Schaltsysteme Arbeitsblätter im Netz (SANE)

Ein neuer Versuch

David Sukiennik

6. April 2017

Fachgebiet Integrierte Kommunikationssysteme

Outline

Bisheriger Stand

Entwurf & Design

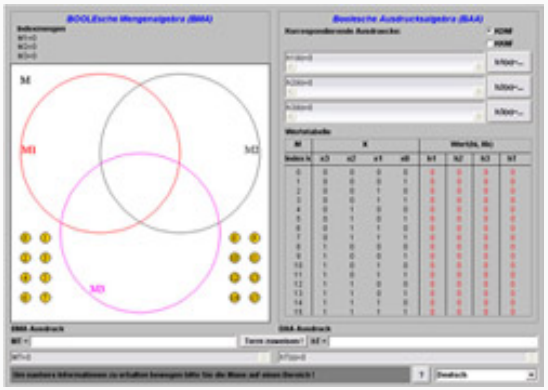
Beispiele

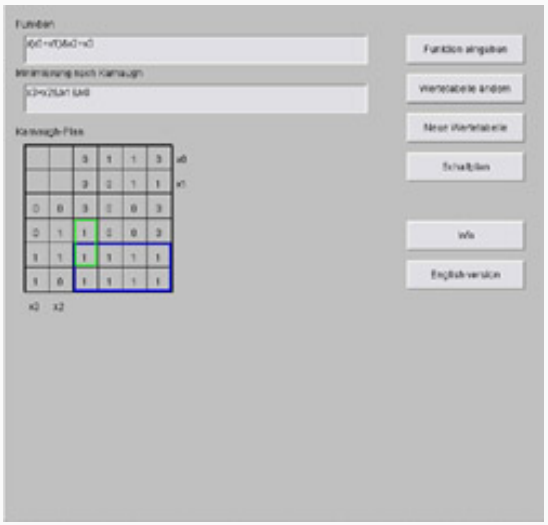
Ausblick

Bisheriger Stand

Bisheriger Stand

- einzelne Java-Applets für z. B. Karnaugh-Plan, Venn-Diagramm etc.
- zusammengeführt in der SANE-Workstation





Probleme

- Java wird nicht mehr von Browsern unterstützt
  - Firefox seit v53 (2017)
  - Chrome seit v45 (2015)
- Code entstand über viele Jahre in studentischen Arbeiten
  - Code ist veraltet, nicht einheitlich und ergo nicht wartbar
- Aussehen, Performance, Plattform nicht auf modernem Stand

Entwurf & Design

Entwurf & Design

Anforderungen

Anforderungen

- Plattformunabhängigkeit
- Unterstützung auch mobiler Plattformen
- einheitliche, moderne Code-Basis
    - wartbar

Entwurf & Design

Lösungsvorschlag

Lösungsvorschlag

- Nutzung des Browsers als Plattform statt Plugins
  - »plattformunabhängigste Plattform«: alle Betriebssysteme, vom Desktop bis zum Smartphone
  - JavaScript (JS), HyperText Markup Language (HTML), Document Object Model (DOM)
- einheitliche Code-Struktur und -Stil
- einheitlicher Datenflusses
- responsives, modernes Layout

Entwurf & Design

Kerntechnologien

Kerntechnologien

- TypeScript (TS): TypeScript
- Polymer: 
- Automatisierung (npm/yarn, gulp), Linter (TSLint), …

Entwurf & Design

TypeScript

TypeScript

- typisiertes JavaScript: erleichtert Programmierung und Debugging
  - statische Typprüfung zur Übersetzungszeit → z. B. korrekter Funktionsaufruf?
  - Refactoring wird erleichtert
  - Angabe der Interfaces erleichtert Verständnis der erwarteten Schnittstellen
- Transpilation in JavaScript erlaubt Nutzung auf Browsern
- Nutzung moderner JS-Technologien (teilweise bis ECMAScript (ES) 7: Klassen, einfacheres Scoping, …) während der Entwicklung
  - Transpilation in alten Code erlaubt Nutzung auch in Browsern ohne Unterstützung neuer Features
- von Microsoft

Entwurf & Design

Polymer

Polymer 

- Framework, um Web Components und weitere moderne Browsertechnologien einfacher zu nutzen
- Erstellung eigener, wiederverwendbarer und gekapselter HTML-Elemente
- grundlegende Elemente bereits erstellt mit modernem Anspruch auf Responsivität, Wiederverwendbarkeit und modernem Aussehen
- von Google

Komposition

- Custom Elements bestehen aus mehreren Elementen
- Custom Elements können auch aus weiteren Custom Elements bestehen
- komplexe Elemente: Komposition aus Elementen
- Vorteile:
  - Kapselung
  - Abstraktion
  - Wiederverwendbarkeit

Datenfluss

- Unidirektionaler Datenfluss
- Data Binding von »oben nach unten«, also Eltern- zu Kinderelementen
    - One-Way Data Binding erlaubt lesenden, aber keinen schreibenden Zugriff
- Kinderelemente kommunizieren per Events Änderungswünsche
    - Events »bubblen« nach oben
- mehrere Elemente (= Module) möchten Änderungen vornehmen
    - Zentralisierung und Vereinheitlichung des Datenflusses erleichtert Verständnis und Debugging
    - Einheitliche Event-Namen, die alle Module verwenden können und sollen
- `<sane-data>` verwaltet Daten

Entwurf & Design

Toolchain

Build

Build wird als npm-Script ausgeführt

1. *Linting* erzwingt einheitlichen Style und vermeidet klassische Fehler
2. *Transpilation von TypeScript in JavaScript* prüft statisch auf Korrektheit der Interfaces (Typisierung) und wandelt den Code in JavaScript um
3. *Polymer Build* minifyt HTML, CSS und JS und erstellt Service Worker

Resultat: eine App; entwickelt mit moderner, einheitlicher Technologie; lauffähig auf allen modernen Browsern

Build

Build wird als npm-Script ausgeführt

1. *Linting* erzwingt einheitlichen Style und vermeidet klassische Fehler
2. *Transpilation von TypeScript in JavaScript* prüft statisch auf Korrektheit der Interfaces (Typisierung) und wandelt den Code in JavaScript um
3. *Polymer Build* minifyt HTML, CSS und JS und erstellt Service Worker

Resultat: eine App; entwickelt mit moderner, einheitlicher Technologie; lauffähig auf allen modernen Browsern

Build

Build wird als npm-Script ausgeführt

1. *Linting* erzwingt einheitlichen Style und vermeidet klassische Fehler
2. *Transpilation von TypeScript in JavaScript* prüft statisch auf Korrektheit der Interfaces (Typisierung) und wandelt den Code in JavaScript um
3. *Polymer Build* minifyt HTML, CSS und JS und erstellt Service Worker

Resultat: eine App; entwickelt mit moderner, einheitlicher Technologie; lauffähig auf allen modernen Browsern

Build

Build wird als npm-Script ausgeführt

1. *Linting* erzwingt einheitlichen Style und vermeidet klassische Fehler
2. *Transpilation von TypeScript in JavaScript* prüft statisch auf Korrektheit der Interfaces (Typisierung) und wandelt den Code in JavaScript um
3. *Polymer Build* minifyt HTML, CSS und JS und erstellt Service Worker

Resultat: eine App; entwickelt mit moderner, einheitlicher Technologie; lauffähig auf allen modernen Browsern

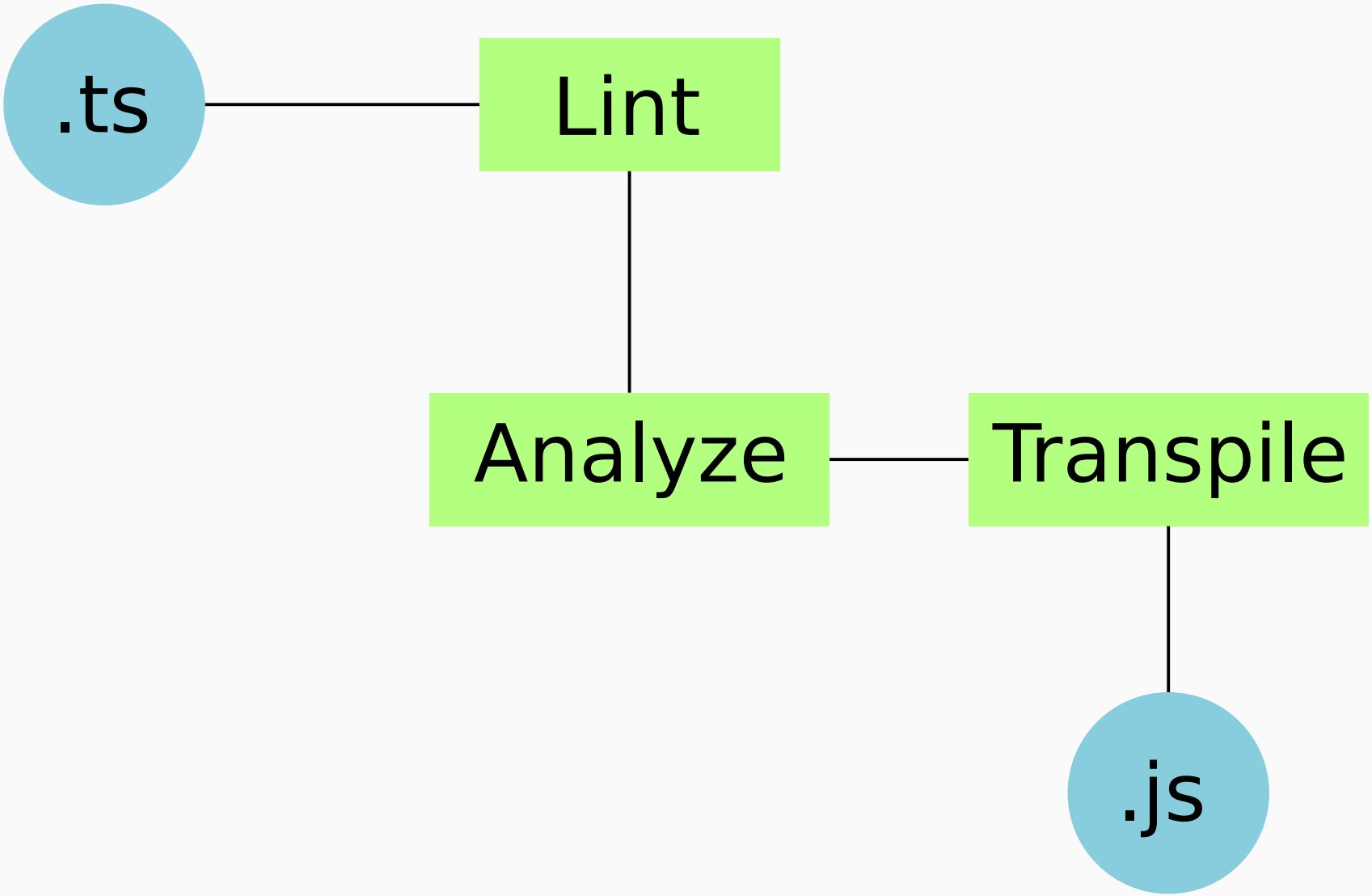


Abbildung 1: TypeScript zu JavaScript

Build

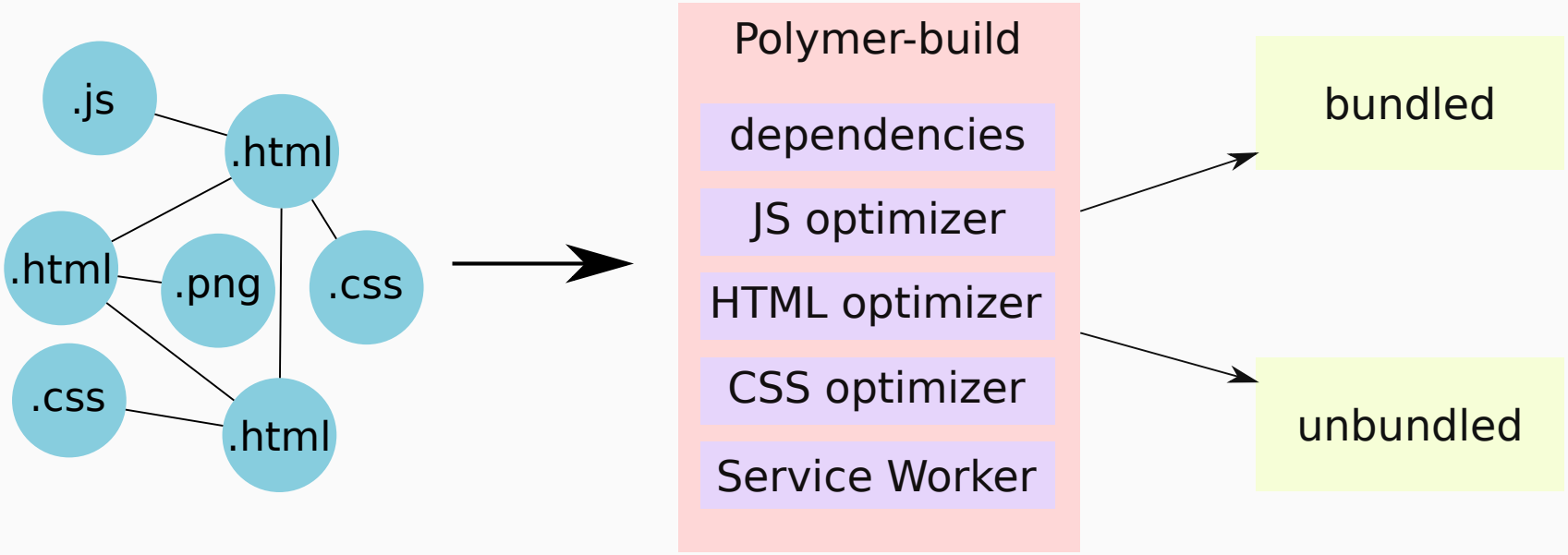


Abbildung 2: Polymer-build

Beispiele

Beispiele

TypeScript

Transpiliertes JavaScript: enum

Listing 1: TypeScript: enum

```
1 enum Language {
2   EN,
3   DE,
4   FR,
5   PL
6 }
```

Listing 2: JavaScript: IIFE

```
1 var Language;
2 (function (Language) {
3     Language[Language["EN"] = 0] = "EN";
4     Language[Language["DE"] = 1] = "DE";
5     Language[Language["FR"] = 2] = "FR";
6     Language[Language["PL"] = 3] = "PL";
7 })(Language || (Language = {}));
```

Transpiliertes JavaScript: Interfaces

Listing 3: TypeScript: Interfaces

```
1 interface Greeting {  // nur während Transpilation verwendet
2   msg: string;
3   lang: Language;
4 }
```

Transpiliertes JavaScript: `class`

Listing 4: TypeScript: `class`

```
1  class Greeter {
2    constructor(private greeting: Greeting = {msg: "Guten Tag!", lang: Language.DE}) {
           }
3    greet() {
4      return "<h1>" + this.greeting + "</h1>";
5    }
6  };
```

Listing 5: JavaScript: IIFE

```
1  var Greeter = (function () {
2      function Greeter(greeting) {
3          if (greeting === void 0) { greeting = { msg: "Guten Tag!", lang: Language.DE
               }; }
4          this.greeting = greeting;
5      }
6      Greeter.prototype.greet = function () {
7          return "<h1>" + this.greeting + "</h1>";
8      };
9      return Greeter;
10 }());
```

Transpiliertes JavaScript

Listing 6: TypeScript

```
const greeter = new Greeter({msg: "Hello, world!", lang: Language.EN});  // const!
document.body.innerHTML = greeter.greet();
```

Listing 7: JavaScript

```
var greeter = new Greeter({ msg: "Hello, world!", lang: Language.EN }); // const!
document.body.innerHTML = greeter.greet();
```

TypeScript

```
enum Language {
  EN,
  DE,
  FR,
  PL
}

interface Greeting {  // nur während Transpilation verwendet
  msg: string;
  lang: Language;
}

class Greeter {
  constructor(private greeting: Greeting = {msg: "Guten Tag!", lang: Language.DE}) {
        }
  greet() {
    return "<h1>" + this.greeting + "</h1>";
  }
};

const greeter = new Greeter({msg: "Hello, world!", lang: Language.EN});  // const!
document.body.innerHTML = greeter.greet();
```

Transpiliertes JavaScript

```
var Language;
(function (Language) {
    Language[Language["EN"] = 0] = "EN";
    Language[Language["DE"] = 1] = "DE";
    Language[Language["FR"] = 2] = "FR";
    Language[Language["PL"] = 3] = "PL";
})(Language || (Language = {}));
var Greeter = (function () {
    function Greeter(greeting) {
        if (greeting === void 0) { greeting = { msg: "Guten Tag!", lang: Language.DE
            }; }
        this.greeting = greeting;
    }
    Greeter.prototype.greet = function () {
        return "<h1>" + this.greeting + "</h1>";
    };
    return Greeter;
}());
;
var greeter = new Greeter({ msg: "Hello, world!", lang: Language.EN }); // const!
document.body.innerHTML = greeter.greet();
```

Beispiele

Web Components & Polymer

Beispiel-Element

Beispielelement `<paper-slider>`:

`https://www.webcomponents.org/element/PolymerElements/paper-slider/demo/demo/index.html`

Beispiele

`index.html`

index.html

```
<html>
  <head>
    <!-- ... -->
    <title>Schaltsysteme Arbeitsblätter im Netz</title>
    <!-- ... -->
  </head>
  <body>
  <sane-app></sane-app> <!-- eigenes HTML-Element -->
  </body>
</html>
```

Das war's! ☺

Data-Binding

```
1 <iron-pages selected="[[page]]" attr-for-selected="name"
      fallback-selection="view404" role="main">
2   <sane-view1 name="view1" language="[[language]]" data=
      "[[sanedata]]"></sane-view1>
3   <sane-view2 name="view2" language="[[language]]" data=
      "[[sanedata]]"></sane-view2>
4   <sane-view3 name="view3" language="[[language]]" data=
      "[[sanedata]]"></sane-view3>
5   <sane-view404 name="view404"></sane-view404>
6 </iron-pages>
```

Beispiele

Aktueller Stand von SANE

Demo: Beispielfunktionalitäten

- Truth Table
- Language Switch
- On-Demand Loading
- Responsives Design
- Quasi-Native app

Ausblick

Geplante Module

- Wahrheitstabelle
- Mengendiagramm
- Karnaugh-Diagramm
- g-Parameter-Bestimmung
- …

Unter Nutzung der vorhandenen Schnittstellen lassen sich beliebige Module hinzufügen.

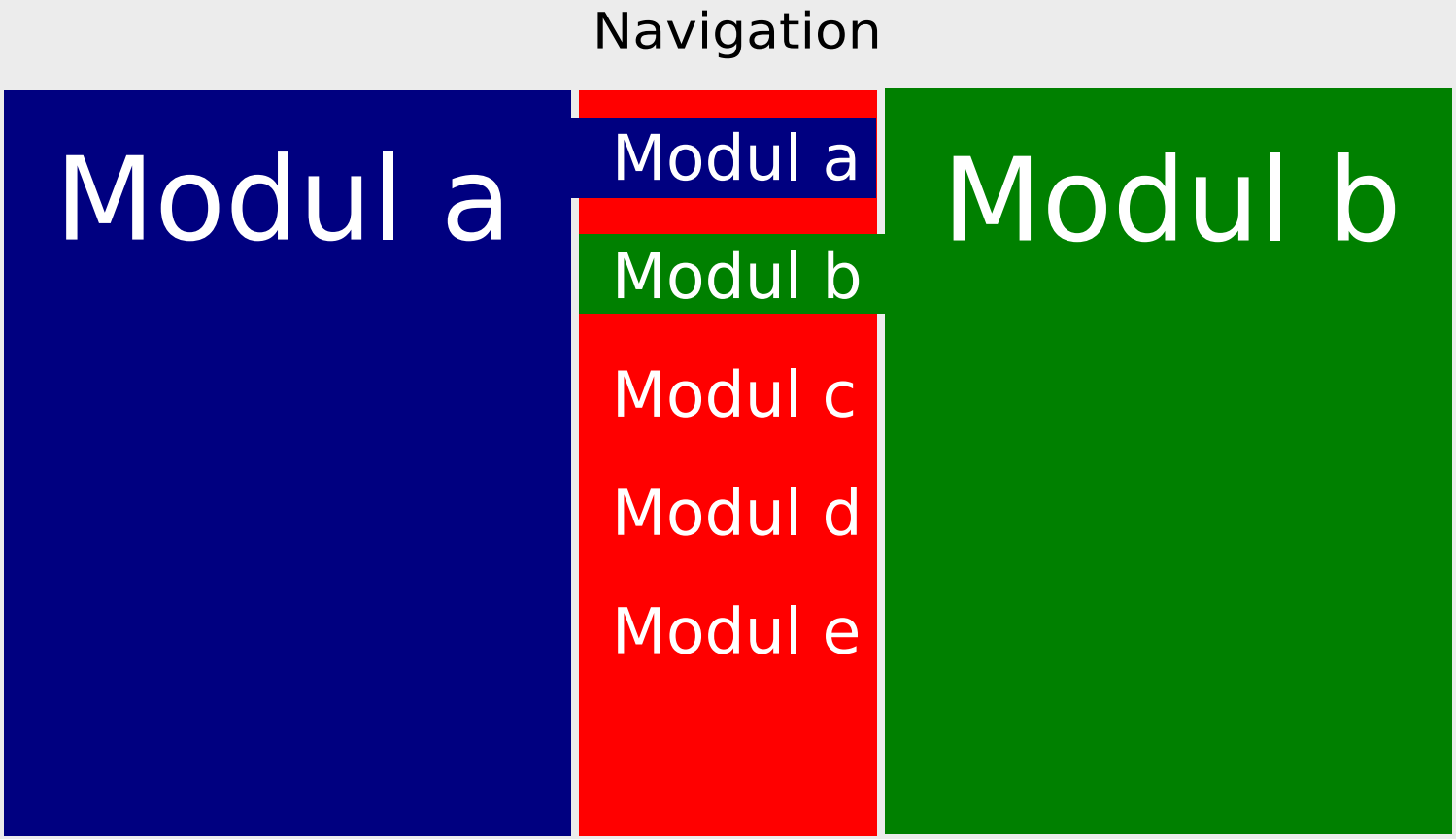


Abbildung 3: Abstrahiertes Konzept des Splitscreens

Dankeschön!

Abkürzungsverzeichnis

CSS	Cascading Style Sheet
DOM	Document Object Model
ES	ECMAScript
HTML	HyperText Markup Language
JS	JavaScript
SANE	Schaltsysteme Arbeitsblätter im Netz
TS	TypeScript

Literatur


Gulp. *gulp.js*. 23. März 2016. URL: http://gulpjs.com/.


Microsoft. *TypeScript - JavaScript that scales.* 20. März 2016. URL: http://www.typescriptlang.org/.


Inc. npm. *npm*. 23. März 2016. URL: https://www.npmjs.com/.


Polymer. *Welcome – Polymer Project*. 20. März 2016. URL: https://www.polymer-project.org/.

Literaturverzeichnis II


Palantir Technologies. *TSLint*. 23. März 2016. URL: https://palantir.github.io/tslint/.


webcomponents.org. 20. März 2016. URL: https://www.webcomponents.org/.


Yarn. *Yarn*. 23. März 2016. URL: https://yarnpkg.com/en/.