Universität Hamburg
Fakultät für Mathematik,
Informatik und Naturwissenschaften
Department Informatik

Bachelorarbeit

Ein JavaScript Framework für interaktive Demos und Animationen zur technischen Informatik

Laszlo Korte

1lkorte@informatik.uni-hamburg.de
Studiengang Software-System-Entwicklung
Matr.-Nr. 6329857
Fachsemester 10

Erstgutachter Universität Hamburg:	Dr. Norman Hendrich
Zweitgutachter Universität Hamburg:	Wolf Posdorfer

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

Abkürzungsverzeichnis

ANSI	American National Standards Institute
API	Application Programming Interface
ASI	Automatic Semicolon Insertion
AST	Abstract Syntax Tree
BLIF	Berkeley Logic Interchange Format
CSS	Cascading Style Sheets
DMF	Disjunktive Minimal Form
DNF	Disjunktive Normal Form
DOM	Document Object Model
DSL	Domain Specific Language
FSM	Finale State Machine
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
JS	JavaScript
JSON	JavaScript Object Notation
KMF	Konjunktive Minimal Form
KNF	Konjunktive Normal Form
KV-Diagram	Karnaugh-Veitch-Diagramm
NPM	Node Package Manager
PEG	Parsing Expression Grammar
PLA	Programmable Logic Array
Rx	Reactive Extensions
RxJS	Reactive Extensions for JavaScript
SVG	Scalable Vector Graphic
TAMS	Technical Aspects of Multimodal Systems
TCP	Transmission Control Protocol

UI ..... User Interface

W3C ..... World Wide Web Consortium

XML ..... Extensible Markup Language

1 Einleitung

Abstract

Diese Arbeit behandelt die Entwicklung interaktiver Werkzeuge, mit denen Themengebiete der technischen Informatik zu Lehrzwecken veranschaulicht werden sollen. Der Fokus wird darauf gelegt, auf freien Webstandards aufzubauen, um eine plattformunabhängige Benutzbarkeit zu erreichen. Die hierbei entwickelten Werkzeuge sollen die Grundlage für ähnliche, zukünftig entwickelte Werkzeuge bilden.

Struktur der Arbeit

Zunächst werden in der Einleitung die Werkzeuge vorgestellt, welche im Rahmen dieser Arbeit umgesetzt werden sollen. Als nächstes gibt das Kapitel 2 *Das Web als Plattform* einen Überblick über die Sprachen und Bibliotheken, die in dieser Arbeit zum Einsatz kommen. Da der Fokus, der zu entwickelnden Werkzeuge, auf deren grafischer Benutzeroberfläche (GUI) liegt, werden in dem Kapitel 3 *Gestaltung der Benutzeroberfläche* die Möglichkeiten, die ein Browser zur Erzeugung von GUIs bietet, miteinander verglichen. Anschließend werden im Kapitel 4 *Modulare Architektur* die grundlegenden Architekturentscheidungen bezüglich des Werkzeugaufbaus erläutert. Im Kapitel 5 *Implementierung der Komponenten* werden schließlich die Details der Werkzeugimplementierung vorgestellt. Um die Einarbeitung in die Implementierung dieser Arbeit zu erleichtern, widmet sich das Kapitel 6 *Projektstruktur* der Strukturierung des Projektverzeichnisses und dem Umgang mit den Entwicklerwerkzeugen. Abschließend wird im *Fazit* das Ergebnis mit den gesetzten Zielen abgeglichen.

Motivation

Die technische Informatik behandelt sehr abstrakte Themen und Konzepte, für die es meist schwierig ist, eine gute Analogie zu etwas Bekanntem und Alltäglichem zu finden.

Um die Zusammenhänge eines Systems oder einer Problemstellung vollständig erfassen und begreifen zu können, genügt es nicht, sich als passiver Zuhörer einer Vorlesung den Sachverhalt erklären zu lassen und das Behauptete ungeprüft zu akzeptieren. Wichtig ist

es, eine individuelle Beziehung zu dem Themenkomplex aufzubauen, Ideen auszuprobieren, eigene Fragen zu entwickeln und Vermutungen überprüfen zu können.

Erst wenn man in der Lage ist, eine Vielzahl an Möglichkeiten und Alternativen innerhalb des zu verstehenden Systems zu erkunden und die jeweiligen Auswirkungen unmittelbar zu erleben, lässt sich ein intuitives Verständnis entwickeln, mit Hilfe dessen man sich sicher in dem Themenkomplex zurechtfindet, um zu eigenen Lösungsansätzen gelangen zu können.

Um solch ein interaktives Lernerlebnis für die Vorlesungen zur technischen Informatik zu ermöglichen, wurden schon vor einigen Jahren am Arbeitsbereich Technical Aspects of Multimodal Systems (TAMS) an der Universität Hamburg interaktive Komponenten in MatLab entwickelt.[CLEVE MOLER 1984]

Mit Hilfe dieser Komponenten können Dozenten in der Vorlesung das Verhalten von Systemen interaktiv vorführen. Studenten können begleitend dazu oder zuhause auf ihrem eigenen Computer selbst mit den Komponenten experimentieren. In Übungsgruppen können die Komponenten verwendet werden, um Lösungen für Aufgaben zu kontrollieren oder nachzuvollziehen. Die Funktionalität dieser Komponenten reicht vom Auswerten und Vergleichen von logischen Ausdrücken, über das Erzeugen von KV-Diagrammen bis hin zur Simulation Endlicher Automaten.

Mit den bestehenden Komponenten gibt es allerdings mehrere Probleme. Der Universität stehen nur begrenzt viele MatLab-Lizenzen zur Verfügung und aufgrund der hohen Kosten für eine dieser Lizenzen, ist es für die Stundenten nicht möglich, eigene Lizenzen zu erwerben. Ein weiteres Problem ist, dass die Komponenten für eine sehr alte MatLab Version gebaut wurden und zu einem großen Teil mit der aktuellen MatLab Version nicht kompatibel sind und daher nicht richtig funktionieren.

Außerdem ist die Verwendung der Komponenten auf mobilen Geräten wie Smartphones oder Tablets nicht möglich. Der Einsatz der Komponenten für aktuelle Vorlesungen ist also mit vielen Hürden verbunden und somit nicht praktikabel.

Neben den MatLab-Komponenten wurden am Arbeitsbereich TAMS auch einige Komponenten als Java-Applets entwickelt. Diese funktionieren zwar an Desktop-Computern problemlos, doch lassen sie sich nicht auf Tablet-Computern verwenden. Zum einen, da ihr Bedienkonzept nicht für einen Touchscreen ausgelegt ist, und zum anderen, weil das nötige Java Runtime Environment (JRE) auf Android und iOS Betriebssystemen nicht zur Verfügung steht.

Zielsetzung

Das Ziel dieser Arbeit ist es, durch den Neubau ausgewählter Komponenten ein sinnvolles Grundgerüst für interaktive Demonstrationen und Animationen zur technischen Informatik zu schaffen, welches auf freien Webstandards aufbaut und so ohne Einschränkung durch Lizenzen auf allen Geräten mit modernem Webbrowser funktionsfähig ist.

Hierbei wird ein besonderes Augenmerk auf die Modularität gelegt. Das Framework soll später unkompliziert um weitere Komponenten ergänzt werden können.

Ein weiterer wichtiger Aspekt ist die Benutzbarkeit auf mobilen Geräten. Die Komponenten sollen sich auch auf einem Tablet mit Touchscreen — und wenn sinnvoll, auch auf dem kleinen Bildschirm eines Smartphones — problemlos bedienen lassen. Konkret befasst sich diese Arbeit mit der Konstruktion folgender Komponenten.

Karnaugh-Veitch-Diagramm Editor

Karnaugh-Veitch-Diagramme bieten eine tabellarische Darstellung Boole'scher Funktionen, die es erlaubt, auf einem optischen Weg einen minimalen Funktionsterm zu ermitteln. Die Minimierung von Funktionstermen ist essenziell für die Vereinfachung von Schaltnetzen.

1998 entwickelte Matthias Meyer ein Werkzeug, um KV-Diagramme am Computer zu erzeugen, bearbeiten und darzustellen. Da dieses Werkzeug aber die Verwendung der Java Virtual-Machine voraussetzt und für den Gebrauch auf Desktopcomputern konzipiert ist, gilt es nun, ein entsprechendes Werkzeug für die Benutzung im Browser zu entwerfen.

Parser für Boole'sche Ausdrücke

Boole'sche Ausdrücke treten in allen Teilgebieten der Informatik auf. Je nach Anwendungsfall unterscheiden sie sich in ihrer Syntax. Für einen Ausdruck, der innerhalb einer Programmiersprache wie C verwendet werden soll, müssen andere Symbole für die Operatoren verwendet werden, als wenn derselbe Ausdruck in ein LaTeX-Dokument eingebettet werden soll, um ausgedruckt zu werden.

Auch innerhalb anderer Komponenten kann mit Boole'schen Ausdrücken gearbeitet werden. So lässt sich beispielsweise ein KV-Diagramm aus einem Boole'schen Ausdruck erzeugen und auch die Übergangsbedingungen in einem Endlichen Automat sind als

Boole'sche Funktionen formuliert. Diese Komponente soll in der Lage sein, verschiedene Dialekte an Ausdrücken, die der Benutzer eingibt, syntaktisch zu analysieren.

Funktionstabelle für Boole'sche Ausdrücke

Die Ausgabewerte einer Boole'schen Funktion für alle Kombinationen an Eingabewerten auf einen Blick sehen zu können, kann sehr hilfreich sein, um interessante Eigenschaften der Funktion schnell zu erfassen. Daher soll es möglich sein, Boole'sche Funktionen auszuwerten und als Tabelle darstellen zu lassen.

Boole'sche Funktionen vergleichen

Oft interessiert nicht jeder einzelne Wert einer Funktion, sondern nur, ob zwei Funktionen insgesamt übereinstimmen. Besonders bei der Korrektur von Übungsaufgaben und Klausuren, soll schnell überprüft werden können, ob eine gegebene Funktion mit dem erwarteten Ergebnis übereinstimmt. Diese Komponente soll den Vergleich zweier, vom Benutzer eingegebenen, Funktionen übernehmen.

LED Editor

Um einfache und praktische Beispiele für die Anwendung Boole'scher Ausdrücke zu liefern, erlaubt es der LED-Editor, eine Boole'sche Funktion interaktiv über das An- und Ausschalten von virtuellen Leuchtsegmenten an einer 7-Segment-Anzeige und anderen LED-Formationen zu erzeugen und als Funktionstabelle zu exportieren.

Interaktiver Zahlenkreis

Die Zuordnung von Bitmustern zu dezimalen Ganzzahlen lässt sich besonders gut an einem, zu einem Kreis eingerollten, Zahlenstrahl erklären. An so einem Zahlenkreis lässt sich visuell erkennen, wie es bei der Addition von großen Zahlen im Computer zu einem Überlauf in den negativen Zahlenbereich kommen kann. Diese Komponente soll das Erzeugen eines Zahlenkreises in verschiedenen Größen erlauben.

Simulator für Endliche Automaten

Mit Hilfe dieser Komponente soll der Übergangsgraph eines Endlichen Automaten bearbeitet werden können. Zudem soll es möglich sein, das Verhalten des Automaten in einer interaktiven Simulation zu betrachten. Der Benutzer soll die Möglichkeit haben, Eingabewerte zu erzeugen und zu beobachten, wie der Automat infolgedessen von einem Zustand in den anderen wechselt und Ausgabewerte erzeugt.

2 Das Web als Plattform

In diesem Kapitel wird ein Überblick über die Möglichkeiten, die ein Webbrowser als Zielplattform bietet, gegeben. Es werden verfügbare Technologien und Programmierschnittstellen vorgestellt und gegeneinander abgewogen.

Die Programmierschnittstellen der Webbrowser werden stetig weiterentwickelt, sodass es inzwischen nicht nur möglich ist, Internetseiten mit einfachen Inhalten wie Texten, Bildern und Videos darzustellen, sondern umfangreiche interaktive Benutzeroberflächen zu entwickeln, per Canvas-Schnittstelle Bitmap-Bilder zu bearbeiten, mit OpenGL 3D Szenen in Echtzeit zu rendern und Audiosignale zu erzeugen und zu verarbeiten.

Somit ist der Webbrowser zu einer idealen Plattform für die Entwicklung von Software geworden. Da ein Browser auf nahezu jedem Gerät — vom Desktop-PC, über Tablet-Computer bis hin zum Smartphone — verfügbar ist, bietet das Web als Plattform eine besonders große Reichweite.

2.1 Sprachen

JavaScript ist die einzige Turing-vollständige Programmiersprache, die von allen verbreiteten Webbrowsern unterstützt wird. Allerdings gibt es eine Vielzahl von alternativen Sprachen, die in der Lage sind, nach JavaScript übersetzt zu werden und sich so ebenfalls für die Entwicklung von Webanwendungen anbieten. Eine kleine Auswahl soll im Folgenden vorgestellt werden.

JavaScript

JavaScript ist eine dynamisch typisierte Scriptsprache, die urprünglich entwickelt wurde, um die ereignisbasiete Veränderung von zuvor statischen Internetseiten zu ermöglichen. So erlaubt es JavaScript beispielsweise mit Hilfe der Document Object Model-Programmierschnittstelle (DOM-API) auf die Mausbewegungen des Benutzers zu reagieren und daraufhin zusätzliche Informationen oder ein Kontextmenü an der Position des Mauszeigers darzustellen.

JavaScript ist unter dem Namen ECMAScript standardisiert und ist in allen modernen Browsern implementiert.[BRENDAN EICH 2016]

Es ist die einzige Scriptsprache, die von allen Browsern unterstützt wird. Allerdings ist JavaScript syntaktisch sowie semantisch nicht besonders elegant entworfen worden, sondern birgt einige Unschönheiten, die teilweise gar zu unerwartetem Verhalten führen können. Beispiele hierfür sind:

Kein Modulsystem

JavaScript hat von Haus aus kein Modulsystem. Somit ist es nicht möglich, größere Projekte in mehrere Namensräume zu teilen. Da sich alle Definitionen einen gemeinsamen globalen Namensraum teilen, besteht ständig die Gefahr von Namenskollisionen — besonders, wenn fremde Bibliotheken verwendet werden.

Automatische Semikolons

Die Verwendung des Semikolons zur Trennung von aufeinander folgenden Befehlen ist in JavaScript in den meisten Fällen optional. Der Parser fügt diese automatisch ein, wenn er nicht in der Lage ist, ein Token anderweitig zu verarbeiten. Dieses Verhalten ist zwar in drei einfachen Regeln festgelegt, allerdings kann es beim Schreiben und Lesen vom Programmcode zu Verwirrung führen. Die Funktion `addA` in Quelltext 2.1 gibt als Ergebnis `65` zurück. Hingegen gibt die Funktion `addB` als Ergebnis `undefined` zurück, weil die Automatic Semicolon Insertion-Regel (ASI) des Parsers automatisch ein Semikolon am Ende der `return`-Zeile einfügt.

```
1  function addA() {
2    return (42 + 23)
3  }
4
5  function addB() {
6    return
7      (42 + 23)
8  }
```

Quelltext 2.1: Zwei Funktionen, die sich aufgrund von ASI unterschiedlich verhalten

Automatische Typumwandlung

Bei der Verwendung des einfachen Vergleichsoperators „==“ bzw. „!=“ werden die verglichenen Werte automatisch zu einen gemeinsamen Typ umgewandelt. Das führt dazu,

dass der einfache Vergleich zweier Werte nicht transitiv ist, wie in Quelltext 2.2 zu sehen ist.[SIMPSON 2014]

```
0 == ""  // true
0 == "0"  // true

"" == "0" // false
```

Quelltext 2.2: Einfacher Vergleich zweier Werte in JavaScript

Um unerwartete Probleme zu vermeiden, sollten in JavaScript stets die Operatoren „===“ und „!==“ zum strengen Identitätsvergleich benutzt werden.

Automatische Deklaration globaler Variablen

Wird wie in Quelltext 2.3 bei der Deklaration das Schlüsselwort `var` vergessen, führt dies zu keinem Fehler, sondern zu der impliziten globalen Deklaration einer Variablen.

```
function foo(p) {
  sum = p + 20; // sum ist nun eine globale Variable
  return 2 * sum;
}
```

Quelltext 2.3: Implizite Erzeugung einer globalen Variablen

Abhilfe

Da diese Inkonsistenzen in der Sprache schnell zu Fehlern im Programm führen können, wurden von der JavaScript-Community verschiedene Werkzeuge entwickelt, die Abhilfe schaffen sollen.

Zum einen gibt es die so genannten Linter, die JavaScript-Quelltext besonders streng auf die Verwendung problematischer Features hin untersuchen. Sie verbieten beispielweise die Verwendung der einfachen Vergleichsoperatoren „==“ und „!=“, warnen bei der impliziten Deklaration globaler Variablen und nehmen viele weitere Analysen am Quelltext vor. Die beiden derzeit verbreitetsten Linter sind JSLint (6000 Sterne auf Github) und ESLint (4500 Sterne auf Github).[WALDRON et al. 2016][ZAKAS 2016]

Zum anderen sind eine Vielzahl neuer Sprachen entwickelt worden, die darauf ausgelegt sind, zu JavaScript Code kompiliert werden zu können. Die Verwendung einer zu

JavaScript alternativen Sprache wurde für diese Arbeit in Betracht gezogen. Daher wird im Folgenden eine Auswahl an Sprachen miteinander verglichen.

CoffeeScript

CoffeeScript ist eine solche Sprache, die sich nur syntaktisch von JavaScript unterscheidet. Anstelle von geschweiften Klammern setzt CoffeScript im Stil von Python auf Zeileneinrückung zur Kennzeichnung von Blöcken.[ASHKENAS 2016][MACCAW 2012]

Der Vergleichsoperator „==“ ist in CoffeScript als strenger Identitätsvergleich definiert. Doch auch CoffeeScript bietet kein Modulsystem, sodass es auch hier zu Namenskollisionen kommen kann. Da CoffeeScript nur eine alternative Syntax darstellt, lassen sich bestehende JavaScript-Bibliotheken problemlos verwenden. Auch in CoffeeScript geschriebene Funktionen lassen sich, nachdem sie zu JavaScript übersetzt wurden, problemlos aus klassischem JavaScript-Code heraus verwenden.

TypeScript

TypeScript ist eine von Microsoft entwickelte Sprache, die als echte Obermenge von JavaScript definiert ist. Somit löst sie zwar nicht die genannten semantischen und syntaktischen Probleme von JavaScript, erweitert die Sprache dafür aber um ein statisches Typsystem.[MAHARRY 2013]

TypeScript ermöglicht unter anderem die Deklaration von Interfaces, Klassen und Generischen Typen sowie die Typanmerkung von Funktionsparametern, Rückgabewerten und Variablen.

Außerdem erlaubt TypeScript auch die Definition von Modulen mit getrennten Namensräumen. Die Typdefinitionen nehmen dabei keinen Einfluss auf die Laufzeitsemantik, sondern dienen lediglich dazu, eine stärkere Unterstützung vom Compiler zu erhalten. Da es sich bei TypeScript um eine Obermenge von JavaScript handelt, lässt sich bestehender JavaScript-Code problemlos verwenden.

Dart

Die Programmiersprache Dart unterscheidet sich nicht nur syntaktisch, sondern auch semantisch stark von JavaScript.

Dart wurde von Google ursprünglich mit dem Ziel entwickelt, JavaScript langfristig abzulösen. Im Vergleich zu den anderen genannten Sprachen, muss Dart nicht nur zu JavaScript kompiliert werden, sondern bietet theoretisch auch die Möglichkeit, in einer eigenen Virtuellen Maschine (DartVM) ausgeführt zu werden. Der Plan, die DartVM tatsächlich in Google Chrome und später auch in andere Browser einzubinden, wurde jedoch aufgegeben, sodass Dart-Code in der Praxis auch zu JavaScript übersetzt werden muss, um im Browser ausgeführt werden zu können.[BAK et al. 2011][BAK und LUND 2015][BRACHA 2015]

Dart ist syntaktisch und semantisch stark an C# und Java angelehnt und bringt eine umfangreiche Standardbibliothek mit. Auch Dart erlaubt die Strukturierung von Projekten in unterschiedlichen Modulen. Aufgrund der gravierenden semantischen Unterschiede lässt sich Dart- und JavaScript-Code nicht so problemlos miteinander verwenden.

PureScript

PureScriptist eine stark von Haskell inspirierte, statisch typisierte, rein funktionale Programmiersprache. Als rein funktionale Programmiersprache unterscheidet sich PureScript sehr stark von JavaScript. Es verfügt über ein mächtiges Typsystem, sowie ein Modulsystem, bringt aber einen hohen Lernaufwand mit sich, wenn nicht schon über Erfahrungen in der funktionalen Programmierung, wie beispielsweise in Haskell, verfügt wird.[BURGESS und FREEMAN 2016][FREEMAN 2014]

ECMAScript 2016 / Babel

Auch JavaScript selbst wird stetig weiterenwickelt. So gibt es schon eine Menge an Features, die sich derzeit im ECMA-Standardisierungsprozess befinden und in zukünftigen JavaScript-Versionen enthalten sein werden.

So spezifiziert der ECMAScript 2015 Standard beispielsweise ebenfalls ein Modulsystem, mit dem es nun möglich ist, auch in JavaScript Module mit eigenen Namensräumen zu definieren und zu importieren.

Es dauert jedoch, bis alle Browser die neuen Funktionen vollständig in ihren JavaScrip-Engines implementiert haben. Auch Spezifikationen, die den Standardisierungsprozess schon durchlaufen haben, sind nicht unbedingt in allen Browsern auch schon implementiert. Hinzu kommt, dass viele Benutzer noch nicht die aktuellste Version ihres Browsers verwenden, sodass ein Entwickler sich nicht darauf verlassen kann, dass eine neue JavaScript-Funktion, die er verwendet, bei jedem Benutzer funktioniert.

Um diese Problematik zu lösen, gibt es Compiler, die in der Lage sind, JavaScript-Code, der für eine neue oder zukünftige JavaScript-Version geschrieben wurde, zu klassischem JavaScript-Code, der mit Sicherheit von allen Browsern verstanden wird, zu übersetzen. Der Verbreitetste dieser JavaScript-zu-JavaScript-Compiler ist Babel mit fast 15.000 Sternen auf Github. Babel lässt sich modular konfigurieren, sodass präzise festgelegt werden kann, welche JavaScript-Funktionalitäten verwendet werden können und welche nicht.[McKenzie 2014b][McKenzie 2014a]

Vorteile und Nachteile

Für das Projekt dieser Arbeit wird ECMAScript 2016 in Kombination mit Babel und ESLint verwendet.

PureScript hätte den schwerwiegenden Nachteil, sich stark von JavaScript zu unterscheiden und ist zudem noch sehr jung und nicht weit verbreitet. Es wäre also nicht gewährleistet, dass heute geschriebener Code auch in einigen Jahren noch problemlos von anderen Entwicklern gepflegt werden kann.

CoffeeScript unterscheidet sich zwar semantisch kaum von JavaScript, bietet aber auch nur wenige Vorteile. Diese Vorteile stehen in keinem Verhältnis zu dem Aufwand, eine komplett neue Syntax lernen zu müssen.[Carleton 2013]

Dart hat ebenfalls den Nachteil, sich stark von JavaScript zu unterscheiden. Die semantischen Unterschiede führen zusätzlich dazu, dass der aus kompiliertem Dart-Code resultierende JavaScript-Code weitaus umfangreicher ist, als handgeschriebener JavaScript-Code, da all die semantischen Unterschiede ebenfalls in den Ergebnis-Code eingebettet werden müssen.

Die Verwendung von Babel erlaubt es, JavaScript-Code zu schreiben, der mit hoher Wahrscheinlichkeit auch in ferner Zukunft noch funktionieren wird, da es sich bei ECMAScript um eine standardisierte Sprache handelt, deren neue Versionen die Browser auch weiterhin implementieren werden. Somit lassen sich die Vorteile von JavaScript als weit verbreitete Sprache mit den Vorteilen neuer Funktionen, wie einem Modulsystem, vereinen. Mit Hilfe von ESLint lassen sich die kleineren Fallstricke von JavaScript vermeiden.

2.2 Frameworks / Bibliotheken

Da der Browser und JavaScript ursprünglich nicht darauf ausgelegt waren, mit ihnen umfangreiche Anwendungen entwickeln zu können, fehlt es ihnen an den entsprechenden Werkzeugen, um dies zu erleichtern.[HAMILTON 2008][MILLS 2012]

So gibt es von Haus aus noch keine von allen Browsern unterstützte Möglichkeit, gekapselte UI-Komponenten zu definieren. Abgesehen von einigen primitiven HyperText Markup Language (HTML) -Elementen, gibt es auch keine umfangreiche Standardbibliothek an Komponenten, wie Listen, Tabellen oder Modalfenstern. Stattdessen müssen alle UI-Komponenten aus einfachen HTML-Elementen komponiert und mit Hilfe von Stylesheets gestaltet werden.

Um das Konstruieren von interaktiven Benutzeroberflächen im Browser zu erleichtern, wurden unterschiedliche Frameworks entwickelt. Die Zahl solcher Frameworks ist unvorstellbar groß. Jeden Monat werden neue Frameworks veröffentlicht und alte geraten in Vergessenheit. Daher kann es besonders schwierig sein, ein geeignetes Framework für seine Anforderung zu finden. Das Risiko, eine Anwendung auf eine Bibliothek oder ein Framework aufzubauen, welches in zwei Jahren vielleicht nicht mehr existiert oder nicht mehr weiterentwickelt wird, ist besonders groß.

Da mit dieser Arbeit selbst ein Framework für die Entwicklung interaktiver Komponenten entwickelt werden soll, welches auch in Zukunft noch gepflegt, erweitert und verwendet werden kann, sollen nicht vorschnell unnötige Abhängigkeiten geschaffen werden. Stattdessen sollen nur wenige ausgesuchte Bibliotheken verwendet werden, um die Effizienz der Entwicklung zu maximieren.

RxJS / CycleJS

RxJS ist die Implementierung der Reactive Extensions (ReactiveX oder Rx) Schnittstelle in JavaScript.[GROSS und KHOMERIKI 2014b]

Die ReactiveX Schnittstelle wurde ursprünglich von Microsoft für die Verwendung innerhalb von C# entwickelt. Inzwischen sind Implementationen für viele verschiedene Sprachen verfügbar — beispielsweise als RxJS für JavaScript.

Die Kernidee von ReactiveX ist die Kombination des Entwurfsmusters „Beobachter“ mit dem Entwurfsmuster „Iterator“, sowie mit einigen Konzepten der Funktionalen Programmierung.[GAMMA et al. 1995][GROSS und KHOMERIKI 2014b]

Die Kombination dieser drei Ideen zu dem Datentyp `Observable` ergeben ein mächtiges Werkzeug, welches es erlaubt, asynchrone Datenflüsse auf deklarative Art und Weise kurz und präzise zu definieren.

Der konkreten Verwendung des `Observable` Datentyps, zur Implementation der Komponenten, widmet sich das Kapitel 4 Modulare Architektur.

CycleJS ist eine kleine Hilfsbibliothek, die auf RxJS aufsetzt. CycleJS erlaubt es, mit Hilfe des `Observable` Datentyp von RxJS einen zyklischen Datenstrom zu erzeugen, um eine Anwendung nach Ideen der Mensch–Computer–Interaktion zu modellieren. Auch hierauf wird im Kapitel 4 Modulare Architektur genauer eingegangen.

Sowohl der `Observable` Datentyp von RxJS, als auch CycleJS, sind im Kern sehr simpel. Somit besteht keine Gefahr, auf ein Framework zu setzen, welches bald nicht mehr unterstützt wird.

ImmutableJS

Ursprünglich bietet JavaScript als komplexen Datentyp nur `Object` an. Objekte in JavaScript verhalten sich ähnlich einer HashMap. Somit ist es jederzeit möglich, Schlüssel-Werte-Paare an einem Objekte hinzuzufügen oder zu entfernen.[FLORIAN SCHOLZ 2015]

Für die Entwicklung umfangreicherer Komponenten ist es hilfreich, auf semantisch reichhaltigere Datenstrukturen zurückgreifen zu können. Zwar wird JavaScript mit ECMAScript Version 6 um weitere Datentypen, wie Sets, erweitert, doch besonders für die Funktionale Programmierung mit dem `Observable` Datentyp ist es von Vorteil, mit unveränderlichen Datenstrukturen arbeiten zu können. ImmutableJS bietet eine Vielzahl an unveränderlichen Datenstrukturen, wie geordneten und ungeordneten Sets und Maps, Listen, Stacks und Records.[LEE BYRON 2014]

Da es sich bei ImmutableJS nur um eine Bibliothek von einfachen Datenstrukturen handelt, besteht auch hierbei kein Risiko, sich langfristig einzuschränken.

2.3 Packagemanager und Abhängigkeiten

Um die Abhängigkeiten von fremden Bibliotheken in JavaScript zu verwalten, ist es sinnvoll, ein System zur Paketverwaltung zu verwenden. Für diese Arbeit wird NodeJS zu-

sammen mit dem Node Package Manager verwendet. Auf die Details der Verwendung wird später im Kapitel 6 Projektstruktur eingegangen.

NodeJS

NodeJS ist eine JavaScript Laufzeitumgebung, mit der sich JavaScript außerhalb eines Browsers — beispielweise auf der Kommandozeile — ausführen lässt. NodeJS basiert auf derselben JavaScript-Engine, die auch von Google Chrome verwendet wird.[DAHL 2009]

Node Package Manager

Der Node Package Manager (NPM) ist ein Packetverwaltungssystem, welches in JavaScript geschrieben ist. Es ermöglicht den Zugriff auf eine Datenbank von über 250.000 JavaScript Bibliotheken. NPM ist in der offiziellen NodeJS-Installation enthalten. [SCHLUETER 2016][DEBILL 2010]

2.4 Bundler

Um das Projekt übersichtlicht zu halten, ist es sinnvoll, die unterschiedlichen Module in mehrere Dateien zu trennen und in einer Verzeichnisstruktur zu organisieren. In anderen Programmiersprachen, wie Java oder C, ist es selbstverständlich, ein Projekt auf eine Vielzahl von Dateien zu segmentieren. In Java z.B. wird üblicherweise für jede Klasse eine eigene `.java` Datei angelegt.

In der Webprogrammierung mit JavaScript stellt sich aber das Problem, dass der Browser alle benötigten JavaScript-Dateien zunächst vom Server herunterladen muss. Für jede Datei wird eine HyperText Transfer Protocol (HTTP) Verbindung, im schlimmsten Fall gar jeweils eine eigene Transmission Control Protocol (TCP) Verbindung mit dem Server hergestellt. Das führt dazu, dass die Zeit, die der Benutzer auf den Aufbau der Seite warten muss, in Abhängigkeit mit der Anzahl der verwendeten JavaScript-Dateien steigt.[GUIDELINES 2016]

So führt eine feinere Aufgliederung in der Strukturierung des Projektes zu einer schlechteren Performance.

Es ist natürlich nicht sinnvoll, auf eine Strukturierung des Quelltextes zu verzichten, um die Ladezeit zu verbessern, denn dies würde den Entwicklungsaufwand in die Höhe treiben.

Zur Lösung dieser Situation gibt es sogenannte Bundler. Ein Bundler ist ein Präprozessor oder Compiler, der die Abhängigkeiten zwischen mehreren JavaScript-Dateien auflöst und alle benötigten Dateien zu einer gemeinsamen, größeren Datei zusammenfügt, sodass der Server nur eine einzige JavaScript-Datei an den Browser übertragen muss. Die beiden verbreitetsten Bundler in der JavaScript-Welt sind Browserify (9.000 Sterne auf Github) und Webpack (14.000 Sterne auf Github).[HALLIDAY 2012][KOPPERS 2012a]

Webpack

Webpack ist ein Modul-Bundler, der sich mit Hilfe einer Vielzahl an Plugins konfigurieren und anpassen lässt. Er ist nicht nur in der Lage, JavaScript-Module zusammenzufassen, sondern auch Stylesheets, Bilder und andere Dateien. BabelJS und ESLint stehen als Plugins für Webpack zur Verfügung, sodass die zusammengefügten JavaScript-Datein in ein und demselben Schritt auf Syntaxfehler überprüft und in eine browserkompatible Version übersetzt werden können.[KOPPERS 2012a]

2.5 Unterstützung älterer Browser

Die in dieser Arbeit entwickelten Komponenten sollen auch in älteren Browsern funktionieren. Beispielweise wird im Fachbereich TAMS an einigen Rechnern noch Firefox in der Version 18 verwendet.

Die an Objekten verfügbaren Methoden in JavaScript unterscheiden sich besonders zwischen aktuellen und älteren Browser-Versionen sehr stark. Beispielsweise ist die `Array.prototype.findIndex()` Methode, mit der sich der Index eines Elementes in einem Array anhand einer Predikatsfunktion ermitteln lässt, erst ab Firefox Version 25 verfügbar.[FLORIAN SCHOLZ AND JOE MEDLEY 2005]

Polyfills

JavaScript erlaubt es, Objekte zur Laufzeit um Methoden zu erweitern. Somit ist es möglich, alle Objekte zur Laufzeit so zu modifizieren, dass auch in älteren Browserversionen alle benötigten Methoden zur Verfügung stehen, auch wenn diese nicht vom Browser

selbst implementiert wurden. Diese Nachbesserung der Objektschnittstellen zur Laufzeit wird Polyfilling genannt.[SHARP 2010]

CoreJS ist eine modulare Bibliothek, die Implementationen aller modernen JavaScript-Methoden auch für ältere JavaScript Versionen bereitstellt.[PUSHKAREV 2016]

Vendor-Prefixes in Stylesheets

Die Browser unterscheiden sich nicht nur in Bezug auf die verfügbaren JavaScript-Schnittstellen, sondern auch dahingehend, was die verfügbaren Stylesheet-Attribute betrifft. Mittels Cascading Style Sheets (CSS) lässt sich das Aussehen von Elementen einer Webseite steuern. CSS ermöglicht unter anderem die Deklaration von Schriftfarben, Hintergrundbildern, Abständen, Elementpositionen, Umrandungen, abgerundeten Ecken oder Schlagschatten. Auch lassen sich mittels CSS komplexe Layoutbeziehungen zwischen Elementen festlegen. Wie JavaScript wird auch CSS stetig weiterentwickelt. So kommt es, dass viele neue CSS-Befehle nicht von allen verbreiteten Browsern unterstützt werden.

```
1  .content-box {
2    display: -webkit-box;   /* iOS 6, Safari 3-6 */
3    display: -ms-flexbox;   /* <= IE 10 */
4    display: -webkit-flex; /* Safari 6.1+, iOS 7.1+ */
5    display: -moz-box-fex; /* Firefox old */
6    display: flex;          /* Firefox, Chrome, Opera */
7  }
```

[HUND 2013]

Quelltext 2.4: Vendor-Prefixes für Flexbox Layout

Allerdings implementieren auch ältere Versionen von Browsern oft schon eigene Varianten von CSS-Befehlen, die sich nur gering von den offiziell standardisierten Befehlen unterscheiden. Ein einfaches Beispiel hierfür ist das Attribut `box-sizing`, welches die Größenberechnung eines Elements steuert. Es wird von Firefox in der standardisierten Form erst seit Version 29 unterstützt. Allerdings implementiert Firefox schon seit Version 2 das Attribut `-moz-box-sizing`, welches sich semantisch nicht von dem offiziellen Attribut unterscheidet. Mit der Verwendung eines browsereigenen Namens sollen unter anderem das Auftreten von Konflikten zwischen der standardisierten Spezifikation eines Befehls und den Implementationdetails des jeweiligen Browsers vermindert werden. Diese Strategie wird Vendor-Prefixing genannt, da dem offiziellen Namen eines Befehls zusätzlich der Name des jeweiligen Browsers voran gestellt wird. Neue Versionen von Google Chrome und Opera verzichten zwar inzwischen auf Vendor-Prefixing, doch in der Praxis ist es für viele Attribute nötig, sie mit bis zu fünf verschiedenen Vendor-Prefixes zu deklarie-

ren, wie in Quelltext 2.4 auf der vorherigen Seite zu sehen ist.[CHROMIUM 2015][MOSLEY 2010]

Webpack ist mittels Plugins so konfigurierbar, dass dem Entwickler das Schreiben von CSS-Vendor-Prefixes abgenommen wird und diese automatisch an den nötigen Stellen eingefügt werden.[SITNIK 2014][KOPPERS 2012b][SITNIK 2013]

Um das Schreiben von Stylesheets weiter zu vereinfachen, wird in dieser Arbeit auch eine zu CSS alternative Sprache, Namens Stylus, verwendet. Sie stellt eine Obermenge von CSS dar, bietet aber eine vereinfachte Syntax, sowie zusätzliche Features, die es ermöglichen, die Menge des zu schreibenen Codes für komplexe Layouts stark zu verringern.[HOLOWAYCHUK 2010][IMMS 2014]

3 Gestaltung der Benutzeroberfläche

Im Folgenden wird auf die Aspekte eingegangen, die bezüglich der grafischen Benutzeroberflächen der Komponenten zu beachten sind. Bei der Gestaltung der Benutzeroberfläche aller Komponenten soll ein Hauptaugenmerk darauf gelegt werden, dass sie sich auf einem möglichst breiten Spektrum an Geräten und Bildschirmgrößen bequem verwenden lassen. Neben Desktop-Computern stellen auch Tablet-Computer eine zentrale Zielgruppe dar.

3.1 Unterstützung für Mobilgeräte

Bei der Unterstützung von Mobilgeräten ist nicht nur die geringere Bildschirmgröße zu beachten, sondern auch die begrenzte Verfügbarkeit von Eingabegeräten wie Maus und Tastatur. Tastenkombinationen wie `Ctrl-Klick`, die bei der Bedienung an einem Desktop-Computer kein Problem darstellen, sind auf einem Touchscreen ohne Tastatur nicht möglich. Des Weiteren ist für die Benutzbarkeit auf einem Touchscreen die Größe der Interaktionsflächen zu beachten. Mit dem Finger lassen sich keine so präzisen Bewegungen ausführen, wie mit einem Mauszeiger. Die Google Developer-Guidelines empfehlen eine Mindestgröße von 48×48 Pixel für Elemente, die auf einem Touchscreen auswählbar sein sollen.[GUIDELINES 2014]

3.2 Möglichkeiten der Darstellung

Für die Darstellung der Benutzberoberfläche gibt es unterschiedliche Möglichkeiten. Im Folgenden werden die verschiedenen Programmierschnittstellen miteinander verglichen.

3.2.1 HyperText Markup Language

HyperText Markup Language (HTML) ist ursprünglich für die semantische Strukturierung von Textdokumenten gedacht und erlaubt das Erzeugen von einfachen Textblöcken, Bildern und Tabellen. HTML-Elemente können ineinander verschachtelt werden, um aus einfachen Elementen komplexere Strukturen zu erzeugen. Das Aussehen von HTML-Elementen kann mit Hilfe von Stylesheets (CSS) sehr ausgiebig verändert werden. Unter

anderem können Größe, Position, Textfarben, Hintergrundfarben und -bilder, Abstände, Umrandungen der Elemente verändert werden. Auch können Elemente zweidimensional und dreidimensional transformiert — also beispielsweise rotiert, skaliert oder verzerrt — werden.

In ihrem Kern stellen HTML-Elemente aber immer zweidimensionale, rechteckige Blöcke dar, sodass geschwungene Formen nur sehr begrenzt erzeugt werden können.[COYIER 2009][HOWE 2015]

Da der Browser das Layouting der Elemente übernimmt, können die Elemente in ihrer Größe automatisch an den von anderen Elementen benötigten Platz angepasst werden, ohne dass hierfür manuelle Berechnungen durchgeführt werden müssen.

3.2.2 Bitmap Canvas

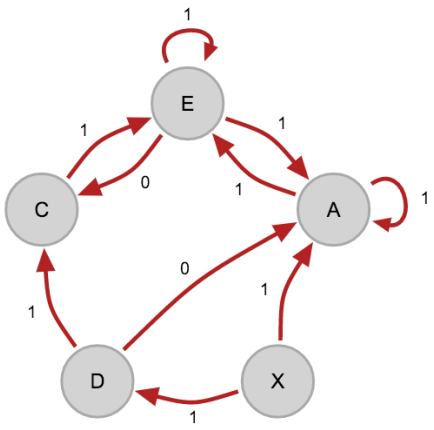


Abbildung 3.1: Ein gerichteter Graph als Bitmapgrafik

Das `<canvas>` Element ist ein spezielles HTML-Element, welches eine Pixelbitmap darstellt, die über eine entsprechende Schnittstelle per JavaScript imperativ manipuliert werden kann.

Die Canvas-Schnittstelle erlaubt das pixelgenaue Erzeugen und Verändern von Bitmapgrafiken und bietet somit mehr Möglichkeiten, eine Darstellung nach genauen Anforderungen zu erzeugen. So ist es unter anderem möglich, Bézierkurven zu zeichnen oder eine Matrix als Nachbearbeitungsfilter auf ein erzeugtes Bild anzuwenden.[HOLLDACK 2011][BUCKLER 2011]

Auch in Bezug auf die Performance kann die Verwendung des Canvas-Elementes einige Vorteile gegenüber des HTML/CSS-Ansatzes haben, denn sämtliche Layout-

Berechnungen, die der Browser zur Darstellung von HTML-Elementen tätigen muss, fallen weg, da die JavaScript-Prozedur exakt vorgibt, wie die Grafik erzeugt — also wie die Bitmap gefüllt — werden soll.

Damit bringt die Verwendung des Canvas-Elementes zur Erzeugung von Benutzeroberflächen aber auch einige gravierende Nachteile mit sich: Dadurch, dass auf sämtliche Layout-Berechnungen des Browsers verzichtet wird, muss der Entwickler sich selbst darum kümmern, wie das Canvas auf unterschiedlichen Bildschirmgrößen oder bei variierenden Textlängen dargestellt werden muss. Wenn ein HTML-Element mehr Text enthält, als in eine Zeile passt, sorgt das Layouting des Browsers selbstständig dafür, den Text sinnvoll umzubrechen. Soll hingegen ein, auf ein Canvas gezeichneter, Text umgebrochen werden, muss der Entwickler alle nötigen Berechnungen hierfür selbst durchführen.

Einen weiteren Nachteil gibt es in Bezug auf die Benutzerinteraktion. Interagiert ein Benutzer mit einem Element in einem klassischen HTML-Dokument, indem er beispielsweise auf einen Button klickt, wird diese Interaktion von dem Browser automatisch dem richtigen Element zugeordnet. Das erlaubt es dem Entwickler einfach auf Benutzeraktionen zu reagieren, indem er für die jeweiligen HTML-Elemente EventListener definiert.

Wenn der Benutzer hingegen mit Elementen, die in die Bitmap eines Canvas gezeichnet wurden, interagieren soll, kann der Browser die Zuordnung zwischen Benutzeraktion und Element nicht übernehmen, da ihm die nötigen Informationen nicht zur Verfügung stehen. Somit muss sich auch hier der Entwickler selbst darum kümmern, die Mauskoordinaten dem vom Benutzer gemeinten Element zuzuordnen.

WebGL

WebGL ist eine alternative Schnittstelle des Canvas Elements, die an OpenGL angelehnt ist und es erlaubt, direkter mit der Grafikkarte zu interagieren, um komplexe Grafiken effizienter erzeugen zu können. Besonders für das Zeichnen von dreidimensionalen Szenen ist WebGL interessant. Die Vor- und Nachteile sind in etwa dieselben, wie die der zweidimensionalen Canvas-Schnittstelle: hohe Freiheit im performanten Erzeugen von Grafiken und keine Unterstützung vom Browser beim Layouting und der Verarbeitung von Benutzeraktionen.[JACKSON und MARRIN 2011]

3.2.3 Scalable Vector Graphics

Scalable Vector Graphics (SVG) ist ein Extensible Markup Language (XML) basiertes Vektorgrafikformat, welches von allen aktuellen Browsern unterstützt wird. Es lässt sich di-

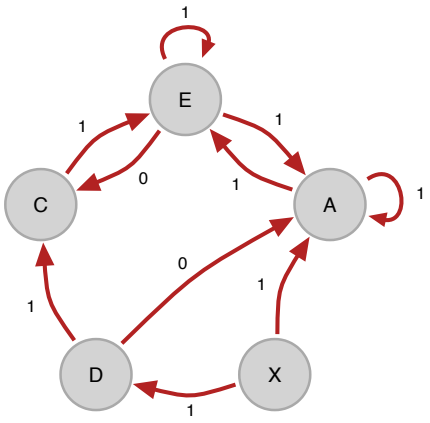


Abbildung 3.2: Ein gerichteter Graph als SVG

rekt in HTML-Dokumente einbetten oder auch als externe Grafik verlinken. Das Aussehen einzelner Elemente einer SVG lässt sich per Stylesheet überschreiben. Außerdem lässt sich ein SVG-Dokument, ähnlich wie ein HTML-Dokument, über eine Document Object Model-Programmierschnittstelle (DOM-API) per JavaScript auslesen und manipulieren.[ERIK DAHLSTRÖM et al. 2001]

Als Vektorgrafikformat unterstützt SVG selbstverständlich eine Vielzahl an Formen wie Linien, Polygone, Ellipsen und Pfade aus Bézierkurven. In Abbildung 3.2 ist ein Graph dargestellt, der per SVG erzeugt wurde. Er unterscheidet sich optisch nicht von dem Graphen aus Abbildung 3.1 auf Seite 20.

Gegenüber einer per Canvas erzeugten Bitmapgrafik hat das SVG-Format den Vorteil, auf allen Bildschirmen gleichermaßen gestochen scharf dargestellt werden zu können. Viele moderne Mobilgeräte sind mit besonders hochauflösenden Displays ausgestattet. Um auf diesen Displays scharf dargestellt werden zu können, müssen Bitmaps, je nach Gerät, in doppelter oder dreifacher Auflösung erzeugt werden. So muss auch bei Verwendung der Canvas-Schnittstelle darauf geachtet werden, die Pixeldichte des Gerätes auszulesen und das Canvas-Element in entsprechender Größe zu erzeugen. Bei Verwendung des SVG-Formates muss sich hierum nicht gekümmert werden.[PLAG 2013]

SVG unterstützt auch die Interaktion mit einzelnen Elementen. So können per JavaScript über die DOM-Schnittstelle Beobachterprozeduren für Ereignisse zu einzelnen Elementen einer Grafik hinzugefügt werden. Somit vereinigt SVG viele Vorteile von HTML/CSS mit denen der Canvas-API.

4 Modulare Architektur

Im Rahmen dieser Arbeit soll eine Architektur für die verschiedenen Komponenten gefunden werden, die es ermöglicht, zukünftig, mit wenig Aufwand, auch weitere Komponenten zu entwerfen und umzusetzen. In diesem Kapitel wird die Architektur der Komponenten — sowie die zugrundeliegenden Überlegungen — dargestellt.

Die Komponenten sollen zwar alleinstehend funktionieren, allerdings soll es in geeigneten Fällen auch problemlos möglich sein, Komponenten miteinander zu verbinden. Ein konkretes Beispiel für solch einen Fall findet sich in der Verbindung der KV-Editor-Komponente und der Logic-Expression-Parser-Komponente. So ist es sowohl sinnvoll, ein KV-Diagramm aus einem, vom Benutzer eingegebenen, Boole'schen Ausdruck erzeugen zu können, als auch einen minimierten Boole'schen Ausdruck als Ergebnis des Minimierungsprozesses im KV-Diagramm exportieren zu können.

4.1 Problemstellung

Im Rahmen dieser Arbeit wird nur der sehr begrenzte Teil einer zukünftig womöglich großen Menge an Komponenten entwickelt. Welche Anforderungen spätere Komponenten an die gesamte Architektur stellen, ist noch nicht mit Sicherheit abzusehen. Zudem sollen die Komponenten auch komplett unabhängig voneinander benutzt werden können. Damit Komponenten miteinander kombiniert werden können, wird eine gemeinsame Schnittstelle benötigt. Diese Schnittstelle darf aber nicht zu genau spezifiziert sein, sodass zukünftige Komponenten genügend Freiheit haben, ihre Anforderungen zu implementieren.

4.2 Funktionale Programmierung

Diese Anforderung lässt sich sehr gut erfüllen, indem jede Komponente als eine Funktion im Sinne der Funktionalen Programmierung betrachtet wird.

Damit ist es jeder Komponente möglich, über ihre Paramter benötigte Daten von anderen Komponenten zu erhalten und über ihren Rückgabewert Daten an andere Komponenten zur Verfügung zu stellen.

Referenzielle Transparenz

Die einzige Einschränkung, an die sich jede Komponente halten muss, ist die Referenzielle Transparenz. Das bedeutet, dass ihr Rückgabewert allein von den Parameterwerten abhängt. Dazu zählt auch, dass die Ausführung einer Funktion, also die Verwendung einer Komponente, keinen eigenständigen Einfluss auf den globalen Zustand der Applikation haben darf. Eine Komponente darf also nicht auf globale Variablen zugreifen.

Funktionale (De)komposition

Durch diese einfache und doch zunächst schwerwiegend scheinende Einschränkung entsteht ein enormer Vorteil: Das Verwenden einer Komponente in einer anderen Komponente verlangt lediglich nach einem simplen Funktionsaufruf. Damit Komponente A auf Daten, die von Komponente B erzeugen werden, zugreifen kann, muss Komponente A lediglich die Funktion, als welche Komponente B definiert ist, aufrufen und erhält die Daten als Rückgabewert. Zudem erlaubt eine solche Architektur die Dekomposition von komplexen Komponenten in kleinere einfachere Komponenten mittels Funktionaler Dekomposition.

4.3 Streams und der „Observable“ Datentyp

Nun sollen Komponenten aber nicht nur einmalig einen Ergebniswert berechnen, sondern dem Benutzer die Möglichkeit geben, mit ihnen zu interagieren.

So soll der Benutzer beispielsweise Zeilen und Spalten in einem KV-Diagramm mit der Maus — oder mit einem Finger auf dem Touchscreen — anklicken können oder in ein Textfeld einen Ausdruck eingeben können, der dann von der entsprechenden Komponente syntaktisch analyisiert wird.

Eine Komponente muss also in der Lage sein, asynchrone Eingaben — wie Tastendrücke des Benutzers — über einen beliebigen Zeitraum hinweg verarbeiten zu können. Es wird also eine Methode benötigt, eine Asynchrone Datenverarbeitung mittels synchronen Funktionsaufrufen zu modellieren.

Die Lösung hierfür bietet der `Observable` Datentyp, der von der Reactive Extensions (ReactiveX) Programmierschnittstelle definiert wird[GROSS und KHOMERIKI 2014b][STALTZ 2016].

Der `Observable` Datentyp erlaubt es, über die Zeit veränderliche Werte zu modellieren und mit ihnen wie mit einer Sammlung von Werten zu arbeiten. Dafür wird eine funktionale Schnittstelle bereitgestellt, über die sich ein `Observable`, vergleichbar mit einem Array oder einer Liste, transformieren lässt. Anders als eine Liste, hält ein `Observable` aber nicht tatsächlich mehrere Werte im Speicher, sondern verwaltet nur die Benachrichtigung von angemeldeten Beobachter-Prozeduren über das Auftreten neuer Werte.[WEBBER 2014][MANSILLA 2015][MEIJER 2012]

Im Folgenden werden die Grundlagen der Schnittstelle des `Observable` Datentyps in Analogie zu der Schnittstelle eines klassischen Arrays bzw. einer Liste erklärt. Es wird darauf verzichtet, auf alle Details der `Observable`-Schittstelle einzugehen. Für eine ausführlichere Erläuterung der `Observable` Schnittstelle ist die Dokumentation von der Reactive Extensions bzw RxJS zuhilfe zu ziehen.[GROSS und KHOMERIKI 2014b][LESH et al. 2015]

Einfaches Beispiel an einem Array

```
1  const allValues = [1, 2, 3, 23, 42, 7, 8, 64];
2  // Filter
3  const bigValues = allValues.filter((x) => x > 10);
4  // [23, 42, 64]
```

Quelltext 4.1: Filterung der Elemente eines Arrays

Zunächst wird in Quelltext 4.1 ein einfaches Array `allValues` (kein `Observable`) aus 8 Werten erzeugt. Als nächstes wird dieses Array mittels des Predikats `x > 10` gefiltert, um ein neues Array `bigValues` zu erzeugen. `bigValues` enthält nur noch drei Werte. Das Ursprungsarray wurde von der Filteroperation nicht beeinflusst.

```
1  const allValues = [1, 2, 3, 23, 42, 7, 8, 64];
2  // Map:
3  const doubleValues = allValues.map((x) => x * 2)
4  // [2, 4, 6, 46, 84, 14, 16, 128]
```

Quelltext 4.2: Multiplikation der Elemente eines Arrays

In Quelltext 4.2 wird ebenfalls ein Array (`allValues`) aus 8 Werten erzeugt. Mittels des `map` Operators werden alle Elemente des Arrays mit `2` multipliziert, um das Array `doubleValues` zu erzeugen. Auch hier wurde das Ursprungsarray `allValues` nicht modifiziert.

```
const allValues = [1, 2, 3, 23, 42, 7, 8, 64];
// Reduce:
const sum = allValues.reduce((acc, x) => acc + x, 0)
// 150
```

Quelltext 4.3: Aufsummierung der Elemente eines Arrays

Zu guter Letzt wird in Quelltext 4.3 erneut ein Array aus Zahlen erzeugt. Mit Hilfe des `reduce` Operators wird dann die Summe aller Werte des Arrays gebildet.

Übertragung auf den Observable Typ

Auch die Schnittstelle eines `Observable` enthält unter anderem die Operatoren `map`, `filter` und `scan`. So lassen sich die vorherigen Beispiele direkt auf den `Observable` Typ übertragen:

```
import {Observable} from 'rx';

const allClick$ = Observable.fromEvent(document, "mousedown");
// Filter:
const rightClick$ = allValues.filter((e) => e.which === 2);
```

Quelltext 4.4: Filterung der Elemente eines Observables

In Quelltext 4.4 wird ein einfaches `Observable` `allClick$` aus DOM-Ereignissen erzeugt. Es enthält alle `mousedown` Ereignisse, die auf der gesamten Seite bzw. dem gesamten Dokument stattfinden. Als nächstes wird ein neues `Observable` erzeugt, indem der `filter`-Operator mit dem Predikat `e.which === 2` angewendet wird. Dabei ist `e` das DOM-Ereignisobjekt und `which` die Eigenschaft, die den Index der gedrückten Maustaste enthält. Das Predikat beschreibt also alle Ereignisse, bei denen die sekundäre Maustaste gedrückt wurde. Entsprechend enthält `rightClick$` nicht mehr alle `mousedown` Ereignisse, sondern nur noch jene, welche durch einen Sekundärklick ausgelöst wurden.

Zu beachten ist, dass die Variablennamen `allClick$` und `rightClick$` auf ein Dollarzeichen enden. Diese Benennung folgt der Konvention, Variablen, die `Observables` enthalten, per ungarischer Notation zu kennzeichnen. Das Dollarzeichen `$` soll dabei an den Buchstaben `S` erinnern, mit welchem in der englischen Sprache der Plural eines Nomens gebildet wird.

Wie weiter oben bereits erwähnt, enthalten die `Observables` `allClick$` und `rightClick$` die Ereignissobjekte nicht in dem Sinne, dass diese tatsächlich irgendwo im Speicher an-

gesammelt werden. Der Konstruktor `Observable.fromEvent` stellt lediglich eine Abstraktion dar, die intern auf die `addEventListener`-Schnittstelle des Document Object Models zurückgreift. Dementsprechend kann — anders als beim Array — nicht per Subscriptoperator `[]` über den Index auf die einzelnen Ereignisobjekte des `Obervable` zugegriffen werden, wie in Quelltext 4.5 verdeutlicht wird.[SCHÄFER 2009][PODWYSOCKI 2015]

```
1  import {Observable} from 'rx';
2
3  const allNumbers = [4, 8, 15, 16, 23, 42];
4  const allClick$ = Observable.fromEvent(document, "mousedown");
5
6  const fifthNumber = allNumbers[4]; // 23
7  const fifthMouseClick = allClick$[4]; // <-- Nicht möglich
```

Quelltext 4.5: Indizierter Zugriff auf Elemente eines Observables ist nicht möglich.

Auf die Werte eines `Observable` kann nur asynchron zugegriffen werden. Hierfür muss, wie beim klassischen Entwurfsmuster „Beobachter“, ein Beobachter angemeldet werden. Ein Beobachter ist eine Prozedur, die für jeden Wert, der in einem Observable auftritt, ausgeführt wird. Als Parameter bekommt der Beobachter den jeweiligen Wert übergeben. So wird in Quelltext 4.6 ein Beobachter definiert, der für jeden Sekundärklick, der im Dokument stattfindet, aufgerufen wird.

```
1  import {Observable} from 'rx';
2
3  const allClick$ = Observable.fromEvent(document, "mousedown");
4  const rightClick$ = allValues.filter((e) => e.which === 2);
5
6  const disposable = rightClick$.subscribe((event) => {
7    console.log("A right click occured");
8  });
9
10 // Nach 10 Sekunden, wird der Beobachter wieder abgemeldet.
11 setTimeout(() => {
12   disposable.dispose();
13 }, 10000);
```

Quelltext 4.6: Hinzufügen einer Beobachterprozedur zu einem Observable

Analog zu den zuvor gezeigten Arraytransformationen lassen sich auch `Observables` vielseitig transformieren.

In Quelltext 4.7 auf der nächsten Seite werden die kartesischen Koordinaten der Ereignisobjekte extrahiert und in ein neues `{x,y}` Objekt transfomiert. So enthält das

```
import {Observable} from 'rx';

const mouseMove$ = Observable
  .fromEvent(document, "mousemove");

const mouseCoords$ = allValues.map(
  (e) => ({x: e.pageX, y: e.pageY})
);
```

Quelltext 4.7: Transformation der Elemente eines Observables

`mouseCoords$` `Observable` nicht mehr die vollständigen DOM-Ereignisobjekte, sondern nur einfache Objekte mit `x` und `y` Attributen.

```
import {Observable} from 'rx';

const mouseClick$ = Observable
  .fromEvent(document, "mousedown");

const clickCount$ = allValues.scan(
  (acc, e) => acc + 1, 0
);
```

Quelltext 4.8: Summierung der Elemente eines Observables

In Quelltext 4.8 wird die Anzahl der Mausklicks mittels des `scan`-Operators aufsummiert. Der `scan`-Operator funktioniert ähnlich wie der `reduce`-Operator, der zuvor anhand von Arrays demonstriert wurde. Er unterscheidet sich jedoch darin, dass sein Ergebnis nicht einen einzelnen Wert darstellt, sondern ein `Observable`, welches alle Zwischenergebnisse enthält.

Neben den genannten Funktionen `map`, `filter` und `scan`, unterstützt der `Observable`-Datentyp eine große Menge weiterer Operatoren, mittels derer sich mächtige asynchrone Berechnungen deklarieren lassen.[Staltz 2015c][Gross und Khomeriki 2014a]

Komponenten mittels Observable definieren

In diesem Abschnitt wird aufgezeigt, wie sich der `Observable` Datentyp verwenden lässt, um interaktive Komponenten als JavaScript-Funktion zu definieren.

In Quelltext 4.9 auf der nächsten Seite und Abbildung 4.1 auf der nächsten Seite wird gezeigt, wie die Verwendung des `Observable` Datentyp es den Komponenten `logicParser` und `kvDiagram` erlaubt, miteinander zu interagieren. Zuerst werden die

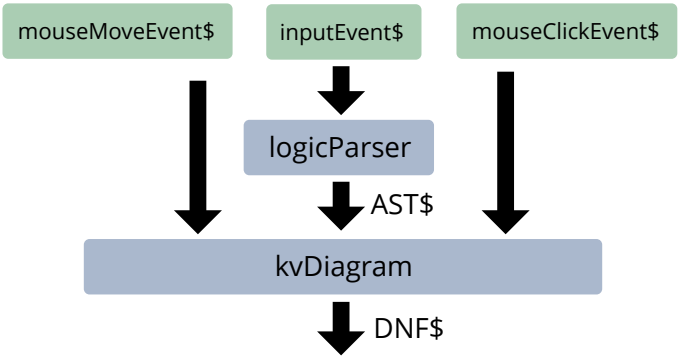


Abbildung 4.1: Die Verbindung zweier Komponente mittels eines Observables

```
import {Observable} from 'rx';

const inputField = document.createElement("input");

const inputEvent$ = Observable
  .fromEvent(inputField, "input");
const mouseMoveEvent$ = Observable
  .fromEvent(document, "mousemove");
const mouseClickEvent$ = Observable
  .fromEvent(document, "click");

const AST$ = logicParser(inputEvent$);

const DNF$ = kvDiagram(
  AST$, mouseMoveEventEvent$, mouseClickEvent$
);
```

Quelltext 4.9: Die Verbindung zweier Komponente mittels eines Observables

`Obervables` erzeugt, die die Eingaben des Benutzers repräsentieren: `mouseMoveEvent$`, `inputEvent$` und `mouseClickEvent$`. `inputEvent$` enthält die Eingaben des Benutzers in das Textfeld. Diese werden von der Logikparser-Komponente, welche als `logicParser` Funktion definiert ist, in das `AST$ Observable` transformiert, welches den Abstrakten Syntaxbaum (AST) des Boole’schen Ausdrucks enthält.

Die ebenfalls als Funktion definierte KV-Diagramm-Komponente bekommt als Parameter sowohl den von der Logikkomponente erzeugen Syntaxbaum (`AST$`), als auch die Mausbewegungen (`mouseMoveEvent$`) und Mausklicks (`mouseClickEvent$`) und transformiert diese drei Eingaben zu einem Boole’schen Ausdruck in disjunktiver Minimalform.

Da es sich sowohl bei den Parametern, als auch bei den Rückgabewerten der Komponenten um `Observables` handelt, können die Komponenten auf asynchrone Eingaben

reagieren: Tritt ein neues `input` Ereignis auf, wird dieses von dem `inputEvent$` Observable verarbeitet. Da `AST$` als Ergebnis der Transformation von `inputEvent$` durch die `logicParser` Komponente entstanden ist, kann durch die Benutzereingabe auch ein neuer Syntaxbaum im `AST$` entstehen. Der neue Syntaxbaum wiederum kann zu einer neuen disjunktiven Minimalform im `DNF$` Observable führen.

So lässt sich mittels des `Observable`-Datentyps ein Datenfluss durch die Komponenten deklarieren.

4.4 Virtueller DOM

Im Folgenden wird das Konzept des Virtuellen DOMs vorgestellt, welches es erlaubt, DOM-Manipulationen mittels Funktionaler Schnittstelle vorzunehmen, bzw. zu beschreiben[CHEDEAU 2013].

In Quelltext 4.9 auf der vorherigen Seite wurde das Textfeld, in welches der Boole'sche Ausdruck eingegeben werden soll, außerhalb der `logicParser` Komponente — bzw. Funktion — erzeugt. Letztendlich soll aber jede Komponente in der Lage sein, die für ihre Darstellung benötigten DOM-Elemente selbst zu definieren.

Es wurde festgelegt, dass Komponenten einfache Funktionen sein sollen und dementsprechend nur ihre Eingabeparameter auf Rückgabewerte abbilden dürfen. Daher ist es einer Komponente nicht möglich, eigenständig DOM-Elemente zu erzeugen und diese in den DOM-Baum einzufügen, um sie anzuzeigen. Der DOM-Baum stellt einen globalen Zustand der Applikation dar. Das Einfügen von Elementen in den DOM-Baum seitens einer Komponente wäre eine Mutation dieses globalen Zustandes, welche die referenzielle Transparenz der Komponente verletzen würde. Besonders wenn mehrere Komponenten gemeinsam auf einer Seite/in einem Dokument benutzt werden sollen, wäre es fatal, wenn sich ihre DOM-Modifikationen in die Quere kämen.

Die Lösung, die es Komponenten trotz dieser Restriktion erlaubt, ihre DOM-Repräsentation selbst zu erzeugen, besteht in dem Konzept des Virtuellen DOMs.

Komponenten dürfen zwar selbst den DOM-Baum nicht manipulieren, doch sie können ein Objekt als Rückgabewert liefern, welches den Teilbaum, der erzeugt werden soll, repräsentiert. Damit wird die tatsächliche Manipulation des DOM-Baumes an den Verwender der Komponente delegiert.

Quelltext 4.10 auf der nächsten Seite zeigt, wie die `logicComponent()` Funktion einen DOM-Baum erzeugt, welcher dann in den globalen DOM eingefügt wird.

```
const container = document.createElement('div');
document.body.appendChild(container);

const logicDomTree = logicComponent();
insertVirtualDomInto(logicDomTree, container);
```

Quelltext 4.10: Einfügen der DOM-Knoten einer Komponente in den globalen DOM-Baum

Tatsächlich soll eine Komponente aber nicht nur eine einzige DOM-Repräsentation erzeugen können, sondern diese in Abhängigkeit der Benutzereingaben und anderen Ereignissen auch ändern und aktualisieren können. Um einer Komponente zu ermöglichen, ihren DOM-Baum in Abhängigkeit von Benutzereingaben zu erzeugen, wird auch der von der Komponente erzeugte DOM-Baum in ein Observable eingebettet.

```
const container = document.createElement('div');
document.body.appendChild(container);

const logicDomTree$ = logicComponent();
logicDomTee$.subscribe((mostRecentDom) => {
  replaceChild(container, mostRecentDom);
});
```

Quelltext 4.11: Kontinuierliche Aktualisierung des DOM-Knotens einer Komponente

In Quelltext 4.11 gibt die `logicComponent()` keinen einfachen DOM-Baum zurück, sondern ein Observable, welches den Zustand des Teilbaumes der Logikkomponente über die Zeit darstellt. An diesem `logicDomTree$ Observable` wird eine Beobachterprozedur angemeldet, welche jedesmal ausgeführt wird, wenn sich für die Komponente ein neuer Teilbaum ergibt. Diese Prozedur ersetzt dann jedes Mal den Inhalt des `container` Elements mit dem aktuellsten Teilbaum.

Tatsächlich ändern sich die von den Komponenten erzeugten DOM-Bäume über die Zeit nicht wesentlich. Daher ist es sinnvoll, nicht für jeden neu erzeugten Teilbaum einer Komponente, den gesamten vorherigen Teilbaum zu ersetzen, sondern nur die Menge an Unterschieden zu ermitteln, um den bestehenden Teilbaum in möglichst wenigen einfachen Schritten in den Zustand des aktuellen Teilbaumes zu überführen. Dieses Vorgehen hat zwei Vorteile:

1. Zum einen kann dadurch der Rechenaufwand der Browsers reduziert werden, weil er nicht das Layout aller Elemente neu berechnen muss, sondern nur das Layout der von den Änderungen betroffenen Elementen.

2. Zum anderen können vom Browser verwaltete Zustände, wie die Fokussierung eines Textfeldes oder die Auswahl eines Textabschnitts, erhalten bleiben, wenn die jeweiligen Elemente von der Änderung nicht betroffen sind.

```
import {diff, patch} from 'virtual-dom';

const container = document.createElement('div');
document.body.appendChild(container);

const logicDomTree$ = logicComponent();
logicDomTee$.startWith([]).pairwise().subscribe(
  ([previousTree, currentTree]) => {
    const patches = diff(previousTree, currentTree);
    patch(container, patches)
  }
);
```

Quelltext 4.12: Kontinuierliche Diff-Berechnung der DOM-Knoten einer Komponente

Quelltext 4.12 zeigt, wie eine minimale Aktualisierung des DOM-Baumes funktionieren kann. Mittels des `pairwise()`-Operators wird das `Observable`, welches die Teilbäume enthält, in ein `Observable` transformiert, welches die Tupel aus zwei aufeinander folgenden Teilbäumen enthält. Die Beobachterprozedur erhält diese Tupel als Parameter und berechnet mit Hilfe der `diff()`-Funktion die Unterschiede der aufeinanderfolgenden Bäume. Die `diff()`-Funktion liefert als Resultat eine Liste aus nötigen Änderungen (`patches`), welche dann mittels der `patch()`-Prozedur auf das `container`-Element angewendet werden. Damit auch für den ersten Teilbaum, der von `logicDomTree$` emittiert wird, ein sinnvolles Tupel gebildet werden kann, wird `logicDomTree$` mit dem `startWith()`-Operator ein zusätzliches Element vorangestellt.

Für das Vergleichen von DOM-Teilbäumen gibt es eine Vielzahl von verfügbaren Bibliotheken, die sowohl das Ermitteln, als auch das Umsetzen der nötigen Änderungen (`patches`) übernehmen. Eine dieser Bibliotheken ist `virtual-dom`, welche auch im Rahmen dieser Arbeit verwendet wird.[ESCH 2013b]

In Abbildung 4.2 auf der nächsten Seite ist beispielhaft dargestellt, wie der Vergleich zweier ähnlicher Bäume zu einem Ergebnis führt.

Der vollständige Vergleich zweier Bäume ist eine Operation mit einer Laufzeitkomplexität von $\mathcal{O}(n^3)$, wobei n die Anzahl der Knoten ist. Dies ist besonders für häufige Änderungen umfangreicher DOM-Bäume nicht praktikabel. Darum werden zwei Bäume nur jeweils ebenenweise verglichen. Mit dieser Heuristik kann die Zeitkomplexität eines der Vergleichsoperation auf $\mathcal{O}(n)$ reduziert werden.[CHEDEAU 2013]

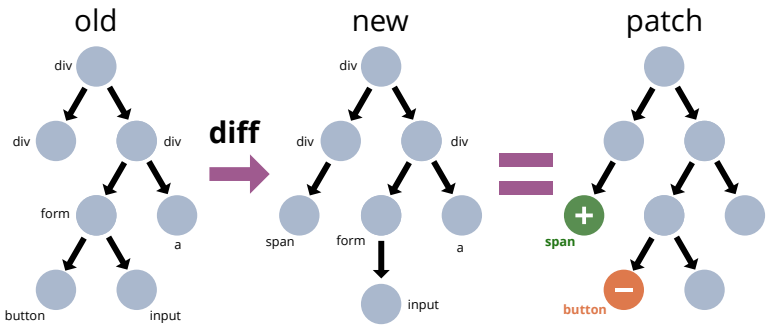


Abbildung 4.2: Der Vergleich zweier DOM-Bäume

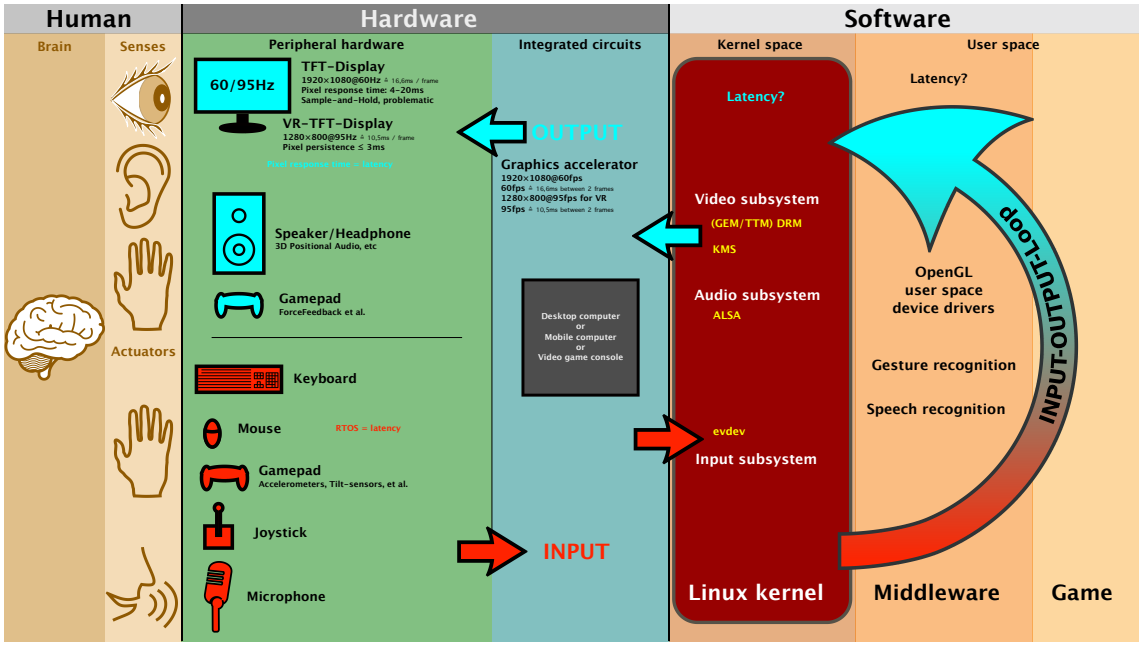
Die Objekte, mit denen der Browser die HTML-Elemente eines Dokumentes in der DOM-API repräsentiert, sind sehr umfangreich. Sie bieten eine Menge an Attributen und Methoden zur Sondierung und Manipulation ihres Zustandes. Unter der Verwendung eines virtuellen DOMs ist es nicht vorgesehen, dass Komponenten, die von ihnen erzeugen Elemente nachträglich manpulieren. Soll etwas am DOM geändert werden, wird einfach ein komplett neuer Baum erzeugt und durch die Vergleichs- und Patchprozedur geschickt. Für diesen Zweck stellen die umfangreichen, vom Browser bereitgestellten, DOM-Objekte eine unnötige Speicherlast dar. Darum werden für die Erzeugung der virtuellen DOM-Bäume einfache — prototyplose — JavaScript-Objekte verwendet. Diese können mittels einer vom Virtual DOM bereitgestellten Domänenspezifischen Sprache (DSL), die als JavaScript-Funktionen definiert ist, erzeugt werden.[TARR 2015][ESCH 2013a]

4.5 CycleJS-Architektur

Im vorherigen Abschnitt wurde erklärt, wie der `Observable`-Datentyp es erlaubt, Komponenten, die asynchrone Berechnung auf Basis von Benutzereingaben durchführen, als Funktionen zu definieren und wie die Verwendung eines Virtuellen DOMs es den Komponenten auch erlaubt, eine DOM-Repräsentation ihrer Daten zu erzeugen, ohne den globalen DOM direkt zu modifizieren, sondern die tatsächliche DOM-Modifikation nach außen zu delegieren. Im Folgenden wird die CycleJS-Bibliothek vorgestellt, die aufbauend auf RxJS und Virtual DOM ein Entwurfsmuster zur Definition von Komponenten definiert.

Mensch-Computer-Interaktion

In dem Fachgebiet der Mensch-Computer-Interaktion wird die Schnittstelle zwischen Mensch und Computer als ein zyklischer Informationsfluss beschrieben. In Abbildung 4.3 auf der nächsten Seite ist dieser Informationsfluss am Beispiel des Linux-



[ScotXW 2014]

Abbildung 4.3: Der zyklische Informationsfluss zwischen Mensch und Computer

Kernels abgebildet. Über Eingabegeräte, wie Maus oder Tastatur, ist es dem Benutzer möglich, Signale an den Computer zu senden. Diese werden von der Hard- und Software des Computers verarbeitet und in ein Ergebnis umgewandelt, welches über die Ausgabegeräte, wie Monitor oder Lautsprecher, an den Benutzer gesendet werden. Dieser hat daraufhin wieder die Möglichkeit, auf diese Ausgabe zu reagieren und mittels der Eingabegeräte neue Signale an den Computer zu senden. So entsteht ein kontinuierlicher Interaktionszyklus zwischen Benutzer und Computer.

Die CycleJS-Bibliothek fasst die beiden Konzepte des Virtuellen DOMs und des `Observable`-Datentyps zu einem Entwurfsmuster zusammen, welches diesen, in der Mensch-Computer-Interaktion beschriebenen, Informationsfluss zwischen Benutzer und Computer widerspiegelt. CycleJS trennt das zu entwickelnde System hierfür konzeptionell in zwei Teile:

1. Die Anwendung (application)

2. Die Treiber (drivers)

Die Anwendung ist, wie der Name schon sagt, die Funktion, die das Verhalten der zu entwickelnden Applikation bzw. Komponente beschreibt. Die Treiber stellen den Zustand und das Verhalten der Außenwelt dar. Sie nehmen die Rolle des Benutzers ein. In Abbildung 4.3 sind die Applikation am rechten Rand und die Treiber links davon einzuordnen.

Eine CycleJS-Anwendung ist eine Funktion, die als Parameter die Daten von Treibern als `Observable`s entgegennimmt (z.B. Mausbewegungen) und als Rückgabewert Daten — ebenfalls in Form von `Observable`s — liefert, die von den Treibern verarbeitet werden können (z.B. einen neuen virtuellen DOM-Baum).

Treiber hingegen sind Prozeduren, die als Parameter die Rückgabewerte einer Anwendung akzeptieren, die nötigen Änderungen ausführen (z.B. die Aktualisierung des DOMs) und als Rückgabewert die Daten liefern, die von der Anwendung verarbeitet werden sollen (z.B. `mousemove`-Ereignisse).

Somit bilden Anwendung und Treiber zwei ineinander greifende Bestandteile. Die Aufgabe der CycleJS-Bibliothek besteht darin, den Zyklus zu initialisieren, indem die Anwendungs-Funktion mit den Rückgabewerten der Treiber-Funktion als Parameter aufgerufen und die Treiber-Funktion mit den Rückgabewerten der Anwendungsfunktion aufgerufen wird.

Eine seltsame Schleife

```
1  import {Subject} from 'rx';
2
3  const run = (driver, app) => {
4    const proxy = new Subject(); //
5
6    const input = driver(proxy); //
7    const output = app(input); //
8
9    output.subscribe(proxy); //
10 };
```

Quelltext 4.13: Die zyklische Verbindung zweier Funktionen

Diese Initiierung scheint zunächst unmöglich zu sein, da jede der beiden Funktionen als Parameter den Rückgabewert der jeweils anderen Funktion benötigt. Da es sich aber bei den Parametern und Rückgaben um Werte des `Observable`-Datentyps handelt, welcher einen asynchronen Datenfluss modelliert, kann diese Abhängigkeitsschleife mit einem zunächst leeren Proxy-Objekt aufgelöst werden. In Quelltext 4.13 wird eine stark vereinfachte Implementation der CycleJS-Bibliothek gezeigt. `app` und `driver` sind zwei Funktionen, die jeweils ein Observable als Parameter erwarten und ein Observable zurück liefern. Einer der beiden Funktionen muss zuerst aufgerufen werden. In Zeile 6 wird zunächst die `driver` Funktion aufgerufen. Damit ein Wert für den benötigten Parameter angegeben werden kann, wird in Zeile 4 ein `Subject` als Proxy erzeugt. Ein `Subject` ist ein leeres `Observable`, welches sich durch ein anderes `Observable` be-

füllen lässt, indem es zusätzlich die Rolle eines Beobachters (`Observer`) annimmt. Nachdem die `driver` Funktion aufgerufen wurde, kann ihr Rückgabewerte `input` in Zeile 7 als Parameter verwendet werden, um die `app` Funktion aufzurufen. Letztendlich wird der Kreis geschlossen, indem der Rückgabewert `output` der `app`-Funktion verwendet wird, um das `proxy`-Objekt zu befüllen.

```
1  import {Observable as O} from 'rx';
2  import {button} from '@cycle/dom';
3
4  const exampleApp = ({DOM, HTTP, audio}) => {
5    const myButton = button('.my-button', ['Click me']);
6    const click$ =  DOM.select('.my-button').events('click');
7    const request$ = click$.map(() => '/some/url');
8    const midi$ = HTTP.mergeAll().map(() => sineWave(200));
9
10   return {
11     DOM: O.just(myButton),
12     HTTP: request$,
13     audio: midi$,
14   };
15 };
```

Quelltext 4.14: Eine einfache CycleJS-Anwendung

Die tatsächliche Implementation der CycleJS-Bibliothek ist noch ein wenig, jedoch nicht wesentlich, umfangreicher, weil noch einige Sonderfälle beachtet werden müssen. Zudem muss eine CycleJS-Anwendung nicht aus einer Funktion mit nur einem einzigen Parameter bestehen, sondern kann mehrere Observables von unterschiedlichen Treibern entgegennehmen und auch über mehrere, in einer HashMap verpackte, Rückgabewerte, Ausgabeströme für mehrere Treiber bereitstellen. In Quelltext 4.14 ist zu sehen, wie eine CycleJS-Anwendung als Funktion definiert ist, die Datenströme von drei verschiedenen Treibern (`DOM`, `HTTP`, `audio`) als Parameter erwartet und als Rückgabewerte Daten für ebenfalls diese drei Treiber liefert. Sowohl Parameter, als auch Rückgabewerte sind als Hashmap kodiert.

Die `exampleApp`-Anwendung macht gebrauch von drei Treibern: `DOM`, `HTTP` und `audio`. Das ist daran zu erkennen, dass sowohl das Objekt, welches die Funktion als Parameter bekommt, als auch das Objekt, welches sie zurückgibt, die drei Attribute `DOM`, `HTTP` und `audio` hat.

Die Funktion `button` wird verwendet, um ein Button-Objekt mit der Beschriftung „Click me“ zu erzeugen. Per `O.just` wird ein `Observable` erzeugt, welches diesen Button als einzigen Wert enthält. Dieses `Observable` wird unter dem Schlüssel `DOM` in dem, per `return` zurückgegebenen, Objekt abgelegt und somit an den DOM-Treiber gege-

ben. Hiermit wird erreicht, dass der Button in das Dokument eingefügt, also dargestellt, wird.

Das `DOM`-Objekt, welches die Funktion als Parameter bekommt, stellt Datenströme für die DOM-Ereignisse bereit. Mittels den Methoden `DOM.select` und `events` wird auf den Datenstrom zugegriffen, welcher alle Mausklick-Ereignisse des Buttons mit der CSS-Klasse `my-button` enthält. Mittels `map`-Operator wird der Datenstrom an Mausklicks in ein `Observable` transformiert, welches Strings enthält, die URLs darstellen. Dieser Datenstrom wird über den `HTTP`-Schlüssel an den HTTP-Treiber gegeben, wodurch dieser veranlasst wird, HTTP-Anfragen für die gegebene URL zu senden. Diese Netzwerkanfragen werden asynchron verarbeitet.

Unter dem `HTTP`-Schlüssel im Parameterobjekt wird der `exampleApp`-Funktion der Datenstrom zur Verfügung gestellt, welcher die HTTP-Antworten des Servers enthält. Das `HTTP`-Objekt ist ein `Observable`, dessen Inhalt jeweils selbst `Observable`s sind. Jedes innere `Observable` stellt eine HTTP-Anfrage dar und enthält die zugehörige Antwort des Servers als Objekt. Der `mergeAll`-Operator verringert die Tiefe der Verschachtelung und bildet ein `Observable`, welches alle Antworten des Servers enthält. Der Strom an HTTP-Antworten wird zu einem `Observable` transformiert, welches für jede HTTP-Antwort ein per `sineWave` Funktion gebildetes Objekt enthält. Dieses Objekt repräsentiert einen 200 millisekunden langen Piepton, welcher vom `audio`-Treiber ausgegeben werden kann.

Treiber

Da CycleJS Treiber, genau wie CycleJS-Anwendungen, einfache JavaScript-Funktionen sind, die `Observable`s als Parameter akzeptieren und `Observable`s als Rückgabewert liefern, lassen sie sich mit wenig Aufwand definieren. Allerdings stellt CycleJS viele Treiber schon von Haus aus bereit, sodass für die Entwicklung einer Anwendung oder einer Komponente nur sehr selten auch noch ein Treiber geschrieben werden muss. Grundsätzlich sollen Treiber nicht spezifisch für die Domäne einer Anwendung konzipiert sein, sondern eine Brücke zu prozeduralen Programmierschnittstellen des Browsers darstellen.

So gibt es unter anderem den `DOMDriver` für die Interaktion mit dem DOM, den `HTTPDriver` für das Senden von HTTP-Requests oder den `LocalStorageDriver`, welcher den Zugriff auf einer persistente HashMap des Browsers erlaubt.[STALTZ 2015d][STALTZ 2015a][LECHELT 2015]

Model-View-Intent

Da eine CycleJS-Anwendung eine einfache Funktion ist, kann diese mittels Funktionaler Dekomposition problemlos in beliebig kleinere Fragmente zerlegt werden. Eine Teilung der Funktion in drei spezielle Unterfunktionen mit spezifischen Aufgaben hat sich bewährt und wird auch in der Dokumentation von CycleJS als Entwurfsmuster unter dem Namen Model-View-Intent (MVI) empfohlen. Der Name ist an das Entwurfsmuster Model-View-Controller (MVC) angelehnt, hat aber inhaltlich nicht viel mit ihm gemeinsam.[STALTZ 2015b][FOWLER 2006][GAMMA et al. 1995]

Besser vergleichbar ist das MVI-Muster mit dem Eingabe-Verarbeitung-Ausgabe-Prinzip (EVA-Prinzip) der Datenverarbeitung. Sinnvoller wäre es, die Buchstabenfolge MVI um eine Stelle nach rechts zu rotieren, um IMV zu erhalten, denn das Muster beschreibt die Verarbeitung von Daten in drei Schritten:

1. Als erstes wird den Eingaben des Benutzers eine domänenbezogene Absicht (Intent) zugeordnet. So wird beispielsweise der Mausklick auf einen Button als der Wunsch, eine Aktion durchzuführen interpretiert.

2. Als nächstes wird aus der Summe aller Aktionen der momentane Zustand der Anwendung modelliert (Model).

3. Dieser Zustand wird über die Benutzeroberfläche angezeigt (View) bzw. ausgegeben.

Diese drei Schritte lassen sich gut als drei eigenständige Funktionen modellieren, die dann miteinander verknüpft werden können. In Quelltext 4.15 auf der nächsten Seite ist zu sehen, wie die Funktion `mviExample` in drei Funktionen zerlegt wurde. Die `intent`-Funktion extrahiert aus dem, vom DOM-Treiber zur Verfügung gestellten, Objekt den Datenstrom an Mausklicks und transformiert diesen in einen Datenstrom aus `1`en, welcher unter dem Schlüssel `increment$` in einer Hashmap abgelegt wird. `increment$` stellt die Aktion dar, einen Zähler um einen Wert (hier um `1`) zu erhöhen. Die Hashmap an Aktionen, die von `intent` erzeugt wurde, wird von der Funktion `model` zu dem Datenstrom (`Observable`), der den aktuellen Zählerstand enthält, transformiert. Die `view` Funktion transformiert die Zählerstände in DOM-Bäume, welche letztendlich zur Verarbeitung an den DOM-Treiber gegeben werden.

Das MVI-Muster erlaubt es so, schnell zu erfassen, welche Aktionen der Benutzer durchführen kann, wie sich diese Aktionen auf die Anwendung auswirken und wie sich die grafische Oberfläche der Anwendung zusammensetzt.

```
import {div} from '@cycle/dom';

const intent = (DOM) => {
  return {
    increment$: DOM.events('click').map(() => 1),
  };
};

const model = ({incremen$}) => {
  return increment$.scan(
    (acc, delta) => acc + delta,
    0
  );
};

const view = (state$) = {
  return state$.map((clickCount) =>
    div(`You have clicked ${clickCount} times`)
  );
};

const mviExample = ({DOM}) => {
  return {
    DOM: view(model(intent(DOM))),
  };
};
```

Quelltext 4.15: Die Zerlegung einer CycleJS-Anwendung mittels MVI-Muster

5 Implementierung der Komponenten

Im Folgenden wird die konkrete Implementierung der einzelnen Komponenten vorgestellt. Die Menge der hier vorgestellten Komponenten unterscheidet sicht leicht von den in der Einleitung genannten Liste. Das hat den Grund, dass sich im Laufe der Entwicklung Gemeinsamkeiten zwischen einigen Komponenten ergeben haben, die sich in zusätzliche Subkomponenten zusammenfassen liessen. So ist unter anderem eine Komponente zum interaktiven Betrachten von SVG-Dokumenten entstanden.

5.1 Karnaugh-Veitch-Diagramm-Editor

Die Karnaugh-Veitch-Diagramm-Editor-Komponente (KV-Editor) soll das Erzeugen, Bearbeiten und Veranschaulichen von Karnaugh-Veitch-Diagrammen (KV-Diagrammen) ermöglichen. KV-Diagramme dienen zur Minimierung Boole'scher Funktionsterme. Es können sowohl disjunktive (DNF) als auch konjunktive Minimalformen (KNF) von Funktionen erzeugt werden.[VEITCH 1952]

Die Anforderungen

Als Orientierung für diese Komponente soll ein Werkzeug dienen, welches 1998 von Matthias Meyer am Fachbereich Informatik an der Universität Hamburg entwickelt wurde. Dieses Werkzeug wurde von Meyer als Java-Applet entwickelt und erlaubt das Erzeugen von KV-Diagrammen für Funktionen mit bis zu 6 Parametern und bis zu 8 Ausgabewerten.[MEYER 1998]

In Abbildung 5.1 auf der nächsten Seite ist das Hauptfenster der Benutzeroberfläche des herkömmlichen KV-Diagram-Applets zu sehen. Mit den Radiobuttons am oberen Fensterrand, kann zwischen den verschiedenen Bearbeitungsmodi umgeschaltet werden:

1. Der **Edit function**-Modus erlaubt das Ändern der Funktionswerte in den einzelnen Zellen per Mausklick. Mögliche Funktionswerte sind *1*, *0* und * *(Dont-care)*. Per Mausklick kann zyklisch durch die möglichen Werte geschaltet werden. Drückt der Benutzer beim Klick die ALT-Taste, wird die Richtung des Zyklus umgekehrt.

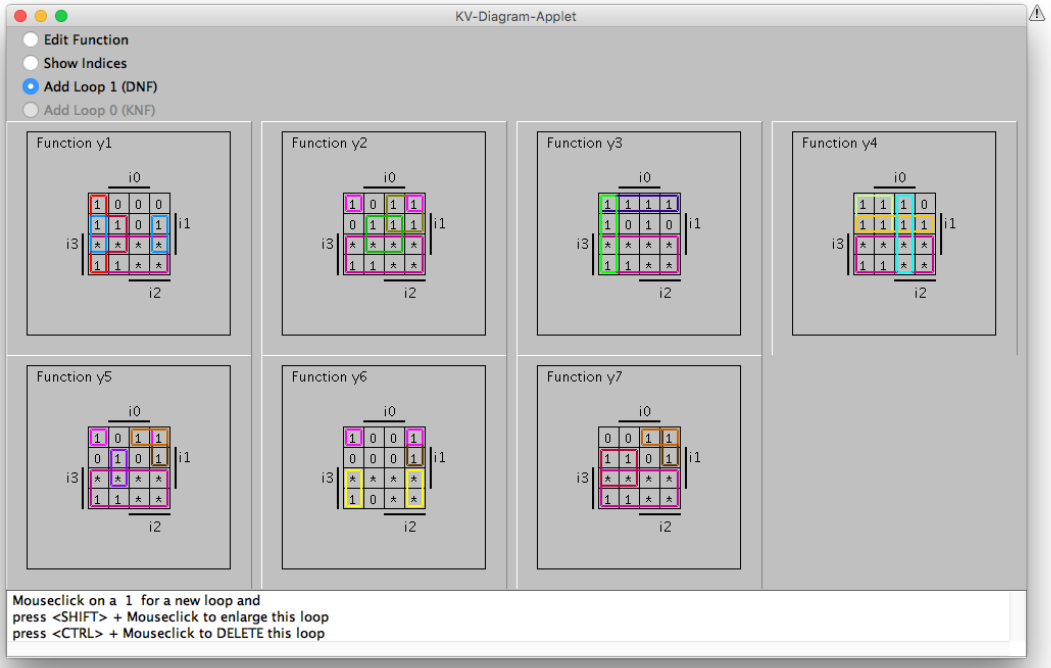


Abbildung 5.1: Das Hauptfenster des KV-Diagramm-Applets von Matthias Meyer

2. Im **Show indices**-Modus werden anstelle der Funktionswerte in den Zellen die Indizes der Zeilen Funktionstabelle angezeigt. Diese Darstellung dient der Verdeutlichung des Aufbaus eines KV-Diagramms.

3. Der **Add Loop 1 (DNF)**-Modus erlaubt das Bilden von Schleifen (*loops*), um Felder mit einem Funktionswert von *1* oder * um eine disjunktive Minimalform der Funktion zu erzeugen. Die erzeugten Schleifen werden als bunte Rechtecke innerhalb des Rasters der Funktionswerte dargestellt. Jede Schleife bekommt eine zufällige Farbe zugewiesen. Die genauen Regeln, nach denen eine solche Schleife erzeugt werden kann, bilden das Kernkonzept der KV-Diagramme. Hierauf wird im Abschnitt *Das Datenmodell* genauer eingegangen. Um eine neue Schleife zu erstellen, muss mit der Maus ein erlaubter Funktionswert im Raster angeklickt werden. Der Klick erzeugt eine kleine Schleife, die nur die ausgewählte Zelle umfasst. Während die *Umschalt*-Taste gedrückt gehalten wird, können nun zusätzliche Felder im Raster angeklickt werden, um die Schleife zu vergrößern. Wird ein Feld angeklickt, ohne dass die *Umschalt*-Taste gehalten wurde, wird eine neue Schleife erzeugt. Eine Schleife kann gelöscht werden, indem sie bei gehaltener *Ctrl*-Taste angeklickt wird.

4. Der **Add Loop 0 (KNF)**-Modus erlaubt, genau wie der vorherige Modus, das Bilden von Schleifen — allerdings mit dem Unterschied, dass die KNF-Schleifen nur Funktionwerte von *0* oder * beinhalten dürfen.

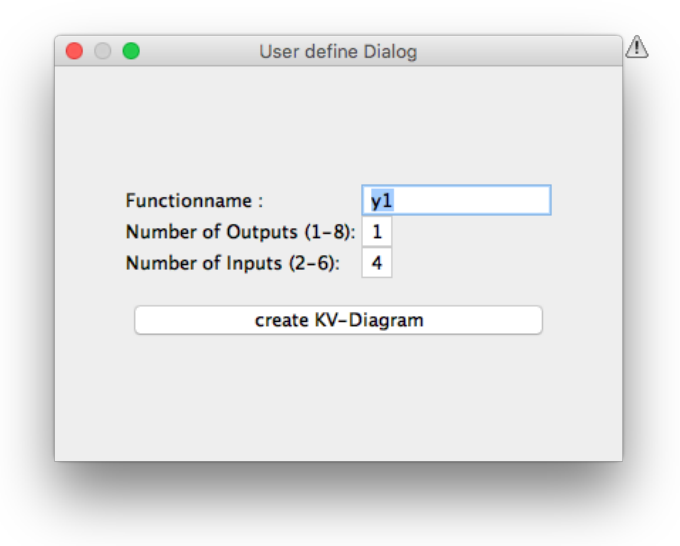


Abbildung 5.2: Das Dialogfenster des KV-Diagramm-Applets

Wie in Abbildung 5.2 zu sehen, kann die Anzahl der Eingabe- und Ausgabeparameter der Funktion und die damit verbundenen Größe des KV-Diagramms vom Benutzer über ein Dialogfenster gesteuert werden. Bei der Änderung dieser Parameter wird ein komplett neues KV-Diagramm erzeugt. Das bedeutet, alle zuvor gesetzten Funktionswerte und erzeugten Schleifen gehen verloren. Anstatt eine Funktion komplett selbst zu definieren, können aber auch vordefinierte Beispieldaten in das Applet geladen werden.

Den Fortschritt bei der Minimierung der Funktion kann der Benutzer in einem separaten Fenster des Applets verfolgen: Wie in Abbildung 5.3 auf der nächsten Seite zu sehen ist, wird das Schaltnetz, welches nötig ist, um die Funktion in einem Programmable Logic Array (PLA) umzusetzen, grafisch dargestellt. Zu Beginn, wenn noch keine Schleifen erzeugt wurden, wird im DNF-Modus für jede Zelle mit einem Funktionswert von 1 ein UND-Gatter benötigt. Im KNF-Modus wird für jede Zelle mit einer 0 ein ODER-Gatter benötigt. Über das Bilden der Schleifen lässt sich die Anzahl der benötigten Gatter reduzieren, da mehrere Funktionswerte, also auch mehrere Gatter, zusammengefasst werden. Die zusammengelegten Gatter werden in der grafischen Darstellung der Farbe der Schleife eingefärbt, durch die sie entstanden sind. So lässt sich sehr gut der Zusammenhang zwischen den Arbeitsschritten im KV-Diagramm und dem daraus resultierenden Ergebnis erkennen.

Schließlich erlaubt das Applet noch den Export des Ergebnisses als Text, welcher die grafisch dargestellte PLA-Konfiguration repräsentiert, wie in Abbildung 5.4 auf Seite 45 zu sehen ist.

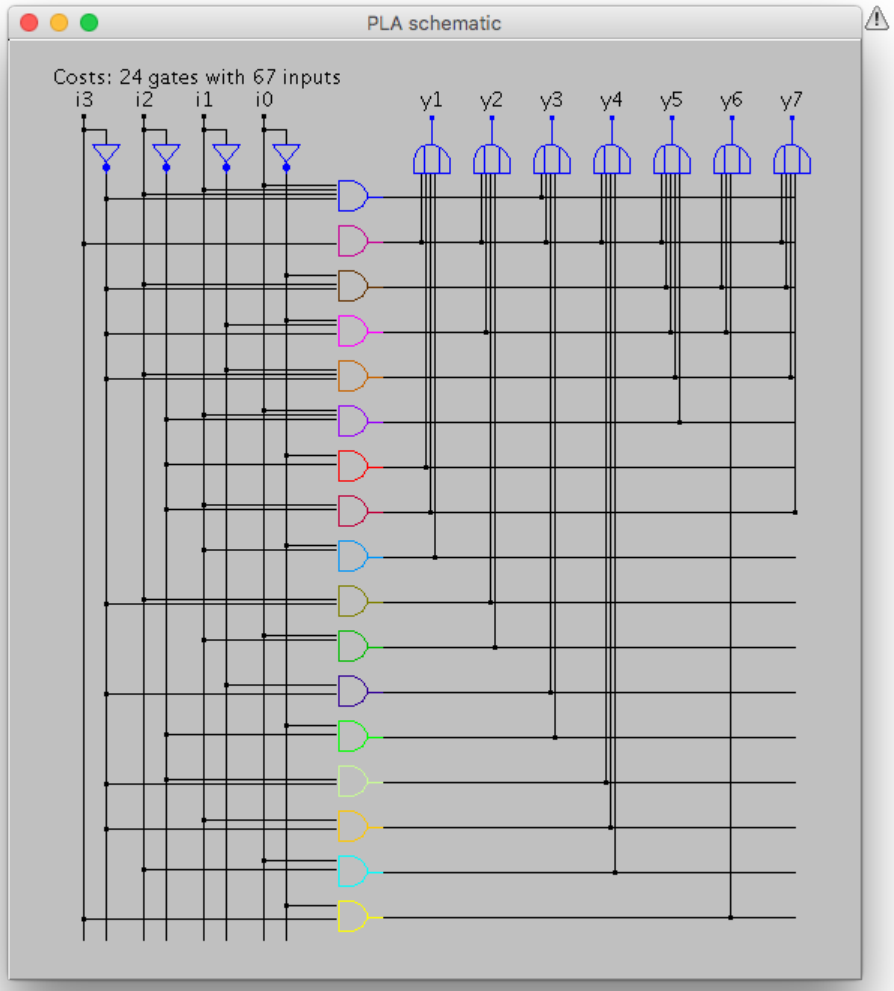


Abbildung 5.3: Die minimierte Funktion als Schaltnetz eines PLA

Diese grundlegenden Funktionen des Java-Applets sollen in die neue KV-Diagramm-Komponente übernommen werden.

Übertragung auf die Entwicklung der Komponente

Nicht alle Funktionen des Java-Applets können direkt übernommen werden, da sie nicht vollständig mit den Anforderungen dieser Arbeit kompatibel sind:

1. Für die Erzeugung der Schleifen macht das Applet Gebrauch von den Modifikatortasten *Umschalt* und *Ctrl*. Diese Tasten stehen auf einem Tablet-Computer nicht zur Verfügung

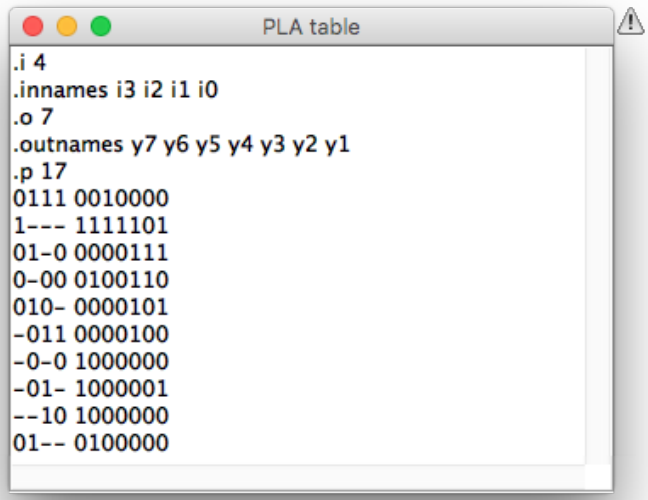


Abbildung 5.4: PLA Export des alten KV-Diagramm-Applets

2. Das Applet benutzt mehrere Fenster, die auf einem Desktop-Computer nebeneinander positioniert werden können, um alle Informationen auf einen Blick zugänglich zu haben. Ein Webbrowser kann zwar mehrere Fenster als sogenannte Popup-Fenster öffnen, jedoch können diese auf einem Tablet-Computer nicht komfortabel nebeneinander positioniert werden. Auch auf einem Desktop-Computer ist die Verwendung mehrererer Browser-Fenster kein übliches Bedienkonzept für Webanwendungen.

3. Das Applet stellt die KV-Tabellen mit einer Zellengröße von 15×15 Pixeln dar. Auf einem Touchscreen liesse sich nicht präzise mit einzelnen Zellen interagieren, da eine Fingerspitze mehrere Zellen auf einmal verdecken würde. Eine simple Vergrößerung der Zellen führt dazu, dass ein Diagramm für eine Funktion mit sieben Ausgangswerten zu groß wird, um es vollständig auf einem Bildschirm darzustellen.

Neben diesen rein technischen Problemen bietet die Neuentwicklung des KV-Diagramm-Editors auch noch die Möglichkeit, das gesamte Benutzererlebnis zu verbessern:

1. Das Applet trennt die Erzeugung eines KV-Diagramms in einen eigenständigen Dialog, sodass der Zusammenhang zwischen der Anzahl der Funktionsparameter und der Größe des Diagramms vom Benutzer nur indirekt erlebt werden kann. Da der primäre Zweck der Komponente ein Lerneffekt sein soll, wäre es sinnvoll, wenn die Änderung der Größe des Diagramms in Abhängigkeit zur Anzahl der Funktionsparameter vom Benutzer unmittelbar erlebt werden kann.

2. Die Umschaltung der Bearbeitungsmodi im Applet führt stets dazu, dass die schon angelegten Schleifen wieder entfernt werden. Technisch ist das keine Notwendigkeit. Eine Schleife muss nur dann zwingend gelöscht werden, wenn sich ein Funktionswert, den sie beinhaltet, ändert. Das Verhalten des Applets führt dazu, dass es zum Beispiel nicht möglich ist, für eine Funktion sowohl eine KNF- als auch eine DNF-Optimierung durchzuführen und die Ergebnisse zu vergleichen. Ein solcher Vergleich der Ergebnisse kann aber zum Verständnis der Zusammenhänge beitragen.

3. Auch unabhängig vom Lerneffekt, ist es vorteilhaft, eine Funktion, für die man schon einige Schleifen angelegt hat, noch nachträglich bearbeiten zu können, ohne dass die zuvor geleistete Arbeit verloren geht.

Während der Entwicklung der KV-Diagramm-Komponente ist es gelungen, auf alle genannten Problematiken einzugehen und passende Lösungen zu finden. Im Folgenden werden die wichtigsten Aspekte vorgestellt.

Bedienung auf Mobilgeräten

Die Notwendigkeit von Modifikatortasten zur Erzeugung von Schleifen kann umgangen werden, indem stattdessen eine Drag'n-Drop-Bewegung mit der Maus oder mit dem Finger verwendet wird. Anstatt zuerst eine Zelle im Diagramm anzuklicken und dann eine weitere Zelle per Umschalt-Klick auszuwählen, um sie der Schleife hinzuzufügen, kann der Mauszeiger, bei gedrückter Maustaste, von der einen zu anderen Zelle bewegt und dann losgelassen werden. Diese Bedienung lässt sich auch mit dem Finger auf einem Touchscreen ausführen:

1. Mit dem Finger wird eine Zelle berührt

2. Ohne den Finger anzuheben, wird dieser auf dem Bildschirm zur nächsten Zelle der Tabelle bewegt

3. Ist die Zielzelle erreicht, wird der Finger angehoben

Abbildung 5.5 auf der nächsten Seite zeigt, wie das Verbinden zweier Zellen per Drag'n-Drop-Geste das Bilden von unterschiedlichen Schleifen erlaubt. Jede Schleife, die in einem KV-Diagramm gebildet werden kann, lässt sich durch die Verbindung zweier Zellen miteinander beschreiben. Das hat den Grund, dass eine Boole'sche Funktion mit N Variablen sich als N-dimensionaler Hyperwürfel betrachten lässt, dessen Eckpunkte die Funktionswerte darstellen. Eine Schleife in einem KV-Diagramm ist ebenfalls ein Hy-

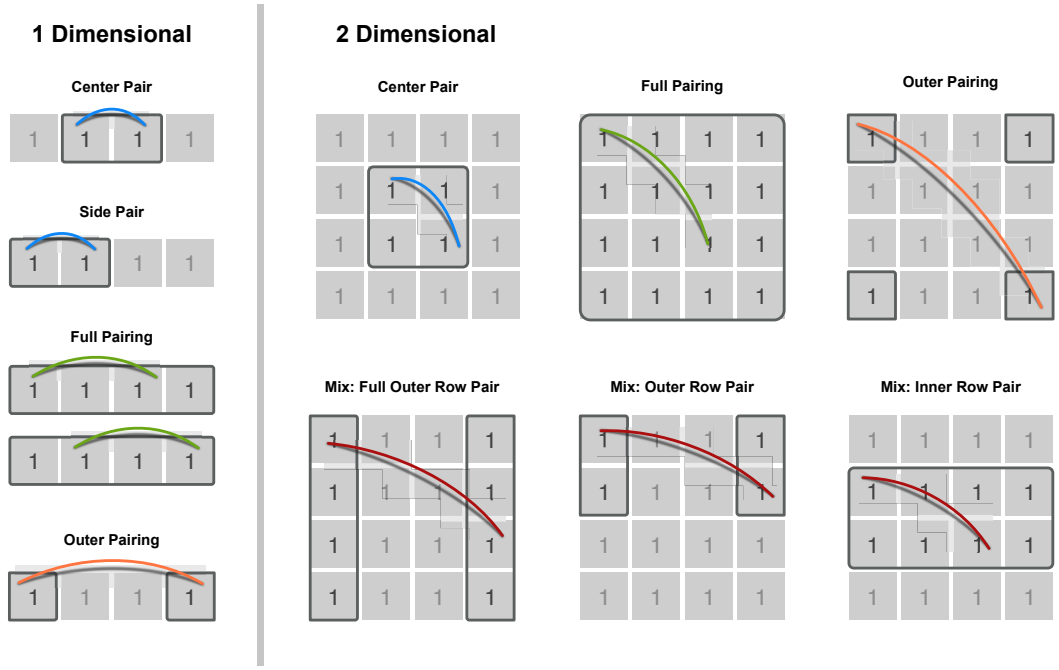


Abbildung 5.5: Bedienkonzept zur Erzeugung von KV-Schleifen

perwürfel, welcher einen Teil des Funktionswürfels bildet. Diese Betrachtungsweise von Boole'schen Funktionen liegt der Funktionsweise von KV-Diagrammen zugrunde. Da jeder Würfel sich durch zwei diagonal gegenüberliegende Eckpunkte vollständig beschreiben lässt, genügen auch stets zwei Funktionswerte bzw. zwei Zellen des KV-Diagramms zur Festlegung einer Schleife.

Grafische Benutzeroberfläche

Für die Entwicklung der grafischen Benutzeroberfläche wurden mehrere Iterationen durchlaufen. Abbildung 5.6 auf der nächsten Seite zeigt einen ersten Entwurf der Benutzeroberfläche auf einem Smartphone. Zu sehen sind die Button zum Umschalten zwischen dem DNF- und KNF-, sowie dem „Funktion-Bearbeiten"-Modus, sowie eine Liste aller existierenden Schleifen. Die Schleifen auch als Liste außerhalb des Diagramms darzustellen, sollte das Bearbeiten und Löschen von Schleifen erleichtern. Die Größe der Tabellenzellen wurde so gestaltet, dass sich diese bequem per Finger auf einem Touchscreen auswählen lassen. Das Konzept dieser Benutzeroberfläche wurde aber schnell verworfen, da ein Smartphone nicht das primäre Gerät für die Bedienung der Komponente ist.

In der nächsten Iteration (Abbildung 5.7 auf Seite 49) wurde die Benutzeroberfläche vorrangig für die Bedienung auf einem Tablet- oder Desktop-Computer ausgerichtet. Die gesonderte Auflistung der Schleifen und die Größe der Tabellenzellen wurde aus dem

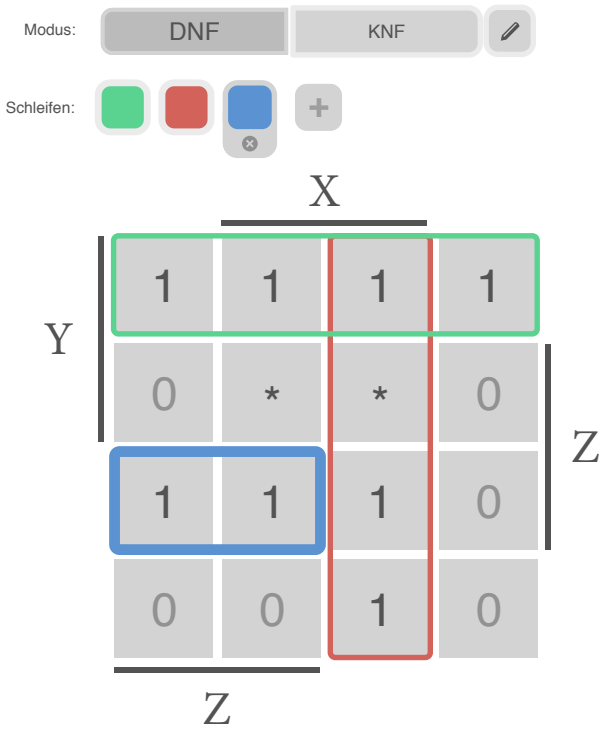


Abbildung 5.6: Entwurf 1 der KV-Komponente

vorherigen Entwurf übernommen. Der getrennte Modus für die Anzeige der Indizes und der Modus zum Bearbeiten der Funktion wurden direkt in die Hauptansicht integriert. Die Anzahl der Eingänge und Ausgänge der Funktion kann über *Plus* und *Minus* Button direkt gesteuert werden. Die Indizes werden zusammen mit den Funktionswerten in den Zellen des Rasters angezeigt. Die unterschiedlichen Ausgänge der Funktion werden als kleine Diagramme angezeigt, von denen sich nur eines zur Zeit zum Bearbeiten auswählen lässt. Diese Darstellungsform leistet zum einen eine gute Bedienbarkeit auf Touchscreengeräten, da die aktiven Zellen groß genug sind, um bequem mit einem Finger ausgewählt zu werden. Zum anderen ermöglicht sie es, alle Funktionswerte auf einen Blick zu sehen. Das ist wichtig, weil auch Schleifen über mehrere Ausgänge hinweg erzeugt werden können.

Die gemeinsame Darstellung von Indizes und Funktionswerten hat sich in der Praxis als zu unübersichtlich erwiesen. Während der üblichen Bearbeitung eines Diagramms werden die Indizes nicht benötigt. Zudem gab es in dem Konzept aus Abbildung 5.7 auf der nächsten Seite keine Möglichkeit, zwischen dem Ändern eines Funktionswertes und dem Erzeugen einer Schleife zu unterscheiden. Auch mussten noch Button für das Öffnen von Beispieldiagrammen, das Exportieren des Ergebnisses und die grafische Darstellung der Funktion als PLA-Schaltnetz untergebracht werden. Diesen Bedürfnissen widmet sich der Entwurf in Abbildung 5.8 auf Seite 50 und Abbildung 5.9 auf Seite 51. In diesem

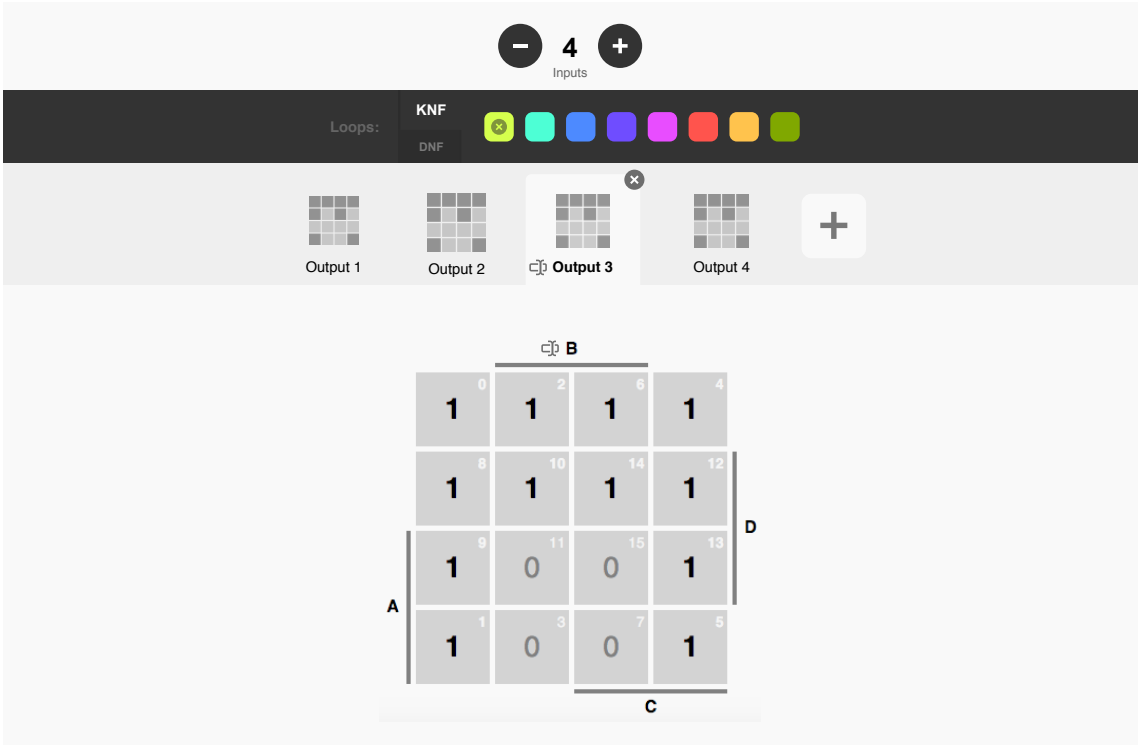


Abbildung 5.7: Entwurf 2 der KV-Komponente

Entwurf soll die Umschaltung zwischen dem Modus „Funktion Bearbeiten“ und dem Modus „Schleifen Bearbeiten“ mittels zweier Pfeilbuttons ermöglicht werden. Die Pfeilsymbolik soll hervorheben, dass das Erstellen der Schleifen zeitlich auf das Definieren der Funktion folgt. Das Umschalten zwischen DNF- und KNF-Modus ist orthogonal zu der Wahl des Bearbeitungsmodus möglich. Da es möglich sein soll, die Funktion auch nach dem Erstellen von Schleifen zu bearbeiten, ohne prinzipiell alle Schleifen zu verlieren, soll es auch im Modus „Funktion Bearbeiten“ möglich sein, die derzeitigen DNF- bzw. KNF-Schleifen zu betrachten, um nachvollziehen zu können, welche Schleifen durch das Ändern eines Funktionswertes verloren gehen. Um auszudrücken, dass die Schleifen im „Funktion Bearbeiten“-Modus jedoch nicht bearbeitet werden können, werden sie in diesem Modus halbtransparent dargestellt. Zudem sind in diesem Entwurf, oberhalb des großen Rasters, drei Button vorgesehen, die das Umschalten zwischen der Darstellung der Zellen erlauben:

1. **Value** zeigt den Funktionswert in der Zelle an
2. **Decimal Index** zeigt den Index der Zelle als Dezimalzahl an
3. **Binary Index** zeigt den Index der Zelle als Binärzahl an

Ein Button in der oberen rechten Ecke erlaubt das Ein- und Ausblenden der grafischen Darstellung des PLA-Schaltnetzes.

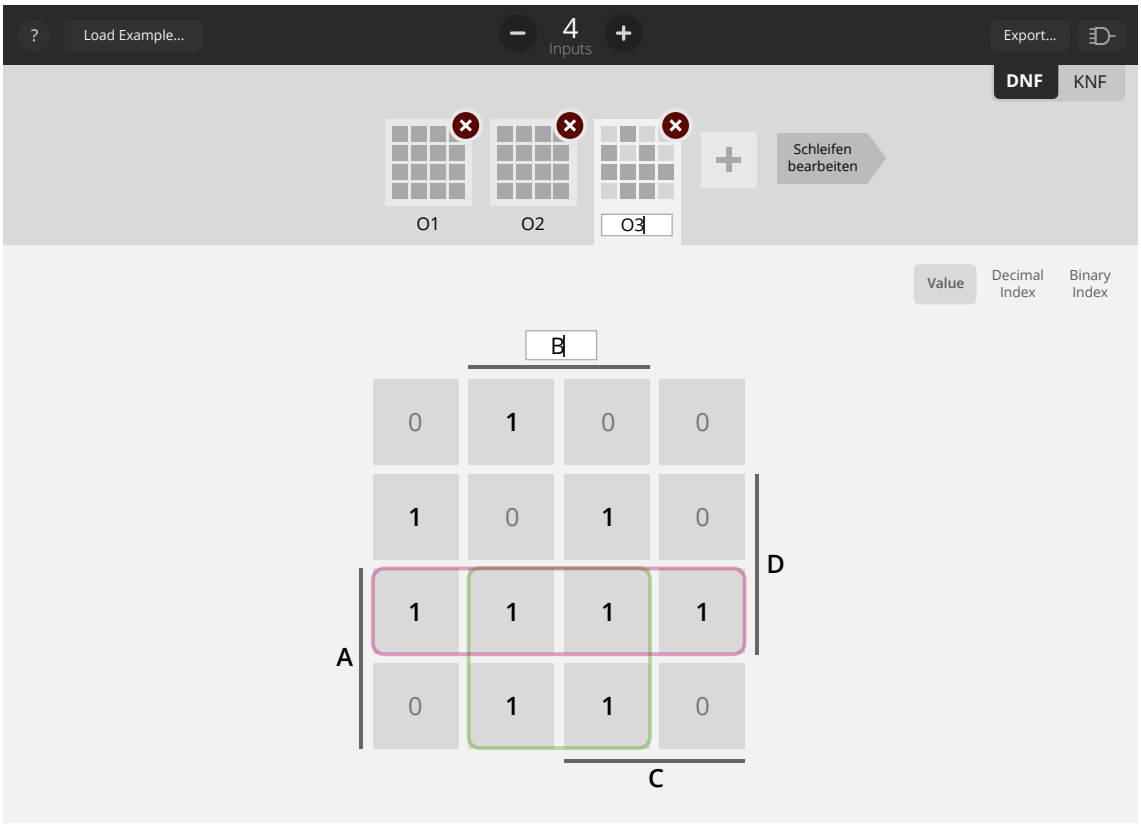


Abbildung 5.8: Entwurf 3 der KV-Komponente im Function-Modus

Die Hierarchie und Zusammenhänge der Bedienelemente im Entwurf aus Abbildung 5.8 und Abbildung 5.9 auf der nächsten Seite haben sich letztendlich als nicht besonders verständlich herausgestellt, sodass die Benutzeroberfläche in einer letzten Iteration zu dem in Abbildung 5.10 auf der nächsten Seite zu sehenden Entwurf vereinfacht wurde. Die Dialoge zum Öffnen und Exportieren von Diagrammen, sowie für die Darstellungsoptionen der Zellen, sind über einfache Icon-Button am oberen Bildschirm zu erreichen. Die Liste der Schleifen wird zusammen mit den DNF- und KNF-Button direkt oberhalb des Diagramms angezeigt. Die unterschiedlichen Funktionsausgänge sind als Registerkarten mit verkleinerten Diagrammen dargestellt.

In Abbildung 5.11 auf Seite 52 ist die finale Umsetzung des Entwurfes im Browser zu sehen. Die grafische Darstellung des PLA-Schaltnetzes nimmt die gesamte rechte Hälfte der Anwendung ein. Die Anzahl der Ein- und Ausgänge wird explizit angezeigt. Zwei große Button erlauben den Wechsel zwischen den, nun wieder auf Englisch benannten, Modi „Edit Function“ und „Edit loops“. Die Trennlinie zwischen der linken und rechten Hälfte der Anwendung lässt sich per Maus — auf einem Touchscreen per Finger — greifen und verschieben. Somit lässt sich der von der Komponente benötigte Platz individuell an die Größe des Bildschirmes anpassen.

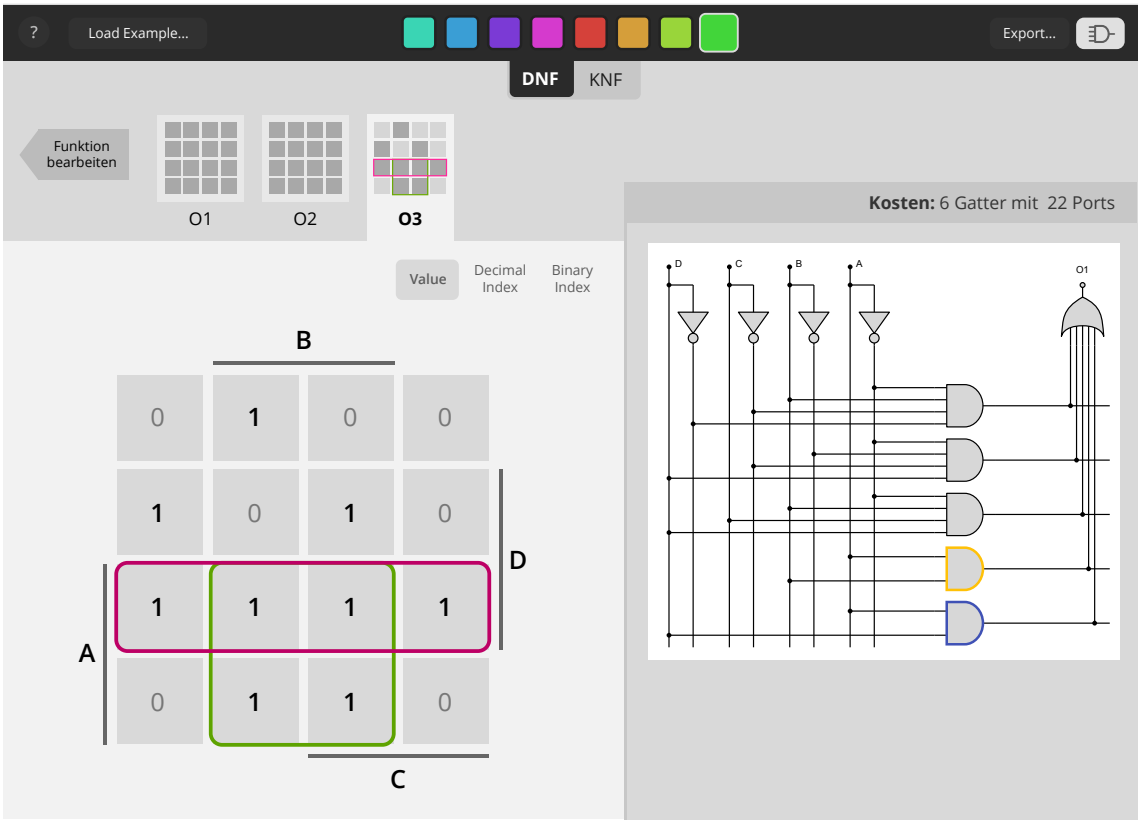


Abbildung 5.9: Entwurf 3 der KV-Komponente im Loop-Modus

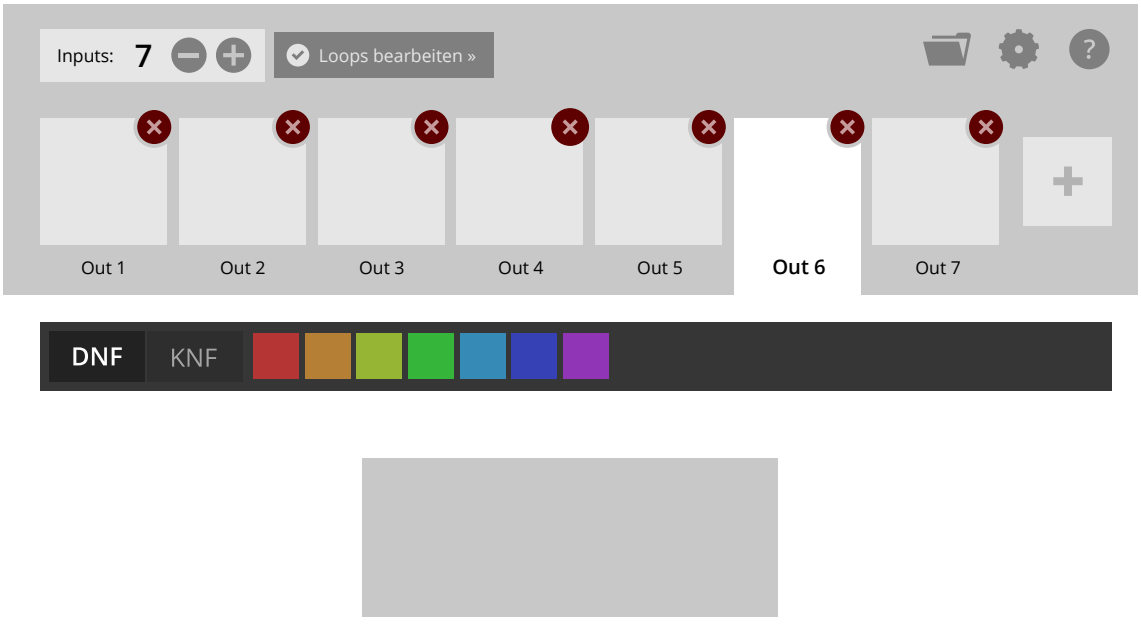


Abbildung 5.10: Entwurf 4 der KV-Komponente

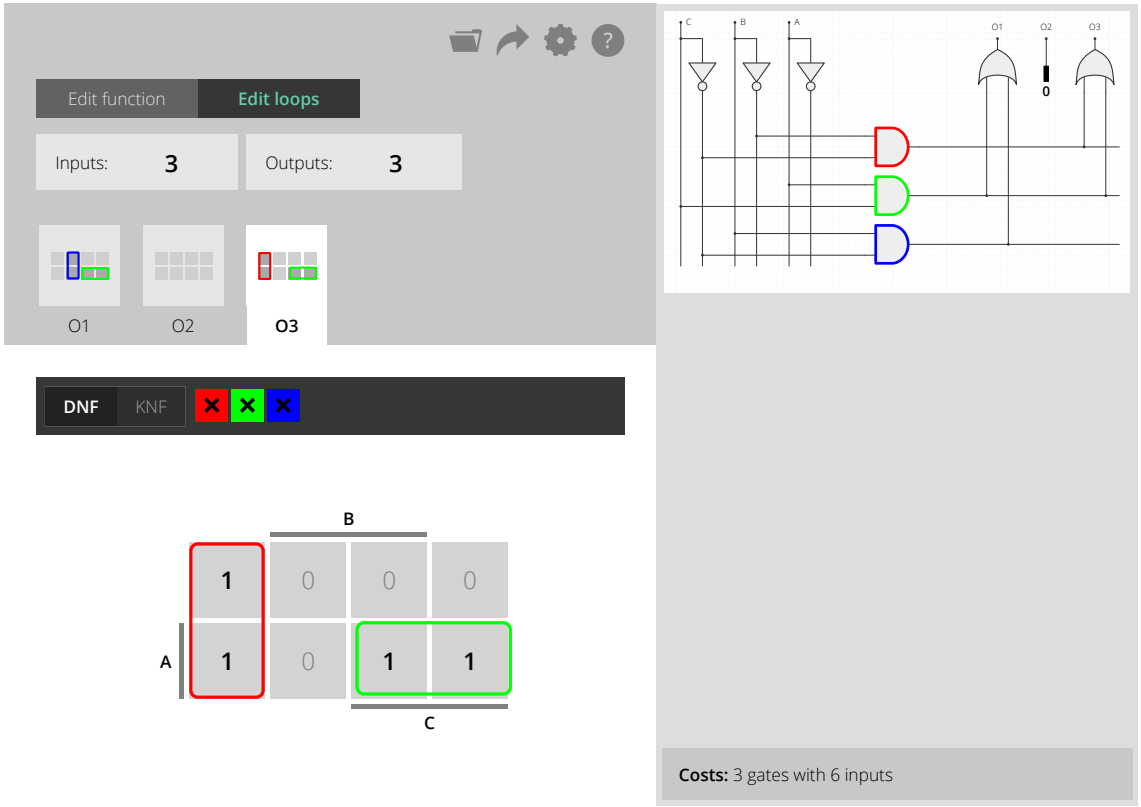


Abbildung 5.11: Implementierte Benutzeroberfläche der KV-Komponente im Browser

Das Rekursive Layout

Das Raster eines KV-Diagramms muss in unterschiedlichen Größen erzeugt werden können. Ein KV-Diagramm für eine Funktion mit N Eingängen besteht aus 2^N Zellen, weil es für jede Eingangsbelegung einen Funktionswert gibt und jeder Funktionswert in einer Zelle dargestellt wird.

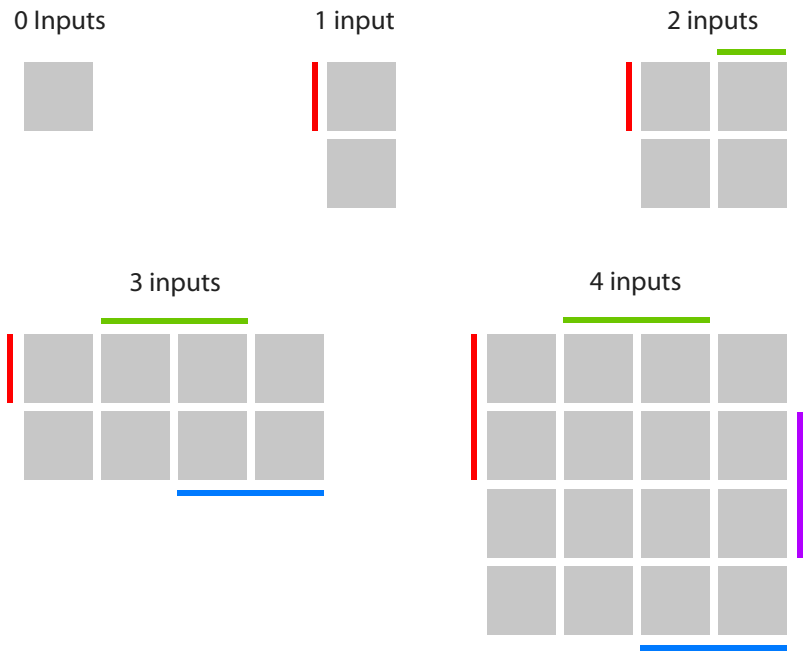


Abbildung 5.12: Das Layout von KV-Diagrammen für Funktionen mit 0 bis 4 Eingängen

In Abbildung 5.12 sind KV-Diagramme für Funktionen mit bis zu 4 Eingängen dargestellt. Die Farben der bunten Balken stehen für die verschiedenen Eingänge der Funktion. Ein *senkrechter* Balken kennzeichnet die *Zeilen* des Rasters deren Zellen sich auf eine Eingangsbelegung beziehen, in welcher der Eingang des Balkens den Wert *1* hat. Ein *waagerechter* Balken kennzeichnet entsprechende *Spalten* des Rasters.

Für Funktionen mit mehr als vier Eingängen können KV-Diagramme erzeugt werden, indem die Raster rekursiv ineinander geschachtelt werden. Dargestellt wird diese Schachtelung in Abbildung 5.13 auf der nächsten Seite. Zu erkennen ist, wie beispielweise ein KV-Diagramm für eine Funktion mit fünf Eingängen zusammengesetzt werden kann, indem zunächst ein KV-Diagramm für einen Eingang erzeugt wird und jeder der beiden Zellen mit einem KV-Diagramm für vier Eingänge befüllt bzw. substituiert wird. Nach diesem Prinzip lassen sich KV-Diagramme beliebiger Größe erzeugen. Die KV-Diagramm-Komponente macht von dieser Rekursion Gebrauch, um nur die fünf Basislayouts der Diagramme definieren zu müssen und größere Diagramme rekursiv daraus ableiten zu können.

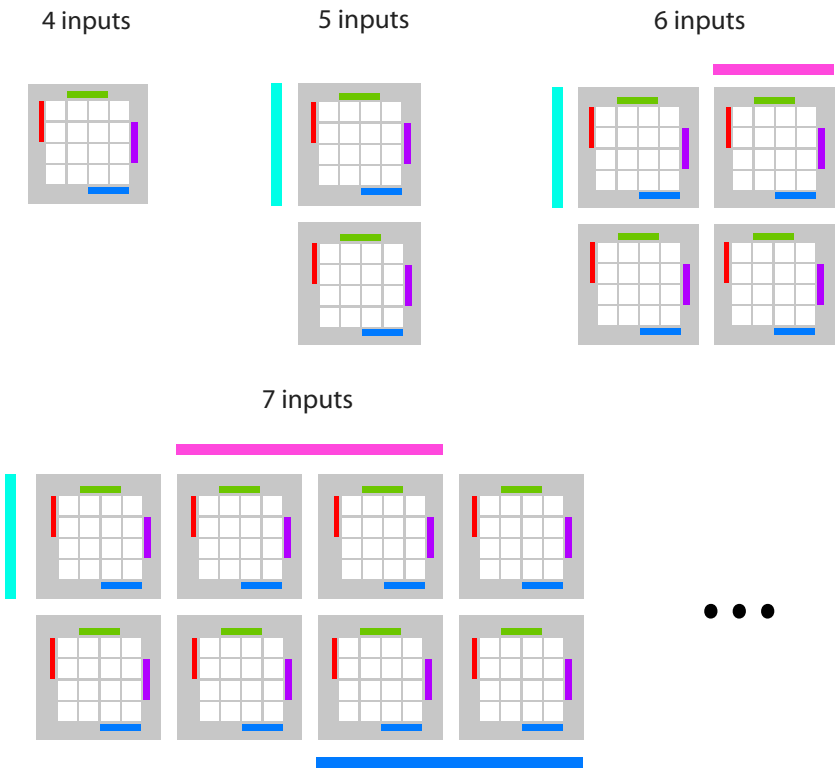


Abbildung 5.13: Das Layout von KV-Diagrammen für Funktionen mit 4 bis 7 Eingängen

Darstellung als HTML-Tabelle

Wie schon im Abschnitt *Möglichkeiten der Darstellung* im Kapitel *Gestaltung der Benutzeroberfläche* erläutert, gibt es zur Visualisierung im Browser vier verschiedene Schnittstellen, von denen Gebrauch gemacht werden kann: Die HTML/CSS-DOM-API, die 2D-Canvas-API, die 3D-Canvas-API und die SVG-DOM-API. Da es sich bei KV-Diagrammen um tabellenartige Raster handelt und HTML die Deklaration von Tabellen unterstützt, bietet sich die Verwendung von HTML zum Erzeugen der Diagramme besonders an.

Gegenüber SVG bietet die Verwendung von HTML-Tabellen den Vorteil, dass das gesamte Layouting vom Browser übernommen wird und nur die Struktur der Zellen definiert werden muss. Die Größe der Zellen und der Schrift, sowie Abstände und Rahmen, können per CSS justiert werden, ohne dass irgendwelche pixelgenauen Berechnungen per Hand angepasst und feinjustiert werden müssen. Der DOM-Baum spiegelt hierbei dieselbe rekursive Struktur wider, wie sie weiter oben beschrieben wurde: Für Funktionen mit bis zu 4 Eingängen wird ein einfaches Tabellenelement mit bis zu 4 Zeilen und bis zu 4 Spalten erzeugt. Für größere Funktionen werden Tabellenelemente ineinander verschachtelt, um die benötigte Gesamtanzahl an Zellen zu erhalten.

Das Datenmodell

Für die interne Repräsentation eines KV-Diagramms wird eine unveränderliche Datenstruktur verwendet. Da JavaScript keine Struct- oder Record-Datentypen kennt, wird die ImmutableJS-Bibliothek verwendet, um verschiedene Datentypen mittels HashMaps zu emulieren.

```
1  import I from 'immutable';
2
3  const kvDiagram = I.Record({
4    inputs: I.List(),
5    outputs: I.List.of(kvOutput()),
6    loops: I.List(),
7  }, 'kv');
8
9  const kvInput = I.Record({
10   name: "I1",
11 }, 'input');
12
13 const kvOutput = I.Record({
14   name: "O1",
15   values: I.List.of(VALUE_1),
16 }, 'output');
17
18 const kvLoop = I.Record({
19   color: '#555',
20   mode: MODE_DNF,
21   cube: kvCube(),
22   outputs: I.Set(),
23 }, 'loop');
24
25 const kvCube = I.Record({
26   include: BitSet(1),
27   exclude: BitSet(1),
28 }, 'cube');
```

Quelltext 5.1: Die Datenstrukturen, aus denen ein KV-Diagramm modelliert wird

In Quelltext 5.1 sind die Definitionen der mittels ImmutableJS emulierten Datenstrukturen zu sehen, die verwendet werden, um ein KV-Diagramm innerhalb der Komponente zu modellieren.

Ein KV-Diagramm (`kvDiagram`) besteht aus einer Liste an Eingängen (`inputs`), einer Liste an Ausgängen (`outputs`) und einer Liste an Schleifen (`loops`).

Ein Eingang (`kvInput`) hat nur einen Namen (`name`). Ein Ausgang (`kvOutput`) hat sowohl einen Namen (`name`), als auch eine Liste an Funktionswerten (`values`). Diese Liste

hat jeweils die Länge 2^N — wobei N die Anzahl der Eingänge ist — und stellt die Funktionstabelle für den jeweiligen Ausgang der Funktion dar.

Eine Schleife (`kvLoop`) hat eine Farbe (`color`), die als CSS-kompatibler String angegeben ist, und einen Modus (`mode`), der angibt, ob es sich um eine Schleife für den DNF- oder für den KNF-Modus handelt. Außerdem hat jede Schleife einen Würfel (`cube`), der die Position der Schleife im Diagramm festlegt. Da eine Schleife sich über mehrere Ausgänge erstrecken kann, ist jede Schleife einer Menge von Ausgängen (`outputs`) zugeordnet.

Ein Würfel (`kvCube`) legt die Position einer Schleife fest bzw. gibt an, welche Zellen sich in der Schleife befinden und welche nicht. Als Implementierung hierfür werden zwei BitSets verwendet: `include` und `exclude`. Jedes Bit in den beiden Sets steht für einen Eingang. Ein Bitset ist eine Menge, in welcher ein Bit entweder gesetzt ist, also enthalten ist, oder nicht gesetzt, also nicht enthalten ist. In diesem Fall steht jedes Bit für einen Eingang der Funktion.

Ein Würfel ist mittels der beiden Bitsets `include` und `exclude` wie folgt implementiert:

1. Ist das $n$te Bit in `include` gesetzt und in `exclude` nicht gesetzt, enthält der Würfel nur die Zellen, für die der $n$te Input 1 ist.

2. Ist das $n$te Bit in `exclude` gesetzt und in `include` nicht gesetzt, enthält der Würfel nur die Zellen, für die der $n$te Input 0 ist.

3. Ist das $n$te Bit weder in `exclude` noch in `include` gesetzt, enthält der Würfel sowohl Zellen, für die der $n$te Input 0 ist, also auch Zellen für die der $n$te Input 1 ist.

4. Ist das $n$te Bit sowohl in `include` also auch in `exclude` gesetzt, ist der Würfel bzw. die Schleife leer und kann gelöscht werden.

Der letzte Fall, in dem ein Bit in beiden Mengen enthalten ist, stellt einen Sonderfall dar. Er ermöglicht es, beim Bearbeiten von Funktionswerten, schon existierende Schleifen/Würfel an die neuen Funktionswerte anzupassen: Wird ein Funktionswert von 1 auf 0 geändert, werden die betreffenden `exclude` Bits von allen DNF-Schleifen auf 1 gesetzt, weil nun keine dieser Schleifen diese Zelle mehr beinhalten darf. Anschliessend können alle Schleifen automatisch entfernt werden für die $include \cap exclude \neq \emptyset$ gilt.

5.2 PLA-Format-Renderer

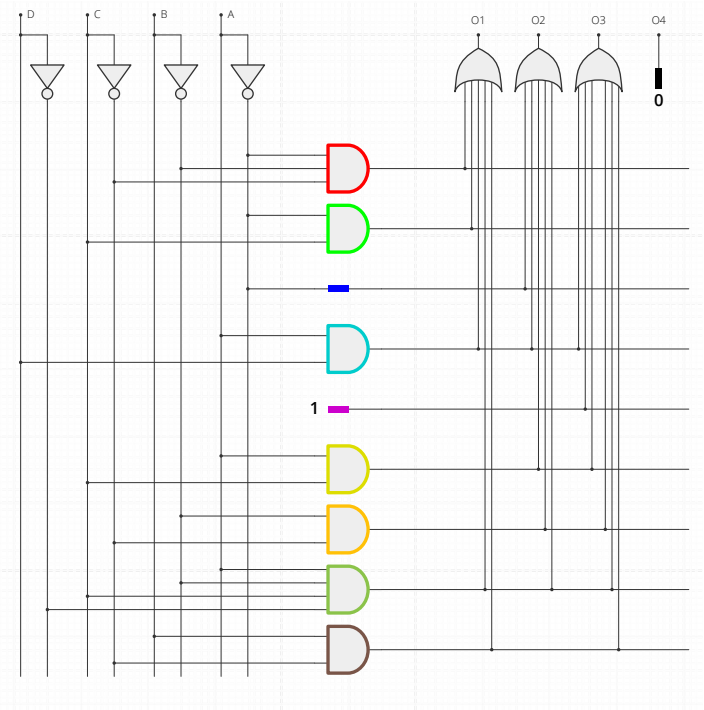


Abbildung 5.14: Die Darstellung eines Schaltznetzes für ein Programmable Logic Arrays

Die Visualisierung des PLA-Schaltnetzes für ein KV-Diagramm wurde in eine eigene Komponente ausgelagert und kann somit theoretisch in Zukunft auch getrennt von der KV-Diagramm-Komponente in anderen Zusammenhängen verwendet werden. Im Folgenden wird diese Komponente vorgestellt.

Das Datenmodell

Als Eingabe erwartet die PLA-Komponente ein JavaScript-Objekt, dessen Struktur an Teile des Berkeley Logic Interchange Formats (BLIF) angelehnt ist — das Format, welches auch von dem ursprünglichen KV-Diagramm-Applet für den Export verwendet wurde.[B 1992]

Quelltext 5.2 auf der nächsten Seite zeigt die Struktur der erwarteten Eingabe als JSON-Objekt. Dabei gibt `mode` an, ob es sich um einen disjunktiven oder eine konjunktiven Term handelt, der visualisiert werden soll. `inputs` ist eine Liste der Namen der Eingänge als `Strings`. `outputs` ist die Liste mit Namen der Ausgänge. `loops` ist eine Liste von

```
{
  "mode": "dnf",
  "inputs": [
    "Input A",
    "Input B",
  ],
  "outputs": [
    "Out 1",
  ],
  "loops": [
    {
      "in": [
        false, true,
      ],
      "out": [
        true,
      ],
      "color": "blue",
      "highlight": false,
    },
  ],
}
```

Quelltext 5.2: Das Format, welches die PLA-Komponente als Eingabe erwartet

Gattern. Je nach `mode` handelt es sich hierbei um UND-Gatter (DNF) bzw. ODER-Gatter (KNF).

Die Namensgebung ist hierbei an die Schleifen der KV-Diagramme angelehnt. Es lässt sich vermuten, dass langfristig eine Umbennung sinnvoll ist, um die Schnittstellen zu entkoppeln.

Jedes dieser Gatter hat eine Liste mit Eingangswerten (`in`), deren Werte entweder `true`, `false` or `null` sein kann. Die Liste muss so viele Elemente haben, wie auch die `inputs`-Liste. Der $n$te Eintrag der `in`-Liste gibt an, ob der $n$te Eingang des Schaltnetzes mit einem Eingang des Gatters verbunden werden soll:

DNF-Modus

- `true` gibt an, das Gatter mit dem Eingang zu verbinden
- `false` gibt an, das Gatter mit dem negierten Eingang zu verbinden
- `null` gibt an, das Gatter nicht mit dem Eingang zu verbinden

KNF-Modus

- `true` gibt an, das Gatter mit dem negierten Eingang zu verbinden
- `false` gibt an, das Gatter mit dem Eingang zu verbinden
- `null` gibt an, das Gatter nicht mit dem Eingang zu verbinden

Zudem kann jedes Gatter eine Farbe (`color`) haben, sowie mittels `highlight`-Flag gekennzeichnet werden, um visuell hervorgehoben zu werden.

Das Rendering

Für die Darstellung des Schaltnetzes wird das SVG-Format verwendet. Die Symbole der Gatter setzen sich teilweise aus geschwungenen Linien zusammen, die sich mittels HTML und CSS nicht erzeugen lassen. Zudem müssen die Positionen der Gatter explizit berechnet werden, sodass hier keine Verwendung des vom Browser geleisteten Layoutings gemacht werden kann. Wie schon zuvor erklärt, bietet das SVG-Format als Vektorgrafik gegenüber der Canvas-API den Vorteil der scharfen Darstellung auf allen Bildschirmgrößen. Es müssen Gatter in unterschiedlichen Größen, also mit verschieden vielen Eingängen, gezeichnet werden.

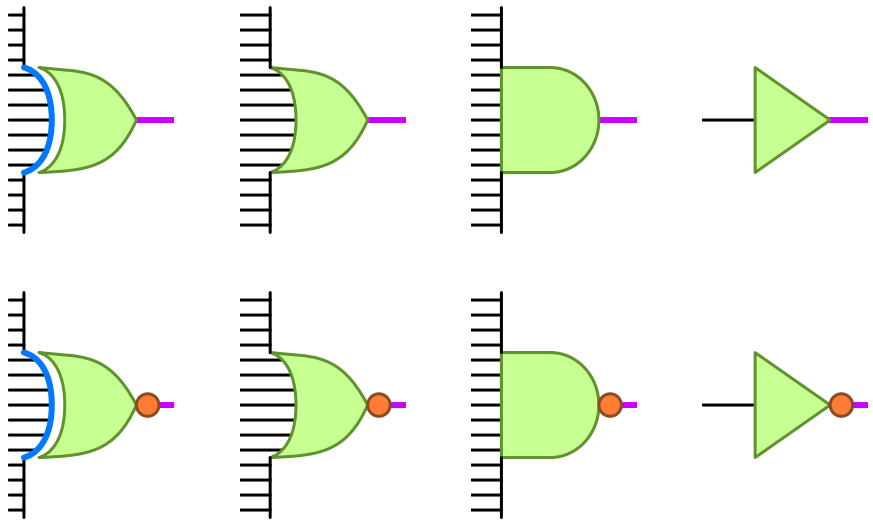


Abbildung 5.15: Die Komponenten der ANSI-Symbole der verschiedenen Logikgatter

Zwar werden für die Visualisierung des Schaltnetzes eines KV-Diagramms nur Inverter, UND- und ODER-Gatter benötigt, doch um auch in Zukunft andere Schaltnetze im gleichen Stil darstellen zu können, wurden im Rahmen dieser Arbeit die Generierung aller in

Abbildung 5.15 auf der vorherigen Seite dargestellten Gatter als JavaScript-Funktionen implementiert. Diese Funktionen lassen sich auch außerhalb der Komponente verwenden, um SVG-Elemente für die diversen Gatter zu erzeugen.

Die Visualisierung einer Schaltung in der PLA-Komponente erfolgt in zwei Schritten: Zuerst wird aus der gegebenen Eingabe eine Menge an Gattern, Verbindungsdrähten und Beschriftungen erzeugt, sowie die Breite und Höhe der entstehenden Schaltung berechnet. Im nächsten Schritt werden daraus die nötigen SVG-Elemente erzeugt. Die Berechnung des Layouts ist also von der tatsächlichen Darstellungstechnik (SVG) entkoppelt.

5.3 SVG-Viewer

Damit auch große Schaltnetze auf nicht so großen Bildschirmen noch angenehm betrachtet und verstanden werden können, soll der Benutzer die Möglichkeit haben, heranzuzoomen und einen gewählten Ausschnitt genauer zu betrachten. Die Möglichkeit, Grafiken in unterschiedlichen Zoomstufen zu betrachten, ist nicht nur für Schaltnetze interessant, sondern kann auch für andere Komponenten benötigt werden. Darum wurde diese Funktionalität in eine eigene Komponente extrahiert: Der SVG-Viewer. Auf die wichtigsten Aspekte des SVG-Viewers wird im folgenden eingagangen.

Modularität

Diese Komponente soll es ermöglichen, beliebige SVG-Inhalte vergrößert oder verkleinert darzustellen und dem Benutzer erlauben, die Zoomstufe und den zu betrachtenden Ausschnitt frei zu wählen. Allerdings muss es auch möglich sein, die Zoomstufe und die Position des Blickfeldes in festgelegten Grenzen zu beschränken, um den Benutzer davor zu bewahren, die Orientierung in der Szene zu verlieren. Mit der Szene ist in diesem Zusammenhang die Menge an Elemente gemeint, die der Benutzer betrachten kann.

Darum benötigt diese Komponente als Parameter den darzustellenden Inhalt zusammen mit einem Begrenzungsrahmen, in dem der Benutzer die virtuelle Kamera bewegen darf. Die virtuelle Kamera stellt hier eine Metapher für die Modellierung des dargestellten Ausschnitts der Szene dar.

Gestenerkennung

Für das Betrachten von grafischen Inhalten ist es ein übliches Bedienkonzept mittels Drehung des Mausrades heran und heraus zoomen zu können. Dabei hat es sich etabliert, dass die Position des Mauszeigers den Pivotpunkt der Skalierung bildet, sodass der Benutzer über die Zeigerposition bestimmen kann, welchen Bereich der dargestellten Szene er genauer betrachten möchte, während er hinein zoomt. Das Drehen des Mausrades stellt dabei das Heranziehen bzw. das Wegschieben der dargestellten Szene dar.

Ebenso ist es üblich, dass das Bewegen der Maus bei gedrückter Maustaste das Verschieben der virtuellen Kamera über der zweidimensionalen Szene bewirkt. Dabei verhält sich die dargestellte Szene, wie ein Blatt Papier, welches gegriffen und auf einer Tischplatte hin und her geschoben wird.

Bei der Bedienung auf einem Touchscreen hingegen hat es sich durchgesetzt, dass Inhalte mittels Pinch-Geste gezoomt werden können. Dafür wird der Bildschirm mit zwei Fingern berührt. Diese werden dann entweder voneinander weg bewegt, um eine proportionale Streckung der Szene zu symbolisieren, wodurch die Szene vergrößert, also herangezoomt, wird — oder sie werden aufeinander zubewegt, um die Szene zusammenzustauchen, also zu verkleinern, also heraus zu zoomen. Dabei bildet der Mittelpunkt zwischen den beiden Berührpunkten der Finger den Pivotpunkt, um den die Skalierung stattfindet. Wichtig ist, dass der Skalierungsfaktor aus der Änderung des Fingerabstandes berechnet wird, damit die Größe der Szene sich im gleichen Maße ändert, wie der Abstand der Finger, und die Punkte der Szene, die zu Beginn der Skalierung direkt unter den Berührpunkten der Finger liegen, sich auch noch am Ende der Skalierung unter den Fingerspitzes des Benutzers befinden. Dadurch entsteht der Eindruck für den Benutzer, die Szene tatsächlich anfassen zu können. Wird die Szene während der Fingerbewegung zu stark oder zu wenig skaliert, entsteht ein unwirkliches Gefühl in der Benutzung, da die Szene den Fingern zu entgleiten scheint.

Die Positionierung der virtuellen Kamera, durch welche die Szene betrachtet wird, funktioniert auf einem Touchscreen analog zu der Bedienung mit der Maus. Während ein Finger den Bildschirm berührt, kann die Szene durch die Bewegung des Fingers verschoben werden.

```
import {Observable as O} from 'rx';

const touchChange$ = O.merge([
  O.fromEvent(element, 'touchstart'),
  O.fromEvent(document, 'touchend'),
]);

const touchMove$ = O.fromEvent(document, 'touchmove');

const panStart$ = touchChange$
  .filter((evt) => evt.touches.length >= 1);
const panEnd$ = touchChange$
  .filter((evt) => evt.touches.length < 1);

const panDistance$ = panStart$.flatMap((startEvent) =>
  touchMove$.map((moveEvent) => ({
    x: center(moveEvent).x - center(startEvent).x,
    y: center(moveEvent).y - center(startEvent).y,
  })).takeUntil(panEnd$)
);
```

Quelltext 5.3: Erkennung einer Pan-Geste mittels des Observable Datentyps

Die Erkennung der Touch-Gesten lässt sich besonders elegant mit Hilfe des `Observable`-Datentyps implementieren, wie in Quelltext 5.3 auf der vorherigen Seite zu erkennen ist. `panStart$` enthält nur die `touchstart` und `touchend` Ereignisse, in die ein oder mehr Finger involviert sind. Hingegen enthält `panEnd$` die Ereignisse, deren Fingeranzahl geringer ist als 1. Mittels `flatMap` Operator werden alle `touchmove` Ereignisse verarbeitet, die zwischen dem Beginn und dem Ende eines "Pans" liegen. Die `x` und `y` Komponenten der Distanz, um welche die Camera verschoben werden muss, wird aus der Differenz der Mittelpunkte der Berührpunkte berechnet. So ist es auch möglich, die virtuelle Kamera zu verschieben, während mehrere Finger den Touchscreen berühren.

```
1  import {Observable as O} from 'rx';
2
3  const element = document.querySelector('svg');
4
5  const touchChange$ = O.merge([
6    O.fromEvent(element, 'touchstart'),
7    O.fromEvent(document, 'touchend'),
8  ]);
9
10 const touchMove$ = O.fromEvent(document, 'touchmove');
11
12 const pinchStart$ = touchChange$
13   .filter((evt) => evt.touches.length >= 2);
14 const pinchEnd$ = touchChange$
15   .filter((evt) => evt.touches.length < 2);
16
17 const zoom$ = pinchStart$.flatMap((startEvent) =>
18   touchMove$.map((moveEvent) => ({
19     factor: radius(moveEvent) / radius(startEvent),
20     pivot: center(moveEvent),
21   })).takeUntil(zoomEnd$)
22 );
```

Quelltext 5.4: Erkennung einer Pinch-Geste mittels des Observable Datentyps

Quelltext 5.4 zeigt analog dazu die Implementierung einer Pinch-Gestenerkennung. Der Unterschied besteht nur in der geforderten Anzahl an Berührpunkten und darin, dass ein Skalierungsfaktor (`factor`) aus dem Verhältnis der Radien der Berührpunkte berechnet wird.

Performance

Diese Komponente macht — wie alle anderen Komponenten auch — Gebrauch von der Virtual-DOM-Bibliothek, um das SVG-Element, welches die darzustellende Szene

beinhaltet, zu erzeugen. Während die virtuelle Kamera gezoomt oder durch die Szene bewegt wird, ändert sich der Inhalt der Szene für gewöhnlich nicht. Die einzige DOM-Änderung, die für die Änderung der Kameraposition vorgenommen werden muss, ist die `viewBox`-Eigenschaft des SVG-Wurzelelements. Standardmäßig durchläuft die Virtual-DOM-Bibliothek für jede Änderung den gesamten alten und den gesamten neuen DOM-Baum, um jeden Knoten auf Änderungen zu überprüfen. Dieses Verhalten führt während der Positionierung der Kamera zu unverhältnismäßig viel Rechenaufwand, was zu Rucklern in der Benutzeroberfläche führt. Um den Rechenaufwand zu vermindern, umhüllt diese Komponente die Elemente, die sie darstellen soll mit einem sogenannten `Thunk`-Objekt. Die Virtual-DOM-Bibliothek erlaubt es mit einem solchen Markierungsobjekt, die Vergleichsberechung für Kindknoten eines Teilbaumes zu unterdrücken.

5.4 Logic-Expression-Parser

Die Logic-Expression-Parser-Komponente soll die Eingabe von logischen Ausdrücken in unterschiedlichen Dialekten erlauben. Zu einem Ausdruck soll eine Funktionstabelle generiert und ein Operatorbaum erzeugt werden können. Eine Funktionstabelle ermöglicht es, alle Funktionswerte eines Ausdrucks auf einen Blick zu erfassen. Ein Operatorbaum stellt die semantische Struktur eines Ausdrucks grafisch dar.

Es soll möglich sein, Ausdrücke in den Dialekten der Sprachen C, LaTeX, Python und in mathematischer Notation anzugeben, um eine möglichst vielseitige Verwendung der Komponente zu erlauben. Zudem soll es möglich sein, einen Ausdruck von dem einen in den andern Dialekt umzuwandeln. Das ist zum Beispiel hilfreich, um einen Ausdruck aus einem geschriebenen C-Programm in ein LaTeX-Dokument zu übertragen.

Benutzeroberfläche

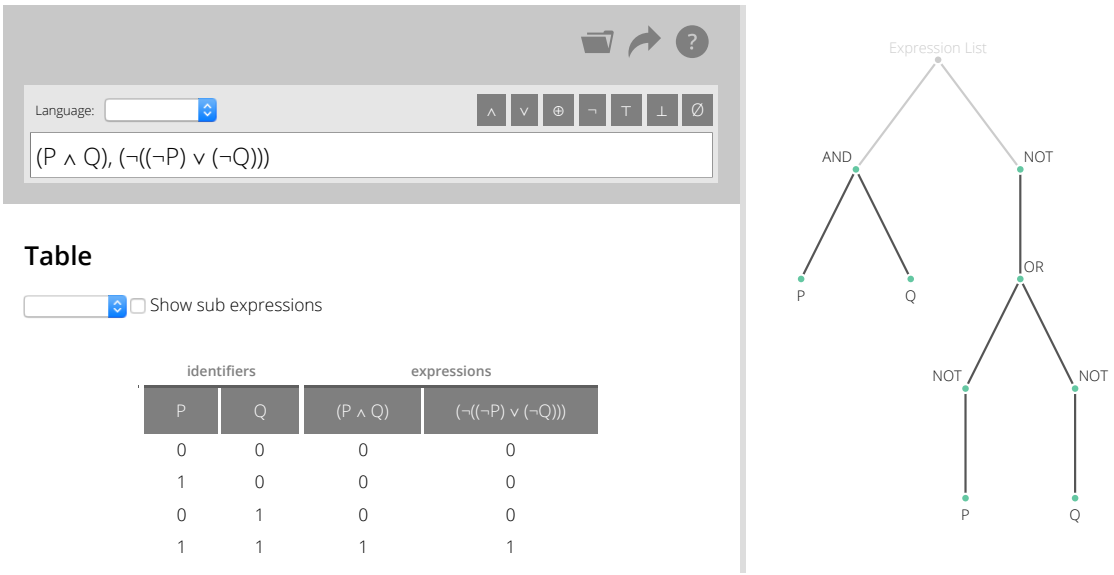


Abbildung 5.16: Die Benutzeroberfläche der Logik-Komponente

In Abbildung 5.16 ist die Benutzerobrfläche der Komponente zu sehen. Oben auf der linken Seite befindet sich ein Textfeld, in welches ein Boole'scher Ausdruck eingegeben werden kann. Oberhalb des Textfeldes befindet sich zum einen die Auswahlbox, über welche der gewünschte Dialekt gewählt werden kann. Ist kein Dialekt gewählt, versucht die Komponente den Dialekt der Eingabe automatisch zu ermitteln. Zum anderen befindet sich über dem Textfeld eine Liste mit Button, die benutzt werden können, um Zeichen für Operatoren und Literale, welche auf einer üblichen Tastatur nicht verfügbar sind, in das Textfeld einzufügen. Diese Button ändern sich je nach eingestelltem — oder von der Komponente ermitteltem — Dialekt.

Unterhalb des Textfeldes ist eine Tabelle zu sehen, welche die Funktionswerte der eingegebenen Ausdrücke für alle möglichen Eingangsbelegungen darstellt. Oberhalb der Tabelle befinden sich eine Auswahlbox und einen Checkbox. Über die Auswahlbox lässt sich das Ausgabeformat der Ausdrücke in der Kopfzeile der Tabelle steuern. Als Optionen stehen hierfür dieselben Dialekte zur Verfügung, die auch für die Eingabe der Ausdrücke angeboten werden: C, Python, LaTeX und mathematische Notation. Die Checkbox erlaubt das Ein- und Ausblenden zusätzlicher Tabellenspalten, in denen nicht nur die Funktionswerte der Ausdrücke, sondern auch die aller Teilausdrücke dargestellt werden.

Auf der rechten Seite der Komponente ist der Operatorbaum des eingegebenen Ausdrucks zu sehen. Wie auch die Darstellung des Schaltnetzes der KV-Diagramm-Komponente, wird der Operatorbaum als Vektorgrafik generiert und lässt sich per Mausrad oder Pinchgeste auf einem Touchscreen zoomen.

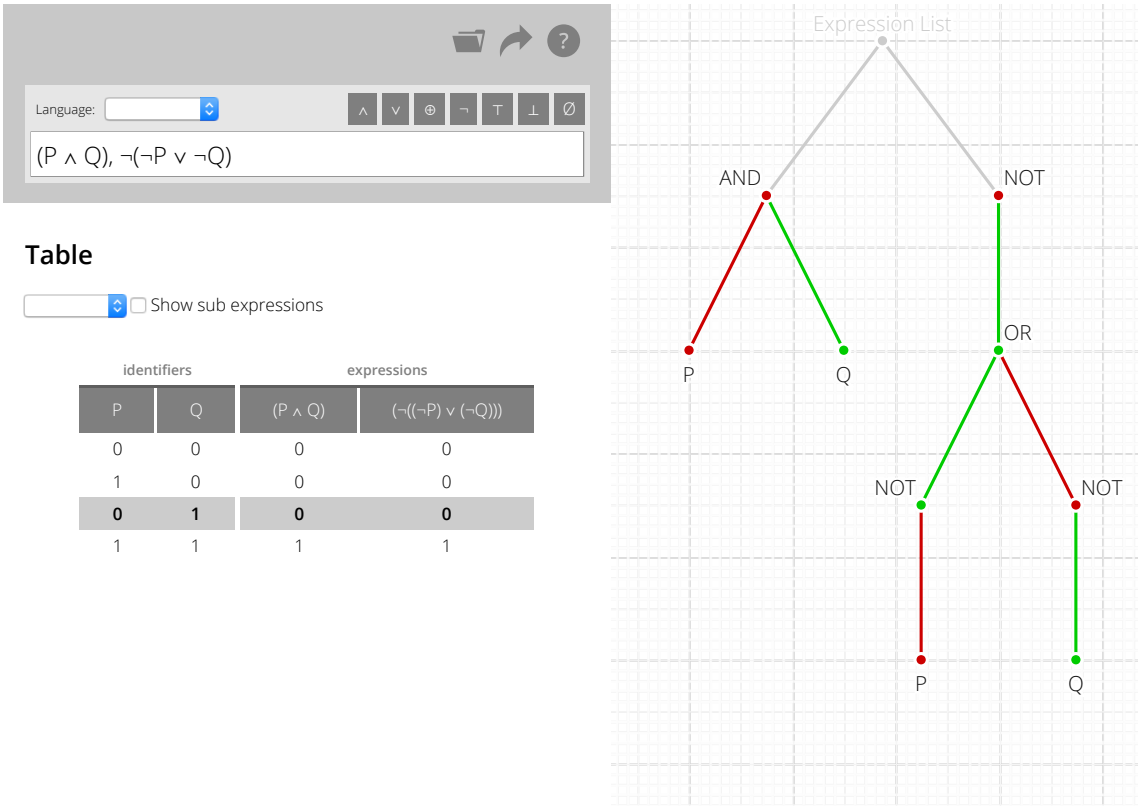


Abbildung 5.17: Einfärbung des Operatorbaumes der Logik-Komponente

Die Zeilen in der Funktionstabelle lassen sich per Mausklick auswählen. Ist eine Zeile der Tabelle selektiert, wird der Operatorbaum auf der rechten Seite für die gewählte Eingangsbelegung eingefärbt. Knoten der Teilausdrücke, die für die gewählte Belegung zu dem Wahrheitswert `Falsch` ausgewertet werden, werden in rot dargestellt. Entgegen werden Knoten, die zu `Wahr` ausgewertet werden, in grün dargestellt. Die Einfärbung

des Operatorbaumes erlaubt es dem Benutzer, die Auswertung des Ausdruckes Schritt für Schritt nachzuvollziehen.

Parser Generatoren

Damit mit einem Ausdruck, der als Zeichenkette eingegeben wurde, gearbeitet werden kann, muss dieser zunächst syntaktisch analysiert (geparsed) werden. Dabei wird die lineare Zeichenkette, die als Eingabe vorliegt, in einen Abstrakten Syntax Baum (AST) umgewandelt, welcher die semantische Struktur der Eingabe repräsentiert. Da es sich bei den zu analysierenden Ausdrücken in allen Dialekten um mathematische Ausdrücke in der Infixnotation handelt, bietet der Shunting-Yard-Algorithmus eine solide Grundlage, um einen Parser per Hand zu schreiben. Ein handgeschriebener Parser stellte aber langfristig vermutlich einen erhöhten Wartungsaufwand dar — besonders, falls die Menge an unterstützten Dialekten zukünftig erweitert werden soll. Schon während dieser Arbeit haben sich die Anforderungen and die Grammatiken der Dialekte des Öfteren geändert. Darum wurde für die Implementierung dieser Komponente auf eine Parsergenerator-Bibliothek zurückgegriffen, die es erlaubt, eine Grammatik deklarativ zu definieren.[DIJKSTRA 1961]

Für JavaScript stehen unterschiedliche Parsergenerator-Bibliotheken zur Verfügung. Im Rahmen dieser Arbeit wurde zuerst Jison verwendet und später auf PEG.js gewechselt.[CARTER 2009][MAJDA 2010][PARR und FISHER 2011][PARR 2014][MEYER 2011]

Jison

Jison ist ein Parsergenerator, der von Yacc und Bison inspiriert ist und das Erzeugen von Parsern für kontextfreie Grammatiken ermöglicht. Konzeptionell trennt Jison die Definition einer Grammatik in die Definition der einfachen lexikalischen Token und die Definition der komplexeren Regeln. Zudem erlaubt Jison die explizite Festlegung der Assoziativität und Präzedenz von Operatoren. [CARTER 2009]

Für Webpack steht ein Jison-Plugin zur Verfügung, welches eine Grammatik schon in einem Vorverarbeitungsschritt in eine JavaScript-Funktion konvertiert, sodass die Verarbeitung der Grammatik selbst nicht zu Laufzeit geschehen muss.[BURGET 2014]

Leider liefert Jison in einigen Fällen keine besonders hilfreichen Fehlermeldungen für den Benutzer und es ist nicht gut dokumentiert, wie sich die Fehlermeldungen für Syntaxregeln indivualisieren lassen. Da die Logic-Expression-Parser-Komponente hauptsächlich für die Lehre gedacht ist, sind aussagekräftige Fehlermeldungen essenziell. Wenn

ein Benutzer einen Boole'schen Ausdruck fehlerhaft eingibt, soll er eine möglichst genaue Beschreibung bekommen, was zu tun ist, um seine Eingabe zu korrigieren. Da Jison dieser Anforderung nicht gerecht geworden ist, war es nötig, auf einen anderen Parsergenerator umzusteigen.

PEG.js

PEG.js ist ein Parsergenerator für Parsing Expression Grammatiken (PEG). Die Bibliothek wirbt damit, exzellente Fehlermeldungen erzeugen zu können. In PEG wird keine Trennung zwischen der Definition lexikalischen Token und den Grammatikregeln vorgenommen. Auch Operatorprezedenzen lassen sich nicht ordinal deklarieren, sondern ergeben sich implizit aus den definierten Regeln.[FORD 2004][MAJDA 2010]

Auch für die Verwendung von PEG.js gibt es ein Webpack-Plugin, welches den Parser für eine Grammatik schon automatisch vorab generiert, um Laufzeitkosten zu sparen.[SUBBOTIN 2014]

Fehlermeldungen für PEG.js-Grammatiken lassen sich sehr einfach angeben. Jede Regel der Grammatik lässt sich optional mit einem String als Annotation versehen, welcher für die Fehlermeldung verwendet wird, falls die Verarbeitung einer Eingabe mit der jeweiligen Regel nicht erfolgreich war. Des Weiteren lässt sich jeder Regel der Grammatik eine JavaScript-Prozedur anhängen, in welcher der Teil der Eingabe, der von der Regel akzeptiert wurde, nachverarbeitet werden kann.

Das Einzige Problem, was bei der Verwendung der PEG.js-Bibliothek auftrat, war, dass es nicht möglich ist, Teile einer definierten Grammatik in einer anderen Grammatik wiederzuverwenden. Im Rahmen dieser Arbeit wurden fünf Grammatiken für fünf verschiedene Dialekte an Boole'schen Ausdrücken definiert, die sich zu einem großen Teil ähneln. Es wäre sinnvoll gewesen, die Gemeinsamkeiten in eine Grammatik auslagern zu können, auf der die anderen Grammatik-Definitionen aufbauen können. Zukünftige Versionen von PEG.js werden voraussichtlich den Import von Grammatiken unterstützen, sodass sich die Logic-Expression-Parser Komponente diesbezüglich dann weiter vereinfachen lässt.[MINGUN 2014]

Vereinfachung der Dialekte

Für die Bildung Boole'scher Ausdrücke sollen die Junktoren Konjunktion, Disjunktion, Kontravalenz und Negation verwendet werden können. Zudem sollen Konstanten zur Repräsentation der Wahrheitswerte `Wahr` und `Falsch` zur Verfügung stehen. In den

	Wahr	Falsch	Unbestimmt	Und	Oder	exkl. Oder	Nicht
Bitwise C	1	0	void	&	|	^	~
Boolean C	true	false	void	&&	||	!=	!
Python	True	False	None	and	or	xor	not
Math	$\top$	$\bot$	$\emptyset$	$\wedge$	$\vee$	$\oplus$	$\neg$
LaTeX	\top	\bot	\nothing	\wedge	\vee	\oplus	\neg

Tabelle 5.1: Syntax der Dialekte für Boole'sche Ausdrücke

Namen von Veränderlichen sollen Leerzeichen verwendet werden können, damit Bezeichnungen wie „Ampel ist grün“ unverändert in einen Ausdruck übernommen werden können. Des Weiteren sollen nebeneinanderstehende Teilausdrücke, die nicht durch einen Operator getrennt sind, automatisch mittels Konjunktion verknüpft werden, um der mathematischen Konvention zu folgen.

Diese Anforderungen sind ersteinmal nicht mit allen fünf zu implementierenden Dialekten kompatibel. So unterstützt C eigentlich keine Leerzeichen in Variablennamen und Python hat neben den Operatoren `and` und `or` keinen echten logischen `xor`-Operator, sondern nur den Vergleichsoperator `!=`. Genauso ist es in C und Python syntaktisch nicht erlaubt, zwei Veränderliche in einem Ausdruck direkt nebeneinander zu schreiben. LaTeX als Schriftsatzsystem erlaubt zwar die visuelle Darstellung logischer Ausdrücke, definiert aber keine Operatorpräzedenzen.

Die Unterschiede zwischen den verfügbaren Dialekten sollen so gering wie möglich gehalten werden, um Verwirrung beim Benutzer zu vermeiden. Darum wurde für die Implementierung eine Vereinheitlichung der fünf Dialekte vorgenommen, die dazu führt, dass die C- und Python-Parser nicht exakt der Semantik der jeweiligen Sprache entsprechen. In Folge der Vereinheitlichung unterscheiden sich die fünf Dialekte nur noch in ihren Literalen und den Lexemen der Operatoren. Folgende Vereinheitlichungen wurden zwischen den Dialekten vorgenommen:

1. Die Namen von Variablen können in Anführungszeichen geschrieben werden und dürfen dann Leerzeichen und andere Sonderzeichen enthalten

2. Direkt nebeneinanderstehende Ausdrücke werden implizit konjugiert

3. Die Präzedenz der Operatoren ist: (stark bindend) `NOT`, `AND`, `OR`, `XOR` (schwach bindend)

4. Es stehen Literale für die Konstanten `Wahr`, `Falsch` und `Don't-Care` zur Verfügung

X	Y	Z	Funktionswert
0	0	0	0
1	0	0	1
0	1	0	0
1	1	0	0
0	0	1	*
1	0	1	*
0	1	1	1
1	1	1	1

Tabelle 5.2: Eine Funktionstabelle mit drei Eingangswerten

Zudem wurde der C-Dialekt in die beiden Dialekte Bitwise-C und Boolean-C aufgetrennt, um sowohl die Verwendung von logischen als auch die Verwendung bitweiser Operatoren zu erlauben. Tabelle 5.1 auf der vorherigen Seite zeigt die Operatoren und Literale der verschiedenen Dialekte.

Dreiwertige Logik

Wie schon in Tabelle 5.1 auf der vorherigen Seite zu sehen war, gibt es neben den Wahrheitswerten `Wahr` und `Falsch` auch die Möglichkeit einen Wert `Unbestimmt` bzw. `Don't Care` anzugeben. Dieser Erweiterung der Boole'schen Logik auf die Dreiwertige Logik ist nötig, um eine vollständigere Kompatibilität zwischen dieser Komponente und der KV-Diagramm-Komponente zu bieten. Wie im Abschnitt zur KV-Komponente erklärt, können Felder im KV-Diagramm mit einem `Don't Care`-Wert belegt sein und bieten dadurch oft bessere Möglichkeiten zur Minimierung. Da das Ergebnis eines KV-Diagramms als Boole'scher Ausdruck exportiert werden soll und andersherum auch ein Boole'scher Ausdruck zur Initialisierung eines KV-Diagramms verwendet werden können soll, hat es sich während der Entwicklung der Komponente angeboten, auch die Syntax für Boole'sche Ausdrücke um ein Literal für einen `Unbestimmt` Wert zu erweitern.

Literal für Funktionstabelle

Eine zusätzliche Anforderung, die sich während der Entwicklung dieser Komponente ergeben hat, ist die Möglichkeit, eine komplette Funktionstabelle als Ausdruck oder Teilausdruck anzugeben. Dies ist zum Beispiel sinnvoll, um eine vorgegebene Funktionstabelle per `XOR`-Operator mit den Funktionswerten eines Ausdrucks vergleichen zu kön-

nen. Um eine Funktionstabelle möglichst bündig in einen Logischen Ausdruck kodieren zu können, wurden alle fünf Dialekte um ein Literal erweitert, welches es erlaubt, eine Funktionstabelle als Liste von Wahrheitswerten anzugeben. Beispielsweise kann eine Funktion, welche in Tabelle 5.2 auf der vorherigen Seite tabellarisch dargestellt ist, mit den drei Parametern X, Y und Z wie folgt kodiert werden:

```
<X,Y,Z:0100**11>
```

Vor dem Doppelpunkt sind die Namen der n Funktionsparameter gelistet. Nach dem Doppelpunkt folgt eine Liste mit 2^n Funktionswerten. Innerhalb dieses Literals werden in allen fünf Dialekten „1“, „0“ und „$*$“ für die Wahrheitswerte $Wahr$, $Falsch$ und $Unbestimmt$ verwendet, um die Länge des Literals zu verringern. In Quelltext 5.5 ist die Definition der Grammatik für die Funktionstabellenliterale dargestellt.

⟨Tabelle⟩ ⊨ < ⟨Namen⟩ : ⟨Werte⟩ >
⟨Namen⟩ ⊨ ⟨Name⟩ | ⟨Name⟩ , ⟨Namen⟩
⟨Werte⟩ ⊨ ⟨Wert⟩ | ⟨Wert⟩ ⟨Werte⟩
⟨Name⟩ ⊨ *Name des Funktionsparameters*
⟨Wert⟩ ⊨ 1 | 0 | *

Quelltext 5.5: Grammatik für Funktionstabellen

Mehrere Ausdrücke

⟨Start⟩ ⊨ ⟨Ausdruck⟩
⟨Ausdrucksliste⟩ ⊨ ⟨Ausdruck⟩ | ⟨Ausdruck⟩ , ⟨Ausdrucksliste⟩
⟨Ausdruck⟩ ⊨ *Ein Boole'scher Ausdruck*

Quelltext 5.6: Grammatik für Listen Boole'scher Ausdrücke

Vom Benutzer soll nicht nur ein einziger Ausdruck, sondern auch eine Liste aus mehreren Ausdrücken eingegeben werden können. Die Eingabe mehrere Ausdrücke ist nötig, um komplexere Schaltnetze, die mehrere Ausgabewerte haben, modellieren zu können. Zum Beispiel die Schaltung einer Verkehrsampel hat nicht nur einen einzelnen Ausgabewert, sondern je einen Ausgabewert für das rote, gelbe und grüne Licht, der für jeden Zustand der Ampel angibt, ob das jeweilige Licht an oder ausgeschaltet ist. Auch die KV-Diagramm-Komponente ermöglicht ja das Modellieren von mehreren Ausgängen einer Schaltung bzw. einer Funktion. Damit mehrere Ausdrücke vom Benutzer eingegeben

werden können, wurden die Grammatiken aller fünf Dialekte erweitert, wie in Quelltext 5.6 auf der vorherigen Seite zu sehen ist.

$$
\begin{aligned}
\langle \text{Ausdruck} \rangle &\models \langle \text{BenannterAusdruck} \rangle \mid \langle \text{AnonymerAusdruck} \rangle \\
\langle \text{BenannterAusdruck} \rangle &\models \langle \text{AusdruckName} \rangle = \langle \text{AnonymerAusdruck} \rangle \\
\langle \text{AusdruckName} \rangle &\models \textit{Gültiger Name} \\
\langle \text{AnonymerAusdruck} \rangle &\models \textit{Ein Boole'scher Ausdruck}
\end{aligned}
$$

Quelltext 5.7: Grammatik für referenzierbare Ausdrücke

Nachdem es möglich war, mehrere Ausdrücke einzugeben, hat es sich während der Verwendung der Komponente als sinnvoll herausgestellt, einen Ausdruck aus einen weiteren Ausdruck referenzieren zu können. Um das Referenzieren von Ausdrücken zu erlauben, ist es nötig, die Ausdrücke eindeutig benennen zu können. Entsprechend wurde die Grammatik wie in Quelltext 5.7 zu sehen ist, erweitert.

$$
\begin{aligned}
C &= A \oplus B, \\
A &= P \wedge Q, \\
B &= P \oplus Q
\end{aligned}
$$

Quelltext 5.8: Liste Boole'scher Ausdrücke

So ist es letztendlich möglich, dass eine Liste von Ausdrücken, wie in Quelltext 5.8 zu sehen, eingegeben werden kann und erfolgreich von der Komponente verarbeitet wird. In diesem Beispiel werden drei Boole'sche Funktionen angegeben, die mit A, B und C bezeichnet werden. Der Ausdruck C beinhaltet dabei Referenzen auf die Ausdrücke A und B. A und B stellen in diesem Fall keine freien Veränderlichen dar, sondern lediglich jeweils eine Referenz auf einen Ausdruck durch den sie substituiert werden können.

Ausdrücke Auswerten

Damit solche Ausdrücke, wie die in Quelltext 5.8 dargestellten, zu einer Funktionstabelle ausgewertet werden können, müssen die Ausdrücke zunächst topologisch sortiert werden, damit sie in der richtigen Reihenfoge verarbeitet werden können. So kann in dem genannten Beispiel der Ausdruck C nicht ausgewertet werden, bevor die Ausrücke A und B ausgewertet wurden. Für die Sortierung wird Kahn's Algorithmus verwendet.[KAHN 1962]

Des Weiteren muss darauf geachtet werden, dass bei der Referenzierung mehrere Ausdrücke untereinander keine zyklischen Abhängigkeiten entstehen. Eine Menge an Aus-

$$C = A \oplus B,$$
$$A = P \wedge Q,$$
$$B = P \oplus C$$

Quelltext 5.9: Zyklische Abhängigkeit zwischen Ausdrücken

drücken, die sich zyklisch referenzieren, kann nicht ausgewertet werden und ist somit als ungültig zu betrachten. Quelltext 5.9 zeigt drei Ausdrücke, die einen Abhängigkeitszyklus bilden. Ausdruck `C` referenziert Ausdruck `B` und Ausdruck `B` referenziert wiederum Ausdruck `C`. Solch ein Zyklus soll erkannt und dem Benutzer während der Eingabe gemeldet werden.

Nachdem eine Eingabe mittels PEG.js erfolgreich in einen Abstrakten Syntax Baum (AST) konvertiert wurde, finden also eine Reihe von Nachbearbeitungsschritten statt:

1. Extrahieren der Referenznamen aller Ausdrücke
2. Erzeugung eines gerichteten Graphen, der die Abhängigkeiten zwischen den Ausdrücken darstellt
3. Topologische Sortierung der Ausdrücke anhand der Kanten des Graphen
4. Eventuelle Erkennung eines Zyklus im Graphen
5. Extrahieren aller freien Veränderlichen aller Ausrücke
6. Erzeugung der Menge aller kombinatorisch möglichen Eingangsbelegungen
7. Auswertung der sortierten Ausdrücke für jede mögliche Eingangsbelegung

5.5 Tree-Renderer

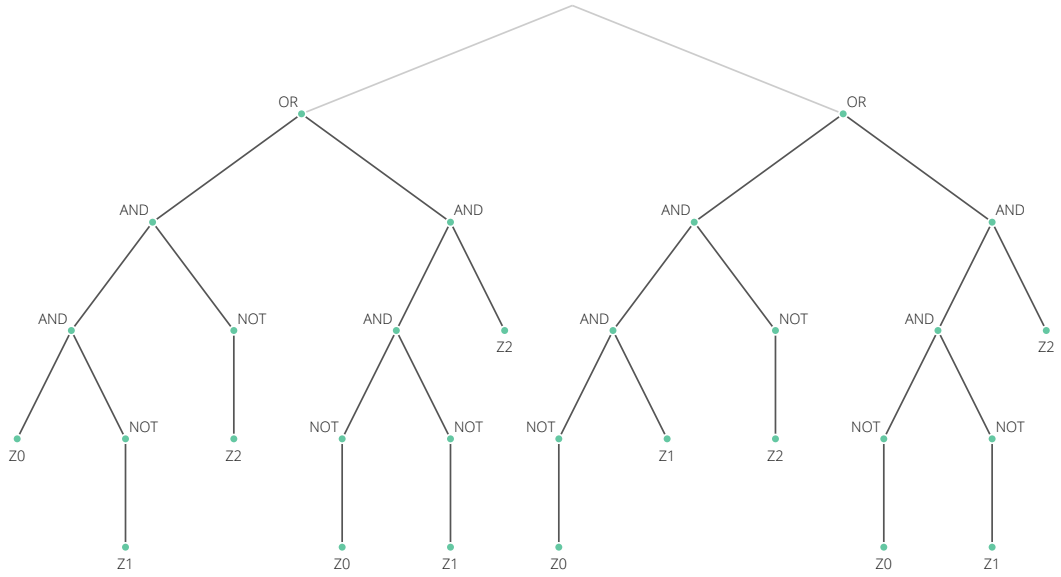


Abbildung 5.18: Ein mittels SVG gezeichneter Operatorbaum

Die Darstellung des Abstrakten Syntax Baumes (AST) der Logic-Expression-Parser-Komponente wurde in eine eigene Komponente ausgelagert. Die Begründung hierfür ist, dass Bäume in vielen Teilgebieten der Informatik eine wichtige Rolle spielen und somit davon auszugehen ist, dass auch zukünftige Komponenten von der Darstellung eines Baumes Gebrauch machen werden. In Abbildung 5.18 ist zu sehen, wie mit Hilfe der Tree-Renderer-Komponente ein Baum auch unabhängig von der Logik-Parser-Komponente dargestellt werden kann. In diesem Abschnitt ist die Funktionalität dieser Komponente beschrieben.

Als Grundlage für die Tree-Renderer-Komponente wird, wie auch für die PLA-Komponente, die SVG-Komponente verwendet. Das erlaubt es dem Benutzer, die Zoom- und Pan-Bedienkonzepte zu verwenden, mit denen es möglich ist, auch einen umfangreichen Baum auf kleinen Bildschirmen im Detail zu betrachten.

Schnittstelle

Um den Operatorbaum für die Logic-Expression-Komponente darzustellen, muss die Tree-Renderer-Komponente die Knoten des Baumes in verschiedenen Farben und mit jeweils einem Bezeichner beschriftet darstellen können.

```
{ // The tree simply as nested objects.
    name: "Root", // The name of the node
    color: "black", // The color of the node
    hidden: false, // If the node should be faded out
    children: [ // The child nodes
    // {
    //   name: "Child",
    //   color: ...,
    //   hidden: false,
    //   children: [...]
    // },
    ],
}
```

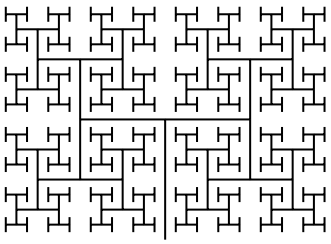
Quelltext 5.10: Die Eingabeschnittstelle der Tree-Komponente

In Quelltext 5.10 ist die Struktur eines Objektes dargestellt, welches von der Komponente als Eingabe erwartet wird. Das Objekt stellt den Wurzelknoten des darzustellenden Baumes dar und enthält in dem `children`-Attribut ein Array aus Kindknoten, deren Struktur der des Wurzelknotens entspricht. Die Eingabe des Tree-Renderer spiegelt die Struktur des Baumes, der dargestellt werden soll, also wider. Das `name`-Attribut gibt die Beschriftung des Knotens an. `color` gibt entsprechend die Farbe an und muss ein gültiger CSS-Farbwert sein. Das Attribut `hidden` beinhaltet einen Wahrheitswert, der angibt, ob der der Knoten verblasst dargestellt werden soll.

Baum-Layout

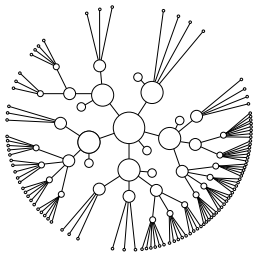
Essenziell für die übersichtliche Darstellung eines Baumes ist es, dass die Knoten sinnvoll positioniert werden. Die Kanten des Graphen sollten sich nicht überschneiden. Knoten sollten nicht unnötig weit voneinander entfernt sein, sollten aber auch nicht so dicht nebeneinander liegen, dass sich ihre Beschriftungen überschneiden.

Es gibt ein Vielzahl an Möglichkeiten, Graphen und insbesondere Bäume grafisch darzustellen. Abbildung 5.19 auf der nächsten Seite zeigt beispielsweise eine Baum, der mittels H-Layout dargestellt wird.[MCHEDLIDZE et al. 2012]



Quelle: [FISER 2011]

Abbildung 5.19: H-Layout eines Baumes



Quelle: [OYSTER 2014]

Abbildung 5.20: Radiales Layout eines Baumes

In Abbildung 5.20 hingegen ist ein Baum zu sehen, dessen Knoten radial um den Wurzelknoten, welcher sich im Mittelpunkt befindet, angeordnet sind.

Diese Komponente wird derzeit in erster Linie für die Darstellung von Operatorbäumen verwendet. Darum soll eine klassiches Baum-Layout verwendet werden, bei dem der Wurzelknoten an oberster Stelle steht und sich die Kindnoten nach unten entfalten, sodass alle Kanten von Eltern- zu Kindknoten monoton fallend sind, wie es schon in Abbildung 5.18 auf Seite 75 zu sehen war. Die vertikale Position eines Knotens soll von seiner Tiefe abhängen. Die horizontale Position eines Knotens soll von der Breite seines Teilbaumes abhängen.

Es existieren bereits einige etablierte Algorithmen zum Zeichnen von Bäumen, die diese Anforderungen erfüllen. Dazu gehört der Algorithmus von Walker, der Algorithmus von Reingold und Tilford, und der Algorithmus von Bucheim, Jünger und Leipert, welcher die beiden vorherigen vereint, um die Laufzeitkomplexität des Walker-Algorithmus von $O(n^2)$ auf $O(n)$ zu verbessern.[WALKER 1990][REINGOLD und TILFORD 1981][BUCHHEIM et al. 2002]

Für diese Komponente wurde eine freie Python-Implementierung des Buchheim-Algorithmus nach JavaScript portiert.[MILL 2008a][MILL 2008b]

5.6 Table-Viewer

identifiers			expressions					
Z0	Z1	Z2	Z0+	Z1+	Z2+	Red	Yellow	Green
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
0	1	0	0	0	1	0	0	0
1	1	0	0	0	0	0	0	0
0	0	1	0	1	1	1	0	0
1	0	1	0	0	0	0	0	0
0	1	1	1	0	0	1	1	0
1	1	1	0	0	0	0	0	0

Abbildung 5.21: Eine von der Table-Viewer-Komponente erzeugte Tabelle

Auch die Darstellung der Funktionstabelle wurde aus der Logic-Expression-Komponente in eine eigene Tabelle extrahiert. Diese Komponente beinhaltet zwar, neben der Möglichkeit eine Zeile per Mausklick auszuwählen, keine weitere eigene Funktionalität, definiert aber Stylesheets für die Darstellung der Tabelle und sorgt damit für ein einheitliches Aussehen von tabellarischen Darstellungen in allen Komponenten. In Abbildung 5.22 auf Seite 83 ist die Darstellung einer solchen Tabelle zu sehen.

```
| Z0 | Z1 | Z2 || Z0+ | Z1+ | Z2+ | Red | Yellow | Green |
|====|====|====||=====|=====|=====|=====|========|=======|
|  0 |  0 |  0 ||  0  |  0  |  0  |  0  |    0   |   0   |
|  1 |  0 |  0 ||  0  |  1  |  0  |  0  |    0   |   1   |
|  0 |  1 |  0 ||  0  |  0  |  1  |  0  |    0   |   0   |
|  1 |  1 |  0 ||  0  |  0  |  0  |  0  |    0   |   0   |
|  0 |  0 |  1 ||  0  |  1  |  1  |  1  |    0   |   0   |
|  1 |  0 |  1 ||  0  |  0  |  0  |  0  |    0   |   0   |
|  0 |  1 |  1 ||  1  |  0  |  0  |  1  |    1   |   0   |
|  1 |  1 |  1 ||  0  |  0  |  0  |  0  |    0   |   0   |
|  0 |  0 |  0 ||  0  |  0  |  0  |  0  |    0   |   0   |
|  1 |  0 |  0 ||  0  |  1  |  0  |  0  |    0   |   1   |
|  0 |  1 |  0 ||  0  |  0  |  1  |  0  |    0   |   0   |
|  1 |  1 |  0 ||  0  |  0  |  0  |  0  |    0   |   0   |
```

Quelltext 5.11: Tabelle als Plaintext

Neben der Darstellung als HTML-Tabelle beinhaltet die Table-Viewer-Komponente auch Funktionen um tabellarische Daten als Text zu formatieren. Ein Beispiel hierfür ist in Quelltext 5.11 zu sehen. Von der Logic-Expression-Komponente wird diese Darstellungsform verwendet, um eine Tabelle als Text exportieren zu können.

Die Schnittstelle

```
{ // the tabel data to display
  columnGroups: [ // the columns groups of the table
    {
      name: "Column Group 1", // name of the group
      columns: [ // the columns of this group
        {name: "First Column"}, // columns have names
        {name: "Second Column"},
      ],
    },
  ],
  rows: [ // the content of the table
    { // first row
      values: [
        // each row has so much values as
        // the total number of columns
        "First value", "the first row",
      ],
    },
    { // second row
      values: ["second", "row"],
    },
  ],
  selectedRow: null, // index of selected row, or null
  error: null, // Error message that is displayed instead of
  ↪ the rows
}
```

Quelltext 5.12: Die Eingabeschnittstelle der Table-Komponente

Als Parameter erwartet die Tabellenkomponente ein Objekt, welches die Spalten und Zeilen der Tabelle beschreibt. Die Struktur ist in Quelltext 5.12 zu sehen. Die Spalten der Tabelle sind in dem Array unter dem `columnGroups` Schlüssel in Gruppen zusammengefasst. Jede Spalte hat als Attribut einen Namen (`name`), welcher in der Kopfzeile der Tabelle angezeigt wird. Das `rows`-Array enthält die Zeilen der Tabelle, die jeweils von einem Objekt mit einem `values` Attribut repräsentiert werden. Dieses Attribut muss ein Array enthalten, dessen Länge mit der Anzahl an Spalten übereinstimmt und als Elemente die Werte der Zeile in der jeweiligen Spalte beinhaltet. `selectedRow` gibt den Index der Zeile an, die als „ausgewählt“ visuell hervorgehoben werden soll. Über das Setzen des `error`-Attributes ist es möglich, anstelle der Zeilen, eine Fehlermeldung anzeigen zu lassen. Dies wird zum Beispiel von der Logic-Expression-Komponente in dem Fall verwendet, dass sich keine Funktionstabelle erzeugen lässt.

Die Grenzen der Leistung

Soll eine Tabelle mit sehr vielen Zeilen (> 1000) kontinuierlich aktualisiert werden, stößt diese Komponenten derzeit an die Grenzen der Leistungsfähigkeit des Document Object Models (DOM) des Browsers und des Vergleichsalgorithmus der Virtual-DOM Bibliothek. In solch einem Fall kann die Aktualisierung der Tabelle mehrere Sekunden in Anspruch nehmen. Aus diesem Grund wurde die Erzeugung von Funktionstabellen in der Logic-Expression-Komponente vorerst auf Ausdrücke mit bis zu acht freien Veränderlichen — also 256 Zeilen — begrenzt. Hierbei handelt es sich aber nicht um eine grundsätzliche Leistungsgrenze des Browsers, der DOM-Schnittstelle, Virtual-DOM oder CycleJS, sondern lediglich um ein Problem der aktuellen Implementierung der Komponente. Im Rahmen dieser Arbeit war es zeitlich nicht möglich, die nötigen Optimierungen an dieser Komponente vorzunehmen. Eine erhebliche Verbesserung der Performance könnte erreicht werden, indem die Höhen und Breiten der Tabellenzellen schon seitens JavaScript festgelegt werden und nur so viele Zeilen als DOM-Elemente erzeugt werden, wie gleichzeitig auf dem Bildschirm des Benutzers dargestellt werden können. Damit liessen sich die für die Darstellung der Tabelle nötigen Layout-Berechnungen des Browsers reduzieren.[WOO 2015][ZACHARY 2012]

5.7 Logic-Checker

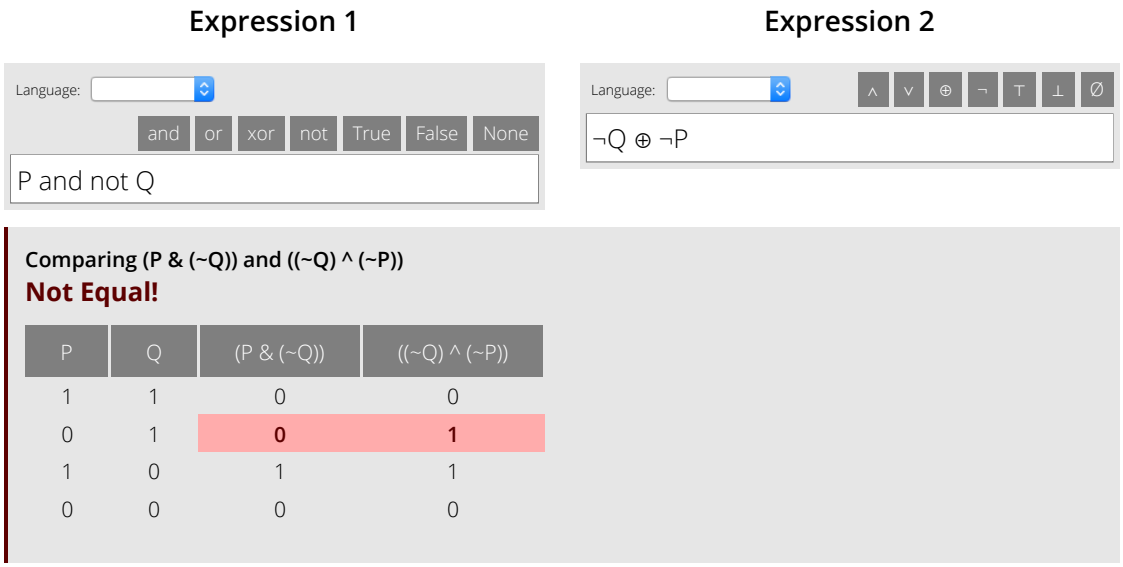


Abbildung 5.22: Die Logic-Checker-Komponente

Die Logic-Checker-Komponente erlaubt, genau wie die Expression-Parser-Komponente, das Eingeben logischer Ausdrücke. Allerdings stellt sie eine vereinfachte Benutzeroberfläche bereit. Sie ist speziell für den Anwendungsfall ausgelegt, die Funktionswerte zweier Ausdrücke zu vergleichen, um die Korrektur von Klausur- oder Übungsaufgaben zu vereinfachen. In Abbildung 5.22 ist die Benutzeroberfläche der Komponente dargestellt. Anders als die Logic-Expression-Parser-Komponente bietet der Logic-Checker zwei Eingabefelder und verzichtet dafür auf die Darstellung des Operatorbaums.

Die Eingabe

In die Textfelder können je ein oder mehrere Ausdrücke eingegeben werden. Mehrere Ausdrücke in einem Textfeld können, wie auch in der Logic-Expression-Parser-Komponente, per Komma von einander getrennt werden. Der Dialekt der Ausdrücke kann für die beiden Textfelder getrennt voneinander eingestellt werden. Damit die Ausdrücke sinnvoll miteinander verglichen werden können, müssen folgende Bedingungen erfüllt sein:

1. Die Anzahl der Ausdrücke muss in beiden Textfeldern übereinstimmen
2. Die Anzahl der benannten Ausdrücke muss in beiden Textfeldern übereinstimmen
3. Die Anzahl der anonymen Ausdrücke muss in beiden Textfeldern übereinstimmen

4. Die Anzahl der freien Veränderlichen muss in beiden Textfeldern übereinstimmen

5. Die Namen der benannten Ausdrücke müssen in beiden Textfeldern übereinstimmen

So können die Eingaben „$P \wedge Q, P \vee Q$“ und „$P \wedge Q$“ nicht miteinander verglichen werden, weil die erste Eingabe zwei Ausdrücke enthält und letztere Eingabe nur einen Ausdruck und somit Bedingung 1 verletzt ist.

Eben so können die Eingaben „$A = P \wedge Q$“ und „$P \vee Q$“ nicht miteinander verglichen werden, da der eine Ausdruck als A benannt ist und der andere nicht (Bedingungen 2 und 3).

Die beiden Eingaben „$A = P \wedge Q$“ und „$B = P \vee Q$“ können ebenfalls nicht verglichen werden. Zwar handelt es sich bei beiden Eingaben um einen benannten Ausdruck, allerdings ist der eine als A und der andere als B benannt, womit Bedingung 5 verletzt ist.

Auch ein Vergleich der Eingaben „$P \wedge Q \wedge R$“ und „$P \wedge Q$“ wird nicht akzeptiert, da die Anzahl der freien Veränderlichen in den Ausdrücken nicht übereinstimmt (Bedingung 4).

Diese Bedingungen wurden absichtlich sehr streng gewählt, um eine Eindeutigkeit der Benutzereingabe zu gewährleisten. Es soll verhindert werden, dass der Benutzer zwei Ausdrücke miteinander vergleicht, die inhaltlich keinen Sinn ergeben.

Die Bedingung 4 wurde als einzige etwas lockerer formuliert. So müssen die freien Veränderlichen nur in ihrer Anzahl und nicht nominal übereinstimmen. Damit soll der Anwendungsfall erleichtert werden, die Musterlösung einer Aufgabe mit einer gegebenen Lösung zu vergleichen. In einer Musterlösung könnte ein Ausdruck beispielsweise als $Rot \wedge Gruen \vee \neg Blau$ gegeben sein. In dem zu vergleichenden Ausdruck könnten die Veränderlichen als R, G und B benannt sein. Es soll nun problemlos möglich sein, die Ausdrücke $Rot \wedge Gruen \vee \neg Blau$ und $\neg(\neg(R \wedge G) \wedge B)$ zu vergleichen. Um dies zu ermöglichen, werden je zwei Veränderliche, die nur in einer der beiden Eingaben vorkommen, automatisch unifiziert.

Diese Unifizierung findet automatisch anhand der Reihenfolge, in denen die Veränderlichen auftreten, statt. Die erste Veränderliche, aus der linken Eingabe, wird mit der ersten Veränderlichen der rechten Eingabe unifiziert. So können beispielsweise die Eingaben $A \wedge B$ und $X \wedge Y$ miteinander verglichen werden. Dabei wird A mit X unifiziert und B mit Y. Sollte dies nicht die vom Benutzer gewünschte Unifizierung sein, kann er selbst die Kontrolle übernehmen, indem er jeder der Eingaben eine Liste mit Veränderlichen

voran stellt: Für die Eingaben „$A, B, A \wedge B$“ und „$Y, X, X \wedge Y$“ wird nun A mit Y und B mit X unifiziert.

Im Falle einer Unifizierung wird der Benutzer darüber informiert, welche Veränderlichen miteinander vereinheitlicht wurden.

Der Vergleich

Wie schon zuvor erklärt, können in ein Textfeld mehrere Ausdrücke eingegeben werden. Für den Vergleich werden die Ausdrücke aus beiden Eingabefeldern paarweise gruppiert. Zwei benannte Ausdrücke werden einander anhand ihres Namens zugeordnet. Für anonyme Ausdrücke findet eine Zuordnung ähnlich der Unifizierung statt: Der n'te anonyme Ausdruck aus dem linken Eingabefeld, wird mit dem n'ten anonymen Ausdruck des rechten Eingabefeldes verglichen. Es finden also genau so viele Vergleiche statt, wie jeweils Ausdrücke in die Felder eingegeben wurden.

Für jeden Vergleich erhält der Benutzer als Ergebnis eine Wahrheitstafel, in welcher die Belegung der Veränderlichen und die Funktionswerte der beiden verglichenen Ausdrücke aufgeführt sind. Zeilen für deren Belegungen sich die Funktionswerte der beiden Ausdrücke unterscheiden, werden in rot markiert hervorgehoben, sodass sich auf einen Blick erfassen lässt, ob zwei Ausdrücke übereinstimmen, und wenn nicht, worin sie sich unterscheiden.

5.8 Zahlenkreis

Die Zahlenkreis-Komponente soll die möglichen Kodierungen von Ganzzahlen im Binärsystem veranschaulichen. Mittels einer N-stelligen Binärzahl (N Bit) lassen sich genau 2^N unterschiedliche Werte modellieren. Sollen nur positive Zahlen kodiert werden, entspricht die Kodierung dem aus dem Dezimal System bekannten Schema, bei welchem jede $n'te$ Stelle mit n multipliziert und dann alle Produkte aufsummiert werden. Für die Kodierung von negativen Zahlen, muss auch das Vorzeichen kodiert werden. Hierfür gibt es unterschiedliche Möglichkeiten: Signed-Kodierung, Einerkomplement und Zweierkomplement.

Die ersten beiden haben den Nachteil, dass die Null als zwei unterschiedliche Werte kodiert wird, wovon einer die negative und einer die positive Null repräsentiert. Das Zweierkomplement bietet eine Kodierung, welche sowohl positive als auch negative Ganzzahlen abbilden kann, aber nur einen einzigen Nullwert kodiert.

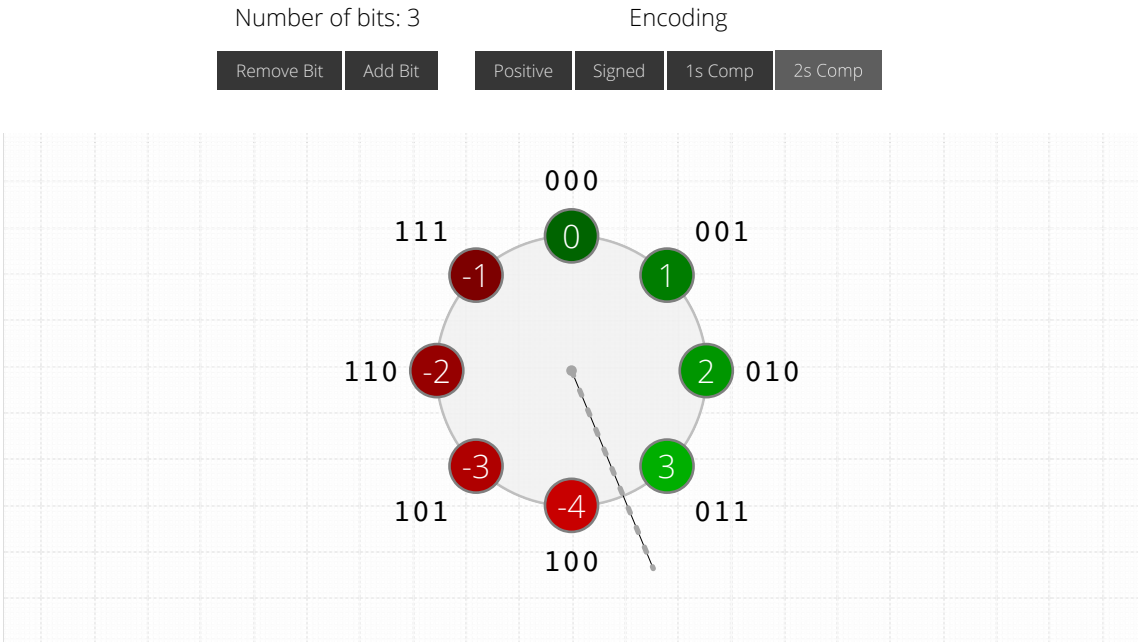


Abbildung 5.23: Die Zahlenkreis-Komponente

In Abbildung 5.23 ist die Benutzeroberfläche der Zahlenkreis-Komponente zu sehen. Über die Button „Add Bit“ und „Remove Bit“ kann die Größe bzw. der Umfang des Kreises variiert werden. Die möglichen Bitmuster sind auf dem Umfang des Kreises angeordnet.

Über die vier Button auf der rechten Seite lässt sich die zu verwendene Kodierung umschalten. Je nach gewählter Kodierung wird jedem Bitmuster im Kreis ein Dezimalwert zugeordnet. Die Dezimalwerte sind ebenfalls auf dem Umfang des Kreises neben den Bit-

mustern angeordnet und farbig hinterlegt dargestellt. Positive Dezimalwerte sind grün hinterlegt. Negative Dezimalwerte hingegen sind rot hinterlegt. Je größer der Betrag des Dezimalwertes, umso heller ist die Hintergrundfarbe des Wertes. So lässt sich die Verteilung der Zahlenwerte auf einen Blick erkennen.

Je nach gewählter Kodierung treffen an ein oder zwei Stellen auf dem Umfang des Kreises negative und positive Zahlen aufeinander. So sind in Abbildung 5.23 auf der vorherigen Seite die Werte -4 und 3 direkte Nachbarn, obwohl sie auf einem gewöhnlichen Zahlenstrahl weit auseinander liegen. Diese Stellen, an denen ein sogenannter Überlauf stattfindet, sind mit einer gestrichelten Linie hervorgehoben.

Auch diese Komponente basiert auf der SVG-Viewer-Komponente und erlaubt es somit den Zahlenkreis zu zoomen. Das ist hilfreich, um auch große Zahlenkreise mit höheren Bitzahlen (bis 7) noch bequem betrachten zu können.

5.9 FSM-Editor

Die FSM-Editor-Komponente, soll das Modellieren und Simulieren Endlicher Automaten ermöglichen. Es sollen sowohl Moore- als auch Mealy-Automaten modelliert werden können.[MEALY 1955][MOORE 1956]

Die Vorlage für diese Komponente stellt das „FSM editor and simulator“ Java-Applet dar, welches von Karola Krönert und Ulrich Dallmann am Arbeitsbereich TAMS der Universität Hamburg entwickelt wurde.[KRÖNERT und DALLMANN 2000]

Aus Zeitmangel, wurde diese Komponente innerhalb dieser Arbeit nicht vollständig implementiert. Im Folgenden werden die Anforderung und der derzeitige Entwicklungsstand der Komponente dargelegt.

Die Anforderungen

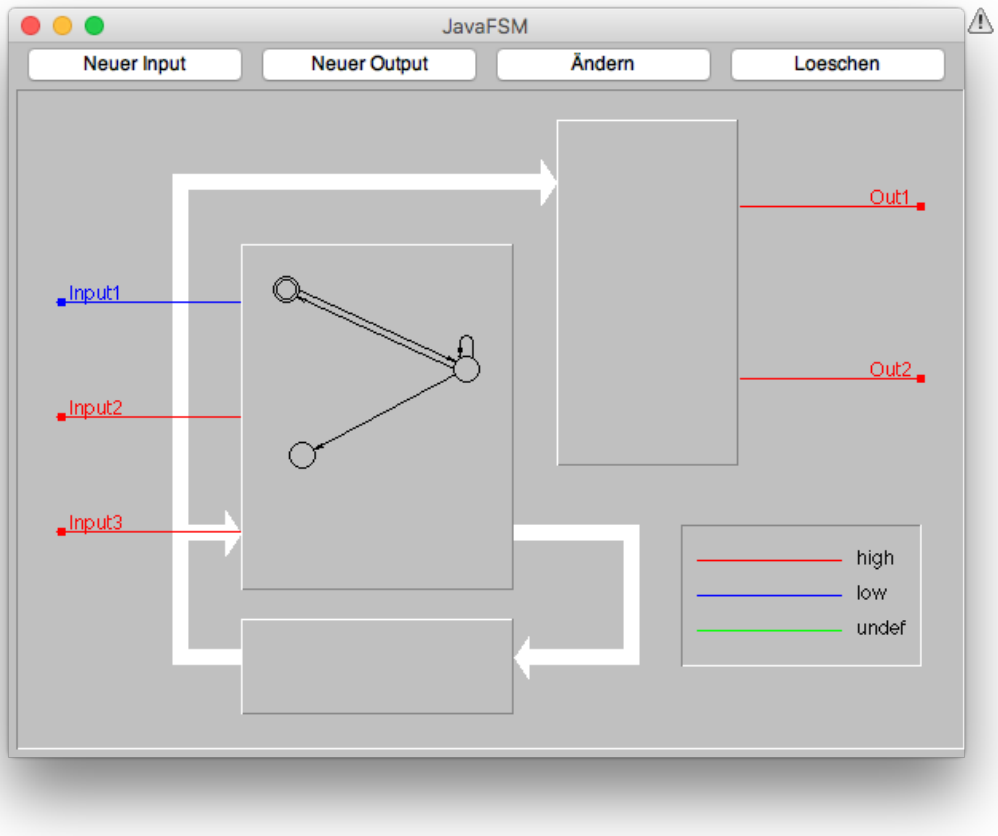


Abbildung 5.24: Hauptfenster des alten Java FSM-Applets

Im Folgenden wird ein Überblick über die Funktionen des Java-Applets gegeben, welche auch von der FSM-Editor-Komponente unterstützt werden sollen.

In Abbildung 5.24 auf der vorherigen Seite ist das Hauptfenster des FSM-Applets zu sehen. Auf der linken Seite sind die Eingänge des Automaten dargestellt. Auf der rechten Seite sind die Ausgänge dargestellt. Die Button am oberen Rand des Fensters lassen den Benutzer Eingänge und Ausgänge hinzufügen, entfernen oder bearbeiten bzw. umbenennen. Die Ein- und Ausgänge des Automaten sind abhängig vom Signal, welches an ihnen anliegt, eingefärbt. In der Mitte des Fensters ist ein Graph dargestellt, welcher die Übergangsfunktion des Automaten beschreibt.

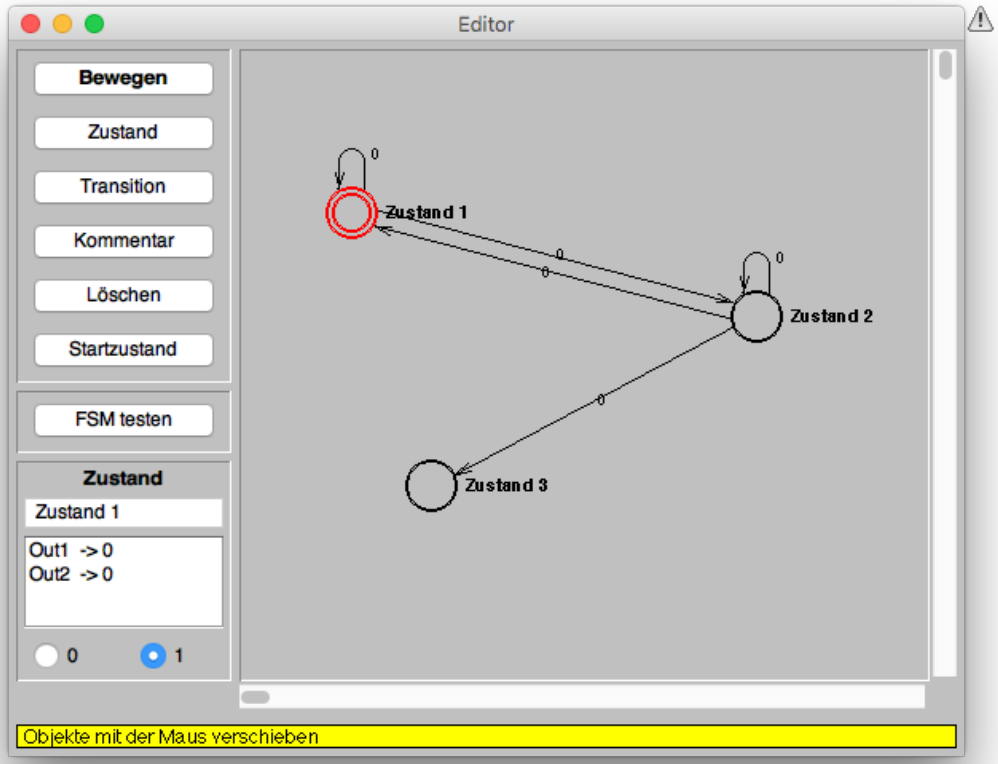


Abbildung 5.25: Der Graph-Editor des alten FSM-Applets

Der Übergangsgraph des Automaten kann in einem eigenen Fenster bearbeitet werden. Dieses ist in Abbildung 5.25 dargestellt. Der Graph-Editor des Applets verfügt über mehrere Modi, die es erlauben Knoten bzw. Zustände zu dem Graphen hinzuzufügen, zu verschieben und zu entfernen, Kanten hinzuzufügen, zu entfernen und deren Übergangsbedingungen zu bearbeiten. Außerdem kann der Übergangsgraph automatisch auf Vollständigkeit überprüft werden.

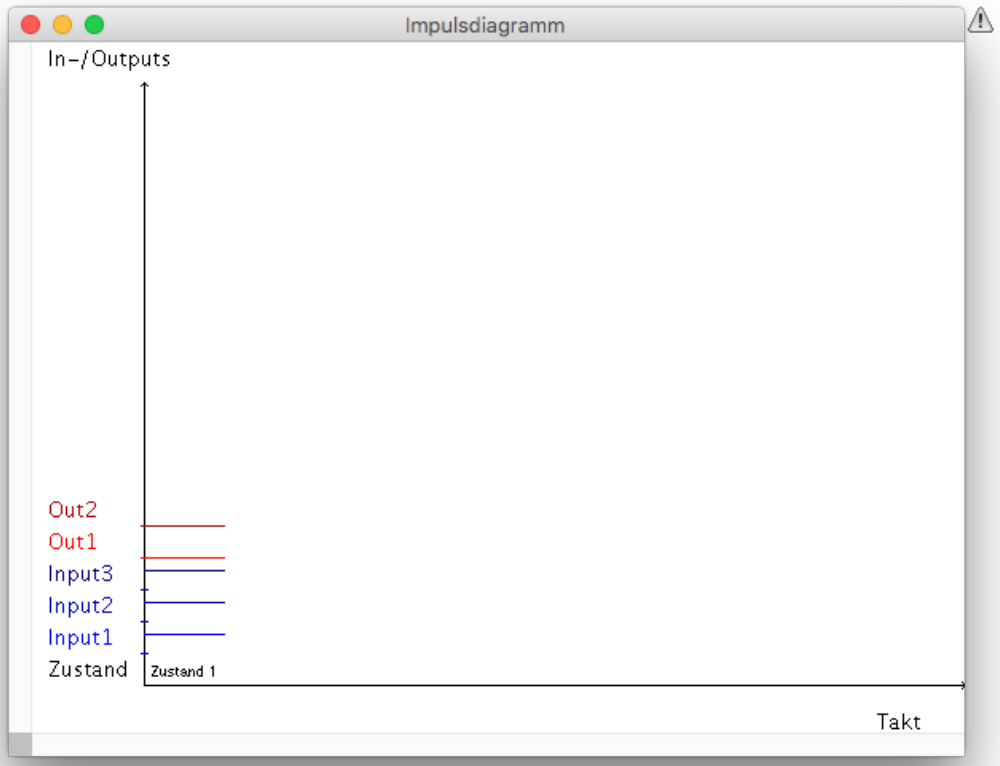


Abbildung 5.26: Impulsdiagramm im alten FSM-Applet

Ist ein Automat fertig modelliert, kann er im Simulationsmodus des Applets simuliert werden. Hierfür stellt das Applet ein eigenes Fenster bereit, welches — wie in Abbildung 5.26 zu sehen — ein Impulsdiagramm beinhaltet. In diesem sind sie Werte der Eingabe- und Ausgabesignale des Automatens, sowie der Zustand, in welchem sich der Automat befindet, über die Zeit abgebildet.

Die Dokumentation des FSM-Applets listet die Funktionen des Applets wie folgt auf:

1. Auswahl zwischen Mealy- und More-Schaltwerken

2. Festlegen der Ein-/ und Ausgänge Entwurf des zugrundeliegenden Automaten mit Hilfe des Editors

3. Definition der logischen Übergangsbedingungen und Ausgabefunktionen

4. Überprüfen des Automaten auf Korrektheit

5. Simulation durch Verändern der Eingangswerte und Taktgebung

6. Ausgabe des Simulationsergebnisses im Impulsdiagramm

7. Laden von Beispielautomaten

8. Speichern und Laden von entworfenen Automaten

9. Export des Automaten im VHDL/KISS-Format

[KRÖNERT und DALLMANN 2000]

Diese Funktionen sollen auch von der FSM-Komponente unterstützt werden.

Benutzeroberfläche

Das FSM-Applet macht Gebrauch von mehreren Fenstern um die unterschiedlichen Teilfunktionen der Anwendung zu gruppieren. Wie schon für die KV-Komponente erklärt, eignet sich die Verwendung mehrere Fenster für Webanwendungen, und besonders für Mobilgeräte, nicht. Daher muss das Konzept der Benutzeroberfläche für diese Komponente neu entwickelt werden.

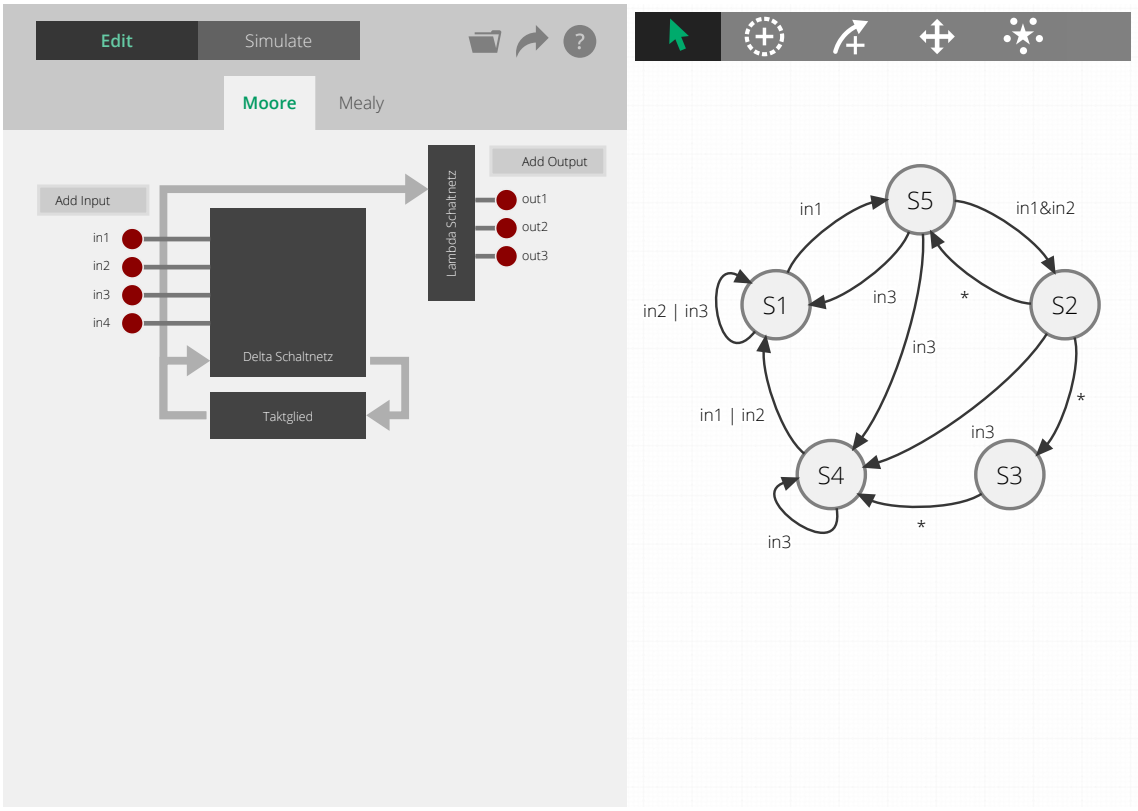


Abbildung 5.27: Die FSM-Komponente

Abbildung 5.27 auf der vorherigen Seite stellt die derzeitige Version der Benutzeroberfläche der FSM-Komponente dar. Diese Grundstruktur wurde stark an die KV-Diagramm-Komponente angelehnt. Auf der linken Seite befindet sich eine Schematische Darstellung des Automaten, welche strukturell aus dem FSM-Applet übernommen wurde. Der Automat wird in drei Elemente — „Taktglied“, „Delta Schaltnetz“ und „Lambda Schaltnetz“ — geteilt dargestellt. Pfeile symbolisieren den Datenfluss zwischen den Elementen.

Wie im FSM-Applet, werden auf der linken Seite die Eingänge und auf der rechten Seite die Ausgänge des Automaten dargestellt. Oberhalb der Ein- und Ausgänge befinden sich Button zum Hinzufügen selbiger. Löschen lassen sich die Ein- und Ausgänge per Mausklick auf den jeweiligen roten Kreis.

Zwei Registerkarten oberhalb der schematischen Darstellung erlauben es, zwischen dem Moore- und Mealy-Automatenmodell zu wechseln.

Noch weiter oben befinden sich zwei Button, die es erlauben, zwischen dem „Bearbeiten“- und dem „Simulation“-Modus hin und her zu schalten. Der Simulationsmodus ist bisher noch nicht implementiert. So ist noch offen, wo genau das Impulsdiagramm dargestellt wird.

Wie auch in den anderen Komponenten stehen Button für das Öffnen und Speichern von Automaten zur Verfügung.

Auf der rechten Seite der Komponente wird der Übergangsgraph des Automaten dargestellt. Es stehen vier verschiedene Modi zur Verfügung, um die Elemente des Graphen zu bearbeiten:

1. Auswahl von Knoten und Kanten
2. Knoten hinzufügen
3. Kante hinzufügen
4. Knoten verschieben

Diese lassen sich über Ikonen am oberen Rand auswählen. Ein Klick auf die fünfte Ikone ordnet alle Knoten des Graphen in einem Kreis an, und nimmt es dem Benutzer ab, die Knoten selbst sorgfältig übersichtlich positionieren zu müssen.

Die Eigenschaften eines Knotens oder einer Kante des Graphen lassen sich bearbeiten, indem dieser per Mausklick — oder Touch — ausgewählt wurde. Wurde ein Elemente des Graphen so ausgewählt, erscheint auf der linken Seite der Komponente ein Panel,

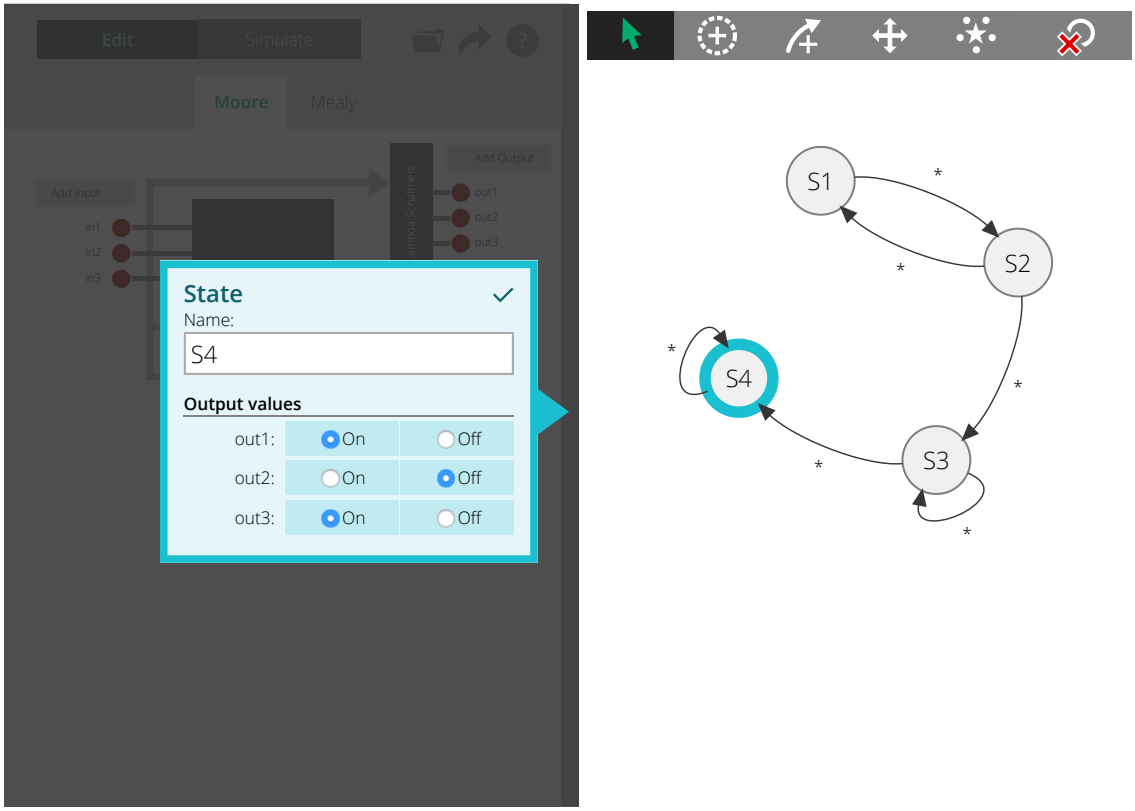


Abbildung 5.28: Auswahl eines Zustands in der FSM-Komponente

welches die Attribute des Knoten bzw. der Kante darstellt. In Abbildung 5.28 ist dieses Panel für die Auswahl eines Knotens zu sehen. Das Panel überdeckt die schematische Darstellung des Automaten, weil diese überlicherweise nicht benötigt wird, während der Übergangsgraph bearbeitet wird. Das Panel ist türkis gefärbt, um dem Benutzer zu zeigen, dass es sich auf den ausgewählten Knoten, der ebenfalls in türkis umrandet ist, bezieht.

Anders als im FSM-Applet, gibt es keinen expliziten Modus für das Löschen von Knoten und Kanten. Stattdessen kann der jeweils ausgewählte Knoten, bzw. die ausgewählte Kante, per Klick auf die entsprechende „Löschen“-Ikone entfernt werden. Diese Ikone ist nur verfügbar, während ein Element des Graphen ausgewählt ist und wird in der oberen rechten Ecke der Komponente angezeigt, wie in Abbildung 5.28 zu sehen ist.

5.10 Graph-Editor

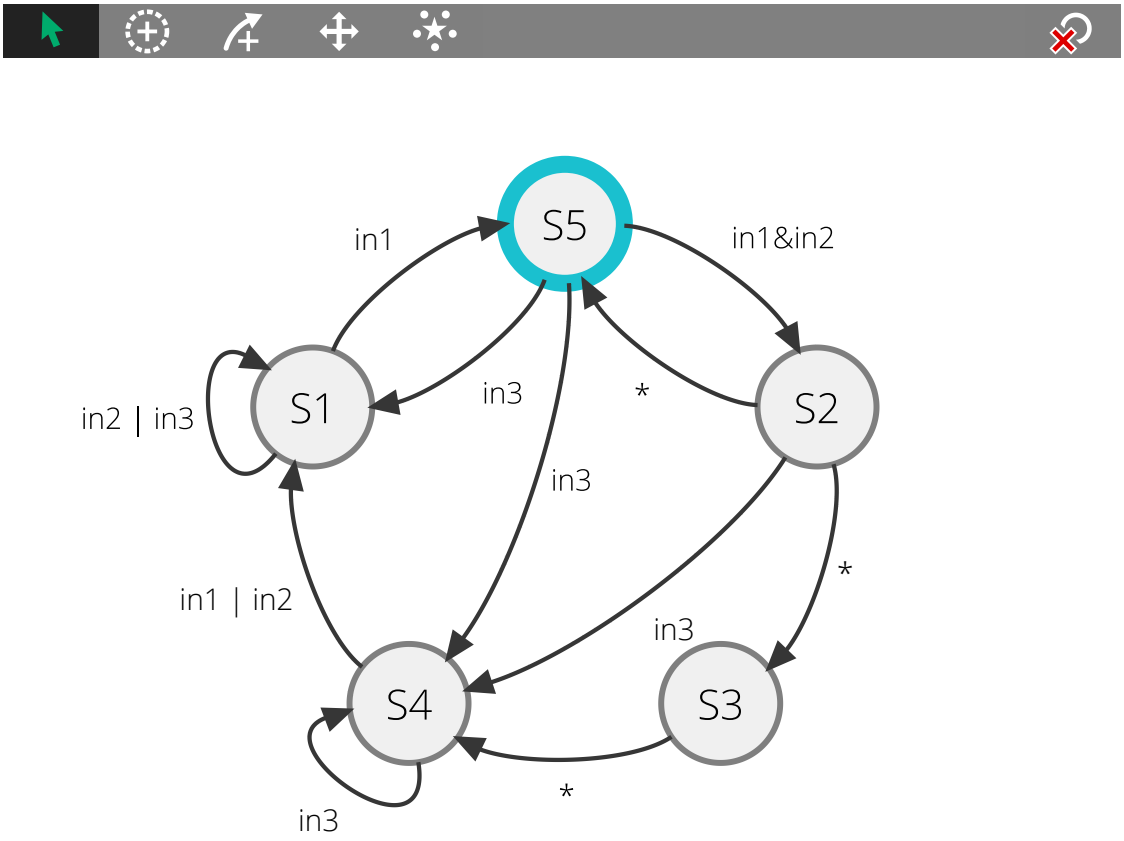


Abbildung 5.29: Die Graph-Komponente

Das Bearbeiten von Gerichteten Graphen kann auch unabhängig von der FSM-Komponente sinnvoll sein. Daher wurde diese Funktionalität in eine eigene Komponente, den Graph-Editor, ausgelagert. Wie schon im Abschnitt zur FSM-Komponente beschrieben, bietet der Graph-Editor vier Modi:

1. Auswahl von Knoten und Kanten
2. Knoten hinzufügen
3. Kante hinzufügen
4. Knoten verschieben

Ausgewählte Knoten und Kanten können entfernt werden. Zudem erlaubt er es, alle Knoten des Graphen automatisch kreisförmig anzuordnen, um eine übersichtliche Darstellung des Graphen zu erhalten.

Selbstverständlich setzt auch diese Komponente auf dem SVG-Viewer auf, sodass sich der zu bearbeitende Graph bequem zoomen lässt. Besonders hierbei ist, dass die Größe der Szene kontinuierlich aus den Positionen der Knoten berechnet wird. Je weiter voneinander entfernt der Benutzer die Knoten des Graphen positioniert, umso mehr Freiraum hat er für die Bewegung der Kamera.

Graph bearbeiten

Im „Knoten hinzufügen"-Modus kann der Benutzer per Klick einen neuen Knoten zum Graph hinzufügen. Mehrmaliges klicken erzeugt mehrere Knoten. Lässt der Benutzer die Maustaste nicht sofort los, sondern hält sie gedrückt, erhält er eine Vorschau des neuen Knoten und kann dessen Position durch Bewegen der Maus noch ändern, bevor dieser endgültig erzeugt wird. Selbes gilt für die Bedienung per Touchscreen: Der neue Knoten wird erst beim Loslassen erzeugt. Ein neu erzeugter Knoten wird automatisch selektiert.

Das Erzeugen einer Kante im „Kante hinzufügen"-Modus erfolg per Drag-'n-Drop. Es wird per gedrückter Maustaste vom Start bis zum Zielknoten gezogen. Auf einem Touchscreen wird der Startknoten mit dem Finger berührt und der Finger dann, ohne ihn vom Bildschirm zu heben, bis zum Zielknoten gezogen. Dort wird der Finger dann vom Bildschirm angehoben. Lässt der Benutzer die Maustaste — oder den Finger — schon direkt über dem Startknoten wieder los, wird eine reflexive Kante erzeugt. Während die Maustaste noch gedrückt ist, wird schon eine Vorschau der zukünftigen Kante angezeigt, um dem Benutzer zu signalisieren, ob ein gültiger Zielknoten erkannt wurde. Befindet sich der Mauszeiger — oder Finger — über einem gültigen Zielknoten, wird eine grüne Kante zwischen dem Start- und dem Zielknoten gezeichnet. Befindet sich der Mauszeiger hingegen nicht über einem Zielknoten, wird eine rote, gestrichelte Kante vom Startknoten bis zur Zeigerposition gezeichnet. Wird die Maustaste losgelassen, während kein gültiger Zielknoten erkannt wurde, wird das Erzeugen der Kante abgebrochen. Eine neu erzeugte Kante wird automatisch selektiert. Mehrfachkanten zwischen zwei Knoten werden verworfen. Im „Knoten verschieben"-Modus können Knoten per Drag-'n-Drop verschoben werden.

Sowohl das Erzeugen von Knoten und Kanten, als auch das Verschieben von Knoten, kann, während die Maustaste noch gedrückt ist, per ESC-Taste abgebrochen werden. Auf einem Touchscreen steht diese Funktionalität im Moment nicht zur Verfügung. Allerdings handelt es sich hierbei um keine essenzielle Funktionalität, da Knoten und Kanten auch nach ihrer Erzeugung problemlos wieder gelöscht werden können und verschobene Knoten zurückgeschoben werden können.

Schöne Kanten

Ein besonderer Fokus wurde bei der Implementierung dieser Komponente auf das Zeichnen der Kanten gelegt. Diese sollen, anders als im Graph-Editor des FSM-Applets, nicht einfach als grade Strecken gezeichnet werden, sondern als geschwungene Pfeile, um erstens den Graphen ästhetisch ansprechender aussehen zu lassen und zweitens den Abstand zweier symmetrischer Kanten zu erhöhen.

Außerdem sollen reflexive Kanten so gezeichnet werden, dass sie sich nach Möglichkeit nicht mit anderen Kanten ihres Knotens überschneiden.

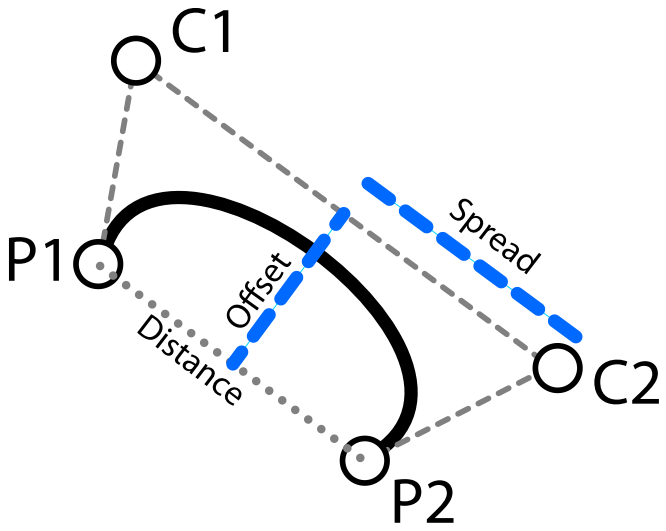


Abbildung 5.30: Bézierkurve der Kante eines Graphen

In Abbildung 5.30 ist die schematische Darstellung einer kubischen Bézierkurve zu sehen. Solch eine Kurve wird verwendet, um die Kanten des Graphen zu zeichnen. Die Endpunkte P1 und P2 der Kurve stellen die Positionen der Start- und Endknoten der Kante dar. Die Kontrollpunkte C1 und C2 bestimmen die Krümmung der Kurve. Für die Darstellung der Kanten des Graphen werden die beiden Kontrollpunkte symmetrisch zur Mittelsenkrechten der beiden Punkte positioniert. So hängt die Position der Kontrollpunkte — und damit die Krümmung der Kante — nur von den Parametern Offset und Spread ab.[HOLLDACK 2011]

$$\text{Offset}(\text{Radius}, \text{Distance}) = \frac{18 * \text{Radius}}{clamp(\text{Distance} - 1.5 * \text{Radius}, \text{Radius}, 4 * \text{Radius})^{0.509}}$$

$$\text{Spread}(\text{Radius}, \text{Distance}) = max(\frac{\text{Distance}}{5}, \text{Offset}(\text{Radius}, \text{Distance}))$$

Quelltext 5.13: Funktionen zur Berechnung der Kante eines Graphen

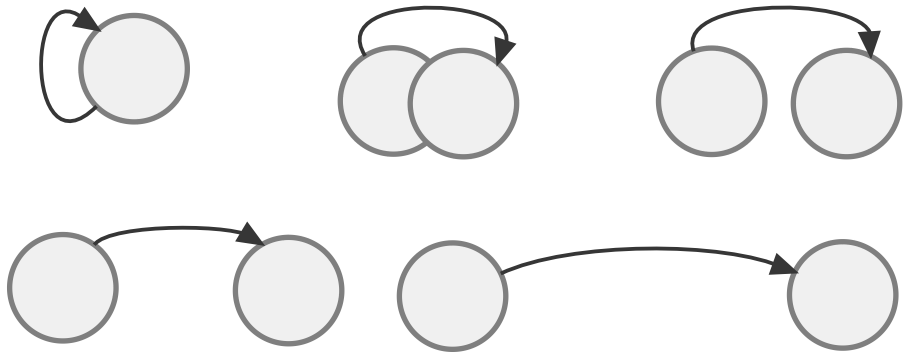


Abbildung 5.31: Krümmung der Kanten eines Graphen

Diese beiden Parameter werden als Funktion des Abstandes (Distance) der beiden Knoten berechnet. Große Werte für Spread und Offset ergeben eine starke Krümmung der Kurve. Kleinere Werte ergeben eine schwache Krümmung. Experimentell haben sich für Offset und Spread die in Quelltext 5.13 auf der vorherigen Seite zu sehenden Funktionen ergeben, wobei mit Radius der Radius der Knoten des Graphen gemeint ist. In Abbildung 5.31 ist das Ergebnis dieser Berechnungen zu sehen. Die reflexive Kante ist stark gekrümmt. Mit zunehmender Distanz der Knoten nimmt die Krümmung der Kante ab.

Für die Positionierung der reflexiven Kante eines Knoten wird zunächst der durschnittliche Winkel aller anderen Kanten den Knotens berechnet. Die reflexive Kante wird um 180° versetzt zu dem Durchschnittswinkel gezeichnet, um zu vermeiden, dass sie die restlichen Kanten überlappt. Das Ergebnis ist in Abbildung 5.32 auf der nächsten Seite zu sehen. Die reflexiven Kanten an dem oberen rechten und dem unteren mittleren Knoten sind so rotiert, dass sie den anderen Kanten des jeweiligen Knoten gegenüber liegen.

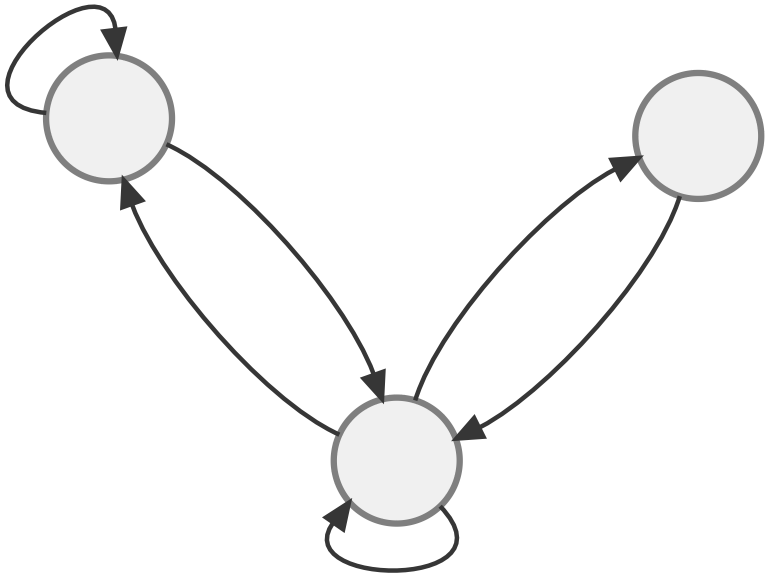
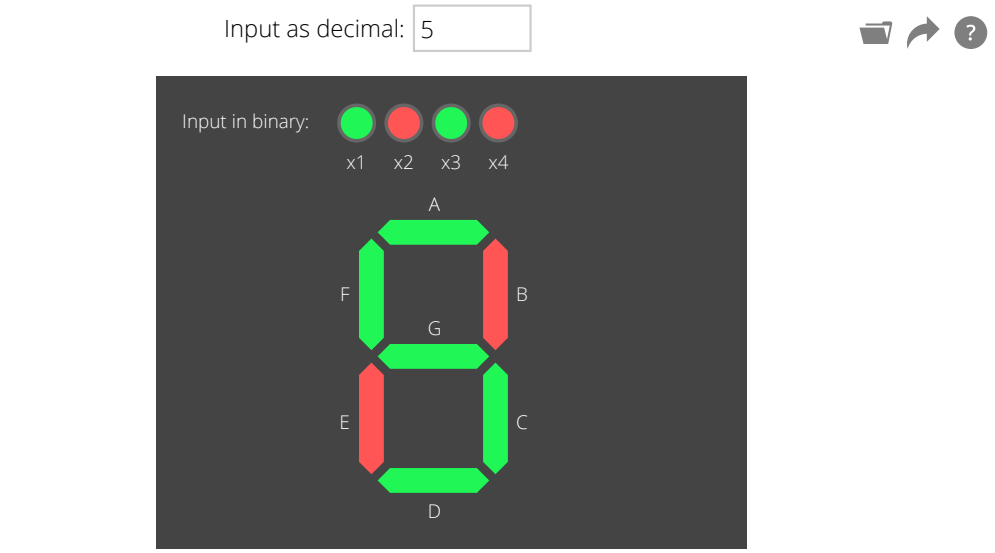


Abbildung 5.32: Reflexive Kanten eines Graphen

5.11 LED-Editor



Inputs				Outputs						
x1	x2	x3	x4	A	B	C	D	E	F	G
0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	0	1	1	0	0	0	0
0	1	0	0	1	1	0	1	1	0	1

Abbildung 5.33: Die LED-Komponente

Die Umsetzung der LED-Editor-Komponente war ursprünglich für diese Arbeit nicht vorgesehen. Während der Entwicklung der KV-Diagramm-Komponente sollte als Beispieldatensatz ein KV-Diagramm für eine 7-Segmentanzeige und einen LED-Würfel angeboten werden. Die Funktionstabellen, die hierfür aus dem alten KV-Java-Applet übernommen werden sollten, stellten sich als womöglich fehlerhaft heraus. Um die Funktionstabelle für eine 7-Segmentanzeige schnell und einfach kontrollieren bzw. sogar erzeugen zu können, wurde die LED-Komponente als Hilfe entwickelt. Zunächst war sie also nur zur Unterstüzung der Entwicklung anderer Komponenten gedacht, wurde dann aber so erweitert, dass sie sich auch außerhalb des Entwicklungsprozesses als eigenständige Komponente verwenden lässt, um den Zusammenhang boolscher Funktionen und Anzeigegeräten, wie eben einer 7-Segmentanzeige, zu erläutern.

Abbildung 5.33 stellt die Benutzeroberfläche der LED-Komponente dar. In dem dunklen Rechteck in der Mitte sind mehrer Leuchtsegmente zu sehen, die jeweils in Rot oder Grün eingefärbt sind. Die runden Leuchtsegmente am oberen Ende des Kastens stellen die Eingangsbelegung einer boolschen Funktion dar. Im Fall von Abbildung 5.33 gibt es vier Eingänge (x1, x2, x3 und x4). Die rote Farbe der Leuchtsegmente stellt einen Eingangswert von 0 bzw. `false` dar. Grün stellt einen Eingangswert von 1 bzw. `true` dar. Die

restlichen Leuchtsegmente, weiter unten im dunklen Rechteck, stellen die Ausgangswerte der Funktion dar. In der Abbildung ist also zu sehen, dass für die Eingangsbelegung $(x1 = 1, x2 = 0, x3 = 1, x4 = 0)$ die Ausgänge A, C, D, F und G eingeschaltet und die B und E ausgeschaltet sind.

Unterhalb des dunklen Rechtecks befindet sich eine Funktionstabelle, in welcher die Funktionswerte für alle Eingangsbelegungen gelistet sind. Oberhalb des Rechtecks befindet sich ein Textfeld, in welchem die Dezimalinterpretation der Eingangsbelegung dargestellt wird.

Die runden Leuchtsegmente der Eingangsbelegung können per Klick ein- und ausgeschaltet werden, um die aktuelle Eingangsbelegung zu wechseln. Eine Eingangsbelegung entspricht genau einer Zeile in der Funktionstabelle. In der Funktionstabelle ist stets die Zeile hervorgehoben, die der aktuellen Eingangsbelegung entspricht. Eine Zeile in der Funktionstabelle kann auch direkt angeklickt werden, um eine gewünschte Eingangsbelegung zu wählen.

Die restlichen Leuchtsegmente, welche die Funktionswerte darstellen, können ebenfalls per Klick ein- und ausgeschaltet werden. Das Umschalten dieser Leuchtsegmente führt zu einer Änderung des zugehörigen Funktionswertes für die aktuell gewählte Eingangsbelegung. Beim Umschalten eines solchen Leuchtsegmentes kann beobachtet werden, wie sich der zugehörige Funktionswert in der Funktionstabelle ändert.

In das obere Eingabefeld, welches die Dezimalinterpretation der Eingangsbelegung anzeigt, kann auch ein Dezimalwert eingegeben werden, um die Eingangsbelegung zu setzen. So kann in das Feld beispielweise eine 9 eingegeben werden, um zu der Eingangsbelegung $(x1 = 1, x2 = 0, x3 = 0, x4 = 1)$ zu wechseln. Für die 7-Segmentanzeige sind unter dieser Belegungen alle Segmente bis auf Segment E eingeschaltet, sodass die Leuchtsegmente ebenfalls die Form einer 9 darstellen.

Über die „Öffnen“-Ikone in der oberen rechten Ecke können auch andere Segment-Konfigurationen geladen werden: Ein LED-Würfel und eine 16-Segmentanzeige.

Der „Export“-Button erlaubt es, die Funktionstabelle zum einen als ASCII-Tabelle und zum anderen als Logischen Ausdruck zu exportieren. Letzterer kann wiederum in die Logik-Parser-Komponente oder in die KV-Diagramm-Komponente importiert werden.

6 Projektstruktur

Dieses Kapitel widmet sich der Projektstruktur dieser Arbeit und zeigt die Zusammenhänge zwischen den einzelnen Entwicklungswerkzeugen auf, um die Einarbeitung in das Projekt zu erleichtern und die zukünftige Pflege und Weiterentwicklung zu ermöglichen.

Zusätzlich zu diesem Kapitel enthält auch das zu dieser Arbeit gehörende Git-Repository einen Ordner namens `docs`, in welchem sich eine Erklärung der Projektstruktur befindet.

Der Einfachheit halber wurden im Rahmen dieser Arbeit alle entwickelten Komponenten in einem gemeinsamen Git-Repository gepflegt und als ein einziges NPM-Packet definiert. Zukünftig mag es sinnvoll sein, die Komponenten als jeweils eigenständiges NPM-Packet in ihr eigenes Repository auszulagern. Doch im Rahmen dieser Arbeit hätte dies zu zusätzlichem organisatorischen Aufwand geführt, sodass vorerst darauf verzichtet wurde. Bei der Definition einzelner Module wurde allerdings bereits darauf geachtet, die spätere Auftrennung in eigene Repositories und NPM-Packetes unproblematisch zu ermöglichen.

6.1 Verzeichnisstruktur

Im Folgenen wird die Verzeichnisstruktur des Projektes erläutert, sowie auf die Rollen der einzelnen Konfigurationsdateien eingegangen. Quelltext 6.1 stellt den Inhalt des Wurzelverzeichnisses des Projektes dar. Zu sehen sind die drei Unterordner `app`, `docs` und `webpack` sowie fünf Textdateien.

```
 /
(/node_modules/)
(/build/)
 /app/
 /docs/
 /webpack/
 /.editorconfig
 /.eslintignore
 /.eslintrc
 /.gitignore
 /package.json
```

Quelltext 6.1: Das Wurzelverzeichnis des Projektes.

1. Der `node_modules` Order wird von NPM angelegt und enthält alle fremden Bibliotheken, die als Abhängigkeiten installiert worden sind

2. Der `build` Ordner wird von Webpack bei der Kompilierung des Projektes angelegt und enthält die kompilierten HTML-, CSS- und JavaScript-Dateien

3. Der `app` Order enthält sämtlichen Quelltext des Projektes

4. Der `docs` Ordner enthält Dokumentationstexte zum Projekt

5. Der `webpack` Ordner enthält Konfigurationsdateien für Webpack

6. `.editorconfig` ist eine Konfiguratonsdatei im INI-Format, in welcher die Quelltextformatierung des Projektes festgelegt ist. Es handelt sich hierbei um eine Konfiguration, die von vielen Texteditoren und Integrated Development Environments (IDE) unterstützt wird. In ihr ist festgelegt, dass die JavaScript-Dateien UTF8-kodiert sein sollen, dass zwei Leerzeichen zur Einrückung verwendet und dass Leerzeichen am Zeilenende automatisch entfernt werden sollen. Es empfielt sich, zur Bearbeitung des Projektes einen Editor zu verwenden, welcher die `.editorconfig`-Einstellungen unterstützt[HUNNER und XU 2011]

7. `.eslintignore`: Es wird ESLint verwendet, um JavaScript-Dateien auf syntaktische und semantische Fehler zu überprüfen. In dieser Datei ist festgelegt, welche Dateien und Ordner von der Überprüfung ausgenommen sein sollen

8. `.eslintrc` ist die Konfigurationsdatei für ESLint. In ihr sind alle Regeln festgelegt, nach denen der JavaScript-Quelltext des Projektes überprüft werden sollen

9. `.gitignore` enthält eine Liste mit Namen von Dateien und Ordnern, die nicht vom Versionskontrollsystem Git verwaltet werden sollen

10. `package.json` ist die Konfigurationsdatei des Projektes für NPM. In ihr sind der Name und die Version des Projektes, sowie alle Abhängigkeiten zu fremden Bibliotheken definiert. Zudem enthält sie eine Liste von Shell-Befehlen, die mittels NPM ausgeführt werden können

package.json

Quelltext 6.2 auf der nächsten Seite zeigt den Inhalt der `package.json`-Datei. Mit `name`, `version`, `description`, `author` sind Metadaten zu dem Projekt definiert. `private`

```
{
  "name": "tams-tools",
  "version": "0.1.0",
  "description": "Set of tools for teaching theoretical
  ↪ computer science",
  "author": "Laszlo Korte <me@laszlokorte.de>
  ↪ (https://www.laszlokorte.de/)",
  "license": "MIT",
  "main": "index.js",
  "private": true,
  "scripts": {
    "compile": "env NODE_ENV=production webpack --config
    ↪ webpack/prod.config.babel.js --progress",
    "lint": "eslint .",
    "start": "webpack-dev-server --config
    ↪ webpack/dev.config.babel.js --progress --inline --hot
    ↪ --host 0.0.0.0",
    "test": "ava",
    "clear": "rm -rf ./build"
  },
  "devDependencies": {
    "babel": "^6.5.2",
    "babel-core": "^6.7.6",
    "babel-eslint": "^6.0.2",
    "babel-loader": "^6.2.4",
    "webpack-dev-server": "^1.14.1",
    "webpack-merge": "^0.10.0"
    // ...
  },
  "dependencies": {
    "@cycle/core": "^6.0.3",
    "@cycle/dom": "^9.4.0",
    "@cycle/isolate": "^1.2.0",
    // ...
  }
}
```

Quelltext 6.2: Die NPM-Packetdefinition des Projektes

gibt an, dass dieses Packet nicht in der offiziellen NPM-Datenbank veröffentlicht werden soll.

`scripts` definiert eine Menge an Befehlen, die von NPM ausgeführt werden sollen. Diese Befehle können in der Shell via `npm run <cmd>`, also beispielsweise `npm run compile` für das `compile`-Script, ausgeführt werden. Die Werte in `scripts` sind einfache Shellbefehle. Diese Befehle in der `package.json` zu definieren, hat zwei Vorteile: Zum einen müssen häufig verwendete, lange Befehle nicht jedes mal per Hand eingege-

ben werden, sondern lassen sich unter einem kürzeren Namen aufrufen. Zum anderen setzt NPM für die Ausführung der Befehle die `$PATH` Variable, sodass alle per NPM installierten ausführbaren Dateien zur Verfügung stehen. So ruft der `compile`-Befehl beispielsweise `webpack` auf. Das `webpack`-Script liegt im `node_modules`-Ordner, sodass es sich auf der Shell nicht ohne Angabe des vollständigen Pfades aufrufen ließe. Wird der Befehl per NPM ausgeführt, kann der Pfad automatisch korrekt aufgelöst werden.

In `devDependencies` und `dependencies` sind die Bibliotheken definiert, die von diesem Projekt benötigt werden. Dabei sind in `devDependencies` die Bibliotheken enthalten, die nur für den Entwicklungsprozess und für die Kompilierung benötigt werden. In `dependencies` hingegen sind die Bibliotheken gelistet, die tatsächlich auch zur Laufzeit von der Anwendung benötigt werden. In der Praxis spielt diese Unterscheidung effektiv keine Rolle, da Webpack alle benötigten Abhängigkeiten zu einer JavaScript-Datei zusammenfasst und keine Bibliotheken zur Laufzeit nachgeladen werden. Der Übersichtlichkeit halber ist die Trennung jedoch trotzdem hilfreich. Konkret sollte die Unterscheidung zwischen `dependencies` und `devDependencies` daran festgemacht werden, ob eine Bibliotheken per `import`-Anweisung im Quelltext des Projektes referenziert wird. Die Versionsnummern von NPM-Packeten sind überlicherweise nach dem Schema des Semantic Versionings definiert.[WILLIAMS 2015][PRESTON-WERNER 2011]

Das app Verzeichnis

```
/app/
/app/components/
/app/drivers/
/app/icons/
/app/lib/
/app/pages/
/app/styles/
/app/index.html
/app/vendor.js
```

Quelltext 6.3: Der Inhalt des app-Verzeichnisses

Das Verzeichnis `app` enthält die tatsächliche Implementierung des Projektes. Wie Quelltext 6.3 zu entnehmen ist, wurde es in sechs Unterordner aufgeteilt:

1. Der Ordner `components` enthält in je einem Unterverzeichnis die Komponenten, die im Rahmen dieser Arbeit implementiert wurden

2. Der Ordner `drivers` enthält Definitionen der Treiberfunktionen für CycleJS. Dabei handelt es sich um Treiber, die nicht offiziell von CycleJS zur Verfügung gestellt

werden, sondern im Rahmen dieser Arbeit implementiert wurden, um kleine Aufgaben, wie die Änderung der Größe eines Textfeldes oder das Einfügen von Text in ein Eingabefeld, zu übernehmen

3. In `icons` sind die in der Benutzeroberfläche der Komponenten verwendeten Ikonen jeweils als SVG-Elemente definiert. Die Ikonen sind nicht in SVG-Dateien, sondern als JavaScript-Funktionen definiert, um die Einbindung in den DOM-Baum der Komponenten zu vereinfachen

4. Das `lib`-Verzeichnis enthält komponentenübergreifene Hilfsfunktionen

5. Im `pages`-Verzeichnis befinden sich Einstiegspunkte für die Komponenten. Nicht jede Komponente hat einen Einstiegspunkt, der vom Benutzer direkt aufgerufen werden kann. Eine Seite (`page`), also ein Einstiegspunkt, kann auch mehrere Komponenten miteinander kombinieren. Die `kvd-editor`-Seite stellt zum Beispiel die KV-Komponente und die PLA-Komponente nebeneinander dar. Die PLA-Komponente kann aber vom Benutzer nicht getrennt von der KV-Komponente aufgerufen werden, weil dafür kein Einstiegspunkt festgelegt ist

6. Der `styles`-Ordner enthält Stylesheets, die komponentenübergreifend verwendet werden. Die Verwendung von gemeinsamen Stylesheets sorgt für das einheitliche Aussehen der Komponente

7. Das `index.html` Dokument ist die Vorlage, die von Webpack verwendet wird, um HTML-Dokumente für alle Einstiegspunkte zu generieren

8. In der `vendor.js` ist eine Liste mit Modulen festgelegt, die von Webpack in eine eigenständige, von den Komponenten unabhängige JavaScript-Datei zusammengefasst werden sollen, um das Potential des HTTP-Caching des Browsers auszunutzen

6.2 NPM

Wenn das Projekt frisch heruntergeladen wurde, oder wenn sich etwas an den in der `package.json`-Datei definierten Abhängigkeiten verändert hat, ist es nötig, per `npm install`-Befehl auf der Shell, die benötigten Bibliotheken zu installieren oder zu aktualisieren. NPM legt hierfür den `node_modules`-Ordner im Hauptverzeichnis des Projektes an, um die heruntergeladenen Bibliotheken darin abzulegen. Dieser Ordner kann jeder Zeit manuell gelöscht und mittels `npm install` erneut initialisiert werden.

NPM Befehle

Um in dem Projekt zu arbeiten, sind praktisch nur die in der `package.json` deklarierten Befehle notwendig. Sie lassen sich mittels NPM über die Shell ausführen:

1. `npm run start` startet den Webpack-Entwicklungsserver
2. `npm run compile` führt die Kompilierung der Quelltextdateien mittels Webpack aus
3. `npm run test` führt die Unittests aus

Der Unterscheid zwischen `compile` und `start` besteht darin, dass `compile` eine einmalige Kompilierung des Quelltextes durchführt und das Ergebnis im `build` Ordner ablegt. Was genau bei der Kompilierung passiert, ist in der Webpack-Konfiguration festgelegt. `start` hingegen erzeugt keine Dateien im Dateisystem, sondern startet einen Webserver auf Port 3000 und stellt die kompilierten HTML-, CSS- und JavaScript-Dateien per HyperText Transfer Protocol (HTTP) bereit. Während dieser Webserver läuft, kann die Applikation über `http://localhost:3000/` im Browser aufgerufen werden.

Während der Entwicklung am Projekt ist es also nicht nötig, den `compile`-Befehl zu verwenden. Ist eine Iteration der Entwicklung abgeschlossen, kann der `compile`-Befehl verwendet werden, um HTML-, CSS- und JavaScript Dateien zu erzeugen, die eigenständig — also ohne NodeJS, NPM, Webpack etc. — funktionieren.

6.3 Webpack

```
/webpack/
/webpack/common.config.babel.js
/webpack/dev.config.babel.js
/webpack/prod.config.babel.js
```

Quelltext 6.4: Die Webpack-Konfigurationsdateien

Die Aufgabe von Webpack ist es, die vielen Dateien aus denen das Projekt besteht, in möglichst wenige, kompakte Dateien zusammenzufassen und dabei eine Reihe von Transformationen am Quelltext vorzunehmen, um diesen für die Funktionsfähigkeit in allen Browsern aufzubereiten.

Wie in Quelltext 6.4 zu sehen ist, ist die Konfiguration von Webpack in drei Dateien verteilt. Das hat den Grund, dass Webpack zum einen im Produktionsmodus (`npm run`

`compile`) und zum anderen im Entwicklungsmodus (`npm run start`) verwendet werden kann.

1. `common.config.babel.js` enthält die Einstellungen, die sowohl für den Produktions- als auch für den Entwicklungsmodus gelten sollen.

2. `prod.config.babel.js` enthält die Einstellungen für die Produktionsmodus.

3. `dev.config.babel.js` enthält die Einstellungen für die Entwicklungsmodus.

Konfiguration

```
1  // ...
2  module.exports = {
3      // ...
4      entry: {
5        home: "./app/pages/home/index.js",
6        kvdEditor: "./app/pages/kvd-editor/index.js",
7        debug: "./app/pages/debug/index.js",
8        logicEditor: "./app/pages/logic-editor/index.js",
9        ledEditor: "./app/pages/led-editor/index.js",
10       fsmEditor: "./app/pages/fsm-editor/index.js",
11       numberCircle: "./app/pages/number-circle/index.js",
12       logicChecker: "./app/pages/logic-checker/index.js",
13       vendor: require("../app/vendor.js"),
14     },
15     // ...
16 }
```

Quelltext 6.5: Die Webpack-Einstiegspunkte

Die `dev` und `prod` Konfigurationen unterscheiden sich hauptsächlich darin, ob die JavaScript-Dateien komprimiert werden sollen und ob der Entwicklungswebserver gestartet werden soll. Für die Entwicklung der Anwendungen müssen diese beiden Konfigurationen nicht verändert werden. Interessanter ist die `common`-Konfiguration. In ihr sind zum einen, wie in Quelltext 6.5 zu sehen, die Einstiegspunkte (`entry`) der Anwendung definiert. Jeder Einstiegspunkt definiert eine JavaScript-Datei, die letztendlich von Webpack erzeugt werden sollen. Für jede Komponente, die Rahmen dieser Arbeit entwickelt wurde, ist ein Einstiegspunkt definiert. Zusätzlich ist ein Einstiegspunkt mit dem Namen `vendor` definiert, unter welchem alle Fremdbibliotheken zusammengefasst werden. Diese Auftrennung bezweckt, dass Bibliotheken, wie CycleJS und RxJS, nicht in jeder Komponente enthalten sein müssen. Das reduziert die Größe der einzelnen

JavaScript-Dateien und ermöglicht es dem Browser, die `vendor.js`-Datei zu cachen und komponentenübergreifend wiederzuverwenden.

```
// ...
module.exports = {
  // ...
  plugins: [
    // ...
    new HtmlWebpackPlugin({
      title: 'KV Diagram Editor',
      minify: htmlMinifyOptions,
      chunks: ['kvdEditor', 'vendor'],
      template: './app/index.html',
      filename: 'kvd-editor.html',
      favicon: './app/pages/kvd-editor/kvd.ico',
    }),
    // ...
  ]
  // ...
};
```

Quelltext 6.6: Erzeugung einer HTML-Datei via Webpack

Um die Einstiegspunkte im Browser auszuführen, wird ein HTML-Dokument benötigt, in welchem die JavaScript-Datei verlinkt wird. Da alle Komponenten vollständig in JavaScript definiert sind und sie ihren DOM-Baum komplett selbst erzeugen, genügt zur Darstellung der Komponenten ein beinah leeres HTML-Dokument, in welches nur der JavaScript-Einstiegspunkt und das zugehörige Stylesheet eingebunden werden muss. Wie in Quelltext 6.6 am Beispiel der KV-Komponente zu sehen ist, wird das HtmlWebpackPlugin verwendet, um jeweils ein HTML-Dokument für die Komponenten zu erzeugen. Im `chunks`-Array ist festgelegt, welche Einstiegspunkte in dem Dokument eingebunden werden sollen.

Des Weiteren ist in der Webpack-Konfiguration eine Liste mit sogenannten `loaders` definiert, wie in Quelltext 6.7 auf der nächsten Seite zu sehen ist. Loaders sind Plugins, die es Webpack erlauben, unterschiedliche Dateitypen richtig zu verarbeiten. In Quelltext 6.7 auf der nächsten Seite sind vier verschiedene Loader angegeben. Der `babel`-Loader sorgt dafür, dass Dateien mit der Dateiendung `js` vom Babel-Compiler verarbeitet werden. Der `pegjs`-Loader ermöglicht es, Dateien mit der Endung `pegjs`, in denen eine Grammatik für einen PegJS-Parser definiert ist, per `import`-Statement direkt im JavaScript zu importieren und automatisch ein Parserobjekt zu erhalten. Entsprechend erlaubt es der `json`-Loader, JSON-Dateien als JavaScript-Objekt zu importieren. Dank des `stylus`-Loaders ist es möglich, Stylesheets per `import`-Anweisung im JavaScript zu laden, ohne sich um die manuelle Verlinkung der CSS-Datei im HTML-Dokument

```
// ...
module.exports = {
  // ...
  plugins: [
    // ...
    new HtmlWebpackPlugin({
      title: 'KV Diagram Editor',
      minify: htmlMinifyOptions,
      chunks: ['kvdEditor', 'vendor'],
      template: './app/index.html',
      filename: 'kvd-editor.html',
      favicon: './app/pages/kvd-editor/kvd.ico',
    }),
    // ...
  ]
  // ...
};
```

Quelltext 6.7: Die in Webpack konfigurierten Loaders

kümmern zu müssen — diese wird von Webpack übernommen. Auch an der Loaderkonfiguration muss üblicherweise während der Entwicklung nichts verändert werden.

Entwicklungsserver

Der Webpack-Entwicklungsserver liefert die, mittels Loader verarbeiteten, JavaScript-Dateien via HTTP an den Browser aus. Gleichzeitig beobachtet er das Dateisystem auf Änderungen an den Quelldateien und löst, im Falle einer Änderung, eine erneute Verarbeitung der Dateien aus. Die Verarbeitung wird inkrementell durchgeführt, sodass die Zeit, die eine erneute Verarbeitung nach einer Änderung benötigt, meist sehr gering ist. Außerdem stellt der Entwicklungserver mit dem Browser, welcher die Dateien abruft, eine WebSocket-Verbindung her. Damit ist es dem Server möglich, Nachrichten an den Browser zu senden, um ihn gegebenenfalls über Änderungen zu informieren und ein Neuladen der Seite zu verursachen. Änderungen an Stylesheets können sogar ohne ein erneutes Laden der gesamten Seite im Browser übernommen werden. Somit ermöglicht der Webpack-Entwicklungsserver es, die im Quelltext vorgenommenen Änderungen direkt im Browser zu sehen, ohne die Seite manuell neuladen zu müssen. Lediglich zu beachten ist, dass die Änderung der Webpack-eigenen Konfigurationsdateien einen Neustart des Servers erfordert.

Source Maps

Die Verwendung von Webpack mit Plugins wie Babel und Stylus führt dazu, dass der CSS- und JavaScript-Quelltext, der letztendlich vom Browser ausgeführt wird, sich strukturell teilweise stark von dem ursprünglich geschriebenen Quelltext unterscheidet.

Der Unterschied zwischen dem vom Entwickler geschriebenen und dem vom Browser ausgeführten Code erschwert das Aufspüren und Beheben von Fehlern enorm, da ein Fehlverhalten des Programms keiner Stelle im geschriebenen Code zugeordnet werden kann. Um diese Problematik zu beheben, wurde von Mozilla und Google das Source Maps Protokoll entwickelt.[JOHN LENZ und NICK FITZGERALD 2011][BASU 2013]

Die Idee von Source Maps ist es, dem Browser nicht nur die verarbeitete CSS- oder JavaScript-Datei auszuliefern, sondern ihm zusätzlich auch die Originaldateien, sowie Metainformationen für die Zuordnung einzelner Zeilen bereitzustellen. Der Browser kann diese Zusatzinformation dann nutzen, um dem Entwickler, im Falle eines Fehlers bzw. einer Exception, die zugehörige Zeile im Originalquelltext zu zeigen. Außerdem können Breakpoints, die vom Entwickler im Originalquelltext gesetzt wurden, automatisch vom Browser auf den verarbeiteten Quelltext übertragen werden.

Webpack ist in diesem Projekt so konfiguriert, dass sowohl im Entwicklungs- als auch Produktionsmodus Source Maps erzeugt werden. So lassen sich die in den Browsern integrierten Entwicklerwerkzeuge problemlos nutzen. Von Firefox werden Source Maps ab Version 23 unterstützt.[SEDDON 2012]

6.4 Neue Komponente anlegen

Für die zukünftige Pflege des Projektes ist es von Interesse, neue Komponenten definieren zu können. Im Folgenen werden die dafür nötigen Schritte erläutert. Neben der hiesigen Beschreibung befindet sich im `docs`-Verzeichnis des Projektes eine ausführlichere Anleitung, die den detailierten Entwicklungsprozess einer kompletten Komponente am Beispiel des Zahlenkreises beschreibt.

Es sind drei Schritte nötig, um eine neue Komponente zu erzeugen.

1. Im `app/components` Verzeichnis wird ein neuer Ordner angelegt. In diesem Ordner wird eine `index.js`-Datei Datei erstellt, welche die Definition der Komponente enthält

```
// [...]
module.exports = {
  target: "web",
  entry: {
// [...]
    ledEditor: "./app/pages/led-editor/index.js",
    fsmEditor: "./app/pages/fsm-editor/index.js",
+   numberCircle: "./app/pages/number-circle/index.js",
    vendor: require("../app/vendor.js"),
  },
// [...]
  plugins: [
// [...]
+   new HtmlWebpackPlugin({
+     title: 'Number circle',
+     minify: htmlMinifyOptions,
+     chunks: ['numberCircle', 'vendor'],
+     template: './app/index.html',
+     filename: 'number-circle.html',
+   }),
// [...]
};
```

Quelltext 6.8: Hinzufügen eines Einstiegspunktes in Webpack

2. Im `app/pages` Verzeichnis wird ein neuer Ordner angelegt. Darin wird eine `index.js` Datei erzeugt, welche die Definition des Einstiegspunktes enthält

3. Der neu definierte Einstiegspunkt muss in der Webpack-Konfiguration registriert werden. Dafür muss die `webpack/common.config.babel.js` bearbeitet werden. In Quelltext 6.8 ist die nötige Änderung der Konfiguration am Beispiel der Zahlenkreis-Komponente zu sehen. Zum einen wird das `entry`-Objekt um den Pfad zu der neuen `index.js`-Datei im `app/pages`-Ordner erweitert. Zum anderen wird ein `HtmlWebpackPlugin`-Objekt zu dem `plugins`-Array hinzugefügt, um Webpack zu veranlassen, ein HTML-Dokument mit dem Namen `number-circle.html` für die Komponente bereitzustellen

Zu Beachten ist, dass nach der Änderung der Webpack-Konfiguration ein Neustart des gegebenenfalls schon laufenden Webpack-Entwicklungsservers erforderlich ist.

7 Fazit

Wie in der Einleitung beschrieben, wurden zu Beginn dieser Arbeit einige Ziele festgelegt. Diese haben während der Arbeit als Orientierung gedient, sowohl hinsichtlicht der Auswahl der Plattform als auch bezüglich zentraler Architekturentscheidungen. Abschließend soll im Folgenden nun das Ergebnis mit den anfänglich aufgestellten Anforderungen abgeglichen werden. Desweiteren soll ein Ausblick auf mögliche, zukünftige Erweiterungen und Verfeinerungen gegeben werden.

Offene Plattform

Ein zentrales Ziel dieser Arbeit bestand darin, eine Sammlung an Lernwerkzeugen zu gestalten, die von jeglichen Lizenzeinschränkungen befreit sind. Zudem sollte ein möglichst breites Spektrum an Geräten abgedeckt werden, um die Werkzeuge möglichst vielseitig zugänglich zu machen.

Die entwickelten Komponenten sind in den Browsern Firefox (ab Version 18), Safari (getestet in Version 9.1), Opera (getestet in Version 36.0), Edge (Version 25.0), Internet Explorer (Version 10 und 11) und in Google Chrome (getestet in Version 50) funktionsfähig. Für jede Komponente wurde die Benutzeroberfläche sowohl für die Bedienung per Maus und Tastatur, als auch per Touchscreen, implementiert. Die Benutzeroberflächen funktionieren auf unterschiedlichen Bildschirmgrößen. Umfangreiche Komponenten, wie der KV-Diagramm-Editor oder die Logic-Expression-Parser-Komponente, sind so gestaltet, dass der Benutzer Teile der Oberfläche verkleinern und vergrößern kann, um sie an seine Bildschirmgröße anzupassen. Somit ist es gelungen ein breites Spektrum an Geräten abzudecken.

Außeracht gelassen wurde bisher die Verwendung der Komponenten auf extrem kleinen Bildschirmen - z.B. auf einem Smartphone. Zwar ist es prinzipiell möglich, die Komponenten auf einem Smartphone zu verwenden, doch die Benutzeroberfläche der KV-Komponente beispielweise ist zu umfangreich, um sie sinnvoll auf einem kleinen Bildschirm abbilden zu können. Der Zahlenkreis hingegen lässt sich sogar angenehm auf einem Smartphone betrachten. Es ist fraglich, ob es einen Anwendungsfall gibt, in welchem ein komplexes KV-Diagramm auf einem Smartphone bearbeitet werden soll. Aus diesem Grund wurde die Konzeption einer für ein Smartphone sinnvoll reduzierten Benutzeroberfläche in dieser Arbeit ignoriert. Sollte ein solcher Anwendungsfall zukünftig auftreten, stellt dies jedoch kein technisches, sondern lediglich ein gestalterisches Pro-

Sprache	Dateien	Leerzeilen	Kommentarzeilen	Code-Zeilen
JavaScript	216	1513	1024	11970
Stylus	26	364	6	1773
JSON	20	11	0	988
HTML	1	0	0	11

Tabelle 7.1: Quelltext Statistik

blem dar. Per Stylesheets lässt sich die Oberfläche der Komponenten spezifisch für ausgewählte Bildschirmgrößen umgestalten.

Was die Lizenzfrage betrifft, wurde das Ziel ebenfalls erreicht. Es wurde auf die freien Webstandards HTML, CSS, JavaScript und SVG aufgesetzt, die kostenlos und quelloffen zur Verfügung stehen. Für die JavaScript-Entwicklung wurden, wie im Kapitel *Das Web als Plattform* und *Modulare Architektur* erläutert, ausgewählte Bibliotheken verwendet, um die Entwicklung der Komponenten zu unterstützen. Diese Bibliotheken stehen selbst quelloffen und kostenlos unter permissiven Lizenzen, wie der MIT-Lizenz oder der BSD-Lizenz, zur Verfügung.[MIT 1998][B 1988]

Eine Liste aller Lizenzen aller Abhängigkeiten befindet sich im Projekt-Ordner in der Datei `DEP_LICENSES.csv`. Inklusive aller transitiven Abhängigkeiten verwenden die entwickelten Komponenten 46 andere NPM-Packete, wovon die meisten unter der MIT-Lizenz veröffentlicht sind. Diese Lizenz stellt keine nennenwerte Einschränkung dar, sondern verlangt lediglich die Einbindung des Lizenztextes — und damit die Nennung der Bibliotheken — in die veröffentlichte Software. Hierbei ergibt sich die rein organisatorische Frage, ob es sinnvoll ist, den Lizenztext fast 50 mal — einmal für jede Bibliothek — in die von Webpack erzeugten JavaScript-Dateien einzubinden. Womöglich genügt die Nennung aller Bibliotheken zusammen mit dem Namen der zugehörigen Lizenz auf einer Übersichtsseite.

JavaScript

Im Kapitel 2 *Das Web als Plattform* wurden verschiedene Sprachen als Alternative zu JavaScript vorgestellt. Ihre Vor- und Nachteile wurden gegeneinander abgewogen. Schließlich wurde JavaScript für die Implementation der Komponenten gewählt. Im Nachhinein hat sich heraus gestellt, dass ein Typsystem vermutlich hilfreich gewesen wäre. Besonders für das Übergeben von Datensätzen einer Komponente an eine andere, wäre es während der Entwicklung praktisch gewesen, stärkere semantische Unterstützung von einem

Compiler zu erhalten. In Tabelle 7.1 ist die Anzahl der Zeilen an Quelltext abgebildet. Wie zu sehen ist, stellt der JavaScript-Quelltext den größten Anteil dar.

Sollten zukünftig weitere Komponenten entwickelt werden, ist es zu empfehlen, eine erneute Evaluation alternativer Sprachen durchzuführen und neue Komponenten möglicherweise in TypeScript zu entwickeln, um von optionaler statischer Typisierung Gebrauch machen zu können.

Perspektive

Die in dieser Arbeit umgesetzten Komponenten funktionieren technisch sehr gut. Inwiefern sie das Lernen in der Praxis unterstützen wird sich bald zeigen. Trotzdem gibt es natürlich noch Raum für einige Verbesserungen. Wie schon im Abschnitt der FSM-Komponente erklärt, wurde diese innerhalb dieser Arbeit nicht fertig gestellt.

Innerhalb dieser Arbeit wurden aus Zeitgründen, abgesehen von wenigen Ausnahmen, keine automatisierten Unit-Tests für die Komponenten geschrieben. Bevor die Komponenten weiterentwickelt werden, sollte dies nachgeholt werden, um ein gewisses Maß an Stabilität zu gewährleisten. Dank der funktionalen, zustandslosen Architektur der Komponenten, sollte es sich als besonders einfach gestalten, Unit-Tests zu definieren. Ein Grundgerüst für die Ausführung von Tests ist bereits eingerichtet und lässt sich über ein NPM-Script starten.

Die gebündelten und komprimierten JavaScript- und Stylesheet-Dateien der einzelnen Komponenten sind mit bis zu 500 Kilobyte recht groß. Das hat vermutlich den Grund, dass zusätzlich Bibliotheken, die nicht unbedingt verwendet werden, von Webpack mit in die Ausgabedateien übernommen werden, weil Webpack nicht in der Lage ist, selbstständig zu entscheiden, ob sie benötigt werden oder nicht. Mittels einer manuellen, präziseren Konfiguration von Webpack könnte hier noch nachgebessert werden, um die Dateigrößen zu reduzieren und damit die Ladezeiten und die benötigte Bandbreite zu reduzieren.

In einigen Anwendungsfällen stoßen die Komponenten an ihre Leistungsgrenze. Ein Beispiel hierfür ist die schon genannte Aktualisierung von HTML-Tabellen mit mehr als 1000 Zeilen. Ein weiteres Beispiel ist die Erzeugung von Schleifen in großen KV-Diagrammen auf leistungsschwachen Mobilgeräten. Der Flaschenhals ist in fast allen Fällen in der Aktualisierung des DOM-Baumes zu vermuten. Um die Leistungsfähigkeit der Komponenten zu verbessern, könnte zum einen auf eine alternative Virtual-DOM Implementierung gewechselt werden und zum anderen die DOM-Struktur der Komponenten so überar-

beitet und reduziert werden, dass dem Browser aufwändige Layoutberechnungen abgenommen werden. Diese Problematik ist im Einzelfall genauer zu untersuchen.

Diese Arbeit hat den Titel *Ein JavaScript Framework für interaktive Demos und Animationen zur technischen Informatik*. In den bisher implementierten Komponenten sind jedoch keine Animationen enthalten, da es sich für die jeweiligen Anwendungsfälle nicht angeboten hat. Für die Zukunft ist es interessant, auch Komponenten mit animierten Darstellungen umzusetzen. Sowohl Stylesheets als auch der `Observable`-Datentyp bieten hierfür eine solide Grundlage.

Da der Browser auch eine Reihe von Schnittstellen für die Netzwerkkommunikation — z.B. per HTTP oder Websockets — bereitstellt, ergibt sich auch die Perspektive, Komponenten anzubieten, die ein kollaboratives Arbeiten von mehreren Geräten aus erlauben. Ein einfaches Beispiel hierfür wäre ein Komponente, die es erlaubt, in einer Vorlesung eine Frage oder Aufgabe an das Publikum zu stellen und Antworten von den Mobilgeräten der Zuschauer entgegen zu nehmen. In der RxJS/Cycle-Architektur ließe sich so ein Bedienkonzept über einen weiteren Treiber realisieren, welcher der Komponente neben den DOM-Ereignissen auch eingehende Netzwerkanfragen als Datenstrom zur Verfügung stellt.

Alles in allem wurden die Komponenten in dieser Arbeit erfolgreich umgesetzt und bieten eine gute Grundlage für weitere interessante Entwicklungen.

Literaturverzeichnis

[ASHKENAS 2016]
ASHKENAS, Jeremy: *CoffeeScript*. http://coffeescript.org/. Version: 2016. – [Online; 6. Mai 2016]

[B 1988]
B, University of C.: *The BSD 3-Clause License*. https://opensource.org/licenses/BSD-3-Clause. Version: 1988. – [Online; 6. Mai 2016]

[B 1992]
B, University of C.: *b Logic Interchange Format*. https://www.ece.cmu.edu/~ee760/760docs/blif.pdf. Version: 1992. – [Online; 6. Mai 2016]

[BAK et al. 2011]
BAK, Lars; BRACHA, Gilad; LUND, Kasper V.: *Dart*. https://www.dartlang.org/. Version: 2011. – [Online; 6. Mai 2016]

[BAK und LUND 2015]
BAK, Lars; LUND, Kasper: *Dart FAQ*. http://news.dartlang.org/2015/03/dart-for-entire-web.html. Version: 2015. – [Online; 6. Mai 2016]

[BASU 2013]
BASU, Sayanee: *Source Maps 101*. http://code.tutsplus.com/tutorials/source-maps-101--net-29173. Version: 2013. – [Online; 6. Mai 2016]

[BRACHA 2015]
BRACHA, Gilad: *The Dart Programming Language*. Addison-Wesley Professional, 2015. – ISBN 0–321–92770–2

[BRENDAN EICH 2016]
BRENDAN EICH, Ecma I.: *ECMAScript 262*. http://www.ecma-international.org/publications/standards/Ecma-262.htm. Version: 2016. – [Online; 6. Mai 2016]

[BUCHHEIM et al. 2002]
BUCHHEIM, Christoph; JÜNGER, Michael; LEIPERT, Sebastian: Improving Walker's Algorithm to Run in Linear Time. In: *Revised Papers from the 10th International Symposium on Graph Drawing*. London, UK, UK : Springer-Verlag, 2002 (GD '02). – ISBN 3–540–00158–1, 344–353

[BUCKLER 2011]
BUCKLER, Craig: *How to Draw Bezier Curves on an HTML5 Canvas*. http://www.sitepoint.com/html5-canvas-draw-bezier-curves/. Version: 2011. – [Online; 6. Mai 2016]

[BURGESS und FREEMAN 2016]
BURGESS, Gary; FREEMAN, Phil: *PureScript*. http://www.purescript.org/. Version: 2016. – [Online; 6. Mai 2016]

[BURGET 2014]
BURGET, Joel: *Jison Webpack Loader*. https://github.com/joelburget/jison-loader. Version: 2014. – [Online; 6. Mai 2016]

[CARLETON 2013]
CARLETON, Brad: *Coffeescript vs. Javascript: Dog eat Dog*. https://lostechies.com/bradcarleton/2013/10/23/coffeescript-vs-javascript-dog-eat-dog/. Version: 2013. – [Online; 6. Mai 2016]

[CARTER 2009]
CARTER, Zach: *Jison*. http://zaach.github.com/jison/. Version: 2009. – [Online; 6. Mai 2016]

[CHEDEAU 2013]
CHEDEAU, Christopher: *React's diff algorithm*. http://calendar.perfplanet.com/2013/diff/. Version: 2013. – [Online; 6. Mai 2016]

[CHROMIUM 2015]
CHROMIUM: *Vendor Prefixes in Blink*. http://www.chromium.org/blink#vendor-prefixes. Version: 2015. – [Online; 6. Mai 2016]

[CLEVE MOLER 1984]
CLEVE MOLER, MathWorks: *MatLab*. http://de.mathworks.com/products/matlab/. Version: 1984. – [Online; 6. Mai 2016]

[COYIER 2009]
COYIER, Chris: *The CSS Box Model*. https://css-tricks.com/the-css-box-model/. Version: 2009. – [Online; 6. Mai 2016]

[DAHL 2009]
DAHL, Ryan: *Node.js*. https://nodejs.org/en/about/. Version: 2009. – [Online; 6. Mai 2016]

[DeBill 2010]
DeBill, Erik: *Package Manager Module Count*. http://www.modulecounts.com/. Version: 2010. – [Online; 6. Mai 2016]

[Dijkstra 1961]
Dijkstra, Dr. E.: *Algol-60 Translation*. http://www.cs.utexas.edu/~EWD/MCReps/MR35.PDF. Version: 1961. – [Online; 6. Mai 2016]

[Erik Dahlström et al. 2001]
Erik Dahlström, Opera S.; Patrick Dengler, Microsoft C.; Anthony Grasso, Canon I.; Chris Lilley, W3C; Cameron McCormack, Mozilla C.; Doug Schepers, W3C; Jonathan Watt, Mozilla C.: *Scalable Vector Graphics*. https://www.w3.org/TR/SVG/. Version: 2001. – [Online; 6. Mai 2016]

[Esch 2013a]
Esch, Matt: *A DSL for creating virtual trees*. https://github.com/Matt-Esch/virtual-dom/tree/master/virtual-hyperscript. Version: 2013. – [Online; 6. Mai 2016]

[Esch 2013b]
Esch, Matt: *virtua-dom Bibliothek*. https://github.com/Matt-Esch/virtual-dom. Version: 2013. – [Online; 6. Mai 2016]

[Fiser 2011]
Fiser, Marek: *L-systems in Haskell*. http://www.marekfiser.com/Projects/Lsystems-in-Haskell. Version: 2011. – [Online; 6. Mai 2016]

[Florian Scholz 2015]
Florian Scholz, Mozilla: *JavaScript Datentypen und Datenstrukturen (Mozilla Developer Network)*. https://developer.mozilla.org/de/docs/Web/JavaScript/Datenstrukturen. Version: 2015. – [Online; 6. Mai 2016]

[Florian Scholz and Joe Medley 2005]
Florian Scholz and Joe Medley: *Array.prototype.findIndex*. https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Array/findIndex. Version: 2005. – [Online; 6. Mai 2016]

[Ford 2004]
Ford, Bryan: Parsing Expression Grammars: A Recognition-based Syntactic Foundation. In: *SIGPLAN Not.* 39 (2004), Januar, Nr. 1, 111–122. http://dx.doi.org/10.1145/982962.964011. – DOI 10.1145/982962.964011. – ISSN 0362–1340

[FOWLER 2006]
FOWLER, Martin: *GUI Architectures*. http://martinfowler.com/eaaDev/uiArchs.html. Version: 2006. – [Online; 6. Mai 2016]

[FREEMAN 2014]
FREEMAN, Phil: *PureScript by Example*. https://leanpub.com/purescript/read. Version: 2014. – [Online; 6. Mai 2016]

[GAMMA et al. 1995]
GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John: *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1995. – ISBN 0–201–63361–2

[GROSS und KHOMERIKI 2014a]
GROSS, David; KHOMERIKI, Georgi: *Reactive Extension Operators*. http://reactivex.io/documentation/operators.html. Version: 2014. – [Online; 6. Mai 2016]

[GROSS und KHOMERIKI 2014b]
GROSS, David; KHOMERIKI, Georgi: *ReactiveX.io*. http://reactivex.io/intro.html. Version: 2014. – [Online; 6. Mai 2016]

[GUIDELINES 2014]
GUIDELINES, Google D.: *Google Developer PageSpeed Insights*. https://developers.google.com/speed/docs/insights/SizeTapTargetsAppropriately. Version: 2014. – [Online; 6. Mai 2016]

[GUIDELINES 2016]
GUIDELINES, Yahoo D.: *Best Practices for Speeding Up Your Web Site*. https://developer.yahoo.com/performance/rules.html. Version: 2016. – [Online; 6. Mai 2016]

[HALLIDAY 2012]
HALLIDAY, James: *Browserify*. http://browserify.org/. Version: 2012. – [Online; 6. Mai 2016]

[HAMILTON 2008]
HAMILTON, Naomi: *The A-Z of Programming Languages: JavaScript*. http://www.computerworld.com.au/article/255293/a-z_programming_languages_javascript/. Version: 2008. – [Online; 6. Mai 2016]

[HOLLDACK 2011]
HOLLDACK, Mario: *Bézierkurven*. http://www.math.uni-frankfurt.de/~numerik/lehre/Vorlesungen/Pros-11/Ausarbeitungen/holl.pdf. Version: 2011. – [Online; 6. Mai 2016]

[HOLOWAYCHUK 2010]
HOLOWAYCHUK, TJ: *Stylus*. http://stylus-lang.com/. Version: 2010. – [Online; 6. Mai 2016]

[HOWE 2015]
HOWE, Shay: *Opening the Box Model*. http://learn.shayhowe.com/html-css/opening-the-box-model/. Version: 2015. – [Online; 6. Mai 2016]

[HUND 2013]
HUND, David: *What CSS to prefix?* http://shouldiprefix.com/#flexbox. Version: 2013. – [Online; 6. Mai 2016]

[HUNNER und XU 2011]
HUNNER, Trey; XU, Hong: *EditorConfig*. http://editorconfig.org/. Version: 2011. – [Online; 6. Mai 2016]

[IMMS 2014]
IMMS, Daniel: *CSS preprocessors are here to stay*. http://www.growingwiththeweb.com/2014/03/css-preprocessors-are-here-to-stay.html. Version: 2014. – [Online; 6. Mai 2016]

[JACKSON und MARRIN 2011]
JACKSON, Dean; MARRIN, Chris: *WebGL Specification*. https://www.khronos.org/registry/webgl/specs/1.0/. Version: 2011. – [Online; 6. Mai 2016]

[JOHN LENZ und NICK FITZGERALD 2011]
JOHN LENZ, Google; NICK FITZGERALD, Mozilla: *Source Map Revision 3 Proposal*. https://docs.google.com/document/d/1U1RGAehQwRypUTovF1KRlpiOFze0b-_2gc6fAH0KY0k/edit?pref=2&pli=1#heading=h.1ce2c87bpj24. Version: 2011. – [Online; 6. Mai 2016]

[KAHN 1962]
KAHN, A. B.: Topological Sorting of Large Networks. In: *Commun. ACM* 5 (1962), November, Nr. 11, 558–562. http://dx.doi.org/10.1145/368996.369025. – DOI 10.1145/368996.369025. – ISSN 0001–0782

[KOPPERS 2012a]
KOPPERS, Tobias: *Webpack*. https://webpack.github.io/. Version: 2012. – [Online; 6. Mai 2016]

[KOPPERS 2012b]
KOPPERS, Tobias: *Webpack Style Loader Repository*. https://github.com/webpack/style-loader. Version: 2012. – [Online; 6. Mai 2016]

[KRÖNERT und DALLMANN 2000]
KRÖNERT, Karola; DALLMANN, Ulrich: *FSM Applet*. https://tams.informatik.uni-hamburg.de/applets/java-fsm/. Version: 2000. – [Online; 6. Mai 2016]

[LECHELT 2015]
LECHELT, Kahlil: *A Cycle.js Driver for using localStorage and sessionStorage*. https://github.com/cyclejs/cycle-storage-driver. Version: 2015. – [Online; 6. Mai 2016]

[LEE BYRON 2014]
LEE BYRON, Facebook: *ImmutableJS*. https://facebook.github.io/immutable-js/docs/. Version: 2014. – [Online; 6. Mai 2016]

[LESH et al. 2015]
LESH, Ben; KWON, OJ; STALTZ, André: *RxJS Introduction*. http://reactivex.io/rxjs/manual/overview.html. Version: 2015. – [Online; 6. Mai 2016]

[MACCAW 2012]
MACCAW, Alex: *The Little Book on CoffeeScript*. O'Reilly Media, 2012. – ISBN 1–4493–2105–4

[MAHARRY 2013]
MAHARRY, Dan: *TypeScript Revealed*. 1st. Berkely, CA, USA : Apress, 2013. – ISBN 1430257253, 9781430257257

[MAJDA 2010]
MAJDA, David: *PEG.js*. http://pegjs.majda.cz/. Version: 2010. – [Online; 6. Mai 2016]

[MANSILLA 2015]
MANSILLA, Sergi: *Reactive Programming with RxJS*. The Pragmatic Programmers, 2015. – ISBN 978–1–68050–129–2

[MCHEDLIDZE et al. 2012]
MCHEDLIDZE, Tamara; NOLLENBURG, Martin; RUTTER, Ignaz: *Algorithmen zur Visualisierung von Graphen*. https://i11www.iti.uni-karlsruhe.de/_media/teaching/winter2012/graphdrawing/divide_and_conquer.pdf. Version: 2012. – [Online; 6. Mai 2016]

[MCKENZIE 2014a]
MCKENZIE, Sebastian: *BabelJS*. http://babeljs.io/. Version: 2014. – [Online; 6. Mai 2016]

[MCKENZIE 2014b]
MCKENZIE, Sebastian: *BabelJS Github Repository*. https://github.com/babel/babel. Version: 2014. – [Online; 6. Mai 2016]

[MEALY 1955]
MEALY, George H.: A Method for Synthesizing Sequential Circuits. In: *Bell System Technical Journal* 34 (1955), Nr. 5, S. 1045–1079

[MEIJER 2012]
MEIJER, Erik: *Your Mouse is a Database*. http://queue.acm.org/detail.cfm?id=2169076. Version: 2012. – [Online; 6. Mai 2016]

[MEYER 2011]
MEYER, Jan M.: *JS/CC*. http://jscc.phorward-software.com/. Version: 2011. – [Online; 6. Mai 2016]

[MEYER 1998]
MEYER, Matthias: *Karnaugh-Veitch Diagram Applet*. https://tams.informatik.uni-hamburg.de/applets/kvd/. Version: 1998. – [Online; 6. Mai 2016]

[MILL 2008a]
MILL, Bill: *Drawing Presentable Trees*. http://billmill.org/pymag-trees/. Version: 2008. – [Online; 6. Mai 2016]

[MILL 2008b]
MILL, Bill: *Drawing Presentable Trees Implementation*. https://github.com/llimllib/pymag-trees/. Version: 2008. – [Online; 6. Mai 2016]

[MILLS 2012]
MILLS, Chris: *A Short History of JavaScript*. https://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript. Version: 2012. – [Online; 6. Mai 2016]

[MINGUN 2014]
MINGUN: *Peg.JS Import feature*. https://github.com/pegjs/pegjs/pull/308. Version: 2014. – [Online; 6. Mai 2016]

[MIT 1998]
MIT, Massachusetts Institute of T.: *MIT License*. https://opensource.org/licenses/MIT. Version: 1998. – [Online; 6. Mai 2016]

[MOORE 1956]
MOORE, Edward F.: Gedanken-Experiments on Sequential Machines. In: SHANNON, Claude (Hrsg.); MCCARTHY, John (Hrsg.): *Automata Studies*. Princeton, NJ : Princeton University Press, 1956, S. 129–153

[MOSLEY 2010]
MOSLEY, Marie: *Box Sizing*. https://css-tricks.com/box-sizing/. Version: 2010. – [Online; 6. Mai 2016]

[OYSTER 2014]
OYSTER, Fred: *File:Radial tree - Graphic Statistics in Management*. WikiMedia. https://commons.wikimedia.org/wiki/File:Radial_tree_-_Graphic_Statistics_in_Management.svg. Version: 2014. – [Online; 6. Mai 2016]

[PARR 2014]
PARR, Terence: *ANTLR*. http://www.antlr.org/. Version: 2014. – [Online; 6. Mai 2016]

[PARR und FISHER 2011]
PARR, Terence; FISHER, Kathleen: LL(*): The Foundation of the ANTLR Parser Generator. In: *SIGPLAN Not.* 46 (2011), Juni, Nr. 6, 425–436. http://dx.doi.org/10.1145/1993316.1993548. – DOI 10.1145/1993316.1993548. – ISSN 0362–1340

[PLAG 2013]
PLAG, Florian: *Retina-Display: HTML-Canvas optimieren*. http://html5-mobile.de/blog/retina-display-html-canvas-optimieren. Version: 2013. – [Online; 6. Mai 2016]

[PODWYSOCKI 2015]
PODWYSOCKI, Matthew: *RxJS Introduction*. https://github.com/Reactive-Extensions/RxJS/blob/master/src/core/linq/observable/fromevent.js#L83-L104. Version: 2015. – [Online; 6. Mai 2016]

[PRESTON-WERNER 2011]
PRESTON-WERNER, Tom: *Semantic Versioning Specification*. http://semver.org/. Version: 2011. – [Online; 6. Mai 2016]

[PUSHKAREV 2016]
PUSHKAREV, Denis: *CoreJS Github Repo*. https://github.com/zloirock/core-js. Version: 2016. – [Online; 6. Mai 2016]

[REINGOLD und TILFORD 1981]
REINGOLD, Edward M.; TILFORD, John S.: *Tidier Drawings of Trees*. http://reingold.co/tidier-drawings.pdf. Version: 1981. – [Online; 6. Mai 2016]

[SCHLUETER 2016]
SCHLUETER, Isaac Z.: *Node Package Manager*. https://docs.npmjs.com/getting-started/what-is-npm. Version: 2016. – [Online; 6. Mai 2016]

[SCHÄFER 2009]
SCHÄFER, Mathias: *JavaScript: Fortgeschrittene Ereignisverarbeitung*. http://molily.de/js/event-handling-fortgeschritten.html. Version: 2009. – [Online; 6. Mai 2016]

[SCOTXW 2014]
SCOTXW: *Linux kernel Input Output*. https://commons.wikimedia.org/wiki/File:Linux_kernel_INPUT_OUPUT_evdev_gem_USB_framebuffer.svg. Version: 2014. – [Online; 6. Mai 2016]

[SEDDON 2012]
SEDDON, Ryan: *Introduction to JavaScript Source Maps*. http://www.html5rocks.com/en/tutorials/developertools/sourcemaps/. Version: 2012. – [Online; 6. Mai 2016]

[SHARP 2010]
SHARP, Remy: *What is a Polyfill*. https://remysharp.com/2010/10/08/what-is-a-polyfill. Version: 2010. – [Online; 6. Mai 2016]

[SIMPSON 2014]
SIMPSON, Kyle: *You dont́ know JS: Coercing*. https://github.com/getify/You-Dont-Know-JS/blob/master/types%20%26%20grammar/ch4.md. Version: 2014. – [Online; 6. Mai 2016]

[SITNIK 2013]
SITNIK, Andrey: *PostCSS*. http://postcss.org/. Version: 2013. – [Online; 6. Mai 2016]

[SITNIK 2014]
SITNIK, Andrey: *Webpack PostCSS Loader Repository*. https://github.com/postcss/postcss-loader. Version: 2014. – [Online; 6. Mai 2016]

[STALTZ 2015a]
STALTZ, André: *A Cycle.js Driver for making HTTP requests*. https://github.com/cyclejs/http. Version: 2015. – [Online; 6. Mai 2016]

[STALTZ 2015b]
STALTZ, André: *MODEL-VIEW-INTENT*. http://cycle.js.org/model-view-intent.html. Version: 2015. – [Online; 6. Mai 2016]

[STALTZ 2015c]
STALTZ, André: *RxMarbles*. http://rxmarbles.com/. Version: 2015. – [Online; 6. Mai 2016]

[STALTZ 2015d]
STALTZ, André: *The standard DOM Driver for Cycle.js based on virtual-dom, and other helpers*. https://github.com/cyclejs/dom. Version: 2015. – [Online; 6. Mai 2016]

[STALTZ 2016]
STALTZ, André: *The introduction to Reactive Programming you've been missing*. https://gist.github.com/staltz/868e7e9bc2a7b8c1f74. Version: 2016. – [Online; 6. Mai 2016]

[SUBBOTIN 2014]
SUBBOTIN, Andrey: *Peg.JS Webpack Loader*. https://github.com/eploko/pegjs-loader. Version: 2014. – [Online; 6. Mai 2016]

[TARR 2015]
TARR, Dominic: *Create HyperText with JavaScript*. https://github.com/dominictarr/hyperscript. Version: 2015. – [Online; 6. Mai 2016]

[VEITCH 1952]
VEITCH, E. W.: A Chart Method for Simplifying Truth Functions. In: *Proceedings of the 1952 ACM National Meeting (Pittsburgh)*. New York, NY, USA : ACM, 1952 (ACM '52), 127–133

[WALDRON et al. 2016]
WALDRON, Rick; POTTER, Caitlin; SHEROV, Mike; PENNISI, Mike; PAGE, Luke: *JSHint*. http://jshint.com/. Version: 2016. – [Online; 6. Mai 2016]

[WALKER 1990]
WALKER, J. Q. II: A Node-positioning Algorithm for General Trees. In: *Softw. Pract. Exper.* 20 (1990), Juli, Nr. 7, 685–705. http://dx.doi.org/10.1002/spe.4380200705. – DOI 10.1002/spe.4380200705. – ISSN 0038–0644

[WEBBER 2014]
WEBBER, Kevin: *What is Reactive Programming?* https://medium.com/reactive-programming/what-is-reactive-programming-bc9fa7f4a7fc. Version: 2014. – [Online; 6. Mai 2016]

[WILLIAMS 2015]
WILLIAMS, Ashley: *NPM: Using a package.json*. https://docs.npmjs.com/getting-started/using-a-package.json. Version: 2015. – [Online; 6. Mai 2016]

[WOO 2015]
WOO, Justin: *How to use Cycle.js to create a scroll-table*. http://qiita.com/kimagure/items/d29ed7b7bdaaf6977b9a. Version: 2015. – [Online; 6. Mai 2016]

[ZACHARY 2012]
ZACHARY: *Butter Performance*. http://developer.streak.com/2012/07/butter-performance.html. Version: 2012. – [Online; 6. Mai 2016]

[ZAKAS 2016]
ZAKAS, Nicholas C.: *ESLint*. http://eslint.org/. Version: 2016. – [Online; 6. Mai 2016]

Eidesstattliche Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.

Hamburg, den _____ Unterschrift: _____