DOI: 10.35191/medsoft_2026_1_38_kofranek_ll_en

SVELTE BODYLIGHT: A NEW TECHNOLOGY FOR BUILDING WEB SIMULATORS

Jiří Kofránek, Eduard Strauss, Michail Pugachov

Abstract

We build on Bodylight.js, the technology with which we have been creating web simulators for medical education since 2019. The new generation, Svelte Bodylight, keeps its principles: the model is computed in the web browser, with no server and no installation, and the application is assembled from standardised web components. What changes is how the application comes into being. Models written in Modelica are translated through the FMI standard into WebAssembly from Dymola as well as from OpenModelica, and every translation is compared automatically with a reference computation. The user interface is built with the open-source compiler Svelte as a set of web components: sliders, charts of time courses and of pressure–volume loops, state lamps and guided scenarios. A teacher assembles the application in a configurator without a programmer, and the finished simulator is delivered as a single HTML file that runs even without an internet connection.

Keywords

Modelica, FMI, WebAssembly, Svelte, web components, web simulators, medical education

1 Introduction: what stays from Bodylight.js and what changes

A medical simulator joins explanation with a mathematical model. The student changes parameters, watches how a model of a physiological system responds, and from the behaviour of the model understands why a disease shows itself the way it does. Building such a simulator, however, is teamwork. In our paper at MEDSOFT 2024 [1.] we described the four roles involved: the modeller, the artist, the teacher and the programmer. The modeller writes the model in the Modelica language, the artist draws the animations, the teacher designs the teaching scenario, and the programmer assembles the application from all of it (Fig. 1, left).

For the last step we have used the Bodylight.js technology since 2019 [2.–6.]. In it the model is translated into WebAssembly and computed directly in the browser, so the simulator needs neither a server nor an installation, and the application is assembled from standardised web components. All of this still holds. The bottleneck, however, remained the programmer: he assembled every application by hand, so every change of a scenario had to wait for him.

Svelte Bodylight moves this role. The teacher assembles the application in a configurator, and the programmer only creates new kinds of components – once, for every application (Fig. 1, right). This paper describes how it works: the path of a model from Modelica into the browser, the verification that the simulator computes the same as the original program, the components of the user interface, the experiment as a description instead of a program, the configurator, and the delivery of the whole set of simulators as a single file. The simulators themselves in use are shown in the companion paper in these proceedings [7.].

2 From Modelica to the browser

We write models of physiological systems in the Modelica language [8.]. It is an acausal language: the modeller writes down equations and relations between quantities as he knows them from physiology, and the compiler decides in which order, and which variable from which equation, to compute. From the model, an FMU (Functional Mock-up Unit) is then produced according to the FMI 2.0 standard [9.] – a self-contained computational unit that any program supporting the standard can run. We use the Co-Simulation variant, in which the FMU carries its own solver of differential equations (CVODE from the SUNDIALS suite [10.]) and outwardly only advances in time step by step. The FMU must contain source code in C, because that is what we then compile for the browser.

Compilation for the browser is done by Emscripten [11.], a compiler from C into WebAssembly [12.] – a binary format that today's browsers run at a speed close to native code. The result is two files: a WebAssembly module with the model and a small JavaScript file that loads it. Twenty seconds of a circulation simulation, that is 5,000 steps, are computed in about a tenth of a second; the rest of the time belongs to drawing the charts. The whole chain is summarised in Fig. 2.

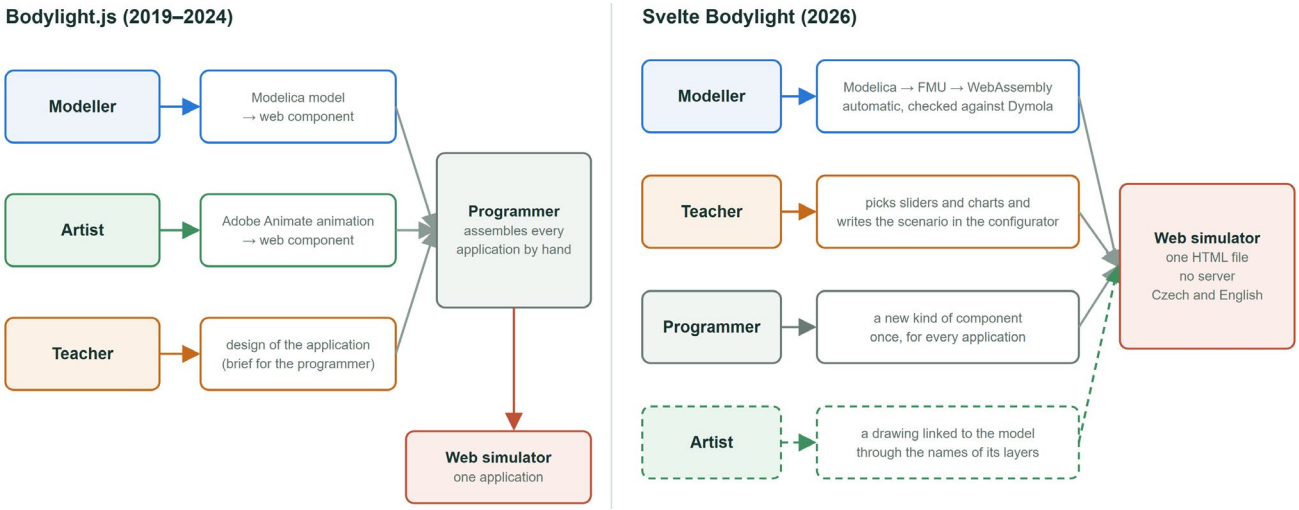


Figure 1 – roles in building a web simulator. Left, the Bodylight.js technology (2019–2024): the modeller, the artist and the teacher all hand their work to the programmer, who assembles each application separately, and the teacher's intention reaches the application only through him. Right, Svelte Bodylight (2026): the teacher assembles the application in the configurator, and the programmer extends the set of components, once for all applications. Dashed is the next step, not implemented yet – the artist's drawing linked to the model.

DOI: 10.35191/medsoft_2026_1_38_kofranek_ll_en

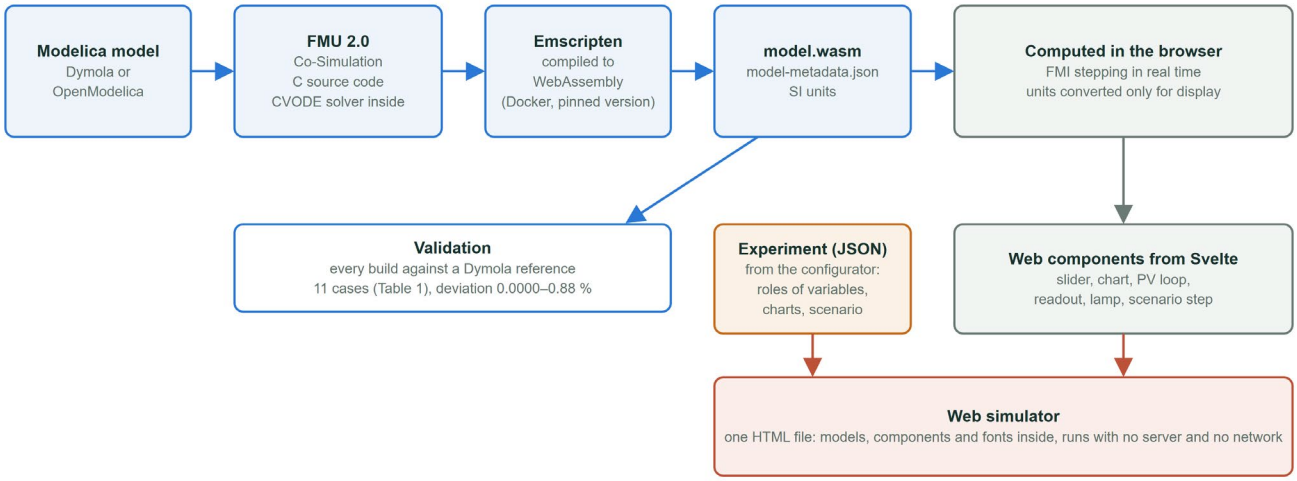


Figure 2 – from a model in Modelica to a page in the browser. In blue, the path of the model: from Dymola or OpenModelica as an FMU with source code in C, compilation by Emscripten into WebAssembly, and automatic verification against the reference computation in Dymola. In grey, the computation in the browser and the web components, shared by all models. In orange, the experiment, which the teacher decides. In red, the result the student gets: a single HTML file that runs with no server and no network.

We accept FMUs from two sources. The first is Dymola, in which most of our models were created. The second is OpenModelica [13.], a freely available environment; the aim is that the whole catalogue can be built even without a Dymola licence. The two sources behave slightly differently – OpenModelica, for instance, resolves events (the opening of a valve) exactly at the moments when the host advances it in time, so a pulsating model needs a finer step. The fineness of stepping is therefore a property of the model, carried by the description of the experiment, not a control for the student.

Inside the model everything is in SI units: pressure in pascals, volume in cubic metres, flow in cubic metres per second. Conversion to the units a physician reads – mmHg, ml, l/min – happens only on display. The model thus stays physically consistent, and units are a matter of the interface, not of the equations.

The compilation chain runs in a Docker container with a pinned version of the Emscripten compiler. The same model therefore gives the same result on any computer, and in a year the same as today – which is a precondition for verifying it to make sense.

3 Verification: the simulator computes the same as Dymola

Translating a model has many steps, and something can be lost in each of them. Every translated model is therefore compared automatically with a reference computation in Dymola: the same scenario is computed in the browser model, and the resulting time courses are compared with those from Dymola point by point. The comparison is part of every build; a change that would spoil the computation does not pass.

For most quantities the deviation is taken relative to the instantaneous value of the reference. For pulsating signals this is not possible: ventricular pressure passes close to zero during every beat, and a small absolute error would give a huge relative one there. For pulsating signals we therefore take the deviation relative to the range of the signal ("of range" in the table).

How small these deviations are is shown in Fig. 3 on the circulation model: left ventricular and aortic pressure from Dymola and from the simulator in the browser overlap completely in the chart, and their difference does not exceed 0.025 mmHg – a fraction of what could be read from any monitor.

model and scenario	largest deviation	tolerance
simple respiration, constant inputs, 60 s	0.0028 %	0.05 %
simple respiration, step change of an input	0.27 %	0.5 %
normal circulation, Dymola	0.0216 % of range	0.025 %
normal circulation, OpenModelica (original model)	0.0086 % of range	0.02 %
normal circulation, OpenModelica (package in SI units)	0.0117 % of range	0.02 %
respiration from the Bodylight library, OpenModelica	0.0025 %	0.05 %
simple respiration from an OPM diagram	0.0026 %	0.05 %
normal circulation from an OPM diagram	0.0157 % of range	0.02 %
Hodgkin–Huxley model at the excitation threshold	0.88 % of range	1 %
iron metabolism, steady state	0.0000 %	0.05 %
iron metabolism, hepcidin gene knocked out	0.12 % of range	0.5 %
iron metabolism, inflammation after lipopolysaccharide	0.013 % of range	0.1 %

Table 1 – comparison of the simulators in the browser with the reference computation in Dymola.

One entry deserves a comment. The nerve membrane model is compared deliberately right at the excitation threshold: a stimulus of 18 mV only bends the membrane, 19 mV elicits an action potential, and the case thus proves that the threshold lies exactly where it lies in Dymola. It is the most sensitive place possible, and the deviation of 0.88 % concerns the moment when the potential changes fastest.

The comparison also revealed errors that would otherwise have stayed hidden. One of them made the model silently stop

DOI: 10.35191/medsoft_2026_1_38_kofranek_ll_en

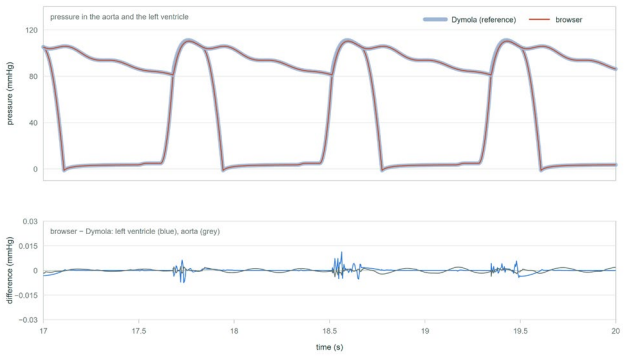


Figure 3 – verification of the simulator against Dymola on the circulation model, the last three seconds of a twenty-second run. Top, aortic and left ventricular pressure: the wide light line is the reference computation in Dymola, the thin dark line the simulator in the browser; the two overlap over the whole course. Bottom, their difference: within the three seconds shown at most 0.011 mmHg in the left ventricle and 0.004 mmHg in the aorta (0.024 and 0.007 mmHg over the whole run); it is concentrated in the moments when the valves open and close.

advancing in time after a while; another made the application write the set values into the model only before its initialisation, so the model then ran with its default values. Neither could be seen by looking at the charts; both were caught by the numerical comparison. For the iron metabolism model, the comparison showed that Dymola and the browser both computed the decaying lipopolysaccharide inaccurately, each in a different way: the model computed it as the difference of two nearly equal numbers. After the model was reformulated, they agree to within 0.013 %.

4 Web components from Svelte

We build the user interface with Svelte [14.], an open-source component compiler. Unlike common libraries, Svelte does not add an extensive runtime to the page; it compiles components into lean code that updates the page directly. The components then become standard web components (custom elements) [15.] – custom HTML tags that can be placed into any web page, regardless of what the rest of the page is written in.

The set has thirteen components today. The most important are:

- **slider with a number box** (<fmu-slider>): changes an input of the model; the value can also be typed, and once the slider has moved, it offers a way back to the default value;
- **chart** (<fmu-chart>): a time course with a history window, as on a bedside monitor, or a chart of one value against another – for example the left ventricular pressure–volume loop with the ESPVR and EDPVR lines (Fig. 4);
- **readout tile** (<fmu-readout>): a number with its unit, as a monitor shows it;
- **state lamp and button** (<fmu-state>, <fmu-trigger>): for qualitative models, in which states change rather than numbers – the lamp is lit when the system is in the given state, the button triggers an event;
- **scenario step** (<fmu-step>): text for the student, Back and Next buttons, and a button that sets the values of the step.

The remaining components control the run (start, pause, new run), show the state and time of the simulation, and connect everything to one model (<fmu-experiment>). What matters is that every component exists only once. The same set serves the application, the configurator and the standalone page, so they cannot drift apart – a fix in one place holds everywhere.

The simulation itself does not belong in the components. The order of calls of the FMI standard, stepping in time, keeping the pace and converting units are the job of a single layer with no tie to the interface, used by all pages. In the application the model is computed in a Web Worker thread so as not to slow the interface down; in the standalone page on the main thread, because the environments in which the page is shared may forbid a separate thread. The layer therefore makes sure that computing and drawing keep the promised pace even on a slower computer: when drawing the charts becomes expensive, they are drawn less often, but the simulation keeps running in real time.

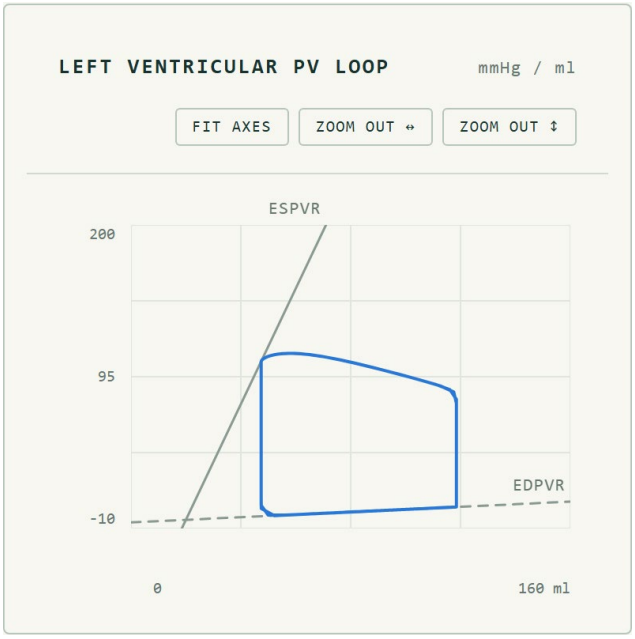


Figure 4 – feft ventricular pressure–volume loop in the normal circulation simulator. The slanted solid line is the ESPVR (end-systolic pressure–volume relationship), the dashed curve the EDPVR (end-diastolic pressure–volume relationship). The buttons above the chart fit the axes to the loop currently shown or zoom out, so that when parameters change one can follow where the loop moves.

5 The experiment as a description, not a program

We do not program the application on top of a model; we describe it. The description is called an experiment, and it is a JSON file. It states which variables of the model are sliders and over what range, which are fixed parameters, which are only displayed and in which units, which charts are drawn and from which curves, how long and how fast the simulation runs, and what the scenario is. The model itself does not change for the experiment; there may be any number of experiments on the same model. On the circulation model we have eight of them today – from normal circulation through ventricular failure, valve defects and haemorrhage to intra-abdominal pressure and anti-shock trousers.

A scenario is a series of steps. Each step has a text for the student, the values it sets, and a flag saying whether it starts again – like a new patient – or continues where the previous

DOI: 10.35191/medsoft_2026_1_38_kofranek_ll_en

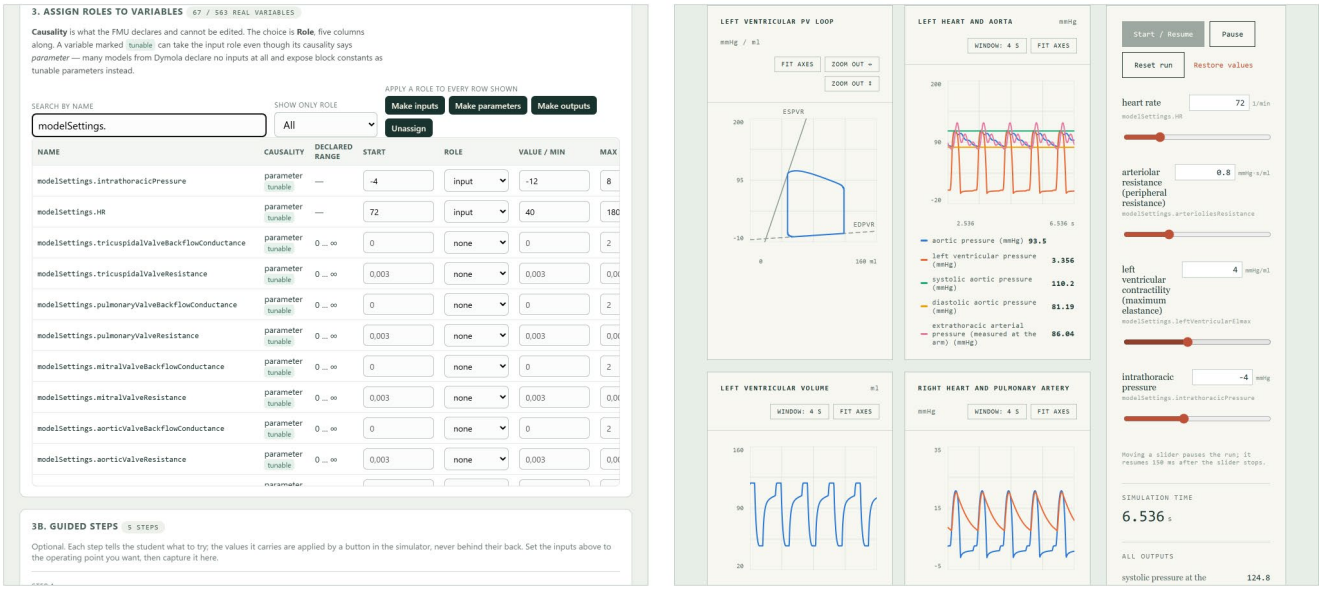


Figure 5 – the configurator (left) and the application it drives (right). In the configurator the teacher assigns roles to the variables of the circulation model – an input controlled by a slider, a fixed parameter, a displayed output – and sets their ranges; on the right runs the real application with the charts and sliders that came out of these choices.

one ended, as with a haemorrhage that progresses. The numbers in scenario texts are not estimated: they are measured on the running model and rounded as the page shows them, and a tool compares them with what the student will actually see on the page. Among other things, this found steps that should have started again but continued from the end of the previous step, and therefore showed different numbers from those the text promised.

One experiment carries both language versions. The text is stored in Czech and in English, but numbers, ranges and settings are shared, so the two versions cannot diverge in content. The Czech page writes a decimal comma, the English one a decimal point; an automatic check verifies that the two versions of the same page differ only in words.

Besides the generic page that the application builds from the experiment by itself, an experiment can also have a designed page – an HTML file in which an artist or a teacher arranges the elements at will and merely labels them with variable names. This is how, for example, the respiration "cockpit" came about, or a live diagram in which the values of the model are shown directly in the nodes of the scheme, or the scheme of iron metabolism drawn after a sketch by the model's author, with live values at the organs and at the places where hepcidin and interleukin 6 block ferroportin.

6 The configurator: the teacher assembles the application

The experiment is not written by hand. The teacher builds it in the configurator: loads the description of the model, chooses for each variable whether it will be a slider, a fixed parameter or a displayed output, selects units, divides the outputs into charts, sets the length and pace of the simulation and writes the scenario. The configurator has no simulation of its own – it drives the real application in the neighbouring window, so the teacher sees exactly what the student will see, and a trial run verifies that the computer keeps up with the chosen pace (Fig. 5).

The configurator also guards against errors that would otherwise only show in class: a variable the model does not have, an input that cannot be changed, or a default slider value that does not lie on its scale.

7 Delivery: one file, no server

The whole set of simulators is built into a single HTML file. It contains the models in WebAssembly, the components, the descriptions of the experiments and the fonts; the file is about 6.5 MB and works without an internet connection – it is enough to open it in a browser, for example in a lecture hall without a network, or to put it on any web server. Today it contains 16 models and 23 experiments (Fig. 6).

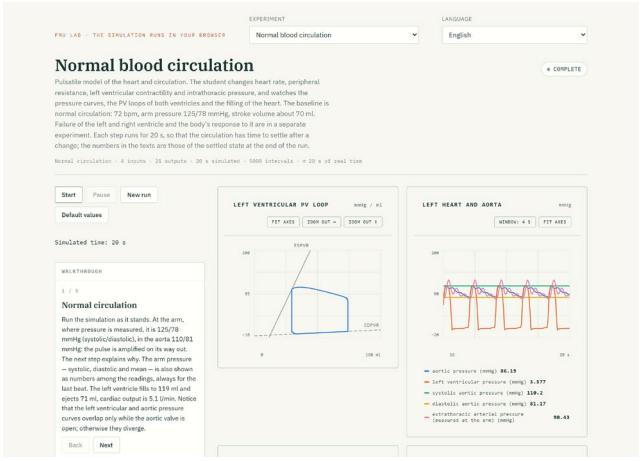


Figure 6 – the shared page with the normal circulation simulator after a finished run: choice of experiment and language, description, run controls, the first step of the scenario, and charts – the left ventricular pressure–volume loop and pressures in the left heart and the aorta with readouts. The whole set of simulators is a single HTML file that works even without an internet connection.

The page is guarded by a series of automatic checks. They verify that it connects nowhere and takes its fonts from itself, that every experiment shows something and responds to the controls, that the two language versions differ only in words, that the page works in the Chrome and Firefox browsers and in the WebKit engine on which Safari is built, that it fits on the display of a mobile phone, and that the experiments keep the promised pace even on a computer twice as slow as the one they were made on. The last two checks were added when it turned out that on a phone the page was rendered shrunk to

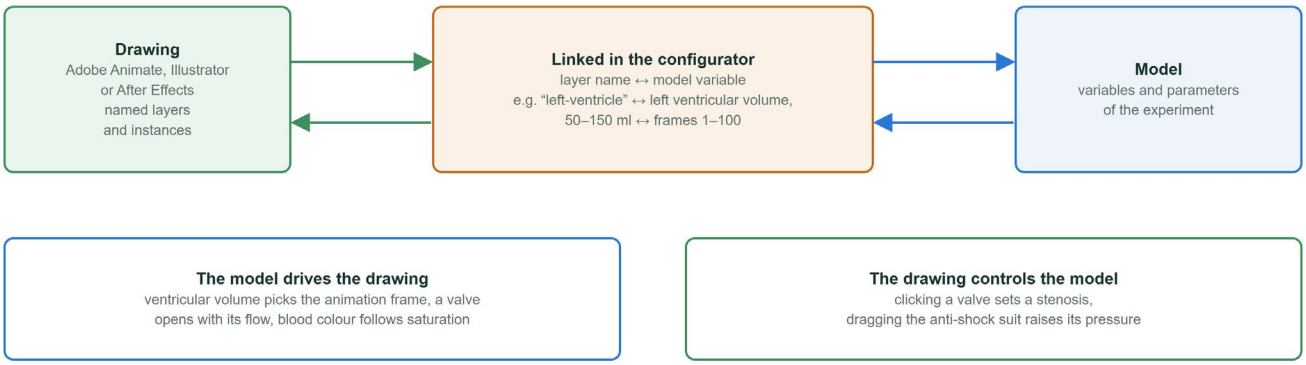


Figure 7 – proposed next step: the artist's drawing driven by the model. A drawing from Adobe Animate, Illustrator or After Effects carries named layers and instances, which are linked to the variables of the model in the configurator. The model then drives the drawing (ventricular volume picks the frame of the animation, a valve opens with its flow), and the drawing controls the model (clicking a valve sets its stenosis). The artist programs nothing; the names of the layers are the contract between the drawing and the model. Not implemented yet.

a third, and that on a slower computer the circulation model could run several times slower than it promised – neither could be seen on the development computer.

8 Links to the running simulators

As in our 2024 paper, readers can try the simulators themselves: a link opens the simulator in their browser. A link can open a single experiment and language directly – for example *https://bodylight.physiome.cz/SvelteBodylight.js/medsoft2026/fmu-lab.html?experiment=normal-circulation/krvaceni&lang=en* opens the haemorrhage scenario in English; with &lang=cs in Czech, and with &view=opd, for a model made from an OPM diagram, directly the diagram. The page itself keeps in the address bar what is currently shown, so a link to any view can simply be copied. Links to all experiments are given in the companion paper [7.]. An overview of all simulators with links is at *https://bodylight.physiome.cz/SvelteBodylight.js/en.html*.

9 Next step: the artist, and models from diagrams

Svelte Bodylight still lacks the role that was most visible in Bodylight.js: the artist. We propose that a drawing made in a tool the artist knows – Adobe Animate, Illustrator or After Effects – should carry names of layers and instances, and that these be linked to the variables of the model in the configurator (Fig. 7). Ventricular volume would then pick the frame of the heartbeat animation, a valve would open with its flow, and the colour of the blood would follow saturation. And the other way round: clicking a valve could set its stenosis. The artist would write no code; the names of the layers would be the contract between the drawing and the model. This part is not implemented yet.

The same chain already simulates models that were not written in Modelica but created from diagrams of the Object-Process Methodology (OPM, standard ISO 19450 [16.]). A Modelica model is generated from the diagram and takes the same path into the browser; Table 1 contains two such models, and their deviations from the references are as small as those of models written by hand. The details will be the subject of a separate paper.

10 Conclusion

Svelte Bodylight keeps what proved itself in Bodylight.js – the Modelica, FMI and web component standards, computation in the browser with no server, and teamwork – and changes what was the bottleneck. The application on top of a model is no longer assembled by a programmer but by the teacher, in a configurator that shows the real running application. Every translated model is verified automatically against the original computation, and the finished set of simulators is a single file that works even without a network. The path from a model to a teaching application thus shortens from weeks to hours – the iron metabolism simulator had its first experiment with hepcidin controls ready on the same day the model was delivered – and moves to the person who knows what the students should learn. What such simulators look like in teaching is shown in the companion paper in these proceedings [7.].

Acknowledgements

The project was funded by Creative Connections s. r. o. and by the Grant Agency of Charles University.

SVELTE BODYLIGHT: NOVÁ TECHNOLOGIE TVORBY WEBOVÝCH SIMULÁTORŮ

Abstrakt

Navazujeme na technologii Bodylight.js, se kterou jsme od roku 2019 vytvářeli webové simulátory pro výuku medicíny. Nová generace, Svelte Bodylight, zachovává její principy: model se počítá v internetovém prohlížeči, bez serveru a bez instalace, a aplikace se skládá ze standardizovaných webových komponent. Mění se způsob, jakým aplikace vzniká. Modely z jazyka Modelica se přes standard FMI překládají do WebAssembly z Dymoly i z OpenModeliky a každý převod se automaticky porovnává s referenčním výpočtem. Uživatelské rozhraní vzniká v open-source překladači Svelte jako sada webových komponent: posuvníky, grafy průběhů i tlakově-objemových smyček, kontrolky stavů a řízené scénáře. Pedagog sestaví aplikaci v konfigurátoru bez programátora a hotový simulátor se doručí jako jediný soubor HTML, který běží i bez připojení k internetu.

Klíčová slova

Modelica, FMI, WebAssembly, Svelte, webové komponenty, webové simulátory, lékařská výuka

References

[1.] *Kofránek J, Kulhánek T. Vysvětlující počítačové modely (explanatory models) v lékařské výuce [Explanatory models in medical education]. Medsoft. 2024;36(1): 1–14. Czech. doi:10.35191/medsoft_2024_1_36_kofranek*

[2.] *Šilar J, Polák D, Mládek A, Ježek F, Kurtz TW, DiCarlo SE, et al. Development of in-browser simulators for medical education: introduction of a novel software toolchain. J Med Internet Res. 2019;21(7): e14160. doi:10.2196/14160*

[3.] *Polák D, Ježek F, Šilar J, Kofránek J. Technologie tvorby webových simulátorů [Technology for building web simulators]. Medsoft. 2019;31: 122–139. Czech. Available from: https://www.medsoft.website/sbornik/2019/Medsoft_2019_Polak_Print.pdf*

[4.] *Kulhánek T, Mládek A, Brož M, Kofránek J. Bodylight.js web components – webové komponenty pro webové simulátory [Bodylight.js web components for web simulators]. Medsoft. 2021;33(1): 48–52. Czech. doi:10.35191/medsoft_2021_1_33_48_52*

[5.] *Kofránek J, Kulhánek T. Standardizace – cesta k open source technologiím pro webové simulátory [Standardisation – a path to open-source technologies for web simulators]. Medsoft. 2021;33(1): 25–34. Czech. doi:10.35191/medsoft_2021_1_33_25_34*

[6.] *Kulhánek T, Mládek A, Kofránek J, Ježek F. Bodylight.js 2.0 – Web components for FMU simulation, visualisation and animation in standard web browser. In: Proceedings of the 15th International Modelica Conference 2023, Aachen, October 9–11. Linköping Electronic Conference Proceedings 204; 2023. p. 443–452. doi:10.3384/ecp204443*

[7.] *Strauss E, Pugachov M, Kulhánek T, Kofránek J. Web simulators in medical education. Medsoft. 2026. (these proceedings)*

[8.] *Modelica Association. Modelica – A Unified Object-Oriented Language for Systems Modeling. Language Specification, version 3. 6. 2023. Available from: https://specification.modelica.org/maint/3.6/MLS.html*

[9.] *Modelica Association. Functional Mock-up Interface for Model Exchange and Co-Simulation, version 2. 0. 2014. Available from: https://fmi-standard.org*

[10.] *Hindmarsh AC, Brown PN, Grant KE, Lee SL, Serban R, Shumaker DE, et al. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. ACM Trans Math Softw. 2005;31(3): 363–396. doi:10.1145/1089014.1089020*

[11.] *Zakai A. Emscripten: an LLVM-to-JavaScript compiler. In: Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications. 2011. p. 301–312. doi:10.1145/2048147.2048224*

[12.] *Haas A, Rossberg A, Schuff DL, Titzer BL, Holman M, Gohman D, et al. Bringing the web up to speed with WebAssembly. In: Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2017. p. 185–200. doi:10.1145/3062341.3062363*

[13.] *Fritzson P, Pop A, Abdelhak K, Ashgar A, Bachmann B, Braun W, et al. The OpenModelica integrated environment for modeling, simulation, and model-based development. Model Identif Control. 2020;41(4): 241–295. doi:10.4173/mic.2020.4.1*

[14.] *Svelte. Available from: https://svelte.dev*

[15.] *WHATWG. HTML Living Standard – Custom elements. Available from: https://html.spec.whatwg.org/multipage/custom-elements.html*

[16.] *ISO 19450:2024. Automation systems and integration – Object-Process Methodology. Geneva: ISO; 2024.*

Contact

Assoc. Prof. Jiří Kofránek, MD, PhD
Department of Biocybernetics
Institute of Pathological Physiology
First Faculty of Medicine
Charles University
U nemocnice 5, 128 53 Prague 2
Czech Republic
+420 777 68 68 68
kofranek@gmail.com

Eduard Strauss, MD
Department of Biocybernetics
Institute of Pathological Physiology
First Faculty of Medicine
Charles University
U nemocnice 5, 128 53 Prague 2
Czech Republic
strausseda@gmail.com

Michail Pugachov, MSc
Department of Biocybernetics Institute of Pathological Physiology First Faculty of Medicine
Charles University
U nemocnice 5, 128 53 Prague 2
Czech Republic
mpugachov@gmail.com