Erstellung von adaptiven Web Components

Christoph Kleber

3. November 2016

Kurzfassung

In dieser Arbeit geht es um Web Components.

Abstract

This thesis is about Web Components.

Listing Verzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

API Advanced Programming Interface

DOM Document Object Model

HTML Hypertext Markup Language

URL Uniform Resource Locator

div division

HTML5 Hypertext Markup Language Version 5

CSS Cascading Style Sheets

W3C World Wide Web Consortium

1 Inhaltsverzeichnis

2 Adaptivität

2.1 Begriffsklärung

Das ist erster Text Test

2.2 Adaptivität bei User Interfaces

3 Web Components

3.1 Was sind Web Components

Web Components sind eine World Wide Web Consortium (W3C) Spezifikation. Diese soll es ermöglichen, eigenständige und wiederverwertbare Komponenten für Web Anwendungen zu erstellen.[1] Sie setzen sich zusammen aus den vier Technologien *Custom Elements*, *Hypertext Markup Language (HTML) Imports*, *Templates* und *Shadow DOM*. Das Nutzen dieser Technologie soll Applikationen im Web leichter wiederverwertbar, wartbar, unabhängiger und kapselbar machen.[2]

3.2 Geschichte von Web Components

Web Components werden vom W3C das erste Mal im Jahr 2012 erwähnt. Hier wurde es auch *Component model for the web* genannt und bestand aus den vier Technologien *Templates*, *Decorators*, *Custom Elements* und *Shadow DOM*. In der derzeitigen Version der *Web Components* wird die *Decorators* Technologie nicht mehr verwendet und stattdessen wird die *HTML Import* Technologie verwendet.[3] Da die *Web Components* aus verschiedenen Technologien zusammengesetzt sind, wird in dem nächsten Abschnitt auf die Geschichte der einzelnen Technologien eingegangen, um einen Überblick zu verschaffen.

3.2.1 Custom Elements

Custom Elements liegen in der Version v0 und in der Version v1 vor. Die Version v0 wurde das erste Mal im Jahr 2014 von den Browsern *Chrome* in der Version 33

[1] vgl. Patel. 2015, S. 1.
[2] vgl. Patel. 2015, S. 2.
[3] vgl. Cooney und Glazkov. 2012.

und *Opera* in der Version 20 unterstützt.[4] Die Version v0 wird jedoch heutzutage als veraltet angesehen.[5] Derzeit wird sie nicht per Standardeinstellung von anderen Browsern unterstützt.[6] Die Version v1 wird derzeit nur von den Browsern *Chrome* und *Opera* unterstützt. Das erste mal wurde sie im Jahr 2016 in der *Chrome* Version 54 und in der *Opera* Version 41 genutzt.[7]

3.2.2 HTML Imports

HTML Imports wurden in den Browsern *Chrome* und *Opera* zuerst 2014 unterstützt. Die Imports wurden als Erstes in der *Chrome* Version 36 und in der *Opera* Version 23 genutzt. Gegenwärtig wird die Technologie von den zuvor erwähnten Browsern auch in den mobilen Browser-Varianten unterstützt.[8]

3.2.3 Decorators

Decorators erscheinen nur in Dokumenten und Artikeln, sie werden nie von Browsern implementiert. So wird im Jahr 2012 in einem *Working Draft* des W3C von einem Beispiel gesprochen: „Here is an example of how decorators could be used to implement a simple variant of the details element“.[9] Dies zeigt, dass an dieser Stelle noch keine Implementierung dieser Technologie vorliegt. Auch in einem *Working Draft* des W3C vom Jahr 2013 wird davon gesprochen, dass „Decorators, unlike other parts of Web Components, do not have a specification yet.“[10] Auf einer aktuellen Übersichtsseite des Konsortiums wird die *Decorators* Technologie nicht mehr erwähnt.[11]

3.2.4 Templates

Templates werden schon über einen längeren Zeitraum in den verschiedenen Browsern unterstützt. Zuallererst wurden sie im *Chrome* im Jahr 2013 verwendet, in der Version 26.[12]

[4] vgl. C. 2015.
[5] vgl. Bidelman. 2016.
[6] vgl. ACHTUNGfirefox. 9999.
[7] vgl. C. 2016.
[8] vgl. Morrita. 2015.
[9] Cooney und Glazkov. 2012.
[10] Cooney und Glazkov. 2013.
[11] vgl. ACHTUNGw3c. 9999.
[12] W und K. 2015.

3.3 Vor- und Nachteile der Webentwicklung mit Web Components

3.4 Technik der Web Components

3.4.1 Custom Elements

Das *Custom Element* ist eine *Advanced Programming Interface (API)*, welches das Bilden eigener, voll funktionstüchtiger *DOM* Elemente ermöglicht.[13] Die *API* beschreibt in diesem Zusammenhang eine Schnittstelle, welche einem anderen Programm ein Tool zur Verfügung stellt, um sich an das eigene Softwaresystem anbinden zu können.[14] Somit ermöglicht eine *API* einen Austausch von Informationen zwischen verschiedenen Programmen oder Systemen. Die *Custom Element API* ermöglicht den Nutzern die Auszeichnungsprache *HTML* zu erweitern.[15] Es können bestehende *HTML* Elemente erweitert, oder neue hinzugefügt werden. Jedes neue oder erweiterte Element wird unter einem *Tag* Namen registriert. Dies ermöglicht eine Kapselung des erstellen Programmiercodes in Elemente. In Listing 3.1 ist ein *JavaScript* Programmcode dargestellt, welcher ein leeres *Custom Element* definiert und und unter dem Namen „new-custom-element“ registriert wird. Für *Custom Elements* sind mehrere

```
class NewCustomElement extends HTMLElement {
        constructor() {
                super();
        }
}
customElements.define('new-custom-element', NewCustomElement);
```

Listing 3.1: Custom Element JavaScript

Callbacks verfügbar. *Callback* Funktionen beschreiben hier Funktionen, die bei bestimmten Ereignissen des *Lifecycle* von außerhalb des *Custom Elements* aufgerufen werden. Im folgenden werden diese Funktionen aufgelistet.[16]

[13] vgl. Denicola. 2016.
[14] vgl. Behrendt. 2016.
[15] vgl. Argelius. 2016.
[16] vgl. Argelius. 2016.

connectedCallback() Diese Funktion wird aufgerufen wenn das *Custom Element* an den *DOM* angehängt wird.

disconnectedCallback() Diese Funktion wird aufgerufen, wenn das *Custom Element* vom *DOM* wieder losgelöst wird.

attributeChangedCallback(name, prevValue, newValue) Diese Funktion wird aufgerufen, wenn sich ein Attribut ändert. Sie wird jedoch nur für Attribute aufgerufen, welche in einer statischen *get* Funktion mit Namen *observedAttributes* definiert wurden.

3.4.2 HTML Imports

HTML Imports ist eine Technologie zum Importieren von externen HTML Dokumenten in ein HTML Dokument. Hier ist zu unterscheiden zwischen importierenden und importierten HTML Dokumenten. Die importierenden Dokumente besitzen einen Link, welcher mindestens die Uniform Resource Locator (URL) des *Imports* und die Eigenschaft *rel=„import“* besitzt,also ein Link eines bestimmten Typ ist, siehe Listing 3.2.[17] Die importierten Dokumente haben keinen außergewöhnlichen Aufbau im Vergleich

```
<link rel="import" href="/imports/imported-document.html">
```

Listing 3.2: Standard HTML Import

zu normalen HTML Dokumenten, sie können aus HTML, *Style* oder *Script* Elementen bestehen. Es kann auch die *Template* Technologie verwendet werden, dazu mehr in 3.4.3. Um auf den Inhalt des importierten Dokuments zuzugreifen wird *JavaScript* verwendet. Wie in Listing 3.3 dargestellt, wird zuerst nach dem Link Element gesucht, welches die Eigenschaft *rel=„import“* besitzt. Daraufhin wird dieses Dokument importiert und ein bestimmter Teil des Dokuments als *JavaScript* Variable *„elemt“* gespeichert. Hier wird ein division (div) Element, welches die Klasse *„element“* besitzt gespeichert. Dieses kann dann in der importierenden Seite genutzt werden.

[17] vgl. Glazkov und Morrita. 2016.

```
var link = document.querySelector('link[rel=import]');
var importedDocument = link.import;
var elem = importedDocument.querySelector('div.element');
```

Listing 3.3: JavaScript Code für Zugriff auf Inhalt des importierten Dokuments

3.4.3 Templates

Das Hypertext Markup Language Version 5 (HTML5) Feature *Templates* ermöglicht Teile einer Seite unabhängig vom DOM zu erstellen. Diese können dann später programmatisch zum DOM hinzugefügt werden.[18] Dies bedeutet, dass der Inhalt des *Templates*, bis er zum DOM hinzugefügt wird, nicht in der Webanwendung angezeigt wird und auch nicht über DOM Selektoren angesteuert werden kann. Gegebenenfalls im *Template* enthaltene Bilder werden nicht geladen und Skripte nicht ausgeführt.[19] In 3.4 sieht man den *JavaScript* Code um ein vorhandenes *Template* zum DOM hin-

```
var inhalt = document.querySelector("template").content;
document.querySelector("body").appendChild(inhalt);
```

Listing 3.4: JavaScript Code für das Hinzufügen eines Templates in das DOM

zuzufügen. In Zeile eins wird der Inhalt des *Templates* zur *JavaScript* Variable „inhalt“ hinzugefügt, um dann in der nächsten Zeile an den *Body* der Seite, also dem Inhalt hinzugefügt zu werden. In diesem Moment werden auch die Bilder des *Templates* geladen und eventuelle Skripte ausgeführt.

3.4.4 Shadow DOM

„Das *Shadow DOM* beschreibt die Fähigkeit eines Browsers, eine neue, völlig abgekapselte Knotenstruktur im bestehenden DOM zu erzeugen.“[20] Dies bedeutet, dass neben dem normalen *Document tree*, wessen Wurzelknoten ein Dokument ist, noch der *Shadow tree* besteht. Der Wurzelknoten des letzteren ist kein Dokument, sondern

[18] vgl. Cameron. 2015, S. 177.
[19] vgl. Potschien. 2013.
[20] Gasston. 2014, Kap. 11.1.4.

der *Shadow root*. Dies ist in Abbildung 3.1 dargestellt.[21] Die Folge dieser Kapselung

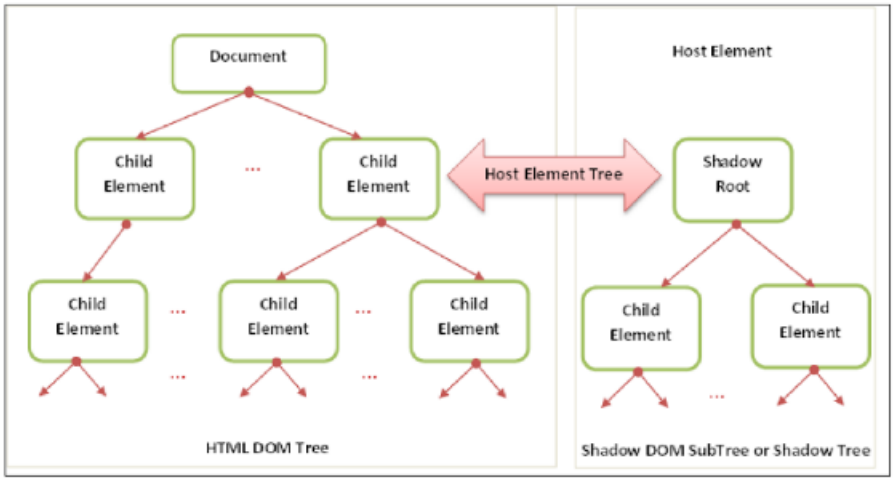

Abbildung 3.1: DOM und Shadow DOM

ist, dass alles was dem *Shadow tree* hinzugefügt wird, nur lokal Einfluss auf diesen hat. Die Gestaltung von Webelementen im *Shadow root* wird dadurch vereinfacht. Cascading Style Sheets (CSS) Selektoren können nicht von außerhalb des *Shadow roots* auf dieses Zugreifen und Selektoren, die innerhalb dieses definiert werden haben keinen Einfluss auf den normalen DOM.[22] In Listing 3.5 ist dargestellt, wie Mithilfe

```
var header = document.createElement('header');
var shadowRoot = header.attachShadow({mode: 'open'});
var headline = document.createElement("h1");
var headlineText = document.createTextNode("headline");
headline.appendChild(headlineText);
shadowRoot.appendChild(headline);
```

Listing 3.5: JavaScript Code für das Erstellen eines Shadow DOM

von JavaScript ein *Shadow DOM* erstellt wird. In Zeile zwei wird zuerst einem bestehendem Element ein *Shadow root* hinzugefügt. Daraufhin wird eine Überschrift und deren Text erstellt und zusammengefügt. Zuletzt wird in Zeile sechs die Überschrift in den *Shadow DOM* eingefügt.

[21] vgl. Patel. 2015, S. 22.
[22] Bidelman. 2016.

4 Methodik dieser Arbeit

5 Adaptive Web Components

5.1 Identifikation passender Web Components

5.1.1 Identifikation Web Components

5.1.2 Identifikation passender Preference Terms

5.2 Preference Sets zur Adaptivität

5.3 Adaptivität der bestehenden Web Components

5.3.1 Web Component Eins

5.3.1.1 Konzeption zur Adaptivität

5.3.1.2 Umsetzung Programmierung

5.3.2 Web Component Zwei

5.3.3 Web Component Drei

6 Vergleich

6.1 Vergleich mit Polymer

Literatur

[1] ACHTUNGfirefox. *Firefox Platform Status*. 9999. URL: https://platform-status.mozilla.org/%7B%5C#%7Dcustom-elements (besucht am 03. 11. 2016).

[2] ACHTUNGw3c. *WEB COMPONENTS CURRENT STATUS*. 9999. URL: https://www.w3.org/standards/techs/components%7B%5C#%7Dw3c%7B%5C_%7Dall (besucht am 03. 11. 2016).

[3] Andreas Argelius. *Create reusable and future-proof UI components with Custom Elements v1 and Web Components*. 2016. URL: https://onsen.io/blog/create-reusable-future-proof-ui-components-with-custom-elements-v1-web-components/ (besucht am 01. 11. 2016).

[4] Björn Behrendt. *Application-Programming-Interface (API) Definition*. 2016. URL: http://www.gruenderszene.de/lexikon/begriffe/application-programming-interface-api (besucht am 01. 11. 2016).

[5] Eric Bidelman. „Custom Elements v1: Reusable Web Components“. In: (2016). URL: https://developers.google.com/web/fundamentals/getting-started/primers/customelements.

[6] Dominic C. *Custom Elements v0 - Chrome Platform Status*. 2015. URL: https://www.chromestatus.com/feature/4642138092470272 (besucht am 03. 11. 2016).

[7] Dominic C. *Custom Elements v1 - Chrome Platform Status*. 2016. URL: https://www.chromestatus.com/feature/4696261944934400 (besucht am 03. 11. 2016).

[8] Dane Cameron. *HTML5, JavaScript, and jQuery 24-Hour Trainer*. 2015.

[9] Dominic Cooney und Dimitri Glazkov. *Introduction to Web Components*. 2012. URL: https://www.w3.org/TR/2012/WD-components-intro-20120522/ (besucht am 02. 11. 2016).

[10] Dominic Cooney und Dimitri Glazkov. *Introduction to Web Components*. 2013. URL: https://www.w3.org/TR/2013/WD-components-intro-20130606/%7B%5C#%7Ddecorator-section (besucht am 03. 11. 2016).

[11] Domenic Denicola. *Custom Elements*. 2016. URL: https://www.w3.org/TR/2016/WD-custom-elements-20161013/ (besucht am 01. 11. 2016).

[12] Peter Gasston. *Moderne Webentwicklung: Geräteunabhängige Entwicklung – Techniken und Trends in HTML5, CSS3 und JavaScript*. dpunkt.verlag, 2014. ISBN: 9783864914652.

[13] Dimitri Glazkov und Hajime Morrita. *HTML Imports*. 2016. URL: https://www.w3.org/TR/html-imports/ (besucht am 02. 11. 2016).

[14] Morrita. *HTML Imports - Chrome Platform Status*. 2015. URL: https://www.chromestatus.com/feature/5144752345317376 (besucht am 03. 11. 2016).

[15] Sandeep Kumar Patel. *Learning Web Component Development*. Community experience distilled. Packt Publishing Ltd, 2015. ISBN: 9781784395568.

[16] Denis Potschien. *HTML5: Wie das Template-Element komplexe Vorlagen ermöglicht*. 2013. URL: https://www.drweb.de/magazin/html5-wie-das-template-element-komplexe-html-vorlagen-ermoeglicht-40414/ (besucht am 02. 11. 2016).

[17] Rafael W und Adam K. *<template> Element - Chrome Platform Status*. 2015. URL: https://www.chromestatus.com/feature/5207287069147136.