Ein Überblick zu Web Components

tekom-Jahrestagung 2015 – Stuttgart, 11. November

Dr. Thomas Meinike

Hochschule Merseburg | FB Informatik und Kommunikationssysteme
http://www.iks.hs-merseburg.de/~meinike/
thomas.meinike@hs-merseburg.de

Motivation

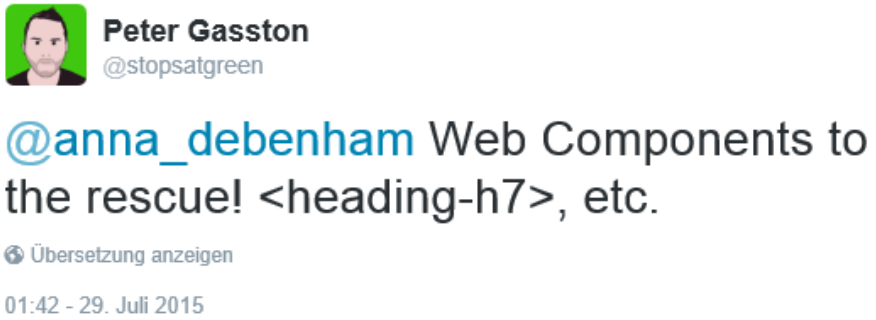
- Moderne Webanwendungen sind mehr als nur miteinander verknüpfte HTML-Dokumente.

- Es kommt eine Vielzahl an Modulen, Bibliotheken und Frameworks zum Einsatz, insbesondere auf JavaScript-Basis.

- ***Web Components*** sind nach frühen Ansätzen von Microsoft (*HTML Components* / W3C-Note von 1998) ein um 2010 geprägtes Konzept.

- Insbesondere durch Aktivitäten von Google und einem seit 2014 vom W3C angestrebten Standardisierungsprozess wird das Thema für die praktische Webentwicklung zunehmend interessant.

{Standards & Technologien …}



Standards & Technologien

➔ **Web Components** setzen bekannte Technologien voraus:

- HTML5 : Auszeichnung von Inhalten (Markup)
- CSS3 : Formatierungsregeln (Stylesheets)
- JavaScript : Programmatisches Bindeglied (Aktionscode)

➔ Dieses Fundament wird erweitert durch:

- HTML Templates : Wiederverwendbare Vorlagen (bereits Teil von HTML5)
- Custom Elements : Eigene Elementdefinitionen
- Shadow DOM : Interaktion und Kapselung
- HTML Imports : Auslagerung/Einbindung externer Komponenten

➔ W3C-Arbeitsentwürfe unter:

- w3c.github.io/webcomponents/

Standards & Technologien

➔ **HTML Templates**:

- Bestandteil der HTML5-Spezifikation
- Ablage von Dokumentteilen und Stylesheets zur Wiederverwendung
- Inhalte werden erst nach Zugriff auf die Fragmente mit JavaScript sichtbar
- Nützlich als Vorlagen für die Umsetzung von Custom Elements

Ansatz (im HTML <head> oder extern abgelegt):

```
<template id="…">
  <style> /* Stylesheet-Regeln … */ </style>
  <div class="…"> <!-- Markup … --> </div>
</template>
```

WebComponents.org

HTML Templates

CHROME	OPERA	FIREFOX	SAFARI	IE/EDGE

Standards & Technologien

➔ **Custom Elements**:

- Neue Elemente, im <body> einsetzbar wie vorhandene
- Elementnamen benötigen Bindestrich, z. B. `<mein-element>…</mein-element>`
- Bekanntmachung im DOM mit der Methode `registerElement()`

Ansatz (im HTML <head> oder extern abgelegt):

```
<script> window.onload = function() {
  var meinPrototype = Object.create(HTMLDivElement.prototype); // Element formal vom div-Blocktyp
  meinPrototype.createdCallback = function() {
    /* Aufbau des Elementinhalts (Shadow DOM) auf Basis eines Templates … */
  }
  document.registerElement("mein-element", {prototype: meinPrototype});
}
</script>
```

WebComponents.org </> Custom Elements

CHROME	OPERA	FIREFOX	SAFARI	IE/EDGE

Standards & Technologien

➔ **Shadow DOM**:

- Wird zusätzlich zum vorhandenen DOM etabliert
- Versteckte, von außerhalb nicht zugreifbare Kapselung der Komponenten
- Getrennte Behandlung und Verarbeitung der selbst definierten Elemente
- Anlegen unterhalb eines Host-Elements mit der Methode `createShadowRoot()`

Ansatz (in der Prototype-Funktion):

```
meinPrototype.createdCallback = function() {
  var shadow = this.createShadowRoot();
  /* Templatezugriffe, in tcontent liegt der geklonte/angereicherte Templateinhalt */
  shadow.appendChild(tcontent);
}
```

WebComponents.org

Standards & Technologien

➔ **HTML Imports**:

- Komponenten in .html-Datei auslagern und einbinden
- Im einbindenden Dokument stehen dann nur noch die Custom Elements und sonstigen HTML-Elemente

Ansatz (im <head>):

```
<!DOCTYPE html> <html lang="de">
  <head> <meta charset="UTF-8" /> <title>Web Components</title>
    <link rel="import" href="meine-komponente.html" />
  </head>
  <body> <h1>Web Components</h1> <mein-element>…</mein-element> </body>
</html>
```

WebComponents.org

Standards & Technologien

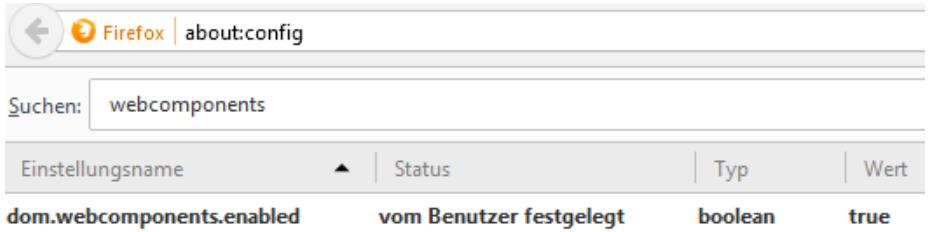
➔ Browser-Tipps:

- Aktuellen Polyfill webcomponents.min.js von WebComponents.org einbinden:

```
<script src="webcomponentsjs/webcomponents.min.js"></script>
```

- Experimentelle Unterstützung im Firefox aktivieren:

dom.webcomponents.enabled → true



- Chrome bzw. Chromium-Browser (Iron, Opera 30+) mit diesem Parameter starten, falls die Anzeige nur via HTTP (z. B. localhost) funktioniert:

--allow-file-access-from-files

{Anwendungsbeispiele …}

Anwendungsbeispiele

➔ **Record-Widget** 1/3

- Custom Element tm-record mit Attributen für jeweils ein Musikalbum:

```
<body> <h1>Record-Widget-Demo</h1>
  <tm-record name="DIE KRUPPS" title="V – Metal Machine Music"
             year="2015" medium="CD" stars="10" cover="diekrupps.png"></tm-record>
</body>
```

- Template:

```
<template id="record_widget"> <style> /* Formate … */ </style>
  <div class="record"> <img src="" alt="Cover" /><!-- Coverbild -->
    <h1></h1><!-- Bandname --> <h2></h2><!-- Plattentitel -->
    <h3><span class="year"></span><span class="stars"></span></h3><!-- Jahr (Medium) und Sterne -->
  </div>
</template>
```

Anwendungsbeispiele

➔ **Record-Widget** 2/3

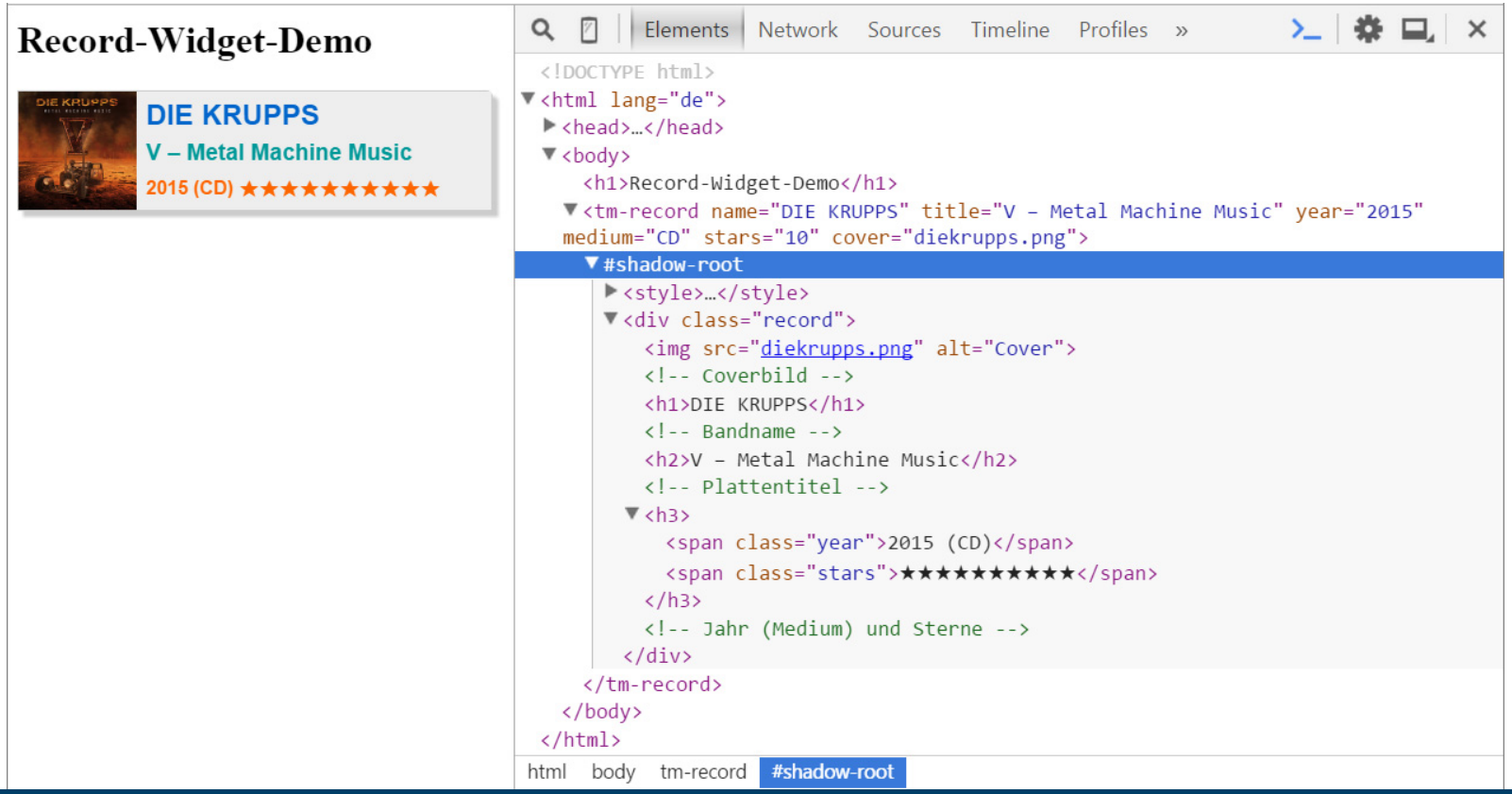
- Die JavaScript-Magie im Hintergrund (in der Prototype-Funktion):

```
var shadow = this.createShadowRoot();
var template = document.querySelector("#record_widget");
var tcontent = document.importNode(template.content, true);
    tcontent.querySelector("h1").textContent = this.getAttribute("name");
    tcontent.querySelector("h2").textContent = this.getAttribute("title");
    tcontent.querySelector(".year").textContent =
      this.getAttribute("year") + " (" + this.getAttribute("medium") + ")";
var stars = this.getAttribute("stars") * 1;
var stars_str = "";
    if(stars > 0) { for(var i = 1; i <= stars; i++) stars_str += "★"; }
    tcontent.querySelector(".stars").textContent = stars_str;
var cover = this.getAttribute("cover");
    if(cover == "")cover = "cover.png";
    tcontent.querySelector("img").src = cover;
shadow.appendChild(tcontent);
```

Anwendungsbeispiele

➔ **Record-Widget** 3/3

- Ergebnis und Elementstruktur in der Browser-Konsole (hier Opera 32):



Anwendungsbeispiele

➔ **Pie-Chart** 1/3

- Custom Element tm-piechart mit Kindelementen tm-data für Kreissegment-Daten:

```
<tm-piechart chtitle="Demochart 1">
  <tm-data value="10" color="#FF0">Text 1</tm-data>
  <tm-data value="20" color="#F00">Text 2</tm-data>
  <tm-data value="30" color="#090">Text 3</tm-data>
  <tm-data value="40" color="#CCC">Text 4</tm-data>
  <tm-data value="50" color="#00F">Text 5</tm-data>
</tm-piechart>
```

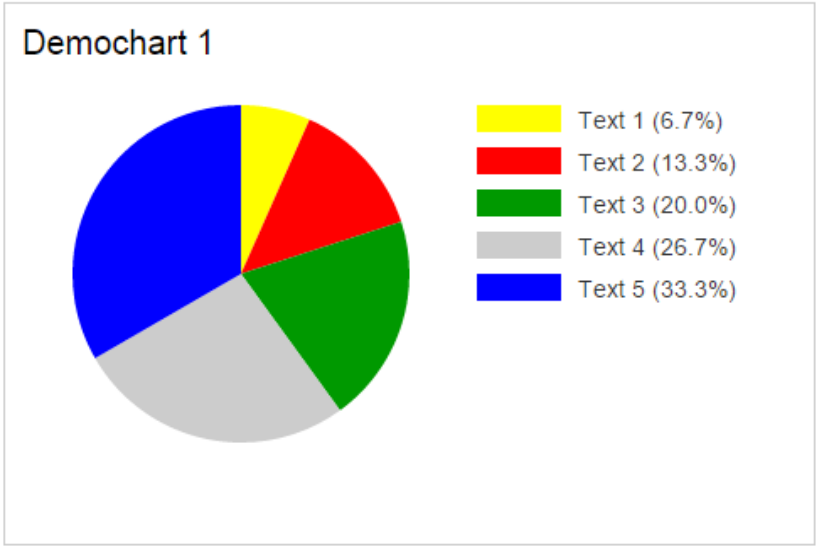
- Template mit div-Container und SVG-Grundgerüst (g kapselt ein Segment):

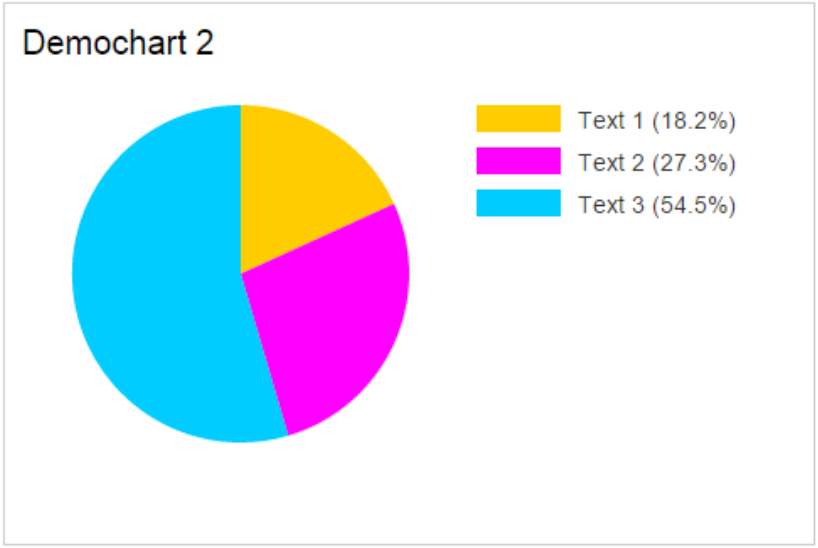
```
<template id="piebasis"> <style>…</style> <div id="tmpiechart">
  <svg xmlns="http://www.w3.org/2000/svg" width="480" height="320" viewBox="0 0 480 320">
    <text id="title" x="0" y="0"></text>
    <g> <path d="" fill="none"/>
        <rect x="0" y="0" width="0" height="0" fill="none"/>
        <text x="0" y="0"></text> </g>
  </svg> </div>
</template>
```

Anwendungsbeispiele

➔ **Pie-Chart** 2/3

- Der JavaScript-Code im Hintergrund ist umfangreich – berechnet im Wesentlichen die Kreissegmente als Pfade sowie die Legendeneinträge:





Anwendungsbeispiele

➔ **Pie-Chart** 3/3

- Browsertests mit dieser Komponente (zum Entstehungzeitpunkt 08/15 aktuell):

Dokument-Status	IE 10/11	Edge	Safari (iOS 8)	Firefox [1] 39	Opera 31	Chrome [2] 44
Ein HTML-Dokument	✗	✗	✗	✓	✓	✓
Mit Import (*tm-pietempl.html*)	✗	✗	✗	✗	✓ [3]	✓
Ein HTML-Dokument + Polyfill [4]	✓	✓	✓	✓	✓	✓
Mit Import (*tm-pietempl.html*) + Polyfill [4]	✗	✗	✗	✗	✓ [3]	✓

Hinweise:

[1] Firefox mit Option *dom.webcomponents.enabled ⇒ true*

[2] Chrome für Desktop und Android (iOS wie Safari)

[3] nur HTTP = funktioniert nicht lokal mit file-Protokoll

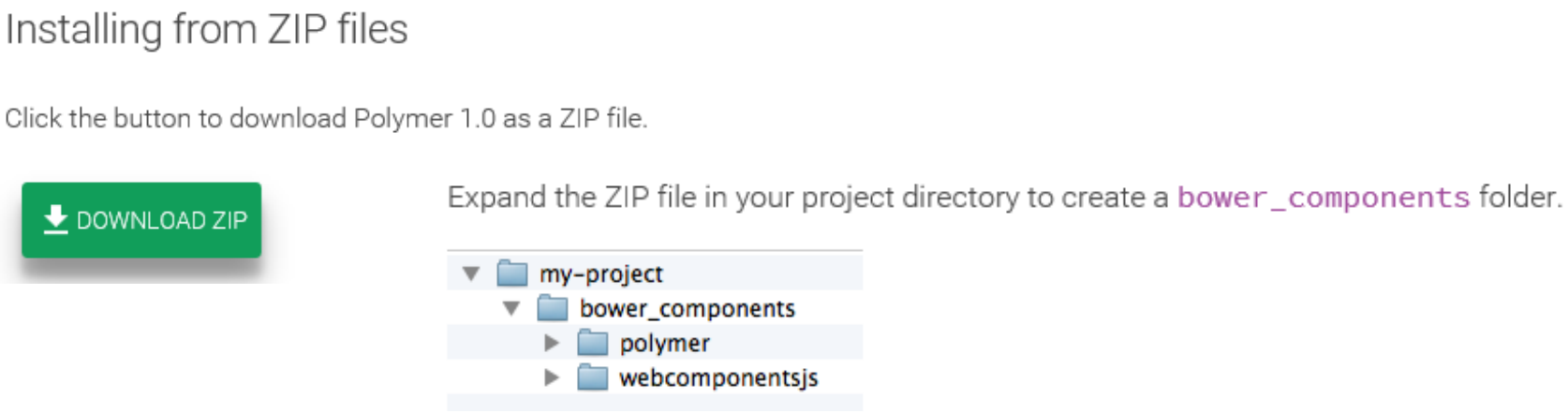
[4] verwendeter Polyfill: *webcomponents.js (Polymer)*

{Web Components mit Polymer …}

Web Components mit Polymer

➔ Was ist Polymer?

- Eine von Google initiierte Bibliothek sowie Polyfill zur vereinfachten Erstellung und Nutzung von Web Components (weitere Ansätze: Bosonic, Skate, X-Tag)
- Projektseite mit Downloads und Dokumentation: polymer-project.org
- Vorgefertigte Elemente verfügbar unter: elements.polymer-project.org
- Aktuelle Version 1.2.x (lt. Inhalt von archive.zip, 11/2015)
- Installation mit Bower oder unter »Get started« mit dem ZIP-Archiv anfangen:



Web Components mit Polymer

➔ Wie wird Polymer verwendet?

- Im <head> einbinden:

```
<script src="bower_components/webcomponentsjs/webcomponents.min.js"></script>
<link rel="import" href="bower_components/polymer/polymer.html" />
```

- Im <body> oder in externer .html-Datei sog. DOM-Module anlegen:

```
<dom-module id="mein-element">
  <style>
    …
  </style>

  <template>
    …
  </template>

  <script>
    Polymer({ is: "mein-element", /* weiterer JS-Code */ });
  </script>
</dom-module>
```

Web Components mit Polymer

➔ **Beispiel-Umsetzung von Sicherheitshinweisen** 1/6

- Custom Element tm-safety:


```
<tm-safety logo="flurfoerderzeuge.png"
           signal="GEFAHR"
           quelle="Staplerfahrer Klaus ist wieder da!"
           abwehr="Rette sich, wer kann!"></tm-safety>
```

Web Components mit Polymer

➔ **Beispiel-Umsetzung von Sicherheitshinweisen** 2/6

- DOM-Module:

```
<dom-module id="tm-safety">
  <style>…</style>
  <template>
    <div id="warnblock">
      <img src="{{logo}}" alt="{{signal}}" />
      <h1>{{signal}}</h1>
      <p><strong>{{quelle}}</strong></p>
      <p><em>{{abwehr}}</em></p>
    </div>
  </template>
  <script> Polymer({ is: "tm-safety", /* weiterer JS-Code */ }); </script>
</dom-module>
```

{{Blöcke}} werden mit den Custom-Inhalten ersetzt.

Web Components mit Polymer

➔ **Beispiel-Umsetzung von Sicherheitshinweisen** 3/6

- CSS-Auszug der Klassennamen für die JS-Zuweisung:

```
h1.symbol::before {
  content: "\26A0"; padding: 0 3px; font-size: 22px; }
h1.vorsicht {
  color: #000; background-color: #EFCB00; }
h1.gefahr {
  color: #FFF; background-color: #BB2724; }
h1.warnung {
  color: #000; background-color: #DE7D00; }
h1.hinweis {
  color: #FFF; background-color: #005989; font-style: italic; }
```

Web Components mit Polymer

➔ **Beispiel-Umsetzung von Sicherheitshinweisen** 4/6

- JavaScript-Auszug zur erweiterten Behandlung der Inhalte:

```
Polymer({
  is: "tm-safety",
  ready: function() {
    // Farben nach ANSI Z535.1 zuweisen:
    var signal = this.signal.toLowerCase();
    var h1_obj = this.querySelector("h1");
    if(signal == "vorsicht")h1_obj.className += " symbol vorsicht";
    else if(signal == "gefahr")h1_obj.className += " symbol gefahr";
    else if(signal == "warnung")h1_obj.className += " symbol warnung";
    else if(signal == "hinweis")h1_obj.className += " hinweis";
  }
});
```

Web Components mit Polymer

➔ **Beispiel-Umsetzung von Sicherheitshinweisen** 5/6

- Als externe Komponente nutzen:
  - Im <head> Polyfill und DOM-Module importieren:

```
<script src="bower_components/webcomponentsjs/webcomponents.min.js"></script>

<link rel="import" href="dom-module-tm-safety.html" />
```

  - In der externen .html-Datei zunächst Polymer referenzieren und DOM-Module ablegen:

```
<link rel="import" href="bower_components/polymer/polymer.html" />

<dom-module id="tm-safety">
  <!-- … -->
</dom-module>
```

Web Components mit Polymer

➔ **Beispiel-Umsetzung von Sicherheitshinweisen** 6/6

- Ergebnis mit mehreren Elementen tm-safety:

Texte der Si-Hinweise 1 bis 3 geborgt von Lesegefahr.de ☺

Browser-Ergebnisse:
– Chrome / Opera ok
– IE / Edge nur extern via HTTP
– Firefox ohne Ausgabe

Fazit und Ausblick

- Web Components bieten interessante konzeptionelle Erweiterungen für die Webentwicklung – noch optimistischer ausgedrückt:

- Nutzung ist durch die noch unzureichende Browserunterstützung beschränkt. Hier ist jedoch mit schnellen Fortschritten zu rechnen.
- Als besonders praktikabel erscheint die Einarbeitung in Polymer, um nicht alle benötigten Techniken neu erfinden zu müssen.

Feedback erwünscht …

- URL direkt aufrufen (auch nach der Tagung möglich):
  IN19.honestly.de

- Oder QR-Code scannen:

Danke für Ihr Interesse!