Bodylight.js 2.0 - Web components for FMU simulation, visualisation and animation in standard web browser.

Tomáš Kulhánek[1,2] Arnošt Mládek[1,2] Filip Ježek[2,3] Jiří Kofránek[1,2]

[1]Creative Connections s.r.o., Prague, Czechia
[2]Institute of Pathological Physiology, Charles University, Prague, Czechia {tomas.kulhanek, arnost.mladek, jiri.kofranek}@lf1.cuni.cz
[3]University of Michigan, Ann Arbor, USA fjezek@umich.edu

Abstract

Simulators used in teaching and education comprise a mathematical model and a user interface that allows the user to control model inputs and intuitively visualize the model states and results. This paper presents web components - that can be used to build an in-browser web simulator. The models used for the web simulators must be written in standard Modelica language and compiled as standard FMU (Functional mockup unit). The toolchain version Bodylight.js 2.0 contains tools to collect FMU into WebAssembly language, able to be executed directly by a web browser. Bodylight.js 2.0 web components can combine models, interactive animations, and charts into a rich web documents in HTML or Markdown syntax without any other programming or scripting. Samples show its usage in education, 2D and 3D graphics, virtual reality, and connected to the hardware.

Keywords: Modelica, JavaScript, WebAssembly, in-browser simulator, client-side simulator, e-learning, web components

1 Introduction

Web-based simulators can be distinguished by where the simulation computation is performed. The server-side simulators provide a user with an interface that controls simulation performed on a remote server, and the creation of such a simulator needs to employ client-server technologies. On the other hand, the client-side simulator's user interface and simulation computation are performed on a client's computer. This, however, comprises several issues that need to be addressed. First, a user may have different types of platforms; in the past, the central platform was Microsoft Windows-based system and therefore, many simulators were distributed as an installable applications on this platform. The locally installed application may need to be manually or semi-automatically updated or upgraded. Nevertheless, MS Windows-based systems are no longer significant platforms for computer or mobile devices.

One can address many different platforms, e.g. by virtualization using technologies such as VirtualBox, VMWare, or containerization such as Docker, etc. However, web standards developed into mature versions, and the vendors of contemporary web browsers cover many platforms, including mobile phones and tablets, giving standard HTML and JavaScript capabilities.

Mathematical models in biomedical engineering can be expressed in different languages or technologies. One is the Modelica language, which covers broad industry domains; therefore, commercial and open-source tools are available. Modelica is very well suited for usage in the physiology domain and biomedical teaching, as discussed elsewhere (Kofránek, Ježek, and Mateják 2019), though, it is not yet widely used in physiology modeling community.

Direct solving of Modelica models in a web browser were demonstrated, e.g., by Franke (Franke 2014). However, accurate web-based client simulation or in-browser simulation was prototyped by Short (Short 2014) and realized in the "Modelica By Example" and "Modelica University" by Tiller and Winkler (M. M. Tiller 2014; Winkler and M. Tiller 2017).

This inspired our team to create an in-browser simulator. We already published a technology called Bodylight.js (Šilar, Ježek, et al. 2019) and sample web simulators, e.g., kidney functioning model (Šilar, Polák, et al. 2019).

The present paper describes the next evolutionary stage of this set of open-source tools, titled Bodylight.js-Components version 2.0. These are distributed as framework-agnostic web components (WebComponents 2021) - i.e., custom elements enhancing the syntax of HTML or Markdown. Further sections describe a brief methodology for creating a web simulator from a model source. A demo is presented with a pulsatile heart web simulator combining buttons, sliders, interactive graphics, and charts. The main aim of the methodology is to enable creative cooperation among domain experts such as computer graphics designers, model developers, educators, and programmers (Figure 1). Their expert work results can be integrated with the Bodylight toolchain.

2 Methods

Modelica model must be exported as FMU. We have prepared scripts to compile such output into WebAssembly using the EMScripten SDK tools. Then the resulting JS with embedded WebAssembly can be controlled using

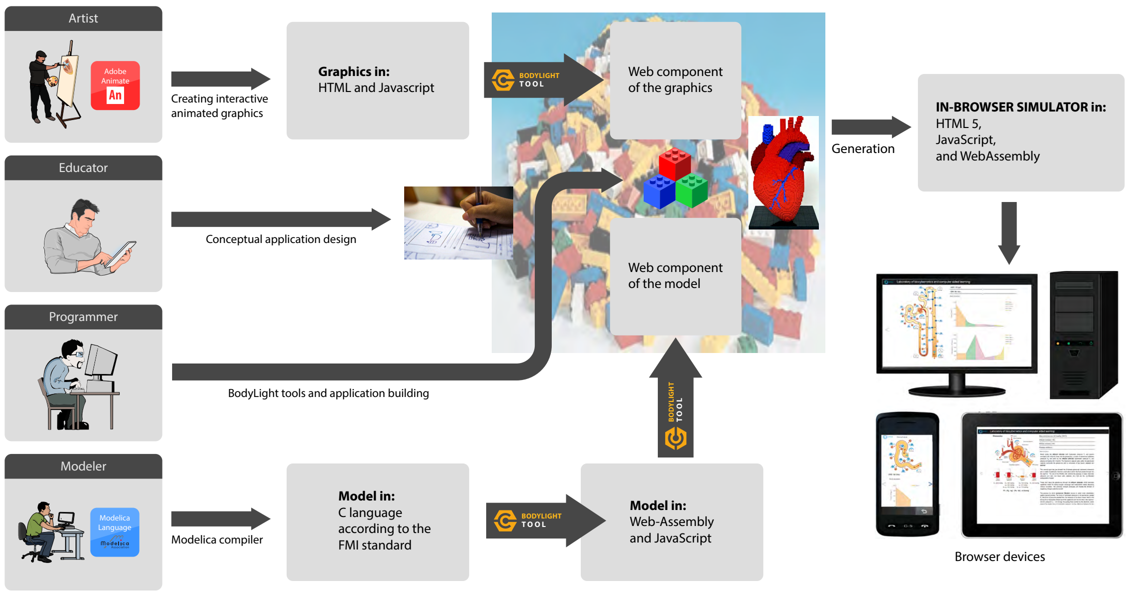


Figure 1. The main aim of the methodology is to enable creative cooperation among different domain experts such as computer graphics designers, model developers, educators, and programmers.

FMI API calls. We have prepared the web-component BDL-FMI that simplifies controlling and integrating it with other simulation related tasks like drawing charts, changing model parameters, reseting the simulation and visualising in 2D and 3D graphics. Next subsections describes the details of each particular step.

2.1 Model to WebAssembly

Modelica model must be exported as FMU v2.0 in co-simulation mode, including C source codes. This can be done either with an advanced CVODE solver in (Dymola 2023) (Dassault Systemes) or only with a more straightforward Euler solver in OpenModelica (Fritzson and et.al. 2019). Then the FMU with included source codes of solver can be compiled to JavaScript with embedded WebAssembly using Bodylight.js-FMU-Compiler[1]. It contains scripts and configuration to utilize the emscripten (EMScripten 2021) library.

In further text, the sample simulator uses exported model from Physiolibrary as seen in Figure 3.

[1]Bodylight.js-FMU-Compiler `https://github.com/creative-connections/Bodylight.js-FMU-Compiler`

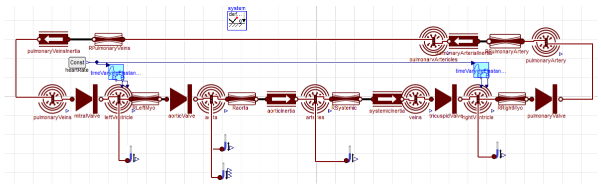
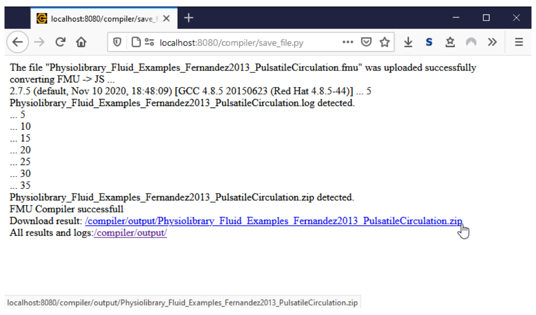


Figure 3. Model of pulsatile circulation (Fernandez de Canete et al. 2013; Kulhánek et al. 2014) in Chemical library(Matejak et al. 2015) and Physiolibrary(Mateják et al. 2014) v 3.0 using Modelica Standard Library v 4.0(Library 2021). This model is used in following sample export and web simulator.

A simple web form facilitates compilation as seen in Figure 4.


localhost:8080/compiler/save_f
localhost:8080/compiler/save_file.py

```
The file "Physiolibrary_Fluid_Examples_Fernandez2013_PulsatileCirculation.fmu" was uploaded successfully
converting FMU -> JS ...
2.7.5 (default, Nov 10 2020, 18:48:09) [GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] ... 5
Physiolibrary_Fluid_Examples_Fernandez2013_PulsatileCirculation.log detected.
... 5
... 10
... 15
... 20
... 25
... 30
... 35
Physiolibrary_Fluid_Examples_Fernandez2013_PulsatileCirculation.zip detected.
FMU Compiler successfull
Download result: /compiler/output/Physiolibrary_Fluid_Examples_Fernandez2013_PulsatileCirculation.zip
All results and logs:/compiler/output/
```

localhost:8080/compiler/output/Physiolibrary_Fluid_Examples_Fernandez2013_PulsatileCirculation.zip

Figure 4. Bodylight FMU Compiler - web form showing process of compiling FMU to JS packed as ZIP archive

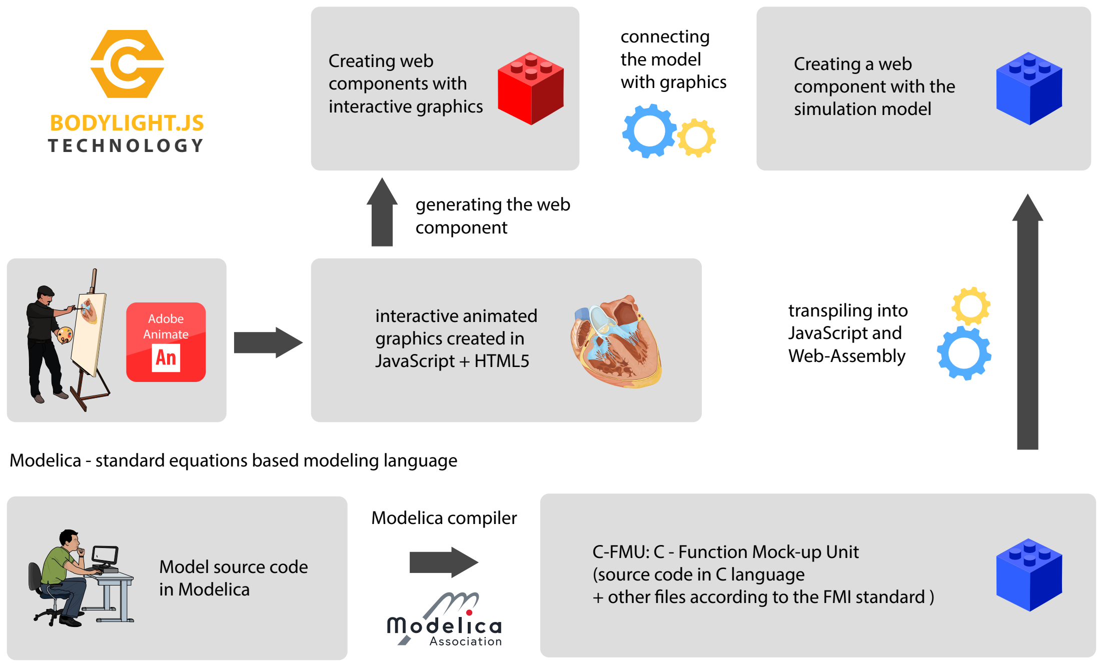

BODYLIGHT.JS TECHNOLOGY
Creating web components with interactive graphics
connecting the model with graphics
Creating a web component with the simulation model
generating the web component
Adobe Animate
An
interactive animated graphics created in JavaScript + HTML5
transpiling into JavaScript and Web-Assembly
Modelica - standard equations based modeling language
Model source code in Modelica
Modelica compiler
Modelica Association
C-FMU: C - Function Mock-up Unit (source code in C language + other files according to the FMI standard)

Figure 2. Presented web simulator creation technology is based on open web standards and available modeling standards. We create interactive animated graphics in Adobe Animate published with CreateJS library as JavaScript controlling an HTML canvas. Such an artifact is encapsulated as a web component. A model created in the Modelica language is exported into FMU with source codes, following FMI 2.0 standards. Our technology then can compile the FMU with C source codes into JavaScript with embedded WebAssembly. This artifact can then be encapsulated into another web component. Bodylight.js Components make it easier to link the graphics web component to the model's web component and create animated graphics like a model-controlled puppet.

2.2 Web components of Bodylight.js

Compiled FMU can be controlled using FMI API standard calls. However, Bodylight.js-Components[2] contains a set of components to simplify interactions among low-level FMI API, some standard HTML elements, third-party charting libraries, and 2D and 3D graphical animations.

The components are distributed as custom elements using standard WebComponent API (WebComponents 2021). It was developed using mainly Aurelia (AureliaJS 2023) framework, however, it can be used in any contemporary web application development framework or framework-agnostic way.

2.3 Changing user input, range web-component

The following sample web component defines HTML slider input and essential interaction (value change is sent as a custom HTML Event). The attributes can determine minimum, maximum, default value, and step by which the slider can change its value when moved right or left (Listing 1, Figure 5).

[2]Bodylight.js-Components `https://github.com/creative-connections/Bodylight.js-Components`

Listing 1. Bodylight Range Component with optional attributes (in blue), limiting user input between 40 and 180 with a step of 1 and default value 60

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

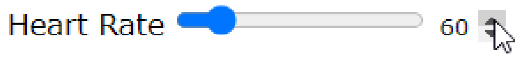


Figure 5. Range component rendered in a web browser

2.4 Control of simulation computation, FMI web-component

The following sample web component instantiates FMU from compiled JavaScript and creates standard HTML buttons to start/stop the simulation (Listing 2). When the simulation begins, a custom HTML event is sent to all potentially listening components. In every simulation step, a list of variable values is distributed as an array. The list of variables is set in `valuereferences` attribute. The components listed in `inputs` are listened to obtain interactively values the user changes during simulation.

The browser's `window.requestAnimationFrame()` method is used to call a simulation step. The browser usually calls this method up to 60 times per second to deliver a smooth user experience to match the refresh rate of the window as well as the performance of the viewing window. This call is usually paused in most browsers when running in background tabs.

Listing 2. Declaration of Bodylight FMI Component. Instantiates model of human pulsatile circulation dynamics from Physiolibrary(Mateják et al. 2014). Setup output values to be only pressure of pulmonary veins and arteries. Input is listened from an element with id1 and changed values are set as input to heartRate parameter which is multiplied by 1 and divided by 60 (converting 'per minute' to 'per second' unit expected by the model).

```
<bdl-fmi id="idfmi" src="
    Physiolibrary_Fluid_Examples_Fernandez
2013_PulsatileCirculation.js" fminame="
    Physiolibrary_Fluid_Examples_Ferna
ndez2013_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>
```

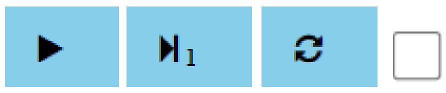


Figure 6. FMI component rendered in a web browser.

2.5 Charting web-components

Charts can make basic visualization of the data obtained from simulation. Bodylight.js library embeds open-source ChartJS (ChartJS 2021) library to support basic line charts using the component `<bdl-chartjs-time>`, see sample component listing in Listing 3. The component `<bdl-chartjs>` supports doughnuts, pie charts, and bar charts.

Listing 3. Bodylight Chart Component taking first one (indexed from 0) value of output values and converts it using expression $\frac{x}{133.322} - 760$ thus converting from Pa to mmHg and deducting ambient normal atmospheric pressure 760 mmHg

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

Initially the chart is empty, however, it is connected to the FMI component and listens to any data obtained from it and draws it interactively as seen in Figure 7. Bodylight.js-Components externally supports time series charts made by Plotly(Plotly 2021) and Dygraphs(Dygraphs 2021) libraries too.

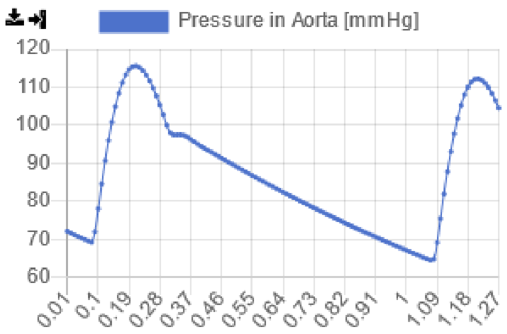


Figure 7. Chart component rendered in a web browser. This chart contains data obtained from FMI component during simulation from time 0 - 1.27s.

2.6 Interactive animation, adobe web-components

Adobe Animate is a multimedia authoring and computer animation program developed by Adobe Inc. Advanced visualization can be exported following as "standardized" open-source Javascript API (CreateJS 2023). By convention, an artist who creates interactive animation names all animatable elements with the suffix '_anim' and animation states between some values e.g. between 0 to 99 which visualizes the animation state. See the following listing (Listing 4).

Listing 4. Bodylight Animate component and components to bind animation element with model variable

```
<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.VentricleRight_anim"
    amin="100" amax="0" fmin="0.00012" fmax="0.00018"></
    bdl-bind2a>
```

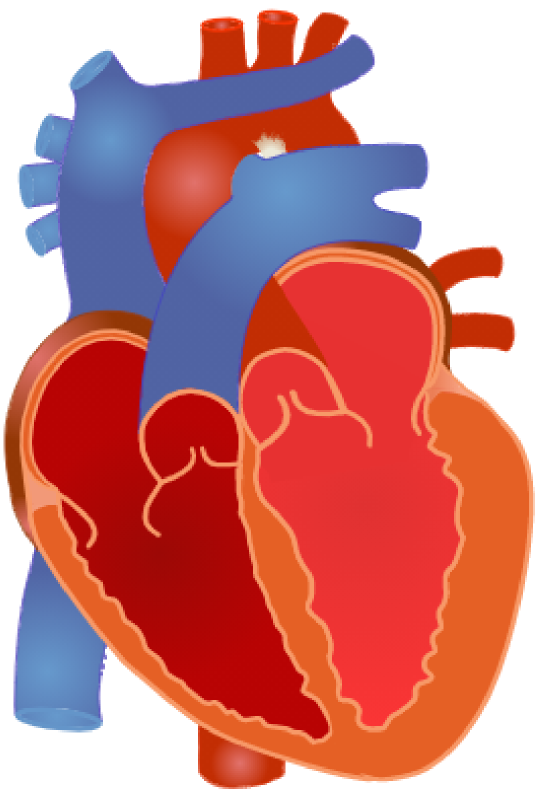


Figure 8. Animated component rendered in a web browser

2.7 Sample demo web simulator

All the previously defined component instances can be put into a single HTML page as seen in the following listing:

Listing 5. index.html containing declaration and use of web components. The aurelia.js framework was used to leverage building the web components thus the attribute 'aurelia-app' points out the DOM where web components can be located and corresponding implementation is injected there

```
<!DOCTYPE html>
<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>
  ...
</body>
</html>
```

The "index.html" must be published along the JavaScript file containing compiled FMU from the Modelica model: `Physiolibrary_Fluid_Examples_Fernandez2013_PulsatileCirculation.js`, JavaScript file containing published animation from Adobe Animate: `CardiaccycleStage.js` and "bodylight.js" library `bodylight.bundle.js`. However the Bodylight.js is published as NPM package and therefore can be taken from some content delivery network (CDN) caching NPM packages.

The resulting application is rendered in a web browser as seen in Figure 9.

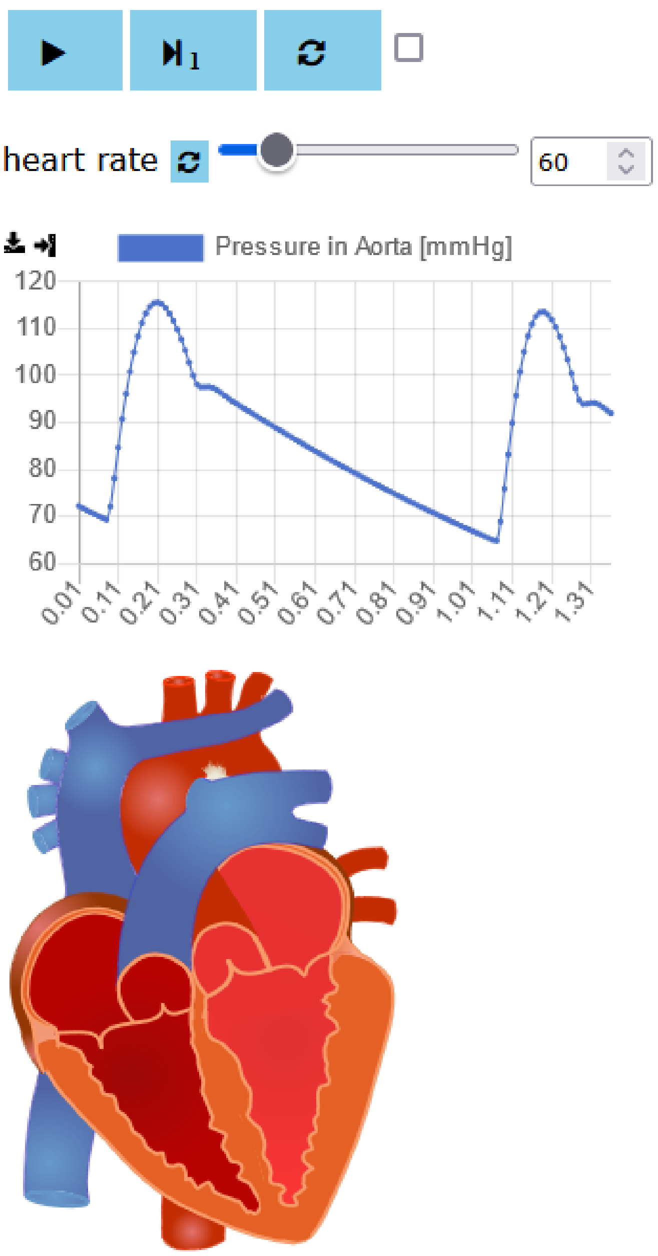


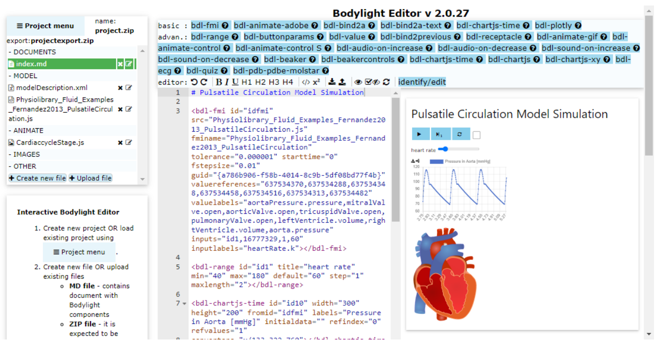
Figure 9. Web Simulator with rendered web components. The simulator can be started/restarted with buttons and the "heart rate" parameter can be changed by user interactivelly while computation of simulation is performed. Chart data is updated accordingly and animation is driven by the model variables.

2.8 Bodylight Editor

Optional tool Bodylight-Editor[3] is distributed as a static web page and allows a live preview of Markdown syntax as well as Bodylight.js-Components. Additional dialogs

[3]Bodylight-Editor `https://github.com/creative-connections/Bodylight-Editor`

facilitate filling the component attribute values, e.g., selecting input/output variables from the model and binding them into the appropriate component. The file management panel simplifies managing multipage documents sharing models, images, and animation, and generates multipage web simulators with shared navigation (Figure 10).


Bodylight Editor v 2.0.27
Project menu
name: project.zip
export: projectexport.zip
- DOCUMENTS
index.md
- MODEL
modelDescription.xml
Physiolibrary_Fluid_Examples_Fernandez2013_PulsatileCirculation.js
- ANIMATE
CardiaccycleStage.js
- IMAGES
- OTHER
Create new file
Upload file
basic : bdl-fmi bdl-animate-adobe bdl-bind2a bdl-bind2a-text bdl-chartjs-time bdl-plotly
advan.: bdl-range bdl-buttonparams bdl-value bdl-bind2previous bdl-receptacle bdl-animate-gif bdl-animate-control bdl-animate-control S bdl-audio-on-increase bdl-audio-on-decrease bdl-sound-on-increase bdl-sound-on-decrease bdl-beaker bdl-beakercontrols bdl-chartjs-time bdl-chartjs bdl-chartjs-xy bdl-ecg bdl-quiz bdl-pdb-pdbe-molstar
editor: B I U H1 H2 H3 H4 x² identify/edit
Pulsatile Circulation Model Simulation
heart rate
Pressure in Aorta [mmHg]

Interactive Bodylight Editor

1. Create new project OR load existing project using Project menu.
2. Create new file OR upload existing files
   - **MD file** - contains document with Bodylight components
   - **ZIP file** - it is expected to be

```
# Pulsatile Circulation Model Simulation

<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="637534370,637534288,637534348,637534458,637534516,637534313,637534482"
valuelabels="aortaPressure.pressure,mitralValve.open,aorticValve.open,tricuspidValve.open,pulmonaryValve.open,leftVentricle.volume,rightVentricle.volume,aorta.pressure"
inputs="id1,16777329,1,60"
inputlabels="heartRate.k"></bdl-fmi>

<bdl-range id="id1" title="heart rate"
min="40" max="180" default="60" step="1"
maxlength="2"></bdl-range>

<bdl-chartjs-time id="id10" width="300"
height="200" fromid="idfmi" labels="Pressure
in Aorta [mmHg]" initialdata="" refindex="0"
refvalues="1"
```

Figure 10. Bodylight Editor with the sample components above and rendered preview.

2.9 Bodylight Virtual Machine

Bodylight.js toolchain comprises several independent tools, some of which need non-trivial configuration. Therefore, we have created an exemplar virtual machine configuration for the Vagrant tool and virtualBox, to provision a standard minimal image of CENTOS Stream 9; scripts are published as Bodylight-VirtualMachine[4].

3 RESULTS

We compared the performance of model simulation translated to FMU executed natively with the implementation of the same model translated to FMU and WebAssembly and performed in a web browser on the same machine. We used Chrome browser version 97.0.4692.71 with simulation times of native code on the same platform (win-64) and performed a simulation that took 6000 steps. Natively it took an average of 9.3 s, while the simulation in the web browser took 34.5 s (1, column 'WASM 1 step').

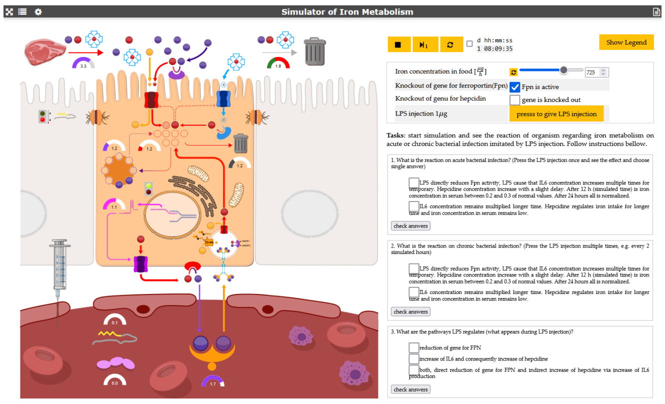
simulation	win64 bin	WASM (1 step)	WASM (3 steps)
time [s]	9.3s	34.5s	10.4s
relative [1]	1x	3.71x	1.12x

Table 1. Sample model simulation performance comparison between binary execution of FMU in win-64 and FMU translated to WASM and performed 1 or 3 FMU step() during web browser frame.

This difference might be explained by overhead due to the browser screen refresh framerate. Therefore we modified the WASM code to perform 2, 3, and 4 FMU step() calls during a frame given by the browser via requestAnimationFrame(). The browser allows max 60 frames per second when used and usually maintains a maximum—of thirty frames to support the smooth running of other apps and the operating system itself. Making more than three steps within one frame gave no better value (result not shown). Therefore in the following table, we offer times in column 'WASM 3 steps'. Thus, it can be concluded that the simulator's performance in WebAssembly (when doing multiple simulation steps during one frame) is comparable with native code (i.e., 1.12x or 12% slower than native code). This result also agrees with the more comprehensive benchmarks of WebAssembly vs. native code given by (Jangda et al. 2019).

We also measured the performance of the simulation with visualisation of charts and animation. It may significantly affect performance as the visualisation can update on each simulation step. Therefore we included configurable "throttle" property in order to do visual update only by default every 100 ms.

As all computation and rendering is done in the web browser, no interaction with a server is needed. The web simulator can be distributed as a static or server-less web page, e.g., using popular GitHub pages(GithubPages 2021). It can be utilized to distribute, e.g., digital appendices of scientific papers. It was already used by (Mazumder et al. 2023) using web simulator deployed at `https://filip-jezek.github.io/Ascites/` This way we published also first version of e-book "The Physiology of Iron metabolism" as seen in 11.



d hh:mm:ss
1 08:09:35

Iron concentration in food [$\frac{\mu g}{h}$]	725
Knockout of gene for ferroportin(Fpn)	[✓] Fpn is active
Knockout of genu for hepcidin	☐ gene is knocked out
LPS injection 1μg	presss to give LPS injection

Tasks: start simulation and see the reaction of organism regarding iron metabolism on acute or chronic bacterial infection imitated by LPS injection. Follow instructions bellow.

1. What is the reaction on acute bacterial infection? (Press the LPS injection once and see the effect and choose single answer)
   - ☐ LPS directly reduces Fpn activity, LPS cause that IL6 concentration increases multiple times for temporary. Hepcidine concentration increase with a slight delay. After 12 h (simulated time) is iron concentration in serum between 0.2 and 0.3 of normal values. After 24 hours all is normalized.
   - ☐ IL6 concentration remains multiplied longer time. Hepcidine regulates iron intake for longer time and iron concentration in serum remains low.

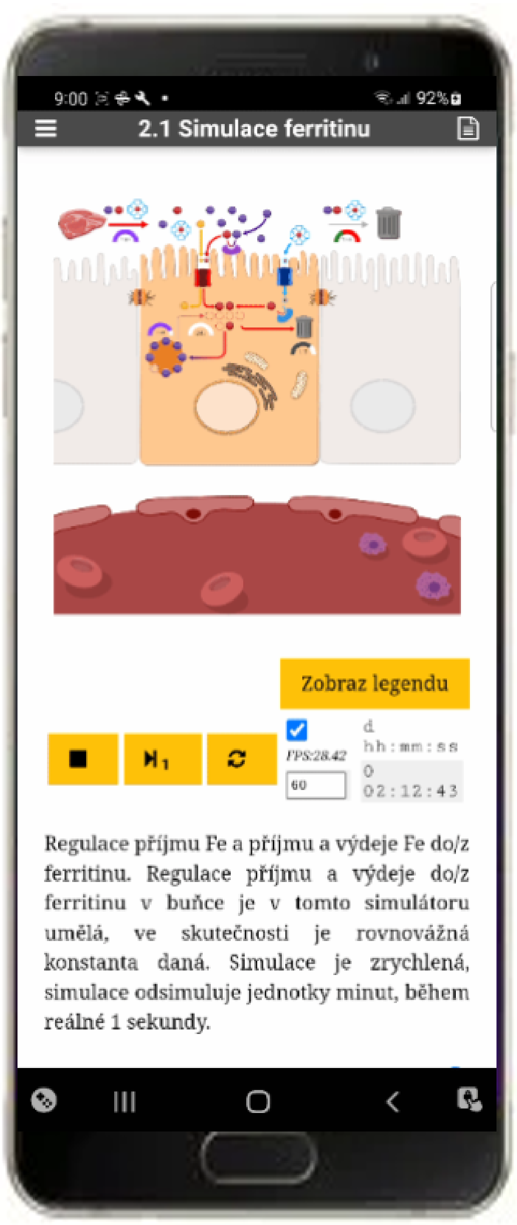
2. What is the reaction on chronic bacterial infection? (Press the LPS injection multiple times, e.g. every 2 simulated hours)
   - ☐ LPS directly reduces Fpn activity, LPS cause that IL6 concentration increases multiple times for temporary. Hepcidine concentration increase with a slight delay. After 12 h (simulated time) is iron concentration in serum between 0.2 and 0.3 of normal values. After 24 hours all is normalized.
   - ☐ IL6 concentration remains multiplied longer time. Hepcidine regulates iron intake for longer time and iron concentration in serum remains low.

3. What are the pathways LPS regulates (what appears during LPS injection)?
   - ☐ reduction of gene for FPN
   - ☐ increase of IL6 and consequently increase of hepcidine
   - ☐ both, direct reduction of gene for FPN and indirect increase of hepcidine via increase of IL6 production

Figure 11. Sample educational simulator of iron metabolism simulating gene knockout of hepcidin hormone resulting in iron overload in internal organs. It allows to enable/disable gene knockout, set a diet to affect illness and its treatment.

The same simulator can be converted also as a native mobile application e.g. by Apache Cordova (Apache Cordova 2023) tool. A sample in Figure 12 shows the digital textbook compiled as an Android application.

[4]Bodylight-VirtualMachine `https://github.com/creative-connections/Bodylight-VirtualMachine`



Regulace příjmu Fe a příjmu a výdeje Fe do/z ferritinu. Regulace příjmu a výdeje do/z ferritinu v buňce je v tomto simulátoru umělá, ve skutečnosti je rovnovážná konstanta daná. Simulace je zrychlená, simulace odsimuluje jednotky minut, během reálné 1 sekundy.

Figure 12. Educational simulator as native Android application.

A sample in Figure 13 shows a web simulator of blood gas exchange connected to the robotized virtual patient mannequin and controls his breathing. The following placed mockup of a medical device controls the extracorporeal membrane oxygenation (ECMO) process parameters. Such parameters are inputs to the model simulator connected via REST API, and the simulation shows direct feedback on user input and healthcare staff intervention to the patient state in graphs.

A sample in Figure 14 shows interactive 3D visualization of simplified human anatomy with charts of simulated hemodynamics. It leverages WebGL standard to visualize 3D objects and view the 3D scene. If this simulator is executed in a browser of a virtual reality device, then WebXR API is detected, and the simulator can be switched to an immersive view. This was tested on Oculus Quest 2 and MS Hololens 2.

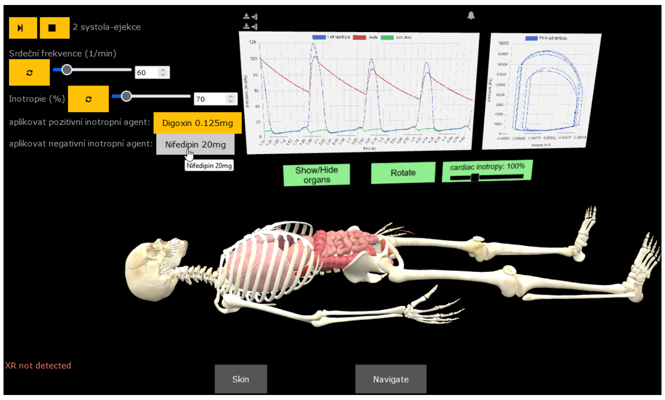


Figure 14. Sample educational simulator in 3D using WebGL and in immersive view for virtual reality using WebXR API. Virtual patient with simplified anatomy and physiology of cardiac hemodynamics and charts and controls are allowed to show the effect of drug treatment interactively.

Bodylight.js-Components is delivered using an open-source MIT License still and is still in the development stage depending on other open-source code [5]and the releases can be cited via Zenodo as (Kulhanek et al. 2023). The complete toolchain documentation and links are available `https://bodylight.physiome.cz`.

4 Discussion

Client-side simulation is appropriate for use cases where one or a few simulations must be performed. This is appropriate for interactive documents like educational materials, technical reports and digital appendices.

Client-side web-based simulation might not be appropriate for system analysis tasks like Monte-Carlo simulation.

The simulation is matched per `Window.requestAnimationFrame()` to the browser performance and is paused when the browser tab is in the background. The optimal inner FMU steps to refresh the framerate ratio have been established for a sample model; the optimal balance would vary though, depending on system performance bottlenecks and model complexity. Automatic adjustment based on the client's performance might be possible, but is currently not included in the development roadmap. The Web Workers (WebWorkers 2021) method might be more appropriate for another type of simulation (esp. for long-term models with higher memory demand etc.).

In the past, web-based simulators depended on non-standard, proprietary, but widely used plugins such as Adobe Flash player or Microsoft Silverlight. However, as technologies become obsolete (or even blocked), many older yet still scientifically relevant simulators cannot be executed on most modern computers or devices without excessive effort on virtualizing or emulating old operating systems and environments. We hope that using standard

[5]Bodylight.js-Components `https://github.com/creative-connections/Bodylight.js-Components`

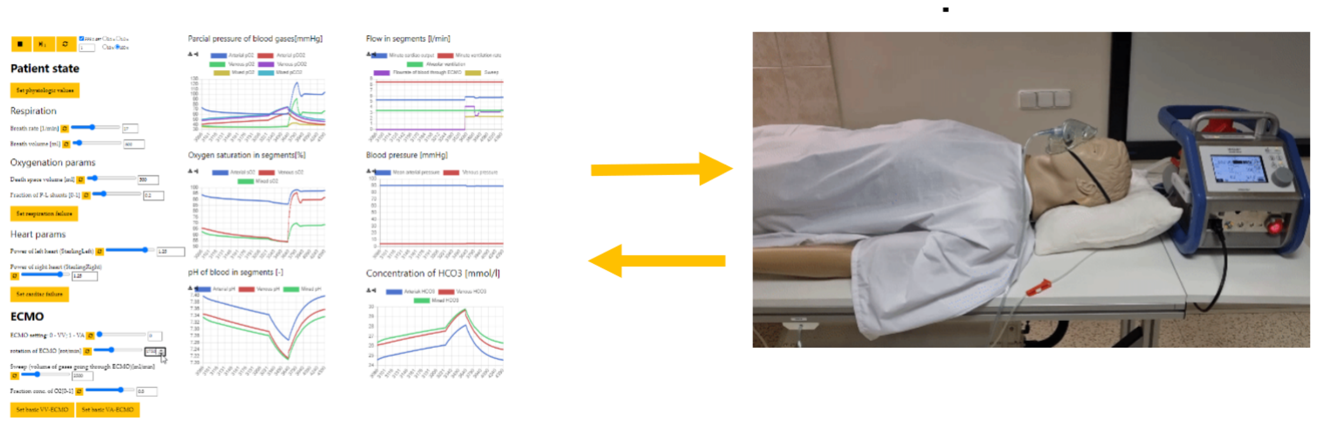

Patient state
Respiration
Oxygenation params
Heart params
ECMO
Parcial pressure of blood gases[mmHg]
Flow in segments [l/min]
Oxygen saturation in segments[%]
Blood pressure [mmHg]
pH of blood in segments [-]
Concentration of HCO3 [mmol/l]

Figure 13. Web simulator connected to hardware mannequin of virtual patient and mockup of the medical device. It communicates via REST API to show breathing and mockup of medical device (extracorporeal membrane oxygenator - ECMO) controls several model parameters. User input on this hardware-in-the-loop gives direct feedback in the connected simulator and visualization of breathing.

languages like Modelica, traditional execution models like FMI, and standard web API to build components may survive over a decade. Web simulators built from now on can be run in the future seamlessly.

Additionally, thanks to the widely accepted standards, the simulators can now be executed on various devices such as mobile phones, tablets, and virtual and augmented reality devices with no or very low code intervention. Bodylight.js library brings the missing piece and tools to integrate already existing standards and technologies between web publishing and mathematical modeling in Modelica.

Acknowledgements

Commercial potential is under further investigation, however, the main tools and libraries are free and open-source thanks to previous grants, funding and support by Creative Connections company, the MPO FV20628 and MPO FV30195 grants provided by Czech Ministry of Industry and Trade, and the Prague Smart Accelerator `CZ.02.2.69/0.0/0.0/18_055/0016956` provided by the City of Prague and partly financed from EU structural funds. We would like to thank to E. Benjamin Randall, Ph.D. for proofreading and grammar correction.

References

Apache Cordova (2023). *Open-source mobile development frameworkto use standard web technologies - HTML5, CSS3, and JavaScript for cross-platform development*. URL: https://cordova.apache.org (visited on 2023-01-06).

AureliaJS (2023). *Aurelia - Aurelia is a JavaScript client framework for web, mobile and desktop*. URL: https://aurelia.io (visited on 2023-11-08).

ChartJS (2021). *Simple yet flexible JavaScript charting for designers and developers*. URL: https://www.chartjs.org (visited on 2021-05-08).

CreateJS (2023). *CreateJS - A suite of modular libraries and tools which work together or independently to enable rich interactive content on open web technologies via HTML5*. URL: https://createjs.com/ (visited on 2023-11-08).

Dygraphs (2021). *Fast, flexible open source JavaScript charting library*. URL: https://dygraphs.com/ (visited on 2021-05-08).

Dymola (2023). *Multi-Engineering Modeling and Simulation based on Modelica and FMI*. URL: https://www.3ds.com/products-services/catia/products/dymola/ (visited on 2023-01-01).

EMScripten (2021). *EMScripten - complete compiler toolchain to WebAssembly*. URL: https://emscripten.org (visited on 2021-05-08).

Fernandez de Canete, Javier et al. (2013-05). "Object-oriented Modeling and Simulation of the Closed Loop Cardiovascular System by Using SIMSCAPE." In: *Computers in Biology and Medicine* 43.4, pp. 323–33. ISSN: 1879-0534. DOI: 10.1016/j.compbiomed.2013.01.007.

Franke, Rudiger (2014). "Client-side Modelica powered by Python or JavaScript". In: *the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*. DOI: 10.3384/ecp140961105.

Fritzson, Peter and et.al. (2019). "The OpenModelica Integrated Modeling, Simulation, and Optimization Environment". In: *Proceedings of The American Modelica Conference 2018, October 9-10, Somberg Conference Center, Cambridge MA, USA*. DOI: 10.3384/ecp18154206.

GithubPages (2021). *Websites for person and projects. Hosted directly from GitHub repository*. URL: https://pages.github.com/ (visited on 2021-05-08).

Jangda, Abhinav et al. (2019-07). "Not So Fast: Analyzing the Performance of WebAssembly vs. Native Code". In: *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. Renton, WA: USENIX Association, pp. 107–120. ISBN: 978-1-939133-03-8. URL: https://www.usenix.org/conference/atc19/presentation/jangda.

Kofránek, Jiı, Filip Ježek, and Marek Mateják (2019-02). "Modelica language - a promising tool for publishing and sharing biomedical models". In: *Proceedings of The American Modelica Conference 2018, October 9-10, Somberg Conference Center, Cambridge MA, USA*. Linköping University Electronic Press. ISBN: 9789176851487. DOI: 10.3384/ecp18154196. URL: http://dx.doi.org/10.3384/ECP18154196.

Kulhanek, Tomas et al. (2023). *creative-connections/Bodylight.js-Components:* version v2. DOI: 10.5281/zenodo.4575354.

Kulhánek, Tomáš et al. (2014). “Simple Models of the Cardiovascular System for Educational and Research Purposes”. In: *MEFANET Journal* 2.2, pp. 56–63. URL: http://mj.mefanet.cz/mj-04140914.

Library, Modelica Standard (2021). *Free (standard conforming) library from the Modelica Association to model mechanical (1D/3D), electrical (analog, digital, machines), magnetic, thermal, fluid, control systems and hierarchical state machines.* URL: https://github.com/modelica/ModelicaStandardLibrary/releases/tag/v4.0.0 (visited on 2021-05-08).

Matejak, Marek et al. (2015). “Free Modelica Library for Chemical and Electrochemical Processes”. In: *Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015*. 118. Linköping University Electronic Press, pp. 359–366.

Mateják, Marek et al. (2014). *Physiolibrary - Modelica library for Physiology*. Lund, Sweden. URL: https://www.physiolibrary.org.

Mazumder, Nikhilesh R et al. (2023). “Portal Venous Remodeling Determines the Pattern of Cirrhosis Decompensation: A Systems Analysis.” In: *Clinical and Translational Gastroenterology*. DOI: 10.14309/ctg.0000000000000590.

Plotly (2021). *Plotly JavaScript Open Source Graphing Library*. URL: https://plotly.com/javascript/ (visited on 2021-05-08).

Short, Tom (2014). *OpenModelica models in Javascript*. URL: https://github.com/tshort/openmodelica-javascript (visited on 2021-05-08).

Šilar, Jan, Filip Ježek, et al. (2019). “Model visualization for e-learning, Kidney simulator for medical students”. In: *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*. 157. Linköping University Electronic Press.

Šilar, Jan, David Polák, et al. (2019). “Development of In-Browser Simulators for Medical Education: Introduction of a Novel Software Toolchain”. In: *J Med Internet Res* 21.7, e14160. ISSN: 1438-8871. DOI: 10.2196/14160.

Tiller, Michael M. (2014). *Modelica By Example*. URL: https://mbe.modelica.university/.

WebComponents (2021). *Web components are a set of web platform APIs that allow to create new custom, reusable, encapsulated HTML tags to use in web pages and web apps.* URL: https://www.webcomponents.org/ (visited on 2021-05-08).

WebWorkers (2021). *Web Workers are a simple means for web content to run scripts in background threads*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers (visited on 2021-05-08).

Winkler, Dietmar and Michael Tiller (2017). “modelica. university: A platform for interactive modelica content”. In: DOI: 10.3384/ecp17132725. URL: https://ep.liu.se/ecp/132/079/ecp17132725.pdf.