia Anwendungen erstellt werden. Somit wird der Fokus auf das gerade benötigte Anwendungsgebiet gelegt. In Kombination mit mehreren Bildschirmen werden so flexible Kombinationen von verschiedenen Anwendungen möglich [IUG]. Eine Schematische Darstellung von drei virtuellen Oberflächen befindet sich in Abbildung 3.5.

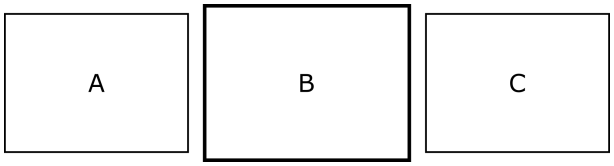

Abbildung 3.5: Drei virtuelle Oberflächen, wobei nur die mittlere auf dem Bildschirm angezeigt wird

Bei vielen Android Homescreens ist ein ähnliches Prinzip implementiert. Die Oberfläche besteht dabei aus mehreren einzelnen Seiten, welche sich meist mit Swipen wechseln lassen. Im Gegensatz zu den virtuellen Oberflächen lassen sich auf den einzelnen Seiten allerdings keine offenen Anwendungen platzieren, sondern Anwendungsstarter und sogenannte Widgets. Diese sind kleine Programme, welche meist Funktionen einer App zur Verfügung stellen. Beispielsweise lässt sich mit ihnen die Musikwiedergabe ändern oder Kalendereinträge anzeigen [MS].

Ein ähnliches Prinzip verfolgt die Windows 8 Startseite. Auf ihr können ebenfalls Anwendungsstarter oder Widgets platziert werden, sogenannte Kacheln. Diese können ähnlich dem Tiling Window Management frei auf der Startseite platziert werden. Im Gegensatz zum Android Homescreen existiert hier nur eine virtuelle Seite auf welcher Kacheln platziert werden können. Dafür lässt sich diese in die Horizontale beliebig erweitern. Navigiert werden kann auf ihr mit Scrollen oder Swipen.

4 Anforderungen

Dieses Kapitel der Arbeit beschreibt die Anforderungen an das in dieser Bachelorarbeit erarbeitete Konzept.

4.1 Verschiedene Geschäftsfelder

PDM–Systeme werden in unterschiedlichen Geschäftsfeldern eingesetzt. In jedem Geschäftsfeld sind verschiedene Daten von Bedeutung. Im Folgenden werden die vier Geschäftsfelder Lager, Werkstatt, Konstruktion und Management aufgezählt.

Die betrachteten Daten sind Teilinformationen, Modelldaten, Strukturdaten und Versionsdaten. Jedes CAD–Modell besteht aus verschiedenen Teilen. Diese werden in Baugruppen zusammengefasst. Teilinformationen sind zum Beispiel Teilnamen oder Material. Die Modelldaten sind die 3D–Zeichnung eines Teils. Die Strukturdaten sind die hierarchische Struktur von mehreren Teilen in einem Gesamtmodell. Die Versionsdaten sind Informationen über verschiedene Versionen eines Teils oder einer Baugruppe.

4.1.1 Lager

Teile werden im Lager aufbewahrt. Um ein bestimmtes Teil schnell zu finden werden Informationen über dieses benötigt. In einem großen Lager hat jedes Teil eine spezifische ID über welche es eindeutig identifizierbar ist. Wird ein Teil aus dem Lager benötigt soll mit wenig Aufwand die ID des entsprechenden Teils herausgefunden werden können.

4.1.2 Werkstatt

In der Werkstatt werden mehrere Teile zu einem Endprodukt zusammengesetzt. Dafür benötigt man mehrere Daten. Einerseits werden wie im Lager Teilinformationen benötigt. Die einzubauenden Teile können zum Beispiel mit IDs eindeutig identifiziert werden. Andererseits sind in der Werkstatt auch Strukturdaten und Modelldaten des Produkts von Bedeutung. Wird das Produkt zusammengebaut, kann man mit diesen in Kombination mit den Teilinformationen zum Beispiel die Positionierung und Ausrichtung in dem Produkt sehen. Den Monteuren sollen

die von ihnen benötigten Daten möglichst übersichtlich präsentiert werden und einzelne Teile sollen in dem kompletten Produktmodell leicht identifizierbar sein.

4.1.3 Konstruktion

In der Konstruktion werden neue Produktteile oder Produkte entworfen. Dafür sind ebenfalls mehrere Daten notwendig. Zum Beispiel werden beim Erstellen eines neuen Produktteils Modelldaten, Strukturdaten, und Teilinformationen angelegt. Wird ein Produktteil verändert, werden auch neue Versionsdaten erstellt oder bei der Änderung des Materials die Teilinformationen verändert. Durch Abrufen älterer Versionen können Änderungen auch rückgängig gemacht werden. Den Konstrukteuren sollen alle von ihnen benötigten Daten möglichst übersichtlich präsentiert werden.

4.1.4 Management

Im Bereich des Managements sind Modelldaten von Wichtigkeit. Diese zeigen das Endprodukt welches zum Beispiel in Meetings vorgezeigt werden kann. Aber auch Versionsdaten sind von großer Bedeutung um einfach Unterschiede zwischen verschiedenen Versionen des Produkts aufzeigen zu können. Teilinformationen können benutzt werden, um zum Beispiel die gesamten Materialkosten eines Produkts zu errechnen. Die Daten sollen hierbei ansprechend präsentiert werden und einfach handhabbar sein.

4.2 Endgeräte

Moderne Anwendungen laufen meist auf mehreren Endgeräten. Jedes Endgerät hat andere Anforderungen an diese. Verschiedene Displaygrößen und Eingabegeräte spielen hier eine große Rolle.

4.2.1 Computer

Eingabe

Wird eine Anwendung auf einem traditionellen Desktop–Rechner ausgeführt, soll der Benutzer diese mit Tastatur und Maus bedienen können. Dies erlaubt auch eine Navigation durch die Anwendung mit Tastenkürzeln.

Bildschirme

Da meist große Bildschirme verwendet werden, ist genügend Platz vorhanden um die einzelnen Menüs mit geringer Tiefe zu erstellen. Es können viele Informationen auf einmal dargestellt werden ohne das die Übersichtlichkeit leiden muss. Der Platz sollte hierbei sinnvoll genutzt werden.

4.2.2 Tablet / Smartphone

Bei der Ausführung von Anwendungen auf mobilen Endgeräten wie Tablets und Smartphones muss vor allem auf die Darstellung der anzuzeigenden Daten geachtet werden.

Eingabe

Der Benutzer interagiert mit der Anwendung meist über einen Touchscreen. Dabei werden häufig Gesten wie

- Tap (Klicken mit einem oder mehr Fingern)
- Swipe (Wischen mit einem oder mehr Fingern)
- Pinch–to–Zoom (Wischen mit zwei oder mehr Fingern in entgegengesetzte Richtung)

verwendet. Die Texteingabe erfolgt meist über eine On–Screen–Tastatur, welche nur eingeblendet wird wenn eine Eingabe getätigt werden muss. Tastenkürzel sind bei modernen virtuellen Tastaturen meist nicht implementiert und es muss darauf geachtet werden, dass die Tastatur keine wichtigen Daten überdeckt.

Bildschirme

Durch die geringe Displaygröße wirken Anwendungen schnell überladen und unübersichtlich. Man sollte deswegen Menüs und Knöpfe eher ausblenden oder verstecken um den Fokus auf die Daten zu legen. Hierbei darf aber die Funktionalität nicht leiden. Oft werden einzelne Anwendungsteile automatisch in eine mobile Ansicht umgeschaltet, welche die Daten für kleine Bildschirme optimiert darstellt.

Häufig beziehen Anwendungen für mobile Endgeräte ihre Daten über ein Netzwerk. Ist dieses Netzwerk nicht verfügbar müssen die Daten offline bereitgestellt werden damit der Benutzer die Anwendung trotzdem verwenden kann.

4.3 Bedienung

Damit Endanwender Anwendungen leicht bedienen können muss auf mehrere Dinge geachtet werden:

- Die Bedienung von Anwendungen, egal welcher Art, sollte möglichst selbsterklärend und intuitiv sein.
- Buttons und Menüs sollen eindeutig identifizierbar sein und der Benutzer soll durch ihre Beschriftung ihre Funktionalität erkennen können.
- Werden Touchscreens verwendet, muss auf die mögliche Verwendung von Gesten Rücksicht genommen werden.
- Bei der Bedienung mit Tastatur und Maus sollen die Touchscreen Gesten Tap und Swipe mit den Mausgesten Klicken und Drag–n–Drop gleichgesetzt werden.
- Die Bedienung innerhalb einer Anwendung sollte sich nicht verändern und konsistent sein.
- Werden Tastaturkürzel verwendet, müssen diese innerhalb der Anwendung eindeutig und nicht doppelt belegt sein.

4.4 Bausteine

Die Daten in der Konzeptanwendung sollen in einzelne, auf verschiedene Arten ausgelegte, Bausteine präsentiert werden. Dafür soll Folgendes gelten:

- Diese sollen so groß wie möglich dargestellt werden und sich nicht überlappen.
- Ihre Größe soll sich beliebig und leicht ändern lassen.
- Passt der Bausteininhalt aufgrund zu wenig Platz nicht in den Baustein selbst, soll der Inhalt auf die Größe angepasst präsentiert werden.
- Dargestellte Daten sollen zum Beispiel Teilinformationen oder Strukturdaten sein.

4.5 Raster

Einzelne Bausteine sollen in einer Art Raster angeordnet werden um größtmögliche Übersichtlichkeit und Nutzung des vorhandenen Platzes zu gewährleisten. Dafür soll auf Folgendes geachtet werden:

- Beliebig viele Raster sollen verfügbar sein um Bausteine darauf anordnen zu können.

- Die einzelnen Raster sollen leicht identifizierbar sein.

Bausteinanordnung

Die Bausteinanordnung soll in einem Einstellungsmenü erfolgen, für welches Folgendes gelten soll:

- Es soll einfach erreichbar und logisch aufgebaut sein.
- Innerhalb der Bausteinauswahl sollen die einzelnen Bausteine eindeutig identifizierbar sein.
- Die Bausteine sollen in einem Raster möglichst intuitiv hinzugefügt und entfernt werden können.
- Ihre Größe und Anordnung soll darin leicht veränderbar sein.
- Die Rasteranordnungen sollen bei dem Schließen des Einstellungsmenüs automatisch gespeichert werden.
- Mehrere Raster sollen einfach hinzugefügt und entfernt werden können.
- Die Rasteranordnungen sollen für jeden Benutzer zentral hinterlegt werden, damit dieser auf jedem Endgerät die gleichen Einstellungen zur Verfügung hat.

4.6 Design

Die Anwendung soll optisch ansprechend und modern gestaltet sein. Die Farben sollen ruhig und passiv wirken um den Benutzer nicht von den eigentlich wichtigen Daten abzulenken. Bei hellem Licht, zum Beispiel Sonnenlicht soll der Inhalt genau so gut lesbar sein wie in dunklen Räumen.

5 Konzept

Dieses Kapitel beschreibt das für die Bachelorarbeit erarbeitete Konzept. Dabei wird auf die Anforderungen aus Kapitel 4, sowie auf die verwandten Arbeiten aus 3, eingegangen.

Um das Konzept an alle möglichen verschiedenen Geschäftsfelder anzupassen, wurde, wie im Thema der Arbeit bereits genannt, ein bausteinbasierter Ansatz gewählt. Der Benutzer kann sich seine eigene Komposition von Anwendungen aus diversen Bausteinen zusammensetzen. Bausteine sind hierbei kleine Anwendungen, die die Aufgabe haben, die Informationen darzustellen. So bekommt der Benutzer genau die Informationen präsentiert, welche er benötigt. Das Konzept ist zum Anzeigen von Informationen gedacht. Die Daten können damit nicht verändert werden.

5.1 Benutzeroberfläche

Die Benutzeroberfläche ist in eine *Titelleiste* am oberen Rand, einer *Indikatorleiste* am linken Bildschirmrand und einer *Oberfläche* in der Mitte, auf welcher zwei Hauptansichten dargestellt werden, aufgeteilt. Diese sind das *Dashboard*– und die *Einstellungsansicht*. Der Benutzer sieht immer nur eine der beiden Ansichten im Vollbild. Wechseln kann er zwischen ihnen über einen *Einstellungsknopf* in der Titelleiste ähnlich der Menüleiste oder der Statusleiste wie in Kapitel 3 beschrieben. Eine schematische Darstellung befindet sich in Abbildung 5.1.

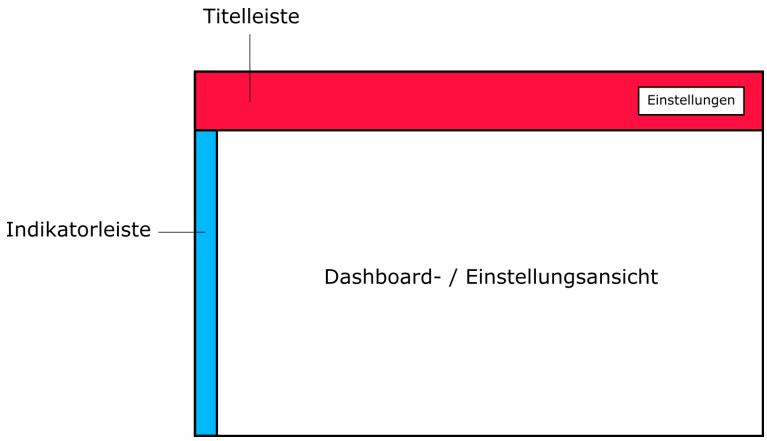

Abbildung 5.1: Schematische Darstellung der Benutzeroberfläche

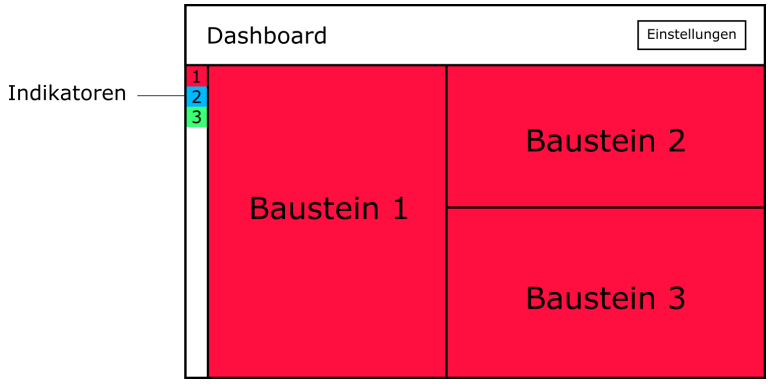

Abbildung 5.2: Darstellung des Dashboards mit Indikatoren für drei verschiedene virtuelle Oberflächen in der Indikatorleiste

5.1.1 Dashboard

Das Dashboard ist die Arbeitsansicht, welche der Benutzer verwendet um Daten zu betrachten. Einzelne Bausteinmodule werden dort überlappungsfrei, ähnlich dem Tiling Window Management, dargestellt um größtmöglich Übersichtlichkeit zu bieten (siehe Kapitel 3). Der Benutzer kann zwischen mehreren virtuellen Oberflächen wechseln auf welchen verschiedene Bausteine angeordnet sind. Dies geschieht mittels eines Klicks auf den entsprechenden Indikator in der Indikatorleiste. Jede virtuelle Oberfläche hat eine Nummer und eine Farbe, in welcher die Bausteine eingefärbt werden um dem Benutzer zu verdeutlichen auf welcher Oberfläche er sich befindet. Eine Konzept–Darstellung für das Dashboard befindet sich in Abbildung 5.2.

5.1.2 Einstellungen

Ist die Einstellungsansicht aktiv, sieht der Benutzer nichts von dem Bausteininhalt, sondern nur Platzhalter für diese. Die Bausteine auf der aktiven virtuellen Oberfläche werden in der Mitte dargestellt. Sind keine Bausteine positioniert, erscheint eine leere Fläche. Zusätzlich sind Trennelemente zwischen den Bausteinen. In der rechten oberen Ecke jedes Bausteins befindet sich ein Löschen–Symbol. Alle verfügbaren Bausteine werden in einer Leiste auf der rechten Seite präsentiert. Per Drag–n–Drop können diese auf die Darstellung der Baumstruktur gelegt werden.

Es sollen diverse Funktionen zur Bearbeitung der Ansicht implementiert werden. Diese sind:

Funktion 1: ***drag*** – Verändert das Teilverhältnis zischen den Bausteinen auf beiden Seiten des Trennelements (siehe Abbildung 5.3).

Funktion 2: ***toggle*** – Verändert die Orientierung der Trennung zwischen horizontal und vertikal. Standardmäßig wird immer vertikal geteilt.

Funktion 3: ***flip*** – Vertauscht die beiden Bausteine auf den Seiten des Trennelements.

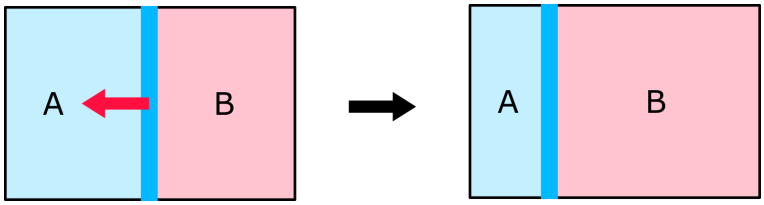

Abbildung 5.3: Wird das Trennelement verschoben, verändert sich das Teilverhältnis zwischen den beiden Bausteinen

Funktion 4: ***delete*** – Löscht einen Baustein aus der Ansicht. Der Baustein auf der anderen Seite des Trennelements wird dann über die gesamte ursprüngliche Fläche beider Bausteine dargestellt.

Funktion 5: ***insert*** – Fügt einen Baustein der Ansicht hinzu. Ist noch kein Baustein vorhanden wird der neue Baustein in der vollen Größe der Arbeitsfläche dargestellt. Sind bereits Bausteine vorhandenen, kommt es darauf an auf welchem der neue Baustein fallengelassen wird. Die Fläche, auf welcher der ursprüngliche Baustein aufgespannt wurde, wird dann genau in der Hälfte geteilt und die beiden Bausteine werden nebeneinander angezeigt.

Durch die Kombination der beschriebenen Funktionen ist es möglich die Bausteine in jeder erdenklichen Anordnung zu positionieren.

In der Indikatorleiste sind, im Gegensatz zur Dashboard–Ansicht, noch ein Hinzufügen und Löschen Knopf zu sehen. Über einen Klick auf den Hinzufügen–Knopf, wird eine neue leere virtuelle Oberfläche erstellt. Somit lassen sich beliebig viele Bausteinanordnungen realisieren. Ein Klick auf den Löschen–Knopf löscht die aktuell aktive virtuelle Oberfläche. Leere Oberflächen werden, sobald sie nicht mehr fokussiert sind, automatisch gelöscht. Die Nummerierung und Einfärbung der virtuellen Oberflächen geschieht automatisch.

In der Titelleiste der Oberfläche werden weitere Einstellungen als Drop–down Menü angezeigt. Diese sind Caching Mode, über welches die Caching–Variante der PDM / CAD–Daten eingestellt werden kann und ein Login über welchen man sich auf dem Server, welcher die Daten bereitstellt, einloggen kann.

Wird die Einstellungsansicht über einen Klick auf den Knopf in der Titelleiste geschlossen, werden die Baustein–Ansichten automatisch auf dem Server gespeichert. Der Benutzer erhält somit auf allen Endgeräten, auf welchen er sich anmeldet, die gleiche Anordnung der Bausteine.

Eine schematische Darstellung der Einstellungsansicht befindet sich in Abbildung 5.4.

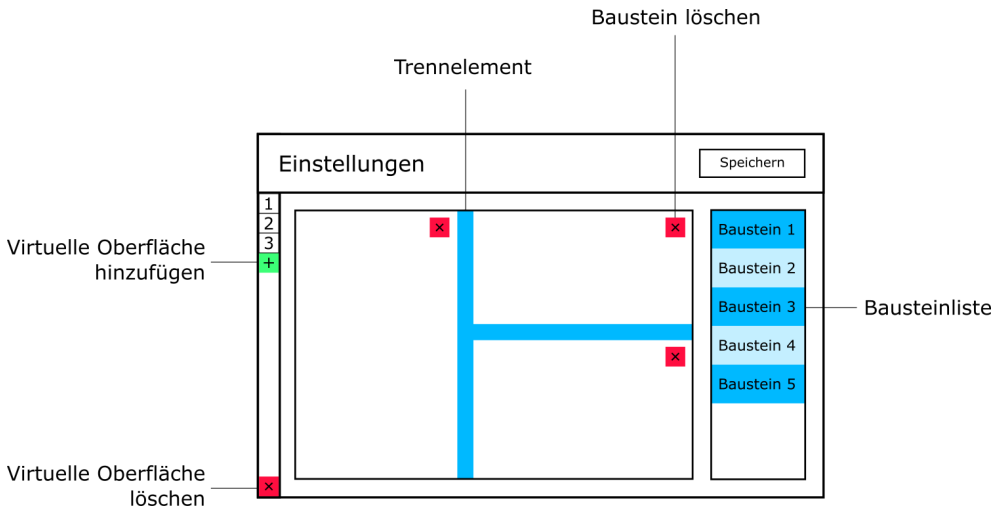

Abbildung 5.4: Schematische Darstellung der Einstellungsansicht

5.2 Bausteine

Bausteine sind kleine Unterprogramme, welche für ein bestimmtes Anwendungsgebiet optimiert sind. Jeder Baustein hat eine eigene Titelleiste in welcher der Name angezeigt wird und je nach Baustein verschiedene Knöpfe oder Eingabefelder. Die Titelleiste wird, je nach aktiver virtueller Oberfläche, in deren Farbe eingefärbt. Ein Baustein erhält Modell– und Strukturdaten über eine Schnittstelle. Er kann auch bestimmte Daten auf der Schnittstelle speichern. Zum Beispiel das aktuell ausgewählte Teil.

5.2.1 3D–Baustein

Der 3D–Baustein hat die Aufgabe, 3D–Daten des vom Server geladenen CAD–Modells darzustellen. Dieses wird immer komplett sichtbar dargestellt. Der Benutzer kann per Drag–n–Drop das Modell drehen. Per Scrollen mit dem Mausrad, oder Pinch–to–Zoom bei Touchscreens kann der in das Modell hineinzoomen. Per Klicken auf einen Teil des Modells wird dieses in der Schnittstelle als ausgewähltes Teil gesetzt. In der Titelleiste befindet sich ein Knopf, über welchen die Bausteineinstellungen aufgerufen werden können. In den Einstellungen kann man auswählen, ob immer nur das von der Schnittstelle bereitgestellte, ausgewählte Teil dargestellt wird. Zusätzlich kann man einstellen, ob die aktuell sichtbaren Teile gesperrt werden. Bei einer Aktualisierung des ausgewählten Teils werden die dargestellten Teile nicht verändert. Dies ermöglicht es, mehrere 3D–Bausteine nebeneinander und mit verschiedenen Teilen darzustellen. Als weitere Einstellung gibt es einen Rotieren–Knopf, welcher das aktuelle Modell langsam dreht. Ausgeblendete Teile des Modells werden standardmäßig nicht angezeigt. Man kann in den Bausteineinstellungen die Transparenz dieser verändern. So kann man sehen,

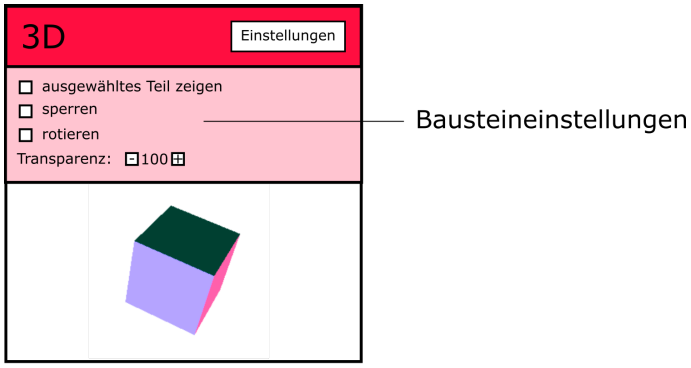

Abbildung 5.5: Konzept–Darstellung des 3D–Bausteins mit ausgeklappten Bausteineinstellungen

wie ein Teil im Gesamtmodell positioniert ist. Eine Konzept–Darstellung des 3D–Bausteins befindet sich in Abbildung 5.5.

5.2.2 Baumstruktur–Baustein

Der Baumstruktur–Baustein hat die Aufgabe die Modellstruktur zu präsentieren. Jedes Teil wird als ein Listeneintrag mit seinem Namen dargestellt. Teile, die zu einer Baugruppe gehören, werden, wenn die Baugruppe aufgeklappt ist, eingerückt dargestellt. Standardmäßig werden alle Knoten eingeklappt dargestellt. Über einen Klick auf einen Indikator, links neben dem Namen einer Baugruppe, werden alle Unterteile und Unterbaugruppen der Baugruppe dargestellt. Ein Klick auf den Namen des Teils oder der Baugruppe wählt das entsprechende Teil in der gemeinsamen Schnittstelle aus. Das aktuell ausgewählte Teil wird eingefärbt.

In der Titelleiste des Bausteins befindet sich ein Knopf über welchen das aktuell ausgewählte Teil oder die Baugruppe aus– oder eingeblendet werden kann. Die entsprechenden Daten

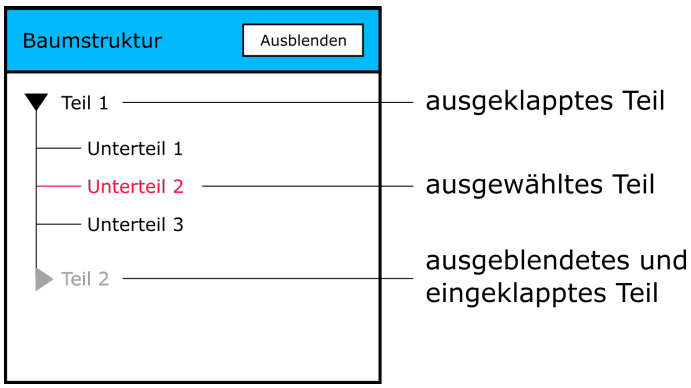

Abbildung 5.6: Konzept–Darstellung des Baumstruktur–Bausteins

werden dann in der Schnittstelle gesetzt. Ausgeblendete Teile oder Baugruppen werden leicht durchsichtig dargestellt. Eine Darstellung des Bausteins befindet sich in Abbildung 5.6.

5.2.3 Informations–Baustein

Der Informations–Baustein beinhaltet Informationen zu Teilen und Baugruppen. Dargestellt werden zum Beispiel Name, Artikelnummer, Material, Anzahl der Unterteile, Kommentare von Konstrukteuren und vieles mehr. Das von dem Baustein präsentierte Teil ist das von der Schnittstelle bereitgestellte ausgewählte Teil.

In der Titelleiste des Bausteins befindet sich ein Knopf über welchen man die Bausteineinstellungen aufrufen kann. In diesen kann der Benutzer einstellen welche Daten er präsentiert haben möchte. Weiterhin ist es möglich das aktuell dargestellte Teil zu sperren. Bei einer Aktualisierung des ausgewählten Teils wird der Baustein nicht aktualisiert. Somit ist es möglich mit mehreren Informations–Bausteinen gleichzeitig Informationen zu verschiedenen Teilen darzustellen.

5.2.4 Suchen–Baustein

Der Suchen–Baustein erlaubt es dem Benutzer nach bestimmten Teilen oder Baugruppen zu suchen. Der Suchbegriff kann in ein Textfeld in der Titelleiste des Bausteins eingegeben werden. In der Titelleiste befindet sich ebenfalls ein Knopf über welchen man die Bausteineinstellungen öffnen kann. Dort lässt sich einstellen nach welchen Kriterien man Filtern möchte. Zur Auswahl stehen zum beispielsweise Teilname oder ID.

Die gefundenen Ergebnisse werden in dem Baustein in einer Liste präsentiert. Durch einen Klick auf ein Teil wird dieses in der gemeinsamen Schnittstelle ausgewählt.

5.3 Schnittstellen

Die Kommunikation der Anwendung mit einem Server geschieht über zwei Schnittstellen. Ein Schnittstelle für die Bereitstellung der Daten und eine weitere für die Bereitstellung der Benutzereinstellungen. Dies erlaubt es einen separaten Server für die Bereitstellung der Daten zu verwenden. Abbildung 5.7 zeigt die Schnittstellenarchitektur des Konzepts.

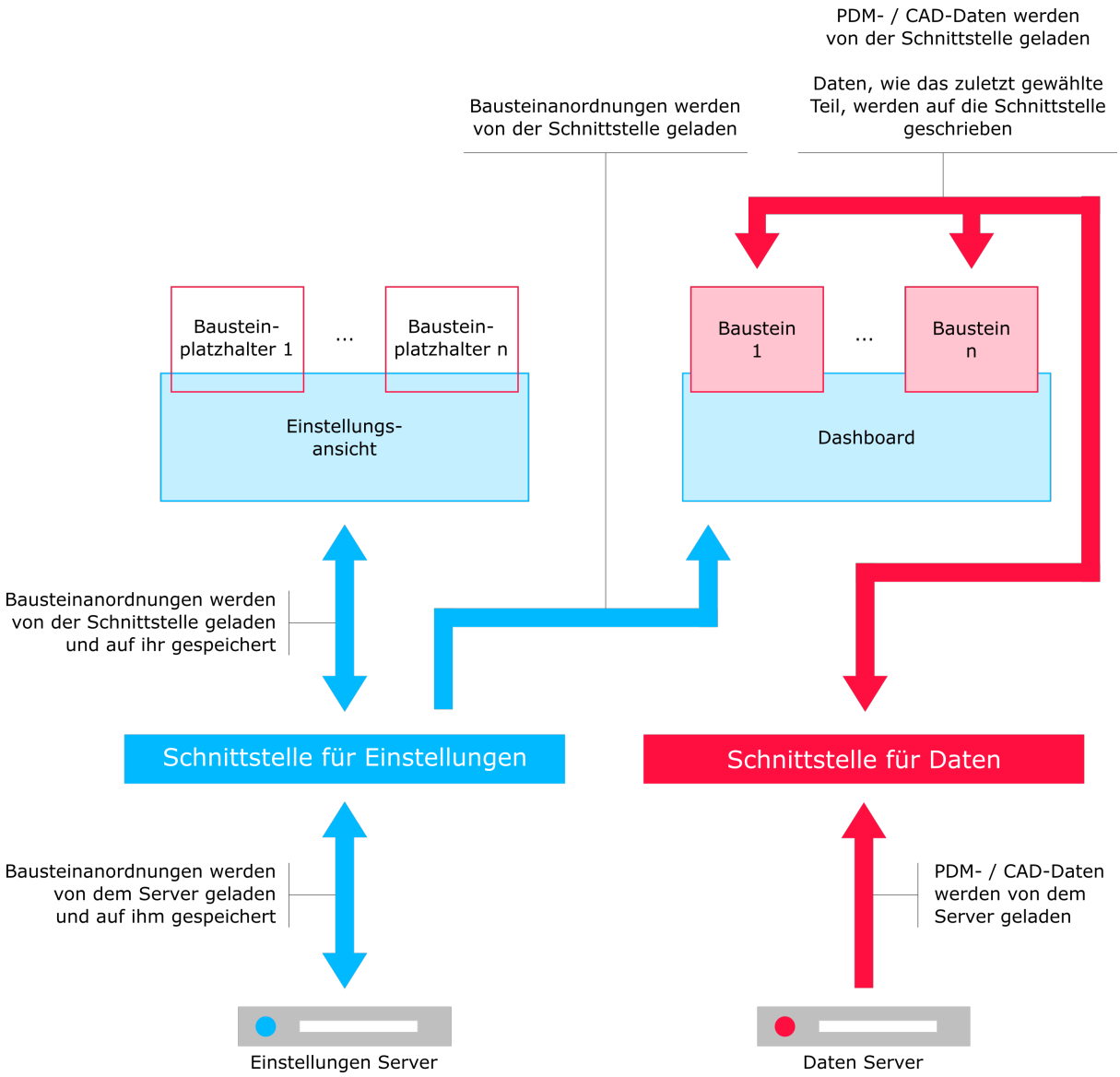

Abbildung 5.7: Schnittstellenarchitektur des Konzepts mit einem Server für die Einstellungen und einem Server für die Daten

5.3.1 Schnittstelle für Daten

Die Datenschnittstelle hat die Aufgabe die CAD–Daten von dem Server abzuholen und den Bausteinen zur Verfügung zu stellen. Zur Verfügung gestellte Daten sind

- Strukturdaten,
- Modelldaten,
- ausgewähltes Teil und

- ausgeblendete Teile.

Die Bausteine können diese dann abonnieren. Werden diese Daten in irgendeiner Form aktualisiert, bekommt jeder Baustein, der diese abonniert hat, die neue Version.

Die Schnittstelle erlaubt es den Bausteinen auch Daten zu verändern. Veränderbare sind hierbei das ausgewählte Teil und die ausgeblendeten Teile. Standardmäßig ist kein Teil ausgewählt und kein Teil ausgeblendet. Wird zum Beispiel ein Teil in dem Baumstruktur–Baustein ausgewählt, wird dieses auf der Schnittstelle gespeichert. Nach dem Speichern wird dieses Teil dann an alle Bausteine gesendet, welche das ausgewählte Teil abonniert haben. Diese können die neuen Daten beliebig handhaben. Ist zum Beispiel der Informations–Baustein gesperrt, ignoriert er diese Aktualisierung. Ist er nicht gesperrt, werden die Informationen des Teils angezeigt.

Sollte ein neuer Baustein in die Ansicht hinzugefügt werden, bekommt dieser alle benötigten Daten, da diese in der Schnittstelle hinterlegt sind.

Die Schnittstelle für Daten ist in Abbildung 5.7 rot dargestellt.

5.3.2 Schnittstelle für Einstellungen

Die Schnittstelle der Einstellungen speichert die aktuellen Einstellungen der Oberfläche jedes Benutzers auf einem Server. Benutzer können sich über die Einstellungsansicht auf diesem anmelden. Meldet sich der Benutzer in der Anwendung an, ruft die Schnittstelle dessen zuletzt gespeicherten Ansichten ab und stellt diese bereit. Somit hat jeder Benutzer auf jedem Endgerät dieselbe Oberfläche und seine personalisierten Bausteinansichten zur Verfügung.

Die Schnittstelle stellt weiterhin die aktuell ausgewählte virtuelle Oberfläche bereit und deren dazugehörende Farben. Die aktuell ausgewählte Oberfläche kann über die Schnittstelle gesetzt werden. Standardmäßig wird die erste virtuelle Oberfläche ausgewählt.

Die Schnittstelle für Einstellungen ist in Abbildung 5.7 blau dargestellt.

6 Implementierung

Dieses Kapitel beschreibt die Prototypimplementierung des Konzepts, das für diese Bachelorarbeit entwickelt wurde.

Der Prototyp wurde als Webanwendung, basierend auf Node.js und Angular 2, implementiert (siehe Kapitel 2). Somit wurde eine komponentenbasierte Implementierung realisiert.

6.1 Daten

Die CAD–Daten sind als eine Baumstruktur in der Klasse `Part` gespeichert. Ein Objekt dieses Typs repräsentiert ein CAD–Teil bestehend aus

- ID,
- Name,
- Dateiname der Modelldaten und
- Unterteile ebenfalls des Typs `Part`.

Das gesamte Modell ist nur ein Objekt dieses Typs als Wurzelknoten. Alle Unterteile können durch propagieren der Unterteilobjekte erreicht werden.

6.2 Schnittstellen

Die Schnittstellen wurden als Angular 2 Service implementiert. Diese Services erlauben es, Daten beliebigen Komponenten zur Verfügung zu stellen. Damit die Komponenten benachrichtigt werden, wenn neue Daten vorhanden sind, wurde eine Implementierung mittels RxJS Subjects gewählt. Dies sind Objekte, die als Observable und als Observer dienen. Somit ist es möglich, auf asynchronem Wege Daten bereitzustellen. Komponenten, die eine dieser Schnittstellen benutzen, können beliebige dieser Subjects importieren und bei neu anliegenden Daten entsprechende Funktionen ausführen lassen.

6.2.1 Datenschnittstelle

Die Datenschnittstelle wurde als Angular 2 Service mit dem Namen `DataService` implementiert. Sie stellt die folgenden Daten zum Abonnement bereit:

- Wurzelteil der CAD–Struktur
- ID des aktuell ausgewählten Teils
- IDs der aktuell ausgeblendeten Teile

Die CAD–Daten werden von dem Backend–Server im JSON–Format geladen und dann in ein `Part`–Objekt deserialisiert. Die ID des ausgewählten Teils wird als `Number` zur Verfügung gestellt und die Liste der IDs der ausgewählten Teile als Liste des Typs `Number`. Die Schnittstelle stellt den Komponenten zwei Methoden bereit. Eine zum Setzen des aktuell ausgewählten Teils und eine zum Aus– / Einblenden von Teilen. Wird eine von beiden aufgerufen, werden die Daten in der Schnittstelle entsprechend gespeichert und an alle Abonnenten weitergegeben.

6.2.2 Einstellungen Schnittstelle

Die Einstellungen Schnittstelle wurde ebenfalls als Angular 2 Service mit dem Namen `SettingsService` implementiert. Zum Abonnement bereitgestellte Daten sind:

- Binärbaume
- aktuell ausgewählter Binärbaum
- Primärfarbe des aktuell ausgewählten Binärbaums
- Sekundärfarbe des aktuell ausgewählten Binärbaums

Die Liste von Binärbäumen wird bei Initialisierung der Anwendung vom Backend–Server geladen. Die Binärbäume werden als Liste von `Node`–Objekten, der aktuell ausgewählte Binärbaum entsprechend als `Node`–Objekt, zur Verfügung gestellt. Die beiden Farben werden als hexadezimaler Text mit Rot–Grün–Blau Werten repräsentiert und werden verwendet um die Darstellung, welche durch einen Binärbaum repräsentiert wird, eindeutig identifizieren zu können. Die Schnittstelle stellt den Abonnenten drei Methoden zur Verfügung. Eine zum Setzen des aktuell ausgewählten Binärbaums, eine zum Hinzufügen von neuen Binärbäumen und eine zum Löschen von Binärbäumen. Die Daten werden bei jeder Aktion auf dem Backend–Server hinterlegt.

6.3 Benutzeroberfläche

Die Hauptkomponente der Gesamtanwendung heißt `AppComponent` und ist die unterste Komponente in Angular 2 Anwendungen. Sie beinhaltet die Titelleiste am oberen Rand und benutzt zwei Unterkomponenten, die `DashboardComponent` und die `SettingsComponent`. Jeweils eine davon wird auf der gesamten Fläche unterhalb der Titelleiste angezeigt. In der Titelleiste befindet sich ein Knopf, mit welchem zwischen den beiden Komponenten gewechselt werden kann.

6.3.1 Baumstruktur

Die Baumstruktur ist der Kern der Benutzeroberfläche. Sie speichert die Bausteinanordnungen als Binärbaum. Jeder Blattknoten stellt dabei einen Baustein dar, in welchem die Art des Bausteins gespeichert wird. Jeder Elternknoten ist ein Platzhalter in welchem Informationen über die Anordnung beiden Kindknoten gespeichert werden. Ein Binärbaum stellt eine virtuelle Oberfläche mit Bausteinen dar. In der Implementierung wurde zwischen den Daten der Binärbaumstruktur und deren Darstellung unterschieden, was eine Wiederverwendung oder Speicherung der vereinfacht. Hierbei wurde das MVC Modell beachtet. Der Zusammenhang zwischen Datenstruktur und Darstellung wird in Abbildung 6.1 veranschaulicht.

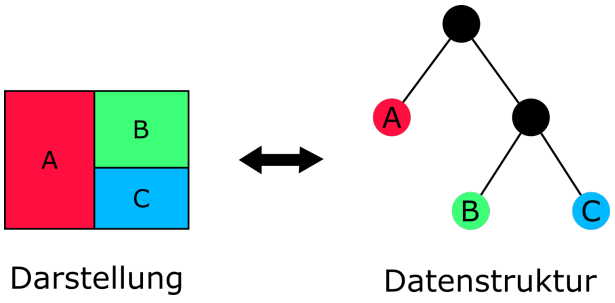

Abbildung 6.1: Zusammenhang zwischen Darstellung und Datenhaltung der Bausteinanordnung

Daten

Realisiert wurde die Baumstruktur mit der Klasse `Node`. Ein Objekt dieses Typs kann sowohl Blattknoten, als auch Elternknoten sein. Ist es ein Blattknoten wird der Bausteintyp gespeichert. Ansonsten werden die Orientierung der Trennung zwischen den Kindknoten (horizontal oder vertikal), deren Teilverhältnis und zwei Objekte ebenfalls der Klasse `Node`, welche die beiden Kindknoten darstellen, gespeichert. Zusätzlich hat ein Knotenobjekt zwei Farben. Eine Primärfarbe und eine Sekundärfarbe. Diese dienen dem Einfärben der Bausteine um die virtuellen Oberflächen leichter identifizieren zu können. Jeder Kindknoten hat die gleichen Farben wie sein Elternknoten. Die Binärbaumstruktur lehnt sich an die in dem Window Manager bspwm verwendete Datenstruktur an [BSP].

Darstellung

Die Darstellung der Baumstruktur wurde als Angular 2 Komponente mit dem Namen `NodeComponent` realisiert. Diese hat die Aufgabe einen Knoten des Baums darzustellen, egal ob es sich dabei um einen Blattknoten oder einen Elternknoten handelt. Wird die Komponente verwendet, muss man ihr mehrere Parameter mitgeben. Diese sind

- das Knotenobjekt,
- x–Position,
- y–Position,
- Breite,
- Höhe,
- Abstand und
- Modifizierbar–Wert.

Das Knotenobjekt enthält die darzustellenden Daten. Diese werden, beginnend mit der x–Position und der y–Position, über die mitgegebene Breite und Höhe abgebildet. Ist dies ein Blattknoten, wird die Komponente `ModuleComponent` über die gesamte Fläche dargestellt. Dies ist eine Platzhalter–Komponente, welche sich über die gegebene Fläche aufspannt und die einzelnen Bausteinkomponenten, je nach Typ, darstellt. Ist das Knotenobjekt ein Elternknoten, werden zwei Unterkomponenten ebenfalls des Typs `NodeComponent` angezeigt. Das Knotenobjekt beinhaltet die Informationen in welchem Teilverhältnis und in welcher Richtung diese dargestellt werden sollen. Die Parameter, welche den beiden Unterkomponenten mitgegeben werden, werden dabei automatisch aus den eigenen berechnet. Ebenso wird der Abstand zwischen den beiden Knoten mit ein berechnet.

Wird der Knotenkomponente ein „Modifizierbar“ mitgegeben, lässt sich die Knotenanordnung verändern. Zwischen den beiden Kindknoten eines Elternknotens wird dann ein Trennelement angezeigt. Durch ein Verschieben von diesem per Drag–n–Drop fängt die Komponentenlogik die Koordinaten ab und berechnet daraus das neue Teilverhältnis der Kindknoten. Damit wird die **Funktion 1** (***drag***) aus Kapitel 5.1.2 implementiert. Mit einem Klick auf das Trennelement wird die Orientierung der Teilung geändert, was der **Funktion 2** (***toggle***) aus Kapitel 5.1.2 entspricht. Durch einen Rechtsklick oder ein Halten bei Touchscreens werden die beiden Kindknoten getauscht. Somit wird die **Funktion 3** (***flip***) aus Kapitel 5.1.2 realisiert. Zusätzlich werden, wenn die Knotenstruktur modifizierbar ist, keine Bausteine angezeigt, sondern nur deren Namen. Dies dient der Übersichtlichkeit beim Arrangieren. Über jedem Baustein–Platzhalter wird ein Löschen–Symbol angezeigt. Wird dies betätigt, macht die Komponentenlogik aus sich selbst ein Kindknoten indem es die beiden Kindknoten löscht und sich selbst den Modultyp des anderen gibt. Dies entspricht der **Funktion 4** (***delete***) aus Kapitel 5.1.2. Weiterhin sind die Bausteinplatzhalter mit einem Drop–Event ausgestattet. Wird auf ihnen irgendein HTML–Element fallengelassen, wird aus den Event–Daten ausgelesen ob es sich um ein Modul handelt.

Dieses wird dann entsprechend hinzugefügt, indem sich der Kindknoten aufspaltet und der rechte Unterknoten den neuen Baustein bildet. Die in Kapitel 5.1.2 beschriebene **Funktion 5** (***insert***) wird so implementiert. Das Teilverhältnis ist dabei standardmäßig die Hälfte und die Orientierung Vertikal.

6.3.2 Dashboard

Die Dashboard Komponente dient der Anzeige der Bausteine. Sie hat den Namen `DashboardComponent` und spannt sich über den gesamten Bereich unterhalb der Titelleiste der `AppComponent` auf. Sie hat von dem `SettingsService` die Liste von Binärbäumen und den aktuell ausgewählten Binärbaum abonniert.

Am linken Bildschirmrand wird eine schmale Indikatorleiste angezeigt. Für jeden Binärbaum in der Liste existiert ein in der Primärfarbe des Baums eingefärbtes Quadrat. Ein Klick auf dieses setzt den aktuell ausgewählten Baum entsprechend in der Schnittstelle. Zusätzlich sind die Quadrate nummeriert.

Rechts von der Indikatorleiste spannen sich für jeden Binärbaum in der Liste mehrere `NodeComponents` auf. Mittels CSS ist immer nur der aktuell ausgewählte Binärbaum sichtbar. Die anderen werden ausgeblendet. Ihnen wird kein Modifizieren–Wert mitgegeben. Somit

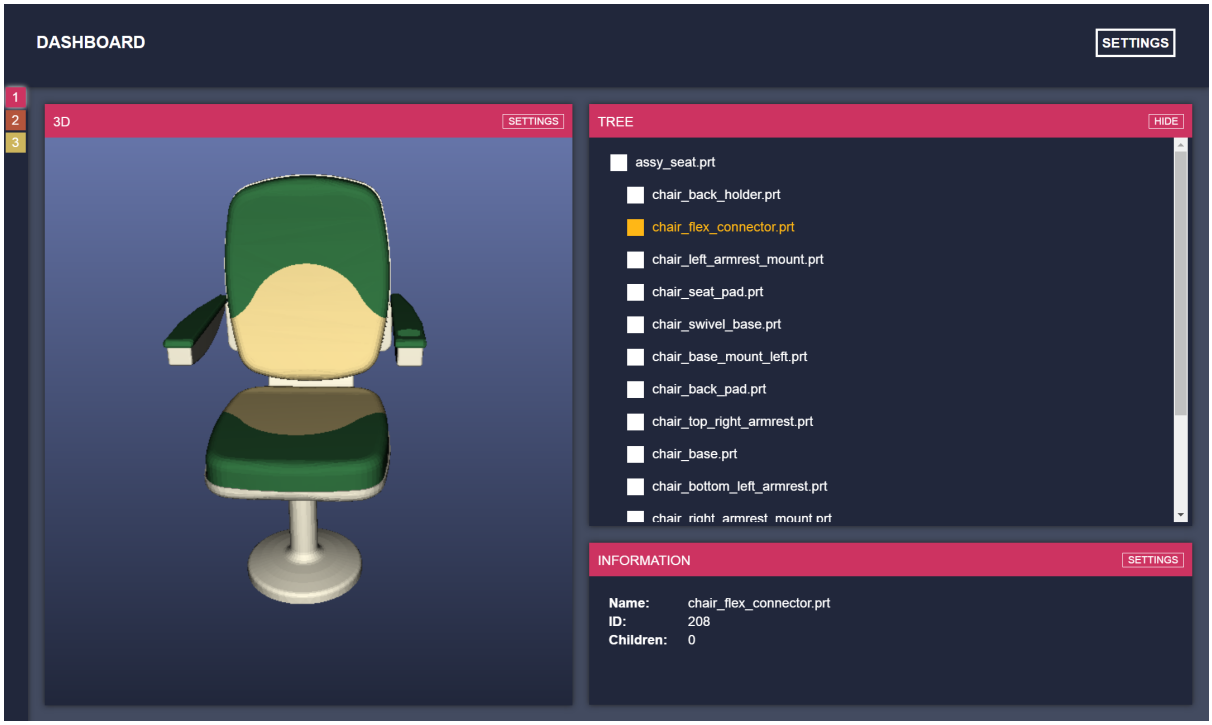

Abbildung 6.2: Dashboard Ansicht mit drei Bausteinen und drei virtuellen Oberflächen

lässt sich die Anordnung nicht verändern und die Bausteine werden angezeigt. Somit wurden virtuelle Oberflächen (siehe Kapitel 2) realisiert.

Ein Screenshot des Dashboards aus der Prototypimplementierung befindet sich in Abbildung 6.2.

6.3.3 Einstellungen

Die Einstellungen Komponente, `SettingsComponent` funktioniert ähnlich wie die `DashboardComponent`. Vom `SettingsService` wurden ebenfalls die Binärbaume und der aktuell ausgewählte Binärbaum abonniert.

Auf der linken Seite befindet sich wieder eine Indikatorleiste. Diese wurde um zwei Knöpfe erweitert. Unter den Quadraten für die einzelnen Binärbäume gibt es ein $+$−Zeichen. Mit einem Klick auf dieses wird eine Funktion in dem `SettingsService` aufgerufen, welche einen leeren Binärbaum an das Ende der Liste hinzufügt. Dieser wird von dem `SettingsService` automatisch als ausgewählter Binärbaum gesetzt und somit fokussiert. Ganz unten in der Indikatorleiste befindet sich ein rotes $\times$−Symbol. Ein Klick auf dieses teilt dem `SettingsService` mit, dass der aktuell ausgewählte Binärbaum gelöscht werden soll. Befinden sich leere Binärbäume in

Abbildung 6.3: Einstellungen Ansicht mit drei modifizierbaren Bausteinen und drei virtuellen Oberflächen

der Liste, werden diese automatisch gelöscht. So verschwinden leere virtuelle Oberflächen automatisch.

In der Mitte wird, wie bei der `DashboardComponent`, ebenfalls eine `NodeComponent` angezeigt, welche als Eingabebinärbaum den aktuell ausgewählten bekommt. Diesem wird ein Modifizieren–Wert mitgegeben. Die Bausteine in dem aktuellen Binärbaum können folglich verändert und arrangiert werden.

Auf der rechten Seite der Einstellungen Komponente befindet sich eine vertikale Liste mit den verfügbaren Bausteinen. Diese werden als Rechtecke mit deren Namen in der Mitte angezeigt. Werden diese per Drag–n–Drop bewegt, fügt die Komponente dem Event den Typ des entsprechenden Bausteins hinzu. Wird ein Rechteck auf einem Baustein in dem aktuell ausgewählten Binärbaum fallengelassen, erstellt dieser, wie oben beschrieben, ein neuen Kindknoten aus dem Bausteintyp.

Die Einstellungen aus der Prototypimplementierung werden in Abbildung 6.3 gezeigt.

6.4 Bausteine

Die einzelnen Bausteine wurden in der Implementierung als einzelne Angular 2 Komponenten angelegt. Die Platzhalterkomponente `ModuleComponent`, welche in der `NodeComponent` für jeden Baustein angezeigt wird, bekommt als Eingabeparameter einen Bausteintyp. Sie benutzt alle in diesem Abschnitt folgenden Bausteinkomponenten und zeigt diese, entsprechend dem Bausteintyp, in voller Größe innerhalb der Platzhalterkomponente an.

6.4.1 3D–Baustein

Der 3D–Baustein wurde als Angular 2 Komponente mit dem Namen `ThreeDModuleComponend` angelegt. Am oberen Rand des Bausteins wird eine Titelleiste in der Primärfarbe des aktuellen Binärbaums angezeigt. Diese wurde von dem `SettingsService` abonniert. In der Titelleiste befindet sich ein Einstellungen–Knopf. Wird dieser gedrückt, erscheint ein kleines Einstellungsmenü unterhalb der Titelleiste in der Sekundärfarbe des Binärbaums. Dort finden sich folgende Einstellungsmöglichkeiten:

- Ausgewähltes Teil anzeigen
- Sperren
- Rotieren
- Sichtbarkeit ausgeblendeter Teile

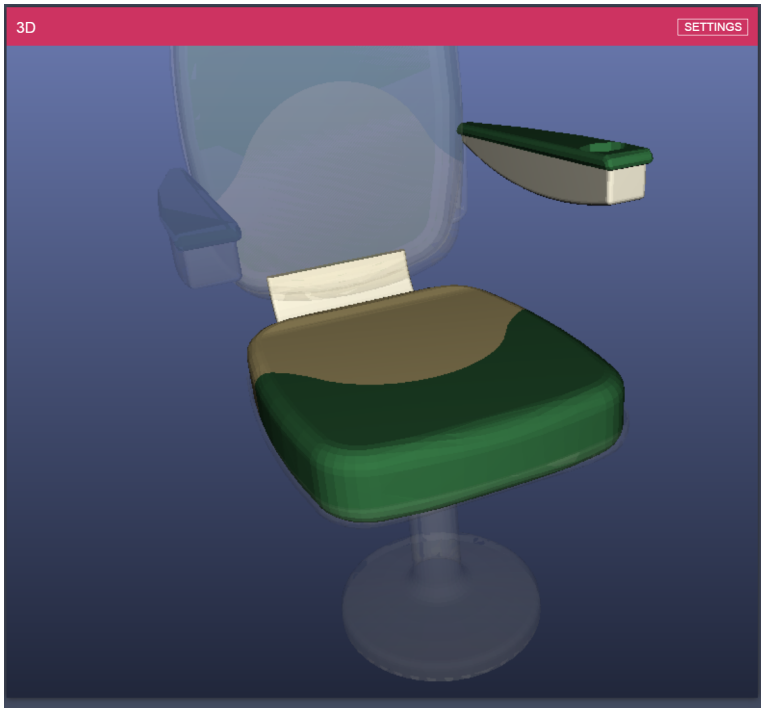

Abbildung 6.4: 3D–Baustein mit mehreren ausgeblendeten Teilen, welche teilweise transparent sind

Die Einstellungen werden in der Komponente gespeichert und an eine Unterkomponente `ThreeDContentComponent` weitergegeben. Diese wird unterhalb der Titelleiste angezeigt und hat die Aufgabe, das 3D–Modell darzustellen. Über den `DataService` bekommt die Komponente die Datenstruktur des Modells, das aktuell ausgewählte Teil und die ausgeblendeten Teile. Die 3D–Daten werden, ausgehend von dem Dateinamen eines Teils, von dem Server geladen. Anschließend werden diese mit dem JavaScript–Framework Three.js in WebGL dargestellt. Standardmäßig wird das ganze Modell gezeigt, wobei ausgeblendete Teile transparent sind. Wurde die Einstellung gewählt, dass nur das ausgewählte Teil angezeigt werden soll, aktualisiert die Komponente das 3D–Modell jedes mal wenn vom `DataService` ein neues ausgewähltes Teil mitgeteilt wird. Dieses wird dann mit allen Unterteilen angezeigt. Ist die Sperren–Einstellung gewählt, ignoriert die Komponente die Aktualisierung und behält die aktuelle Ansicht bei. Bei der Rotieren–Einstellung wird das Modell langsam gedreht. Die Sichtbarkeit ausgeblendeter Teile verändert deren Transparenz von 0 bis 100. Standardmäßig ist der Wert 0 eingestellt.

Das Modell kann mit intuitiven Touch– oder Mausgesten bewegt werden. Dies wurde ebenfalls mit Three.js implementiert. Per Pinch–to–Zoom oder mit Scrollen kann das Modell vergrößert oder verkleinert werden. Mit Swipen oder Drag–n–Drop kann das Modell gedreht werden.

Durch die Kombination der Einstellungen und den Bewegungsmöglichkeiten lässt sich jede beliebige Ansicht des Modells erreichen. Da Teile in der Ansicht gesperrt werden können, ist

es auch möglich, mehrere verschiedene 3D–Modelle in mehreren Bausteinen anzuzeigen und zu vergleichen.

Das Auswählen von Teilen innerhalb des 3D–Bausteins wurde mittels Raycasting implementiert. Bei einem Klick auf das 3D–Objekt wird ein Strahl, ausgehend von der Klickposition, durch das Objekt gelegt. Das erste dadurch getroffene Teil wird in dem `DataService` als aktuell Ausgewähltes gesetzt sofern dies nicht eines der ausgeblendeten Teile ist. Wurde in dem Baustein das Anzeigen von aktuell ausgewählten Teilen gesetzt, wird die Aktualisierung des `SettingsService` ignoriert, da ansonsten das Modell verschwindet und nur das angeklickte Objekt übrig bleibt.

In Abbildung 6.4 ist ein Screenshot des 3D–Bausteins aus der Prototypimplementierung zu sehen.

6.4.2 Baumstruktur Baustein

Der Baumstruktur Baustein wurde als Angular 2 Komponente mit dem Namen `TreeModuleComponent` implementiert. Er benutzt den `DataService` um die Modelldaten und das aktuell ausgewählte Teil und den `SettingsService` um die Farben des aktuellen Binärbaumes zu bekommen.

An dem oberen Rand des Bausteins befindet sich, wie auch beim 3D Baustein eine Titelleiste. Diese wird ebenfalls in der Primärfarbe des Binärbaums eingefärbt. In der Titelleiste ist ein Knopf positioniert über welchen das aktuell ausgewählte Teil ausgeblendet werden kann. Bei einem Klick auf den Knopf ruft die Bausteinlogik die Methode aus dem `DataService` auf um das aktuell ausgewählte Teil auszublenden.

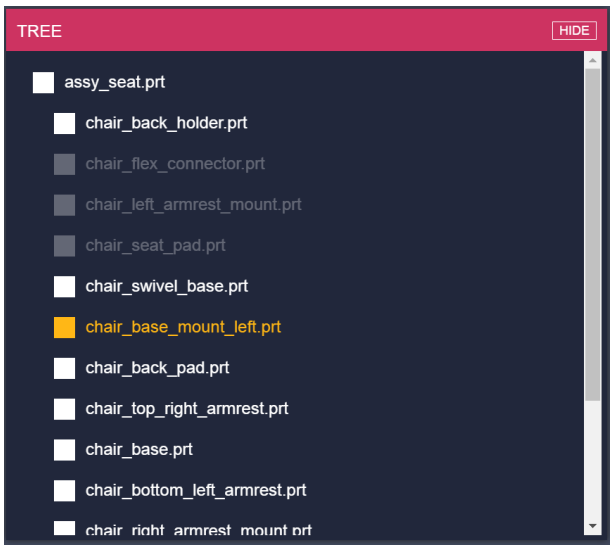

Abbildung 6.5: Baumstruktur–Baustein mit ausgeklapptem Teil, mehreren ausgeblendeten Unterteilen und ausgewähltem Unterteil

Die Darstellung der einzelnen Teile innerhalb des Bausteins geschieht mittels einer anderen Angular 2 Komponente namens `TreeModuleEntryComponent`. Diese hat die Aufgabe genau ein Teil darzustellen. Dieses muss der Komponente als Eingabeparameter mitgegeben werden. Mittels den Daten aus dem `DataService` werden die CSS–Attribute für die Darstellung gesetzt. Ist das anzuzeigende Teil das aktuell ausgewählte Teil oder ist es ausgeblendet, wird es entsprechend eingefärbt. An der linken Seite des Teils befindet sich ein Indikator über welchen die Unterteile des Teils aufgeklappt werden können. Standardmäßig sind diese eingeklappt. Werden diese aufgeklappt, werden unterhalb des Eintrags, etwas eingerückt, alle Unterteile dargestellt. Angezeigt werden diese jeweils wieder als eigene `TreeModuleEntryComponent`. Bei einem Klick auf den Teilnamen setzt die Komponentenlogik das Teil, welches sie darstellt, in dem `DataService` als ausgewählt.

Ein Screenshot des Baumstruktur–Bausteins ist in Abbildung 6.5 zu sehen.

6.4.3 Informations–Baustein

Der Informations–Baustein wurde als Angular 2 Komponente mit dem Namen `InformationModuleComponent` implementiert. Er abonniert vom `DataService` das aktuell ausgewählte Teil und vom `SettingsService` die Binärbaumfarben.

Am oberen Rand des Bausteins befindet sich wieder eine Titelleiste in der Primärfarbe des Bausteins. Ein Einstellungen Knopf befindet sich auf der rechten Seite in der Titelleiste. Wird dieser betätigt, öffnet sich unterhalb der Titelleiste, in der Sekundärfarbe des Binärbaums, ein Einstellungsmenü. Dieses enthält die Funktion Sperren über welche das aktuell ausgewählte Teil nicht mehr aktualisiert wird.

Unterhalb der Titelleiste befindet sich eine HTML–Tabelle in welcher die Informationen

- Teil ID,
- Teilname und
- Anzahl der Kinder

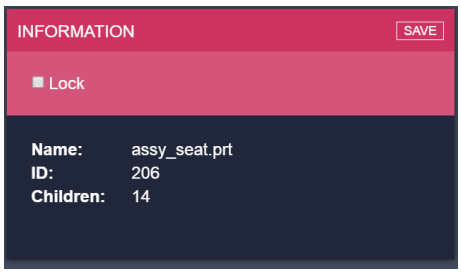

Abbildung 6.6: Informations–Baustein mit ausgeblendeten Einstellungen und aktuell ausgewähltem Teil

dargestellt werden. Standardmäßig werden die Informationen des aktuell ausgewählten Teils angezeigt. Sind mehrere Bausteine vorhanden ist es mit Hilfe des Sperrens von Bausteinen so möglich, Informationen zu mehreren Teilen gleichzeitig anzeigen zu lassen.

Ein Informations–Baustein mit ausgeklappten Einstellungen ist in Abbildung 6.6 abgebildet.

6.4.4 Suchen–Baustein

Der Suchen–Baustein wurde als Angular 2 Komponente mit dem Namen `SearchModuleComponent` implementiert. Er abonniert von dem `DataService` alle Teile. Diese werden innerhalb des Bausteins als eine lineare Liste gespeichert. Wie bei den anderen Bausteinen werden hier ebenfalls die Farben des aktuellen Binärbaums importiert.

Der Baustein besitzt eine Titelleiste in welcher sich ein Eingabefeld befindet. Über dieses kann nach Teilnamen gesucht werden. Bei einer Veränderung des Strings in dem Textfeld, wird die Liste aller Teile nach passenden Teilen durchsucht. Zu dem Suchtext passende Teile werden in einer separaten Liste gespeichert. Wird das Textfeld geleert, wird auch die Liste aller Treffer geleert.

Unterhalb der Titelleiste wird die Liste der gefundenen Teile angezeigt. Bei einem Klick auf ein Teil wird dieses durch die Bausteinlogik in dem `DataService` als ausgewähltes Teil gesetzt.

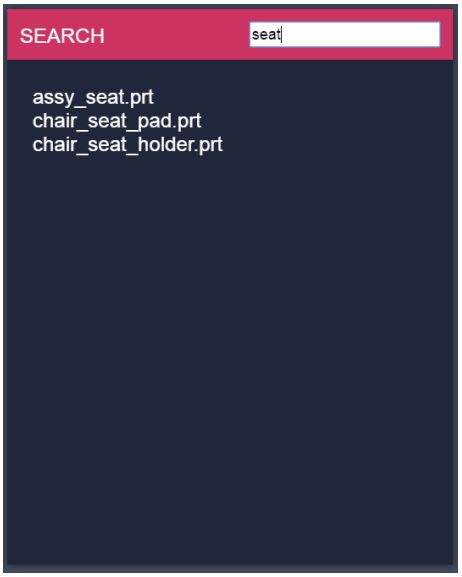

Abbildung 6.7: Suchen–Baustein mit gefilterten Teilen

6.5 Backend

Das Backend wurde als kleiner Java Tomcat Server mithilfe des Spring Frameworks[1] implementiert. Die gesamte Kommunikation findet über HTTP statt. Dabei werden nur JSON–Dateien ausgetauscht. Daten können mit der HTTP GET–Methode abgeholt werden und mittels HTTP POST auf dem Server gespeichert werden. Lokal werden die Daten als JSON–Dateien auf dem Dateisystem abgelegt. Die beiden Controller, welche die Daten und die Einstellungen bereitstellen, wurden separat implementiert. Somit ist es leicht möglich diese auf unterschiedlichen Rechnern laufen zu lassen.

[1] https://projects.spring.io/spring-framework/

7 Evaluierung

Das Ziel der Evaluierung ist es zu zeigen, dass die in Kapitel 4 genannten Anforderungen mithilfe des erarbeiteten Konzepts erfüllt werden.

7.1 Verschiedene Geschäftsfelder

Durch die Anordnung von verschiedenen Bausteinen ist das Konzept an beliebige Geschäftsfelder im Engineering–Bereich anpassbar. Die Bausteine sind dabei nicht auf ein Anwendungsgebiet ausgelegt, sondern auf die Darstellung von bestimmten Informationen. Je nachdem welche Informationen benötigt werden, können so Lageristen, Monteure, Konstrukteure und Manager ihre eigenen Oberflächen zusammenstellen, um die gewünschten Daten präsentiert zu bekommen. Durch mehrere virtuelle Oberflächen ist es auch möglich, schnell zwischen verschiedenen Geschäftsfeldern zu wechseln.

7.2 Endgeräte

Mittels moderner Web–Technologien wie Angular 2, Node.js und WebGL ist es möglich die Anwendung auf beliebigen Endgeräten zu verwenden. Der Einsatz wird noch flexibler, da bereits genutzte Endgeräte weiterhin verwendet werden können. Dies gilt auch für mobile Endgeräte. Somit hat der Nutzer die Informationen überall zur Hand. Zusätzlich kann er die Anwendung mit den gewohnten Gesten wie Swipen oder Pinch–to–Zoom bedienen. Mithilfe eines Backend–Servers werden die Ansichtseinstellungen automatisch gespeichert. Der Nutzer hat somit auf all seinen Endgeräten Zugriff auf die Informationen, die er benötigt.

Durch das Prinzip der Anordnung von Bausteinen, ähnlich dem des Tiling Window Managements, wird die Bildschirmgröße immer optimal ausgenutzt. Auf größeren Bildschirmen können problemlos viele Bausteine nebeneinander platziert werden. Auf kleineren Bildschirmen hilft das Prinzip der virtuellen Oberflächen, die Übersicht nicht zu verlieren.

7.3 Bedienung

Durch die präzise Beschriftung der Buttons (z.B. *Settings* auf dem Einstellungsknopf) und der Bausteineinstellungen ist die Funktionalität dieser eindeutig identifizierbar. Durch die Implementierung als Webapplikation sind sowohl die Verwendung mit Tastatur und Maus, als auch mit Touchscreens möglich. Die Bausteinanordnung funktioniert mittels Drag–n–Drop, was den meisten Nutzern bereits aus gängigen Desktop–Umgebungen bekannt ist.

7.4 Bausteine

Das Anordnungsprinzip der Bausteine ermöglicht es, dass diese immer so groß wie möglich dargestellt werden und sich nicht überlappen. Die in dem Einstellungsmenü vorhandene Möglichkeit, die Größe der Bausteine zu verändern, ist durch die Verwendung von Trennelementen gelöst. Durch das Verwenden einer Schnittstelle, über die die Bausteine kommunizieren, stehen immer aktuelle Informationen zur Verfügung, da diese automatisch synchronisiert werden.

7.5 Raster

Die Darstellung eines Rasters wurde durch eine virtuelle Oberfläche gelöst, auf welcher Bausteine platziert werden können. Durch die Einfärbung und Nummerierung der virtuellen Oberflächen sind diese leicht zu identifizieren. Die Einfärbung der Bausteine in der aktuellen Oberflächenfarbe erleichtert dies zusätzlich.

In dem separaten Einstellungsmenü liegt der Augenmerk ausschließlich auf der Anordnung von Bausteinen. Hier kann der Nutzer problemlos und intuitiv Bausteine anordnen, hinzufügen und löschen. Da nur Bausteinplatzhalter angezeigt werden, ist es möglich Bausteine leicht zu identifizieren und übersichtlich zu präsentieren. In der Dashboard–Ansicht dagegen kann die Bausteinanordnung nicht mehr verändert werden. Dies verhindert ein versehentliches Umordnen von Bausteinen, was den Benutzer von den Informationen ablenken könnte.

7.6 Design

Das Design der Benutzeroberfläche in der Prototypimplementierung ist flach und modern gestaltet (Siehe Abbildungen in Kapitel 6). Es wurden dunkle Farben gewählt um den Benutzer nicht von den eigentlichen Informationen abzulenken. Helle Akzentfarben untermalen wichtige Informationen, wie zum Beispiel das aktuell ausgewählte Teil. Durch einen hohen Kontrast ist der Text auch bei Sonneneinstrahlung gut lesbar.

7.7 Use–Cases

Im Folgenden werden Use–Cases behandelt, welche die vier Engineering–Felder Lager, Werkstatt, Konstruktion und Management abdecken.

7.7.1 Lager

Ein Lagerist muss dringend ein Teil finden, zu welchem er nur die ersten vier Ziffern einer sechsstelligen Teilnummer bekommen hat. Er weiß, dass es sich um eine Armlehne eines Sitzes handeln muss.

Um nicht unnötig lange nach dem Teil suchen zu müssen, zückt er sein Smartphone und loggt sich in der Konzeptanwendung auf dem Firmenserver ein. Er erstellt eine neue virtuelle Oberfläche bestehend aus einem Suchbaustein und einem 3D–Baustein. Nachdem er die vier gegebenen Ziffern in den Suchbaustein eingegeben hat, geht er systematisch die einzelnen gefundenen Teile durch und überprüft das Teilmodell. Innerhalb kürzester Zeit hat er die Armlehne gefunden.

Die Bausteinanordnung ist in Abbildung 7.1 dargestellt.

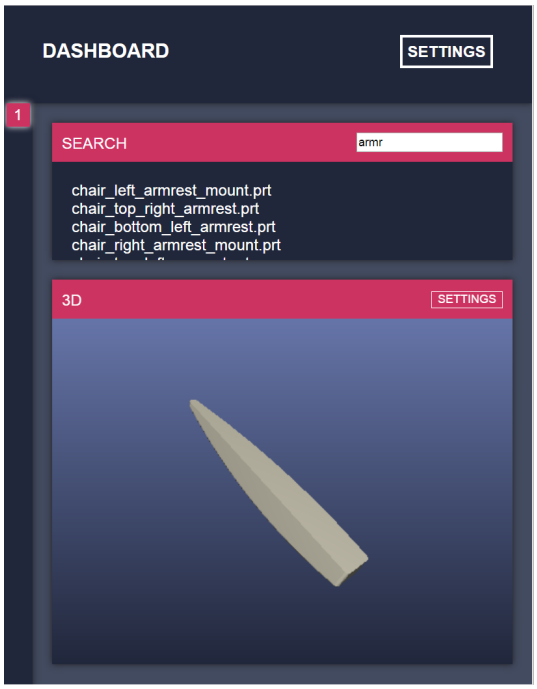

Abbildung 7.1: Bausteinanordnung bestehend aus einem Suchbaustein und einem 3D–Baustein auf einem kleinen Bildschirm

7.7.2 Werkstatt

Ein Monteur muss ein neues Teil in einen Sitz einbauen. Er ist sich aber unschlüssig, in welcher Richtung das Teil eingebaut werden soll.

Deshalb öffnet er auf dem Werkstatt–Rechner die Konzeptanwendung und loggt sich auf dem Firmenserver ein. Er erstellt eine neue virtuelle Oberfläche, bestehend aus einem Suchbaustein, einem Baumstruktur–Baustein und einem 3D–Baustein. In dem Eintellungsmenü des 3D–Bausteins ändert er die Transparenz der ausgeblendeten Teile auf 80%. Er gibt die Teilnummer in das Suchfeld ein und wählt das gefundene Teil aus. In dem Baumstruktur–Baustein blendet er alle Teile außer dem Ausgewählten aus. Die Darstellung der drei Bausteine auf dem kleinen Werkstattbildschirm ist allerdings nicht gut sichtbar. Deswegen öffnet er das Einstellungsmenü und löscht den 3D–Baustein. Er erstellt eine neue virtuelle Oberfläche nur bestehend aus einem 3D–Baustein. Nun ist auch aus größerer Distanz gut erkennbar wie das Teil einzubauen ist.

Eine Darstellung von der Bausteinanordnung, bevor der 3D–Baustein gelöscht wird, befindet sich in Abbildung 7.2.

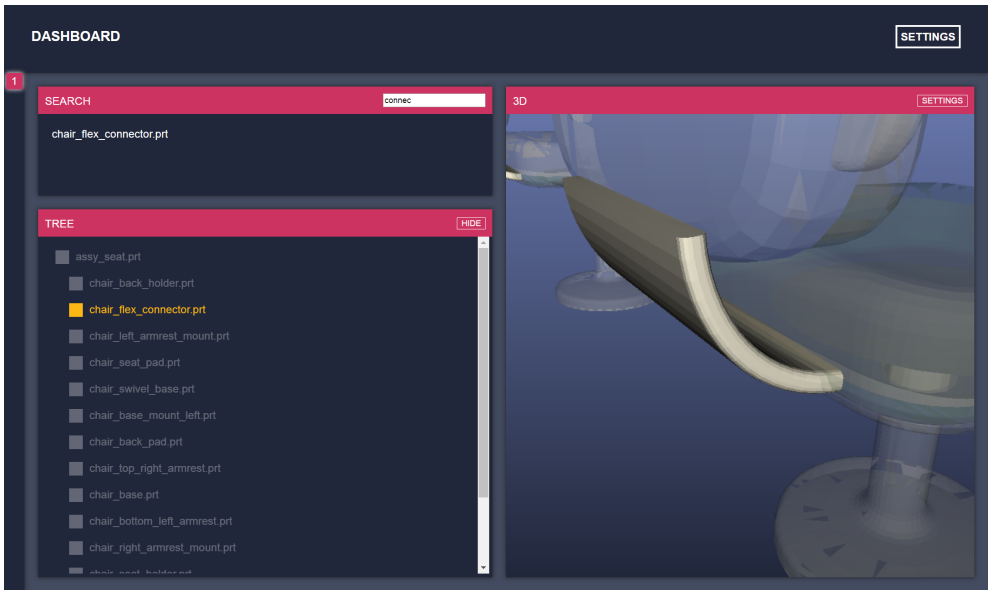

Abbildung 7.2: Bausteinanordnung bestehend aus einem Suchbaustein, einem Baumstruktur–Baustein und einem 3D–Baustein mit mehreren ausgeblendeten Teilen

7.7.3 Konstruktion

Ein Konstrukteur soll drei Teile miteinander vergleichen. Jedes mal wenn er in seinem CAD–Programm ein Teil anklickt, werden nur Informationen über dieses angezeigt.

Er öffnet die Konzeptanwendung auf seinem Desktop–PC und loggt sich auf dem Firmenserver ein. Daraufhin erstellt er eine neue virtuelle Oberfläche bestehend aus einem Suchbaustein, drei 3D–Bausteinen und drei Informations–Bausteinen. Die 3D–Bausteine stellt er allesamt so ein, dass sie nur das aktuell ausgewählte Teil anzeigen. Er gibt die Teilnummer des ersten Teils in die Suchleiste ein und wählt es aus. Alle 3D– und Informations–Bausteine zeigen nun das ausgewählte Teil an. Er sperrt einen 3D–Baustein und einen Informations–Baustein. Für die anderen beiden Teile wiederholt er diese Schritte. Da er einen großen Bildschirm hat, passen alle Bausteine problemlos auf eine virtuelle Oberfläche und er kann die einzelnen Teile vergleichen.

Ein Screenshot der Bausteinanordnung befindet sich in Abbildung 7.3.

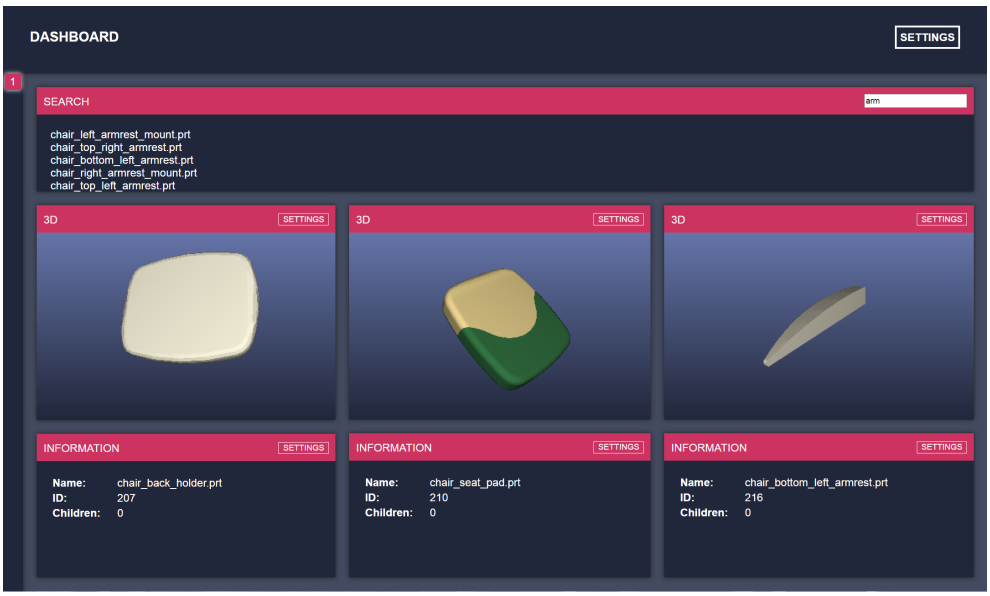

Abbildung 7.3: Bausteinanordnung bestehend aus einem Suchbaustein, drei 3D–Bausteinen und drei Informations–Bausteinen auf einem großen Bildschirm

7.7.4 Management

Ein Manager soll ein neues Produkt bei einem Kundentreffen vorstellen. Der Kunde hat besonderes Interesse an den Struktur– und Modelldaten des Produkts. Wie gewohnt bereitet er sich auf das Treffen vor.

Er öffnet die Konzeptanwendung auf seinem Arbeitsplatz–Rechner und loggt sich auf dem Firmenserver ein. Er erstellt drei neue virtuelle Oberflächen. Eine bestehend aus dem Baumstruktur–Baustein, eine bestehend aus dem 3D–Baustein und eine auf welcher beide Bausteine positioniert sind. Da das Treffen nicht bei ihm im Büro stattfindet, sondern im Firmensitz des Kunden, schaltet er den Rechner aus und macht sich auf den Weg. Beim Kunden angekommen holt er sein Tablet heraus und loggt sich erneut ein. Die Anordnung der

Ansichten wurden automatisch auf dem Server hinterlegt. Er muss somit nicht alles erneut konfigurieren. Nun kann er das Produkt mit den gewünschten Daten vorstellen.

Abbildung 7.4 zeigt die virtuelle Oberfläche auf welcher sowohl Baumstruktur–Baustein, als auch 3D–Baustein platziert sind.

Abbildung 7.4: Bausteinanordnung bestehend aus einem Baumstruktur–Baustein und einem 3D–Bausteinen auf der dritten virtuellen Oberfläche

8 Zusammenfassung und Ausblick

Das Ziel dieser Bachelorarbeit war es, ein bausteinbasiertes Konzept für Engineering Web Apps zu entwickeln. Dafür wurden diverse Oberflächenkonzepte behandelt und verglichen. Es wurde der Zusammenhang zwischen einzelnen Anwendungen und kompletten Betriebssystemoberflächen aufgezeigt. Weiterhin wurden moderne Web–Technologien vorgestellt, welche es erlauben interaktive Webanwendungen zu bauen. Betrachtet wurden HTML, CSS, JavaScript und diverse Frameworks wie Angular 2 und Node.js.

Es wurden Anforderungen an verschiedene Endgeräte und Geschäftsfelder im Engineering–Bereich gestellt, welche es umzusetzen galt. Bei den Endgeräten lag der Fokus auf verschiedenen Bildschirmgrößen und der Touch–Unterstützung für mobile Endgeräte. Die verschiedenen Geschäftsfelder wurden im Hinblick auf die Daten, welche in ihnen von Interesse sind, betrachtet.

Durch das entwickelte Konzept kann der Nutzer seine eigene Oberfläche aus Bausteinen zusammensetzten um genau die Informationen präsentiert zu bekommen, welche er benötigt. Bausteine sind dabei kleine Unterprogramme, welche auf eine Art der Informationsdarstellung optimiert sind. Diese werden dabei in einer Art Raster, ähnlich dem Tiling Window Management, angeordnet um den Platz auf allen Bildschirmgrößen optimal auszunutzen. Durch moderne Web–Technologien ist es möglich die Anordnungen auf beliebig vielen Endgeräten zu verwenden.

Basierend auf diesem Konzept wurde ein Prototyp entwickelt. Dieser wurde mithilfe von Angular 2 und Node.js erstellt und ist auf gängigen Browsern ausführbar.

Das Konzept wurde durch diverse Use–Cases evaluiert und stellt eine neue Möglichkeit der Darstellung von PDM– und CAD–Daten dar. Es ist an beliebige Geschäftsfelder im Engineering–Bereich anpassbar.

Die Anforderungen an das Konzept könnten in Zukunft mit Benutzerstudien verfeinert werden. Diese könnten in verschieden Geschäftsfeldern durchgeführt werden um genauere Anforderungen an Daten und Endgeräte zu erheben.

Das Konzept könnte im Hinblick auf mehrere Bildschirme erweitert werden. Dies würde es ermöglichen Bausteine auf einem Tablet und einem Beamer parallel anzeigen zu lassen. Dies könnte in der Werkstatt oder im Management von Bedeutung sein.

Literaturverzeichnis

[AFB] *Angular - Features & Benefits*. URL: https://angular.io/features.html (zitiert auf S. 14).

[And14] D. Anderson. „How Node.js can accelerate Enterprise application development“. In: (2014) (zitiert auf S. 13).

[ASU] *Actify SpinFire Ultimate*. URL: http://www.actifyeurope.de/cad-viewer/ (zitiert auf S. 19).

[ASW] *Actify SpinFire Web*. URL: http://www.actifyeurope.de/3d-viewer/ (zitiert auf S. 20).

[BAT14] G. Bierman, M. Abadi, M. Torgersen. „Understanding typescript“. In: *European Conference on Object-Oriented Programming*. Springer. 2014, S. 257–281 (zitiert auf S. 13).

[BSP] *bspwm*. URL: https://github.com/baskerville/bspwm (zitiert auf S. 23, 25, 43).

[CSA] *CSS3 Animations*. URL: http://www.w3schools.com/css/css3_animations.asp (zitiert auf S. 13).

[CST] *CSS3 Transitions*. URL: http://www.w3schools.com/css/css3_transitions.asp (zitiert auf S. 13).

[CST3] *CSS3 3D Transforms*. URL: http://www.w3schools.com/css/css3_3dtransforms.asp (zitiert auf S. 13).

[ECS] *Engineering Client Smaragd*. URL: https://academy.mbtech-group.com/mb-tech/catalog/INTERNET/PLM+%26+IT/~/browsecatalog.jsp?K=INTERNET&itemId=AAAAAGX (zitiert auf S. 11).

[ES09] M. Eigner, R. Stelzer. *Product Lifecycle Management: Ein Leitfaden für Product Development und Life Cycle Management*. Springer Science & Business Media, 2009 (zitiert auf S. 9).

[Fla06] D. Flanagan. *JavaScript: the definitive guide*. O'Reilly Media, Inc., 2006 (zitiert auf S. 13).

[HTM5] *HTML5*. URL: https://www.w3.org/TR/2014/REC-html5-20141028/ (zitiert auf S. 12).

[IDM] *ImageSite Engineering Document Management*. URL: http://www.equorum.com/imagesite-engineering-document-management/ (zitiert auf S. 19).

[IUG] *i3 User's Guide*. URL: http://i3wm.org/docs/userguide.html (zitiert auf S. 23, 25).

[IWM] *i3 - improved tiling wm.* URL: http://i3wm.org/ (zitiert auf S. 23, 25).

[LBLJ05] H. W. Lie, B. Bos, C. Lilley, I. Jacobs. „Cascading style sheets“. In: *WWW Consortium,(September 1996)* (2005) (zitiert auf S. 12).

[Lew98] S. M. Lewandowski. „Frameworks for component-based client/server computing“. In: *ACM Computing Surveys (CSUR)* 30.1 (1998), S. 3–27 (zitiert auf S. 11).

[LR01] A. Leff, J. T. Rayfield. „Web-application development using the model/view/controller design pattern“. In: *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International.* IEEE. 2001, S. 118–127 (zitiert auf S. 12).

[LX01] D. T. Liu, X. W. Xu. „A review of web-based product data management systems“. In: *Computers in industry* 44.3 (2001), S. 251–262 (zitiert auf S. 10, 11).

[MMP04] S. Mesihovic, J. Malmqvist, P. Pikosz. „Product data management system-based support for engineering project management“. In: *Journal of Engineering Design* 15.4 (2004), S. 389–403 (zitiert auf S. 9, 10).

[MS] T. B. MohdAnis, S. Subramaniam. „Operating System: The Power of Android“. In: *International Journal of Science and Research (IJSR)"[Online] Vol-03, PP* (), S. 2319–7064 (zitiert auf S. 21, 26).

[NAV] *Material design guidelines - Patterns - Navigation drawer.* URL: https://material.google.com/patterns/navigation-drawer.html (zitiert auf S. 21).

[Par14] T. Parisi. *Programming 3D Applications with HTML5 and WebGL: 3D Animation and Visualization for Web Pages.* Ö'Reilly Media, Inc.", 2014 (zitiert auf S. 15).

[PLS] *Plasma/Kontrollleisten.* URL: https://userbase.kde.org/Plasma/Panels/de (zitiert auf S. 21).

[RXJ] *Reactive Extensions for JavaScript.* URL: https://github.com/Reactive-Extensions/RxJS (zitiert auf S. 14).

[SF16] C. Schuster, C. Flanagan. „Reactive programming with reactive variables“. In: *Companion Proceedings of the 15th International Conference on Modularity.* ACM. 2016, S. 29–33 (zitiert auf S. 14).

[TKS] *Teamcenter - Konstruktions- und Simulationsmanagement.* URL: https://www.plm.automation.siemens.com/de_de/products/teamcenter/design-data-management/index.shtml (zitiert auf S. 11).

[Vau10] S. J. Vaughan-Nichols. „Will HTML 5 restandardize the web?“ In: *Computer* 43.4 (2010), S. 13–15 (zitiert auf S. 12).

Alle URLs wurden zuletzt am 06. 12. 2016 geprüft.

Erklärung

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

Ort, Datum, Unterschrift