DOI: 10.35191/medsoft_2021_1_33_48_52

BODYLIGHT.JS WEB COMPONENTS – WEBOVÉ KOMPONENTY PRO WEBOVÉ SIMULÁTORY

Tomáš Kulhánek, Arnošt Mládek, Martin Brož, Jiří Kofránek

Abstrakt

S rozvojem standardu HTML5, podpory Javascriptu ve verzi EcmaScript6 a CSS stylů napříč prohlížeči a platformami je možné vyvíjet webové aplikace, které se svojí interaktivitou blíží, nebo i předčí klasické aplikace pro PC. S tímto rozvojem se taktéž rozšířila celá řada tzv. frameworků (React JS, Angular JS, Aurelia, Polymer, VUE), která doplňuje a ulehčuje tvorbu interaktivních aplikací, nicméně každý z těchto frameworků používá mírně jiné přístupy a komponenty v nich vytvořené lze obtížně znovu použít při použití jiných frameworků.

Webové komponenty (web components) se prosazují pomalu jako standardní forma sdílení interaktivních prvků moderních webových aplikací.

V tomto příspěvku představíme knihovnu "Bodylight.js web components", je distribuována dle standardu Web components. Představuje tzv. vlastní elementy, kterými lze obohatit HTML stránku, WIKI zdroj nebo dokumentaci v Markdownu apod. o prvky, které webový simulátor vybaví numerickým řešičem modelu, propojí proměnné modelu s animací a propojí jednodušší grafy s proměnnými modelu a vstupní tlačítka a posuvníky s měnitelnými parametry modelu.

Úvod

Rozvoj webových technologií, tj. značkovacího jazyka HTML, skriptovacích jazyků (Javascript) a podpora 2D a 3D grafiky daly vzniknout novému fenoménu tzv. webových aplikací, které se komfortem blíží a někdy už i předčí klasické desktopové aplikace.

Standardizace technologií způsobila, že webová aplikace dodržující standard je zobrazitelná a vykonatelná jednak v různých prohlížečích, ale i na různých platformách. Vývojář aplikace se tak nemusí starat o kompatibilitu a přenositelnost na jiné platformy.

Standard HTML a frameworky řešící nedostatky v standardu a prohlížečích

HTML je značkovací jazyk, ve kterém jsou přímo či nepřímo napsány prakticky všechny webové stránky a aplikace na webu.

Např. `<h1>Titulek</h1>` označuje titulek a prohlížeč ho zobrazí větším písmem odděleným od ostatního textu. Značka <button>Play</button> označuje tlačítko a prohlížeč ho zobrazí jako obdélník s textem, který při kliknutí myší nebo na dotykové obrazovce se promáčkne, nebo jinak předvede, že byl stisknut.

Finální specifikace HTML (někdy též HTML5) nabízí přímou podporu přehrávání multimédií (videí a zvuků) v prohlížeči pomocí značek (HTML elementů) <audio> <video> <source>, nebo zobrazení 2D bitmapové grafiky pomocí elementu <canvas> nebo vektorové grafiky pomocí elementu <svg>. HTML specifikace je dnes podporována většinou moderních webových prohlížečů (Internet Explorer, Mozilla Firefox, Google Chrome, Microsoft Edge, …) a tím postupně vytlačila různé pluginy třetích stran, které multimediální a interaktivní obsah do prohlížečů přidávali dříve (FLASH, Silverlight, …).

Obsluha a interaktivita obsahu je úkolem skriptů, které se píší v jazyce Javascript. Javascript je také standardizován normou ECMAScript a od verze ECMAScript 5 se výrobci všech hlavních prohlížečů dohodli na jeho podpoře a na podpoře standardního API, které zpřístupňuje pomocí jazyka Javascript datové struktury prohlížeče. Pomocí skriptu lze ovládat, co se v prohlížeči uživateli zobrazuje a reakce na uživatelské události (tj. kliknutí myši na odkaz, stisk tlačítka, posunutí posouvátka). Prohlížeče tento standard podporují i napříč platformami Windows, Linux, Apple a další.

API poskytované prohlížeči a možnost přistupovat do zobrazovaného webového obsahu pomocí struktury DOM je nízkoúrovňové a různé prohlížeče implementují některé interaktivní prvky nejednotně. Tato ztráta kontroly vedla a stále vede k vývoji knihoven a frameworků, které na jednu stranu zjednodušují a sjednocují interaktivitu a vylepšují tzv. user experience – česky lze přeložit nejlépe jako uživatelský prožitek.

Například odkaz (link) je ve standardním HTML definován elementem <a>:

```
<a href=”strana2.html”>Jdi na stranu 2</a>
```

Pokud uživatel klikne na odkaz, pak webová aplikace ztratí kontrolu, předá kontrolu prohlížeči, který nahraje novou stránku. Co se děje mezi kliknutím na odkaz a zobrazením nového obsahu záleží na prohlížeči. Některé zobrazí bílou stránku do té doby, než se nahraje nový obsah, novější prohlížeče zobrazují původní stránku a nový obsah zobrazí, až je celý nahrán. [1]

Tvůrce webové aplikace však může požadovat jiné chování podobné elementu <a>. Tak pomocí Javascriptu a API může reimplementovat a mít kontrolu nad chováním např. takto pomocí frameworku React:

```
var Component = React.createClass({
    getInitialState: function() { return
{query: ‘’} },
   queryChange: function(evt) {
         this.setState({query: evt.target.
value)};
   },
   handleSearch: function() {
     window.location = ‘/search/’+this.sta-
te.query+’/some-action’;
   },
   render: function() {
     return {
       <div classname=”component-wrapper”>
          <input type=”text” value=”{this.
state.query}”>
           <button onclick=”{this.handle-
Search()}”      classname=”button”>Search</
button>
       </div>
     }; } });
```

Link je v tomto případě interaktivní – mění se podle toho co uživatel chce vyhledat, a místo značky <a> se použije <button> a při stisknutí tlačítka je pomocí API volání “window.location” změněna adresa v prohlížeči, což napodobuje chování značky <a> při kliknutí na odkaz. Odkaz je ale v tomto případě interaktivní - mění se podle obsahu v textovém poli před tlačítkem “Search”.

Tento způsob vede postupně k nabalování závislosti na knihovnách a frameworcích se všemi důsledky komplexity vývoje takové aplikace [1]. Frameworky, knihovny a další nástroje řeší současné nedostatky ve webových prohlížečích a technologiích a mohou zjednodušit a snížit čas nutný pro vývoj webové aplikace. Bohužel při růstu komplexnosti aplikace se ukazuje, že vývojář ztrácí čas odstraňování jiných závad, aktualizováním knihoven, údržbou nástrojů kvůli na první pohled neviditelným závislostem [2].

DOI: 10.35191/medsoft_2021_1_33_48_52

Technologie Bodylight.js a závislost na frameworcích

V roce 2019 jsme představili technologii tvorby webových simulátorů Bodylight.js [3], která byla založena na revolučním použití a propojení různých, ale standardizovaných technologií HTML, Javascript (ECMAScript6), WebAssembly a podporujících většinou open-source knihoven tak, aby

1. využila model v jazyku Modelica, překlad simulátoru podle FMI standardu do jazyka C a dál kompilace pomocí nástroje EMSDK do WebAssembly, díky čemuž lze řídit simulaci modelu pomocí Javascriptu nativně ve webovém prohlížeči
2. export animace z aplikace Adobe Animate do Javascriptu, který pomocí knihovny CreateJS lze řídit a animovat ve webovém prohlížeči
3. kompozice dalších elementů a návrhu webové stránky a propojení animujících objektů, interaktivních tlačítek a posuvníků s proměnnými modelu v interně vyvinuté aplikaci Bodylight-Composer, jež využívá zdrojové kódy obecného frameworku GrapesJS pro vizuální tvorbu webových aplikací a udržování webového simulátoru ve struktuře podle frameworků React,Redux.
4. export webového simulátoru, zapouzdřující návrh webového simulátoru do HTML stránky, kterou lze publikovat na webu, nebo např poslat e-mailem.

Interně vyvinutá aplikace Bodylight-Composer umožnila interaktivně vizuálně vytvářet návrh webového simulátoru, propojit animovaný obrázek (vygenerovaný do javascriptu) s matematickým modelem na pozadí a model propojit s dynamicky vytvářenými grafy, a s některými předem připravenými ovládacími prvky v nabídce Bodylight-Composeru (tlačítkem, posuvným táhlíčkem, zaškrtávacím políčkem, atd).

Problém ale nastal při rozšiřování tohoto systému a údržbě na první pohled neviditelných závislostí.

1. Framework React a Redux se posunul do nové verze s vylepšeným uživatelským prožitkem, ale i s narůstajícím upraveným API. Člověk programující webovou aplikaci, ale neprogramuje čisté HTML, nebo Javascript, musí se přizpůsobit stylu, který mu je někdy více, někdy méně vnucován frameworkem.
2. Knihovna GrapesJS (použitá pro uživatelské rozhraní aplikace Bodylight Composer) včetně aktualizací a bezpečnostních záplat změnila některé interní API a vydala už další 2 majoritní verze s výraznými změnami.
3. Nástroj Adobe Animate změnil exportované animace a API změnil na verzi 2.0, které je nyní stabilní už 2 roky, nicméně není kompatibilní s předchozími verzemi webových simulátorů vyvinutých v Bodylight-Composeru.
4. Prostředí EMSDK změnilo kompilaci zdrojáků z jazyka C do WebAssembly podle ECMAScript 6 a představilo několik bezpečnostních záplat, výstupy jsou nekompatibilní s předchozím typem volání v Bodylight-Composeru.
5. Pokud při vytváření aplikace bylo potřeba upravit vzhled nebo chování aplikace, změna v exportovaném kódu byla prakticky vyloučena pro někoho, kdo nezná interní struktury a volání a bylo nutné použít opět nástroj Composer.

Interaktivní tvorba aplikace a dobře přijímaný uživatelský prožitek byl vykoupen složitostí přidávání nové funkcionality a udržováním závislostí na použitém frameworku. Nový typ prvku ve webové simulaci na sebe nabaloval další neviditelné závislosti a rozvoj a údržba aplikace Composer se stávala pracnější a nákladnější. I drobná úprava již existujícího webového simulátoru vždy vyžadovala znovu načíst simulátor do composeru, provést změnu a opět simulátor exportovat. A pokud bylo potřeba upravit zásadnější změny použitých komponent, nebo přidat chování, které Composer ve své nabídce komponent zatím nenabízel, bylo nutné to v Composeru pomocí Reactu z gruntu naprogramovat. Vyvinutí tohoto nástroje ukázalo proveditelnost tohoto konceptu, údržba však ukázala ekonomické limity produktu, který je vyvíjen z grantu pro potřeby výzkumu.

Z toho důvodu jsme postupně vydali Bodylight.js verzi 2.0 a vydělili jsme funkcionalitu webových simulátorů do tzv. webových komponent detailněji v samostatné sekci níže a nástrojů podporujících tvorbu. Architektura viz obrázek níže.

Nástroje pro tvorbu webových simulátorů

Jedním z řešení, jak udržet závislost na dalších knihovnách a nástrojích na uzdě a umožnit pracovat na vývoji v heterogenním vědeckém prostředí je virtualizace a příprava vzorové konfigurace knihoven ve vzorovém prostředí. Z toho důvodu jsme vytvořili konfigurační skripty, které pomocí nástroje VAGRANT a VIRTUALBOX vytvoří vzorový virtuální stroj na bázi Linuxového operačního systému Scientific Linux 7 *(https://github.com/creative-connections/Bodylight-Virtualmachine*).

Webové komponenty

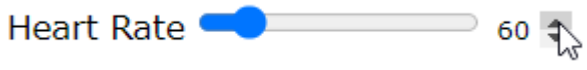
Dalším z řešení redukující skryté závislosti třetích knihoven a frameworků při výrobě webových aplikací (simulátorů) představuje koncept tzv. webových komponent a custom-elementů. Custom-element vypadá jako HTML element a standard web-components definuje jednoduché API, jak takové nové elementy registrovat, aby je mohl webový prohlížeč zobrazit a interpretovat. Komponentu je možné distribuovat jako samostatnou jednotku bez dalších závislostí.

Komponentu, která zobrazí posuvník s intervalem čísel 40 až 180 a předvolenou hodnotou 60, je možné navrhnout jako následující element a může se vyskytovat vedle ostatních značek HTML:

```
<bdl-range
  id="id1"
  title="Heart Rate"
  min="40"
  max="180"
  default="60"
  step="1"></bdl-range>
```

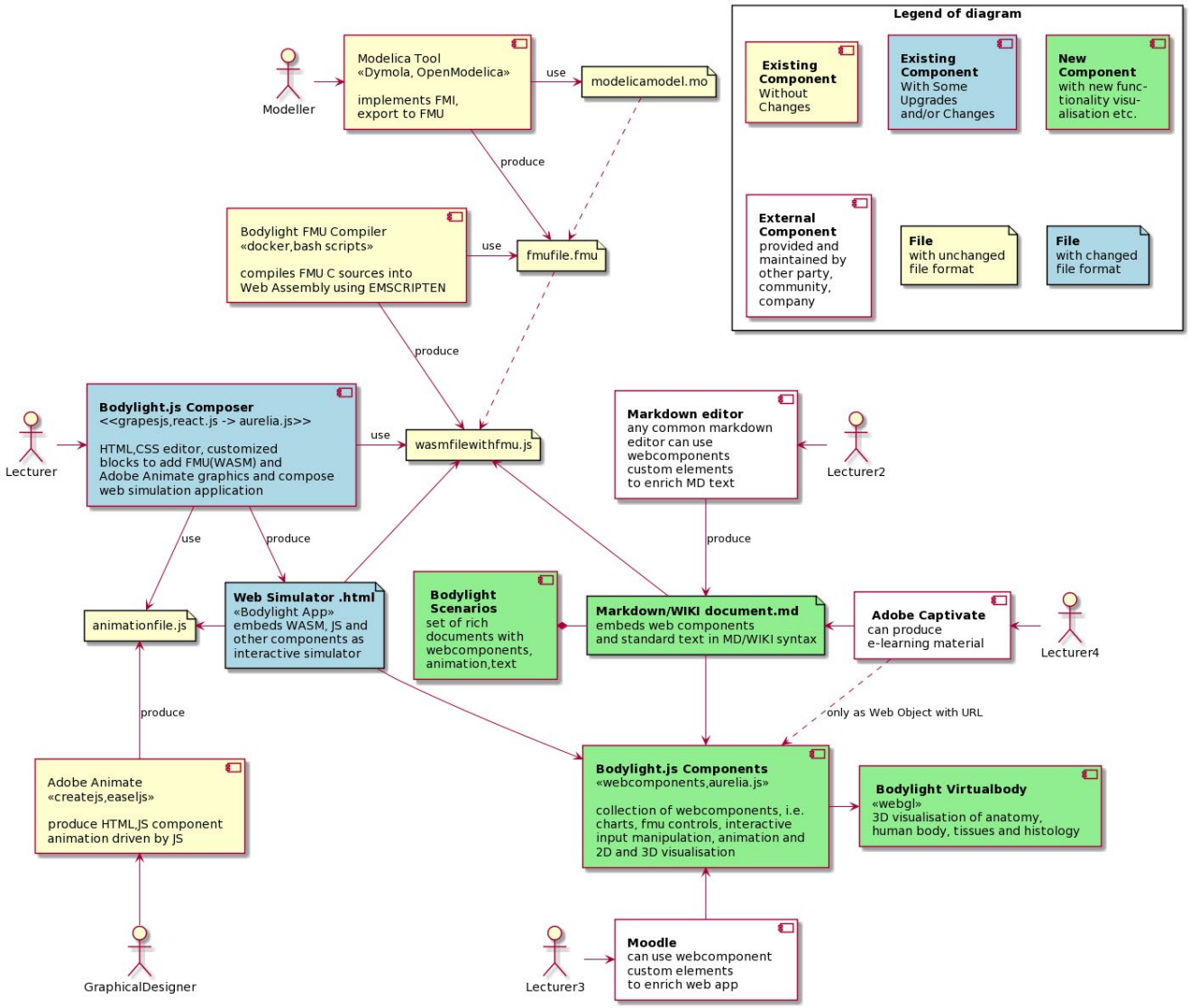
Kde id je identifikátor komponenty, který lze použít k propojení s jinou komponentou, title – titulek, min a max definují dolní a horní limit numerické škály v posuvníku, default – definuje přednastavenou hodnotu a step definuje povolený minimální krok v posuvníku.

Implementace takto navržené webové komponenty je součástí nového balíčku Bodylight.js-Components a takto navržená webová komponenta je pomocí API registrována v prohlížeči tak, že pokud se v HTML kódu stránky objeví element <bdl--range>, pak se její implementace postará o zobrazení a interaktivitu, viz obrázek 1.



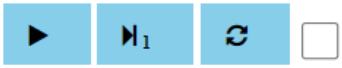
Komponenta pro řízení simulace exportované z jazyka Modelica definuje výstupní proměnné (`pulmonaryVeins.pressure,arteries.pressure`), jejichž hodnoty budou k dispozici v simulovaných časových intervalech a vstupní proměnnou (`heartRate.k`), která bude nastavována jinou komponentou (s id1) viz předchozí bdl-range:

DOI: 10.35191/medsoft_2021_1_33_48_52




```
  <bdl-fmi
  id="idfmi"
  src="Physiolibrary_Fluid_Examples_Fernandez2013_PulsatileCirculation.js"
  fminame="Physiolibrary_Fluid_Examples_Fernandez2013_PulsatileCirculation"
  tolerance="0.000001"
  starttime="0"
  fstepsize="0.01"
  guid="{a786b906-f58b-4014-8c9b-5df08bd77f4b}"
  valuereferences="637534263,637534417"
  valuelabels="pulmonaryVeins.pressure,arteries.pressure"
  inputs="id1,16777329,1,60"
  inputlabels="heartRate.k"></bdl-fmi>
```

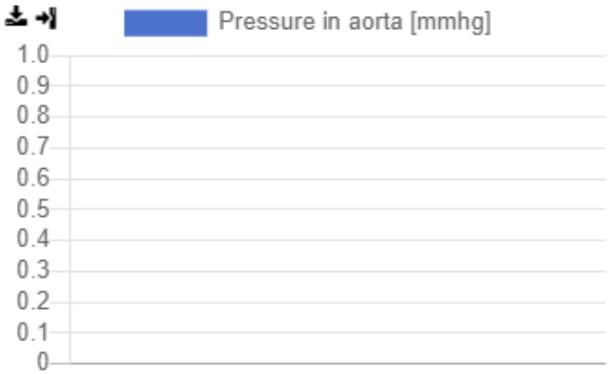
Tato komponenta se jmenuje bdl-fmi, neboť přistupuje k modelu přes rozhraní FMI API. Tato komponenta zobrazí řídící tlačítka, které ovládají simulátor (zastaví, pustí, krokuje) viz obr. 2.



Komponenta grafu zobrazí hodnoty ze simulace v grafu, definuje šířku, výšku grafu a jak zkonvertovat hodnotu jdoucí z modelu:

```
<bdl-chartjs-time
  id="id10"
  width="300"
  height="200"
  fromid="idfmi"
  labels=""
  initialdata=""
  refindex="0"
  refvalues="1"
 convertors="x/133.322-760"></bdl-chartjs-time>
```

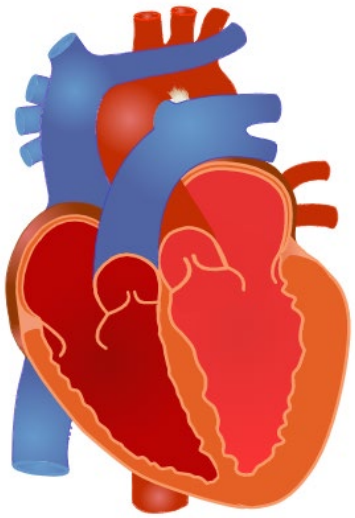
Tato komponenta se zobrazí jako graf, viz obrázek 3



DOI: 10.35191/medsoft_2021_1_33_48_52

Komponenta animace zobrazí animační objekt exportovaný z Adobe Animate a další komponenty (bind2a) propojí animovatelné objekty s hodnotami proměnných, které posílá předchozí komponenta FMI.

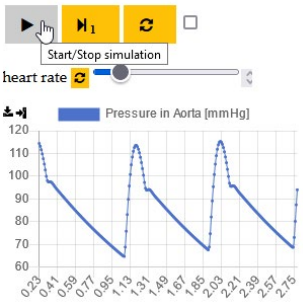
```
<bdl-animate-adobe src="CardiaccycleStage.
js" name="Faze_srdce" fromid="idfmi"></
bdl-animate-adobe>


<bdl-bind2a findex="1" aname="ValveMV_anim"
amin="99" amax="0" fmin="0" fmax="1"></bdl-bind2a>

<bdl-bind2a findex="2" aname="ValveAOV_anim"
amin="0" amax="99" fmin="0" fmax="1"></bdl-bind2a>

<bdl-bind2a findex="3" aname="ValveTV_anim"
amin="99" amax="0" fmin="0" fmax="1"></bdl-bind2a>

<bdl-bind2a findex="4" aname="ValvePV_anim"
amin="0" amax="99" fmin="0" fmax="1"></bdl-bind2a>

<bdl-bind2a findex="5" aname="ventricles.
ventriclesTotal.VentricleLeft_anim" amin="100"
amax="0" fmin="0.00015" fmax="0.00021"></
bdl-bind2a>

<bdl-bind2a findex="6" aname="ventricles.
ventriclesTotal.children.0.VentricleRi-
ght_anim" amin="100" amax="0" fmin="0.00012"
fmax="0.00018"></bdl-bind2a>
```

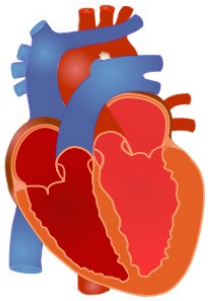


Webový simulátor

Webový simulátor napsaný v HTML a používající výše zmíněné komponenty vypadá takto:

```
<html>
  <head>
   <script src="bodylight.bundle.js"></script>
</head>
<body aurelia-app="webcomponents">
  <bdl-range id="id1" ...></bdl-range>

  <bdl-fmi d="idfmi" ...></bdl-fmi>
      <bdl-chartjs-time id="id10" ...></
bdl-chartjs-time>
  <bdl-animate-adobe ...></bdl-animate-adobe>

  <bdl-bind2a findex="1" ...></bdl-bind2a>
  <bdl-bind2a findex="2" ...></bdl-bind2a>
  <bdl-bind2a findex="3" ...></bdl-bind2a>
  <bdl-bind2a findex="4" ...></bdl-bind2a>
  <bdl-bind2a findex="5" ...></bdl-bind2a>
  <bdl-bind2a findex="6" ...></bdl-bind2a>
</body>
</html>
```





Tvorba

Přestože pro výrobu webového simulátoru s komponentami Bodylight lze použít jakýkoliv textový, nebo HTML editor, po zkušenosti z aplikace Composer, přece jenom chyběl nástroj, který by přečetl a nabídl seznam proměnných modelu a seznam animačních prvků. Proto byl vyvinut prototypový nástroj Bodylight-Editor, ve kterém uživatel může editovat zdrojový kód webového simulátoru v HTML nebo v syntaxi Markdown a generovat náhled aplikace živě. Tutoriál [4] v anglickém jazyce provází od exportu modelu z Modeliky po vytvoření webového simulátoru s grafem a interaktivní animací.

Závěr

Komponenty Bodylight.js jsou distribuovány jako samostatná knihovna bez závislostí jako NPM balíček bodylight-components. Zdrojové kódy jsou k dispozici jako open-source *https://github.com/creative-connections/Bodylight.js-Components* a releasy jsou citovatelné ze služby Zenodo [5].

Dovolíme si tvrdit, že standardní webové komponenty – hlavně použití vlastních custom-elementů, tj. na-míru implementovaných elementů eliminuje problém nabalování závislostí na knihovnách a frameworcích. Díky použití webových komponent tvůrce webového simulátoru může použít čisté HTML bez nutnosti dalších frameworků, a přitom má dostupnou vizualizační možnosti a sílu HTML a CSS. Nebo může použít jakýkoliv jiný oblíbený framework nebo knihovny.

Tvorba webového simulátoru nevyžaduje specializovanou aplikaci, nicméně použití Bodylight-Editoru je vhodné pro vytvoření základní kostry s webovými komponentami. Pokud se používání komponent Bodylight rozšíří v širší komunitě, případně budou prostředky v dalších grantech, bude aktualizován i nástroj Bodylight-Composer.

Tato práce vznikla za přispění grantů MPO Trio FV20628 Lékařský trenažér a MPO Trio FV30195 Robotické mechanotronické trenažéry s rozšířenou realitou pro lékařskou výuku.

Reference

[1.] Stuart Langridge, GOTO Conference 2019, https://www.youtube.com/watch?v=rxlJRydqmk8

[2.] Garrett Dimon, blog, https://garrettdimon.com/2019/openness-and--longevity/

[3.] Polák, D., Ježek, F., Šilar, J., & Kofránek, J. Technologie tvorby webových simulátorů. MEDSOFT. 2019; 31: 122, 139. http://www.creative-connections.cz/medsoft/2019/Medsoft_2019_Polak.pdf

[4.] Bodylight.js Tutorial. https://bodylight.physiome.cz/Bodylight-docs/tutorial

[5.] Bodylight.js Components, version 2.0.37, https://dx.doi.org/10.5281/zenodo.5094269

Kontakt

Mgr. Tomáš Kulhánek, Ph.D.
Oddělení biokybernetiky
U nemocnice 5
128 53 Praha 2
e-mail: *tmkulhanek@gmail.com*