SCHOOL AS A (MULTIMEDIA SIMULATION) GAME: THE USE OF OBJECT TOOLS FOR DESIGNING MULTIMEDIA APPLICATIONS FOR BIOMEDICAL TEACHING

Jiří Kofránek, Marek Mateják, Pavol Privitzer

Laboratory of Biocybernetics, Institute of Pathological Physiology, First Faculty of Medicine, Charles University of Prague

Abstract

Nowadays, Comenius's old motto – "schola ludus" ("school as play") has found a modern use in interactive educational programs using simulation games. Educational applications using simulation games, available through the web, represent a new educational aid, very efficient from the didactic point of view in explaining complex pathophysiological processes. However, the process of creating them is not very easy – it requires multidisciplinary team cooperation and the use of suitable object-oriented development tools.The development of multimedia simulation games is a combination of research and development work. The research work consists in formalizing physiological reality by designing mathematical models, while development work is the very creation of multimedia simulators, which make use of the mathematical models designed. Creative interconnection of the various professions and various object-oriented tools and applications is the key to success. A scenario of good quality, created by an experienced pedagogue, still remains the foundation of the e-learning program. The creation of animated images is the responsibility of artists who create interactive animation in Adobe Flash or in Microsoft Expression Blend environment. The artists use the Animtester software tool developed by us, to create and test animations to be subsequently controlled by the simulation model. The core of the simulators is the simulation model, created in the environments of special development tools designated to create simulation models (we have used Simulink from Mathworks). Now, we use a very efficient object-oriented environment, which utilizes the Modelica simulation language. We are working on the Modelica language compiler to compile into the .NET component form, which, together with the differential equations solver implemented on the .NET platform as well, shall serve as the "data layer" of the simulator with the implemented model. The user interface is connected with the simulation model using the data binding concept, which provides the intelligent automatic propagation of values between the layers, thus data transfer. We use hierarchical state automatons to design the inner application logic (the automatons make it possible to remember the relevant model context and the user interface context). We have also developed a visual environment (Statecharts editor), which allows creating the graphic design of the automatons, generating a code, and debugging them. The resulting simulator is a web application for the Silverlight platform, which makes it possible to distribute the simulator as a web application running directly in the internet browser (even on computers with various operating systems – it is only necessary that the relevant plugin is installed in the browser).

1. Schola ludus in modern form

Educational ***multimedia programs*** with *simulation components* are not just a modern replacement for traditional textbooks. They are an entirely *new teaching aid* that allows vivid examination of the studied problem by means of educational simulation games.

The internet as a distribution medium can make these new teaching aids easily available worldwide.

A combination of the internet, a multimedia environment serving as an audio and visual user interface, with simulative models allows clarification of the dynamic relations between studied terms to students connected to the magical internet network with the help of an educational simulation game. The

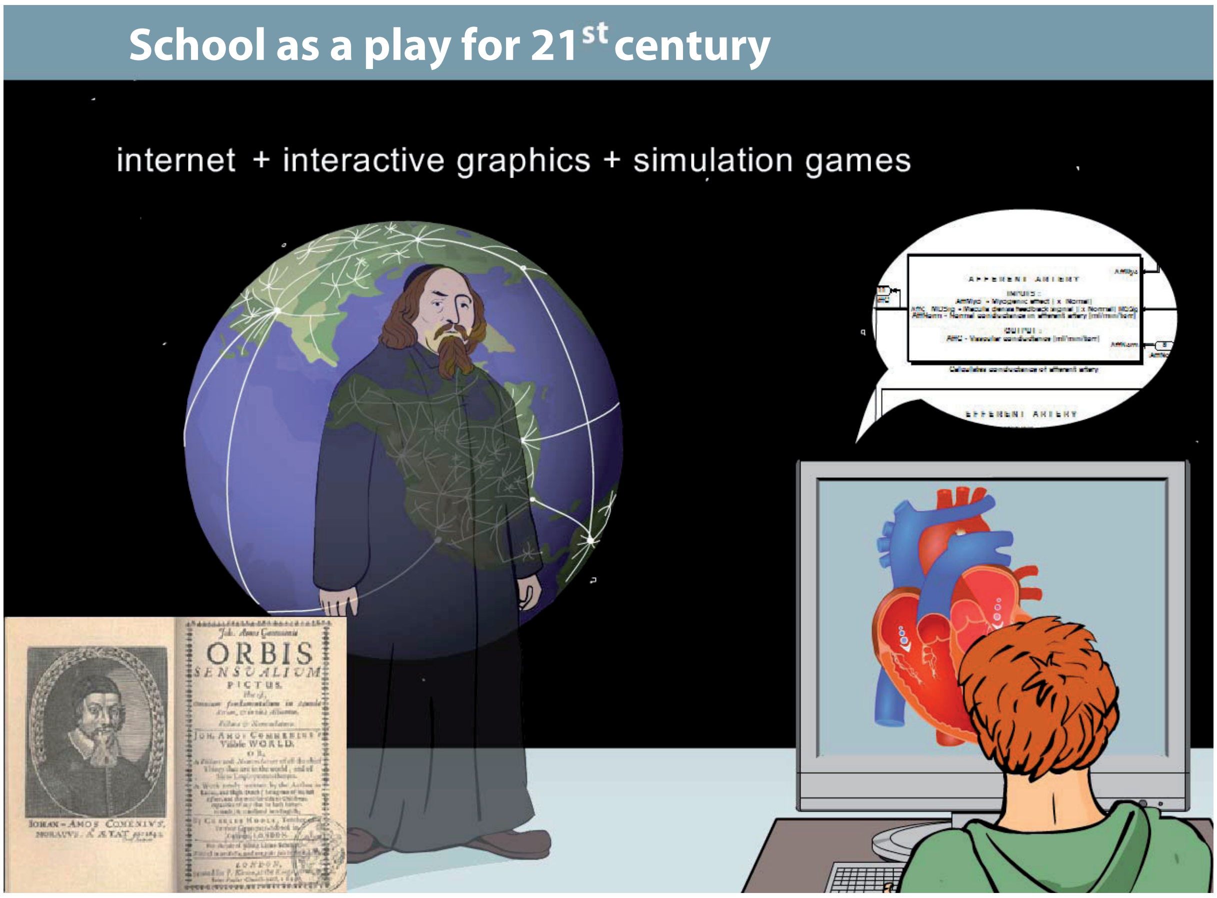

School as a play for 21st century
internet + interactive graphics + simulation games
ORBIS SENSUALIUM PICTUS
AFFERENT ARTERY
EFFERENT ARTERY

Figure 1: *The combination of the internet and interactive graphics with simulation models in educational programs allows students to get "hands-on" experience with the studied problem in virtual reality. This is the modern field of application for Comenius's old credo – "School as play".*

integration of multimedia educational games into teaching brings about entirely new pedagogical opportunities, in particular when explaining complex interrelations and actively exercising practical skills and checking theoretical knowledge. In a simulation game, it is possible to test the behaviour of a simulated object without risk – e.g. try to land a virtual aircraft or, with medical simulators, treat a virtual patient or test the behaviour of physiological subsystems.

An old Chinese proverb says: "That which I hear, I shall forget; that which I see, I shall remember; that which I do, I understand". This old Chinese piece of wisdom is proved by modern teaching methods, sometimes called "learn by doing", where simulation games play a major role. In addition, simulation games introduce an element of experience and a bit of playful enjoyment into teaching. This is the modern field of application of John Amos Comenius's old credo "Schola Ludus" (school as play) (Comenius, 1656] which was promoted by this European pedagogue as early as the 17th century (Fig. 1).

Teaching with the help of simulation games available on the internet is common in physics or chemistry; the utilization of simulation games and simulators in biomedicine is rarer, which is probably due to the complexity of the necessary simulation models. Nonetheless, there are a number of educational applications with simulation games for medicine available on the internet. Many educational *simulators of individual physiological subsystems* can be found on the Web. For instance, there is ECGsim (http://www.ecgsim.org/download.html), a simulator that allows examining the generation and propagation of the electric potential in the ventricles and studying the origination mechanism of the ventricular QRS complex for various pathologies from heart blocks to ischaemias and infarctions (Oostendorp, 2004). Pressure circulation curves in the ventricles with various heart pathologies (valve defects, left or right heart failures) can be observed on a heart simulator from Columbia University (http://www.columbia.edu/itc/hs/medical/heartsim) (Burkhoff and Dickstein, 2002, Kelsey et al., 2002); simulators of anaesthesia machines from the University of Florida allow giving anaesthesia to a virtual patient (http://vam.anest.ufl.edu/) and monitoring related physiological responses, etc. (however, the more complex simulators require paid access).

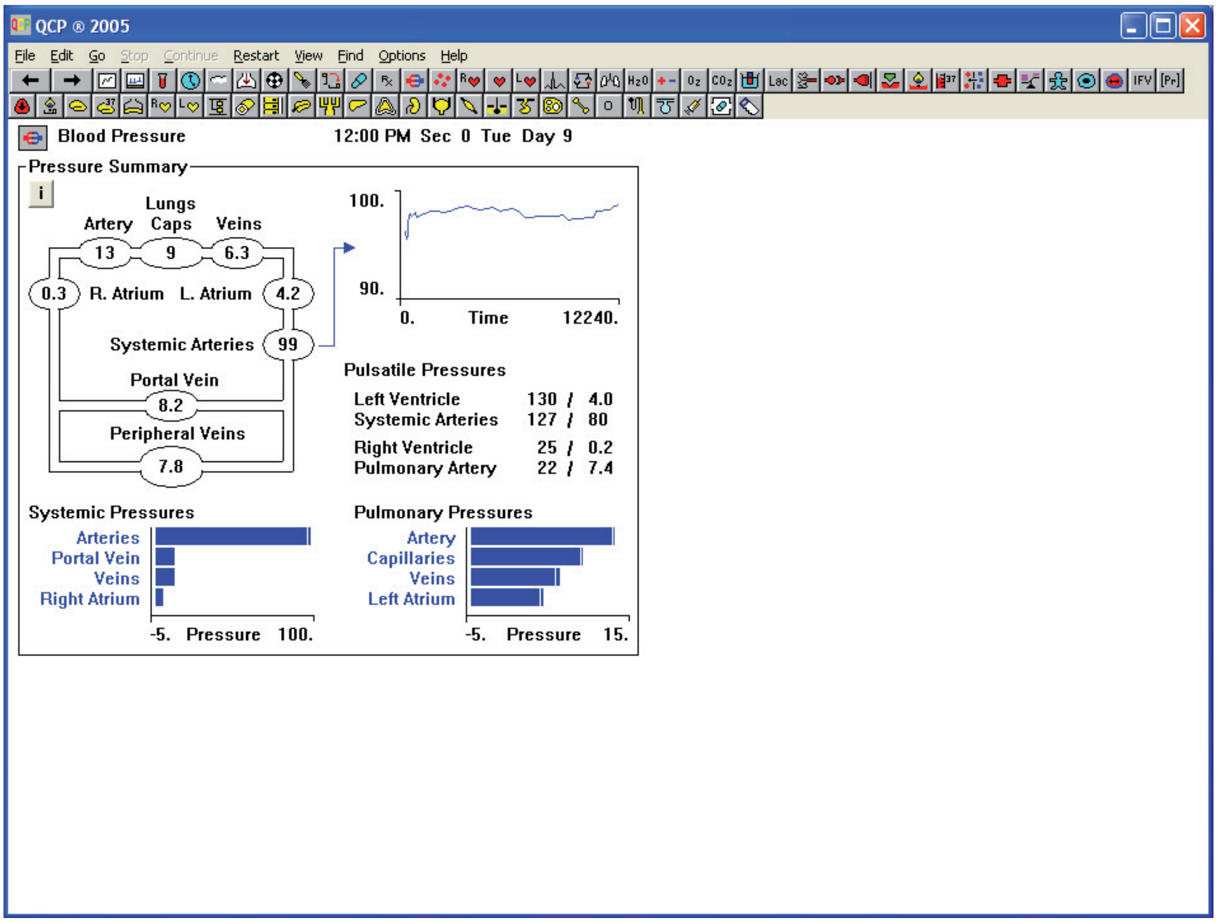
2. Complex models for integrative physiology and education

Pathophysiology teaching and the study of the pathogenesis of various pathological states can make good use of complex simulators including *models of not only individual physiological subsystems but also their interconnection into a more comprehensive whole*. The creation of such models was pioneered by Prof. Guyton, who used a mathematical model to describe the physiological regulations of the circulatory system and its broader physiological relations and links with the other subsystems in the body – the kidneys, volumetric and electrolyte balance control, oxygen transfer, nerve and endocrine control, etc. – in the Annual Review of Physiology in 1972 (Guyton et al., 1972).

Guyton's model, which we described in detail in the previous chapter of this book, was the first extensive mathematical description of the physiological functions of interconnected body subsystems and launched the field of physiological research that is sometimes described as *integrative physiology* today. The model was not of purely theoretical importance – Guyton soon realized the great significance of models used as specific teaching aids.

Guyton and his disciples continued developing the model. In 1982, Guyton's colleague Thomas Coleman created the "Human" model intended mainly for educational purposes. The model allowed simulating a number of pathological states (cardiac and renal failure, haemorrhagic shock, etc.) and the effect of certain therapeutic interventions (infusion therapy, the effect of some medicines, blood transfusion, artificial pulmonary ventilation, dialysis, etc.) (Coleman and Randal, 1983). Meyers et al. (2008) have recently made Coleman's original model available on the Web by implementing it in Java.

In 2005, Coleman et al. published a large educational simulator, Quantitative Circulatory Physiology (QCP) which they made freely accessible on the Web (http://physiology.umc.edu/themodeling-workshop/) to support its use as a medical teaching aid (Abram et al., 2007), see Fig. 2.



Pulsatile Pressures

Left Ventricle	130 / 4.0
Systemic Arteries	127 / 80
Right Ventricle	25 / 0.2
Pulmonary Artery	22 / 7.4

Figure 2: *The environment of the Quantitative Human Physiology educational simulator. The simulator offers monitoring of hundreds of variables, but is difficult to control and requires study of the extensive simulator structure as well as good knowledge of which processes need to be monitored during simulations of certain pathological states.*

This was further expanded into the *Quantitative Human Physiology* educational simulator including more than 4,000 variables, which is probably the largest model of physiological regulations available today (Coleman et al, 2008, Hester et al. 2008)

We have also been engaged in the development of complex educational models for medical training and previously created the *"Golem" educational simulator*, which was based on a complex model of integrated physiological controls (Kofránek et al, 2001). Our "Golem" simulator focused primarily on teaching complex disorders of the inner environment (Kofránek et al., 2005).

3. Simple is better

However, experience with the deployment of complex models in teaching has shown that large, complicated models have a significant disadvantage from the didactic point of view in that they are difficult to control.

The large numbers of input variables and wide range of possibilities in monitoring output variables require that the user have a deeper understanding of the actual structure of the simulation model and know which processes need to be monitored during simulations of certain pathological states. Otherwise the complex, sophisticated model will seem just a "complicated and hard to understand technical toy" to users (similarly to when you place them in front of a complex airliner simulator with no previous theoretical training).

Therefore, educational models (and perhaps not just the complex ones with hundreds of variables) are insufficient for use in teaching on their own. They have to be accompanied by an explanation of how they should be used – preferably using interactive educational applications. Only a *combination of teaching and simulation play* provides the opportunity to take full advantage of virtual reality when explaining complex pathophysiological processes. To combine the advantages of interactive multimedia and simulation models for medical training, we came up with the project of an internet, computer-based *Atlas of Physiology and Pathophysiology* (Kofránek et al, 2007), designed as a *multimedia teaching aid* that should use visual, internet-based simulation models to help explain the function of individual physiological subsystems and the causes and symptoms of relevant disorders – see http://physiome.cz/atlas/. The Atlas thus combines explication (using animation with sound) and interactive simulation play with models of physiological subsystems. All is freely available on the internet (Fig. 3).

4. Educational application framework – the script

A good script is the key to success for any educational program. The first person upon whom the success of an application in the making depends is an experienced teacher, who has to be sure of what they want to explain their students using the multimedia educational application and by which means, and where and how a simulation model can be used to clarify the studied subject.

The basis of any script is usually instructional text – a textbook, a chapter in a textbook, etc. However, when creating the script for a multimedia educational application, we have to think of how the e-learning program will be displayed on the screen, what the order of the screens will be, how they will be designed, where interactive elements will be placed, where sound can be turned on, what the individual animations will look like, where a simulation model will be put and how it will be controlled, where a test should be put, what it will look like, how it will be evaluated and what the reaction to the results should be, etc.

We have found it useful to take an approach known from animated film – to draw (preferably in cooperation with an artist) a "storyboard" – a rough sequence of screens - and then write comments for each screen (or a reference to the appropriate part of the text created in a standard text editor).

However, an interactive multimedia program is not a textbook that has been simply transformed into computer form. Nor is it a linear sequence of texts, sounds and moving pictures like an animated cartoon. An important feature of an educational program is its interactivity – and the related possibility of branching and interconnecting its parts. Transforming a linear, text and picture script into a branched script interconnected by hypertext links is not easy, though.

One of the problems that need to be solved is how the script should capture the actual structure of

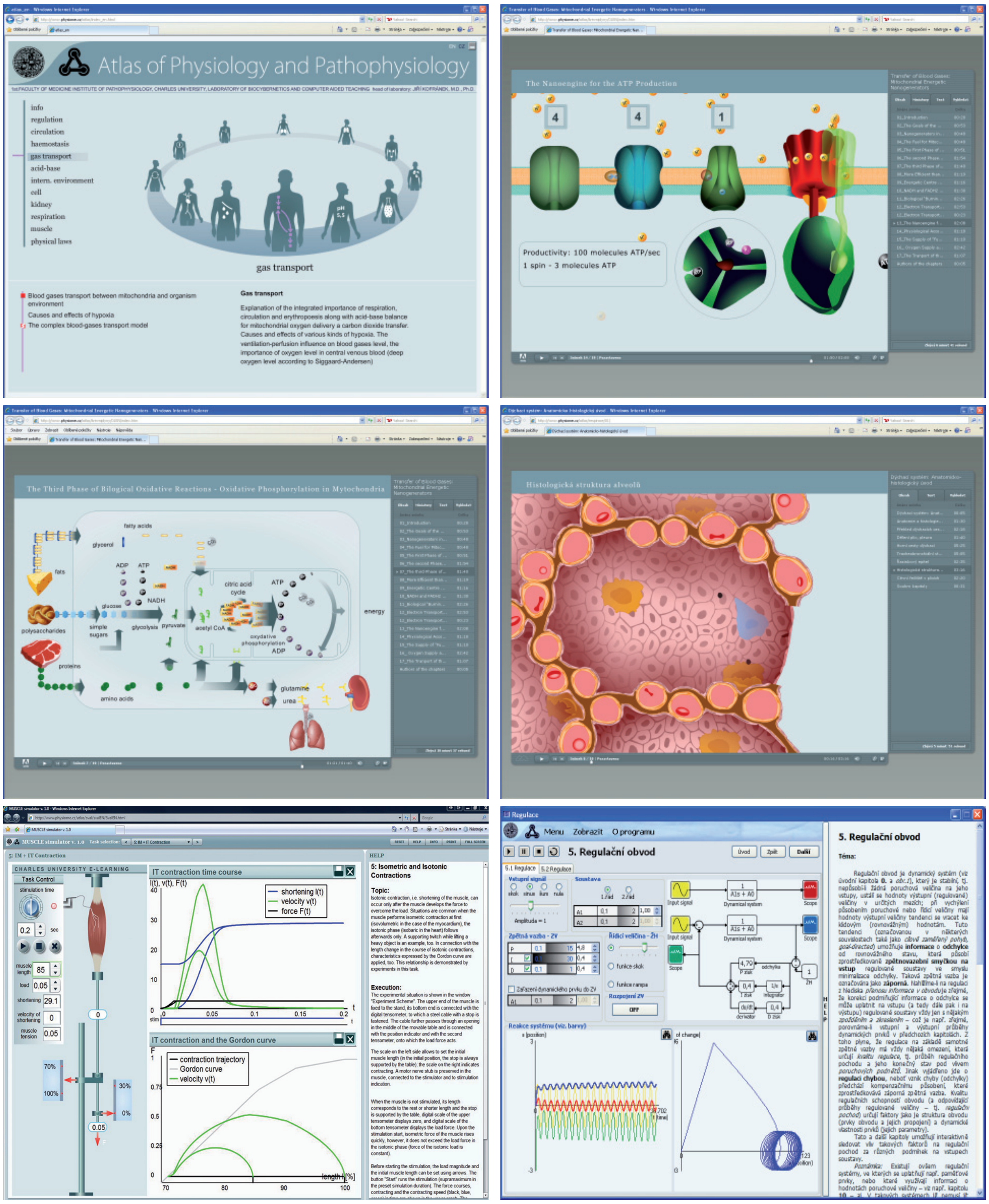


Figure 3: *The Atlas of Physiology and Pathophysiology (www.physiome.cz/atlas) combines audio interactive lectures with animations and simulation games. Atlas is designed in Czech and English version.*

the educational program, involving lectures, interaction with the user, program branching, etc. The easiest way is using standard flowcharts or block diagrams in a text or image editor to describe the relevant branches, alternative boxes, etc. together with the necessary references to text pages and appropriate images stored in additional files.

When writing scripts, we found it useful to take advantage of modern text editors' capability to create the relevant hypertext links – the script itself thus features some of the future interactivity.

A modern interactive educational program is not a computerized instructional animated film, either – the most advanced feature of interactivity is the option to put in a simulator that allows clarification of the studied problem in virtual reality by means of a simulation game.

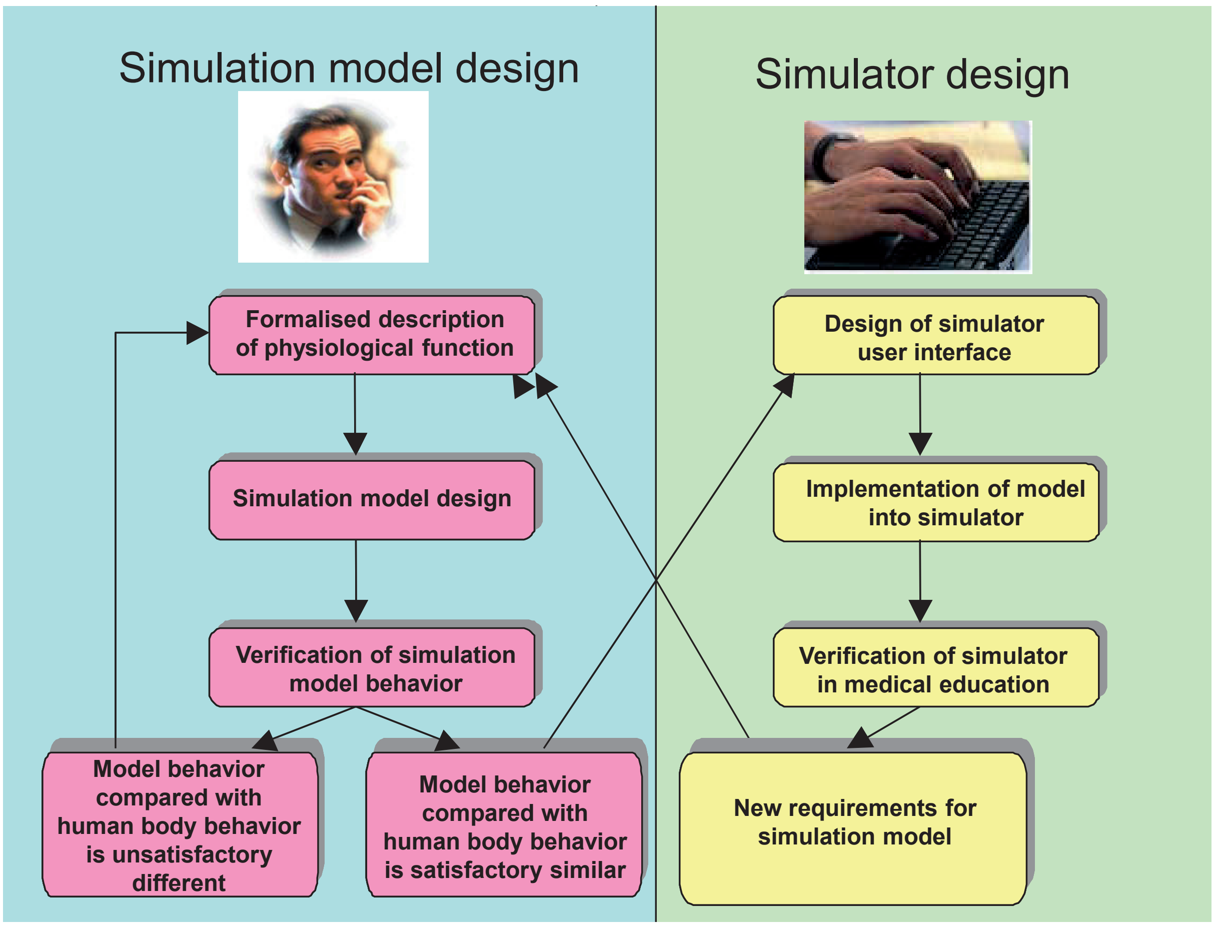
5. Two types of problems in the creation of educational simulators

Two types of problems must be solved when creating simulators and educational simulation games (Fig. 4):

1. *Creation of a simulation model* – the actual theoretical research work, consisting in a formalized representation of reality described by a mathematical model. The result should be a verified simulation model that sufficiently reflects the behaviour of the modelled reality at a specified level of accuracy.
2. *Creation of the actual multimedia simulator*, or the creation of an educational program using simulation games – is the practical application of theoretical results, which builds on the results of the research. The basis for a simulator is the created (and verified) mathematical models. This is demanding development work that requires combining the ideas and experience of the teachers who create the educational program script, the creativity of the artists who create the interactive multimedia components and the efforts of the programmers who "concoct" the resulting work in its final form.

Each of these problems has its own specifics and therefore requires the use of entirely different development tools.

While the creation of the actual simulator is mostly developer and programmer work, the creation of a simulation model is not development but a (rather difficult) research problem associated with finding an adequate formalized description for the modelled reality. The formalized description is used to create a simulation model that simulates the behaviour of the modelled reality (by solving the relevant equations of the mathematical model) using a computer. The model's behaviour is compared to the behaviour of the real-world system. Differences in the behaviours necessitate corrections in the formalized description (e.g. by specifying new values of some coefficients in the mathematical model or even changing the equations in the model) until the model's behaviour matches the behaviour of the modelled reality within specified limits of accuracy. This is called model verification.



Simulation model design
Simulator design
Formalised description of physiological function
Simulation model design
Verification of simulation model behavior
Model behavior compared with human body behavior is unsatisfactory different
Model behavior compared with human body behavior is satisfactory similar
Design of simulator user interface
Implementation of model into simulator
Verification of simulator in medical education
New requirements for simulation model

Figure 4: *Two types of problems in the creation of educational simulators.*

In the past, simulation models were created directly in the same development environment as the actual simulator (e.g. in Fortran, C++ or Java). Today, the creators and testers of simulation models make increasing use of specialized development tools – such as Mathworks's Matlab/Simulink, and others. Particularly promising are "acausal" simulation environments (using e.g. Modelica, a special language), which were discussed in the previous chapter.

The development tools for the creation of simulation models are intended for specialists. They are not appropriate for the average user who just wants to "play" with the simulation model. Although the environment of these tools enables programming a rather friendly user interface to control the created model, the interface may still be too complicated, especially for simulation model applications in medical training; moreover, it often requires purchasing additional (rather costly) licences.

A medical student or a doctor prefers a simulator user interface that resembles the illustrations and diagrams found in books, such as an atlas of physiology or an atlas of pathophysiology.

It is, therefore, necessary to program the educational simulator, including its multimedia user interface, separately. This makes the user control features of the simulator much more natural for the target group.

The creation of educational simulators and educational programs that use simulation games is primarily programmer/developer work that is based on the educational program script created by an experienced teacher on the one hand and on the results of research, i.e. the created (and verified) mathematical models on the other hand.

An irreplaceable component of each educational simulator is the program part that implements the simulation model. If we know the structure of the simulation model (which we created in some of the development tools for the creation of simulation models), we just have to transform the model structure into the form of a computer program in our preferred programming language (such as Java, C++ or C#).

Then it is necessary to draw interactive multimedia components for the creation of the user interface. After that, the components must be interconnected with the simulation model behind the simulator. In modern educational applications, such multimedia components often consist in interactive animated images. Consequently, programmers have to collaborate with artists who create the animated images to build a professional educational application.

The actual simulator is usually created in a standard development environment for the creation of software and web components (e.g. Visual Studio .NET, a Java development environment such as NetBeans, and others). Interactive graphics may be designed and programmed using other specialized environments (such as Adobe Flash and its ActionScript, and others). Educational simulators are also developed using tools for the visualization of industrial applications, such as Control Web.

The creation of a simulation model and the creation of a simulator are closely interrelated (see Fig. 4) – the creation of an educational simulator is conditional upon a sufficiently verified model, while the use of the simulator in teaching brings about a new demand for the creation of new simulation models or the modification of the existing ones.

If we use different development tools for the creation of simulation models and the development of the actual simulator, we have to ensure sufficiently flexible transfer of the results from one development environment to another.

For instance, if we modify a simulation model in a tool for the creation of simulation models (such as Matlab/Simulink), it is advisable to make sure that the changes in the model can be reflected quickly and easily in updates applying the changes to the actual simulator (developed e.g. in Visual Studio .NET).

It is useful to create custom software aids or use specialized integrated development tools to facilitate such transfer.

6. Simulator development in Control Web and LabView

To integrate hypertext, interactive animations and other multimedia components with simulation models, we set high standards for a graphical user interface on the one hand but the GUI cannot be too

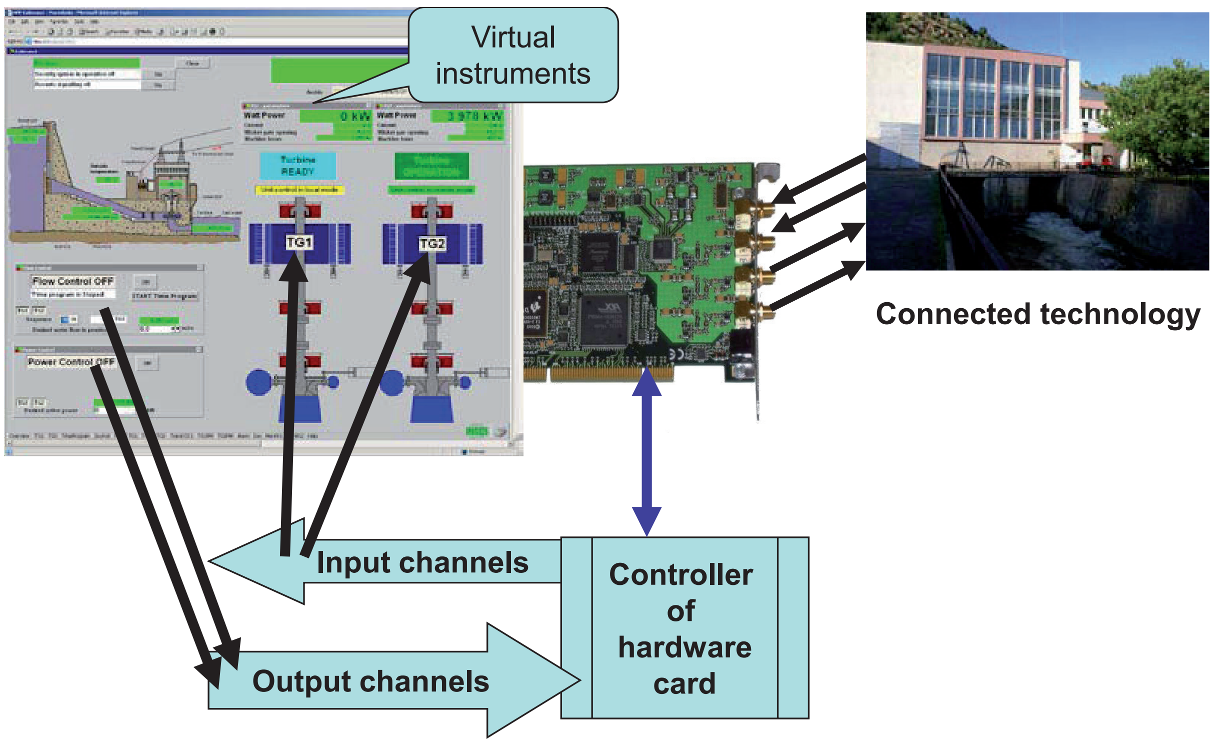


Figure 5: *Control Web's communication with the driver of a control/measuring card in the development of industrial applications. The data logger or control centre of an industrial application created in Control Web communicates via input and output channels with the controller of a measuring/control card, which communicates with the connected process equipment.*

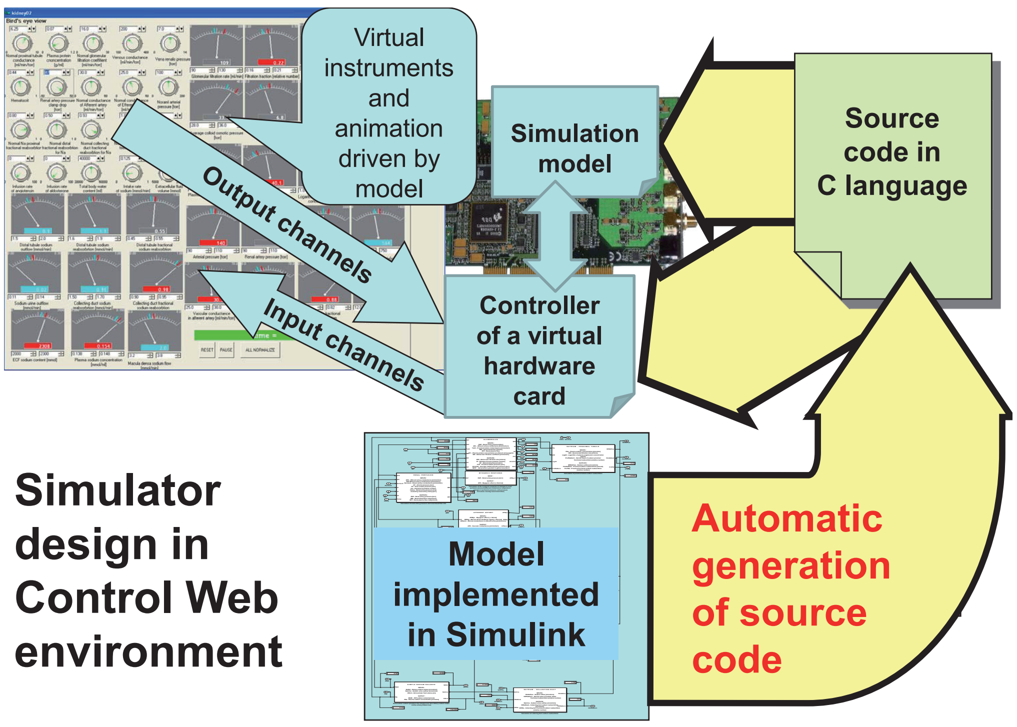

Virtual instruments and animation driven by model
Simulation model
Source code in C language
Output channels
Input channels
Controller of a virtual hardware card
Simulator design in Control Web environment
Model implemented in Simulink
Automatic generation of source code

Figure 6: *Incorporation of a simulation model into a "virtual card" driver during the creation of a simulator in Control Web. The simulator created in Control Web communicates via input and output channels with the controller of a virtual measuring/control card (with no existing hardware). The simulation model is "hidden" in the controller. Input variables are "sent" to the simulation model through the Control Web application's output channels. The model's outputs are read in the same way as the measured outputs of process equipment, by means of input channels. To avoid the necessity to manually program a controller with a "hidden" simulation model, we have developed a special tool that automatically generates C source code for the controller directly from a model implemented in Simulink.*

resource-intensive because, on the other hand, we want to keep enough resources for the numerical calculations needed to run the simulation model. Similar requirements can be found in industry – when building control centres, we would like to display various measuring instruments flexibly enough on the computer screen but we would also like to preserve sufficient numerical capacity for the actual measurement and control in the industrial application.

Therefore, one of the development tools that can be used to create educational simulators is software utilized in the creation of industrial applications (data loggers and control centres).

Development tools for the visualization of industrial applications usually offer a rich, easy to implement user interface that communicates with a simulation model programmed inside the application. Then users can manipulate the simulation model in an educational simulator as if they are controlling complex process equipment from a control centre: They wish to read (and display in varied graphical or numerical form) a great deal of diverse measured data (as in an industrial data logger) while they also wish to have a simple way (pushing a button, turning a knob or pulling a rod) of controlling the simulation model (as if controlling equipment from a control centre).

To this end, biological and medical educational simulators often utilize the LabView environment (Davis, 2001, Davis and Gore 2001).

We used the Control Web development environment from Moravian Instruments to develop educational simulators. In it, we implemented, among other things, an educational simulator of physiological functions: Golem (Kofránek et al, 2001).

The Control Web environment is intended primarily for the development of industrial, WIN32-based visualization and control applications – data collection/storage/evaluation, creation of human-machine interfaces, etc. (see http://www.mii.cz). The basic building elements of a user application developed in Control Web are virtual instruments (communicating with one another by means of variables and messages). In industrial applications, the virtual instruments receive values measured in the outside world through input channels and can send control signals to their environment through output channels.

The Control Web system provides powerful tools for the development of user interfaces. For example, it is possible to easily drag a necessary instrument from the palette of virtual instruments with the mouse, drop it on the appropriate panel and set the values of its relevant attributes and define its local variables or individual procedures (object methods) in an interactive dialogue box.

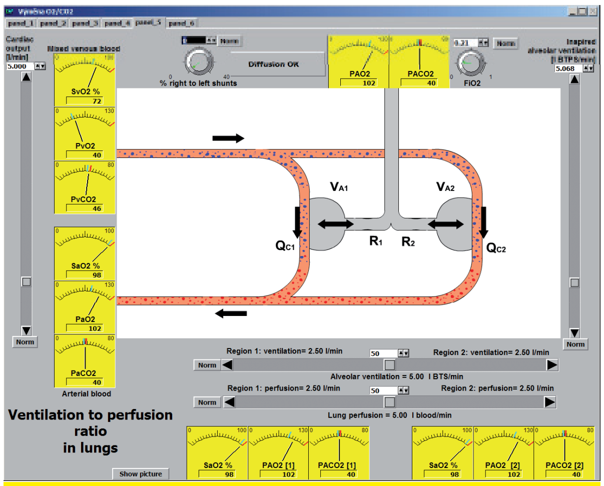
To be able to take advantage of Control Web's development comfort, we had to use the following, rather simple trick. In industrial applications, Control Web communicates with industrial process equipment (through the appropriate software channels) via the driver of a measuring/control card (Fig. 5).

However, it is possible to write a special driver with a simulation model as an internal part. Such a driver is able to communicate with Control Web objects (through software channels) but unlike the drivers of real measuring and control cards, it communicates with the simulation model instead of the hardware. If the driver is written well, Control Web is "tricked": it believes that the input channels (for display elements on the screen) are real-world, measured signals from the computer's surrounding equipment while in fact they are the output variables of a simulation model. And conversely, Control Web is convinced that the output channels running from its control elements set some industrial active elements via the appropriate driver but they change the inputs of the simulation model instead (Fig. 6).

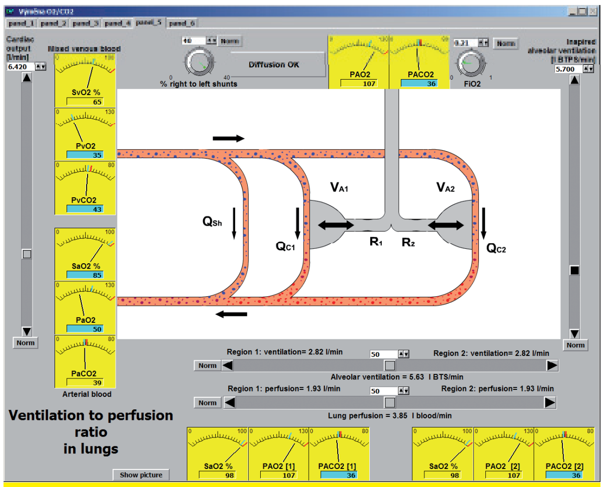
Control Web will facilitate the creation of a user interface for an educational simulator. It is fast enough and provides an efficient solution for displaying graphics without making high demands on the processor.

However, the actual mathematical model that constituted the core of our simulators was developed in Matlab/Simulink.

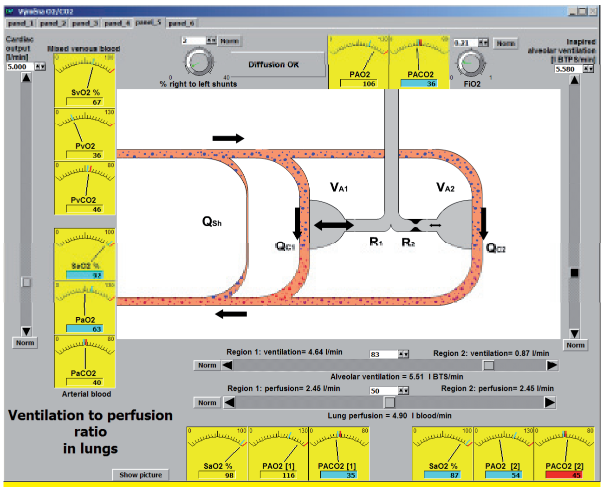
To facilitate the development of "virtual measuring/control card drivers" containing a simulation model and to avoid writing a driver for each model in the C programming language "manually", we developed a special program that will allow us to automate the development of such drivers (Kofranek et al., 2003, Stodulka et al, 2007). This allows us to generate C source code for a virtual driver directly from a Simulink diagram. Thus we can modify our Control Web driver quickly and easily for any modi-



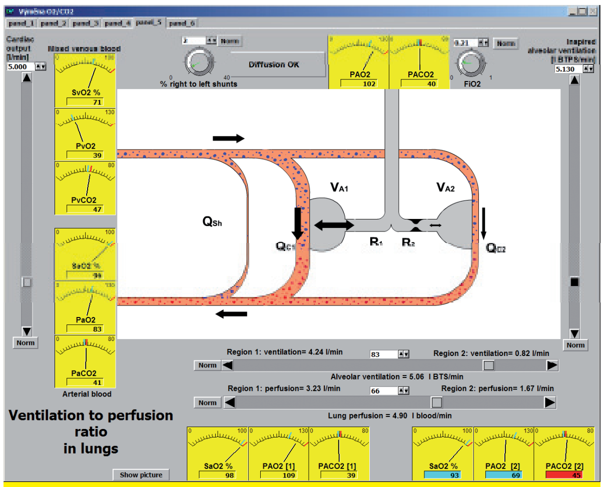
1. Normal physiological conditions



2. The increase of right-to left pulmonary shunt leads to decrease of oxygen saturation in arterial blood



3. The bronchial obstruction to one of the lung compartments leads to disturbances of ventilation to perfusion ratio with decreased oxzden saturation in mixed arterial blood

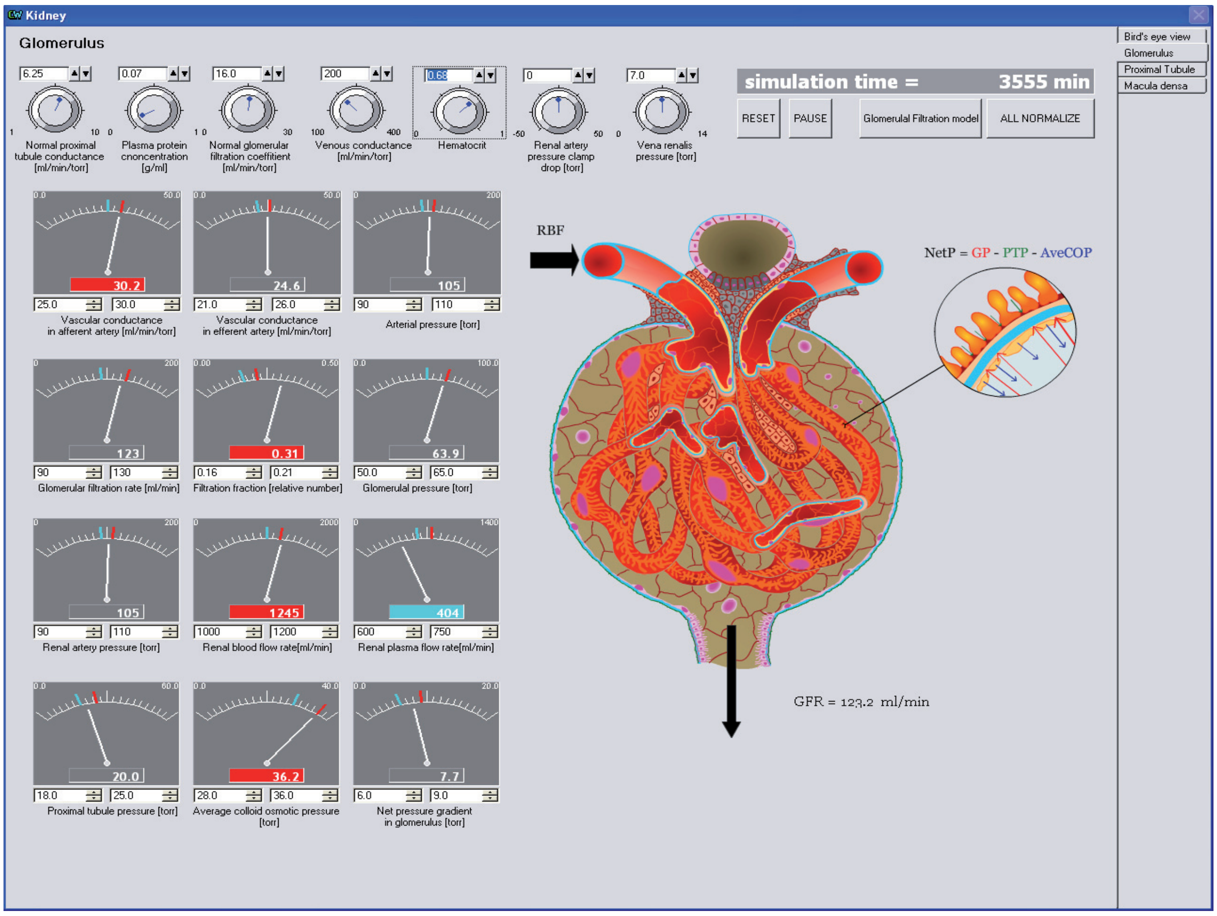


4. The compensatory effect of bronchial obstruction diminishes perfusion in hypoventilated compartment and increases oxygen saturation in mixed arterial blood

Figure 7: *An example of the various visual interface states in a simulation game in a respiration disorder teaching program. The user interface, created in Control Web and using a Flash animation, schematically shows the various degrees of ventilation and perfusion of two parts of the lungs. A student "plays" with the moving picture and the model in the background calculates the relevant physiological parameters, displayed as values on virtual measuring instruments (in Control Web). This also changes the "depth of breathing" and "amount of perfusion" of the lungs on the schematic picture consisting in Flash animation. First panel shows normal physiological conditions, second shows the increase of right-to left pulmonary shunt, third introduces a bronchial obstruction to one of the lung compartments with disturbances of ventilation to perfusion ratio and fourth the compensatory effect of perfusion opposing the bronchial obstruction. Student can watch the changes of physiological parameters in arterial, mixed venous and end pulmonary capillary blood.(in meters). Varying of ventilation and perfusion are depicted by changes of animated central panel.*

fications and new versions of our simulation model.

In addition to class instances from a wide range of ready-made virtual instruments, the Control Web object-oriented development environment allows using containers as user environment elements on the screen; such containers may contain an animation created in Adobe Flash and connected with the appropriate input or output channels of Control Web via the ActiveX interface. This will allow creating animated images controlled by the simulation model in the background (see Fig. 7 and Fig. 8).



Normal proximal tubule conductance [ml/min/torr]: 6.25
Plasma protein cnoncentration [g/ml]: 0.07
Normal glomerular filtration coefficient [ml/min/torr]: 16.0
Venous conductance [ml/min/torr]: 200
Hematocrit: 0.68
Renal artery pressure clamp drop [torr]: 0
Vena renalis pressure [torr]: 7.0

Vascular conductance in afferent artery [ml/min/torr]: 30.2 (25.0 – 30.0)
Vascular conductance in efferent artery [ml/min/torr]: 24.6 (21.0 – 26.0)
Arterial pressure [torr]: 105 (90 – 110)
Glomerular filtration rate [ml/min]: 123 (90 – 130)
Filtration fraction [relative number]: 0.31 (0.16 – 0.21)
Glomerular pressure [torr]: 63.9 (50.0 – 65.0)
Renal artery pressure [torr]: 105 (90 – 110)
Renal blood flow rate[ml/min]: 1245 (1000 – 1200)
Renal plasma flow rate[ml/min]: 404 (600 – 750)
Proximal tubule pressure [torr]: 20.0 (18.0 – 25.0)
Average colloid osmotic pressure [torr]: 36.2 (28.0 – 36.0)
Net pressure gradient in glomerulus [torr]: 7.7 (6.0 – 9.0)

Figure 8: *An example of an educational simulator of the kidneys created in Control Web. Model outputs are displayed on pointer instruments and at the same time affect the shape of the animated picture of the renal glomerule (size, thickness of arrows, numerical values, etc.) created with Adobe Flash.*

7. The "muscles" of an educational simulation application – interactive multimedia Flash components

Even though Control Web allows the easy insertion of any animated elements, created in Adobe Flash, into the created user interface, we often use the ready-made virtual instruments when developing simulators in this environment. The overall appearance of an educational simulator thus often cannot conceal the original focus of the development tool in which it was created, more or less resembling an industrial visualization application.

If we want to allow the educational simulator to have any graphical appearance we wish, we need use a less limiting development tool to create the multimedia components for its user interface.

A very suitable development environment for the creation of interactive multimedia components is Adobe Flash, which allows us to create animated interactive components whose behaviour can be programmed (and interconnected with a simulation model in our applications). Created Flash components can be easily played back right on web pages (using a built-in interpreter, freely downloadable from the internet), or they can be used as ActiveX components in the creation of educational applications in the Visual Studio .NET environment.

Flash has undergone rather a long development. Originally, it was principally a Macromedia program intended just for the creation of animated images. Over time, there were more opportunities to control the created animations with scripts, whose syntax was gradually developed and extended. Since version 7 (marketed as Macromedia Flash MX 2004), the Flash environment has included a truly object-oriented control language (ActionScript), with syntax very similar to Java, which gives rather easy and sophisticated control over the behaviour of visual interactive elements.

The big success of the Macromedia Flash development environment is based, among other things, on the fact that its creators were rather successful in implementing an interface for artists (creating basic

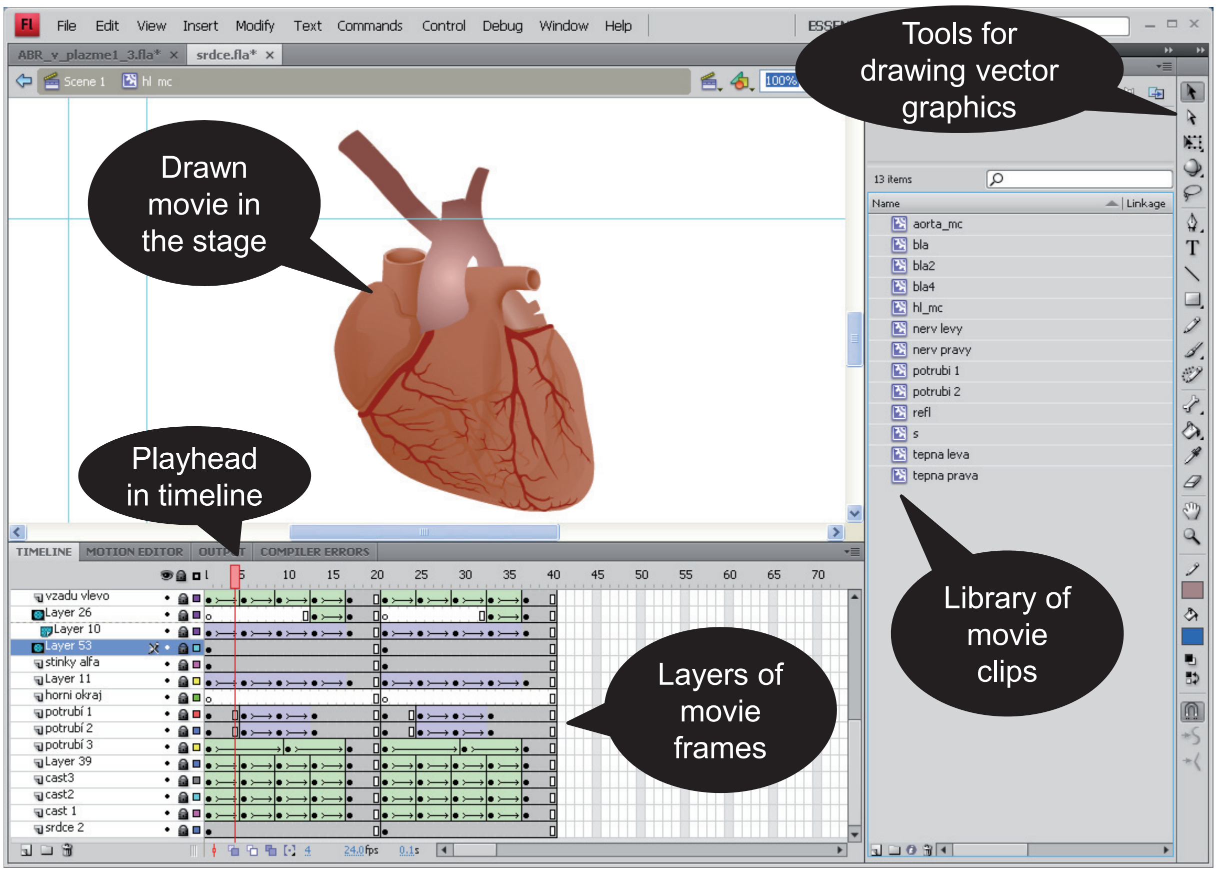


Figure 9: *Adobe Flash provides artists with tools for drawing vector graphics. However, it is also possible to insert a MovieClip instance from a library into the individual layers of movie frames (as in the example shown). The behaviour of each visual (and non-visual) component can be programmed (in a programming window).*

animation elements) and programmers that can use the above-mentioned object-oriented language to make the components interactive.

The basic component of Flash animations is a movie. A movie can be divided into scenes that can be played in sequence or programmatically (out of sequence). A scene consists in a sequence of individual frames. Movies can be linked arbitrarily – each movie can be programmed to load another movie and play it back. This is especially useful in internet applications, where the first part of an animation, when loaded and launched, can download the other parts from the server in the background.

The creation of computer animations is based on the traditional method of cartoon animation, with the individual parts of each frame drawn on transparent cells laid over one another. Some cells need not be redrawn for each additional frame, or they just need a slight shift (e.g. the background), while others (e.g. a figure) are redrawn in each frame either completely or partially to produce animated movement.

In Flash (see Fig. 9), each scene consists in several layers that function similarly to cells in the manual creation of an animated cartoon.

Each movie/scene frame has several layers in which individual image elements are stored. The visual elements may be drawn separately in each layer – Flash includes a relatively powerful tool for the creation of vector graphics (naturally, vector graphics may be imported from a number of other external drawing programs). Another possibility is choosing a graphic from a library and creating an instance. However, an instance may be something other than just a static image. An instance may be e.g. a MovieClip, which in fact is an instanced class of a previously created movie. For example, when creating the picture of an aircraft, we can insert an instance of a movie clip with a rotating propeller from a library as an element in one of its layers. A special type of movie clips is Buttons, for which it is possible to define not only the appearance (on mouse-over and on click) but also their behaviour to serve the launched event. The actual MovieClip may have a rather complex hierarchical structure – the clip it consists in can contain other movie clip instances. For example, a car MovieClip can contain the MovieClips of turning wheels. Each MovieClip instance has its own properties (coordinates of location in the scene, size, colouring, transparency, etc.), which can be changed dynamically from the program. In addition, the MovieClip class has a number of methods that we can use (e.g. a method for detecting collisions with another MovieClip instance in the scene, etc.).

When creating a MovieClip, we can also program specific methods that we can then call in all of its instances. This allows programming rather complex behaviour for the visual components. It is relatively easy to create special MovieClips as real components and then set up their properties in a special component editor and call their methods during runtime. This gave many creators the opportunity to create (and then distribute or sell) various visual (as well as non-visual) components and contributed to the popularity of Flash among visual artists.

The development environment allows creating individual movies (both graphically and programmatically), testing them and translating them into an intermediate language (as .swf files) that can be interpreted by a freely downloadable interpretive program (Flash Player) and either played back as an independently executable animation or viewed directly in a web browser. In addition, the created .swf file can be interpreted by means of a special ActiveX component that can be built into another program – e.g. into an application created in Control Web or Microsoft Visual Studio. Importantly, the component can exchange messages with the application, allowing us to easily control the behaviour of an interactive animation by another application. The application can also receive messages from the interactive animation, informing it of user interventions.

As a result of the great success of Flash, Macromedia was bought by Adobe and Flash became one of the integral parts of Adobe's portfolio of computer graphics tools.

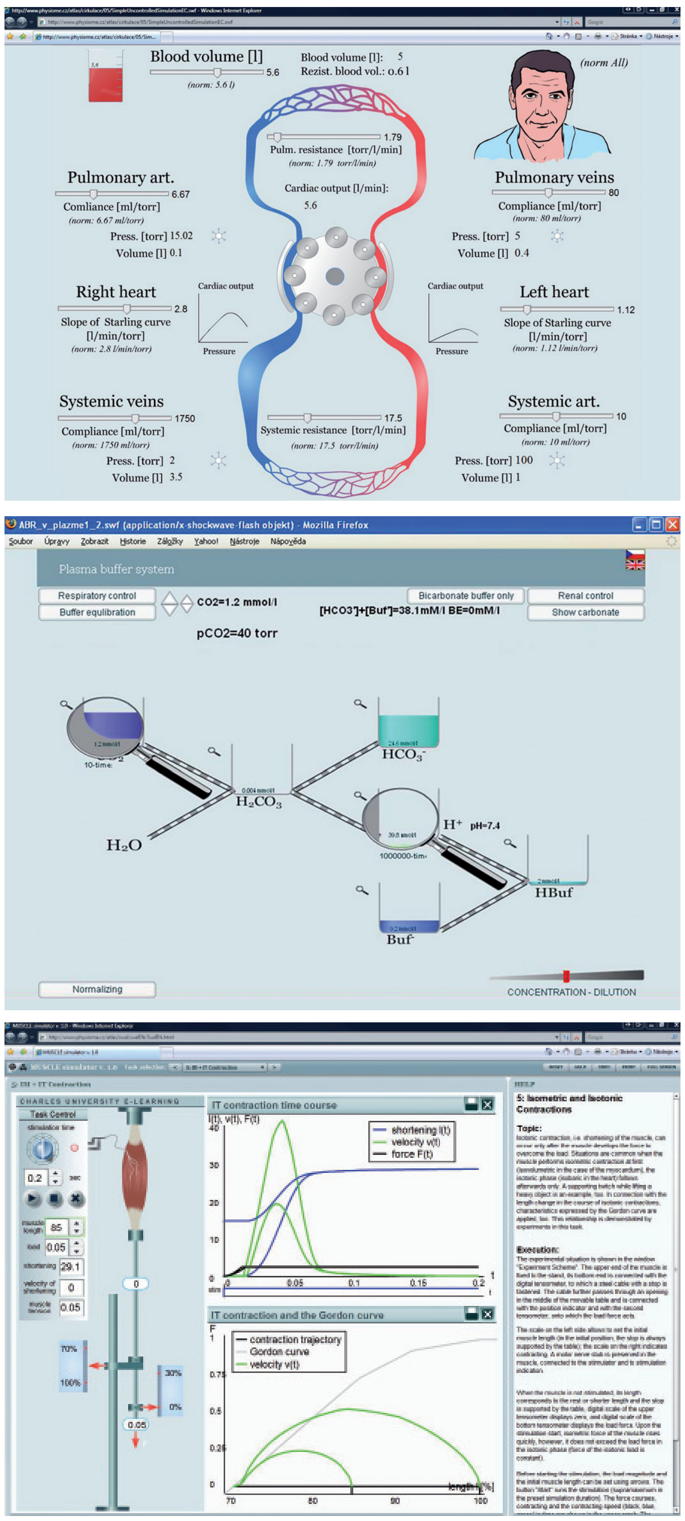
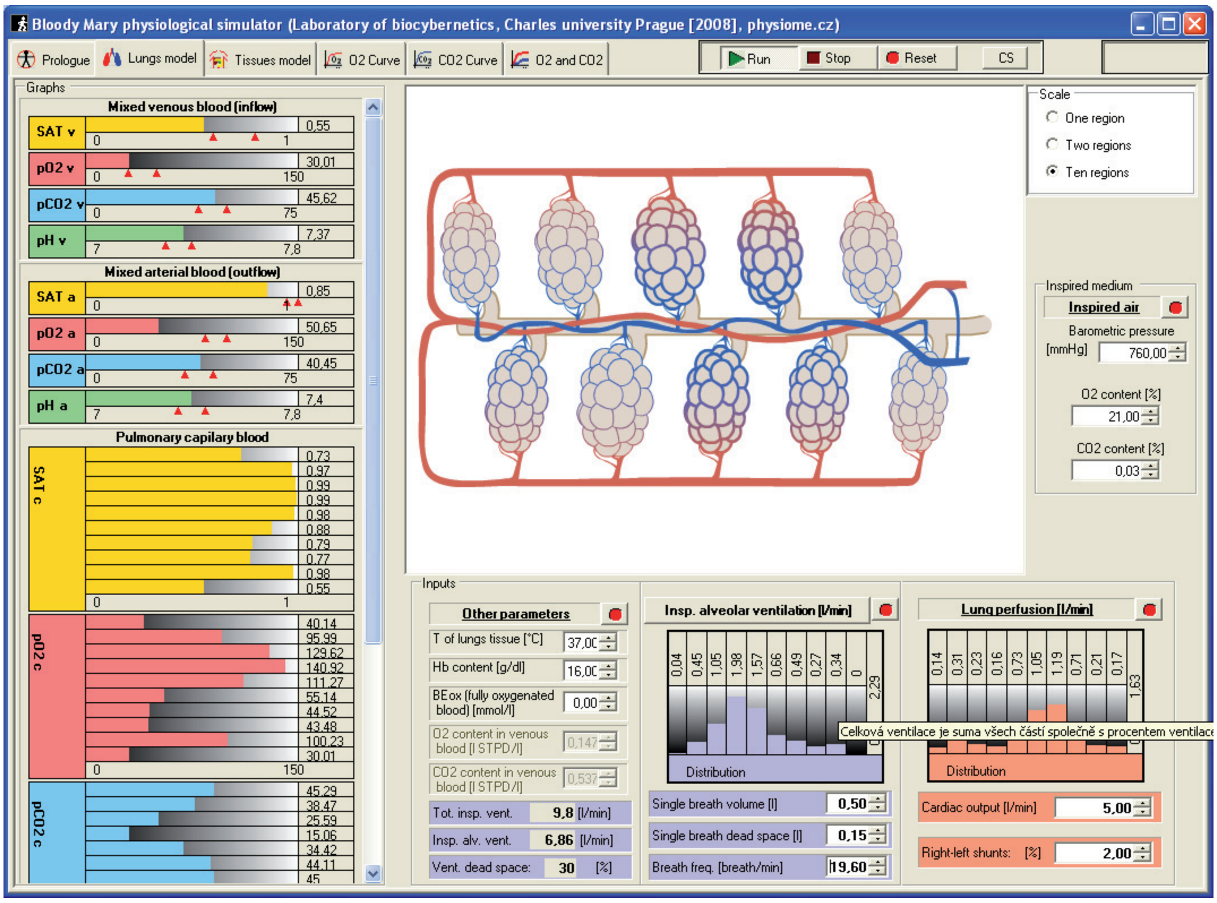


Figure 10: *Adobe Flash can be used to implement simulators that can run right in a web browser – examples of flash simulators for a simple circulation model, acid-base balance and an educational program explaining the mechanical properties of the skeletal and heart muscles by means of simulation games.*

Today, Flash components can be used in RIAs (Rich Internet Applications) – a new generation of multiplatform web applications with elaborate, comprehensive user interface design, created with Adobe Flex, or as desktop applications created with Adobe Air.

The speed of the .swf interpreter (Flash Player) has grown and ActionScript can now be used to create the actual simulation core of educational simulators. The advantage of pure Flash educational ap-


Bloody Mary physiological simulator (Laboratory of biocybernetics, Charles university Prague [2008], physiome.cz)
Prologue
Lungs model
Tissues model
O2 Curve
CO2 Curve
O2 and CO2
Run
Stop
Reset
CS
Graphs
Inputs
Distribution
Celková ventilace je suma všech částí společně s procentem ventilace

Mixed venous blood (inflow): SAT v 0,55 (0–1); pO2 v 30,01 (0–150); pCO2 v 45,62 (0–75); pH v 7,37 (7–7,8)

Mixed arterial blood (outflow): SAT a 0,85 (0–1); pO2 a 50,65 (0–150); pCO2 a 40,45 (0–75); pH a 7,4 (7–7,8)

Pulmonary capilary blood: SAT c 0,73 0,97 0,99 0,99 0,98 0,88 0,79 0,77 0,98 0,55 (0–1); pO2 c 40,14 95,99 129,62 140,92 111,27 55,14 44,52 43,48 100,23 30,01 (0–150); pCO2 c 45,29 38,47 25,59 15,06 34,42 44,11 45

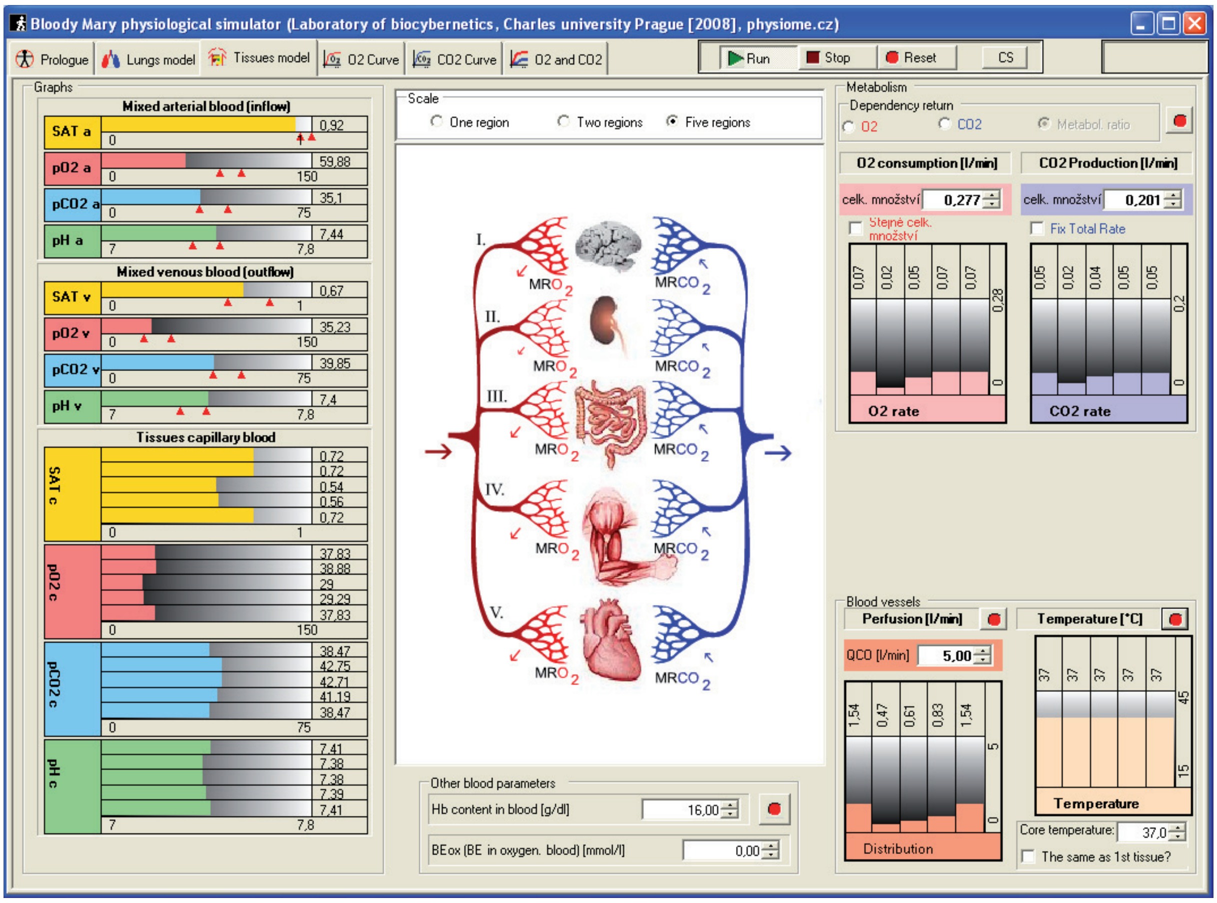
Scale: One region / Two regions / [✓ Ten regions]

Inspired medium: Inspired air; Barometric pressure [mmHg] 760,00; O2 content [%] 21,00; CO2 content [%] 0,03

Other parameters: T of lungs tissue [°C] 37,00; Hb content [g/dl] 16,00; BEox (fully oxygenated blood) [mmol/l] 0,00; O2 content in venous blood [l STPD/l] 0,147; CO2 content in venous blood [l STPD/l] 0,537; Tot. insp. vent. 9,8 [l/min]; Insp. alv. vent. 6,86 [l/min]; Vent. dead space: 30 [%]

Insp. alveolar ventilation [l/min]: 0,04 0,45 1,05 1,98 1,57 0,66 0,49 0,27 0,34 0 (2,29); Single breath volume [l] 0,50; Single breath dead space [l] 0,15; Breath freq. [breath/min] 19,60

Lung perfusion [l/min]: 0,14 0,31 0,23 0,16 0,73 1,05 1,19 0,71 0,21 0,17 (1,63); Cardiac output [l/min] 5,00; Right-left shunts: [%] 2,00



Mixed arterial blood (inflow): SAT a 0,92 (0–1); pO2 a 59,88 (0–150); pCO2 a 35,1 (0–75); pH a 7,44 (7–7,8)

Mixed venous blood (outflow): SAT v 0,67 (0–1); pO2 v 35,23 (0–150); pCO2 v 39,85 (0–75); pH v 7,4 (7–7,8)

Tissues capillary blood: SAT c 0,72 0,72 0,54 0,56 0,72 (0–1); pO2 c 37,83 38,88 29 29,29 37,83 (0–150); pCO2 c 38,47 42,75 42,71 41,19 38,47 (0–75); pH c 7,41 7,38 7,38 7,39 7,41 (7–7,8)

Scale: One region / Two regions / [✓ Five regions]

Metabolism – Dependency return: O2 / CO2 / [✓ Metabol. ratio]

O2 consumption [l/min]: celk. množství 0,277; Stejné celk. množství; 0,07 0,02 0,05 0,07 0,07 (0,28)

CO2 Production [l/min]: celk. množství 0,201; Fix Total Rate; 0,05 0,02 0,04 0,05 0,05 (0,2)

Other blood parameters: Hb content in blood [g/dl] 16,00; BEox (BE in oxygen. blood) [mmol/l] 0,00

Blood vessels – Perfusion [l/min]: QCO [l/min] 5,00; 1,54 0,47 0,61 0,83 1,54 (5)

Temperature [°C]: 37 37 37 37 37 (15–45); Core temperature: 37,0; The same as 1st tissue?

Figure 11: *A complex blood gas transport model – an example of a simulator implemented in Microsoft .NET. The simulator installs on the client computer (by clicking the appropriate button on the web page). However, it requires that the .NET platform be present; if not already installed on the client computer, it was automatically downloaded from a Microsoft server..*

plications (which may be complex RIAs composed in Adobe Flex) is the fact that they can be launched directly from a web browser (having the appropriate plugin) and look the same on all platforms.

We have created some educational simulators and educational multimedia applications with simulation games in this environment (Fig. 10). Flash is also the current platform in which our Atlas of Physiology and Pathophysiology is implemented.

However, the Flash Player environment is still an environment based on the interpretation of Flash .swf files. There is a certain power barrier that we hit with numerically intensive calculations in more complex simulators. The Adobe Flash environment itself is not sufficient for the more complex simulators (yet).

A very suitable platform for the development of simulators that we are using prevailingly now is the Microsoft .NET platform and the Microsoft Visual Studio .NET programming environment, which offers great options to programmers.

In this environment, we are not limited by “premade” user interface elements as in the Control Web environment and can take advantage of the full potential of a modern software development tool; on the other hand, we have to program many elements for our created application on our own. However, we can use graphical user interface components created in Adobe Flash and interconnect them (by means of ActiveX) with the simulator core, which is the simulation model; the graphical components can then behave as puppets controlled by the simulation model (Fig. 11).

8. The brain of an educational application – the simulation model

The implementation of simulation games in an educational program is not trivial. To create simulation models, we use special development tools for the creation, debugging and verification of simulation models, which were mentioned in the previous chapter.

The creation of simulation models in biomedical sciences is research rather than development work, often of a multidisciplinary nature – on the one hand, there is a system analyst – an expert in the formalization and creation of simulation models (theoretical physiologist, creating a formalized description of a physiological system and testing its behaviour by means of a simulation model). On the other hand, there is a traditional experimental physiologist or clinician, who perhaps cannot make head or tail of a physiological system description using differential equations but who can easily see to which extent the behaviour of a computer simulation model corresponds to the biological reality.

In our experience, the communication problem between these two groups of experts may be significantly alleviated by a consistent use of “simulation chips” (Kofránek et al., 2002) and modern development tools for the creation of simulation models. Verified simulation models (whose behaviour corresponds to the biological reality to a given degree of accuracy) can then be used in educational programs.

However, this requires that the model is first transferred from the development environment in which we created, debugged and verified it into the environment in which the actual educational simulator is created. With simple models, this can be done “manually” – as we often do with pure Flash educational simulators.

Nevertheless, we preferred creating software tools that automate this work for us with more complex models. As mentioned above, we created a generator for converting a simulation model from Matlab/Simulink into a driver for a virtual measuring/control card that the Control Web runtime environment communicates with; the generator creates the driver C source code directly from a Simulink model.

Likewise, to facilitate the creation of simulators in Visual Studio .NET (i.e. to avoid having to program a debugged simulation model in Visual Studio .NET “manually”), we developed a special software tool (Stodulka et al., 2007, Kofránek, 2009) that generates a simulation model as a component for .NET from Simulink automatically (see Fig. 12).

9. Combining the model, simulator and animation development platforms

When creating simulators, we had to work with three types of different software tools:

1. Software tools for the creation and debugging of mathematical models that will constitute the basis for a simulator – *Matlab/Simulink*. This environment is useful and efficient for the development of simulation models, but running simulators in it is difficult.
2. A software *tool for the development of the actual simulator* – here we used the Microsoft Visual Studio .NET environment. The simulators were developed in C#. Another environment we used for the creation of educational applications was Control Web from the Czech company Moravian Instruments, especially because it offers excellent possibilities for the quick creation of the user interface of a simulator – but such an interface is too "technical".
3. *Software tools for the creation of interactive multimedia graphics* – the user interface for simulators. Here, we used *Adobe Flash* (formerly Macromedia Flash) for a long time. This tool allows creating interactive animations that can be also programmed using a special programming language, ActionScript. The animations can then be inserted into programs created in the environment as well as into programs created in Control Web. Importantly, the animations can (thanks to the above-mentioned ActionScript programming capability) communicate at the software level with a simulation model programmed in C# in Microsoft Visual Studio .NET. Likewise, the animations can be inserted into Control Web.

As we used different development tools for the creation of simulation models and for the devel-

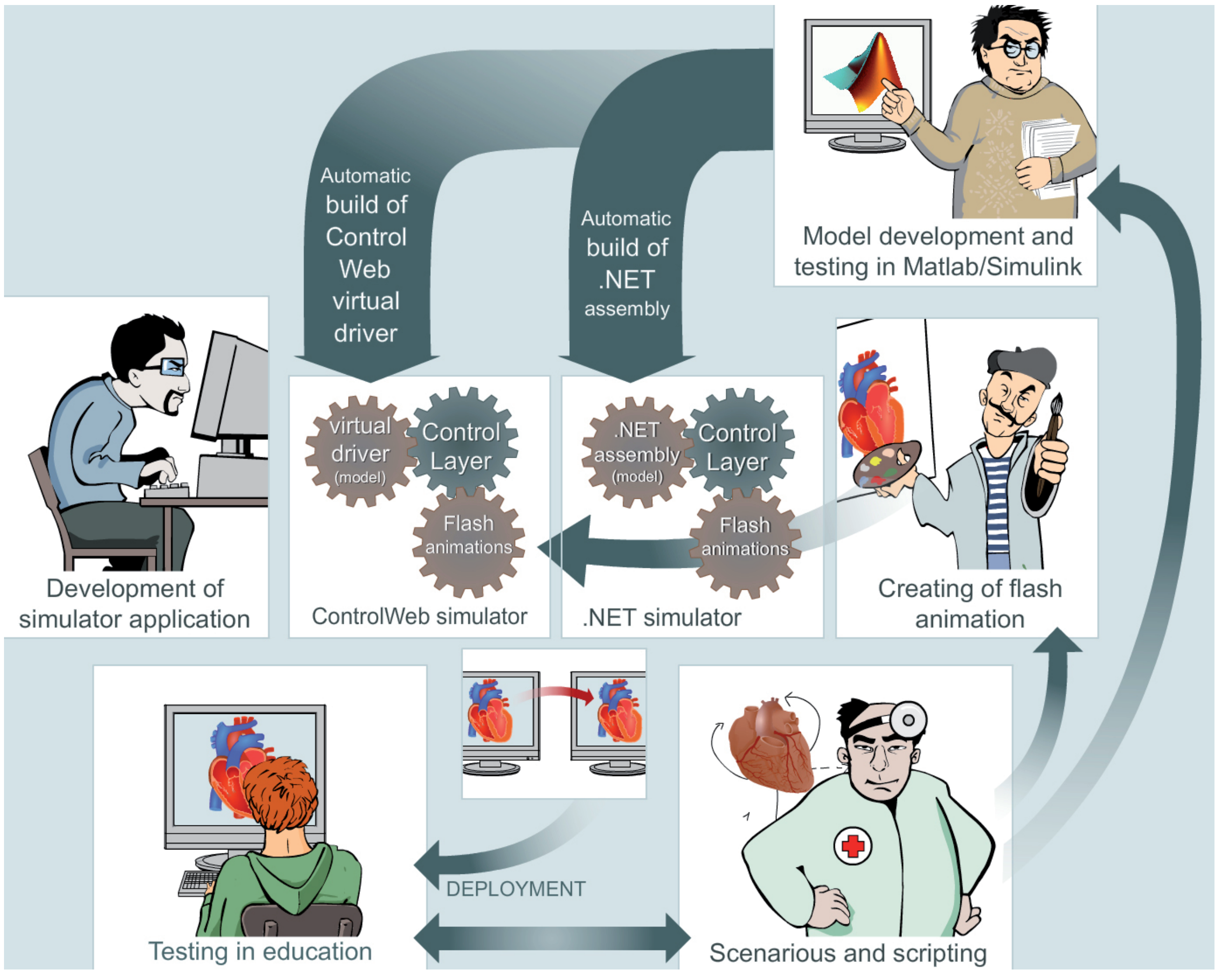


Figure 12: *The original solution for the creative interconnection of tools and applications for the creation of simulators and educational programs using simulation games. The basis of an e-learning program is a quality script, created by an experienced teacher. The creation of animated pictures is up to artists, who create interactive animations in Adobe Flash. The core of a simulator is a simulation model, created in the environment of special development tools for the creation of simulation models. We used Mathworks' Matlab/Simulink for a long time. The development of a simulator is a difficult programming task; to make it easier, we have developed special programs that facilitate the conversion of created simulation models from Matlab/Simulink to Control Web and Microsoft .NET.*

opment of the actual simulator, we had to ensure sufficiently flexible transfer of the results from one development environment to another. Because we created mathematical models in Matlab/Simulink and built the actual educational simulators in the Microsoft Visual Studio .NET environment (or in the Control Web environment), we had to develop software tools that would allow automating the conversion of models from Matlab/Simulink to Microsoft Visual Studio .NET.

Those "interconnection" tools allowed us to develop and continually update a mathematical model in the best environment intended for the development of mathematical models, while developing the actual simulator in Visual Studio .NET (or Control Web) without having to "manually" re-program the mathematical model. They enabled easy multidisciplinary collaboration among project team members – system analysts, creating the mathematical models, and programmers, implementing the simulator.

Nonetheless, this meant working in three software environments and each innovation of a single environment often required updating the relevant interconnection tools.

From the user's point of view – simple models implemented directly in ActionScript on the background of Flash animations could be worked with right in a browser with Flash Player installed. However, more complicated models, such as the complex model of blood gas transport (http://physiome.cz/atlas/sim/BloodyMary_cs/) required, before the first launch, that the model be installed on the client computer (and that the .NET platform be present; if not already installed on the client computer, it was automatically downloaded from a Microsoft server).

To install applications, the user has to have the appropriate administrator rights on his/her computer and in addition, a model running as a stand-alone application connects only indirectly to the web interface where the interactive multimedia lecture is running.

10. Simulators that can be run right in the browser – now an achievable goal (thanks to Silverlight)

In terms of pedagogical effect, it would be useful to be able to run and control even complex models right in a web browser. This has proven to be possible if we create the entire simulator so that it can be run in Microsoft's new environment – *Silverlight*. Silverlight is Microsoft's response to Adobe's ubiquitous Flash Player.

Silverlight is a web platform based on the .NET technology, which fully derives from the operating system and hardware on which an application runs. It is intended for the creation of and interactive work with dynamic online content. It combines text, vector and bitmap graphics, animations and video.

An application runs primarily in a web browser with no need for installation (the only necessary installation is that of the Silverlight plugin itself). By means of a small downloadable component (plugin), Silverlight allows controlling applications interactively in most modern web browsers (Internet Explorer, Firefox, Safari) on various hardware and software platforms. Currently, there is direct support for the Windows and Mac operating systems for the most popular browsers. A fully compatible open-source implementation, Moonlight, is being developed for Linux. Applications created for this platform use a major part of the .NET framework, which is included in the plugin (and can thus perform relatively complicated calculations).

This makes Silverlight a platform that allows distributing *simulators that can run right in a web browser* over the Internet (and for *computers with different operating systems* – the browser just has to have the appropriate plugin).

The Microsoft Visual Studio with extensions for Silverlight can then be used to develop an actual simulator. Simulator implementation languages are C# and F# (a functional language for .NET suitable for the implementation of scientific calculations).

In the end, we will convert the source code for our model created in the *Modelica* acausal modelling language (which was discussed in a previous chapter) into the .NET platform and thus become able to interconnect this modern modelling language easily with an environment in which we will develop a simulator that will be able to run in a web browser window (Fig. 13).

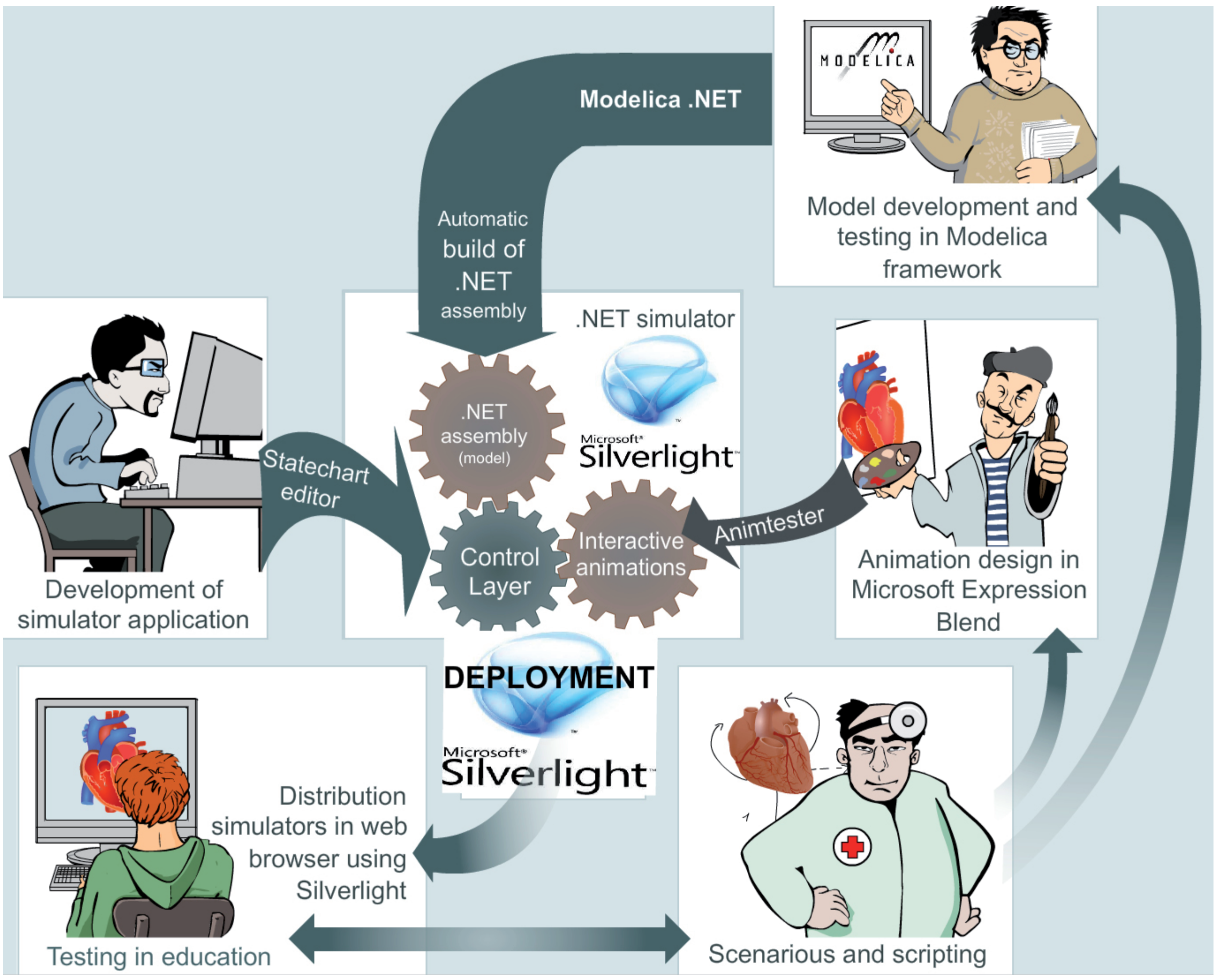

Modelica .NET
Automatic build of .NET assembly
.NET simulator
Model development and testing in Modelica framework
.NET assembly (model)
Microsoft Silverlight
Statechart editor
Control Layer
Interactive animations
Animtester
Animation design in Microsoft Expression Blend
Development of simulator application
DEPLOYMENT
Microsoft Silverlight
Distribution simulators in web browser using Silverlight
Testing in education
Scenarious and scripting

Figure 13: *The new solution for the creative interconnection of tools and applications for the creation of simulators and educational programs using simulation games. The basis of an e-learning program is still a quality script, created by an experienced teacher. The creation of animated pictures is up to the artists, who create interactive animations in Expression Blend. The artist uses Animtester, a software tool we have developed, to create and test animations that will be controlled by a simulation model in the end. The core of a simulator is a simulation model, created in the environment of special development tools for the creation of simulation models. We are now using a very efficient environment that makes use of the Modelica simulation language. We are currently working on a translator from Modelica to a .NET component that will combine with a differential equation solver, also implemented in .NET, to form the "data layer" of a simulator with the implemented model. The user environment is interconnected with the simulation model by means of Data Binding, which ensures smart automatic propagation of values between the layers, i.e. data transmission. We use state automatons (which can be used to remember the relevant model context and user interface context) in the design of the internal application logic. We have also developed a visual environment (Statecharts Editor) that allows designing such automatons graphically, generating their code and debugging them. The resulting simulator is a web application for the Silverlight platform, which allows distributing the simulator as a web application running right in a web browser (on computers with different operating systems – the browser just has to have the appropriate plugin).*

11. Moving animations as puppets on the strings of simulation models

When creating a user interface for an educational simulator, it is very useful to present the simulator outwardly as a moving picture. That is why we interconnect our simulation model with a multimedia application created by means of Adobe Flash (Rusz and Kofránek, 2008, Kofránek 2009).

To give the application a professional appearance, it is necessary to have an artist do the animations – the results are incomparably better than animations created by a graphically gifted programmer. However, this meant putting effort into training artists who had to learn how to work with tools for the creation of interactive graphics. There is a critical shortage of such trained artists in the labour market.

That is why we began to cooperate closely with the Wenceslas Hollar Art School several years ago, opening an “interactive graphics laboratory” as a detached department at this school.

We put much effort into training professional artists to work in the environment of Adobe Flash and other tools for the creation of interactive graphics (including 3D). Our effort brought success. Artists ceased to be shy of computers and quickly realized that a “digital brush” is just another tool for their creative artistic expression and even that managing it would offer them new career opportunities.

We also initiated the establishment of a Higher Art College, which teaches “interactive graphics” as a three-year course. Our laboratory personnel participate, among other things, in teaching at this Higher Art College.

Animated pictures may be controlled by the outputs of an implemented simulation model and graphically represent the meaning of numerical values – e.g. the schematic drawing of a blood vessel may expand or compress, an alveolus may “breathe” deeply or shallowly, a measuring apparatus pointer may move and continuously indicate the value of a model output value read from the simulation model running in the background.

On the other hand, we can enter various inputs into the simulation model by means of visual elements created in Adobe Flash (various push-buttons, knobs, rods, etc.).

12. From Adobe to Microsoft, even in the creation of graphical interfaces

In addition to interconnecting the .NET environment with modelling tools (Modelica or Simulink), it is important to provide easy connection to created graphical user interface components. Flash components can be incorporated into a created simulator by means of the ActiveX component. However, bridging the gap between the two incoherent worlds of Adobe Flash and .NET requires more (manual) programming work.

With the new WPF (Windows Presentation Foundation) technology, it is possible to create complex graphical components containing animations, vector graphics, 3D elements etc. right in the .NET platform (similarly to, and even with potentially more capabilities than in Adobe Flash).

Importantly, the created graphical user interface is directly integrated with .NET, avoiding the necessity to bridge the gap between the non-homogeneous worlds of .NET and Adobe Flash.

An important feature of *Silverlight* is that it *includes native support for animations*. Thus, anima-

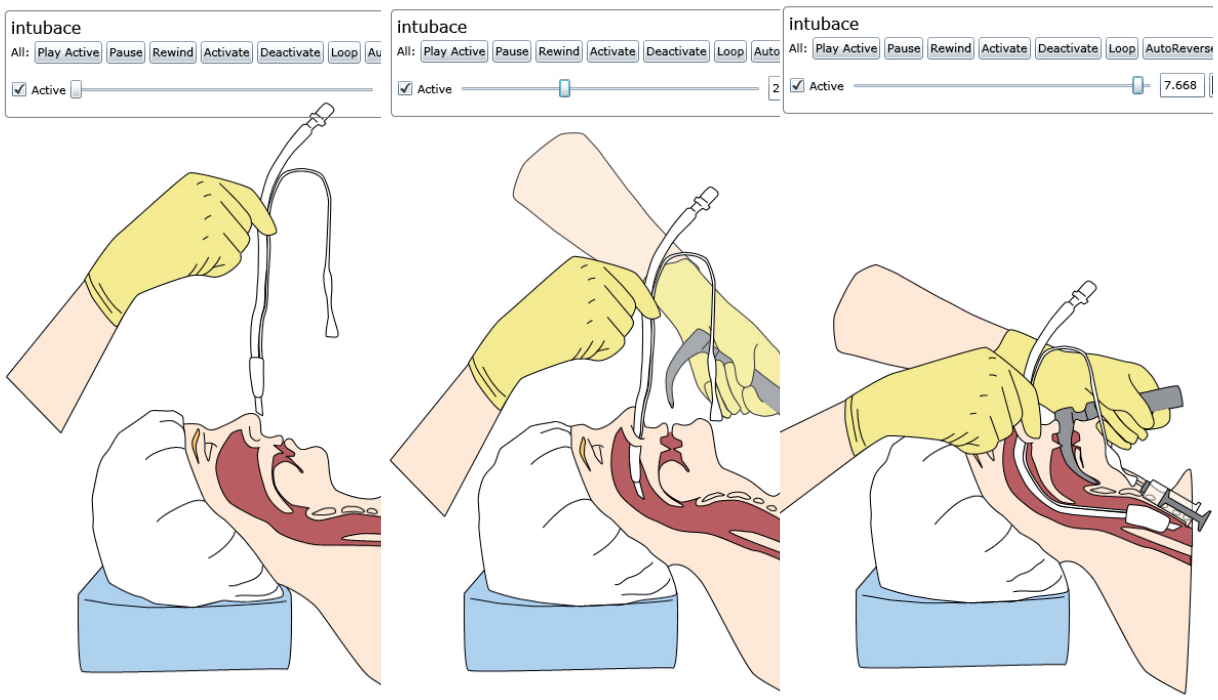
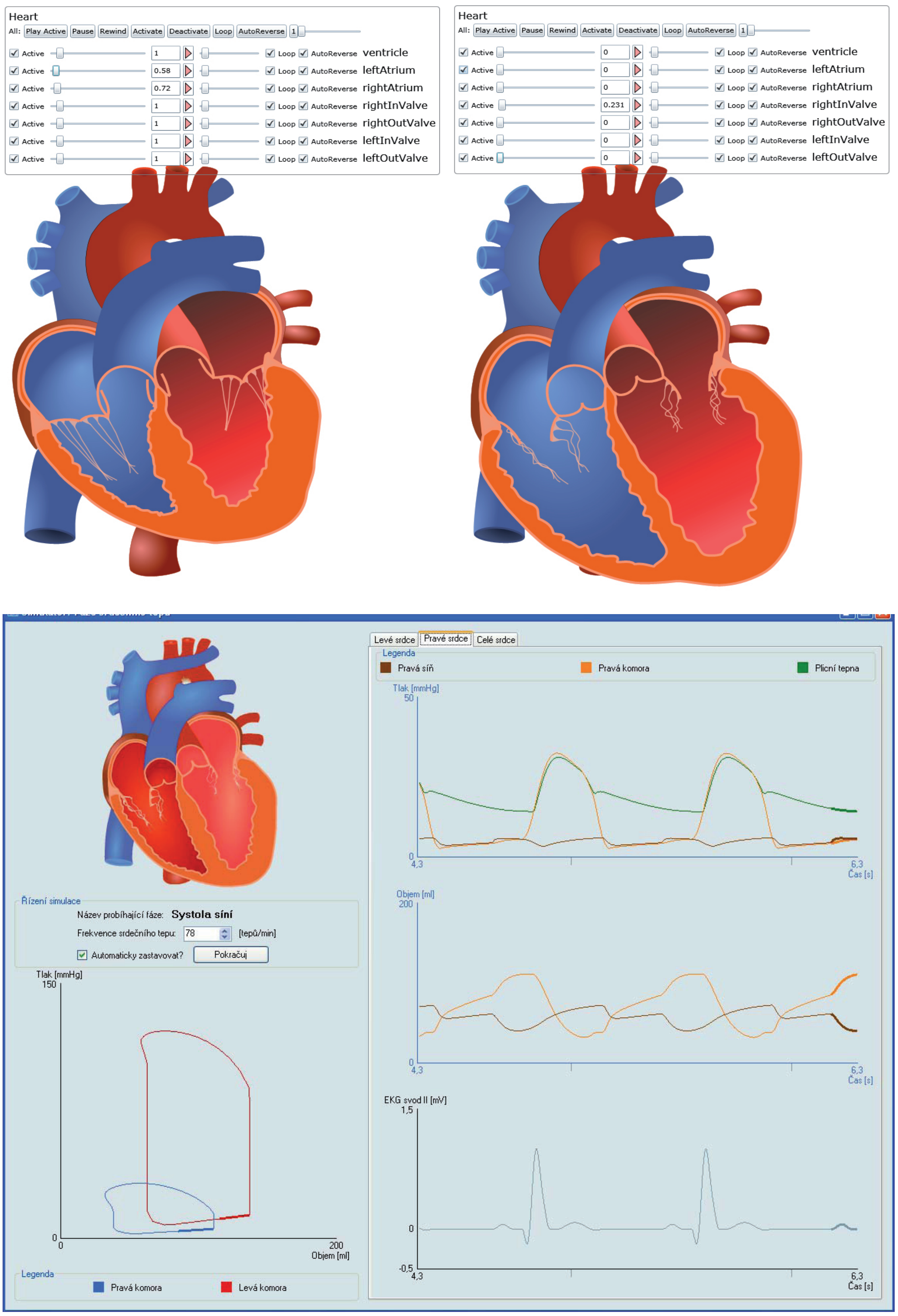

intubace
All: Play Active Pause Rewind Activate Deactivate Loop
Active
intubace
All: Play Active Pause Rewind Activate Deactivate Loop
Active
intubace
All: Play Active Pause Rewind Activate Deactivate Loop AutoReverse
Active
7.668

Figure 14: *Patient intubation. An example of complex animation created in Expression Blend using Animtester.*



Heart

All: Play Active | Pause | Rewind | Activate | Deactivate | Loop | AutoReverse | 1

✓ Active	1	✓ Loop	✓ AutoReverse	ventricle
✓ Active	0.58	✓ Loop	✓ AutoReverse	leftAtrium
✓ Active	0.72	✓ Loop	✓ AutoReverse	rightAtrium
✓ Active	1	✓ Loop	✓ AutoReverse	rightInValve
✓ Active	1	✓ Loop	✓ AutoReverse	rightOutValve
✓ Active	1	✓ Loop	✓ AutoReverse	leftInValve
✓ Active	1	✓ Loop	✓ AutoReverse	leftOutValve

Heart

All: Play Active | Pause | Rewind | Activate | Deactivate | Loop | AutoReverse | 1

✓ Active	0	✓ Loop	✓ AutoReverse	ventricle
✓ Active	0	✓ Loop	✓ AutoReverse	leftAtrium
✓ Active	0	✓ Loop	✓ AutoReverse	rightAtrium
✓ Active	0.231	✓ Loop	✓ AutoReverse	rightInValve
✓ Active	0	✓ Loop	✓ AutoReverse	rightOutValve
✓ Active	0	✓ Loop	✓ AutoReverse	leftInValve
✓ Active	0	✓ Loop	✓ AutoReverse	leftOutValve

Řízení simulace

Název probíhající fáze: **Systola síní**

Frekvence srdečního tepu: 78 [tepů/min]

✓ Automaticky zastavovat? Pokračuj

Figure 15: *A beating heart animation. Model outputs affect the heartbeat phases, the opening and closing of the valves, etc. The Auxiliary Animtester controls are above the actual animation; they help the graphic designer to debug individual sub-animations. Created animation can be subsequently directly connected to model outputs in educational application and model can take full control of this animation.*

tions are directly included in the applications and there is no need to use an additional platform (such as, formerly, Adobe Flash) for the graphical layer.

Our tool of choice for graphic designs and the creation of animations is *Microsoft Expression Blend*, in which we can create graphical interfaces, including those for *Silverlight*.

Microsoft Expression Blend has *interfaces for both the artist* and for *the programmer*. In addition, Microsoft Expression Blend works directly over the project of an application created in Visual Studio .NET. This eliminates the need to transfer the designer's drafts to the application project, which makes *cooperation between the programmer and the design artist much easier*.

The animation approach in Silverlight (originally designed for WPF) is based on *animation using changes to certain scene properties in time* (as opposed to the frame-based system in Adobe Flash).

This allows *animating different scene proportions concurrently and smoothly, independently of one another*, thus creating a kind of *multidimensional animation*, or animation "puppets" that *greatly facilitate the implementation of the representation of various states and values in simulation models*.

To make the creation of the graphical layer more efficient, we have developed an *auxiliary software tool – Animtester*, which designers-artists can use to create and debug such animation "puppets" with no need for further programming (Figs. 14–15). Such created "puppets" can then be *connected directly to model outputs* and there is *no need* to add an *extra program interlayer for data propagation*, as used to be the case with *Flash animations*.

The utilization of the graphical possibilities of *Silverlight* more than substitutes for the original approach of using animations based on Adobe Flash. We *no longer need the Adobe Flash platform* to create animations as a visual interface for simulators; it can be *fully replaced with new animation tools from Microsoft.*

The actual simulator (created in the Microsoft Visual Studio .NET environment), graphical components (created in Microsoft Expression Blend for Silverlight) and the actual distribution channel (Sil-

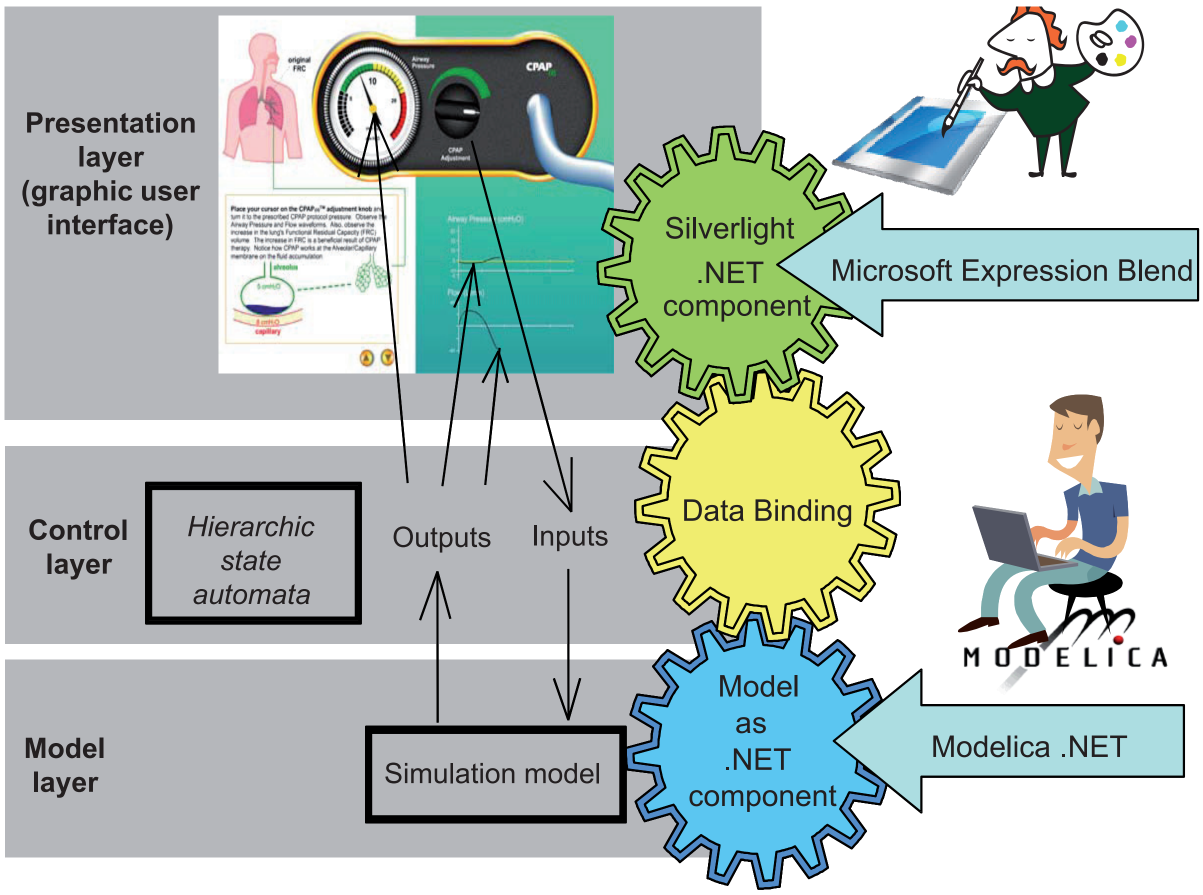

Presentation layer (graphic user interface)
Control layer
Hierarchic state automata
Outputs
Inputs
Model layer
Simulation model
Silverlight .NET component
Microsoft Expression Blend
Data Binding
Model as .NET component
Modelica .NET
MODELICA

Figure 16: *The MVC architecture of a simulator in the making.*

verlight) that allows running the created educational simulation application in a browser can thus be created on a single platform.

13. Simulator structure – MVC architecture

With more complicated architecture, the logic of interconnection between the visual user interface and the simulation model can be rather complex, so it is advisable to place a control layer in between the layer of visual elements and the simulation model layer; the control layer controls all logic for the communication between the user interface and the model and stores the relevant context. Literature calls it MVC simulator building architecture (Model – View – Controller).

Such an arrangement is especially necessary for more complex models and simulators whose user environment is represented by many virtual instruments on multiple interconnected screens. The advantages of such an arrangement are particularly prominent in modifications to both the model and the user interface (Fig. 16).

When creating the control layer interconnecting the simulation model layer with the user interface, we found it very useful to use interconnected state automatons (which can be used to remember the relevant model context and user interface context). Therefore, we created a special software tool that allows us to visually design interconnected state automatons, interactively test their behaviour and automatically generate application source code for the Microsoft .NET environment (Stodulka et al. 2007, Kofránek 2009). This tool allows streamlining the programming of links between the simulation model and the visual objects of the user interface in an educational simulator.

14. Wrapping simulation games in multimedia lectures

One of the projects we develop in our laboratory is the above-mentioned Atlas of Physiology

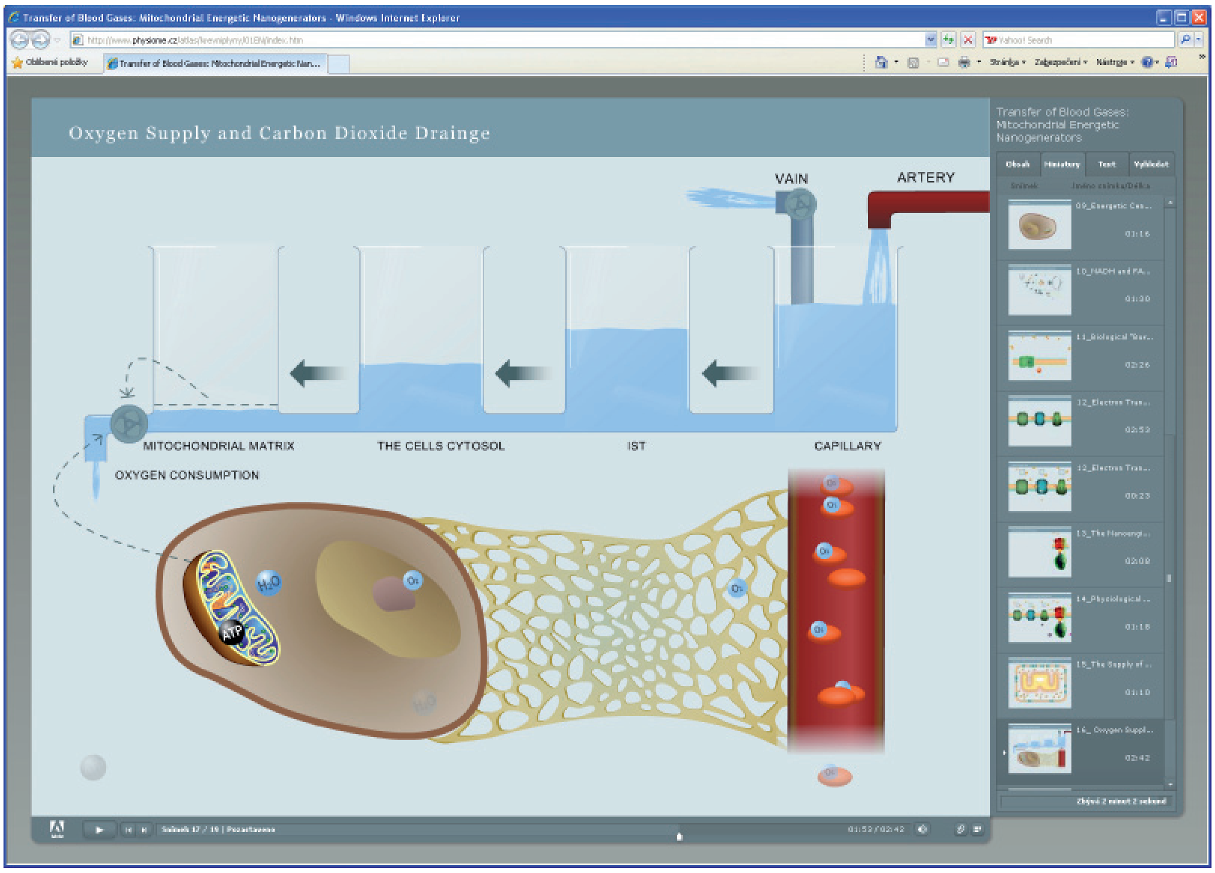
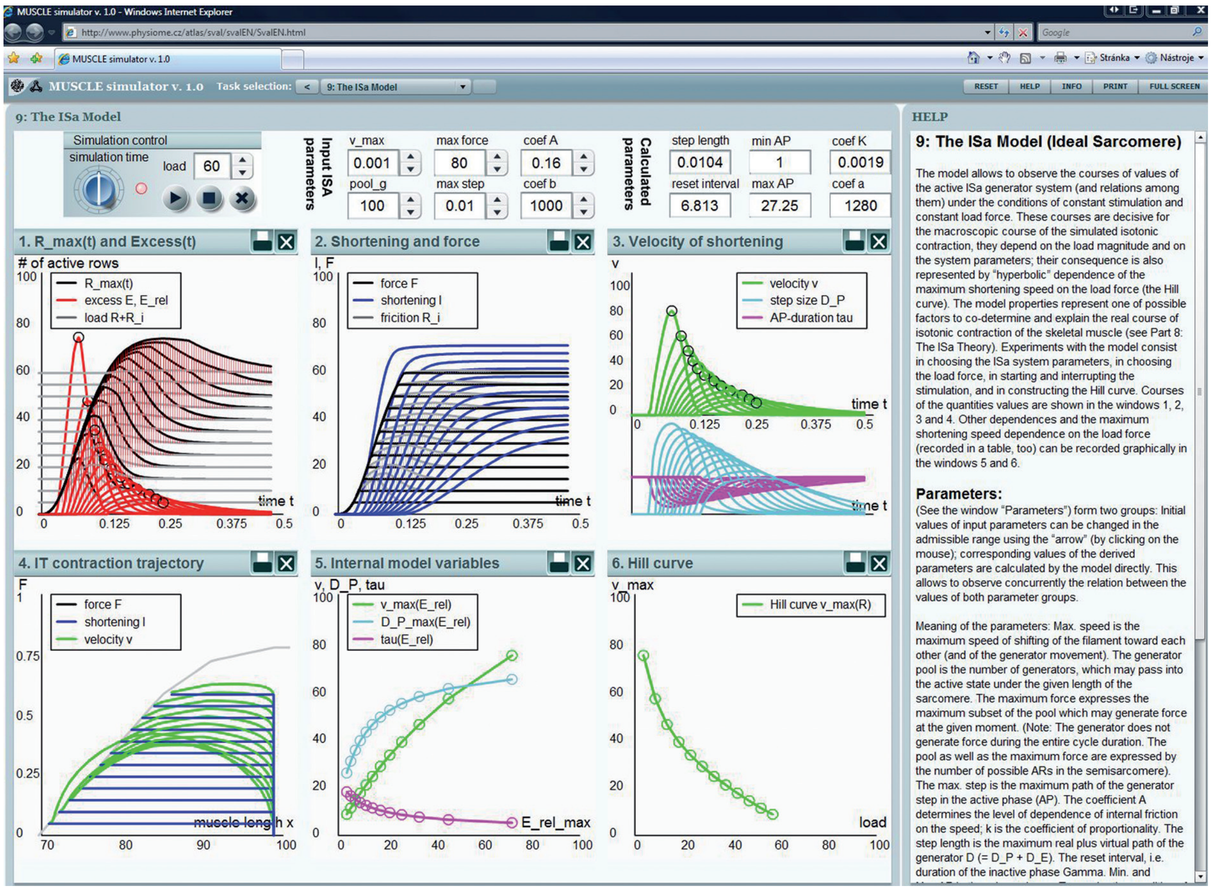

Transfer of Blood Gases: Mitochondrial Energetic Nanogenerators - Windows Internet Explorer
Oxygen Supply and Carbon Dioxide Drainge
VAIN
ARTERY
MITOCHONDRIAL MATRIX
THE CELLS CYTOSOL
IST
CAPILLARY
OXYGEN CONSUMPTION
ATP
Transfer of Blood Gases: Mitochondrial Energetic Nanogenerators

Figure 17: *Audio interactive lecture in the explanatory part of the Atlas of Physiology and Pathophysiology. Every explanation is accompanied by animated images synchronized with the explanatory part. Explanation can be stopped in any moment, to have a more detailed look at the accompanying animation. Explanation including the synchronized animations can also be moved backward using the slide in the bottom part of the player.*



9: The ISa Model

load 60

Input ISA parameters: v_max 0.001; max force 80; coef A 0.16; pool_g 100; max step 0.01; coef b 1000

Calculated parameters: step length 0.0104; min AP 1; coef K 0.0019; reset interval 6.813; max AP 27.25; coef a 1280

HELP

9: The ISa Model (Ideal Sarcomere)

The model allows to observe the courses of values of the active ISa generator system (and relations among them) under the conditions of constant stimulation and constant load force. These courses are decisive for the macroscopic course of the simulated isotonic contraction, they depend on the load magnitude and on the system parameters; their consequence is also represented by "hyperbolic" dependence of the maximum shortening speed on the load force (the Hill curve). The model properties represent one of possible factors to co-determine and explain the real course of isotonic contraction of the skeletal muscle (see Part 8: The ISa Theory). Experiments with the model consist in choosing the ISa system parameters, in choosing the load force, in starting and interrupting the stimulation, and in constructing the Hill curve. Courses of the quantities values are shown in the windows 1, 2, 3 and 4. Other dependences and the maximum shortening speed dependence on the load force (recorded in a table, too) can be recorded graphically in the windows 5 and 6.

Parameters:
(See the window "Parameters") form two groups: Initial values of input parameters can be changed in the admissible range using the "arrow" (by clicking on the mouse); corresponding values of the derived parameters are calculated by the model directly. This allows to observe concurrently the relation between the values of both parameter groups.

Meaning of the parameters: Max. speed is the maximum speed of shifting of the filament toward each other (and of the generator movement). The generator pool is the number of generators, which may pass into the active state under the given length of the sarcomere. The maximum force expresses the maximum subset of the pool which may generate force at the given moment. (Note: The generator does not generate force during the entire cycle duration. The pool as well as the maximum force are expressed by the number of possible ARs in the semisarcomere). The max. step is the maximum path of the generator step in the active phase (AP). The coefficient A determines the level of dependence of internal friction on the speed; k is the coefficient of proportionality. The step length is the maximum real plus virtual path of the generator D (= D_P + D_E). The reset interval, i.e. duration of the inactive phase Gamma. Min. and

Figure 18: *The Internet-based Atlas of Physiology and Pathophysiology is much more than just an animated explanation synchronized with an audio track. Simulation games accompanied by explanation are the foundation of its didactic efficiency.*

and Pathophysiology (see http://physiome.cz/atlas/.). The Atlas (Kofránek et al.2007) is a continually developed internet multimedia teaching aid from the field of normal and pathological physiology that uses simulation models to help explain the functions and disorders of individual physiological systems.

Simulation games are part of multimedia e-learning lectures, based on a script created by an experienced teacher. The teacher proposes the explanatory text and the accompanying illustrations and animations interconnected with the text. Animations are created in Adobe Flash with close cooperation between the teacher and an artist.

The text is then recorded and synchronized with the start of individual animations. Every animation is synchronized accurately with the explanatory text (see fig 17). However, the Internet-based Atlas of Physiology and Pathophysiology is much more than just an animated explanation with an audio track. The foundation of didactic efficiency is represented by explanation accompanied by a simulation games (fig. 18).

Individual components of Atlas are assembled into training lectures

We use the software environment of an Adobe Connect (formerly Macromedia Breeze) server to create and assemble the multimedia lectures.

The Atlas of Physiology and Pathophysiology is an open project. All educational texts, interactive animations and simulation models, including their source code are available to all interested users.

15. From enthusiasm to technology and multidisciplinary collaboration

In spite of the fact that the use of computers in teaching has been the topic of a number of conferences and expert and popularizing articles, in spite of the fact that hardware capabilities and software tools have advanced to a level that allows the creation of sophisticated interactive multimedia, multimedia educational applications have not yet become significantly widespread in medical training.

There are several causes.

- First, the development of educational programs has proven to be more time-consuming and intensive concerning human and material resources than usually planned.
- Second, the creation of high-quality medical training programs requires the multidisciplinary team collaboration of experienced teachers, doctors, mathematicians, physicists, programmers and artists.
- Finally, there must be aptly chosen development tools available for the creative interconnection of the various professions participating in the development of an educational multimedia application (but mastering the tools requires effort and time).
- The demands are even stricter if there should be a simulation program running in the background of the educational application to enable interactive simulation games – such a development team must include experts who are able to design, formalize and debug the appropriate models (doctors, mathematicians, physicists and IT experts).

We believe that the most important result we have managed to achieve in our laboratory so far is that we have built a multidisciplinary team of doctors, mathematicians, programmers and artists who can overcome the above-mentioned barriers.

The interdisciplinary team need tools facilitating communication and cooperation during work on the commonly developed projects (fig. 19).

There are a number of software tools available to support and coordinate team collaboration today. WikiDoc, an open-source tool has proven to be very useful in our laboratory; it allows the easy use of a web interface for the mutual communication of our team members. To take a peep into our interdisciplinary team's "kitchen" and become acquainted with the function of the "wiki" interface, readers are welcome to visit our laboratory's "wiki-web": http://physiome.cz/wiki.

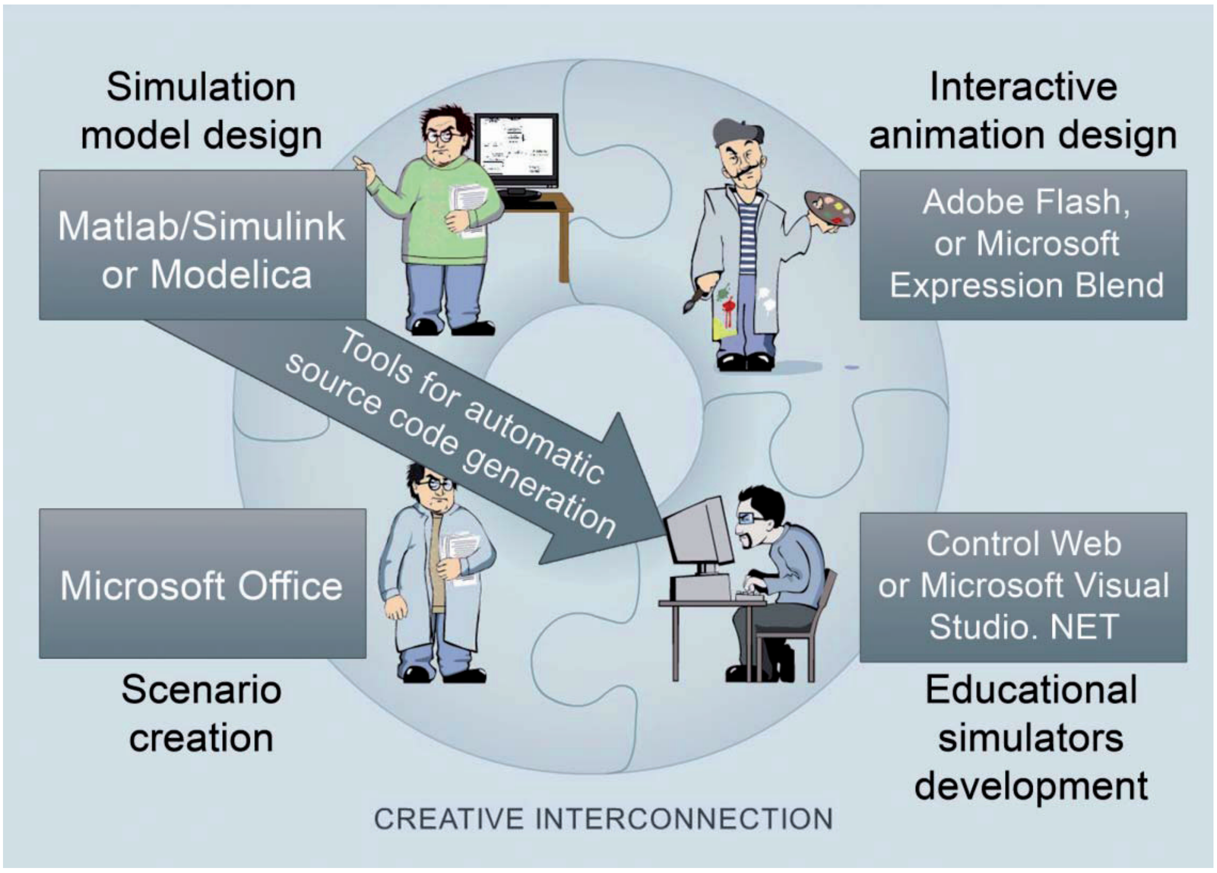

Simulation model design
Matlab/Simulink or Modelica
Interactive animation design
Adobe Flash, or Microsoft Expression Blend
Tools for automatic source code generation
Microsoft Office
Scenario creation
Control Web or Microsoft Visual Studio. NET
Educational simulators development
CREATIVE INTERCONNECTION

Figure 19: *Atlas of Physiology and Pathophysiology is a joint work of a multidisciplinary team of pedagogues, system analyst, artists and computer scientists. In its process of creation, we strive to connect specialists of various professions as well as the development tools.*

16. New challenge and opportunity for colleges and universities

Our times are characterized by major changes in technologies that in turn change our economy, society and lifestyle. The original competition for tons of products turned into competition for better and faster information.

A new market segment is emerging with trade in intangible products, ideas, inventions and know-how. Advances in technology generate pressure on the flexibility of the labour force and increase the demand for continual retraining. Lifelong learning is becoming a necessity in an increasing number of fields.

The creation and implementation of retraining courses and education in the lifelong learning process is facilitated and supported by the use of information technology. E-learning allows increasing the capacity of colleges and universities and can bring them additional income from the development of distance retraining and specialized postgraduate programmes.

From this point of view, e-learning is a new challenge for colleges and universities, one that will need much effort but is also a huge opportunity for further development.

It seems that the time when the creation of educational programs was a matter of verve and hard work for groups of enthusiasts is almost over. The development of modern biomedical educational applications is a demanding and complicated project that requires collaboration by a number of professionals – experienced teachers creating the underlying script, the creators of simulation models, doctors, artists and programmers. To make such interdisciplinary collective development efficient, it is necessary to use specific development tools for each development stage, with sufficient technical support, to allow component-oriented creation of simulation models, development of interactive multimedia and the final integration into a compact whole according to the given script. Mastering such tools requires a lot of effort, but it pays off.

References

[1] Abram, S.R., Hodnett, B.L., Summers, R.L., Coleman, T.G., Hester R.L (2007).: Quantitative Circulatory Physiology: An Integrative Mathematical Model of Human Physiology for medical education. *Advannced Physiology Education*, 31 (2), 202 - 210.

[2] Burkhoff D, Dickstein ML. (2002): *The heart simulator*. Available at: http://www.columbia.edu/itc/hs/medical/heartsim./ Accessed June 3, 2009.

[3] Coleman T. G. and Randall J.E. (1983): HUMAN. A comprehensive physiological model. *The Physiologist*, 26, (1), 15-21.

[4] Coleman T.G, Hester, R., Summers, R. (2008): *Quantitative Human Physiology* [Online] http://physiology.umc.edu/themodelingworkshop/

[5] Comenius, I. A. (1656): *Schola Ludus, seu Enciclopaedea Viva hoc est Janvae Linvarum Praxis Comica.* Sarospatak, 1656. Pars IV.

[6] Davis, M.J. (2001): Basic principles of synaptic physiology illustrated by a computer model *Advan Physiol Educ,* 25, 1 – 12

[7] Davis, M.J, and Robert W. Gore, R.W (2001): Determinants of cardiac function: simulation of a dynamic cardiac pump for physiology instruction *Advan Physiol Educ*, 25, 13-35.

[8] Guyton AC, Coleman TA, and Grander HJ. (1972): Circulation: Overall Regulation. *Ann. Rev. Physiol*., 41, 13-41.

[9] Hester L. R., Coleman T., Summers, R. (2008): A multilevel open source model of human physiology. Tea *FASEB Journal*, 22, 756

[10] Kelsey, R., Botello, M., Millard, B., and Zimmerman, J, (2002): An online heart simulator for augmenting first-year medical and dental education. *Proc AMIA Symp.* 2002, 370–374.

[11] Kofránek J. Anh Vu L. D., Snášelová H., Kerekeš R., Velan T., (2001): GOLEM – Multimedia

simulator for medical education. In Patel, L., Rogers, R., Haux R. (Eds.). *MEDINFO 2001, Proceedings of the 10th World Congress on Medical Informatics.* 1042-1046, IOS Press, London.

[12] Kofránek J., Andrlík M., Kripner T, and Mašek J. (2002): From Simulation chips to biomedical simulator. In: Amborski, K. and Meuth, H. (Eds.) *Modelling and Simulation 2002 Proc. of 16th European Simulation Multiconference*, 431-436, SCS Publishing House,Darmstadt, 2002.

[13] Kofránek, J., Kripner, T., Andrlík, M., and Mašek, J. (2003): Creative connection between multimedia, simulation and software development tools in the design and development of biomedical educational simulators. Orlando. In: *Simulation Interoperability Workshop, Position papers,* Volume II, paper 03F-SIW-102, 677-687. 2003.

[14] Kofránek J., Andrlík M, Kripner T and Matoušek S. (2005): Biomedical Educations with Golem. In Mařík, V., Jacovkis, P., Štěpánková O., and Kléma J. (Eds). *Interdisciplinary Aspects of Human-Machine Co-existence and Co-operation*, 142-151, Czech Technical University, Prague.

[15] Kofránek J., Matoušek S., Andrlík M., Stodulka P., Wünsch, Z., Privitzer P., Hlaváček J., Vacek O. (2007): Atlas of physiology - internet simulation playground. In: *Proceedings of the 6th EUROSIM Congress on Modeling and Simulation,*(Zupanic B., Karba R., Blažič S. Eds.), Vol. 2. Full Papers (CD). 1-9, University of Ljubljana, Ljubljana, 2007.

[16] Kofránek, J (2009): What is behind the curtain of a multimedia educational games? (2009): In *EATIS 09 Contribution Proceedings. Euro American Conference on Telematics & Information Systems*, *Prague 3-5 June 2009*, (Svítek Ed.),Wirelesscom sro., 2009, ISBN 978-80-87205-07-5, 229-236.

[17] Meyers R., Doherty C, Geoffrion L. (2008): *Web-Human Physiology Teaching Simulation (Physiology in Health, Disease and During Treatment)* Available http://placid. skidmore.edu/human/index.php

[18] Oostendorp, T. (2004): ECGSIM: an interactive tool for studying the genesis of QRST waveforms. *Heart.* 9,165-168.

[19] Rusz J., Kofránek J. (2008): Tools development for physiological educational simulators. In *Digital Technologies 2008* (Daša Ticha Ed.) [CD-ROM]. Žilina: University of Žilina, Fakulty of electrical engineering, 2008, vol. 1, ISBN 978-80-8070-953-2,(CD ROM), 1-4

[20] Stodulka P., Privitzer P., Kofránek J., Tribula M., Vacek O.(2007): Development of WEB accessible medical educational simulators. In *Proceedings of the 6th EUROSIM Congress on Modeling and Simulation,* (Zupanic B., Karba R., Blažič S. Eds.) , Vol. 2. Full Papers (CD) 1-6, University of Ljubljana, Ljubljana, 2007.

Acknowledgement

Work on the development of medical simulators is supported under the National Research Programme, project ***MSMT. 2C06031 ("eGolem")***, by research ***MSM 0021620806*** and by ***Creative Connections s.r.o.***

Jiří Kofránek, M.D., Ph.D.
Laboratory of Biocybernetics,
Institute of Pathological Physiology,
U nemocnice 5, 128 53 Praha 2, Czech Republic
email: kofranek@gmail.com
phone: +420-777686868

Marek Mateják, M.Sc.
Laboratory of Biocybernetics,
Institute of Pathological Physiology,
U nemocnice 5, 1208 53 Praha 2, Czech Republic
email: marek@matfyz.cz
phone: +420-776301395

Pavol Privitzer M.Sc., M.D.
Laboratory of Biocybernetics,
Institute of Pathological Physiology,
U nemocnice 5,1208 53 Praha 2, Czech Republic
email: pavol.privitzer@lf1.cuni.cz
phone: +420-608274982