HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG
UNIVERSITY OF APPLIED SCIENCES

[Thema der Bachelorarbeit]

Sandro Tonon

Konstanz, 15.01.2016

BACHELORARBEIT

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science (B. Sc.)

an der

Hochschule Konstanz
Technik, Wirtschaft und Gestaltung

Fakultät Informatik
Studiengang [Software-Engineering/Technische Informatik/Wirtschaftsinformatik]

Thema: [**Thema der Bachelorarbeit**]

Bachelorkandidat: Sandro Tonon, [Straße], [PLZ][Ort]

1. Prüfer: [Titel, Vor- und Zuname des 1. Prüfers]
2. Prüfer: [Titel, Vor- und Zuname des 2. Prüfers]

Ausgabedatum: [Datum]
Abgabedatum: 15.01.2016

Zusammenfassung (Abstract)

Thema:	[Thema der Bachelorarbeit]
Bachelorkandidat:	Sandro Tonon
Firma:	[HTWG oder Firmenname]
Betreuer:	[Titel, Vor- und Zuname des 1. Prüfers] [Titel, Vor- und Zuname des 2. Prüfers]
Abgabedatum:	15.01.2016
Schlagworte:	[Platz, für, spezifische, Schlagworte, zur, Ausarbeitung]

[Text der Zusammenfassung etwa 150 Worte. Es soll der Lösungsweg beschrieben sein.]

Ehrenwörtliche Erklärung

Hiermit erkläre ich *Sandro Tonon, geboren am 02.07.1990 in Test*, dass ich

(1) meine Bachelorarbeit mit dem Titel

[**Thema der Bachelorarbeit**]

bei der [HTWG oder Firmenname] unter Anleitung von [Titel, Vor- und Zuname des 1. Prüfers] selbständig und ohne fremde Hilfe angefertigt und keine anderen als die angeführten Hilfen benutzt habe;

(2) die Übernahme wörtlicher Zitate, von Tabellen, Zeichnungen, Bildern und Programmen aus der Literatur oder anderen Quellen (Internet) sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Konstanz, 15.01.2016

(Unterschrift)

Inhaltsverzeichnis

0.1 Custom Elements

Der erste Begriff unter dem Dachbegriff Web Components sind die Custom Elements, die es ermglichen eigene HTML Elemente zu definieren.

0.1.1 Einleitung

Webseiten werden mit sogenannten Elementen, oder auch Tags, aufgebaut. Das Set an verfgbaren Elementen wird vom W3C definiert und standardisiert. Somit ist die Auswahl an den verfgbaren Elementen stark begrenzt und nicht von Entwicklern erweiterbar, sodass diese ihre eigenen, von ihrer Applikation bentigten Elemente, definieren knnen. Betrachtet man den Quelltext einer Webseite im Internet, wird schnell deutlich, worin das Problem liegt.

Abbildung 1: Bild: Webseite mit semantisch nicht aussagekrftigem Markup

Die Webseite der Google Mail Applikation ist stark geschachtelt in -Elemente. Diese sind notwendig um der Webseite die gewnschte Funktionalitt und Aussehen zu verleihen. Die Probleme dieser Struktur bzw. des DOM sind deutlich: Es ist sehr schwer zu erkennen, welches Element nun was darstellt und welche Funktion hat. Abgesehen von der fehlenden, schnell ersichtlichen Semantik, also der Zuordnung der Bedeutung zu einem Element, ist der gesamte DOM nur schwer wartbar. Dieser Problematik widmen sich die Custom Elements. Sie bieten eine neue API, welche es ermglicht eigene, semantisch aussagekrftige, HTML-Elemente sowie deren Eigenschaften und Funktionen zu definieren. Wrde das obige Beispiel nun also mit Hilfe von Custom Elements umgesetzt werden, so knnte der zugehrige DOM folgender Maen aussehen [citeulike:13844982].

[] ¡hangout-module¿ ¡hangout-chat from=Paul, Addy
¿ ¡hangout-discussion¿ ¡hangout-message from=Paul
profile=profile.png
profile=”11807591949662637579 1
datetime=”2013-07-17T12:02
¿ ¡p¿Feelin’ this Web Components thing.¡/p¿ ¡p¿Heard of it?¡/p¿ ¡/hangout-message¿ ¡/hangout-discussion¿ ¡/hangout-chat¿ ¡hangout-chat¿...¡/hangout-chat¿ ¡/hangout-module¿

Die Spezifikation des W3C ermglicht nicht nur das Erstellen eigener Elemente, sondern auch das Erstellen von eigenen Elementen, die native Elemente erweitern. Somit knnen die APIs von nativen HTML Elementen um eigene Eigenschaften und Funktionen erweitert werden. Dies ermglicht es, eigene gewnschte Funktionalitten in eigens erstellten HTML Elementen zu bndeln.

0.1.2 Neue Elemente registrieren

Um nun ein eigenes Custom Element zu definieren, muss der Name des Custom Elements, laut der W3C Spezifikation, zwingend einen Bindestrich enthalten, beispielsweise `my-element`. Somit ist gewhrleistet, dass der Parser des Browsers die Custom Elements von den nativen Elementen unterscheiden kann [citeulike:13845061]. Ein neues Element wird mittels JavaScript mit der Funktion `var MyElement = document.registerElement(my-element);` registriert. Zustzlich zum Namen des Elementes, kann optional der Prototyp des Elementes angegeben werden. Dieser ist jedoch standardmig ein `HTMLElement`, somit also erst wichtig, wenn es darum geht vorhandene Elemente zu erweitern, auf dieses Thema wird jedoch gesondert eingegangen. Durch das Registrieren des Elementes wird es in die Registry des Browsers geschrieben, welche dazu verwendet wird die Definitionen der HTML-Elemente aufzulsen. Nachdem das Element registriert wurde, muss es zunchst mittels `document.createElement(tagName)` erzeugt werden, der `tagName` ist hierbei der Name des zuvor registrierten Elementes. Danach kann es per JavaScript oder HTML-Deklaration im Dokument verwendet werden [citeulike:13844979].

JavaScript
[] var myelement = document.createElement('my-element'); document.body.appendChild(myelement

HTML
[] ¡div class=ßome-html
¿ ¡my-element¿¡my-element¿ ¡/div¿

0.1.3 Vorteile von Custom Elements

Ist ein Element noch nicht definiert und nicht beim Browser registiert, steht aber im Markup der Webseite, beispielsweise <`myelement`>, wird dies kein Fehler verursachen, da dieses Element das Interface von `HTMLUnkownElement` benutzen muss [citeulike:13851253]. Ist es jedoch definiert oder beim Browser registriert worden, beispielsweise <`my-element`>, so benutzt es das Interface eines `HTMLElement`. Dies bedeutet, dass fr neue eigene Elemente, eigene APIs fr dieses Element erzeugt werden knnen, indem eigene Eigenschaften und Methoden hinzugefgt werden [citeulike:13844982]. Eigene Elemente, mit einem spezifischen Eigenverhalten und Aussehen, wie beispielsweise ein neuer Video-Player, sind dadurch mit einem Tag, statt mit einem Gerst aus Divs oder hnlichen umsetzbar.

Nachteil

Ein Custom Element, das zwar standardkonform deklariert oder erstellt, aber noch nicht beim Browser registriert wurde, ist es ein `Unresolved Element`. Steht dieses Element am Anfang des DOM, wird jedoch erst spter registriert, kann es nicht von CSS angesprochen werden. Dadurch kann ein FOUC entstehen, was bedeutet, dass das Element beim Laden der Seite nicht gestylt dargestellt wird, sondern erst nachdem es registriert wurde, das definierte Aussehen bernimmt. Um dies zu verhindern, sieht die HTML Spezifikation eine

neue CSS-Pseudoklasse `:unresolved` vor, welche deklarierte aber nicht registrierte Elemente anspricht. Somit knnen diese Elemente initial beim Laden der Seite ausgeblendet, und nach dem Registrieren wieder eingeblendet werden. Dadurch wird ein ungewolltes Anzeigen von ungestylten Inhalten verhindert [citeulike:13844984].

```
my-element:unresolved {
  display: none;
}
```

0.1.4 Vorhandene Elemente erweitern (Type extensions)

Statt neue Elemente zu erzeugen knnen sowohl native HTML Elemente, als auch eigene erstellte HTML Elemente, um Funktionen und Eigenschaften erweitert werden, auch "Type extensions" genannt. Diese erben von einem spezifischen HTMLElement, also "Element X ist ein Y". Zustzlich zum Namen des erweiterten Elementes wird nun der Prototyp, sowie der Name des zu erweiternden Elementes der `registerElement`-Funktion als Parameter bergeben. Soll nun ein erweitertes `button`-Element erzeugt werden, muss folgendes gemacht werden:

[] var ButtonExtendedProto = document.registerElement('button-extended', { prototype: Object.create(HTMLButtonElement.prototype), extends: 'button' });

Das registrierte, erweiterte Element kann nun mit dem Namen des zu erweiternden Elementes als erstem Parameter und dem Namen des erweiterten Elementes als zweitem Parameter erzeugt werden. Alternativ kann es auch mit Hilfe des Konstruktors erzeugt werden [citeulike:13752379].

JavaScript:

[] var buttonExtended = document.createElement('button', 'button-extended');

// Alternativ var buttonExtended = new ButtonExtendedProto();

Um es nun im DOM zu benutzen, muss der Name des erweiterten Elementes via dem Attribut `is=ëlementName‘‘` des erweiternden Elementes angegeben werden.

HTML:

[] ¡div class=“wrapper

¿ ¡button is=“button-extended

¿¡/button¿ ¡/div¿

Verwendung bei Github

Eine Umsetzung der Type extensions ist auf der Webseite von GitHub zu finden. Dort werden die "Latest commit" Angaben eines Repositories als ein erweitertes time-Element dargestellt. Statt des Commit-Datums und der Zeit, wird die berechnete Zeit seit dem letzten Commit angezeigt.

GitHub verwendet hierzu ein selbst erzeugtes `time-ago`-Element, welches eine Type extension auf Basis des `time`-Elementes umsetzt. Mittels dem `datetime`-Attribut wird die absolute Zeit des Commits an das interne JavaScript weitergegeben. Als Inhalt des `time`-Elements wird dann die mit JavaScript berechnete relative Zeit ausgegeben. Falls der Browser nun keine Custom Elements

Abbildung 2: Bild: Github Einsatz eines Custom Element

untersttzt oder JavaScript deaktiviert ist, wird dennoch das nicht erweiterte, native HTML `time` Element mit der absoluten Zeit angezeigt.

0.1.5 Eigenschaften und Methoden definieren

Anhand des Beispiels auf GitHub wird deutlich, wie ein Custom Element eingesetzt werden kann, jedoch sind die internen JavaScript Mechanismen nicht ersichtlich. Custom Elements machen allerdings erst so richtig Sinn, wenn man fr diese auch eigene Eigenschaften und Methoden definieren kann. Wie bei nativen HTML Elementen, ist das auch bei Custom Elements auf analoge Weise mglich [citeulike:13844979]. So kann einem Element eine Funktion zugewiesen werden, in dem diese dessen Prototyp mittels einem noch freien Namen angegeben wird. Selbiges gilt fr eine neue Eigenschaft.

[] // Methode definieren ButtonExtendedProto.alert = function () { alert('foo'); };

// Eigenschaft definieren ButtonExtendedProto.answer = 42;

0.1.6 Custom Element Life Cycle Callbacks

Custom Elements bieten eine standardisierte API an speziellen Methoden, den `Custom Element Life Cycle Callbacks`, welche es ermglichen Funktionen zu unterschiedlichen Zeitpunkten, vom Registrieren bis zum lschen eines Custom Elements, auszufhren. Diese ermglichen es zu bestimmen, wann und wie ein bestimmter Code des Custom Elements ausgefhrt werden soll.

createdCallback Die `createdCallback`-Funktion Wird ausgefhrt, wenn eine Instanz des Custom Elements erzeugt mittels `var mybutton = document.createElement(custom-elem` wird.

attachedCallback Die `attachedCallback`-Funktion wird ausgefhrt, wenn ein Custom Element dem DOM mittels `document.body.appendChild(mybutton)` angehngt wird.

detachedCallback Die `detachedCallback`-Funktion wird ausgefhrt, wenn ein Custom Element aus dem DOM mittels `document.body.removeChild(mybutton)` entfernt wird.

attributeChangedCallback Die `attributeChangedCallback`-Funktion wird ausgefhrt, wenn ein Attribut eines Custom Elements mittels `MyElement.setAttribute()` gendert wird.

So knnen die Life Cycle Callbacks fr ein neues erweitertes Button-Element wie folgt definiert werden [citeulike:13844988].

[] var ButtonExtendedProto = Object.create(HTMLElement.prototype);

ButtonExtendedProto.createdCallback = function() {...}; ButtonExtended-Proto.attachedCallback = function() {...};

var ButtonExtended = document.registerElement('button-extended', {prototype: ButtonExtendedProto});

0.1.7 Styling von Custom Elements

Das Styling von eigenen Custom Elements funktioniert analog dem Styling von nativen HTML Elementen in dem der Name des Elementes als CSS Selektor angegeben wird. Erweiterte Elemente knnen mittels dem Attribut-Selektor in CSS angesprochen werden [citeulike:13844979].

[] /* Eigenes Custom Element */ my-element { color: black; }

/* Erweitertes natives HTML Element*/ [is="button-extended

] { color: black; }

0.1.8 Browserunterstzung

HTML Imports sind noch nicht vom W3C standardisiert, sondern befinden sich noch im Status eines "Working Draft" [citeulike:13845061]. Sie werden deshalb bisher nur von Google Chrome ab Version 43 und Opera ab Version 33 nativ untersttzt.

Abbildung 3: Bild: Browserunterstzung von Custom Elements

0.1.9 Quellen

- [citeulike:13844979] Jarrod Overson & Jason Strimpel, Developing Web Components, O'Reilly 2015, S.127-138
- [citeulike:13845061] W3C Custom Elements, http://w3c.github.io/webcomponents/spec/custom/#
- [citeulike:13752379] Eiji Kitamura, Introduction to Custom Elements, http://webcomponents.org/a to-custom-elements/
- http://w3c.github.io/webcomponents/spec/custom/

- [citeulike:13844982] Eric Bidelman, Custom Elements, http://www.html5rocks.com/en/tutorials/w
- [citeulike:13844983] Can I Use, http://caniuse.com/#feat=custom-elements
- [citeulike:13844984] Peter Gasstton, A Detailed Introduction To Custom Elements, http://www.smashingmagazine.com/2014/03/introduction-to-custom-elements/
- [citeulike:13844988] Raoul Schaffranek, Web Components – eine Einfhrung, https://blog.selfhtml.org/2014/12/09/web-components-eine-einfuehrung/
- [citeulike:13851253] WHATWG, HTML Specification, https://html.spec.whatwg.org/multipage/do

0.2 Abkrzungsverzeichnis

HTML Hypertext Markup Language

W3C World Wide Web Consortium

DOM Document Object Model

CSS Cascading Style Sheets

API Application Programming Interface

FOUC Flash Of Unstyled Content

LIFO Last In First Out

MVC Model View Controller

XSS Cross Site Scripting

CORS Cross Origin Resource Sharing

SOP Same Origin Policy

HTTP Hypertext Transfer Protocol

TCP Transmission Control Protocol

TLS Transport Layer Security