Model visualization for e-learning Kidney simulator for medical students

Jan Šilar[1] Filip Ježek[1,2] Arnošt Mládek[1] David Polák[1] Jiří Kofránek[1]

[1]Institute of Pathological physiology, First Faculty of Medicine, Charles University, Prague, Czech republic,
{jan.silar, filip.jezek, arnost.mladek, david.polak, jiri.kofranek}@lf1.cuni.cz
[2]Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic

Abstract

The present paper introduces a recently developed tool for building web-based simulators called *Bodylight.js*. Simulators are applications composed of a mathematical model and a graphical user interface that allows the user to easily interact with the model and visualize the results. A modelica model is first exported to FMI with sources, transcompiled into JavaScript and WebAssembly and connected to a GUI, comprised of graphical animations created in Adobe Animate and elements that allow to control the input model such as sliders, buttons, etc.

A physiological e-learning application explaining the function of a nephron – the basic functional unit of kidneys – is presented later as a use-case. The model was developed primarily as a teaching aid for use in courses of physiology for medical students at our university.

Purpose of this work is to describe the new Bodylight.js tool and to prove its usability by building the medium-complex e-learning kidney simulator. The simulator helps medical students to better understand renal function at the very basic level.

Keywords: Modelica, JavaScript, WebAssembly, web technologies, physiology, kidney, e-learning

1 Introduction

Mathematical models are powerful tools for gaining insight into the systems under study. It is sometimes feasible to experiment with a human body directly. For example a man can drink a lot of water and, due to the enhanced urine production, has to urinate sooner. But this is just an outer behavior of the system. It is not so easy to grasp the underlying mechanisms: how was the swallowed water absorbed into the intestine blood vessels? Why the blood did not get diluted? Further, how is the primary urine produced? Here models and simulation applications may help us to comprehend the underlying mechanisms. Functional models are already being used in medical education (Kofranek *et al,* 2011) (namely in physiology and pathophysiology). But we believe that the benefits are still widely underestimated and that illustrative models should be used regularly in lectures and practical classes.

Our ultimate goal is to produce simulation applications for teaching physiology that both look and behave like the simulated system so that they are as much understandable as possible. We need a modeling tool, graphical animation tool and tool to connect both the model and the animation together.

In physiological modeling, a number of modeling tools are used – e.g. Mathworks Matlab/Simulink, CellML, JSim, OpenModelica etc – and each one requires installation and at least some familiarization with the tool to be able to run the models. Some tools even require a (very expensive) commercial license. To overcome this, a standalone simulator is required, preferably without the need of installing anything. Web-based technologies do offer a convenient solution (Kofránek *et al,* 2009) and allow the simulator applications to be accessed as simply as the rest of the contemporary world-wide web.

However, development of a simulator is often a demanding task. Some effort has been put into development of web-based simulators, e.g. the proprietary Modelica.university (Tiller and Winkler, 2017) or the Bodylight framework (Ježek *et al,* 2013), based on the discontinued (Smith, 2015) Microsoft Silverlight.

Some researchers (e.g. (Christ and Thews, 2016; Kulhánek *et al,* 2013; Zhang, 2001) and a number of others) aimed at a client-server simulation. Such solution relies on a server which performs the computation and client only receives the resulting data. Based on user input, the client asks for a new set of data. The second possible approach is fully client-side, in which the client is responsible for both the computation and user interaction see Figure 1.

The client-side concept is initially more demanding task, as the whole calculation has to be performed in a web-enabled language, i.e. JavaScript, it however offers some advantages. Especially for educational purposes, the server does not have to bear entire classroom's computational load at the same time. The requirement of a smooth visualization, including continuous simulation graphing, movement of animated components and prompt interaction therefore prefers the client-side approach. Although the usage of a modern cloud technologies with scalable computational power and decentralized geographical location would reduce the client-server lag to satisfying levels, the price of the infrastructure is substantially higher and scales up with any new user. Of course, very computationally demanding simulators are not meant to be client-simulated, but those are out of scope of the discussed physiological models.

As of 2018, no open web-based simulator platform which is capable of running complex equation-based models exists. Our aim is to develop a client-side simulator technology, based on the chosen Modelica language and a simulator-producing toolchain. This technology has been named Bodylight.js

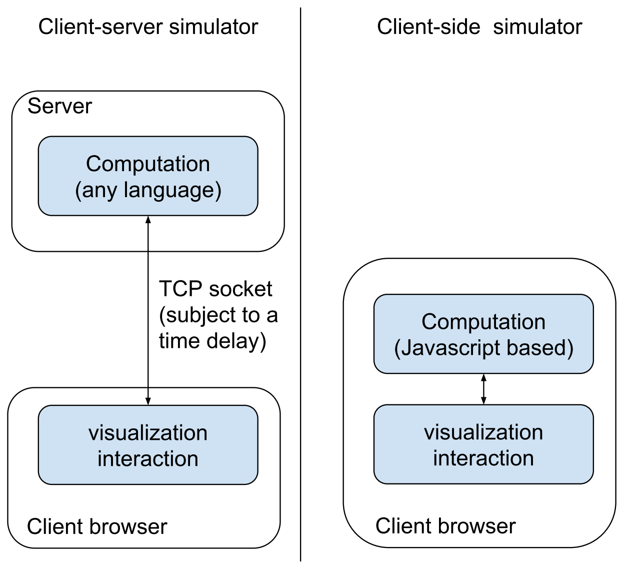
Client-server simulator
Client-side simulator
Server
Computation (any language)
TCP socket (subject to a time delay)
visualization interaction
Client browser
Computation (Javascript based)
visualization interaction
Client browser

Figure 1. Client server and Client only architectures.

2 Methods

Our group (Kofranek's group) develops medical e-learning simulators since 1996. We have focused on web-based simulators since 2012 (Ježek *et al,* 2012). After designing a set of simulators (a sample is shown on Figure 2), based on the custom Bodylight framework, built on a Microsoft Silverlight web technology (Ježek *et al,* 2013), the core Silverlight platform has been discontinued (Smith, 2015). Lessons learned – do not rely on proprietary platforms.

2.1 The Bodylight.js build process

The effort has been recently restarted, and consequently the approach has been based on open standards:

- Modelica language for modeling (Fritzson and Engelson, 1998)
- Functional Mockup Interface for model simulation
- HTML5 + JavaScript for model presentation and interaction

Driven by industrial needs to share and co-simulate models of various languages and tools, the Functional Mockup Interface (FMI) (Blochwitz *et al,* 2012) emerged as an open standard. Developed and maintained by the Modelica association (Lund, Sweden), it quickly gained wide support from tool vendors.

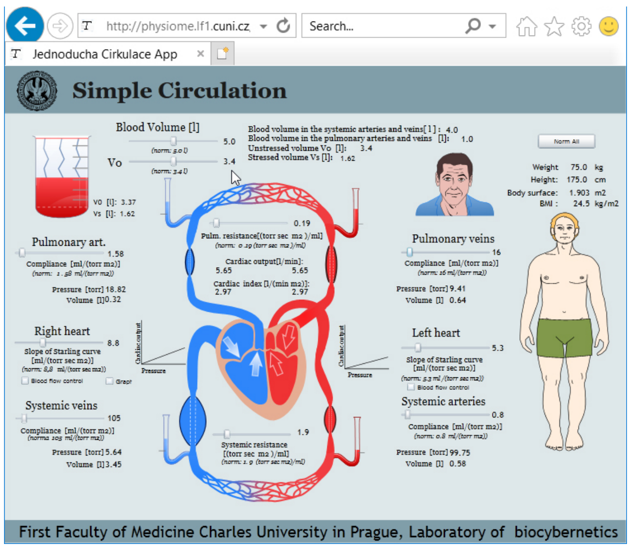

Figure 2. The simulator of simple circulation, built using the Silverlight technology (Tribula *et al,* 2013).

As of September 2018, 110 tools are capable of either FMI export, import or both (Modelica Association, 2017).

In first stage of our work-flow the model is exported as FMU for Co-simulation version 2.0 including source code. The advantage of using FMI is the standardization, which ensures further compatibility of export from multiple tools and their future versions.

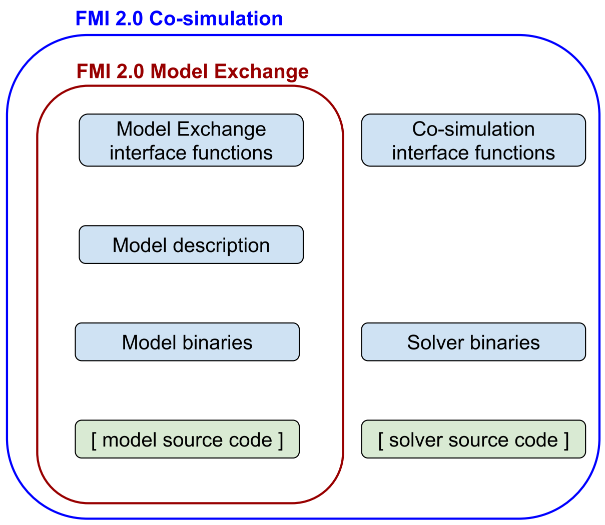

Figure 3. Content of FMU can vary, depending on usage (simplified).

The task is to get the FMU into JavaScript, so it can run in the browser. As shown in Figure 3, the FMU can contain source code of both the model and the solver. The C code could be then translated to JavaScript using Emscripten (Zakai, 2011). The Emscripten translation offers two targets: ASM.JS and WebAssembly (or WASM). Asm.js is a turing-complete subset of the JavaScript language, used as a compilation target. WebAssembly is currently a more effective binary version of Asm.js, but it is designed to "*Define* *a portable, size- and load-time-efficient binary* format *to serve as a compilation target which can* be compiled to execute at native speed by taking advantage of common hardware capabilities available on a wide range of platforms, including mobile and IoT" (WebAssembly High-Level Goals – WebAssembly, n.d.). The model compilation to a binary format effectively obfuscates the model code, so this method is suitable for proprietary or undisclosable models as well. The translated FMU code is then linked to model controls (such as start, stop, parameters input etc), graphs and animated components.

For educational purposes especially in non-technical fields, the value of a graph alone is usually not enough. Simulators should provide rich content, including images and animations controlled by the model's output. Thus, the animation components are designed and animated in Adobe Animate and then exported as an HTML component, exposing their animation time-lines as Javascript functions, which are linked to the model. The animations time-lines could be nested, so it is possible to animate e.g. width and height of a component independently, but the animations have to be stackable, that is they are not truly independent. The whole web-simulator build process is visualised in Figure 4.

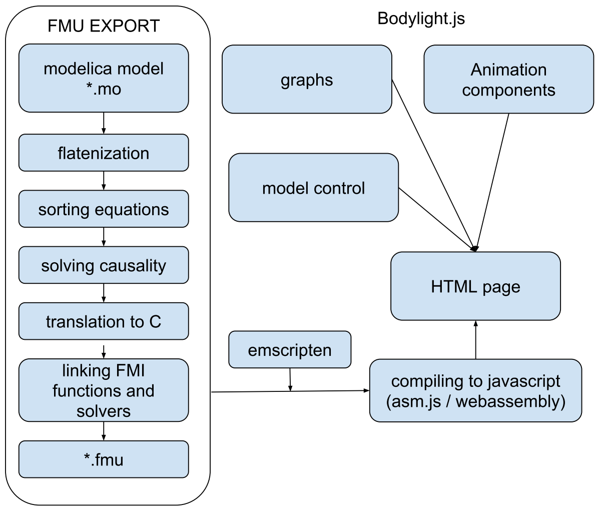

Figure 4. FMU build process. The FMU is exported and the packaged sources are translated into the JavaScript to enable web-simulation.

2.2 The Bodylight.js Composer

As illustrated by Figure 5, the development of an educational simulator is a multi-disciplinary task (Kofránek *et al,* 2009):

- The domain expert (teacher) sets the simulator objectives and designs a simulation scenario
- The modeler develops and implements the mathematical model
- The graphic designer draws and animates the components and prepares the layout
- The integrator composes the simulator together.

In fact, the simulator integration could be simplified to a bare minimum – all inputs are already known and prepared, thus it is only necessary to interconnect the controls and graphical components with the model inputs and outputs. And that could be mostly automatized. Therefore, to make the simulator development more efficient, a special helper composer tool has been developed.

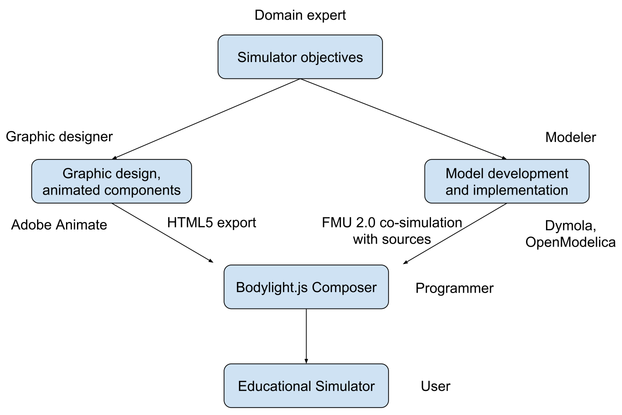

Figure 5. Simulator design and build process scheme.

The composer allows to upload the FMU, manage the model settings, upload animations and other graphical components, insert graphs and HTML controls, and interconnect it all together and then export a standalone HTML5 application.

3 Applications

The Simple Circulation simulator implemented earlier in Silverlight (Figure 2) was already reimplemented[1] using the new technology.

Another new simulator on iron regulation[2] was implemented, see Figure 6. Here, we have implemented a mathematical model of systemic iron regulation based on the work of Enculescu et al. (Enculescu *et al,* 2017). The model incorporates dynamics of organ iron pools as well as regulation by the hepcidin/ ferroportin system. The model was calibrated and validated with time-resolved measurements of iron responses in mice challenged with dietary iron overload and/or inflammation. The model demonstrates that inflammation mainly reduces the amount of iron in the bloodstream by reducing intracellular ferroportin transcription, and not by hepcidin-dependent ferroportin protein destabilization. In contrast, ferroportin regulation by hepcidin is the predominant mechanism of iron homeostasis in response to changing iron diets for a big range of dietary iron contents.

3.1 Nephron simulator

A nephron simulator[3] was implemented recently. Nephron is the structural and functional unit of the

[1] at www.physiome.cz/apps/SimpleCirculation/
[2] at www.physiome.cz/apps/IronMetabolism
3 at www.physiome.cz/apps/Nephron/

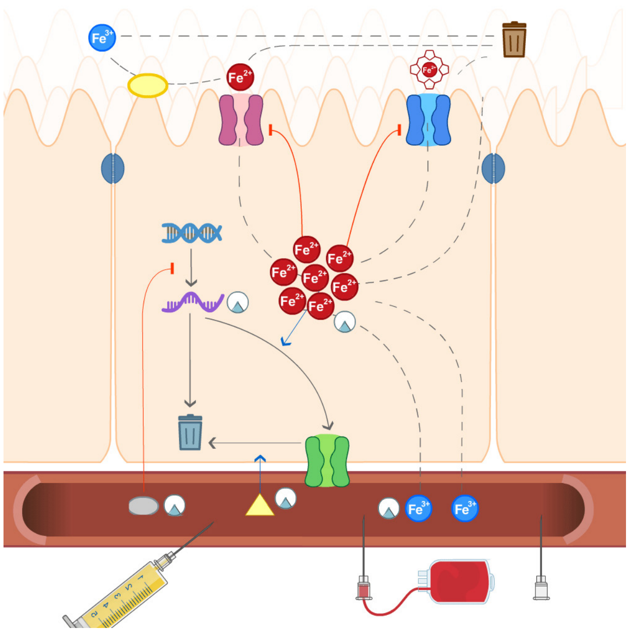

Figure 6. Iron regulation. The present screen shows the basic iron metabolism in duodenal cells. The dashed lines symbolize iron transfer between cell compartments and blood stream. It is possible to regulate food iron income as well as blood transfusion and loss. Further it is possible to initiate an inflammation process via injection of lipopolysacharide (LPS) into to blood vessels.

kidney. In the following text we explain some of the kidney's main functions and present the model behind the simulation application. Finally we present the application itself, which is composed of several consecutive simulation screens, and discuss how it is used to clarify physiological processes.

3.1.1 Basic kidney functions

The urinary system is comprised of two kidneys connected via ureters to the urinary bladder, and an urethra. The kidneys produce urine containing excess water, electrolytes and body waste products. The urine then flows down the ureter into the bladder where it is temporarily stored. The bladder is then reflexively emptied via the urethra.

The kidney has many important homeostatic, hormonal, and metabolic functions making its (patho) physiology very complex and difficult for medical students to comprehend. To mention several of these:

1. The water balance and electrolyte homeostasis.
2. The regulation of acid-base balance.
3. Excretion of metabolic waste products, especially the toxic nitrogenous compounds and xenobiotics.
4. Production of renin enzyme for arterial blood pressure control and erythropoietin, which stimulates red blood cell production in the red bone marrow.
5. Conversion of vitamin D into an active form for the regulation of calcium balance.

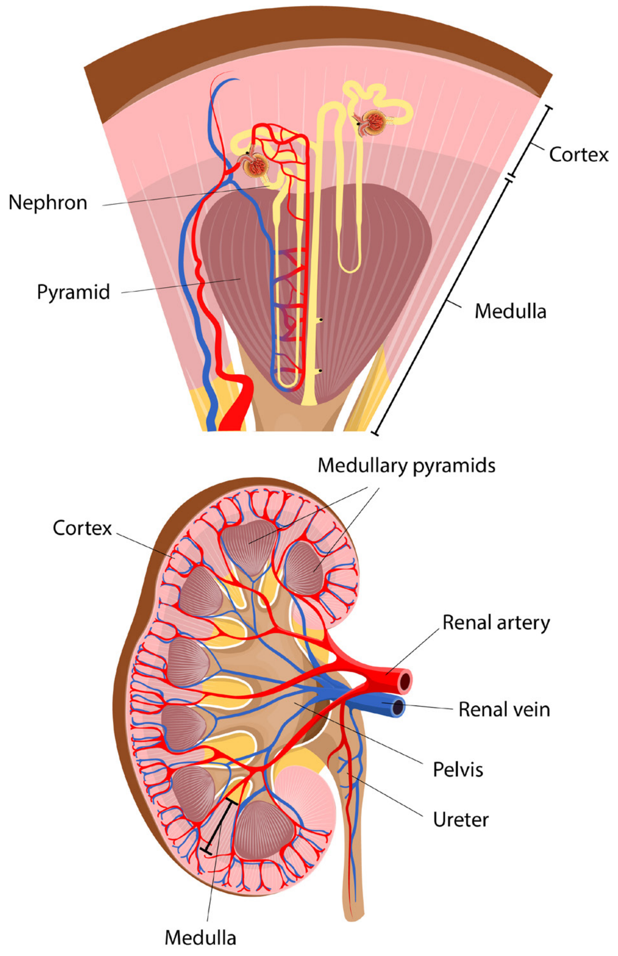

Figure 7. Kidney anatomy

Anatomically a kidney is composed of two layers: an outer layer named cortex and an inner layer called medulla. The medulla contains multiple cone-shaped lobes, known as medullary pyramids. The urine drains into the renal pelvis, which is the initial part of the ureter. The hilum of the kidney is the site of entry and exit for renal artery, renal vein, and ureter.

There are two types of nephrons (Figure 7) differing in length and urine concentration capacity: short cortical (70 – 80%) and long juxtamedullary (20 – 30%) nephrons, in total there is about one million nephrons in each kidney. The nephrons begin in the cortex; the tubules descend down to the medulla, then make a U-turn and return to the cortex before draining into the collecting duct. The collecting ducts then descend towards the renal pelvis and empty the final urine into the ureter.

Each nephron has the following parts (Figure 8):

1. Glomerulus and Bowman's capsule (Figure 9).
2. Proximal tubule.
3. Loop of Henle (descending and ascending parts).
4. Distal convoluted tubule.
5. Collecting duct.

Throughout the length of the nephron, peritubular capillaries lie adjacent to all segments of the tubule and help to maintain the dynamic stationary state. The capillaries originate from the efferent glomerular arteriole and remove the water and solutes excreted by the tubules.

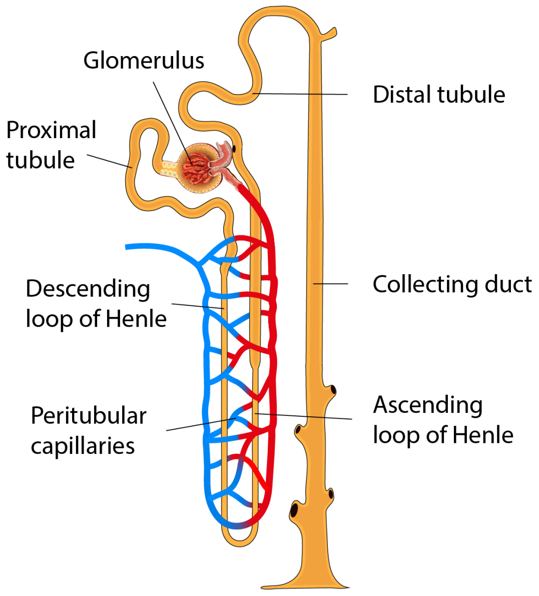

Figure 8. Nephron

The present application focuses on the urine production in terms of water and sodium ion (Na^+) only. Prospectively we intend to extend the model by including more solutes and kidney processes in future.

3.1.2 Physiology and models

There are two separate models used in the application. One for the glomerulus simulator (Figure 10) and one for the other (nephron tubules) simulators (Figure 11). Both glomerulus and nephron tubule models utilize the PhysioLibrary (Matejak and Kofranek, 2015), which was also developed within our group. The models are considered to be in the steady state and the model does not take temporal evolution into account.

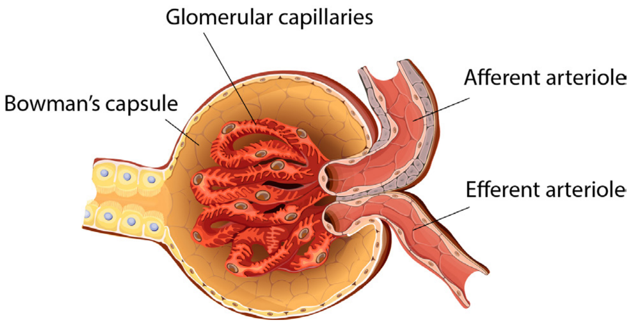

Figure 9. Glomerulus

Glomerulus

Blood enters the afferent arteriole and flows into the glomerular cluster of intertwined capillaries buried within the Bowman's capsule. The blood leaves the glomerulus through the efferent arteriole.

The wall of the glomerular capillaries is penetrated with many microscopic slits through which the fluid and small solutes passes into the Bowman's capsule. The size of the filtration slits restricts the large molecules from being filtered out the plasma (such as protein albumin/ globuline) and cells (such as erytrocytes or leucocytes).

The process of the glomerular filtration is often called renal ultrafiltration. The force of the hydrostatic pressure in the glomerulus (the force of the pressure exerted from the pressure of the blood vessel itself) is the driving force that pushes the filtrate out of the capillaries.

The osmotic pressure (the pulling force exerted by the albumins) works against the greater force of the hydrostatic pressure, and the difference between the two determines the effective filtration pressure and the glomerular filtration rate (GFR), along with a few other factors.

GFR is physiologically kept constant for a wide interval of arterial pressure. The hydrostatic pressure can also be controlled by widening (vasodilation) or narrowing (vasoconstriction) the afferent and efferent arterioles.. The glomerulus model was implemented from scratch. It utilizes the hydraulic domain (connector composed of *pressure* and flow *VolumeFlowRate* variables) of PhysioLibrary. The model is analogy of electrical voltage divider. The model represents all the glomeruli contained in pair of kidneys, e.g. the flows in the model are summed up over all nephrons.

Resistance of the glomerular capillary wall is modeled with the *filterResistance* component. Afferent and efferent arterioles are modelled with the *afferentResistance* and *efferentResistance* components. The two resistances are variable and affect the pressure on input of *filterResistance* and thus flow through it (which is GFR). The osmotic pressure is included simply by adding two extra pressure columns (*osmoticBlood* and *osmoticUrine*) around *filterResistance* component.

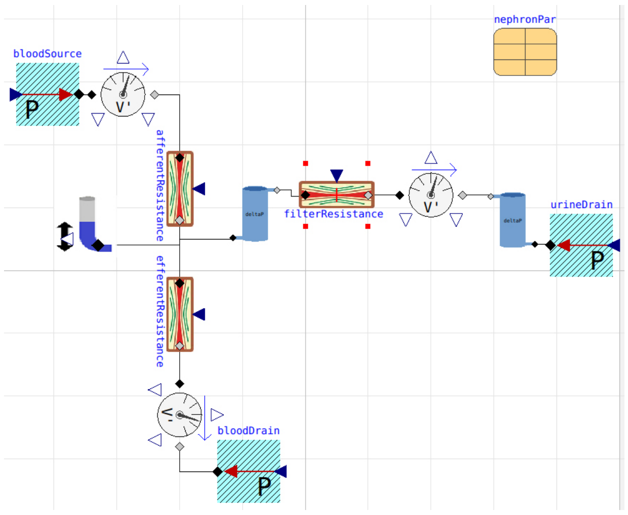

Figure 10. Glomerulus model, analogy of electrical voltage divider: afferent and efferent resistances control pressure on the left connector of filter resistance. Osmotic pressure is modeled by pressure columns.

Tubules

The nephron tubule model utilizes the osmotic domain (connector composed of *Concentration* and flow *VolumeFlowRate* variables) of the PhysioLibrary. It is implemented according to (Hoppensteadt and Peskin, 1992), but there are several changes and extensions. It is basically a system of ODE in a space coordinate. This coordinate is manually discretized so that fields are replaced with arrays and derivatives with forward differences. The model is composed of several tubules components/classes. All tubule models extend from common general tubule with this equations:

$$\begin{aligned} \frac{dQ}{dx} + f_{H_2O} = 0 \\ \frac{d(Qo)}{dx} + f_{Na} = 0 \end{aligned} \qquad (1)$$

where Q is filtrate volumetric flow [m^{3}s^{-1}] through the tubule, o is osmolarity of the filtrate [mOsm L^{-1}], f_{H2O} water volumetric flow through the tubule wall in the outer direction per unit length [m^{2}s^{-1}] and f_{Na} similarly the Na molar flow through the tubule wall in the outer direction per unit length.[mmol s^{-1}m^{-1}]. The particular tubules differ with the equations for f_{H2O} and f_{Na} according to the tubule function.

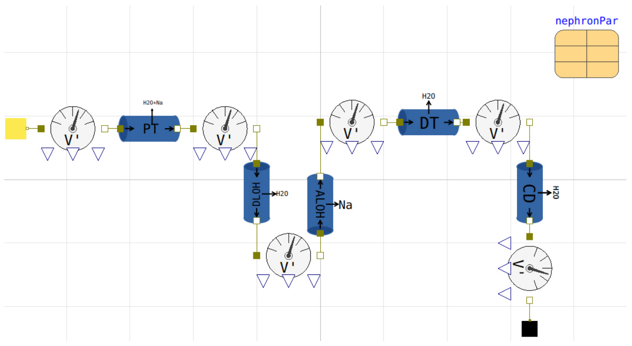

Figure 11. Nephron model is composed of UnlimitedVolume source as a simplified glomerulus and a sequence of tubules (proximal tubule, descending loop of Henle, ascending loop of Henle, distal tubule, collecting duct)

Proximal tubule

The proximal tubule is the major resorptive segment of the nephron and accounts for resorption of nearly two-thirds of all filtered water and sodium. The water is reabsorbed along with all the dissolved sodium, so that the filtrate osmolarity is preserved. The additional equations in this component are

$$\begin{aligned} f_{H_20} = k_{H_2O} \\ o = o_{in} \end{aligned} \qquad (2)$$

where k_{H2O} is chosen so that ⅔ of water is reabsorbed in the proximal tubule under the normal GFR. oin is input osmolarity of the proximal tubule.

Descending Loop of Henle

The descending Loop of Henle displays a high permeability to water but is virtually impermeable for sodium. The osmolarity of medulla surrounding the tubule rises from 300 near the cortical layer down to 1200 mOsm/l deep in medullary layer. The water leaves passively the tubule so that the osmolarity in the tubule equalizes with osmolarity of the ambient medulla. Approximately 20% of water is reabsorbed here.
Additional equations are

$$\begin{aligned} o = o_{med} \\ f_{Na} = 0 \end{aligned} \qquad (3)$$

where omed is an array. Its value rises linearly with the index to model the medulla osmolarity gradient.

Ascending Loop of Henle

The ascending loop of Henle accounts for resorption of nearly a quarter of the filtered load of sodium. It is virtually impermeable to water. Given the large amount of solute resorption that occurs in the absence of water resorption, the tubular fluid becomes progressively dilute as it travels through the ascending loop. This feature is why this segment is frequently referred to as the “Diluting Segment” of the nephron. The resorption is active and consumes energy in form of ATP. This enables lower osmolarity in the duct compared to surrounding medula, but the difference can’t be higher than 200 mOsm/l. The osmolarity drops down to 100 mOsm/l in the duct. The equations are

$$\begin{aligned} f_{H_20} = 0 \\ f_{Na} = \text{limit}(k_{Na}, o_{med}) \end{aligned} \qquad (4)$$

where kNa is chosen to meet the osmolarity 100 mOsm/L at the outflow of the loop of Henle under normal condition. The limiter function ensures that the osmolarity difference does not exceed 200 mOsm/L namely at decreased GFR.

Distal tubule and collecting duct

The distal tubule and the collecting ducts represent the final functional segment of the nephron after which any remaining tubular fluid is excreted as the final urine. By this segment, the vast majority of solutes and water is resorbed and thus the late distal tubule and collecting ducts are responsible only for a small fraction of total resorption. However, this represents the major locus of regulated tubular resorption and given the enormous quantities of glomerular filtration that occur per minute, even small changes in resorption rates at this segment can have enormous impacts on the composition of the body’s extracellular fluid.

The distal tubules of several nephrons empties into one shared collecting duct.

The distal tubule and collecting duct system is under the control of antidiuretic hormone (*ADH*). When *ADH* is present, the tubules becomes permeable to water. The high osmotic pressure in the medulla (generated by the counter-current multiplier system/loop of Henle) then passively draws out water from the tubules to medulla and blood vessels drain it away. Than the final urine osmolarity is about 1200 mOsm/l and as much as possible water is retained in body.

With no *ADH*, tubule walls are impermeable to water, no water is reabsorbed. The urine osmolarity is 100 mOsm/l (as it leaves ascending loop of Henle) and the body loses water rapidly. (4)

The equations of both distal tubule and collecting duct are

$$f_{H_2O} = ADH \cdot q_{H_2O}(o_{med} - o) \qquad (5)$$
$$f_{Na} = 0$$

Where *ADH* ϵ (0,1) and q_{H2O} is constant.

3.1.3 The simulators

The simulator is composed of several screens, each for a section of the nephron. All screens contain besides others sliders to control some model parameters and plots visualising usually flow and osmolarity along tubules. But these are not shown on the following screenshots.

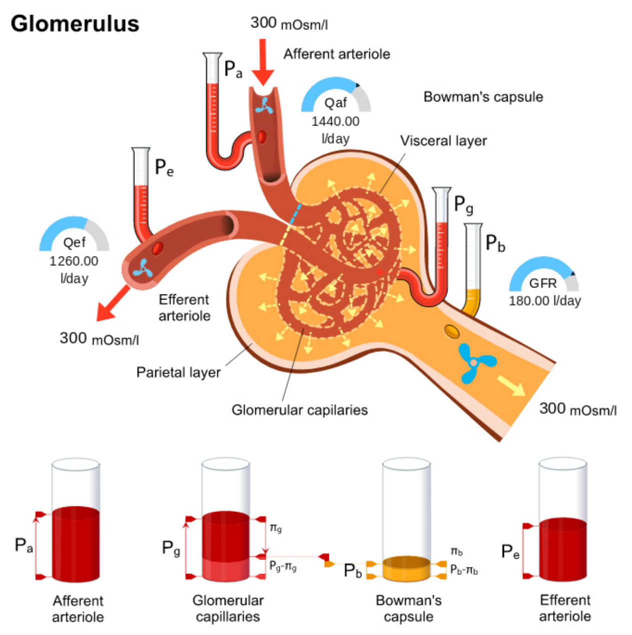

Figure 12. Top: Bowman's capsule (yellow), afferent and efferent arterioles. Red arrows symbolize the blood flow direction, yellow arrow represents urine flow direction. For each part of glomerulus both the hydrostatic and osmotic pressure is calculated. The difference between the total effective pressure in capillaries and in the Bowman's capsule drives the net filtration flow.

Glomerulus

Figure 12 depicts the glomerulus screen. Pressures are depicted with the liquid-column gauge, flows through the tubules with the half-circle measures and turbines, flow through the vessel walls with width of the dashed moving arrows. This visualized sensors are used also in all other parts of the application.

Student can change resistance of afferent and efferent arteriole and thus control the hydrostatic pressure in glomerular capillaries and GFR. Mean arterial pressure (pressure at afferent arteriole entry) may be also modified. The filtration resistance may be controlled as well to simulate certain renal pathologies. The goal is to explain how the glomerulus maintains constant GFR despite changing arterial pressure by means of changing afferent and efferent arteriole resistance.

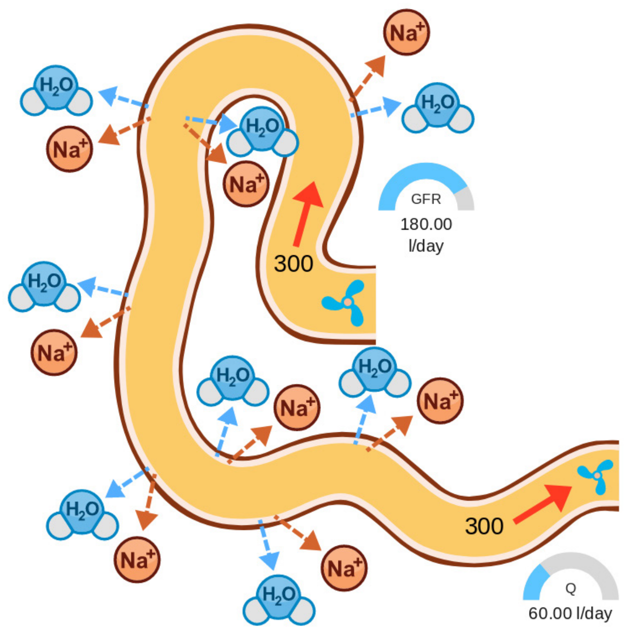

Figure 13. Diagram of the proximal loop. The red arrows show the direction of the urine flow. Along the proximal tubule sodium ions and water molecules are reabsorbed across the cellular boundary.

Proximal tubule

Proximal tubule is shown on Figure 13. The number inside the tubule depicts the osmolarity of the filtrate. GFR may be controlled. Decrease of flow may be observed on the flow measures whereas the osmolarity is maintained.

Loop of Henle

Loop of Henle is shown on Figure 14. Ascending and descending tubules of Loop of henle are together on one screen. Student can control the GFR and observe changes in reabsorption rates, flow and osmolarity. The sodium transport limiter is applied in the ascending section so the osmolarity never drops below 100mOsm/l at the outflow.

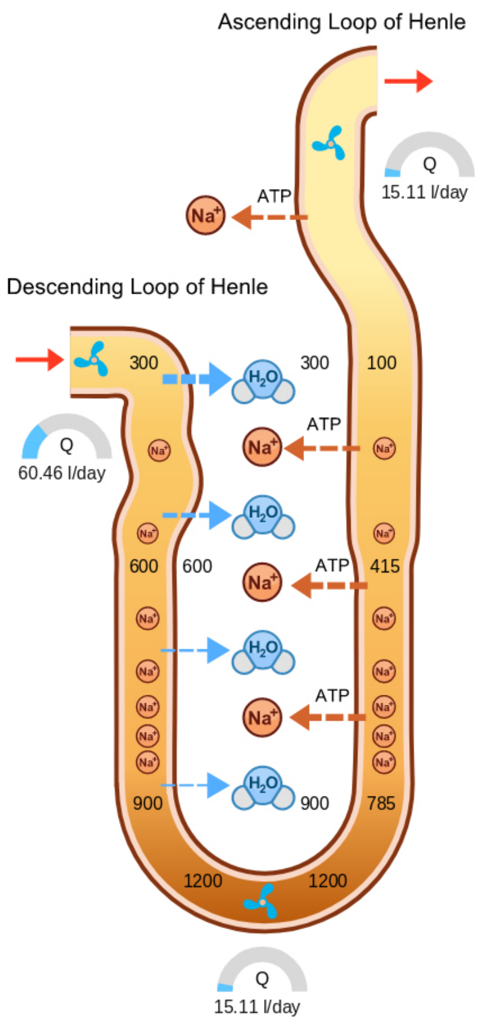

Figure 14. Loop of Henle. Red arrows show the urine flow direction. Water molecules passively leave the urine making it more concentrated from physiological 300 mosm/l down to 1200 mosm/l in the descending tubule. The blue dashed arrows indicate the amount of water molecules transfer from tubulus along its osmotic gradient. The ascending tubule is virtually impermeable to water molecules while sodium ions are actively pumped outside the tubulus. The width of the purple arrows indicates the amount of the sodium reabsorption.

Distal tubule and collecting duct

Figure 15 depicts the distal tubule and collecting duct. The amount of ADH may be controlled by the student. Higher permeability of the tubule wall for water is represented by widening of the blue water channels. The goal is to explain how ADH affects the reabsorption: With no ADH, there is no reabsorption. Filtrate goes through unaffected. Urine production rate is high and its osmolarity is low. With full ADH so much water is reabsorbed that the urine osmolarity equalizes with osmolarity of the surrounding medulla. Only small amount of highly osmotic urine is produced.

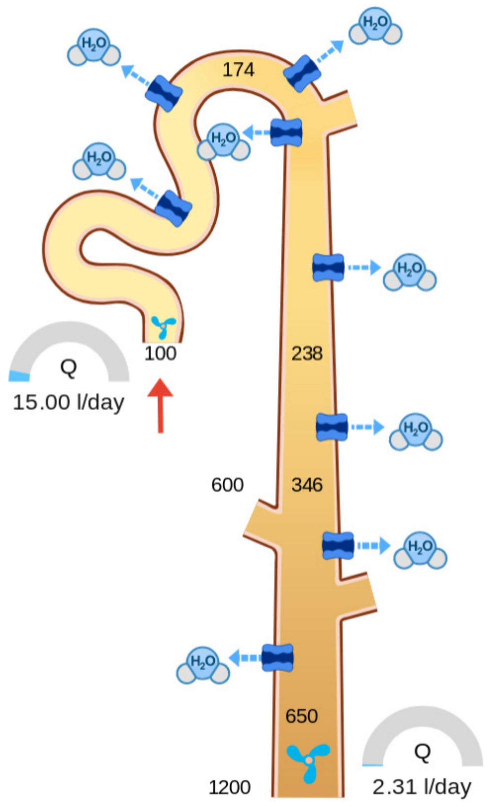

Figure 15. Distal tubule and collecting duct represent the last two segments of a nephron. The distal tubule and the collecting duct are permeable for water molecules, however, the net flow across the tubule wall is endocrinologically regulated via ADH. As a result the body is able to fine tune the urine osmolarity according to the circumstances.

Complete nephron

Figure 16 shows a complete nephron. All the information from previous screens is recapitulated here. For simplicity the GFR is controlled directly instead of controlling afferent and efferent arteriole resistance as it was in the glomerulus section. ADH is controlled as well.

4 Discussion and conclusion

New framework Bodylight.js for building web-based simulators was presented. Bodylight.js will be available for public use, but it is still under heavy development and not ready for widespread deployment. We encourage interested parties to contact us and we will gladly provide access and documentation. We welcome any feedback and code contributions others can provide. The Bodylight.js was already used to compose three teaching simulators and proved to be really useful. One of them, the Nephron simulator was presented within this paper. This simulator will be used in physiology lectures at our faculty and will be updated according to the feedback from teachers and students alike. Physiological model for this simulator was developed as

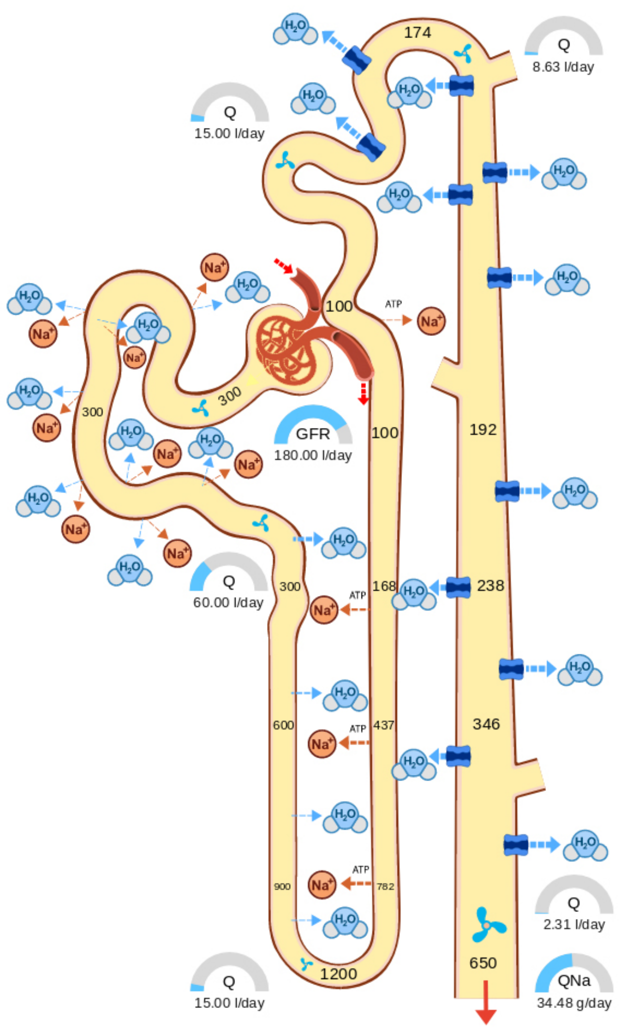

Figure 16. Screen of the whole nephron model, i.e. glomerulus, proximal tubule, descending and ascending loop of Henle, distal tubule and collecting duct. Besides others, the mass flow rate of excreted Na is displayed.

well. Results of the model were checked by physiologists and are approximately correct, enough for the teaching purposes. We plan to add a follow-up simulator including more solutes and additional regulation mechanisms.
We have built an extensive library of simulators with the previous, now defunct, Silverlight technology, covering large portions of physiological systems. We hope that by relying on standardized web technologies we can provide a plethora of new and future-proof web based teaching applications.

Acknowledgement

This work was supported by the TRIO MPO FV20628, SVV260371, PROGRESS Q26 grants and the Creative Connections s.r.o.. We thank Martin Brož for the artwork.

References

Blochwitz T, Otter M, Akesson J, et al. (2012) Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In: *Proceedings of the 9th International MODELICA Conference*; September 3–5; 2012; Munich; Germany, 2012, pp. 173–184. Linköping University Electronic Press. Available at: http://www.ep.liu.se/ecp_article/index.en.aspx?issue=076%20;article=017.

Christ A and Thews O (2016) Using numeric simulation in an online e-learning environment to teach functional physiological contexts. *Computer methods and programs in biomedicine* 127: 15–23. DOI: 10.1016/j.cmpb.2016.01.012.

Enculescu M, Metzendorf C, Sparla R, et al. (2017) Modelling Systemic Iron Regulation during Dietary Iron Overload and Acute Inflammation: Role of Hepcidin-Independent Mechanisms. *PLoS computational biology* 13(1): e1005322. DOI: 10.1371/journal.pcbi.1005322.

Fritzson P and Engelson V (1998) Modelica—A unified object-oriented language for system modeling and simulation. In: *European Conference on Object-Oriented Programming*, 1998, pp. 67–90. Springer. Available at: https://link.springer.com/chapter/10.1007/BFb0054087.

Hoppensteadt FC and Peskin CS (1992) *Mathematics in Medicine and the Life Sciences*. DOI: 10.1007/978-1-4757-4131-5.

Ježek F, Privitzer P, Mateják M, et al. (2012) Demonstration of the Risk of Fixed Ejection Volume in Ventricular Assist Devices in Small Patients Using Web Simulator. In: *5th European Conference of the International Federation for Medical and Biological Engineering*, 2012, pp. 489–492. Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-23508-5_127.

Ježek F, Tribula M, Kolman J, et al. (2013) Sada výukových simulátorů – výsledky vývoje frameworku bodylight. *MEDSOFT 2013: sborník příspěvků*: 38–48. Available at: http://www.medvik.cz/link/bmc13015203.

Kofránek J, Privitzer P, Matoušek S, et al. (2009) Schola Ludus in Modern Garment: Use of Web Multimedia Simulation in Biomedical Teaching. *IFAC Proceedings Volumes* 42(12). Elsevier: 413–418. DOI: 10.3182/20090812-3-DK-2006.0087.

Kofranek J, Matousek S, Rusz J, et al. (2011) The Atlas of Physiology and Pathophysiology: Web-based multimedia enabled interactive simulations. *Computer methods and programs in biomedicine* 104(2): 143–153. DOI: 10.1016/j.cmpb.2010.12.007.

Kulhánek T, Mateják M, Šilar J, et al. (2013) Hybridní architektura pro webové simulátory. *MEDSOFT 2013: sborník příspěvků*: 115–121. Available at: http://www.medvik.cz/link/bmc13015212.

Matejak M and Kofranek J (2015) Physiomodel – an integrative physiology in Modelica. *Conference proceedings: ... Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference* 2015: 1464–1467. DOI: 10.1109/EMBC.2015.7318646.

Modelica Association (2017) Tools | Functional Mock-up Interface. Available at: http://fmi-standard.org/tools/ (accessed 21 July 2017).

Smith J (2015) Moving to HTML5 Premium Media – Microsoft Edge Dev Blog. Available at: https://blogs.windows.com/msedgedev/2015/07/02/moving-to-html5-premium-media/ (accessed 27 August 2018).

Tiller MM and Winkler D (2017) modelica.university: A Platform for Interactive Modelica Content. In: *Proceedings of the 12th International Modelica Conference, Prague, Czech Republic*, May 15–17, 2017, 4 July 2017, pp. 725–734. Linköping Electronic Conference Proceedings. Linköping University Electronic Press. DOI: 10.3384/ecp17132725.

Tribula M, Ježek F, Privitzer P, et al. (2013) Webový výukový simulátor krevního oběhu. *MEDSOFT 2013: sborník příspěvků*: 197–204. Available at: http://www.medvik.cz/link/bmc13015231.

WebAssembly High-Level Goals – WebAssembly (n.d.). Available at: https://webassembly.org/docs/high-level-goals/ (accessed 24 September 2018).

Zakai A (2011) Emscripten: An LLVM-to-JavaScript Compiler. In: *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, New York, NY, USA, 2011, pp. 301–312. OOPSLA ’11. ACM. DOI: 10.1145/2048147.2048224.

Zhang S (2001) *An IIOP architecture for Web-enabled physiological models*. Massachusetts Institute of Technology.