HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

DIPLOMARBEIT

Herr
Daniel Raabe

Konzeption und Implementierung eines grafischen, webbasierten XML-Editors zur Erstellung und Bearbeitung von XML-Dateien

2014

Fakultät **Elektro- und Informationstechnik**

DIPLOMARBEIT

Konzeption und Implementierung eines grafischen, webbasierten XML-Editors zur Erstellung und Bearbeitung von XML-Dateien

Autor:
Daniel Raabe

Studiengang:
Multimediatechnik

Seminargruppe:
MK08s1-D

Erstprüfer:
Prof. Dr. Frank Zimmer

Zweitprüfer:
Dipl.-Ing. Mario Rieß

Mittweida, Februar 2014

Bibliografische Angaben

Raabe, Daniel: Konzeption und Implementierung eines grafischen, webbasierten XML-Editors zur Erstellung und Bearbeitung von XML-Dateien , 73 Seiten, 44 Abbildungen, Hochschule Mittweida, University of Applied Sciences, Fakultät Elektro- und Informationstechnik

Diplomarbeit, 2014

Referat

Diese Diplomarbeit widmet sich der Entwicklung eines grafischen Online-XML-Editors. Im ersten Abschnitt wird die Firma vorgestellt und die Idee hinter der Themenwahl näher betrachtet. Es werden ausschlaggebende Webtechnologien behandelt und deren Wahl begründet.
Der zweite Teil konzentriert sich ausschließlich auf die praktische Realisierung. Detailliert werden die Funktionen des Editors beschrieben und die programmtechnische Umsetzung erklärt.
Im anschließenden Kapitel setzt sich die Arbeit mit dem Testen der Software auseinander, gefolgt vom letzten Abschnitt, der eine Zusammenfassung liefert. Dabei werden das Ergebnis und Herausforderungen während der Erstellung beschrieben. Zudem gibt das Kapitel Einblicke in mögliche zukünftige Versionen des Prototyps.

I. Inhaltsverzeichnis

II. Abbildungsverzeichnis

III. Tabellenverzeichnis

IV. Abkürzungsverzeichnis

AJAX ..... Asynchronous JavaScript And XML, Seite 19
BLOB ..... Binary Large Object, Seite 51
CSS ..... Cascading Style Sheet, Seite V
DHTML ..... Dynamic Hypertext Markup Language, Seite 21
DOM ..... Document Object Model, Seite 10
GUI ..... Graphical User Interface, Seite 1
HTML ..... Hypertext Markup Language, Seite 1
JS ..... JavaScript, Seite II
JS ..... JavaScript, Seite 20
JSON ..... JavaScript Object Notation, Seite 1
MBO ..... Merchant Back Office, Seite 5
MIFF ..... Magick Image File Format, Seite 8
PHP ..... Hypertext Preprocessor, Seite 37
SFTP ..... Secure File Transfer Protocol, Seite 36
SVG ..... Scaleable Vector Graphics, Seite 15
W3C ..... World Web Consortium, Seite 15
WHATWG ..... Web Hypertext Application Working Group, Seite 15
XHTML ..... Extensible Hypertext Markup Language, Seite 15
XML ..... Extensible Markup Language, Seite 5

V. Vorwort

Fast jeder ist im Besitz eines Smartphones, Tablets oder Ultrabooks. Oft nennt man sogar mehrere Geräte sein Eigen. Für jede Situation ist etwas dabei. Zum Kommunizieren für unterwegs nehmen wir das Smartphone, zum Lesen auf dem Sofa greifen wir zum Tablet und wenn die Arbeit ruft ist das Ultrabook oder ein Laptop am besten geeignet. Sobald wir das Gerät unserer Wahl einschalten, begrüßt uns ein hochauflösender, kontrastreicher und farbenfroher Bildschirm. Dank leistungsstarker Hardware und ausgestattet mit komplexer Software rufen wir ohne Verzögerung den Browser auf, blättern durch die Bildergalerie, rufen die Mails ab oder hören unsere Lieblingsmusik. Möglich wird all dies durch einfaches Anklicken von Buttons mit der Maus oder dem Touchpad oder einem berührungsempfindlichen Bildschirm, der uns den direkten Kontakt mit der Benutzeroberfläche bietet.
Aber was passiert, wenn wir unsere Mails über komplizierte Befehle absenden und empfangen müssten, anstatt einfach auf einen Button zu klicken. Oder eine komplexe Befehlskette eintippen sollen um in unserer Musikbibliothek navigieren zu können und einen Titel abzuspielen oder ihn auf der Speicherkarte zu verschieben.
Vermutlich legen wir das Gerät schnell wieder aus der Hand, weil uns die Bedienung frustrieren und zu viel abverlangen würde. Die Lösung des Problems sind grafische Benutzeroberflächen, ohne die Ultrabooks, Smartphones oder Tablets nicht einsetzbar wären, da nur ein IT-Profi sie bedienen könnte.
Dank moderner Websprachen wie CSS oder JQuery lassen sich visuell anspruchsvolle Bedienoberflächen auch ins Web verlagern und mit einem Browser über das Internet aufrufen, um Anwendungen geräteunabhängig nutzen zu können. Genau hier liegt der Anreiz für die Bearbeitung dieses Themas. Das Herunterladen und Installieren von großen Softwarepaketen auf den eigenen Rechner oder spezielle Hardwareanforderungen rücken in den Hintergrund. Webbasierte Programme können weltweit, unter Voraussetzung eines Internetzugangs, genutzt werden. Auch die Entwicklung von Webapplikationen erfordert keine speziellen Entwicklungsumgebungen. Ein Texteditor und ein Browser sind ausreichend um zumindest den Einstieg in die Webprogrammierung zu erlernen.
Das Studium bot einen guten Einstieg, jedoch wurden Projekte meistens gemeinschaftlich gelöst, sodass die Abschlussarbeit die beste Gelegenheit bietet eine praxisnahe Idee zu realisieren und die eigenen Programmierkenntnisse zu festigen.

1 Einleitung

1.1 Motivation

Vor dem Aufkommen visueller Bedienoberflächen wurden Programme vom Anwender mit Befehlen über eine Konsole gesteuert. Peinlich genau musste auf jedes Zeichen eines Befehls geachtet werden. Grafische Benutzeroberflächen, im Englischen *Graphical User Interface*(GUI), übernehmen diese Aufgabe.
Sie bieten für sich genommen keine Funktionalität, sondern übermitteln lediglich Befehle an das Programm. Mitunter können diese Befehlsketten sehr lang und unübersichtlich werden. Eine GUI bietet den Vorteil, dass sie vom Nutzer keinerlei Kenntnisse über Programmbefehle verlangt.
Ein anderes, für diese Arbeit relevantes Szenario, ist die Speicherung von Informationen in einer bestimmten Syntax[1], meist zum Datenaustausch mit anderen Anwendungen. Grafische Benutzeroberflächen können im Vergleich zu einer spezifischen Syntax frei gestaltet werden und speichern trotzdem alle Informationen im gewünschten Format. Somit muss der Anwender sich nicht mit Notationen wie XML oder JSON beschäftigen. Er kann sich ausschließlich dem Inhalt widmen.

Die weltweit stetig steigende Datenrate bei Internetanschlüssen [Uni13] führt dazu, dass ganze Applikationen oder webbasierte Ableger mit ihrer GUI ins Web verlagert werden. Als praxisnahes Beispiel für eine Weboberfläche sei Dropbox, ein Dienstanbieter von Online-Speicherplatz, angeführt. Er bietet jedem Nutzer an, seine Dateien online zu speichern. Über die Oberfläche, die einem Dateiexplorer ähnelt, können die Dateien auch für Laien einfach verwaltet werden. Webbasierte Dienste und dafür geschaffene Benutzeroberflächen können orts- und plattformunabhängig genutzt werden. In den meisten Fällen sind ein Browser und ein Internetzugang ausreichend.

Das Ziel der Diplomarbeit ist die Umsetzung eines webbasierten grafischen Editors zur Erstellung und Bearbeitung von XML-Dateien, insbesondere der im ePages-System verwendeten transform.xml[2]. Alle Grundfunktionen, die ein einfacher Texteditor bietet, sollen nach Möglichkeit implementiert werden. Dazu gehören das Einlesen und Modifizieren vorhandender, sowie das Erstellen und Ausgeben neuer XML-Dateien. Als Ergebnis wird eine Basisversion angestrebt, die in nachfolgenden Versionen den Aufbauprozess einer transform.xml optimieren soll.
Als Programmiersprachen werden vorrangig HTML5 und JavaScript, bzw. das JavaScript-Framework JQuery, eingesetzt. Für die grafische Umsetzung empfiehlt sich die Suche nach einer geeigneten JavaScript-Bibliothek, die einen großen Funktionsumfang für den Umgang mit Grafiken bietet.

[1] Regeln für die Bildung wohlgeformter Ausdrücke in einer Programmiersprache
[2] spezielle XML-Datei für die Anpassung des Layouts eines ePages-Shops

1.2 Kapitelüberblick

Die Arbeit gliedert sich in fünf Hauptkapitel, die **Grundlagen**, das **Formulieren von Lösungsansätzen**, die **Implementierung**, der **Softwaretest** und die **Zusammenfassung**.
Das Kapitel Grundlagen gibt einen allgemeinen Überblick über die Firma ePages und eine grobe Erklärung der Funktionsweise ihrer Software. Besonders im Fokus steht die transform.xml, eine XML-Datei, die maßgeblich an der Bearbeitung von ePages Onlineshop-Styles beteiligt ist. An ihrer Struktur orientieren sich die beschriebenen Anforderungen an die erste Programmversion des Editors.

Das Kapitel Formulieren von Lösungsansätzen wird sich mit den eingesetzten Webtechnologien, unter anderem HTML5 und JavaScript, beschäftigen und ausgewählte JavaScript-Frameworks analysieren.

Das dritte Kapitel Implementierung geht detailliert darauf ein wie die Anforderungen des Prototyps in der Praxis realisiert wurden. Dazu bot sich eine Unterteilung in die einzelnen Schritte für die Darstellung und die Funktionsweise des Programms. Unter anderem wird auf die Projektorganisation und das ausgewählte Design eingegangen.

Das folgende Kapitel Softwaretest beschäftigt sich mit der Vorgehensweise beim Testen des Programms. Es werden zwei verschiedene Testmodelle beschrieben und ihre Anwendung auf dieses Projekt erklärt.

Abschließend wird im Kapitel Zusammenfassung das Ergebnis betrachtet und darauf eingegangen, welche Herausforderungen das Projekt begleiteten. Zusätzlich werden Ideen formuliert, die in einer zukünftigen Version implementiert werden könnten.

2 Grundlagen

2.1 ePages

Alles begann im Jahr 1983, als Wilfried Beeck in Kiel die “Beeck & Drahms Software GbR“ gründete. 1987 fand eine Umfirmierung statt, aus der “Beeck & Drahms Software GbR“ wurde die “d’Art Computer GmbH“. Nach vier weiteren Jahren zog die gesamte Firma nach Hamburg. 1992 wurde die Schwestergesellschaft “NetConsult Communications GmbH“ ins Leben gerufen. Diese wurde später in die noch heute existierende “Intershop Communications GmbH“ umbenannt.
Die erste einsetzbare Onlineshop-Software wurde 1995 vorgestellt. Mit der “Intershop epages“ entstand zwei Jahre später die erste E-Commerce Hosting Plattform weltweit. 2002 verabschiedete Wilfried Beeck sich aus dem Vorstand der Intershop und übernahm mit seiner Firma “d’Art“ die Intershop-4-Produktlinie. Wenig später wurde die Firma in “ePages GmbH“ umbenannt. [ePa13]
Das Geschäftsmodell von ePages sieht den direkten Verkauf der Software an Shopbetreiber nicht vor. Stattdessen wird die Shopsoftware an sogenannte Internet Service Provider vermietet. Diese wiederum können bestimmte Shoptypen dem Endkunden, einen Onlineshopbetreibenden, zur Verfügung stellen.

Zur Zielgruppe gehören vor allem Anwender, die eine einfache Möglichkeit suchen ihre Waren online zu verkaufen. Der Vorteil für den Händler liegt darin keine teure Hardware anschaffen zu müssen oder aufwendige Installationsroutinen zu absolvieren. Neben einem Internetzugang wird lediglich ein Browser benötigt.

Die zwei Hauptstandorte von ePages befinden sich in Hamburg und Jena. 2006 wurden zusätzlich Niederlassungen in London und Barcelona eröffnet, später auch in Boston. Bereits 2010 wird ePages von über 50.000 Kunden weltweit eingesetzt. Mittlerweile betreibt ePages weit über 80.000 Shops in 75 Ländern und ist damit der größte Anbieter von E-Commerce Software.
Zum Erfolg von ePages trägt auch die weltweite Kooperation von mehr als 60 Technologiepartnern bei, darunter Preisvergleichsportale und Anbieter von Zahlungsmethoden. Desweiteren unterstützt die Software, die bereits in Version 6 vorliegt, 15 Sprachen mit den dazugehörigen Währungen. Zu den wichtigsten Vertriebspartnern gehören hierzulande die Deutsche Telekom und Strato. [ePa13] Dabei bleibt es dem Vertriebspartner überlassen, wie sein Shop gestaltet sein soll. ePages bietet dafür individuelle Serviceleistungen um spezielle Anpassungen zügig umzusetzen. Wie eingangs erwähnt wird vom Shopbetreiber nur ein Browser benötigt, um den Internetauftritt für seine Kunden zu gestalten und zu betreiben. Dabei laufen alle Prozesse auf den Servern des Providers, der Browser dient nur als Benutzeroberfläche. Vereinfacht ausgedrückt gliedert sich die Softwarestruktur in das Backoffice und die für den Käufer sichtbare Storefront.

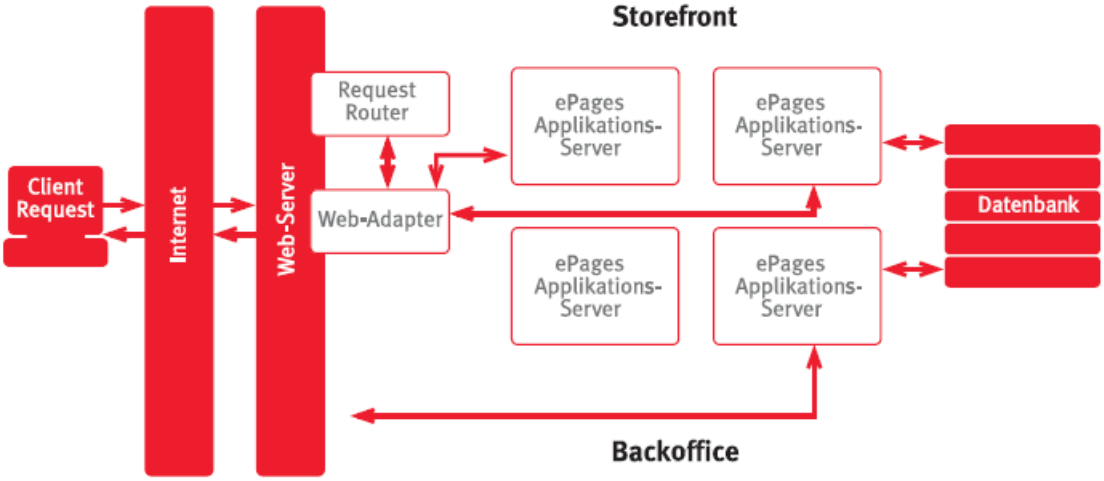


Abbildung 2.1: ePages-Architektur im Überblick [ePa11]

Die Geschäftslogik von ePages läuft auf als *Applikations-Server* bezeichneter Hard- bzw. Software. Daneben existieren Datenbank- und Dateiserver. Wenn ein Benutzer in der Storefront eine Seite lädt, dann wird über einen Webserver eine Verbindung zu den Servern der Provider aufgebaut. Im Hintergrund wartet eine MySQL[3]-Datenbank auf Anfragen von den Applikations-Servern des Providers. Sie enthält alle shoprelevanten Daten, unter anderem über Kunden, Produkte und Design des Shops. Ein Request Router verteilt dabei die Anfragen gleichmäßig auf die Applikation-Server, um einzelne Systeme nicht zu überlasten. Falls die angeforderten Dateien bereits auf einem der Dateiserver liegen, werden sie von der Datenbank nicht neu angefordert, sondern direkt über den Webserver zum Browser des Benutzers geschickt. Diese Art und Weise der Datenbereitstellung gehört zu einem von ePages entwickelten Cache-System. Ein einmal aufgerufener Shop wird auf den Applikations-Servern zwischengespeichert. Wird die Shopseite nun durch andere Kunden erneut geladen, müssen nicht alle anzuzeigenden Daten erneut aus der Datenbank abgefragt werden, was die Ladezeiten erheblich reduziert.
Die Applikations-Server wurden von ePages selbst entwickelt und sind in PERL programmiert. Auf einem einzelnen physischen System können mehrere Instanzen des Servers parallel laufen, abhängig von Prozessorleistung und verfügbarem Arbeitsspeicher. Dabei bedeuten mehr Instanzen eine schnellere Verarbeitung von Anfragen. Sämtliche multimediale Dateien, wie Bilder, Videos, aber auch Design bestimmende Elemente, wie CSS[4]-Dateien, befinden sich aus Performance-Gründen nicht in der Datenbank, sondern auf extra Servern. Lediglich ein Verweis auf diese Dateien stellt die MySQL-Datenbank zur Verfügung [ePa11].

[3] MySQL - Datenbankverwaltungssystem zum einfacheren Umgang mit Datenbanken auf SQL-Basis
[4] Cascading Style Sheet - Sprache zum Beschreiben des Aussehens einer Webseite

2.2 Styles und Anforderungen an den Prototyp

Dieses Kapitel geht auf den grundsächlichen Aufbau der Shopstyles und deren Möglichkeiten zur Anpassung durch den Shopbetreiber ein. Zum weiteren Verständnis wird eine *transform.xml* gezeigt und ihr Aufbau detailiert erklärt. Daraus leiten sich die Anforderungen an den zu erstellenden Prototyp ab.

2.2.1 Styles

Ein Style setzt eine Designidee um, nach welchem alle Elemente der Shop-Storefront[5] gestaltet werden. Dazu gehören unter anderem das Aussehen der Navigationsleiste, des Produktauswahlmenüs oder der Detailseite für ein bestimmtes Produkt. Buttons, Farben, Icons, Schriftarten und sonstige sichtbare Elemente wurden visuell ebenfalls an das Designthema angepasst.
Die Styles setzen sich aus einer Gruppe vordefinierter Templates zusammen. Für bestimmte Bereiche und Funktionen gibt es spezifische, dafür erstellte, Templates. Über das “Merchant Back Office“ - MBO[6], dargestellt in **Abbildung 2.2**, hat der Shopbetreibende die Möglichkeit seinen Shop zu verwalten.

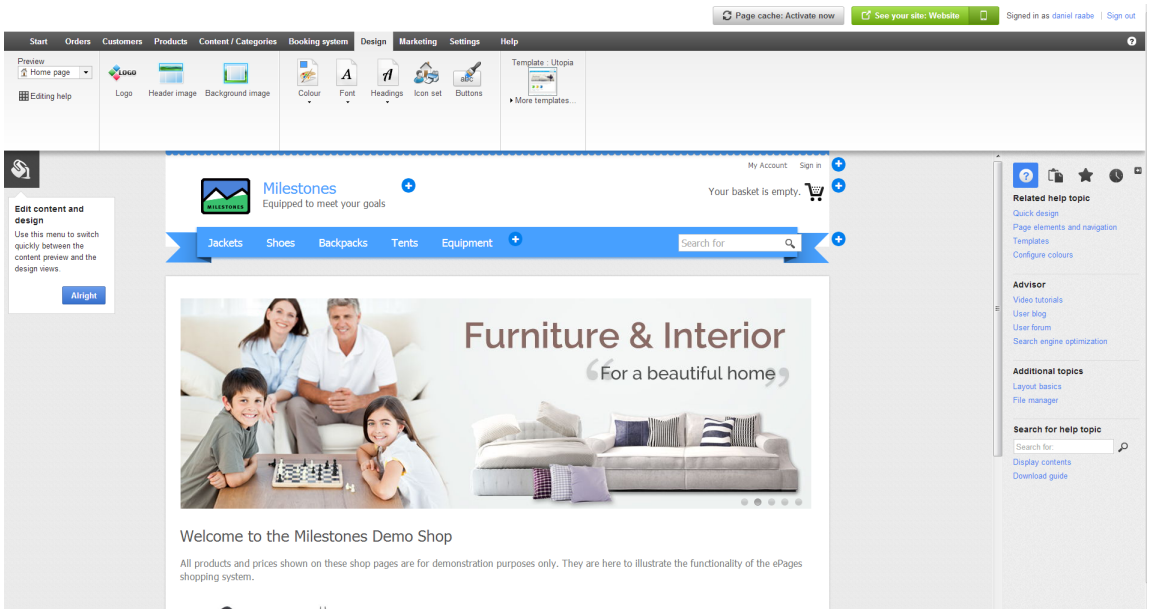


Abbildung 2.2: MBO - Merchant Back Office

Neben den vorrangigen Optionen neue Produkte einzustellen, zu löschen oder die Preise festzulegen, kann er auch das Aussehen des Onlineshops nach seinen Wünschen verändern. Zu diesem Zweck wurde mit ePages 6 der MBO Design Editor eingeführt.

[5] Sichtbarer Bereich für Besucher eines Online-Shops
[6] Im MBO kann der Shopbetreibende seinen Onlineshop gestalten, aber auch alle angebotenen Produkte verwalten. Für Besucher/Käufer im Onlineshop ist dieser Bereich nicht einsehbar.

Ein besonderes Feature ist die Möglichkeit beispielsweise Textfarben oder Hintergrundbilder nach Änderung sofort zu sehen. Das Aussehen des Shops lässt sich in zwei Stufen anpassen. Der “Quick Design“ - Modus bietet dem Betreiber in einem ersten Schritt an schnell und einfach ein Farbschema zu wählen oder die Seitenbreite festzulegen.

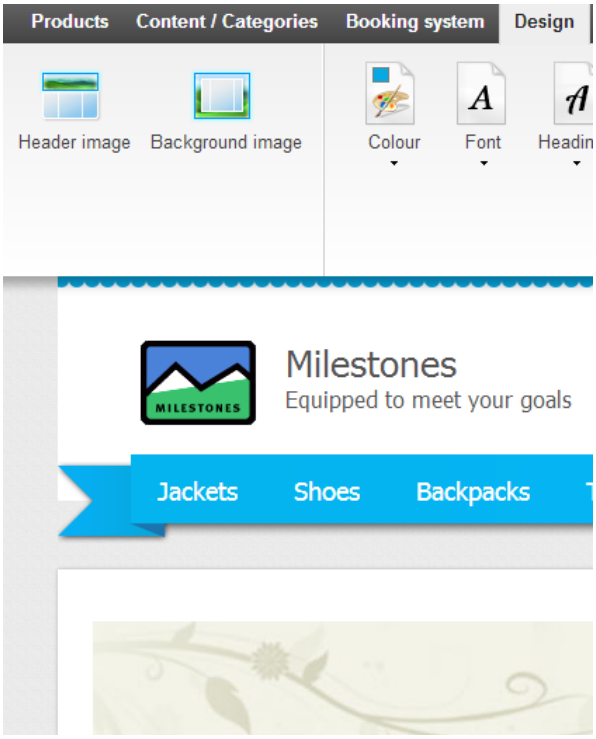


Abbildung 2.3: Ausschnitt MBO - “Quick Design“-Modus

Das nächste Beispiel (siehe **Abbildung 2.4**) zeigt, wie der Shopbetreiber die Farbe der Produktleiste und damit verbundene Elemente farblich anpassen kann. Das Menü *“Colour“* öffnet den Colorpicker[7]. Nach Auswahl der gewünschten Farbe und einem Klick auf *“Apply“* wird im Hintergrund eine Farbtransformation angestoßen. Ohne dass die Seite neu geladen werden muss, wird das Ergebnis sichtbar.

[7] Colorpicker - Auswahlmenü für Farben

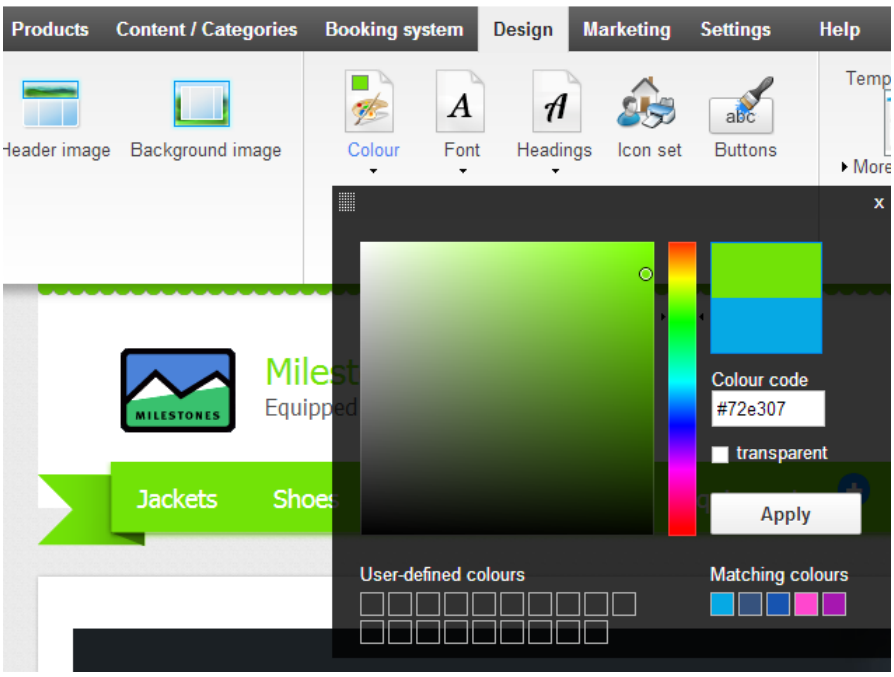


Abbildung 2.4: Angewandte Farbtransformation im Quick Design Modus

Der “Advanced Design“ - Modus erlaubt im zweiten Schritt speziell für bestimmte Bereiche im Shop Farben zu ändern, die Schriftart anzupassen oder Hintergrundbilder hochzuladen.
Die tatsächliche Transformation erledigt das serverseitig über Perl gesteuerte Programm ImageMagick.

2.2.2 PerlMagick und ImageMagick

ImageMagick ist ein frei verfügbares Softwarepaket, mit dessen Hilfe sich Rastergrafiken erstellen und bearbeiten lassen. Sein Quellcode ist offen, was die Bearbeitung der Software für Jedermann möglich macht. ImageMagick kann über 100 der am häufigsten verwendeten Bildformate verarbeiten. Dazu gehören das Einlesen, Modifizieren und Abspeichern. Für die Anwendung im Webbereich besonders interessant ist die Möglichkeit Bilder dynamisch zu generieren. Das Programm wird über eine Kommandozeile bedient [LLC13a].
ePages setzt ImageMagick nicht direkt ein, sondern nutzt die Funktionen von ImageMagick über das Werkzeug PerlMagick. Das ist eine objektorientierte Perl-Schnittstelle, die es erlaubt den vollen Funktionsumfang von ImageMagick in Perl-Skripten zu nutzen. Das ist in sofern günstig, da ePages alle serverseitigen Aufgaben mit Perl erledigt. ePages setzt PerlMagick auf dem Server zum Uploaden von Bildern, z.B. Produktbilder von einem Shopbetreiber, ein. Aber auch zur Anpassung von Grafiken und Farben im MBO eines Shops. Das **Listing 2.1**, entnommen von der Webseite der Entwickler gibt einen Einblick wie PerlMagick angewendet wird. Es werden drei Bilder gelesen und zu-

geschnitten. Anschließend werden sie zu einer animierten Sequenz im MIFF[8]-Format zusammengesetzt [LLC13c]. Das Beispiel demonstriert nur einen Bruchteil der Möglichkeiten, die ImageMagick bietet. Neben Perl existieren auch ImageMagick-Schnittstellen zu anderen Programmiersprachen wie PHP, C++ oder Java.

```
1  #!/usr/local/bin/perl
2  use Image::Magick;
3
4  my($image, $p, $q);
5
6  #Bilder einlesen
7  $image = new Image::Magick;
8  $image->Read('x1.png');
9  $image->Read('j*.jpg');
10 $image->Read('k.miff[1, 5, 3]');
11 $image->Contrast();
12
13 #alle Bilder zuschneiden
14 for ($x = 0; $image->[$x]; $x++)
15 {
16   $image->[$x]->Frame('100x200') if $image->[$x]->Get('magick')
17   eq 'GIF';
18   undef $image->[$x] if $image->[$x]->Get('columns') < 100;
19 }
20 $p = $image->[1];
21 $p->Draw(stroke=>'red', primitive=>'rectangle',
22          points=>20,20 100,100');
23 $q = $p->Montage();
24 undef $image;
25
26 #als MIFF schreiben
27 $q->Write('x.miff');
```

Listing 2.1: Bildbearbeitung-Konvertierung mit ImageMagick

Das **Listing 2.1** hat für dieses Projekt keine Bedeutung. Es gibt lediglich einen Eindruck wie der Quellcode eines PerlMagick-Skripts aussehen kann.
Der nächste Abschnitt zeigt die transform.xml. Sie enthält Werte, die an ein PerlMagick-Skript übermittelt werden, wenn der Benutzer eine Farbe ändert oder ein Bild austauscht.

2.2.3 Aufgabe der transform.xml

Dem Shopbetreiber soll nicht nur die Möglichkeit gegeben werden verschiedene Styles auswählen zu können, sondern auch spezifische Dinge wie beispielsweise die Schrift-

[8] Magick Image File Format - ImageMagick's eigenes plattformunabhängiges Dateiformat, um Bitmap-Bilder zu speichern [LLC13b]

farbe, Farbe der Produktauswahlleiste oder Grafiken in bestimmten Bereichen seiner Shopwebseite zu verändern. Im bereits erwähnten “Quick Design“-Modus kann zum Beispiel die Farbe der Produktauswahlleiste verändert werden. Das Quellcodebeispiel **Listing 2.2** einer *transform.xml* enthält unter anderem das Element *Color* (Zeile 7), dass für solche Farbumwandlungen zuständig ist.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2    <Transformations>
3
4      <!-- object definition -->
5      <Objects>
6        <!-- the main color as given by the application -->
7        <Color Alias="WizardColor1" Value="#FFCC00">
8          <!-- also use this color for h1 headlines -->
9          <Target Alias="ContentHeadline1Color" />
10       </Color>
11
12       <!-- color for h2 headlines -->
13       <Color Alias="ContentHeadline2Color" Value="#FFCC00" />
14
15       <!-- the generated header image -->
16       <Image Alias="HeaderBackground" Value="bgr_header.jpg" />
17
18       <!-- temporary header image -->
19       <Image Alias="tmp_keyvisual" Value="tmp_keyvisual.png"
20                                    Temporary="1" />
21
22       <!-- header image mask -->
23       <Image Alias="header_overlay" Value="header_overlay.png"  />
24     </Objects>
25
26     <!-- transformations -->
27
28     <!-- create h2 color by brightening up WizardColor1 -->
29     <ColorTransformation Source="WizardColor1"
30                          Target="ContentHeadline2Color">
31       <AddColorOffset Hue="0" Saturation="-25" Brightness="20" />
32     </ColorTransformation>
33
34     <!-- header image generation step 1 -->
35     <ImageTransformation Source="WizardImage1"
36                          Target="tmp_keyvisual">
37       <Resize geometry="910" />
38       <Crop gravity="center" geometry="910x165+0+0" />
39       <Composite compose="DstIn" image="header_alpha" />
40     </ImageTransformation>
41
42     <!-- header image generation step 2 -->
43     <ImageTransformation Source="header_overlay"
44                          Target="HeaderBackground">
45       <Canvas background="WizardColor1" gravity="NorthWest"
```

```
46                                              geometry="910x165+0+0" />
47          <Composite compose="SrcOver" image="tmp_keyvisual" />
48        </ImageTransformation>
49
50    </Transformations>
```

Listing 2.2: typischer Aufbau einer transform.xml

Jeder Style hat eine für ihn speziell angepasste *transform.xml*. Sie besteht aus zwei Teilbereichen zwischen den *<Transformations>*-Tags[9].
Im ersten Bereich werden zwischen den *<Object>*-Tags Objekte definiert, die in dem jeweiligen Style vorhanden sind. Gegenwärtig stehen die drei Objekttypen *Color*, *Image* und *Object* zur Auswahl. Jedes Objekt erhält zur Identifizierung ein *Alias*-Attribut. Der Wert kann für Farben *WizardColor1*, *WizardColor2* oder *WizardColor3* sein. Für Bilder entsprechend *WizardImage1* oder *WizardImage2*. Diese Werte werden später im Transformationsteil überschrieben, sobald eine Transformation vom Shopbetreiber vorgenommen wird. Ein *Color*-Objekt kann ein zweites Attribut *Value* aufnehmen, dessen Wert sich nach den möglichen Farbformat-Vorgaben von ImageMagick richtet. Der "Quick-Design"-Modus verwendet das gebräuchlichste Format *#RRGGBB*.
Ein *Image*-Objekt kann ebenfalls ein *Value*-Attribut erhalten. Der Wert ist immer ein Dateiname, meist mit der Endung **.jpg* oder **.png*.

Der zweite Dateiabschnitt, durch den Kommentar *transformations* erkenntlich, widmet sich der eigentlichen Transformation. Es gibt zwei Arten von Transformationen, die *ColorTransformation* und die *ImageTransformation*. Egal welche der beiden ausgeführt werden soll, es sind in jedem Fall die Attribute *Source* und *Target* anzugeben. Ihre Werte müssen jeweils vom gleichen Typ sein. Für eine ImageTransformation bedeutet es, dass die Wertzuweisungen für das *Source*- und das *Target*-Attribut auf eine Bilddatei verweisen müssen, da eine Bildtransformation nicht auf eine Farbe angewendet werden kann. Der Transformationstyp wird dabei als Eltern-DOM-Knoten[10] erstellt, zum Beispiel *<ColorTransformation>*. Seine Kindelemente sind eine Abfolge von Funktionen, die nacheinander beginnend beim *Source*-Objekt ausgeführt werden. Die Werte der Transformation werden schließlich im *Target*-Objekt gespeichert. Soll beispielsweise die Produktleiste unabhängig von der Farbe einen Sättigungswert von *-25* erhalten, so wird nach einer Farbänderung durch den Shopbetreiber der Wert *-25* für die Sättigung sofort auf die neue Farbe angewendet. Bei der *ImageTransformation* verhält es sich ähnlich. Soll ein Bild in einem bestimmten Bereich der Shop-Webseite ausgetauscht werden, so setzt die *ImageTransformation* verschiedene Funktionen wie *Resize* oder *Crop* ein, um das neue Bild für den entsprechenden Bereich anzupassen.

[9] Tag - Dienen dem Auszeichnen und Klassifizieren von Daten. Siehe Kapitel "Recherche relevanter Webtechnologien"

[10] Document Object Model - Wirkt als Schnittstelle zwischen Struktur eines Dokumentes, seinem Design (z.B. CSS) und seiner Funktionalität (z.B. durch JavaScript).

Wie aus dem Beispiel zu erkennen, ist die *transform.xml* der Kern einer jeden Transformation. Die Werte müssen sorgfältig eingetragen und getestet werden. Ein kleiner Fehler endet in einem nicht korrekt angezeigten Design. In dem Fall würden nicht nur Shopbetreibende, sondern auch deren Kunden davon Kenntnis nehmen.

2.2.4 Anforderungen an den Prototyp

Der Fokus beim Erstellen einer XML-Datei sollte sich ausschließlich auf den Inhalt richten. Spitze Klammern, Anführungs- und Gleichheitszeichen sind Teil der korrekten Notation, spielen für den Betrachter jedoch eine untergeordnete Rolle. Um die Übersicht weiter zu verbessern kann ebenso auf schließende Tags verzichtet werden, das Programm erzeugt diese beim Generieren der Datei automatisch. Das reduziert die Fehleranfälligkeit.

Abbildung 2.5: Fokussierung auf inhaltlich wichtige Elemente. Das End-Tag entfällt.

Daraus leitet sich die Anforderung zum Verzicht auf inhaltlich überflüssige Zeichen ab.

Da die ePages-Software sehr komplex ist und XML-Dateien für die Umsetzung oder Speicherung wichtiger Funktionen und Informationen häufig eingesetzt werden, sollte die Option in betracht gezogen werden einen Basis-Editor zu entwickeln, der mit jeder XML umgehen kann. Auf dieser Grundlage könnte der Entwickler aufbauen und den Prototyp speziell für bestimmte XML-Dateien, wie hier die *transform.xml*, anpassen.
Eine weitere Anforderung, welche nicht als “Muss“ formuliert ist, wäre die direkte Ausga-

be einer XML ohne den Umweg über JSON[11] oder serverseitiger Scripte zur Abspeicherung. XML-Dateien können im einfachsten Fall in einem Texteditor wie Notepad erstellt werden. Zur besseren Übersicht wird nach einem Start- und einem Endtag ein Zeilenumbruch zur besseren Übersicht eingefügt. Ebenso werden Kindelemente eingerückt um ihre Abhängigkeit von einem Elternelement schneller zu erkennen. Die Struktur des Aufbaus ist somit vorgegeben.
Daraus leitet sich die Idee für die Trennung von logischer Struktur und der visuellen Darstellung einer XML-Datei ab. Erreicht werden könnte dies durch Elemente, die XML-Knoten repräsentieren und sich auf der Browseroberfläche in eine beliebige Richtung bewegen ließen.

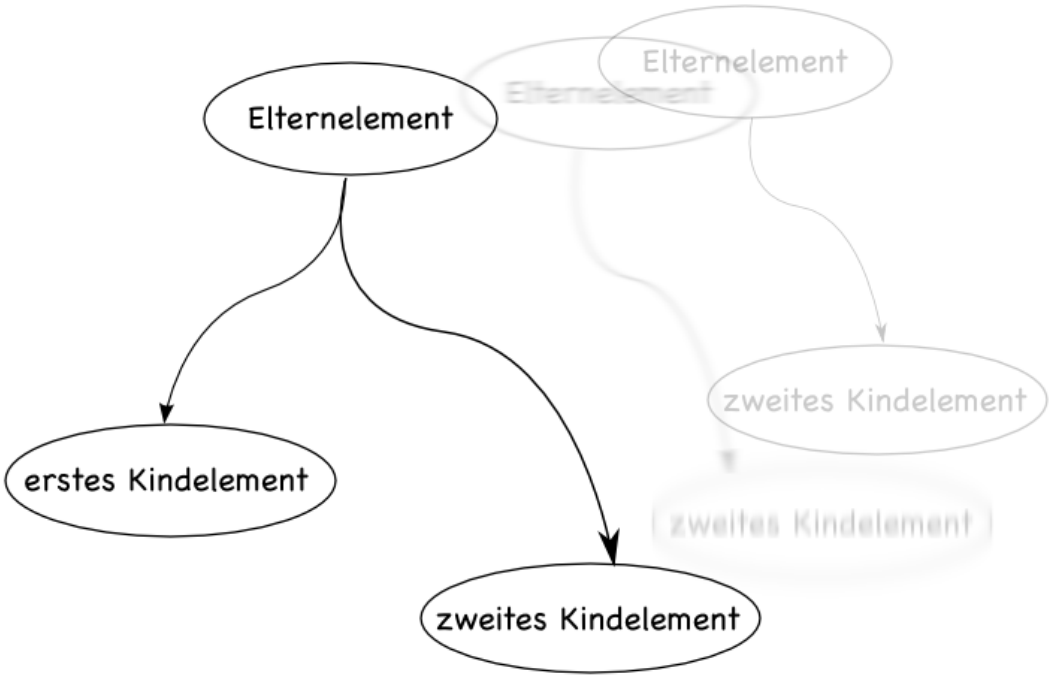
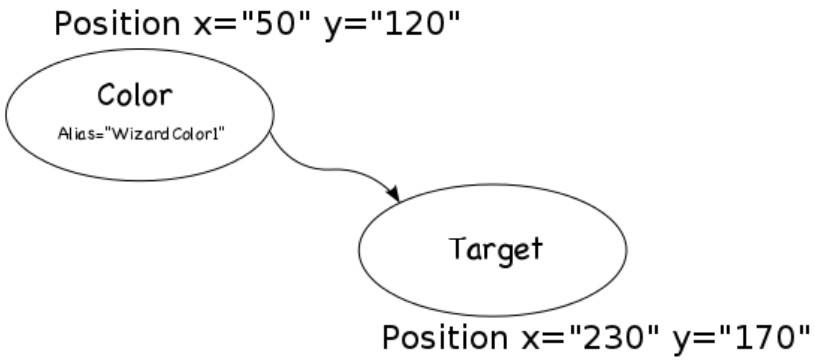


Abbildung 2.6: Alle Elemente sind in jede Richtung verschiebbar.

Dabei könnte eine geometrische Figur in Form eines Rechteckes, Kreises oder einer Ellipse einen Knoten aus der XML-Struktur repräsentieren. Die Abhängigkeiten untereinander ließen sich durch gerichtete Graphen erkennbar machen. Die Position eines Knotenelements auf der Browseroberfläche spielt dann keine Rolle mehr, da die Eltern-Kind-Beziehung durch einen Pfeil dargestellt ist. Ein Benutzer wäre dann in der Lage sein XML Dokument nach seinen Wünschen zu visualisieren. Ein denkbarer Ansatz wäre die Positionen als Attribute in der selben Datei zu speichern.

[11] JavaScript Object Notation - Ein Textformat, dass an die Art und Weise in JavaScript Objekte zu notieren angelehnt ist. Es dient meistens, genauso wie XML, dem Datenaustausch.




```
<Color Alias="WizardColor1" x="50" y="120">
  <Target x="230" y="170" />
</Color>
```

Abbildung 2.7: Überlegung: Positionskoordinaten als Attribute speichern

Diese ließen sich beim Auslesen mit einer JQuery-Methode entfernen und beim Speichern neu schreiben. Beim Laden der generierten XML-Datei wären alle Elemente wieder auf der Position, an die sie zuletzt verschoben wurden.

Die Attribute sollen nach Möglichkeit vorgegeben werden. Um die Anzahl der gleichzeitig angezeigten Informationen auf dem Bildschirm auf einem moderaten Level zu halten, könnten die Attribute über ein ausklappbares Menü zu sehen sein. Alle verfügbaren Attributnamen ließen sich in ein Auswahlmenü, zum Beispiel mittels des HTML5 <select>-Elements laden und zutreffender automatisch vorwählen.

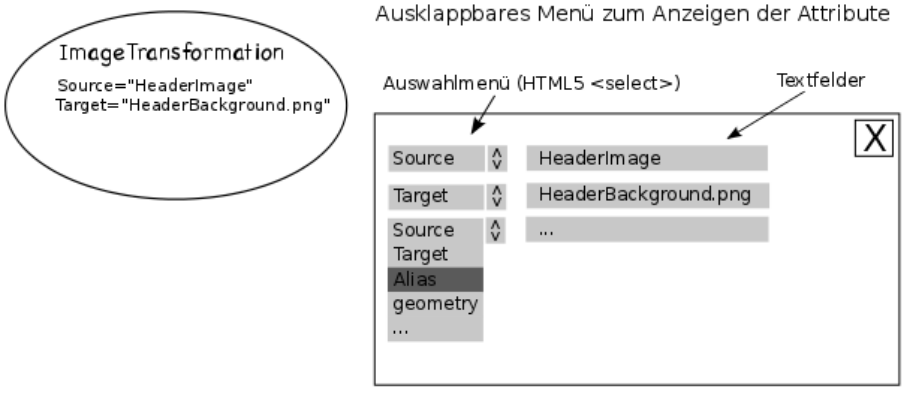


Abbildung 2.8: Vorstellung eines Menüs zum Anzeigen, Auswählen, Hinzufügen und Löschen von Attributen

Die erste Version des Editors soll nach Möglichkeit mit bei ePages eingesetzten Programmiersprachen realisiert werden. Dazu gehören HTML/HTML5, CSS und JavaScript bzw. JQuery als Skriptsprachen.

2.3 Recherche relevanter Webtechnologien

Die folgenden Teilabschnitte geben einen Überblick über die benutzten Webtechniken. Beschrieben werden im ersten Abschnitt die Auszeichnungssprache HTML bzw. der aktuelle Standard HTML5, ohne die keine Webseite existieren würde. Weiterhin wird CSS, eine Sprache zur Webseitengestaltung, vorgestellt. Der zweite Teil gibt einen Überblick über die Skriptsprache JavaScript und das JavaScript-Framework JQuery. Darüber hinaus wird AJAX, eine Technik für dynamischen Seitenaufbau, erklärt.

2.3.1 HTML5

Jede Webseite besteht, unabhängig von ihren Funktionen, aus einem HTML-Grundgerüst. Die erste Zeile gibt an, von welchem Typ das Dokument, d.h. nach welcher HTML-Version es aufgebaut ist.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
                      "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>Titel der Webseite</title>
  </head>
  <body>
  Inhalt der Webseite
  </body>
</html>
```

Listing 2.3: Das HTML-Grundgerüst. Der Dokumententyp ist HTML 4.01

Erst wenn dieses erstellt wurde, kann eine Dynamik mittels Skriptsprachen wie JavaScript implementiert werden. HTML ist eine Textauszeichnungssprache, die 1989 von Tim Berners-Lee auf Grundlage der Metasprache SGML, von der auch XML abstammt, entwickelt wurde. HTML dient dabei lediglich der Strukturierung von Webseiten. Diese können beliebig unterteilt werden, in Kopf-, Inhalts- und Fußbereich. Andere Elemente wie Navigationsleisten, Tabellen oder Überschriften können ebenfalls sehr schnell eingefügt werden.
Eine Textauszeichnung in HTML wird erreicht, in dem alle Elemente in sogenannte Tags gesetzt werden. Um beispielsweise eine Überschrift dritter Ordnung und einen Absatz mit beliebigem Text zu erzeugen benutzt man die Tags <h3> und <p>. Dabei ist zu beachten, dass ein Start-Tag entsprechend der Richtlinien für HTML auch ein End-Tag

benötigt. In dem Fall muss <h3> mit </h3> wieder geschlossen werden. Der Inhalt, der sich nun zwischen diesen Tags befindet, wird größer dargestellt als Text in einem Absatz, der mit <p>-Tags angelegt wird.

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Titel</title>
  </head>
  <body>
    <h3>Überschrift dritter Ordnung</h3>
      <p>Beliebiger Text</p>
  </body>
</html>
```

Listing 2.4: Implementierung einer Überschrift mit einem Absatz. Ebenfalls gezeigt: Stark vereinfachte DOCTYPE-Angabe in HTML5

Da HTML anfangs keinem Standard unterlag und es innerhalb von zwei Jahren in den Versionen 2.0 (1995), 3.2 (1997) und 4.0 (1997) und der bis heute gültige Standard HTML 4.01 (1999) erschien, gründete sich das World Wide Web Consortium, kurz W3C. Diese Vereinigung legte in den Folgejahren Regeln für den Umgang mit HTML fest. Nachdem in kurzen Abständen vier Versionen erschienen, wurde es ruhig um HTML. Der Schwerpunkt lag nun mehr auf XHTML und XHTML 2, Weiterentwicklungen durch das W3C. Die Unterschiede bestanden im Wesentlichen darin bestimmte Tags im neuen Standard als ungültig zu erklären und stattdessen neue Elemente in den Standard aufzunehmen. Die Tags wurden von neueren Browsern zwar noch akzeptiert, verloren aber ihre Gültigkeit, was in Darstellungsfehlern resultieren konnte. Um zu überprüfen, ob eine Webseite tatsächlich ihrem angegeben Standard entspricht, bietet das W3C einen Validierer an mit dessen Hilfe ungültige Tags ermittelt und gegebenfalls vom Entwickler geändert werden können. [LAS10, S. 1]
Seit 2004 wird auf Basis von HTML 4.01 der Nachfolgestandard HTML5 entwickelt, diesmal spezifiziert durch die *Web Hypertext Application Working Group*, WHATWG. 2006 nahm auch das W3C wieder Einfluss auf den neuen Standard und veröffentlichte 2008 den ersten Entwurf von HTML5. Ein Jahr später werden die Arbeiten an XHTML 2 eingestellt. HTML5 bringt viele neue nützliche Elemente mit sich, unter anderem wird das Einbinden von Videos und Audiodateien vereinfacht. Bisher wurden mediale Inhalte häufig über Flash implementiert. HTML5 führt für diese Aufgabe das neue <video>- und <audio>-Element ein. Dieses verkürzt die zu schreibenden Quelltextzeilen erheblich. Eine weitere Neuerung ist die Unterstützung für Inline-SVG. SVG dient der Beschreibung von 2D-Vektorgrafiken mithilfe des XML-Standards. Musste ein Programmierer vor HTML5 eine externe *.svg-Datei einbinden, so ist es nun erlaubt mit aktuellen HTML5-DOM <svg>-Elemente zu erstellen und eine Grafik zu erzeugen. Diese Art der Vektorgrafikerstellung nutzt auch der XML-Editor. [LAS10, S. 1]

HTML5 basiert auf verschiedenen Prinzipien, formuliert im WHATWG Standard. Dazu gehören zum einen Kompatibilität und die Priorität von Quelltext des Entwicklers. Kompatibilität bedeutet, dass nicht unterstützte HTML5 Features keinen Einfluss auf die Darstellung einer Webseite haben. Stattdessen wird der Browser ein anderes Element verwenden, was vom Betrachter unbemerkt bleibt. Beispielsweise wird das neue <header>-Element zur Einteilung der Bereiche einer Webseite eingeführt. Bisher wurde dies mit einem DIV id="header" gelöst.
Die eingangs erwähnte Priorität von Schreibweisen des Entwicklers bedeutet soviel wie, dass eine korrekte Schreibweise, wie sie der Standard vorsieht, im Zweifelsfall vom Entwickler ignoriert werden kann. Dies hätte trotzdem keinen Einfluss auf die Darstellung der Seite. Das **Listing 2.5** zeigt drei unterschiedliche Schreibweisen. Bis HTML 4.01 und XHTML 2 war nur die erste erlaubt [LAS10, S. 4].

```
1 <div id="header">
2 <div id=header>
3 <div ID="header">
```

Listing 2.5: Verschiedene zulässige Schreiben in HTML5

2.3.2 CSS

Neben HTML und seinen Nachfolgern hat sich Cascading Style Sheets, kurz CSS, sehr erfolgreich in die Welt der Webseiten integriert. So gut wie jede Webseite nutzt gegenwärtig CSS zur Umsetzung einer Design-Idee. Mitte der Neunziger Jahre kam der Vorschlag auf die Struktur einer Webseite und ihr Aussehen voneinander zu trennen. Statische Navigationsleisten, umfangreiche Gestaltung von Buttons oder aufklappbare Menüs sind einige der unzähligen Funktionen von CSS. Dabei wurde besonders viel Wert darauf gelegt den Gestaltungsprozess möglichst einfach und übersichtlich zu halten. Alle CSS-Informationen über das Aussehen einer Webseite lassen sich auslagern, wodurch die Struktur der Seite nicht angerührt wird. CSS arbeitet über das Festlegen von Regeln. Enthält ein Text beispielsweise Überschriften erster und dritter Ordnung, so lassen sich mit zwei unterschiedlichen Regeln alle Überschriften erster Ordnung in blauer und alle Überschriften dritter Ordnung in roter Farbe darstellen (siehe **Listing 2.6**). Sollen beispielsweise alle h3-Überschriften in grüner Farbe erscheinen, wird einfach *red* durch *green* ersetzt. Die Funktionsweise von CSS erlaubt es die Regeln für HTML-Elemente an einer zentralen Stelle, etwa einer externen CSS-Datei zu ändern und anschließend auf alle gewünschten Elemente anzuwenden. Den gleichen Effekt erreicht man, in dem alle CSS-Regeln im <head>-Bereich des Dokuments definiert werden. Darüber hinaus besteht die Möglichkeit Stylesheet-Regeln in die Attribute eines HTML-Elements aufzunehmen. Bei größeren Projekten müssen im Falle von Änderungen alle Elemente einzeln angepasst werden, was einen erheblichen Mehraufwand für Designmodifizierungen bedeutet.

```
<style type="text/css">
<!-- Alle Überschriften 1. Ordnung werden blau dargestellt -->
h1 {color: blue};

<!-- Alle Überschriften 3. Ordnung werden rot dargestellt -->
h3 {color: red};
</style>
```

Listing 2.6: einfache CSS-Regeln

Die Vorteile von CSS bestehen also darin, dass es viel mehr Gestaltungsmöglichkeiten als HTML bietet. Es lassen sich Absätze formatieren wie sie in Printmedien, etwa Magazinen oder Zeitungen, erscheinen. Ebenso können Zeilenabstände beliebig gewählt werden. Außerdem lässt sich mit der CSS-Syntax im Vergleich zu HTML der Umfang an Quelltext deutlich reduzieren. Dadurch wird die Webseite nicht nur besser gestaltet, auch die Ladezeiten nehmen ab. Die Trennung der Struktur einer Webseite von ihrer Gestaltung ist mit CSS ebenso vollständig umsetzbar. Design-Regeln lassen sich in einer externen Datei kapseln, wodurch sie für mehrere Webseiten durch Einbinden im *<head>*-Bereich verfügbar sind. Durch Modifizieren der CSS-Datei, welche alle Gestaltungsmerkmale enthält, können in wenigen Schritten mehrere Webseiten auf einmal in ihrer Erscheinung angepasst werden. [McF12]

2.3.3 SVG

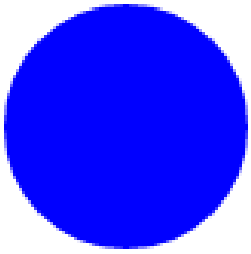
SVG ist eine Sprache, die unter Verwendung des XML-Formats der Beschreibung zweidimensionaler Grafiken dient. Es unterscheidet zwischen drei verschiedenen Grafik-Objekten: geometrische Formen als Vektorgrafik, Bilder und Text. Die grafischen Elemente können unter anderem transformiert, gruppiert oder mit vorgerenderten Objekten zusammengeführt werden. Darüber hinaus bietet SVG Filter-Effekte oder Alpha-Masken an.
Ebenso können Objekte auf zwei Arten animiert werden. Entweder direkt über SVG-Animations-Elemente oder mithilfe einer Skriptsprache, etwa JavaScript. Für jedes SVG-Element wird ein eigener Knoten im DOM der Webseite angelegt. Das bedeutet, dass ein grafisches Objekt mit Methoden zur DOM-Manipulation direkt angesprochen und modifiziert werden kann [W3C11].

Bis zum Erscheinen von HTML5 stand die SVG-Beschreibung einer Grafik in einer externen Datei mit der Endung *.svg. Diese wurde anschließend in das HTML-Dokument eingebunden. Das **Listing 2.7** stellt den Inhalt einer SVG-Datei dar, die beim Aufrufen einen blau gefüllten Kreis zeichnet.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
```

```
3 "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
4
5 <svg xmlns="http://www.w3.org/2000/svg" version="1.1">
6   <circle cx="100" cy="50" r="40" fill="blue" />
7 </svg>
```

Listing 2.7: Inhalt einer SVG-Datei



(a) Kreis aus **Listing 2.7**

(b) Rechteck aus **Listing 2.8**

Der HTML5-Standard erlaubt die direkte Einbettung von SVG an der Stelle im Dokument, wo es benötigt wird. Im zweiten Beispiel (**Listing 2.8**) ist ein HTML5-Grundgerüst zu sehen, in dessen body-Bereich eine SVG-Grafik eingefügt wurde.

```
1 <!DOCTYPE html>
2 <html>
3   <body>
4
5     <svg xmlns="http://www.w3.org/2000/svg" version="1.1">
6      <rect x="50" y="20" rx="20" ry="20" width="150" height="150"
7        style="fill:red;stroke:black;stroke-width:5;opacity:0.5" />
8     </svg>
9
10  </body>
11 </html>
```

Listing 2.8: In HTML5 eingebetteter SVG-Code [w3sa]

2.3.4 JavaScript

JavaScript ist die “Sprache des Web“ [Cro08, S. 2]. Die Anforderungen an heutige Webseiten wären ohne JavaScript nur schlecht zu realisieren. Möchte ein Entwickler neben fortschrittlichem Design Funktionen wie beispielsweise aufklappbare Dialoge in seine Seite implementieren, wird er vermutlich als erstes an JavaScript denken.
Obwohl diese Programmiersprache kurz nach ihrem Erscheinen nicht sehr beliebt war, da schlecht spezifiziert, hat sie sich zu einer der meist verwendetsten Sprachen im Bereich der Webprogrammierung entwickelt. JavaScript wird standardmäßig clientseitig,

das bedeutet direkt zur Laufzeit im Browser des Nutzers ausgeführt. Ein großer Vorteil beim Einsatz von JavaScript ist das dynamische Erzeugen und Manipulieren von HTML-Elementen ohne die Webseite neu zu laden. Müsste der Browser einen neuen Request senden, würde sich die Ladezeit beträchtlich erhöhen.
Für die Umsetzung des Prototyps wurde ausschließlich JavaScript bzw. JQuery zur Programmierung von Funktionen eingesetzt.

2.3.5 AJAX

“AJAX“ - das steht für *Asynchronous JavaScript and XML*. Seit dem Jahr 2005 erlangte AJAX in der Webwelt einen regelrechten Durchbruch. Es war nun möglich Webseiteninhalte dynamisch mit frischen Informationen zu laden, ohne dass der Browser dafür einen neuen Request senden musste, der sich in einer höheren Ladezeit bemerkbar macht. AJAX benötigt keine neuen Browser-Plugins, da alle Techniken dem Browser bereits bekannt sind. Aber die Art und Weise wie Auszeichnungs- und Skriptsprachen, etwa XML und JavaScript, zusammenwirken ist neu und unterstützt die Personalisierung des Web.

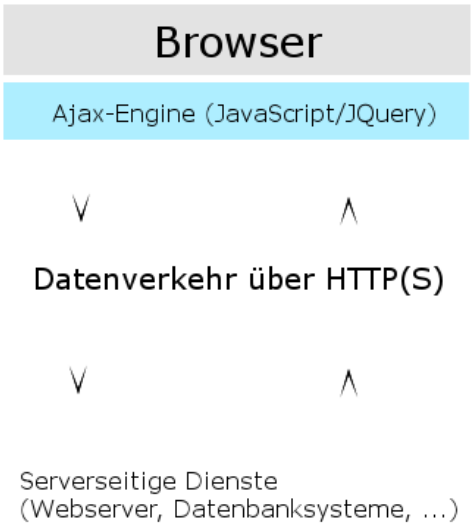


Abbildung 2.10: AJAX kann gezielt Daten zwischen Browser und Webserver austauschen

In diesem Prototyp wird der Einsatz von AJAX dadurch bemerkbar, dass beim Laden der XML-Datei die Menüleiste nicht neu geladen wird. Bei diesem eher kleinen Projekt macht dies kaum einen Unterschied. Stelle man sich jedoch ein großes webbasiertes Programm vor mit einem beträchtlichen DOM-Umfang, interaktiven Elementen und multimedialen Inhalten, so ist die kürzere Ladezeit durch gezieltes Neuladen einzelner Webseitenbereiche für den Benutzer deutlich angenehmer. Beispielsweise können auch multimediale Inhalte wie Video- oder Audiodateien in einem Bereich der Webseite abgespielt werden und gleichzeitig andere Teilbereiche der Webseite aktualisiert werden,

ohne das Abspielen zu unterbrechen.
So schön wie die Technologie für Entwickler ist, sie besitzt auch einige Nachteile. Das dynamische Ändern von einzelnen Webseitenelementen bewirkt im Browser kein Anlegen einer Webseite im Verlauf mit diesen Änderungen. Dies kann zur Falle werden, wenn man nun versucht eine Änderung mit dem *Zurück*-Knopf rückgängig zu machen. Denn in diesem Fall landet der Benutzer auf der vorher besuchten Seite. Um dieser Problematik entgegenzuwirken müssen vom Entwickler eigens für seine AJAX-basierte Anwendung *Rückgängig*- und *Speichern*-Buttons angeboten werden. [BB05]

2.3.6 JQuery - ein JavaScript Framework

JQuery ist ein JavaScript-Framework, dass von John Resig entwickelt und 2006 in New York auf dem BarCamp veröffentlicht wurde. Ein Framework ist eine Kapselung von zusammenwirkenden Methoden.

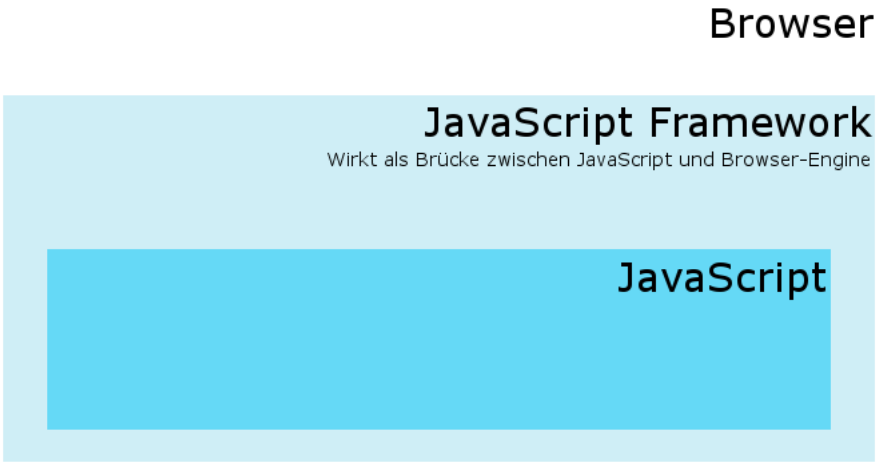


Abbildung 2.11: Ein JS-Framework vermittelt zwischen JavaScript und Browser

Mit der eigentlichen Logik muss sich der Entwickler später nicht mehr auseinandersetzen. Er muss sich lediglich mit der Funktion vertraut machen und wissen, welche Werte eine bestimmte Methode erwartet. Bevor die kurzen JQuery-Befehle benutzt werden können, muss eine JQuery Scriptdatei in das HTML-Dokument eingebunden werden. Durch seine Beliebtheit hat sich um JQuery eine große Entwicklergemeinde gebildet, was darin resultiert, dass in kurzen Abständen neue JQuery-Module veröffentlicht werden. Zudem existiert zu JQuery eine überaus gute Dokumentation, die sich in einen Teil mit Erklärungen und anschließendem Beispielteil gliedert. Am Rande sei erwähnt, dass es spezielle JQuery-Bibliotheken unter anderem zur Benutzerflächenentwicklung (jqueryui) und für den mobilen Bereich (jquerymobile) gibt. Da JQuery nichts anderes als verpacktes JavaScript ist, verlor auch letzteres seinen anfangs schlechten Ruf. Der

Begriff DOM-Scripting, der sich neben dem selten verwendeten Wort DHTML durchgesetzt hat, bedeutet in den meisten Fällen den Einsatz von JQuery oder JavaScript, um die komplexen Funktionen, wie wir sie heute aus dem Weballtag kennen zu realisieren. Der Erfinder von JQuery, John Resig, legte viel Wert auf möglichst kurzen Code. Dieser sollte jedoch aussagekräftig bleiben, sodass er sich für die Kernfunktion *$()* entschied, womit eine Doppelfunktion für die Selektion der DOM-Elemente als auch für Methoden zur Verfügung gestellt wird. [VB11]
Welche Einsparung, JQuery an Quellcode bringt zeigen die **Listing 2.9** bis **Listing 2.11**. Die Selektierung mit reinem JavaScript bedeutet mehr Schreibarbeit. Gerade bei größeren Projekten trägt JQuery deutlich zu einer besseren Übersicht des Quelltextes bei.

```
1 <div id="nav"></div>
```

Listing 2.9: Erzeugung eines div-Elementes im DOM

```
1 var s = document.getElementById("nav");
```

Listing 2.10: Selektierung mit JavaScript

```
1 var s = $("#nav");
```

Listing 2.11: Verkürzte Schreibweise mit JQuery

3 Formulieren von Lösungsansätzen

Das Kapitel gibt Aufschluss darüber, warum das lange Zeit für solche Aufgaben eingesetzte Flash nicht geeignet ist. Außerdem wird das neue HTML5 <*Canvas*>-Tag, zusammen mit Frameworks, die es nutzen, kurz vorgestellt. Zum Schluss wird auf die Funktionsweise der für den Prototyp verwendeten JavaScript-Grafik-Bibliotheken RaphaëlJS und Dracula Graph Library eingegangen.

3.1 Flash

Noch vor einigen Jahren hätte ein Entwickler für die Erstellung eines webbasierten XML-Editors vermutlich Flash von Adobe eingesetzt. Der Umgang mit Vektor- und Rastergrafiken, sowie deren Animation und Manipulation sind die Stärke von Adobes Softwaretechnik Flash. Mithilfe der integrierten objektorientierten Programmiersprache ActionScript können Inhalte beliebiger Art kreiert werden. Vor allem für die Programmierung von Browserspielen oder interaktiver Lernsoftware ist Flash sehr beliebt. Das Einbetten von Video- und Audiodateien ist ebenso möglich, wie deren Streaming oder die Ansteuerung von Webcams und Mikrofonen. Um animierte Vektorgrafiken zu erstellen, bietet Flash eine gute Grundlage, da Adobe neben der Technologie Flash auch eine Entwicklungs- und eine Laufzeitumgebung anbietet. Damit sind, anders als bei HTML5, alle relevanten Softwaremodule aus einem Haus, was die Kompatibilität nicht vom verwendeten Browser abhängig macht. Dem lässt sich hinzufügen, dass Flash auf beinahe jedem PC installiert ist, selbst ältere Browser wie der Internet Explorer 6 unterstützen Flash [Hea13]. HTML5 hingegen wird bis heute nicht einmal von den neuesten Browserversionen vollständig unterstützt. Allerdings beschränkt sich dies auf einige wenige Elemente, wie in **Abbildung 3.1** zu sehen. Deutlich vorn liegt der Google Chrome. Er unterstützt nahezu 90% aller HTML5-Elemente. Dicht darauf folgen Opera (84%) und Firefox (81%). Es schließen sich Apples Safari 7.0 (72%) und Microsofts Internet Explorer 11 (69%) an.

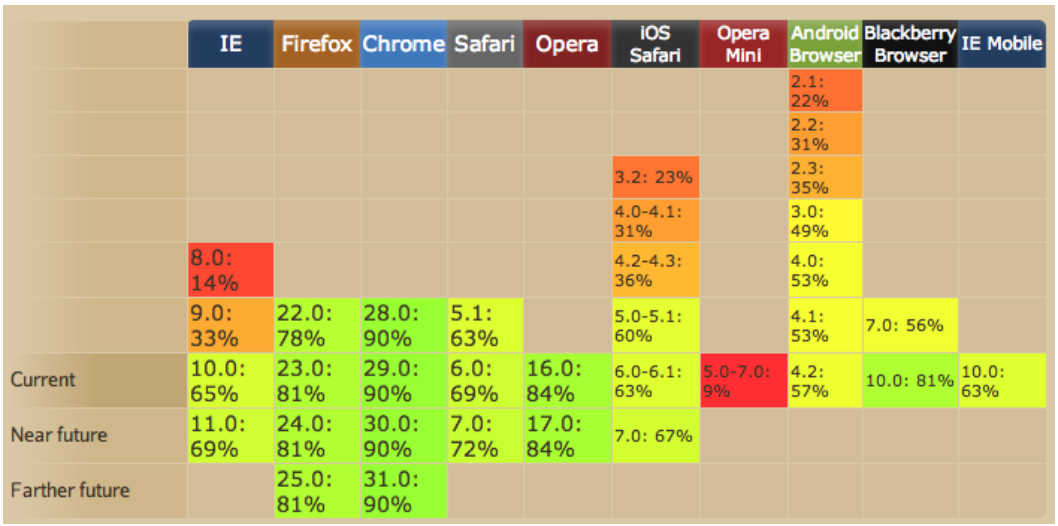


Abbildung 3.1: Prozentualer Anteil unterstützter HTML5 Elemente [can13]

Auffallend ist, dass die Verwendung von Flash einen zunehmenden Abwärtstrend aufweißt. Verwendeten im September 2012 noch 22,8% aller Webseiten die Flashtechnologie, so sank die Zahl im September 2013 auf 17,5% [W3T13].

	2012 1 Oct	2012 1 Nov	2012 1 Dec	2013 1 Jan	2013 1 Feb	2013 1 Mar	2013 1 Apr	2013 1 May	2013 1 Jun	2013 1 Jul	2013 1 Aug	2013 1 Sep	2013 1 Oct	2013 28 Oct
None	7.5%	7.6%	7.7%	7.4%	7.3%	7.2%	7.2%	7.2%	11.0%	10.4%	10.5%	10.4%	10.3%	10.4%
JavaScript	92.2%	92.1%	92.1%	92.4%	92.5%	92.6%	92.6%	92.6%	88.7%	89.3%	89.2%	89.4%	89.4%	89.3%
Flash	22.5%	22.0%	21.5%	21.1%	20.6%	20.3%	19.8%	19.4%	18.5%	18.3%	17.9%	17.5%	17.1%	16.6%
Silverlight	0.2%	0.2%	0.2%	0.2%	0.2%	0.2%	0.2%	0.2%	0.2%	0.2%	0.2%	0.2%	0.2%	0.2%
Java	0.2%	0.2%	0.2%	0.2%	0.2%	0.2%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%

Abbildung 3.2: Prozentualer Anteil der Nutzung client-seitiger Programmiersprachen [W3T13]

Am Beispiel des HTML5 <video>-Elements kann ein deutlicher Zuwachs aller weltweit eingesetzten Browserversionen verzeichnet werden, die es unterstützen. Bereits 50,5% aller Browser können mit dem <video>-Tag umgehen (Stand 2011) [Hef11]. Das ist, gemessen seit 2009, ein Zuwachs von 66%. Dieser Umstand betont die zunehmende Unterstützung der HTML5-Technologie. Die Angaben gehen aus einem globalen Vergleich der verwendeten Browser hervor [Sta11].

Neben den anfangs erwähnten Vorteilen, bringt Flash auch einige Nachteile mit sich. Um flashbasierte Programme auszuführen, muss das Endgerät und das darauf installierte Betriebssystem neben einem geeigneten Browser zusätzlich den Adobe Flashplayer unterstützen. Selbst wenn die Hardware über die Leistungsfähigkeit verfügt, Flash ausführen zu können, so wurde von Apple das Ausführen von Flashanwendungen auf mobilen Endgeräten, wie dem iPad oder dem iPhone, nicht vorgesehen. Auch das Google-Betriebssystem Android unterstützt in neueren Versionen kein Flash mehr. Das

ist der Grund weshalb die ePages-Software auf Flash verzichtet bzw. flashbasierte Projekte im Anfangsstadium verblieben.
Ein weiterer Nachteil, der häufig diskutiert wird, ist die mangelnde Sicherheit, weswegen der Flashplayer von Entwicklern und Fachpublikum immer wieder in die Kritik gerät. Häufige Sicherheitsupdates sind die Folge, was für den Nutzer meist zusätzlichen Installations- und Zeitaufwand bedeutet.
Seit 2005 gibt es mittlerweile eine gute Alternative zu Flash. Das HTML5 <Canvas>-Element stellt für viele grafiklastige Programme einen guten Ersatz dar.

3.2 HTML5 <Canvas>

Mit der Einführung von HTML5 und der starken Verbreitung von JavaScript ist es mittlerweile möglich grafische Inhalte weitestgehend ohne Flash zu erstellen. Das Canvas-Element stellt Skripte zur Verfügung, die es erlauben auf einer auflösungsabhängigen virtuellen Leinwand Grafiken für Diagramme, Kunst oder Spiele zu erzeugen. Das neue Canvas-Element bietet Methoden für die pixelgenaue Ansteuerung und Manipulation von grafischen Elementen auf der Zeichenfläche [WHA13]. Jedes Objekt wird mit JavaScript erstellt und beim Zeichnen in eine Rastergrafik umgewandelt. Da Canvas keine Skalierbarkeit unterstützt, verlieren einmal gerenderte Grafiken beim Vergrößern schnell an Schärfe. Beim Einsatz des Canvas-Elementes muss die Vorüberlegung getroffen werden, ob Mausevents vom Programm unterstützt werden müssen. Ist das der Fall, sind beim Einsatz von Canvas vom Entwickler selbst Mausevents anzulegen.

3.3 Paper.js und Processing.js

Paper.js, Processing.js und das im folgenden Abschnitt behandelte RaphaëlJS sind JavaScript-Grafikbibliotheken, die das virtuelle Zeichnen auf Webseiten erleichtern und gegenwärtig die führenden Bibliotheken für diese Zwecke sind.
Der Entwickler steht vor der Wahl sich für eine der drei Bibliotheken zu entscheiden. Dabei lehnt sich seine Entscheidung an das spätere Einsatzgebiet seiner Anwendung an. Neben dem Unterstützen von älteren Browsern oder einer Unterstützung für Android, spielt vor allem Interaktion mit den grafischen Elementen eine wichtige Rolle. Paper.js und Raphael.js unterstützen bereits Aktionen wie Klickevents oder das Verschieben von Elementen. Liegt der Fokus eher auf Animationen ist Processing.js vorzuziehen. Aber für die Entwicklung dieses Programms steht die Interaktion im Vordergrund.

Kategorie	**Paper.js**	**Processing.js**	**RaphaëlJS**
Technologie	<canvas>	<canvas>	SVG
Sprache	PaperScript	Processing Script	JavaScript
Model	Vektor und Raster	Raster	Vektor

Tabelle 3.1: Vergleich von drei JavaScript-Grafikbibliotheken [Gro12]

Wie in der Tabelle erkennbar, nutzen alle Frameworks unterschiedliche Ansätze bezüglich der Sprache, in der sie geschrieben sind.
Paper.js wurde in der JavaScript-Erweiterung PaperScript geschrieben. PaperScript wird dabei in einer Art Sandbox ausgeführt, das bedeutet der globale Namensraum von JavaScript wird nicht mit Variablen und Funktionen von PaperScript "verunreinigt". Das ist in soweit gut, da es auf diese Weise zu keinen Namenskonflikten von Funktionen oder Variablen des Standard-JavaScript kommen kann. Darüber hinaus wird von Paper.js, genau wie RaphaëlJS, nicht nur eine geometrische Form gezeichnet, sondern ebenfalls ein Objekt angelegt. [Gro12]
Im folgenden Kapitel wird darauf näher eingegangen.

Processing.js bildet als Einzige der drei Bibliotheken eine Ausnahme. Es basiert auf der Programmiersprache *Processing*. Zugleich ist es eine Entwicklungsumgebung und Internetgemeinschaft. [Com13b] Processing läuft in einer virtuellen Java Maschine. Das Framework weißt gewisse Ähnlichkeiten mit Java auf. Beispielsweise ist die Klassennotation stark an Java angelehnt. Processingcode wird beim Kompilieren direkt in Javacode umgewandelt [Com13a]. Gute JavaScript Kenntnisse sind allerdings von Vorteil um mit Processing entwickeln zu können.
Im Vergleich zu den anderen Bibliotheken wird von Processing beim Zeichnen der Elemente kein Objekt angelegt. Das Quellcodebeispiel **Listing 3.1** zeigt das Zeichnen eines Rechtecks. Dieses wird rot gefüllt. Die Funktion *draw()* erzeugt das Rechteck, gibt allerdings kein Objekt des Rechtecks zurück, dass man manipulieren könnte.

```
1  //Zeichenfläche anlegen
2  void setup() {
3     size(420, 300);
4  }
5
6  //Rechteck zeichnen
7  void draw() {
8     background(#FFF);
9     translate(100, 100);
10    fill(#00BF32);
11    noStroke();
12    rect(10, 10, 80, 80);
13 }
```

Listing 3.1: Processing.js - Ein Objekt des Rechtecks wird nicht zurückgegeben

Das Rechteck ist eine Rastergrafik und ein fester Bestandteil der gerenderten Zeichenfläche. Soll später die Farbe des Rechteckes geändert werden, muss ein neues Rechteck gleicher Größe über dem vorhandenen erzeugt werden. Ein Vorteil dieser Methode ist allerdings, dass Animationen mit einer hohen Bildwiederholfrequenz ablaufen, da der Speicher nicht mit Objekten gefüllt ist. Außerdem werden von Processing.js, im Gegensatz zu RaphaëlJS, keine DOM-Knoten der Objekte angelegt. Das Suchen und Mani-

pulieren von DOM-Elementen bedeutet für den Browser viel Arbeit, die er in diesem Fall nicht leisten muss. [Gro12]

3.4 RaphaëlJS Library

Die RaphaëlJS Library [Bar13] vereint viele Werkzeuge, die den Umgang mit SVG erleichtern. Ohne SVG-Kenntnisse lassen sich beispielsweise schnell und einfach komplexe Diagramme oder interaktive Karten erstellen. SVG-Elemente sind nach dem XML-Schema aufgebaut. Dadurch können sie als DOM-Knoten (siehe **Abbildung 3.7**) mithilfe von Skriptsprachen wie JavaScript manipuliert werden. Zum Beispiel ist es vom Programmierer nicht erforderlich selbst Mouseevents zu erstellen.

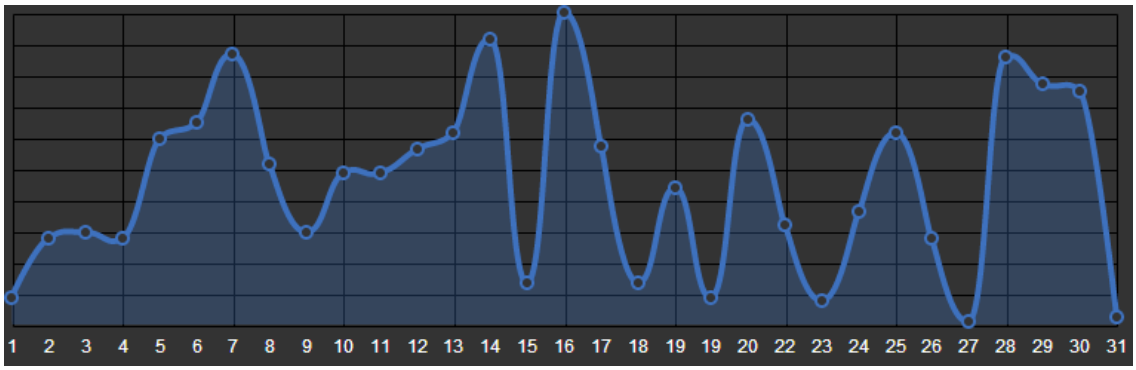


Abbildung 3.3: Mit RaphaëlJS erstelltes Diagramm

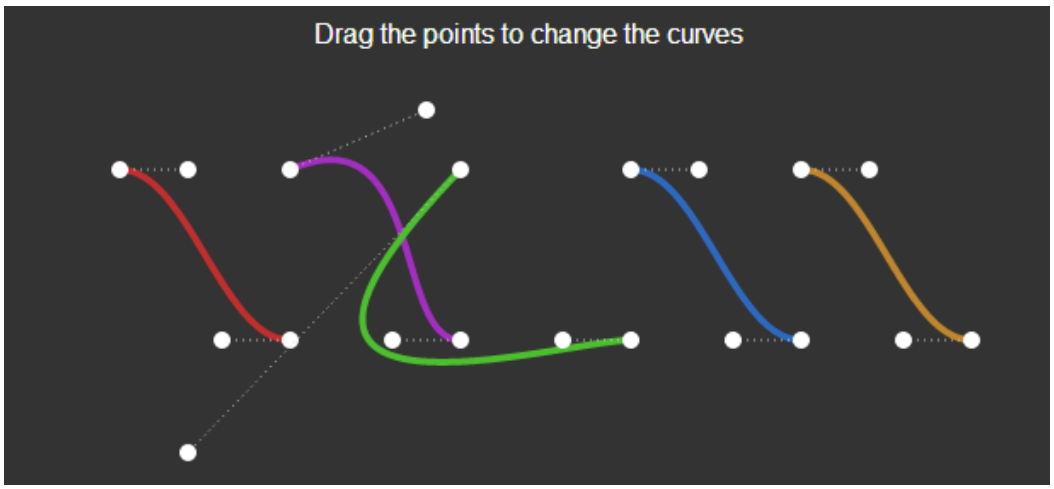


Abbildung 3.4: Bezier-Kurven mit RaphaëlJS

Ein weiteres interessantes Beispiel (siehe **Abbildung 3.5**) ist die vereinfachte Darstellung eines Graphen mithilfe von RaphaëlJS. Die untereinander verbundenen Elemente lassen sich mit der Maus verschieben.

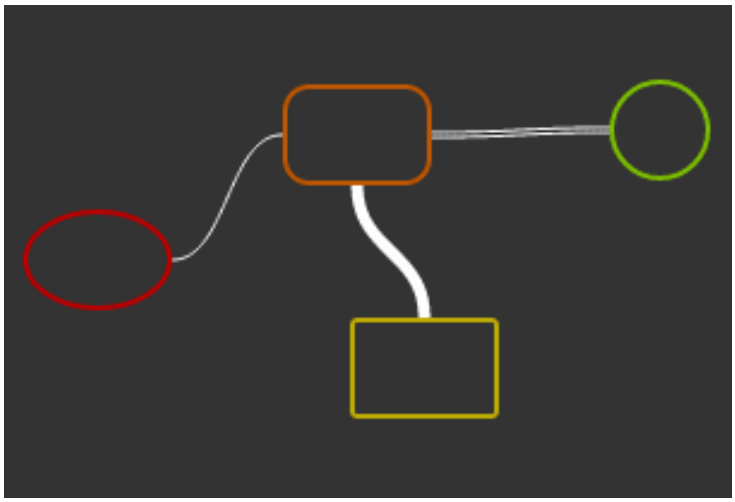


Abbildung 3.5: Mit SVG-Technologie entstandener Graph

Nach dem Einbinden der Datei *raphael.js* kann man, wie im Quellcodebeispiel gezeigt, die Funktionen nutzen, um beispielsweise ein Rechteck zu zeichnen. Das Erstellen von geometrischen Formen mit RaphaëlJS kann auch als objektorientiertes Zeichnen beschrieben werden, da wie im Beispiel ein Rechteck gezeichnet und gleichzeitig ein Objekt des selbigen angelegt wird. RaphaëlJS bietet darüber hinaus sehr viele Werkzeuge, um die erstellten Elemente zu modifizieren. So lässt sich, wie im Beispiel **Listing 3.2** gezeigt, durch Anhängen der Methode *.attr()* und dem Attribut *fill* das Rechteck mit einer Farbe füllen.

```
1 //Zeichenfläche anlegen bei x=0, y=0, Breite=420px, Höhe=300px
2 var r = Raphael(0, 0, 420, 300);
3 //Rechteck erzeugen
4 var rect = r.rect(10, 10, 80, 80);
5 //Mit Farbwert 00BF32 füllen, Randstärke 0
6 rect.attr({"fill":"#00BF32", "stroke-width": 0});
```

Listing 3.2: Erzeugung des Raphael-Canvas mit einem Rechteck

Die **Abbildung 3.6** zeigt das gezeichnete Rechteck. Es wird auf der Canvas-Oberfläche an den Koordinaten (0,0) beginnend jeweils 80 Pixel hoch und 80 Pixel breit erstellt. Seine Attribute füllen es mit einem Grünton, außerdem wird über die optionale Angabe *stroke-width* und dem Wert *0* der standardmäßig schwarze Rahmen entfernt.

Abbildung 3.6: Erzeugtes Rechteck mit RaphaëlJS

Gut zu erkennen in **Abbildung 3.7** ist der erzeugte DOM-Knoten im HTML-Dokument. Aus Platzgründen wird lediglich ein Ausschnitt gezeigt. Das *rect*-Tag, welches gleichzeitig ein Kindelement von *svg* ist, enthält alle zuvor gesetzten Attribute, wie die Höhe und die Breite des Rechtecks. Diese könnten mithilfe von Skriptsprachen manipuliert werden.

```
<html>
  <head>
  <body>
    <script type="text/javascript">
    <svg height="300" version="1.1" width="420" xmlr
        <desc>Created with Raphaël 2.1.0</desc>
        <defs>
        <rect x="10" y="10" width="80" height="80" r
    </svg>
  </body>
</html>
```

Abbildung 3.7: Für das Rechteck angelegter DOM-Knoten (blau markiert)

Ebenfalls lassen sich mit wenigen Zeilen Elemente animieren. Das Beispiel verdeutlicht, wie sich das Rechteck aus **Listing 3.2** auf der x-Achse um 135 Pixel verschieben lässt. Über das angelegte Objekt *rect* kann die Methode *.animate()* das Rechteck modifizieren oder verschieben.

```
1 //Rechteck um 135px nach rechts verschieben
2 //Skalierungsfaktor 1, Zeit 140ms
3 rect.animate({transform: "t135,0,s1"}, 140);
```

Listing 3.3: einfache Animation

Der Hauptgrund für die Entscheidung von RaphaëlJS liegt darin, dass es keine eigene Skriptsprache verwendet, sondern das normale JavaScript für die Manipulation der angelegten SVG-Objekte genutzt werden kann. Ebenso ist der Entwickler nicht auf die Programmierung eigener Mausevents angewiesen, da diese bereits integriert sind. Ein

zusätzlicher Grund, warum die Wahl auf RaphaëlJS fiel, ist die Existenz einer großen Entwicklergemeinde, die sich um Dmitry Baranovskiy, dem Entwickler von RaphaëlJS, gebildet hat. Die Mitglieder der Community[12] stellen viele Tutorials und Beispiele zur Verfügung. Es gibt viele nützliche Erweiterungen, wie beispielsweise eine Bibliothek zur Inline Texteditierung, bereitgestellt von dem Nutzer marmelab [mar13]. Die meisten dieser Zusatzmodule werden auf Github[13] bereitgestellt. Allerdings konnte auf Erweiterungen nahezu verzichtet werden, da sich mit RaphaëlJS fast jede Idee zügig umsetzen ließ. Somit konnte die Zeit für ein umfangreiches Testen anderer JavaScript Bibliotheken in andere Aufgaben investiert werden. Ein weiterer Grund ist die sehr gute Dokumentation, welche auf anschauliche Weise alle verfügbaren Funktionen, teilweise mit Beispielen, auflistet, was den praktischen Einsatz vereinfacht. Neben den bereits erwähnten Vorteilen gegenüber Paper.js und Processing.js unterstützt Raphaël als einzige der drei Kandidaten auch ältere Browser wie den Internet Explorer 7 oder 8, auch wenn dies eine untergeordnete Rolle spielt.

Das vorgestellte Grafik-Framework bietet Funktionen für den einfacheren Umgang mit Vektorgrafiken. Um die Grundlogik für die Realisierung eines Graphen zu implementieren, empfiehlt sich der Einsatz des nachfolgend beschriebenen Frameworks.

3.5 Dracula Graph Library

Die RaphaëlJS ist verantwortlich für die Darstellung grafischer Elemente. Diese Bibliothek beinhaltet keinerlei Funktionalität für die Erstellung von Graphen. Um diese Aufgabe zu erfüllen bedarf es eines weiteren Script-Moduls. Die Dracula Graph Library wird für diesen Zweck von dem Nutzer strathausen auf Github angeboten [Str12]. Damit der Graph auch sichtbar wird, greift sie auf RaphaëlJS zu und nutzt ihre Funktionen zur Visualisierung.

[12] Im Web gebräuchliche Bezeichnung für eine Gemeinschaft, die gleiche Interessen vertritt

[13] Setzt auf Git auf, einer verteilten Versionskontrolle von Softwareprojekten. Github ist ein webbasierter Anbieter von Software, die über Git verwaltet wird [Neu11].

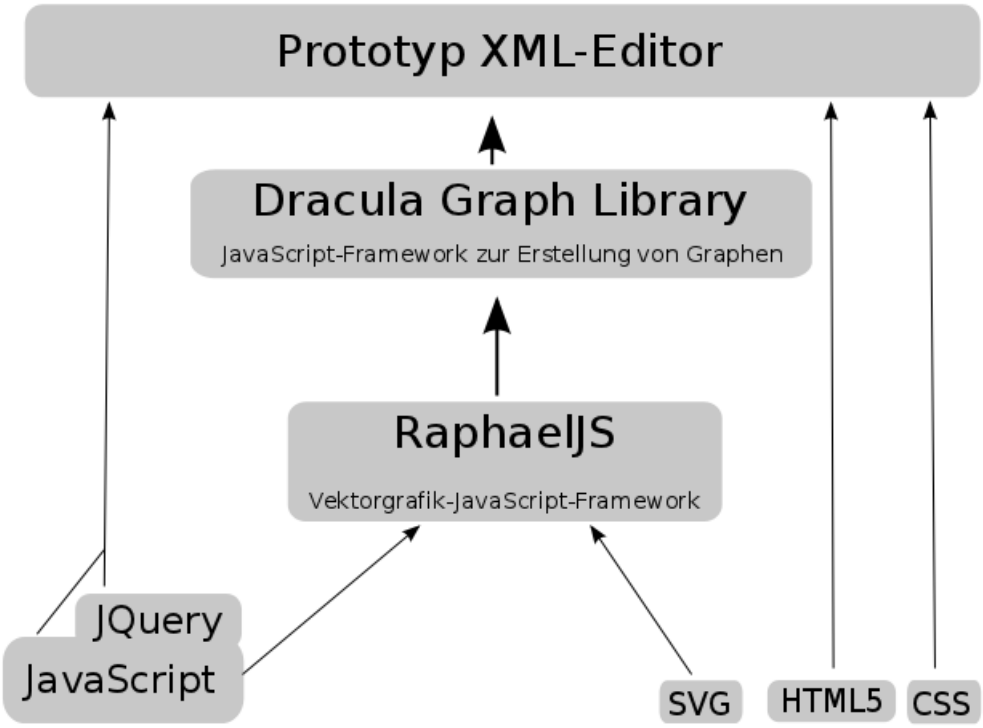


Abbildung 3.8: Genutzte Auszeichnungs- und Schriftsprachen, sowie Frameworks zur Realisierung des Editors

Die **Abbildung 3.9** weißt starke Ähnlichkeit mit der **Abbildung 3.5** des vorangegangenen Kapitels auf. Die Graphelemente wurden um Beschriftungen erweitert. Zusätzlich wurde der Button *redraw* eingefügt, der beim Klicken eine neue zufällige Anordnung der Knoten bewirkt.

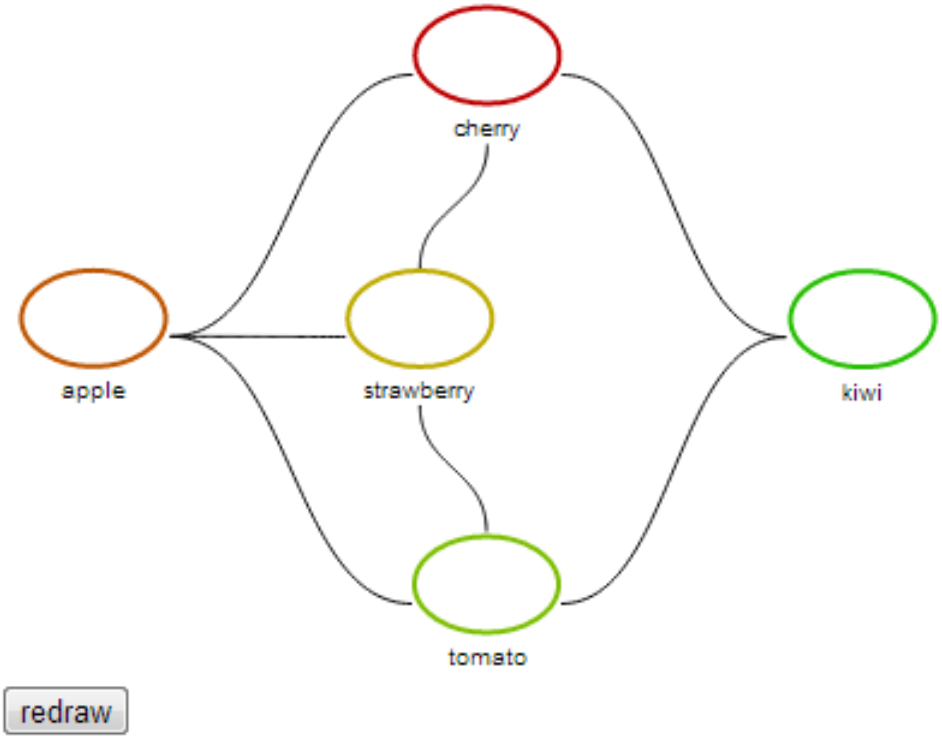


Abbildung 3.9: Beispiel auf graphdracula.net

Insgesamt werden drei JavaScript-Dateien in das Hauptprogramm eingebunden. Die Datei *dracula_graph.js* beinhaltet alle Funktionen für die Struktur des Graphen. Für den Entwickler sind zwei Funktionen dabei besonders wichtig. Mittels *addNode()* bekommt er die Möglichkeit neue Elemente dem Graphen hinzuzufügen. Jedes Element kann über das Hinzufügen einer ID in Form eines Zahlenwertes oder eines Strings identifiziert werden. Dieser Wert wird der *addNode()*-Funktion als Parameter übergeben. Neben diesem Parameter kann der Funktion noch ein zweiter Wert als Inhalt übermittelt werden. Ein Kommentar im Quellcode der Library enthält die Information, dass der *content* alle Werte enthalten kann, die vom Layout-Algorithmus oder der Funktion für die Graphdarstellung interpretiert werden kann. Die zweite essentielle Funktion *addEdge()* dient der Verbindung zwischen den Elementen. Es sind jeweils drei Parameter anzugeben. Die ersten beiden Werte übermitteln das Quell- und das Zielelement. Mithilfe des dritten Parameters kann die Art Verbindung angeben werden. Als *directed* wird dabei eine Verbindung bezeichnet, welche über einen Pfeil eine eindeutige Eltern-/Kindbeziehung repräsentiert.

```
1 //Graph-Object im Speicher anlegen
2 var g = new Graph();
3 //3 Knoten dem Graph hinzufügen
4 g.addNode(1, {label: "cherry"};
5 g.addNode(2, {label: "apple"};
6 g.addNode(3, {label: "strawberry"};
7 ...
```

```
8
9  //Knoten 2 und 3 als Kindelemente von Knoten 1
10 g.addEdge(1, 2, {directed: true});
11 g.addEdge(1, 3, {directed: true});
12 ...
```

Listing 3.4: Eine von mehreren Möglichkeiten den Graph aus **Abbildung 3.9** zu erstellen

Ebenfalls implementiert in der *dracula_graph.js* ist die Zeichnen-Funktion, die auf der RaphaëlJS basiert. Sie ermöglicht die eigentliche Visualisierung. Somit bleiben die geometrische Gestaltung der Elemente als Rechteck, Kreis oder Ellipse sowie die Farbgebung vollkommen dem Entwickler überlassen. Eine Eingrenzung seitens der Dracula Graph Library findet nicht statt, da diese für den Aufbau des Graphen zuständig ist. Neben der Gestaltung der Elemente ist außerdem eine Pfad zwischen diesen zu zeichnen. Die Datei *dracula_graffle.js* ist für die Gestaltung und den Verlauf der Pfade verantwortlich. Mithilfe des *path*-Elements aus der SVG-Spezifikation wurde ein Default-Pfad in Form einer kubischen Bezier-Kurve implementiert. In einem Array wird der Pfad nach SVG-Richtlinien zusammengesetzt. Er beginnt mit einem großen *M*. *M* steht für *moveto* und legt den Startpunkt des imaginären Stiftes fest. Die Koordinaten werden im vorhergehenden Schritt dynamisch ermittelt und nach dem *M* eingetragen. Nun folgt ein *C* für *cubic Bézier curve*, welches zusammen mit nachfolgenden Koordinaten einen kubischen Bezier-Kurvenverlauf vorgibt. Nachdem alle Kommas mittels *.join()* aus dem Array entfernt wurden, ist der Pfad fertig und kann gezeichnet werden. Der Verlauf des Pfades, sowie einfache Gestaltungsmöglichkeiten wie beispielsweise die Farbe können nach Belieben verändert werden. So wurde zur besseren Übersicht die Pfadfarbe von schwarz in ein helleres Grau angepasst.

3.6 Underscore.js

Die Underscore.js ist eine JavaScript Bibliothek und eine Sammlung an nützlichen Werkzeugen. Beispielsweise wurden Standard JavaScript-Funktionen für Arrays um einige Funktionen erweitert.
Zum Beispiel lässt sich mit der Funktion *findWhere* nach einem bestimmten Wertepaar in einem Array suchen. Das erste, welches den Suchkriterien entspricht, wird ausgegeben. Hingegen findet die Funktion *where* alle Wertepaare, die den Suchkriterien entsprechen.
Die Underscore.js wird in diesem Prototyp eine nicht unwesentliche Rolle einnehmen, da sie den Umgang mit Arrays erleichtert und dem Entwickler viel Arbeit abnimmt.
Darüber hinaus bietet dieses Framework auch viele Funktionen für die Arbeit mit Funktionen und Objekten an, was für dieses Programm allerdings weniger von Bedeutung ist.

Die Deklaration beginnt immer mit einem Unterstrich und einem Punkt.

```
1 //Array mit Wertepaaren anlegen
2 var animals   = [["katze", "maus"], ["hund", "wolf"],
3                                          ["mammut", "wolf"]];
4 //im Array animals nach "wolf" an Stelle 1 suchen
5 var value = _.findWhere(arr, [, "wolf"]);
```

Listing 3.5: Underscore.js

Bis auf Flash werden alle vorgestellten Websprachen eingesetzt. Der Hauptteil des Editors wird in JavaScript und JQuery geschrieben, da die Skriptsprachen für die Implementierung der Grundfunktionen verantwortlich sind.
Es folgt der Praktische Teil, der sich ausführlich der Realisierung des Projekts widmet.

4 Implementierung

Im ersten Abschnitt des Praktischen Teils geht es um die Organisation des Projektes. ePages setzt dafür das Programm *Jira* von der Software-Firma Atlassian ein. In einem kurzen Überblick sollen die Funktionen und Vorteile erklärt werden.
Der zweite Abschnitt wird sehr ausführlich auf die einzelnen Funktionen des Programms eingehen und erklären wie sie technisch realisiert werden.

4.1 Projektorganisation

4.1.1 Jira

Um das Projekt zu organisieren, Ziele zu setzen und Fortschritte kenntlich zu machen, wurde die Aufgabenmanagement-Verwaltung Jira eingesetzt. Sie ist webbasiert und funktioniert nach einem Ticketsystem. Jeder Benutzer erhält ein Konto mit Nutzername und Passwort. Dadurch wird die Oberfläche personalisiert. Somit sieht jeder Benutzer genau die Aufgaben, die ihm zugeteilt wurden. Wird beispielsweise ein Fehler in einem Softwaremodul gefunden, wird ein sogenanntes Ticket für den Benutzer erstellt und ihm damit eine Aufgabe (Task) zugeteilt. Um die Jira Oberfläche nicht permanent im Browser geöffnet zu haben, verfügt die Software über eine Email-Funktion. Wird einem Benutzer ein Ticket zugewiesen, erhält er automatisch eine Email mit der Ticketnummer.

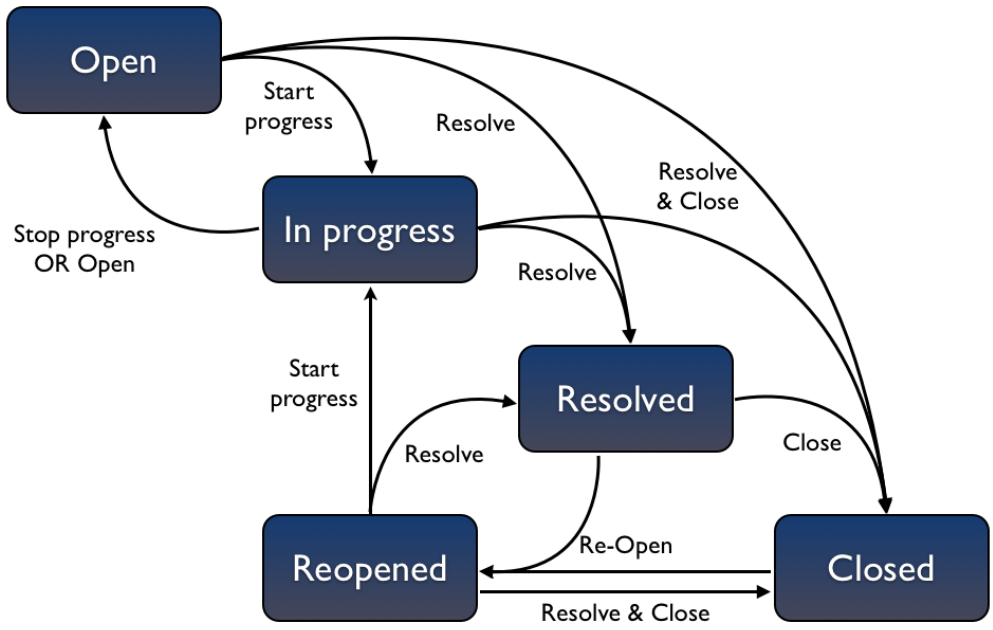


Abbildung 4.1: Jira Workflow (Quelle: atlassian.com)

Die **Abbildung 4.1** veranschaulicht den Workflow beim Einsatz von Jira. Nachdem ein Task geöffnet wurde kann er je nach Umfang und dafür benötigter Zeit mehrere Status annehmen. Während *In progress* daraufhin deutet, dass die Arbeit an der Aufgabe noch nicht abgeschlossen ist, kann die Arbeit im Status *Closed* bereits abgeschlossen sein. Wurde der Task nicht vollständig bearbeitet oder treten weiterhin Fehler auf, muss der Task erneut geöffnet werden und der Status ändert sich in *Reopened*. Einmal geöffnete Fälle werden somit nicht gelöscht. Jeder Mitarbeiter kann einen bereits als abgeschlossen gekennzeichneten Task jederzeit wieder öffnen und einem Benutzer zuweisen.

4.1.2 Sublime Texteditor

Zum Entwickeln von Webseiten mit HTML5, CSS und JavaScript reicht unter einfachsten Umständen ein Texteditor wie das in Windows integrierte Notepad. Da Notepad keine farbliche Hervorhebung der Syntax oder sonstige Hilfe beim Programmieren bietet, wird das Entwickeln nach kurzer Zeit unübersichtlich.
Es gibt zahlreiche, auf Webentwicklung spezialisierte Texteditoren am Markt, unter anderem der von vielen Mitarbeitern von ePages eingesetzte *Sublime Text*. Alle Funktionen dieses Editors aufzuzählen und zu erklären würde den Rahmen dieser Arbeit sprengen.
Ein Highlight ist die Paketverwaltung von Sublime Text. Das Modul *Package Control* wird einmal installiert. Von nun an kann der Benutzer über das Menü alle verfügbaren Pakete einsehen und bei Bedarf herunterladen. Das gewählte Paket wird sofort installiert und muss gegebenenfalls konfiguriert werden.
Ein wesentliches Unterscheidungsmerkmal im Verleich zu anderen Editoren ist das Anpassen der Einstellungen im JSON-Format. Am Beispiel (siehe **Abbildung 4.2**) des Einrichtens einer SFTP-Anbindung über ein spezielles Paket wird das Verändern von Einstellungen gezeigt.

```
{
    // The tab key will cycle through the settings when first created
    // Visit http://wbond.net/sublime_packages/sftp/settings for help

    // sftp, ftp or ftps
    "type": "sftp",

    "sync_down_on_open": true,

    "host": "ftp-www.hs-mittweida.de",
    "user": "username",
    "password": "password",
    "port": "22",

    "remote_path": "/example/path/",
    //"file_permissions": "664",
    //"dir_permissions": "775",

    //"extra_list_connections": 0,

    "connect_timeout": 30,
    //"keepalive": 120,
    //"ftp_passive_mode": true,
    //"ssh_key_file": "~/.ssh/id_rsa",
    //"sftp_flags": ["-F", "/path/to/ssh_config"],

    //"preserve_modification_times": false,
    //"remote_time_offset_in_hours": 0,
    //"remote_encoding": "utf-8",
    //"remote_locale": "C",
}
```

Abbildung 4.2: Sublime Texteditor - Einstellungen im JSON-Format

Die Projektdateien wurden vorzugsweise auf einem Server der Hochschule Mittweida gespeichert, um den direkten Aufruf über eine HTTP-Verbindung zu testen. Das SFTP-Paket erlaubt zur Optimierung des Workflows das direkte Speichern auf diesem Server mit (S)FTP-Unterstützung.

Sublime Text unterstützt den Entwickler neben dem erwähnten SFTP-Paket unter anderem mit Tabs wie man sie aus aktuellen Browsern kennt. Jede Datei wird in einem anderem Tab geöffnet. Verschiedene Ansichtseinstellungen, zum Beispiel eine Dateiliste am linken Rand, helfen beim Navigieren zwischen den Tabs.
Multiple Selections, zu deutsch etwa *Mehrfach-Auswahl*, ist beim Programmieren ein gern gesehenes Feature. Sublime Text kann einen bestimmten Variablennamen an jeder Stelle im Quellcode finden und jeden einzelnen gleichzeitig markieren. Durch Eintippen eines neuen Wertes, kann die Variable an jeder Stelle sofort geändert werden.

Der Sublime Editor bietet standardmäßig mehrere Farbthemen an. Es kann nach persönlichem Empfinden aus zahlreichen Themen gewählt werden. Zusätzlich kann eine bestimmte Programmiersprache ausgewählt werden, um die Syntax hervorzuheben. Unter anderem lassen sich C, C++, HTML, CSS, JavaScript oder PHP auswählen.

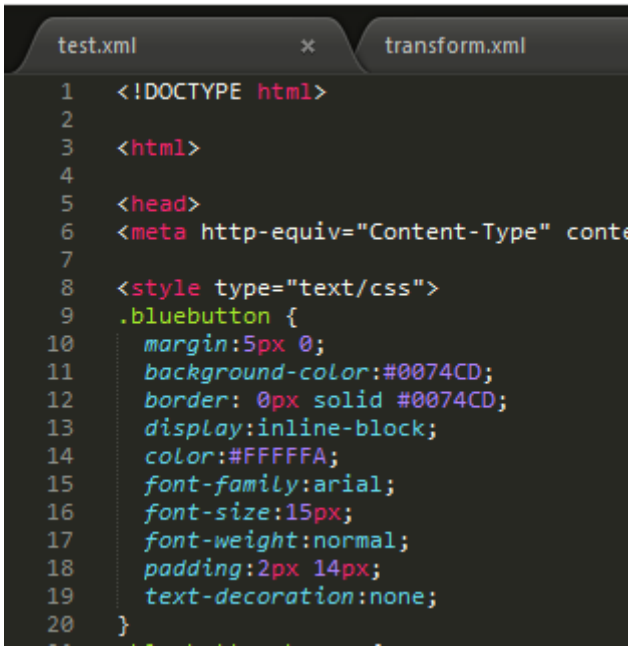


Abbildung 4.3: Standard Farbschema - Syntaxhervorhebung CSS

Als plattformunabhängiger Editor kann er sowohl auf Windows, Mac OS oder Linux installiert werden.

4.2 Umsetzung der Grundfunktionen

4.2.1 Einlesen der XML

Das Einlesen der transform.xml geschieht über einen AJAX-Request. JQuery bietet dafür eine verkürzte Schreibweise. In einer zweiten JavaScript-Datei wurde dazu ein Objekt mit essentiellen Funktionen erstellt, unter anderem die AJAX-Funktion. Der große Vorteil dieser Funktion ist der asynchrone Datentransfer. Sobald ein Benutzer den Editor aufruft, wird im Hintergrund bereits eine *transform.xml* eingelesen. Als Methode wird in den meisten Fällen *get* benutzt. Da die einzulesende XML aber immer den gleichen Dateinamen besitzt, bietet es sich an die Methode auf *post* zu ändern, da diese niemals auf den Browsercache zugreift. Wurde der Request erfolgreich ausgeführt, werden die Informationen aus der XML als *data* an die *success*-Funktion übergeben. Mittels JQuery wird *data* selektiert und zwei Methoden nacheinander ausgeführt. Die *find()*-Methode findet über die Angabe * alle in der transform.xml vorkommenden Elemente. Die JQuery Methode *toArray()* erstellt daraus ein Standard Array. Dieses Array wird der Variablen *receivedNodesFromXML* zugewiesen. Sie enthält nun jedes XML-Element als ein JavaScript-Objekt, auf das mit allen JavaScript-Methoden zur DOM-Manipulation zugegriffen werden kann. Sobald der Nutzer den LoadXML-Button drückt wird diese Variable geladen und für die nachfolgenden Prozesse ausgelesen.

Alternativ besteht die Möglichkeit eine XML-Datei von einem lokalen System zu laden. Nachdem Auswählen der gewünschten Datei wird diese auf den Server hochgeladen und anschließend über ein *FileReader*[14]-Objekt eingelesen. Der FileReader übergibt die XML-Daten an einen *Parser*[15], mit dessen Hilfe alle XML-Elemente ebenfalls der Variable *receivedNodesFromXML* zugewiesen werden.

```
ajax:    function(){
     var result = null;
     var scriptUrl = "transform.xml";
     $.ajax({
        url: scriptUrl,
        type: 'get',
        dataType: 'xml',
        async: true,
        success: function(data) {
            result = $(data).find("*").toArray();
            save.receivedNodesFromXML = result;
        }
     });
  },
```

Listing 4.1: Einsatz von AJAX

Wenn alle Dracula Graph Dateien eingebunden sind, müssen zunächst die Elemente aus der XML dem Graph hinzugefügt werden. Dafür wird mit dem *new*-Operator ein Object des Graphen im Speicher angelegt. Anschließend muss die *addNode()*-Funktion des Graph-Objects ausgeführt werden. Diese verlangt zwei Parameter. Eine Identifikationsnummer bzw. einen String und eine beliebige Variable als *content*. Absolut notwendig ist die Angabe einer angepassten Renderfunktion im *content*-Bereich, da ansonsten die Standardrenderfunktion benutzt wird. Des Weiteren müssen Koordinaten übergeben werden, um die Elemente günstig zu positionieren. Wurde eine XML erzeugt, speichert das Programm automatisch alle Positionen als Attribute in der XML. Die *addNode()*-Funktion des Graph-Objects wird durch die Funktion *addNodes* aufgerufen. Ihr wird das Graph-Object und das Array mit allen XML-Elementen übergeben. Als erstes wird in einer Schleife geprüft, ob die eingelesenen Elemente bereits Positionen als Attribute aufweisen. Gilt die Bedingung als erfüllt, werden die Koordinaten übernommen und als x- und y-Wert gespeichert. Befinden sich noch keine Positionskoordinaten in der Datei, so wird die Anzahl aller Elternelemente, beginnend beim Wurzelelement, ermittelt, über die das aktuelle Element erreicht werden kann. Ein Knoten, der das Wurzelelement als Elternelement hat, wird um die halbe Breite eines Knotens eingerückt. Tiefer verschachtelte Elemente müssen relativ zum Wurzelknoten öfter eingerückt werden. Die in **Listing 4.2** zu sehende while-Schleife überprüft, ob das Elternelement des aktuellen Knotens mit dem Wurzelelement übereinstimmt. Ist das nicht der Fall, wird das

[14] FileReader - Bietet Webanwendungen eine Möglichkeit zum Lesen des Inhalts von Dateien. [dev13]
[15] Parser - Wandelt ein XML-Dokument in ein XML-DOM-Objekt um, dass anschließend mit Methoden zur DOM-Manipulation bearbeitet werden kann. [w3sb]

nächst höhere Elternelement gesucht. Dies wird solange fortgesetzt, bis das Wurzelelement mit dem aktuellen Elternelement übereinstimmt. Nach jedem Schleifendurchlauf wird die Anzahl der Elternelemente um 1 erhöht. Sie wird anschließend mit einem fixen x-Ausgangswert multipliziert. Je höher die ermittelte Anzahl der Elternknoten, desto weiter wird ein Knoten auf der x-Achse eingerückt. Die **Abbildung 4.4** demonstriert wie die Elemente beim ersten Laden einer XML-Datei eingerückt werden. Ein bemerkenswerter Vorteil dieser Methode zeigt sich, wenn eine missgestaltete XML-Datei, z.B. ohne Einrückungen oder mit überdimensionalen Leerzeichen, geladen wird. In so einem Fall werden alle ungünstigen Formatierungen ignoriert und die XML-Datei wird deutlich übersichtlicher dargestellt.
Die y-Koordinate wird mit dem Schleifenindex hochgezählt. Um beim späteren Editieren der Attribute nicht in Konflikt mit den Koordinaten zu geraten, werden sie nach dem Hinzufügen zu dem Graph unverzüglich herausgelöscht. Möchte man statt der Tag-Namen die Attribute sofort sehen, können auch diese mit der *content*-Variable an das Graph-Objekt übergeben werden.

```
1  function addNodes(savekn, g) {
2              $.each(savekn, function(i, element){
3               //Wurzelelement festlegen
4               if(i === 0){
5                save.root = element.nodeName;
6               }
7               if(!$(element).attr("x")){
8                //Elternelement vom aktuellen Knoten ermitteln
9                var parentNode = element.parentNode;
10               //Zähler für die Anzahl der Elternelemente
11                 auf 1 setzen
12               var parentNodeCounter = 1;
13               while(parentNode.nodeName != save.root
14                         && element.nodeName != save.root){
15             //Elternelement neu festlegen
16                 parentNode = parentNode.parentNode;
17                 //Zähler hochzählen
18                 parentNodeCounter++;
19               }
20               if(i === 0){
21                 var xCoo = 80;
22                 var yCoo = 20+30*(i+1);
23               }else{
24                 var xCoo = 80+70*parentNodeCounter;
25                 var yCoo = 20+30*(i+1);
26               }
27            }
28              else{
29               var xCoo = parseInt($(element).attr("x"))+65;
30               var yCoo = parseInt($(element).attr("y"));
31              }
32                 g.addNode(...);
33
```

```
34                $(element).removeAttr("x");
35                $(element).removeAttr("y");
36              });
37            };
```

Listing 4.2: Eingebettete addNode()-Funktion

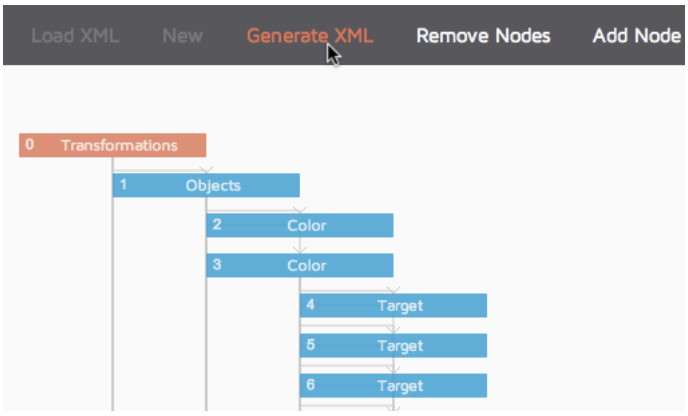


Abbildung 4.4: Automatisches Einrücken beim ersten Laden einer XML-Datei

In einem zweiten Schritt müssen nun alle Kindelemente den richtigen Elternelementen zugeordnet werden. Für diese Aufgabe bietet das Graph-Object eine Funktion *addEdge()*. Die *addEdge()*-Funktion gibt ein Array zurück, welches die Verknüpfungen in der Form *[Elternelement, Kindelement]* speichert. Wie bereits bei der *addNode()*-Funktion wird diese in einer zusätzlichen Funktion *addEdges()* aufgerufen. In einer Schleife werden nun die Indizes zu jedem Knoten als Attribut gespeichert. Dieser Schritt ist wichtig, da anschließend jedem Knoten sein Elternelement zugeordnet wird. Wenn mehrere Kindelemente ein gemeinsames Elternelement besitzen, so muss in einer Auflistung der Elternelemente dieses bestimmte Elternelement immer den gleichen Index bekommen. Würde man diesen Schritt auslassen, würde jedes Elternelement den Schleifenindex erhalten. Der Schleifenindex wird hochgezählt, was zur Folge hätte, dass ein bestimmter Elternknoten bei mehrmaligem Vorkommen immer einen anderen Index bekäme und somit nicht mehr als derselbe Elternknoten zu identifizieren wäre. Die unten zu sehende **Abbildung 4.5** zeigt rot eingerahmt die Indizes der Elternelemente. Beispielsweise sind alle *Target*-Elemente Kindelemente von *Color*. Daraus resultierend wird jedem *Target* das Element *Color* zugeordnet. Da es sich bei jedem *Color* um das selbe Elternelement handelt, muss es den gleichen Index, in dem Fall die *3* erhalten.

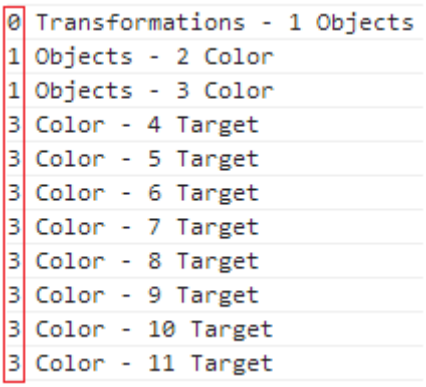


Abbildung 4.5: Verknüpfungen in einer XML - Ansicht Google Chrome Konsole

Nachdem alle Verknüpfungen dem Graphen hinzugefügt wurden, können die Indizes wieder aus den Attributen entfernt werden, da sie für das spätere Bearbeiten nicht mehr erforderlich sind.

```
function addEdges(savekn, g){
      $.each(savekn, function(i, element){
        //Attribut "id" anlegen
        $(element).attr("id", i);
        //Elternelement zu jedem Knoten finden
        var pEl = element.parentNode;
        if(i != 0){
            g.addEdge(parseInt($(pEl).attr("id")), i,
                                              {directed: true});
        };
      });

      //Entfernen der Indizes
      $.each(savekn, function(l, el){
        $(el).removeAttr("id");
      });
};
```

Listing 4.3: Eingebettete addEdge()-Funktion

Der Graph ist nun vollständig und kann gezeichnet zu werden.

4.2.2 Visualisierung des Graphen

Nachdem das Graph-Objekt vollständig mit den Knotennamen und Verknüpfungen befüllt wurde, kann der Graph gezeichnet werden. Dazu wird die Funktion *redraw()* aufgerufen, die wiederum nacheinander zwei Funktionen für das Rendering anstößt. Im ersten Schritt muss das Layout für den Graph festgelegt werden. Aus den zur Verfügung stehenden Layouts wird das Modell *Ordered* ausgewählt. Dieses erlaubt im Ge-

gensatz zum *Spring*-Layout feste Koordinaten anzugeben, um eine zufällige Verteilung der Elemente zu verhindern. Die zweite Funktion führt das eigentliche Rendern aus.

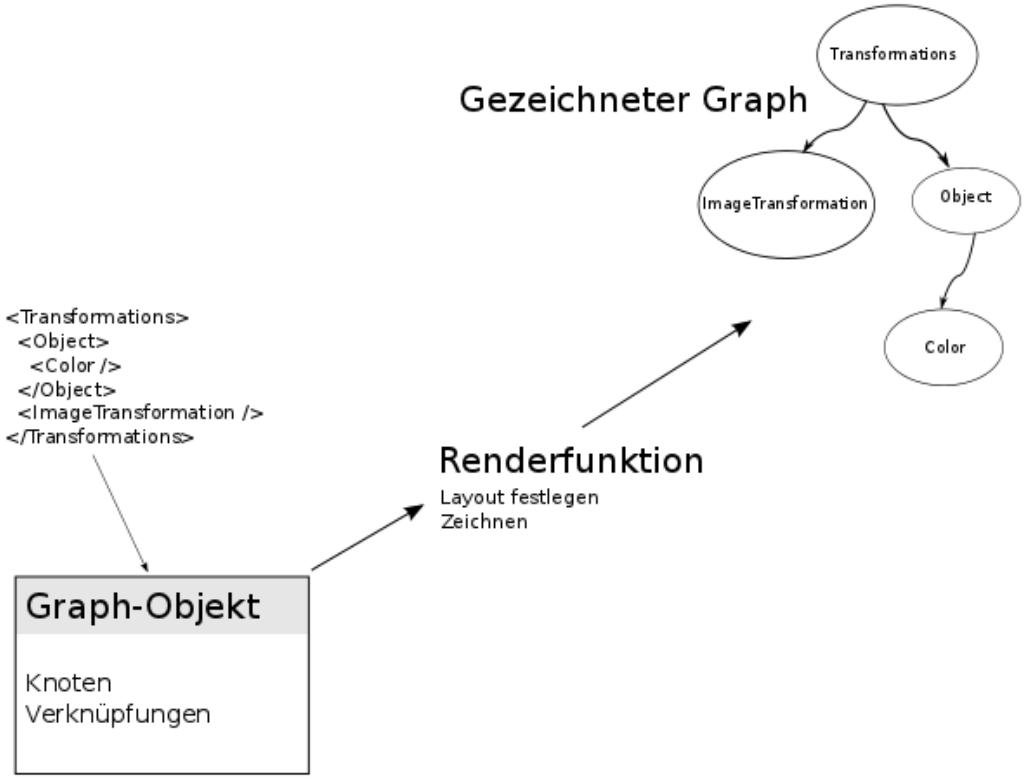


Abbildung 4.6: Ablaufschema zur Darstellung des Graphen

Um SVG-Elemente mit RaphaëlJS auf die Browseroberfläche zu zeichnen, muss zunächst eine Art Leinwand durch das Raphael-Objekt erzeugt werden. Erst danach können die einzelnen Grafiken dargestellt werden. Dazu wird am Anfang des Dokumentes ein *div*-Container erzeugt, welcher dem Renderer aus der Graph Library zusammen mit dem Graph-Object und zwei Parametern für die Größe des *div*-Elementes übergeben werden. Die bereits im vorhergehenden Kapitel erwähnte *render()*-Funktion wird nun für jeden einzelnen Knoten aufgerufen und erstellt ein neues *Set*. Diesem *Set* werden ein grünes Rechteck, das Label des XML-Elementes und der jeweilige Index übergeben. Der Verlauf der Pfade zwischen den Knoten wird ebenfalls durch die Dracula Graph Library gesteuert. Ein Algorithmus in der Datei *dracula_graffle.js* orientiert sich an der Position des Knotenelementes auf dem Canvas und setzt den Pfad an die für die Darstellung günstigste Himmelsrichtung des jeweiligen Knotenrechteckes. Die Zusammensetzung der Pfade nach SVG-Richtlinien, wie im Theorieteil erklärt und in **Listing 4.4** veranschaulicht, geschieht ebenfalls in der *dracula_graffle.js*.
Standardmäßig wird die Verbindung als Bezierkurve (siehe **Abbildung 4.7**) gezeichnet. Durch eine einfache Abänderung, wie in **Listing 4.5** kommentiert, wird die Verbindung aus Geraden dargestellt. Die SVG-Spezifikation bietet viel Spielraum, um die Pfade für

eine gute Übersicht anzupassen.

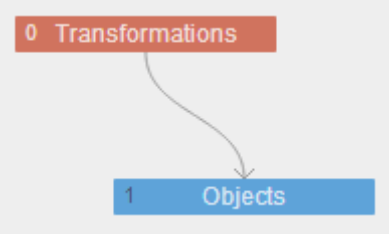


Abbildung 4.7: Verbindung zwischen zwei Knoten als Bezierkurve

```
var path = [ "M" + x1.toFixed(3),
                   y1.toFixed(3),
             "C" + x2, y2, x3, y3,
                   x4.toFixed(3),
                   y4.toFixed(3) ].join(", ");
```

Listing 4.4: Ausschnitt aus der dracula_graffle.js - Erstellung eines Bezier-Pfades

```
var path = [ "M" + x1.toFixed(3),
                   y1.toFixed(3),
             //Änderung des Wertes C zu L (line to)
             "L" + x2, y2, x3, y3,
                   x4.toFixed(3),
                   y4.toFixed(3) ].join(", ");
```

Listing 4.5: Ausschnitt aus der dracula_graffle.js - Erstellung eines Pfades aus Geraden

```
<path fill="none" stroke="#999999" d="M80,68L80,89L130,89L130,110M125,105L130,110L135,105"
;"></path>
```

Abbildung 4.8: Aus **Listing 4.5** generierter SVG-Pfad als DOM-Element.

Die **Abbildung 4.8** zeigt einen Ausschnitt aus der Entwicklerkonsole von Google Chrome. Das *path*-Element gehört zur SVG-Spezifikation. Der Wert seines Attributs *d* wurde aus dem voran gegangenen Quellcodebeispiel dynamisch generiert und bestimmt den Verlauf des Pfades bzw. der Verbindung zwischen zwei Knoten (siehe **Abbildung 4.9**).

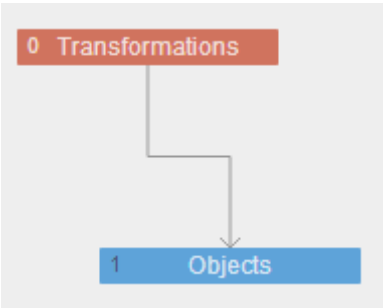


Abbildung 4.9: Verbindungspfad zwischen zwei Knoten, zusammengesetzt aus Geraden.

4.2.3 Verschieben von XML-Knoten

Die Dracula Library bietet unter anderem eine Funktion für das Verschieben von Elementen. Dabei macht es sich die *translate()*-Funktion aus der RaphaëlJS zunutze. Dieser Funktion werden Koordinaten übergeben, an die das Element verschoben werden soll. Die in Dracula implementierte Funktion *dragMove()* ermittelt bei gedrückt gehaltener linker Maustaste die aktuellen Koordinaten und die Abmessungen des jeweiligen *Elements*. Beim Bewegen der Maus werden die sich aktualisierenden Mauskoordinaten abzüglich der Quellkoordinaten der *translate()*-Funktion (siehe **Listing 4.6**) übergeben und das Element in Echtzeit an die neue Position verschoben.

```
1
2  .
3  //clientX (x-Mauskoordinate), clientY (y-Mauskoordinate)
4  //dx (ursprüngliche x-Position), dy (ursprüngliche y-Position)
5  .
6  this.set.translate(clientX - Math.round(dx),
7                                  clientY - Math.round(dy));
8  .
9  .
```

Listing 4.6: Kernfunktion für das Verschieben

Um die aktuelle Position für zukünftiges Aufrufen festzuhalten, werden die Koordinaten in einem Array gespeichert und beim *mouseup*-Event neu geschrieben. Das Positionsarray setzt sich folgendermaßen zusammen:

[Index Element$_1$*, [*x_1*,* y_1*], Index Element*$_2$*, [*x_2*,* y_2*], ..., Index Element*$_n$*, [*x_n*,* y_n*]]*

```
[ 0, [ 88, 62 ], 1, [ 136, 142 ] ]
```

Abbildung 4.10: Positionsarray für die Objekte aus **Abbildung 4.9**, Ansicht im Firebug

4.2.4 Hinzufügen und Löschen von DOM-Elementen

Soll ein DOM-Element hinzugefügt werden, so geschieht dies über den Button *Add Node*. Es wird als erstes das Canvas geleert, da die Renderfunktion für den Knoten neu ausgeführt wird, was zur Folge hätte, dass ein neues leeres Canvas unter dem alten entsteht, worauf der neue Knoten platziert wird. Anschließend wird das Graph-Object neu angelegt und das DOM-Element mit dem höchsten Index ermittelt. Nachdem dieser um *1* erhöht wurde, kann der neue Index in einer Variablen gespeichert und der *addNode()*-Funktion als ID für den neuen Knoten übergeben werden. Über ein *input*-Feld in

der Kontrollleiste, bekommt der Nutzer die Möglichkeit einen Tagnamen vor der Erzeugung des neuen Knotens hinzuzufügen. Über eine *if*-Abfrage wird ermittelt, ob dieses Feld einen Wert enthält, anderenfalls wird eine Defaultbezeichnung vergeben. Da das neue Element einem Elternelement zugeordnet werden muss, wird ebenfalls eine neue Verknüpfung in das Verknüpfungsarray aufgenommen. Nachdem der neue Knoten dem Graph-Objekt hinzugefügt wurde, müssen alle bisherigen Elemente, ihre Verknüpfungen und Positionen ebenfalls wieder dem Graph-Objekt hinzugefügt werden. Nachdem abschließend die Renderfunktion wieder ausgeführt wird, erscheinen alle bisherigen und der neue Knoten auf dem neuen Canvas.

4.2.5 Editieren der Tags und Attribute

Die RaphaëlJS bietet für das Zeichnen eines Textstrings die Methode *text()*. Wird die *renderMainFunction()*-Funktion ausgeführt, wird automatisch jedem Rechteck, dass einen Knoten repräsentiert, ein Label hinzugefügt.
Durch die umfangreichen Transformationsmöglichkeiten, die Raphael bietet, ist für das Editieren von Text eine zusätzliche JavaScript Library notwendig, die den zu bearbeitenden Text auf Transformationen prüft und gegebenenfalls entfernt. Für diesen Zweck wurde eigens von dem Nutzer *marmelab* über GitHub die *Raphael.InlineTextEditing*-Library erstellt. Sobald ein Textstring angeklickt und editiert werden soll, wird über dem aktuellen SVG-Element ein Raphael Container platziert und geprüft, ob eine Transformation vorliegt. Ist dies der Fall, wird sie entfernt, um die Originaldimensionen des Textelementes zu ermitteln und zu speichern. Anschließend wird die Transformation auf das Element wieder angewandt und selbiges versteckt. Im nächsten Schritt wird ein CSS-Objekt mit den Originalabmessungen des Textfeldes erstellt und die Schriftart des Textes ermittelt. Daran folgend erstellt die Library ein *input*-Element in oben genanntem Container und wendet die gespeicherten CSS-Informationen darauf an. Abschließend wird der Fokus auf das *input*-Feld gesetzt, um neuen Text eingeben zu können.

Neben dem Bearbeiten-Modus für die Label braucht der Nutzer ebenfalls eine Möglichkeit Attribute zu setzen, zu entfernen und zu editieren. Die Realisierung dieser Funktion erfolgt prototypisch über ein Menü, welches sich individuell für jeden Knoten öffnen lässt und unter anderem die Verwaltung der Attribute ermöglicht.

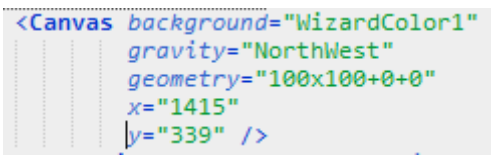


Abbildung 4.11: Attribute des XML-Knotens *<Canvas>* - Bildausschnitt Texteditor

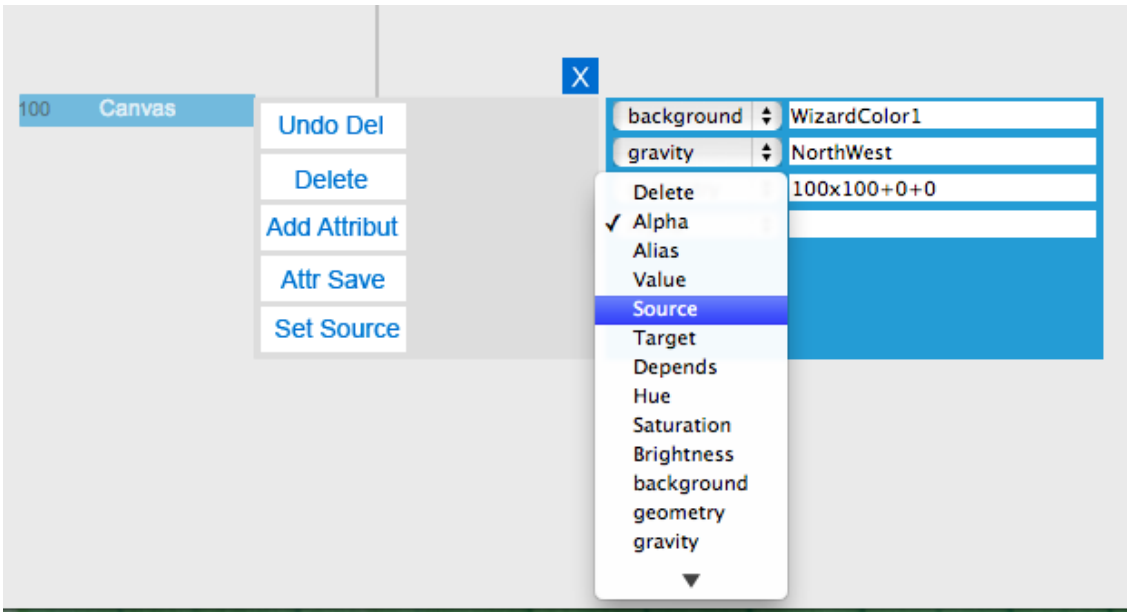


Abbildung 4.12: Alle eingelesenen Attribute aus **Abbildung 4.11**

Beim Laden des Attribut-Menüs wird die jeweilige ID des Elementes ermittelt und darüber die jeweiligen Attribute aus einem Array ausgelesen. Gut zu erkennen ist in **Abbildung 4.12**, dass die Positionskoordinaten *x* und *y* nicht geladen wurden. Wie anfangs erwähnt, dienen sie nur dem Speichern der Position auf der Zeichenfläche und werden schließlich aus den Eigenschaften der Knoten entfernt.
Anschließend erstellt der Prototyp für jedes Attribut ein *div*-Element, in welchem mit dem HTML5 *<select>*-Tag ein Dropdown-Menü und ein Textfeld kreiert werden. In dieses *select*-Element werden alle verfügbaren Attribute-NodeNames[16] geladen. Es wird nun geprüft, welcher NodeName mit dem des Knotens übereinstimmt. Dieser wird vorselektiert. Anschließend wird in das generierte Textfeld der Wert des Attributes geschrieben.
Die für jedes *select*- und *input*-Feld erstellten *div*-Elemente werden mit einem Index versehen, da auch die Möglichkeit bestehen muss diese individuell anzusprechen, um Attribute zu löschen. Um das Design nicht mit unnötigen Elementen zu überfrachten, wurde für die Löschen-Funktion auf einen extra Button verzichtet. Das erste Element in der Attributliste bekommt den Wert *"Delete"*. Jedes *select*-Feld bekommt als Attribut *onchange* übergeben. Der Wert dieses Attributes ist eine Funktion, die jeweiliges *select*-Feld und das dazugehörige *input*-Element löscht, sobald *"Delete"* angewählt wird. Sobald alle Änderungen vorgenommen wurden, wird mit dem Button *Attr Save* das Speichern der Attribute ausgelöst. Die Speicherung geschieht unabhängig davon, ob etwas geändert wurde. Das bedeutet, schaut ein Benutzer nur welche Attribute vorhanden sind, so werden diese beim Schließen des Dialoges trotzdem gespeichert. Auf diese Weise kann eine zusätzliche Abfrage gespart werden. Das Auslesen der *select*- und

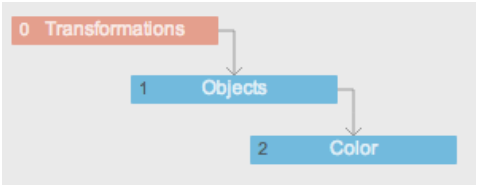
[16] Ein NodeName ist der vergebene Name für das Attribut. In **Abbildung 4.12** sind das *background*, *gravity* und *geometry*. Der dazugehörige Wert, z.B. *WizardColor1*, ist der NodeValue.

input-Felder geschieht in einer Schleife, die ermittelt wieviele Felder mit Werten existieren. Es werden nur Werte aus Feldern gespeichert, die keine leeren Strings enthalten. Zum Beispiel wird das Attribut *Alpha* aus **Abbildung 4.12** nicht gespeichert, da ihm kein Wert zugewiesen wurde.

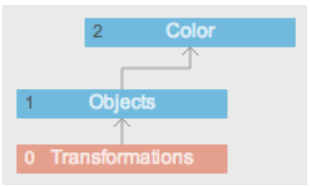
4.2.6 Neuverknüpfung

Wie bereits erwähnt besteht eine XML-Datei aus Elementen, die wiederum Unterelemente, sogenannte Kindelemente, enthalten können.

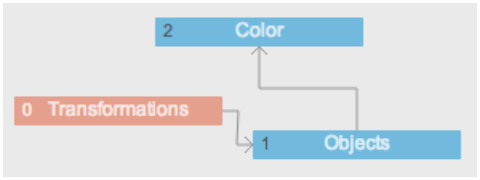
(a) Ansicht Texteditor



(b) Ansicht XML-Editor



(c) Ansicht 2



(d) Ansicht 3

Abbildung 4.13: Inhalt einer XML-Datei unterschiedlich grafisch dargestellt

Die Beziehung der Elemente zueinander erkennt man daran, dass sich ein Kindelement zwischen dem Start- und End-Tag seines Elternelementes befindet. Mithilfe der für den Prototyp genutzten Bibliotheken ist der Entwickler in der Lage diese Beziehung grafisch darzustellen. Der Vorteil besteht darin auf die schließenden Tags zu verzichten und eine Abhängigkeit durch einen gerichteten Pfeil zu repräsentieren. Die Verbindungen liegen in einem Array als Paar *[Quellknoten, Zielknoten]* vor. In **Abbildung 4.14** zu sehen, sind das *[Transformations, Objects]* und *[Objects, Color]*.

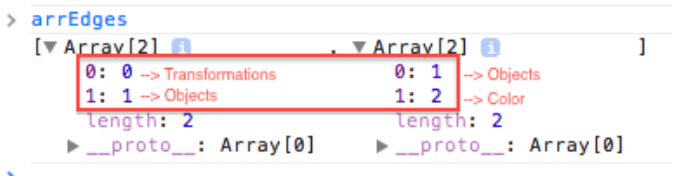


Abbildung 4.14: Rot eingerahmt sind die Verknüpfungen über den Index gespeichert.

Jeder DOM-Knoten in einem Dokument, dass nach der XML-Spezifikation aufgebaut ist, kann nur einmal als Ziel- bzw. Kindelement vorkommen. Aufgrund dieser Tatsache wird die Neuverknüpfung dadurch erreicht, dass zunächst über ein *input*-Feld ein neuer Quellknoten festgelegt wird. Anschließend wird mit einem Doppelklick der Zielknoten markiert, dem das neue Elternelement zugewiesen werden soll. Dabei wird im Hintergrund in dem Array, das alle Verbindungspaare enthält, nach dem Paar gesucht, welches an zweiter Stelle den markierten Knoten enthält, da dieses Paar nur einmal vorkommen kann. Wurde das Paar erfolgreich gefunden, wird die erste Stelle durch den neuen Elternknoten ersetzt.

```
1  set.dblclick(function(){
2     //Knoten als Zielelement festlegen
3     var tar = node.id;
4     //Prüfen ob Verbindung schon existiert - ggf. neue anlegen
5     if(!node.edges[0]){
6       arrEdges.push([parseInt($("#source").val()), node.id]);
7       ...
8     //im Verknüpfungsarray nach dem Zielelement suchen
9     }else{
10      for(var i=0; i<arrEdges.length; i++){
11      var where = _.findWhere(arrEdges, [, tar]);
12      var ind = _.indexOf(arrEdges, where);
13      };
14      //neues Elternelement aus Input-Feld durch altes ersetzen
15      where.splice(0, 1, parseInt($("#source").val()));
16      ...
17     };
18   });
```

Listing 4.7: Verknüpfungsalgorithmus

Sichtbar wird die neue Verbindung erst nach dem das Canvas neu gezeichnet wurde.

4.2.7 Generieren der XML

Das Generieren und die Ausgabe einer XML-konformen Struktur wird bei diesem Projekt über die Zusammensetzung eines Strings gelöst. Dazu werden nacheinander mehrere Schritte ausgeführt.

In der ersten Phase werden alle Elemente ermittelt, die Kindelemente besitzen. Dabei bleibt eine Verschachtelung, sowie Nur-Kindelemente unberücksichtigt. Es wird auch noch nicht geprüft, ob bestimmte Elemente gleichzeitig als Eltern- und Kindelement vorkommen. Noch in derselben Phase werden die Tagnamen und Attribute dieser Elemente als String mit spitzen Klammern zusammengesetzt und nacheinander in einem Array gespeichert. Mittels der *.join()*-Methode werden die Kommas entfernt und es ergibt sich ein XML-konformer DOM-Knoten. Da der String im letzten Schritt als ein Binary

Large Object der *createObjectURL*-Funktion übergeben wird, müssen zusätzlich nach jedem schließenden Tag Zeilenumbrüche eingefügt werden. Zusatzinformationen wie x- und y-Koordinaten werden ebenfalls eingefügt, um beim Wiederöffnen der XML alle Elemente an den Positionen zu haben, wo sie sich zuletzt befanden.

```
1  .
2  .
3  //<Tagname
4  cP.push("<"+arrLabel[indi-1]+"")
5     //Attribute für dieses Tag ermitteln
6     var specificAtt = save.allAttr[0][indi-1];
7     //Attribute auslesen - <Tagname Attribute
8     for(var z=0; z<specificAtt.length; z++){
9       cP.push(" "+specificAtt[z][0]+"=\""+specificAtt[z][1]+"\"");
10    }
11    //Position einfügen - <Tagname Attribute Position>
12    cP.push(this.positions(index1[0])+">\r\n");
13    //Kommas entfernen
14    readyForOutput.push(cP.join(""));
15    //End-Tag anfügen - <Tagname Attribute Position></Tagname>
16    readyForOutput.push("</"+arrLabel[indi-1]+">\r\n");
17 .
18 .
```

Listing 4.8: Auszug Quelltext - Zusammensetzen eines XML-Tag

```
1  [<Trans></Trans><Obj></Obj><Col Alias="WizardColor1"></Col>]
```

Listing 4.9: Alle Elterntags in einem Array gespeichert (Tagnamen abgekürzt)

In Phase 2 werden alle Elternelemente darauf hin untersucht, ob sie untereinander selbst voneinander abhängen. Beispielsweise ist *<Obj>* ein Kindelement von *<Trans>* und *<Col>* wiederum ein Kindelement von *<Obj>*. Würde nun *Col* zwischen den *Obj*-Tags platziert werden, dann dürften die *Obj*-Tags nicht mehr verschoben werden, da die *Col*-Elemente sich dann nicht mehr zwischen diesen befinden. Aus diesem Grund muss *Obj* zunächst zwischen seine Eltern-Tags *Trans* verschoben werden. Die **Abbildung 4.15** demonstriert die Reihenfolge. Dieser Vorgang kann nur einmal geschehen, da jedes Element nur ein Elternelement besitzen kann. Anschließend kann *Col* zwischen *Obj* platziert werden. Um diese Problematik zu lösen wurde ein Zuordnungsalgorithmus entworfen.

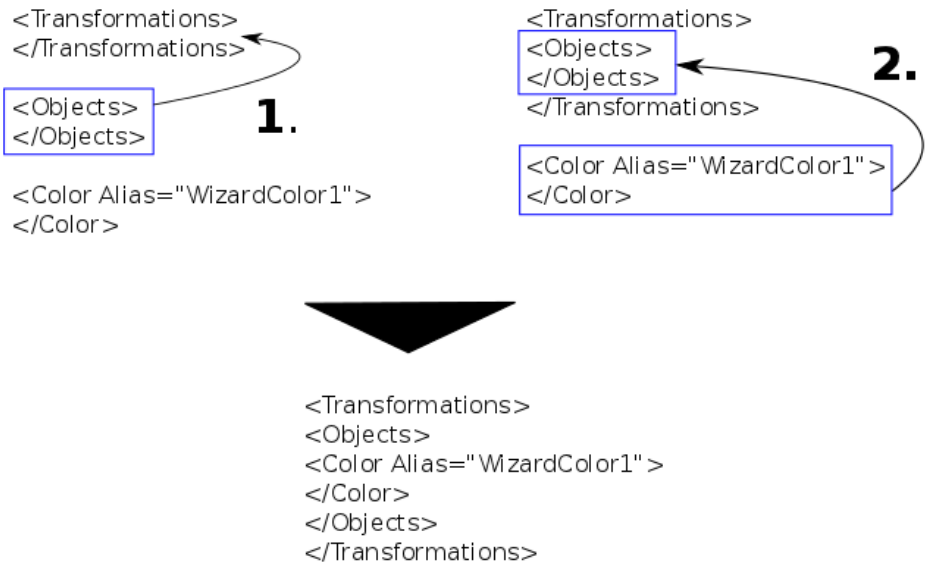


Abbildung 4.15: Elternknoten werden nacheinander neu platziert

Dieser prüft vor dem Verschieben die Indizes der Elternelemente im Array und passt gegebenenfalls die Verschiebungsreihenfolge an. Der Algorithmus startet solange von vorn bis sicher gestellt ist, dass jedes Element, nachdem es einmal verschoben wird, auch wirklich am richtigen Platz ist. Dem Zuordnungsalgorithmus wird eine besondere Rolle zuteil, da es in manchen Verknüpfungsvariationen passieren kann, dass die Knoten in einer ungünstigen Reihenfolge nacheinander verschoben werden und die korrekte Beziehung zu ihren Elternelementen nicht mehr gegeben wäre.
Das zu verschiebende Element wird anschließend zwischengespeichert, herausgelöscht und zwischen den Tags des ihm zugeordneten Elternelements eingefügt.

Anschließend müssen alle Kindelemente gefunden werden, die selbst kein Elternelement sind. Sie werden genauso zusammengebaut wie die Elternelemente, erhalten jedoch kein schließendes Tag. Stattdessen wird ein Leerzeichen, gefolgt von einem Slash, eingefügt.
Nachdem mit der *.join()*-Methode wieder alle Kommas entfernt wurden, ist der String fertig und kann dem *Blob()*-Object[17] übergeben werden. Es erzeugt anschließend einen URL. Durch Angabe des MIME-Typs *application/xml* kann die Datei direkt mit der Endung *.xml* heruntergeladen werden.

[17] Binary Large Object - repräsentiert ein Datei-ähnliches großes Objekt aus unveränderlichen binären Daten, etwa einem Bild oder einer Audiodatei. Es wird häufig zur Speicherung von Daten in Datenbanken benutzt [Chr13].

4.2.8 Design

Für diesen Editor spielt das Design eine untergeordnete Rolle, weshalb es der letzte Punkt dieses Kapitels sein soll. Der Prototyp wurde 2013 entwickelt und orientiert sich an den Trends für Webdesign des gleichen Jahres. Das sogenannte “Flat-Design“ ist nicht nur im Web-Bereich anzutreffen, auch aktuelle Betriebssysteme auf den verschiedensten Endgeräten sind in diesem Design gestaltet. Dabei stellt der Nutzer schnell fest, dass Flat-Design auf Einfachheit und leichte Bedienung ausgelegt ist. Aufwendige Farbläufe, Schatten, abgerundete Ecken oder auffällige Umrandungen an Buttons oder Menüleisten wurden bewusst weggelassen.

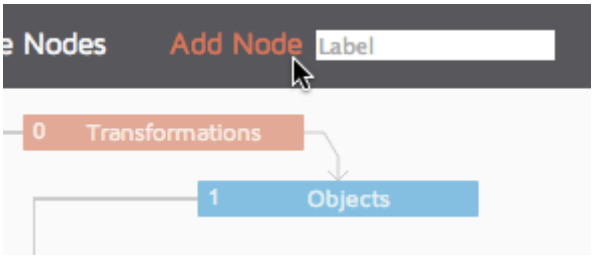


Abbildung 4.16: Fixe Menüleiste und Gestaltung der XML-Elemente

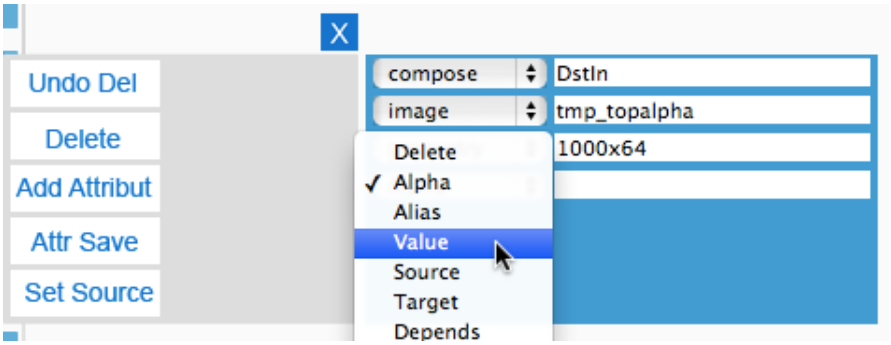


Abbildung 4.17: Design für die Verwaltung der Attribute

Der Fokus richtet sich auf das Wesentliche. Diese “Regeln“ wurden auch in diesem Projekt befolgt. Die Menüleiste zieht sich über die gesamte Breite und ist fest am oberen Rand der Browserfläche fixiert. Erreicht wurde dieses Verhalten mit den CSS Regeln *width:100%* und *position:fixed*. Scrollt der Nutzer nach unten, bleibt die Leiste weiterhin sichtbar. Alle Buttons sind transparent, sodass nur die Beschriftungen zu sehen sind. Umrandungen von Textfeldern oder Auswahlboxen wurden ebenfalls entfernt. Das Design setzt sich konsequent mit den Elementen für die XML-Knoten fort. Diese wurden farblich an die Menüleiste angeglichen.
Die Verbindungspfeile zwischen diesen Elementen werden in einem dezenten Grauton gezeichnet, um nicht zu aufdringlich zu wirken. Das Erscheinungsbild der Knoten und des Auswahlmenüs wurden in dieser Version nicht mit CSS umgesetzt, da Raphael auch das Einbinden von Bildern unterstützt, um beispielsweise Buttons zu er-

stellen. Allerdings ist diese Lösung nicht optimal, da beim Arbeiten mit Rastergrafiken mehr Arbeitsaufwand notwendig ist als beim Anpassen von CSS-Regeln. Letztere verlangen kein geeignetes Bildbearbeitungsprogramm, um Veränderungen vorzunehmen. Die Konsolen aktueller Browser bieten sehr gute Möglichkeiten, neue CSS-Regeln zu erstellen und sofort sichtbar auf ein Element anzuwenden.
Als experimentelle Erweiterung verfügt der Editor über ein Responsive Design. Das bedeutet, dass die Oberfläche auf die Bildschirmabmessungen des verwendeten Gerätes reagiert. Bei einer Bildschirm- bzw. Browserfensterbreite von unter 1200px, verlagert sich die Menüleiste an den linken Rand, sodass alle Bedienelemente weiterhin sichtbar bleiben.

5 Softwaretest

Je nach Größe des Projekts können Softwaretests sehr aufwendig werden und gesonderte Textumgebungen und Personal erfordern. Das Kapitel Softwaretest beschreibt die Vorgehensweise beim Testen des Prototyps. Es gliedert sich in zwei grobe Unterkapitel, die zwei grundverschiedene Heransgehensweisen für Softwaretests veranschaulichen.

Die Software überschreitet zum jetzigen Entwicklungsstand nicht die Grenze, die einen automatisierten Test erfordert. Alle zu erwartenden Reaktionen des Programms wurden manuell getestet. Als große Hilfe erwiesen sich die Entwicklungswerkzeuge des Google Chrome Browsers und das Zusatzprogramm Firebug für den Firefox Browser. Um beispielsweise Werte für HTML-Elemente adressieren zu können, ist es hilfreich zu wissen, wo diese sich in den Eigenschaften des Elements befinden. Über die Entwicklerkonsole kann ähnlich wie in einem Dateiexplorer genau zu den gewünschten Werten navigiert werden. Die benötigten Elemente können in einem Array gespeichert werden und beim Aufrufen in der Konsole kann der Entwickler überprüfen, ob das Array die gewünschten Elemente zurückgibt. Durch Benutzeraktionen werden oft Änderungen an Arrays vorgenommen. Es werden Werte hinzugefügt, verändert oder entfernt. Dabei spielt es keine unwesentliche Rolle an welcher Stelle dies in dem Array geschieht. Der JavaScript-Befehl *console.log(...)* bietet die Möglichkeit solche Arrays automatisch nach jeder Änderung in der Konsole ausgeben zu lassen, sodass die Korrektheit des Inhalts eines Arrays unmittelbar analysiert werden kann. Fehlerhafte Inhalte in Arrays haben in der Regel negative Auswirkungen auf die Programmfunktionalität und ziehen eine direkte Bearbeitung des Quellcodes nach sich. Das ist ein Beispiel für das erste beschriebene Testmodell White-Box-Test.

5.1 White-Box-Test

Mit zunehmendem Umfang des Programms bedurfte es umso gründlicherer Tests. Nachdem Hinzufügen einer neuen Funktion bzw. dem Ändern bereits vorhandenen Codes, wurden alle Funktionen erneut geprüft. Da viele Funktionen zusammenhängen und programmübergreifend agieren, war es nicht ausreichend, lediglich neu integrierte Module zu testen, sondern die gesamte Software. Dadurch sollte ausgeschlossen werden, dass sich Fehler negativ auf andere Quellcodebereiche auswirken.
Als Entwickler sein eigenes Programm zu testen bedeutet häufig, dass Fehler übersehen werden, weil dieser genau weiß, welche Benutzeraktionen zu vermeiden sind. Instinktiv wird er in den meisten Fällen “positiv“ testen. Allerdings kennt er das Programm am besten und wird deshalb der erste sein, der sich auf die Suche nach Fehlern begibt. Damit sollte rechtzeitig begonnen werden, um nicht den Überblick zu verlieren,

wenn das Projekt größere Ausmaße annimmt.
Vom Entwickler selbst ausgeführte Tests gehören zu den “White-Box-Tests“ oder auch “Unittests“ genannt [SBS11]. Unittests werden mit Kenntnissen über den Quelltext ausgeführt. Dabei wird geprüft, ob die vom Entwickler erstellten Softwarekomponenten so ausgeführt werden wie gewünscht [agi]. In größeren Projekten werden automatisierte Tests erstellt. Per Mausklick werden nacheinander alle zu testenden Softwarekomponenten geprüft. Der Hintergrund ist unter anderem der, dass neu geschriebene Module in ein bestehendes System integriert werden müssen ohne das bisherige zu beschädigen [agi]. So kann es passieren, dass neue Komponenten Auswirkungen auf unterschiedliche Teile der Software haben, obwohl sie auf den ersten Blick nicht zusammenhängen. Dieser Fall könnte eintreten wenn bestehende Teile eines Programms verändert werden, andere Komponenten des selben Programms aber auch darauf zugreifen. Daher ist es ratsam, wie anfangs bereits erwähnt, nach einer Änderung mehrere Funktionalitäten einer Software zu testen.
Als praxisnahes Beispiel dient die Funktion *deleteAttribute*. Sie überprüft, ob das Feld *Delete* (siehe **Abbildung 5.2a**) angewählt wird. Trifft dies zu, wird das *select*-Feld gelöscht. Um weiterhin eine korrekte Funktionalität zu gewährleisten, müssen nun alle *div*-Elemente, die die *select*- und *input*-Felder enthalten neu durchnummeriert werden. Dazu ermittelt die Funktion nach jedem Löschen die Kindelemente des umschließenden *div id=“dropdown“* und gibt ihnen einen neuen Index. Die folgenden Abbildungen veranschaulichen die Herangehensweise beim Überprüfen dieser Funktion auf ihre korrekte Ausführung.

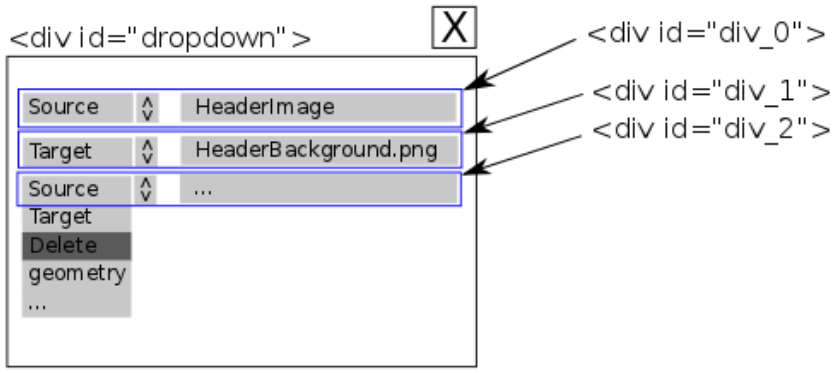
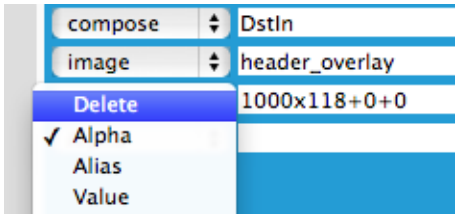


Abbildung 5.1: Zu prüfende div-Elemente

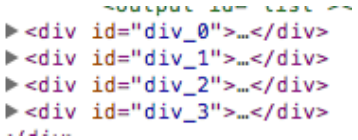
```
deleteAttribute: function (id){
                  var drop = $("#dropdown");
                  if($("#div_"+id).children()[0].value == "Delete"){
                   $("#div_"+id).remove();
                    for(var i=0; i<drop.children().length; i++){
                     if(drop.children()[i] != undefined){
                      drop.children()[i].id = "div_"+i;
                          }
                        }
                     }
              }
```

Listing 5.1: Diese Funktion löscht nach der Auswahl von *Delete* das zugehörige Attribut.

Ein Blick in das Attribut-Menü zeigt vier *input*- und vier dazugehörige *select*-Felder. Diese werden teilweise verdeckt. Zum Vergleich sieht man in der Entwicklerkonsole vier *div*-Elemente. Sie enthalten die erwähnten HTML-Knoten. Ebenfalls zu erkennen ist der Index, der ihnen nach einem Unterstrich zugeteilt wurde.

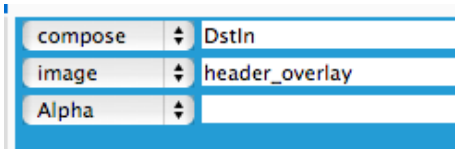


(a) 4 Elemente vorhanden.

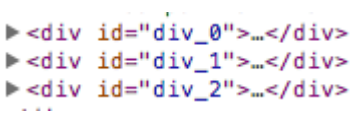


(b) 4 erzeugte div-Elemente.

Nachdem das Feld *Delete* angewählt wurde, sind das entsprechende *input*- und *select*-Feld entfernt worden. Die Anforderung an die Funktion war, alle *div*-Elemente nach dem Löschen neu zu indizieren. Schaut man sich das Ergebnis in der Konsole an, stellt man fest, dass wieder ein *div*-Element mit dem Index *2* vorhanden ist, obwohl das Element mit dem gleichen Index gelöscht wurde. Würde die Funktion nicht richtig ausgeführt, würde es die *2* nicht mehr geben und nach dem *div_1* das *div_3* folgen. Eine Ausführung der Funktion wie beabsichtigt wäre dann nicht mehr gewährleistet. Das ist nicht der Fall, die Funktion arbeitet somit korrekt.



(a) 1 Elemente wurde entfernt.



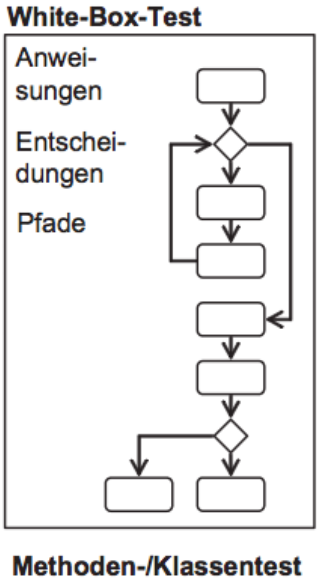
(b) Korrekte Neuindizierung

Eine ähnliche Herangehensweise wie soeben beschrieben, konnte auf sämtliche Funktionen des Prototyps angewendet werden. Die Entwicklerwerkzeuge der Browser Google Chrome und Firefox bieten herausragende Fähigkeiten, um die Programmstruktur

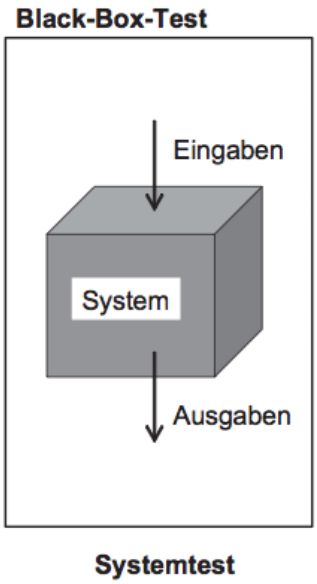
detailliert zu analysieren und die Aufmerksamkeit auf Fehler und deren Korrektur zu richten. Die Möglichkeit HTML-, CSS- oder JavaScript-Code direkt in der Konsole anzupassen und auszuführen bedeutet für den Entwickler die Änderungen unverzüglich sehen zu können. Dieses Vorgehen spart Zeit, da Zwischenschritte wie das Hochladen von Quelldateien und das Neuladen der Webseite vorerst außen vor bleiben.

5.2 Black-Box-Test

Um das angesprochene Übersehen von Fehlern zu vermeiden ist es sinnvoll andere Personen, ohne Quelltextkenntnisse, einzubeziehen, um mehrere Testergebnisse zu erhalten. Zusätzlich könnte der Personenkreis durch potentielle Endanwender erweitert werden, die das System nur flüchtig bis gar nicht kennen. Das Bedienverhalten von Endanwendern unterscheidet sich meist völlig von dem Verhalten der ausgebildeten Tester einer Softwarefirma [SBS11]. Durch ihre Herangehensweise können deutlich mehr Fehler aufgespürt und korrigiert werden. Diese Art der Tests wird als “Black-Box-Test“ oder “Systemtest“ bezeichnet. Die **Abbildung 5.4b** verdeutlicht das Vorgehen bei einem Systemtest. Auf Nutzereingaben, etwa Mausklickereignisse oder Texteingaben, reagiert das Programm und gibt etwas zurück. Entspricht die Reaktion den Erwartungen, verlief der Test erfolgreich.
Das Ziel eines Systemtests besteht in diesem Fall hauptsächlich darin das Vertrauen des Benutzers in den Prototyp zu gewinnen. Die Funktionen des Prototyps sollten sich mit den gestellten Anforderungen weitestgehend decken [SBS11]. Beispielsweise würde eine fehlerhafte Integration der Speichern-Funktion für eine XML-Datei das Programm unbrauchbar machen. Dabei ist es für den Benutzer unwesentlich, ob die verantwortliche Softwarekomponente effizienten Programmierrichtlinien entspricht. Für den Endanwender ist entscheidend, dass er XML-Dateien auch speichern kann.



(a) Unittest [Kle11]



(b) Systemtest [Kle11]

5.3 Browserübergreifendes Testen

Da ein möglicher Einsatz des Programms nur intern in Frage kommt, konnte auf umfangreiche browserübergreifende Tests verzichtet werden.
Einige Funktionen wurden dennoch, neben dem Google Chrome, auch im Firefox, Opera und Safari getestet. Zum Zeitpunkt der Tests setzte lediglich der Chrome-Browser in der Version 29 alle Funktionen nach den gewünschten Erwartungen um. Der Firefox-Browser war in der Version 22 vorhanden und brachte deutliche Leistungsprobleme beim Verschieben von XML-Elementen mit sich. Ein flüssiges Verschieben war nur mit einer geringen Anzahl an Knoten möglich. Getestet wurde auf einem Dell Latitude E6510 Notebook von 2010 mit Windows 7.
Ein weiterer Test auf einem MacBook Air (Mid 2013) und Mac OSX in der Version 10.8.5 kurz vor Abschluss der Arbeit erbrachte ein gegenteiliges Ergebnis. Es gab keinerlei Anzeichen von Leistungsengpässen mit dem Firefox in der Version 24.
Hier lässt sich festhalten, dass unterschiedlich alte Hardware, die Wahl des Betriebssystems und die Aktualität der Browser wesentlichen Einfluss auf die Testergebnisse haben. Je größer der Kreis der Endanwender wird, umso mehr unterschiedliche Systeme müssen durch die Software unterstützt werden. Onlineshop-Software muss in der Regel auf jedem erdenklichen Endgerät lauffähig sein, da hierbei in erster Linie wirtschaftliche Interessen im Vordergrund stehen. Gründliche Tests durch professionelles Personal der Qualitätssicherung ist unverzichtbar und ausschlaggebend für den Erfolg der Software.

6 Zusammenfassung

Das letzte Kapitel befasst sich mit Herausforderungen während der Umsetzung und Themen, die bislang nicht gelöst wurden. Außerdem gibt es einen Ausblick für künftige Projekte, die den Editor als Basis nutzen könnten.

6.1 Herausforderungen bei der Umsetzung

Der Studiengang Multimediatechnik vermittelt einen guten Einblick in aktuelle Webstandards. Aufgrund der Vielfalt an Webtechnologien ist es kaum möglich alle mit der gleichen Priorität zu lehren. Dieser Tatsache geschuldet, fiel der praktische Umgang mit JavaScript und JQuery eher gering aus. Erst während der Programmierung des Prototyps nahm die Erfahrung zu, was in einem immer besser werdenden Programmierstil resultierte. Als die Fertigstellung mehr in den Fokus rückte, konnten erlangte Fähigkeiten dazu genutzt werden erhebliche Codeeinkürzungen vorzunehmen ohne an Funktionalität einbüßen zu müssen.

Neben den fehlenden JavaScript-Kenntnissen hinderten vor allem in der Anfangsphase fehlende Vorlagen an einem zügigen Einstieg in die Problematik. Selbst nach intensiver Websuche konnten keine webbasierten XML-Editoren ausfindig gemacht werden, an denen eine Orientierung sinnvoll war. Die Benutzeroberfläche war entweder sehr rudimentär oder die Software zum Zeitpunkt des Tests nicht fertiggestellt, sodass unausgereifte Funktionen einen erfolgreichen Test verhinderten. Aus einer anderen Perspektive betrachtet fiel der Lerneffekt dadurch höher aus, was als sehr positiv zu bewerten ist.

Ein spezielleres Problem soll in dieser Arbeit nicht unerwähnt bleiben. Die programmtechnische Umsetzung wie die Elemente verknüpft werden orientiert sich an der Dracula Graph Library. Mit steigender Anzahl an Knoten nehmen auch die Möglichkeiten diese neu zu verknüpfen stark zu. Um beim Speichern der XML-Datei alle Tags in korrekte Beziehung zu setzen, war die Implementierung eines eigenen Algorithmus notwendig. Dieser wies anfangs viele Schwachstellen auf, wodurch die Tags nicht immer an die korrekte Stelle in der XML geschrieben wurden. Nach vielen Tests gelang es schließlich den Algorithmus einsetzbar zu optimieren.

6.2 Offen gebliebene Themen

6.2.1 User Story

Auf das Anlegen einer User Story wurde zu Beginn des Projekts verzichtet. Erst kurz vor Fertigstellung des praktischen Teils wurde Bekanntschaft mit einer möglichen künftigen Anwenderin gemacht. Dabei stellte sich heraus, dass ihr Workflow[18] beim Bearbeiten einer transform.xml sich noch nicht optimal mit dem Prototyp umsetzen ließ. Um vorteilhafte Anpassungen vornehmen zu können, bietet sich die Erstellung einer User Story an.
Eine User Story legt die Anforderungen an ein Softwareprojekt aus der Perspektive eines Benutzers fest. Die **Abbildung 6.1** zeigt eine Story-Karte, wie sie für einen hier benötigten XML-Editor aussehen könnte.

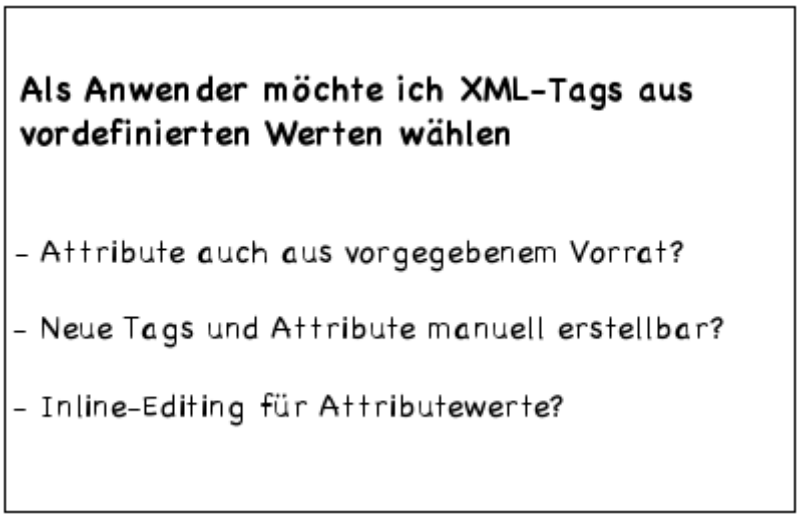


Abbildung 6.1: Projektbezogene User Story

In den meisten Fällen ist die User Story absichtlich unvollständig formuliert und bietet viel Freiraum für Veränderungen, da sich ein Kunde erst relativ spät für ein Konzept entscheidet. Erst nachdem er eine neue Software kennengelernt und selbst ausprobiert hat, kann er konkrete Vorstellungen für Teile eines Systems festlegen [Wir11]. Der letzte Satz verdeutlicht, dass Softwareprototypen schon im frühen Entwicklungsstadium dem Benutzer vorgestellt werden sollten, um weitere Funktionen und designspezifische Umsetzungen besser an seine Vorstellungen anlehnen zu können. Insbesondere dem “Selbst-Ausprobieren“ durch den Kunden ist eine große Bedeutung beizumessen.

[18] In der Fachsprache verwendeter Begriff für Arbeitsablauf

6.2.2 Neuladen der Zeichenfläche verhindern

Eines der offenen gebliebenen Fragen ist eine Lösung für das Hinzufügen von Knoten und die Neuverbindung ohne vorheriges Löschen der Zeichenfläche.
Für diese Problematik muss erwähnt werden, dass die Struktur und der Aufbauprozess des Graphen im Hintergrund und für den Benutzer nicht sichtbar abläuft. Die implementierte RaphaëlJS-Renderfunktion wird erst aufgerufen, nachdem der Graph vollständig ist. Ein nachträgliches Hinzufügen von Elementen erfordert zwei Schritte. Im ersten Schritt muss ein neues Element erstellt und dem Graph-Objekt, dass alle Knoten enthält, hinzugefügt werden. Im zweiten Schritt wird der Knoten auf die Zeichenfläche gerendert und ein Objekt im DOM angelegt.
Wenn Elemente neu verknüpft werden sollen, sind diese Schritte ebenfalls erforderlich. Die Renderfunktion muss neu aufgerufen werden. Diese zeichnet aber bei jedem Aufruf eine neue Zeichenfläche. Ist eine solche bereits vorhanden, wird eine zweite unter dieser positioniert. Alle neu hinzugefügten Elemente würden sich dann auf der zweiten und somit nicht auf der selben Fläche befinden.

Ein möglicher Ansatz dieses Problem zu lösen, wäre das Umschreiben der Dracula Graph Library. Die Renderfunktion erwartet bei jedem Aufruf, etwa nach dem Ändern einer Verknüpfung oder dem Hinzufügen eines Knotens, einen Parameter mit der ID eines div-Elements (siehe **Listing 6.1**). Das übergebene div stellt die virtuelle Leinwand dar.

```
1  //wird von der Renderfunktion als Zeichenfläche verwendet
2  <div id="canvas"></div>
3  .
4  .
5  //div mit der id "canvas" wird als Parameter übergeben
6  var renderer = new Graph.Renderer.Raphael(
7                  "canvas", g, save.width, save.height="5000");
8  renderer.draw();
9  .
10 .
```

Listing 6.1: Das div-Element wird dem Renderer als Parameter übergeben.

Die Dracula Graph Library müsste nun dahingehend abgeändert werden, dass die Funktion für das Zeichnen der Objekte nicht bei jedem Aufruf einen Parameter für die Erzeugung einer Zeichenfläche verlangt. Zu überprüfen wäre, ob bereits eine Zeichenfläche existiert. Ist die Bedingung erfüllt, wird auf die vorhandene Fläche gerendert. Die Funktion nur ein einziges Mal, nämlich beim ersten Laden einer XML-Datei zu starten, ist nicht möglich, da das Graph-Objekt ebenfalls als Parameter übergeben werden muss, um seine Elemente auf die Leinwand zu rendern. Das ist der Grund, weshalb es erforderlich ist, diese Funktion nach jeder Änderung am Graphen neu anzustoßen.

6.2.3 XML-Schema

Häufig ist es nicht erforderlich in welcher Reihenfolge die Knoten in der XML aufgeführt werden. Wichtig ist lediglich die richtige Beziehung zwischen Eltern- und Kindsknoten. Allerdings kann es unter Umständen auch erforderlich sein die Elemente in einer bestimmten Reihenfolge abzulegen, weil ein Interpreter dies zur korrekten Verarbeitung verlangt. In diesem Fall müsste ein zusätzlicher Mechanismus zur Erstellung von XML-Schema integriert werden. Da jeder Nutzer die repräsentierenden Elemente nach Belieben verschieben kann, sind zusätzlich Überlegungen zur Festlegung der Reihenfolge durch den Nutzer anzustellen. Eine Überarbeitung des Algorithmus zum Generieren der XML muss dann ebenfalls einer Anpassung unterzogen werden.
Das bedeutet im Umkehrschluss entweder die Integration von XML-Schemata zur allgemeinen Festlegung der Reihenfolge oder eine Anpassung für die Erstellung eines ganz bestimmten Typs von XML-Dateien, deren Struktur sich nach Vorgaben für gezielte Software-Prozesse richtet. In diesem Fall wäre der Einsatz von Templates sehr sinnvoll, die für den jeweiligen Typ einer XML nur bestimmte Erstellungsmuster zulassen und gleichzeitig den Aufwand einer Bearbeitung minimieren könnten.

6.3 Ergebnis und Ausblick

Das Ergebnis dieser Arbeit ist ein prototypischer XML-Editor, der nahezu alle Funktionen zum Modifizieren vorhandener XML-Dateien bereitstellt. Darüber hinaus ist es möglich komplette XML-Dateien beginnend beim Root-Element[19] aufzubauen. Die Positionierung der verschiebbaren Elemente bleibt vollkommen dem Anwender überlassen. Die XML-Struktur, also die Beziehung zwischen Eltern- und Kindknoten, wird nicht beeinflusst. Anhand dieser Tatsache benötigt der Anwender theoretisch keine Kenntnisse über die korrekte Notation von XML-Daten. Für den Fall, dass schlecht geformte XML-Dateien geladen werden, stellt der Prototyp einen Algorithmus bereit, der unabhängig von der Formatierung eine einheitliche Darstellung mit eingerückten Kindelementen zeigt, um die Verknüpfungen zu erkennen.

[19] Wurzelelement - Das in der Hierarchie zuerst vorkommende Element. Es besitzt selbst kein Elternelement.

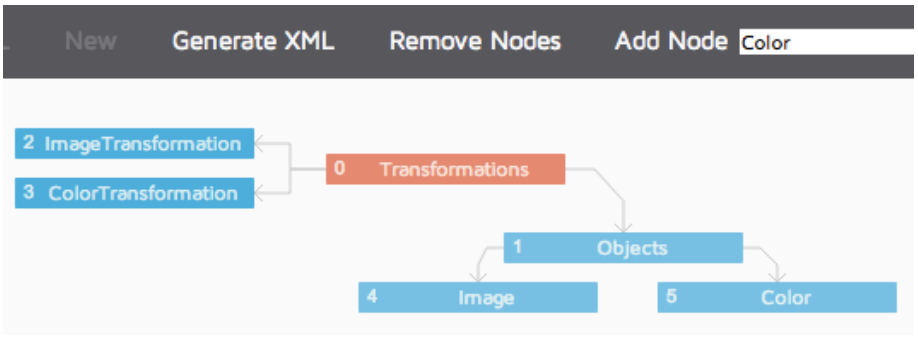


Abbildung 6.2: Neu erstellte XML-Struktur mit dem Prototyp

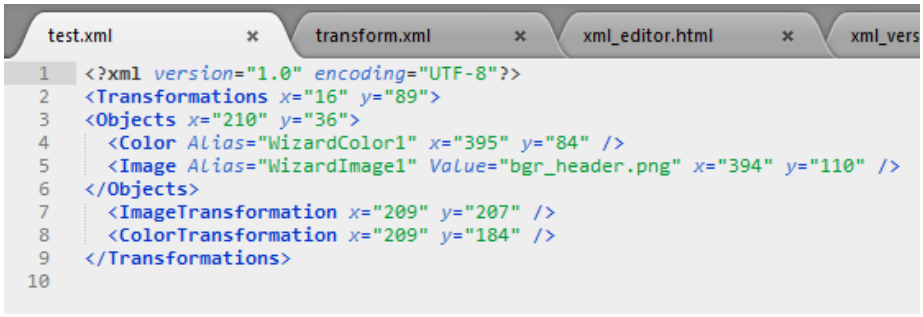


Abbildung 6.3: Generierte XML aus **Abbildung 6.2** in einem Texteditor

In dieser Version wird der Workflow beim Erstellen bestimmter XML-Strukturen nicht berücksichtigt. Da XML-Dateien sehr oft für bestimmte Prozesse eingesetzt werden, wie in diesem Fall Transformationen von Farben und Bildern, ist die nachträgliche Anpassung (siehe **User Story**), etwa das Anbieten von Templates, eine sinnvolle Erweiterung. Beispielsweise könnten nur speziell beschriftete Knoten auswählbar sein, deren Attribute je nach Bedeutung des Elementes automatisch hinzugefügt werden. Die Option der Inline-Texteditierung könnte gesperrt werden, was die Anzahl der Fehler durch den Nutzer verringern würde.
Ebenso wäre eine Unterstützung für das JSON-Format denkbar. Es ist für einen Menschen einfacher zu lesen und könnte XML in einigen Jahren verdrängen [Per12]. Die Gründe dafür sind vielfältig. Unter anderem ist JSON besser standardisiert und strukturiert als XML. Es ist aufgebaut wie ein JavaScript-Objekt und kann weniger aufwendig verarbeitet werden als ein XML-Dokument [Rev08].
Wie bereits erwähnt, sind bei korrekter Funktionsweise des Programms, genaue Kenntnisse über das Dateiformat, in dem gespeichert wird, nicht notwendig. Das Achten auf die richtige Verknüpfung bleibt allerdings weiterhin Aufgabe des Benutzers.

6.4 Abschließende Worte

Der Reiz für das Angehen des Projekts war das Auseinandersetzen mit einer umfangreichen Aufgabe im Bereich der Webprogrammierung. Grafische Benutzeroberflächen trugen maßgeblich zur Massentauglichkeit von Computern und mobilen Endgeräten bei. Diese Tatsache blieb bis in die Gegenwart unverändert. Dank ihnen können anspruchsvolle Aufgaben mit Notebooks, Tablets und Smartphones komfortabel gelöst werden. Mit der fortschreitenden Entwicklung heutiger Websprachen können Benutzeroberfläche ins Web verlagert werden, wodurch sie die Eigenschaft der Plattformunabhängigkeit erwerben. Es ist keine Entwicklung für mehrere Betriebssysteme notwendig.
Obwohl die eingesetzte Programmiersprache JavaScript für den Autor zu Beginn des Projekts größtenteils fremd war, bot der Studiengang Multimediatechnik die nötigen Voraussetzungen für das Aneignen von Fähigkeiten im Umgang mit dieser Sprache. JavaScript und JQuery gestalteten den Arbeitsalltag, wodurch sich nach kurzer Zeit zunehmend auf die Lösung programmtechnischer Fragen konzentriert werden konnte und das Lernen der Sprachen in den Hintergrund rückte. Aufgrund dieser Tatsache lässt sich rückblickend sagen, dass die Lernkurve bereits nach wenigen Wochen sehr stark anstieg. Dies ist nicht zuletzt dem sehr guten Arbeitsumfeld geschuldet.

Literaturverzeichnis

[agi] AGILE it: *Unit-Tests*. http://www.it-agile.de/unittests.html, Abruf: 30.01.2014

[Bar13] BARANOVSKIY, Dmitry: *Raphaël—JavaScript Library*. http://raphaeljs.com. Version: 2013, Abruf: 30.01.2014

[BB05] BERGMANN, Olaf ; BORMANN, Carsten: *AJAX - Frische Ansätze für das Web-Design*. SPC TEIA VERLAG, 2005

[can13] CANIUSE.COM: *Compatibility tables for support of HTML5, CSS3, SVG and more in desktop and mobile browsers*. http://caniuse.com/#cats=HTML5. Version: 2013, Abruf: 30.01.2014

[Chr13] CHRISTENSSON, Per: *Blob*. http://www.techterms.com/definition/blob. Version: 2013, Abruf: 30.01.2014

[Com13a] COMMUNITY, Processing: *I know Java, is this Java? How do I use it that way?* http://wiki.processing.org/w/FAQ#I_know_Java.2C_is_this_Java.3F_How_do_I_use_it_that_way.3F. Version: 2013, Abruf: 30.01.2014

[Com13b] COMMUNITY, Processing: *Processing*. http://www.processing.org. Version: 2013, Abruf: 30.01.2014

[Cro08] CROCKFORD, Douglas: *Das Beste an JavaScript*. O'Reilly, 2008

[dev13] DEVELOPER.MOZILLA.ORG: *FileReader*. https://developer.mozilla.org/en-US/docs/Web/API/FileReader. Version: 2013, Abruf: 30.01.2014

[ePa11] EPAGES: *Technical Whitepaper*, 2011

[ePa13] EPAGES: *Shopsoftware für Erfolg im E-Commerce*. http://www.epages.com/de/unternehmen/. Version: 2013, Abruf: 30.01.2014

[Gro12] GROSSBART, Zack: *Web-Drawing Throwdown: Paper.js Vs. Processing.js Vs. Raphael*. http://coding.smashingmagazine.com/2012/02/22/web-drawing-throwdown-paper-processing-raphael/. Version: 2012, Abruf: 30.01.2014

[Hea13] HEATH, Nick: *Why HTML5 and Flash need each other*. http://www.zdnet.com/why-flash-and-html5-need-each-other-7000016737/. Version: 2013, Abruf: 30.01.2014

[Hef11] HEFFERNAN, Steve: *Over 50% of web users now support HTML5 Video*. http://blog.videojs.com/post/35885839763/html5-video-statistics. Version: 2011, Abruf: 30.01.2014

[Kle11] KLEUKER, S.: *Grundkurs Software-Engineering mit UML*. 2. Auflage. Wiesbaden : Vieweg+Teubner, 2011

[LAS10] LUBBERS, Peter ; ALBERS, Brian ; SALIM, Frank: *Pro HTML5 Programming*. MyCopy Softcover Edition. Apress, 2010

[LLC13a] LLC, ImageMagick S.: *About ImageMagick*. http://www.imagemagick.org/script/index.php. Version: 2013, Abruf: 30.01.2014

[LLC13b] LLC, ImageMagick S.: *ImageMagick Magick Image File Format*. http://www.imagemagick.org/script/miff.php. Version: 2013, Abruf: 30.01.2014

[LLC13c] LLC, ImageMagick S.: *PerlMagick API*. http://www.imagemagick.org/script/perl-magick.php. Version: 2013, Abruf: 30.01.2014

[mar13] MARMELAB: *Raphael.InlineTextEditing*. https://github.com/marmelab/Raphael.InlineTextEditing. Version: 2013, Abruf: 30.01.2014

[McF12] MCFARLAND, David S.: *CSS3 - The Missing Manual*. O'Reilly, 2012

[Neu11] NEUMANN, Alexander: *GitHub populärer als SourceForge und Google Code*. http://www.heise.de/developer/meldung/GitHub-populaerer-als-SourceForge-und-Google-Code-1255416.html. Version: 2011, Abruf: 31.01.2014

[Per12] PERKINS, Luc: *Why JSON will continue to push XML out of the picture*. http://blog.appfog.com/why-json-will-continue-to-push-xml-out-of-the-picture/. Version: 2012, Abruf: 31.01.2014

[Rev08] REVILL, Leon: *Why use JSON? – 3 Reasons why you should use JSON*. http://www.revillwebdesign.com/why-use-json/. Version: 2013-02-08, Abruf: 31.01.2014

[SBS11] SNEED, Harry M. ; BAUMGARTNER, Manfred ; SEIDL, Richard: *Der Systemtest*. 3. Auflage. Carl Hanser Verlag GmbH & Co. KG, 2011

[Sta11] STATCOUNTER: *StatCounter Global Stats*. http://gs.statcounter.com/#browser_version-ww-daily-20111201-20111231. Version: 2011, Abruf: 31.01.2014

[Str12] STRATHAUSEN, Johann P.: *Dracula Graph Library*. https://github.com/strathausen/dracula. Version: 2012, Abruf: 31.01.2014

[Uni13] UNION, International T.: *ICT STATISTICS NEWSLOG - BROADBAND SPEED*. http://www.itu.int/ITU-D/ict/newslog/CategoryView,category,Broadband%2Bspeed.aspx. Version: 2013, Abruf: 31.01.2014

[VB11] VOLLENDORF, Maximilian ; BONGERS, Frank: *jQuery - Das Praxisbuch*. Galileo Computing, 2011

[W3C11] W3C: *SVG 1.1 (Second Edition)*. http://www.w3.org/TR/SVG/intro.html. Version: 2011, Abruf: 31.01.2014

[w3sa] W3SCHOOLS.COM: *A rectangle with rounded corners*. http://www.w3schools.com/svg/tryit.asp?filename=trysvg_rect4, Abruf: 31.01.2014

[w3sb] W3SCHOOLS.COM: *XML Parser*. http://www.w3schools.com/xml/xml_parser.asp, Abruf: 31.01.2014

[W3T13] W3TECHS: *Historical trends in the usage of client-side programming languages for websites*. http://w3techs.com/technologies/history_overview/client_side_language/all. Version: 2013, Abruf: 31.01.2014

[WHA13] WHATWG: *The canvas Element*. http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html#the-canvas-element. Version: 2013, Abruf: 31.01.2014

[Wir11] WIRDEMANN, Ralf: *Scrum mit User Stories*. 2. Auflage. Carl Hanser Verlag GmbH & Co. KG, 2011

Stichwortverzeichnis

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, 4. Februar 2014