DEVELOPMENT OF WEB ACCESSIBLE MEDICAL EDUCATIONAL SIMULATORS

Petr Stodulka, Pavol Privitzer, Jiří Kofránek, Martin Tribula, Ondřej Vacek

Institute of Pathophysiology
1st Faculty of Medicine, Charles University in Prague

petr.stodulka@gmail.com (Petr Stodulka), pavol.privitzer@lf1.cuni.cz (Pavol Privitzer)

Abstract

One of our main goals is to incorporate the usage of computers into the process of education. The great part of our e-learning activities lies in the development of applications that would demonstrate non-trivial physiological systems behavior, their dynamics and regulation. Students of medicine generally differ from students of engineering and technical universities; being less used to purely abstract mathematical thinking. Therefore it is necessary to transfer the mathematically expressed pathophysiological concepts into more schematic, easily understandable, yet still precise manner. Best results are achieved, when the students find the form familiar, e.g. similar to illustrations from the textbooks. In the process of the development of e-learning simulators, our main focus is creating the model and creating high quality animations to visualize the simulation results. In this paper, we describe our approach to usage of simulators in e-learning, the development of these applications, their layered architecture and technologies we use. We use the Matlab Simulink for creating physiological models, .NET framework or Control Web as the main platforms and Adobe Flash for controllable animations. We describe our original tools for accessing Simulink models from the .NET framework or Control Web and we introduce our approach to maintaining the simulator state based on statecharts.

Keywords: e-learning, Simulation games, Simulators, Statecharts, Flash, Simulink, .NET, Control Web

Presenting Author's biography

Pavol Privitzer, M.Sc., MD graduated in 2001 from the Faculty of Mathematics and Physics, Charles University in Prague, his major being Computer Science. In 2007 graduated as Doctor of Medicine, Charles University in Prague. Currently, he is a Ph.D student of Computer Science in Biomedicine supervised by Jiří Kofránek, M.D.

ISBN 978-3-901608-32-2

1 Simulation games in e-learning

Our main motivation for developing simulators is their ability of delivering deeper insights into real biological systems. Important question is *what* properties a simulator must have to have this ability. Another important question is *who* will use the simulator.

As researchers in simulation and mathematical modeling we are more technically oriented and we have sufficient mathematical skills to understand a mathematically expressed physiological model. For us a simulator is good enough when it shows just graph charts and enables only numerical inputs.

However, our simulators are used mainly by medical students and such technically oriented interface seems to them too raw and non-attractive. We have realized that for a simulator to be successful in medical education it must have some gaming elements in it or even its whole architecture should be similar to a computer game. This is why we call our applications also *simulation games*. See also [1]. This concept is nowadays recognized worldwide and there is a great movement in the e-learning research and the computer game industry called *immersive learning simulations*, which can be divided into *serious games* and *educational simulations* [4].

Generally, children are attracted by beautiful colors, shapes, movements and by the possibility to play. Analogically, e-learning applications based on simulation models should be styled by an artist, animated and highly interactive, much like successful computer games are. The concept of learning by playing games is fundamental to education itself and goes back as far as to the work of the great Czech scholar and didactic Jan Amos Comenius (1592-1670) *Schola ludus* (school as a game / school by play) [3].

Also our research process can be viewed from this perspective. We have fun when we are developing a simulation model; we are playing with the simulation, changing the structure of the model, switching signals and so on. At the same time we are learning and deepening our insight of a real biological system because we are comparing and verifying the simulations against real data and/or our observations.

We want to deliver this immersive inner experience and fun to our medical students because then the learning and also teaching is in the form of a game.

There are two processes which can be employed for education by simulation: *creation* of a model and *playing* the game based on a model. Both can be used separately or at the best together in one application.

We use Simulink for creating mathematical models. It is too raw and technical for medical students even if we would use prepared higher level subsystem blocks. We need to develop some kind of a high level visual system language for the creational part of the game and also accompanying software framework which can be embedded in our applications. This is going to be a part of our future research and development.

For simulators we are using high quality animations designed by our artists. Art styled animations provide more attractive interfaces for our audience. This is trying to fill the gap between technical versus educational simulator and it is our main technology improvement in development of e-learning simulators based on more complex models.

2 Architecture of e-learning simulators

The simulator application itself consists of three parts – the model, a user interface and a control layer. The model is the data and computational core of the simulator. The user interface must properly present the behavior of the model and give the user the possibility to control it. The middle tier, so called controller, establishes communication between the model and the GUI and controls the simulation flow. This approach is known as the MVC architecture (*model – view – controller*) or the UCM architecture (*user interface – control object – model layer*).

2.1 Model layer

The model is the implementation of the problem domain. Our models are typically capable of producing a timeline of outputs as they run. The inputs are set at the beginning and can also be changed at run time. The run of the model is controllable - it can be paused, resumed or reset to the beginning. In the future we are going to implement richer time control of the model (rewinding), model cloning (to be able to show differences between several possible progressions of the model depending on different inputs) and the replacement of several implementations of the model (or its parts) at runtime. That would allow the user to focus on selected topic and replace the general simulation model with another more precise.

2.2 Presentation layer

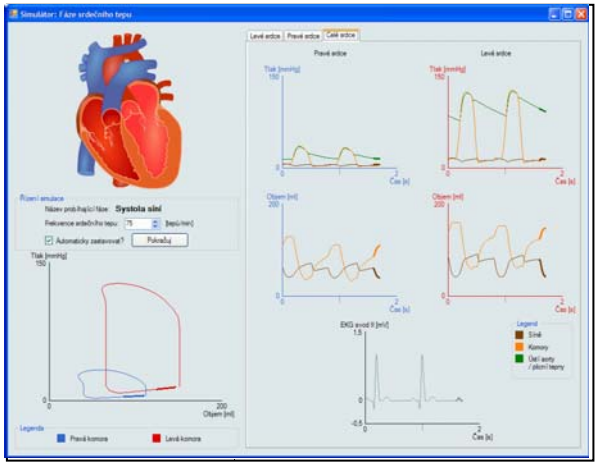

Fig. 1 Heart phases simulator (for .NET).

ISBN 978-3-901608-32-2

Rich visualization of the problem that lets the user interacts with the model usually consists of common UI controls like buttons or sliders, output control like graphs and controllable animations that can also optionally serve as an input.

2.3 State management

Since the logic of a simulator can be rather complex we found out that we need a way to maintain the state of the simulation (and of the application as a whole) and to react to its change. The trivial solution – set of state variables with centralized control logic – soon becomes too complicated and difficult to maintain as the complexity of the simulator grows. We have found the solution in hierarchical state automata (or Statecharts). We use the statecharts for design of the simulation flow logic and timing. Afterwards they are automatically used to control the application.

Hierarchical state automata are a concept introduced by David Harel [5] for defining complex state information and well-defined transitions between states. Harel's statecharts were further extended in the Matlab Stateflow tool and one variant is now a part of the UML 2.0 specification. Statecharts are based on finite state automata and extend them by providing high level modeling possibilities such as hierarchy of states and concurrency (more active states at the same time). The hierarchy reflects the fact that subsystem cannot be activated unless its supersystem is active. The concurrency (sometimes referred as "and" machines) brings the possibility to model several parallel subsystems in one statechart. The automaton is controlled by events coming from outside or raised in the automaton itself, which can cause a change of a state and an execution of an action. Actions can be defined in states and on transitions. Additionally statecharts allow specifying guard expressions on transitions.

We map user interactions onto events for the state machine and the machine can in response control the model (starting it, pausing, resetting), the controls of the user interface (enabling/disabling some controls, setting boundaries of the input controls) or change the interconnection infrastructure (switching controls to display another model outputs, disabling some connections). See also [2].

We have developed a standalone IDE for visual design and debugging of hierarchical state automata. The statechart is exported in the form of its source code and compiled to be used as a part of the simulator application.

2.4 Interconnection infrastructure

After the creation of our first few simulators we felt a need for an infrastructure that would help us bind the UI controls with the model data. We developed a interconnection middleware framework that supports binding several UI controls to one model input or output, two-way propagation of data, which ensures

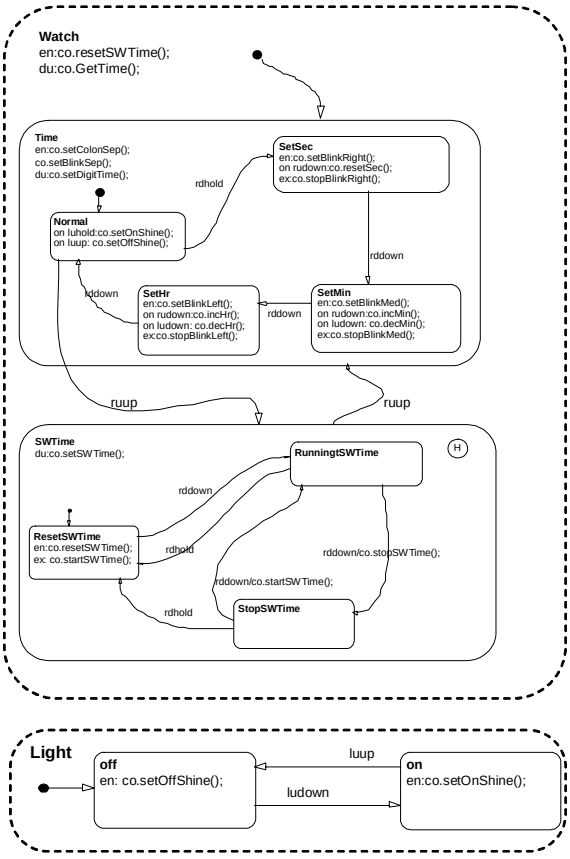

Fig. 2 Sample statechart. Fragment of the wristwatch simulator.

that all controls display consistent values, custom data conversion, automatic checking of input boundaries and custom handlers on value change. Connections can be temporarily enabled or disabled. This solution automates the data flow control and makes it trivial to add or remove UI controls. In the future we are going to extend this infrastructure with the possibility of recording of user actions and the work of a simulator. Replaying will give us a possibility to create tutorials.

3 Software technologies

We are using more software technologies to build simulators, because we haven't found a single environment that would satisfy all our needs regarding to effective visual expressiveness of physiological models, richness and maturity of the resulting simulator's user interface and the ability to produce standalone and web-accessible applications. Generally we use either Microsoft .NET or Control Web as the base platforms we build upon. The models are created in Matlab Simulink and the animations used as a user interface are created in Adobe Flash.

We have spent significant amount of time on development of our own unique tools that allow the Simulink model to be used in a .NET or Control Web application. Flash animations are inserted in the form of ActiveX objects.

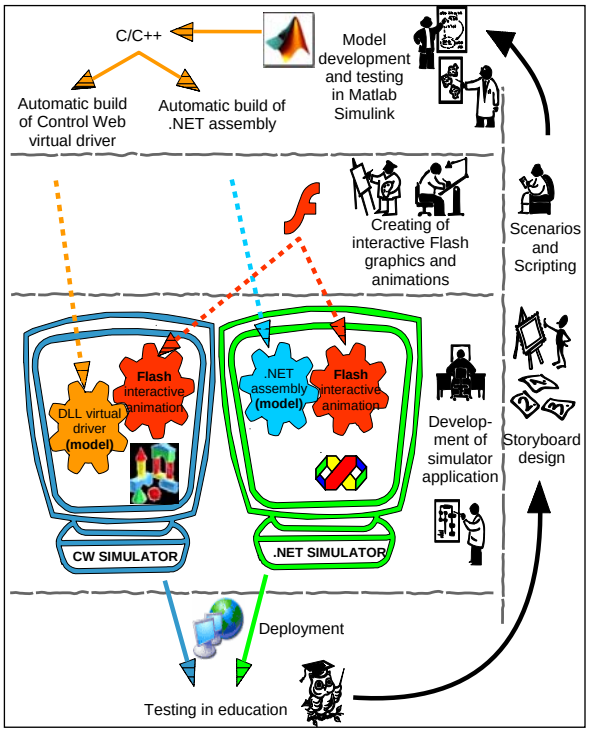

Fig. 3 Development cycle of a simulator application.

3.1 Simulink

Matlab Simulink is a modeling tool that provides graphical design and simulation of mathematical models. Elements of a model can be grouped and hierarchically ordered, giving rise to subsystems with user defined inputs and outputs. We have adopted the concept of *simulation chips* that represent the building blocks of our simulation models.

Another Matlab toolbox, Real-Time Workshop (RTW), exports the model as a C/C++ code. This code consists of a routine that executes the model logic, a structure that holds the model data and an ODE solver. We have developed two new RTW targets (we call them Wizards) that transform this code to be compatible with the .NET or Control Web. For the .NET the code is wrapped in a C++ .NET class that exposes the model's inputs and outputs as its interface. For Control Web the code is transformed into a C++ class that implements the interface making it possible to be used as a virtual driver in a CW application. Thus, the final output of the Wizards is a Matlab independent dynamic library.

3.2 The .NET platform

The .NET represents a modern and strong platform that allows rapid and comfortable application development with support for visual design of the GUI, creation of custom visual controls and the web deployment.

The .NET platform is nowadays most supported for Windows, but open source projects such as Mono or Portable.NET promise the future of portable simulators based on this platform.

3.3 Control Web

Control Web (Moravian instruments inc., Czech Republic) is a platform mainly intended for development of industrial visualization and control applications on the WIN32 platform. It is optimized for real-time controlling and visualization. Its visual framework is component based and gives a great efficiency and rapid application development.

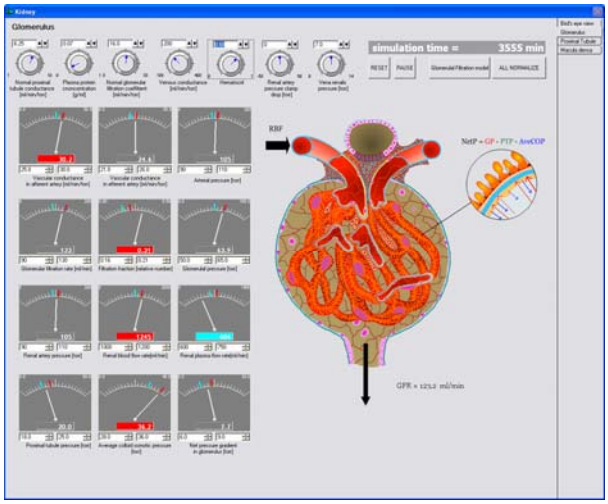

Fig. 4 First prototype of renal function simulator (Control Web implementation).

We used the Control Web platform mainly before the release of the .NET, but we still support it for its advantages such as rapid prototyping and easy visual interconnection of model and visual controls.

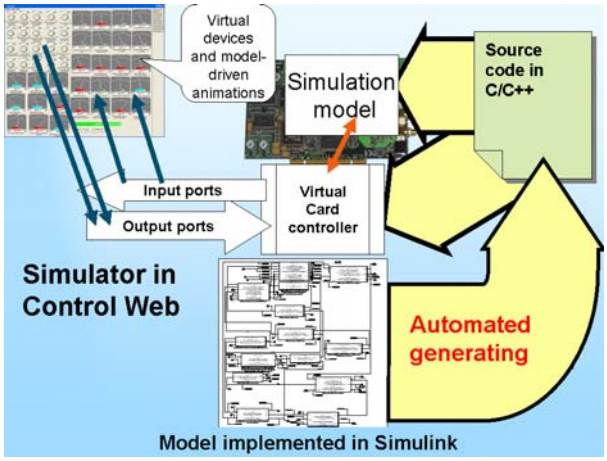

Fig. 5 Virtual device generation for Control Web.

We have a redistributable of CW runtime at our disposition and we have developed a simple tool (SimPlayer) that allows us easy distribution of the CW simulators over the web. Unfortunately there is only Win32 runtime for Control Web.

ISBN 978-3-901608-32-2

3.4 Flash

Although it is possible to create the presentation layer of the application using standard or custom made .NET (or CW) visual components, for more sophisticated visualizations (such as an animation of a beating heart) we use movie clips created in Adobe Flash. Fundamental feature of these animations is the possibility to control them programmatically (using Action Script) and thus affect the presentation. It is also possible to get the user input from a Flash clip and that way create animated user controls.

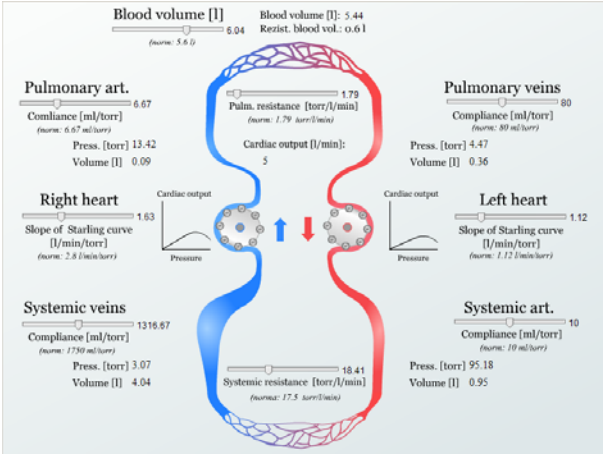

Fig. 6 Right heart failure demonstrated using simple model of circulation (Flash explanation model)

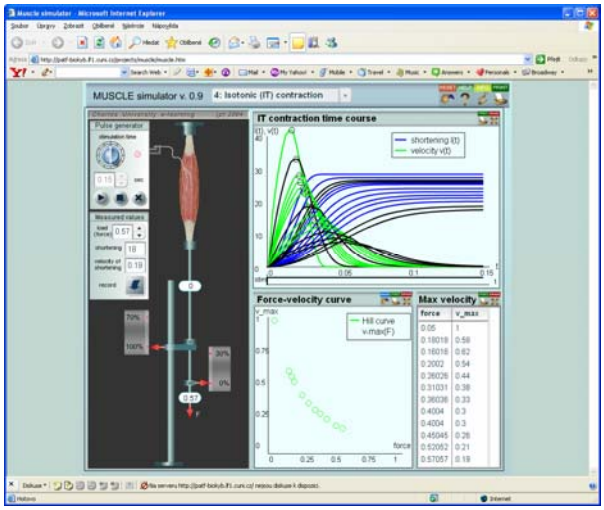

Fig. 7 Properties of muscle (Flash exercise model)

As the execution speed of the Flash and Action Script is increasing over the time, it is a good platform for implementation of simulators based on less complex models. Pure Flash based simulators are very portable and in-browser what is the most desirable deployment scenario for our simulation games. Some of our Flash based simulators can be seen in our Atlas of Physiology and Pathophysiology, cited in other article of the Eurosim 2007 conference [6].

In our complex simulators based on the .NET or Control Web the Flash animations are embedded as ActiveX objects. Their behavior is than controlled by the underlying model layer.

3.5 Deployment

Pure Flash simulators give the best portability and the easiest web deployment. Unfortunately, we have no technology for exporting Simulink to Action Script and therefore only simulators based on simpler model can be implemented as pure Flash applications.

Control Web based simulators need the Control Web runtime on the client machine. Deployment of CW simulators consists of writing a simple SimPlayer XML description file which defines the location of the simulator (zipped) and location of the runtime (installer). The SimPlayer utility transparently ensures the presence of the runtime, downloads, installs and runs the simulation.

Most of our simulators are based on the .NET framework, which offers simple way of the web deployment using the ClickOnce technology. This technology creates a description manifest file recognized by the framework that allows the application to be transparently downloaded from the internet and run in a security sandbox.

4 Future development

The longer way we have passed, the more we see what can be created.

Our vision is becoming much wider than producing high quality simulation games for education. We see a great potential in exchanging scientific information using open simulation games as a medium instead of a linear text with static pictures. Nonlinear information exchange will be the future way of sharing scientific knowledge. Knowledge expressed in a visual system language, which is linked to multimedia and directly showing its behavior will provide the platform for rapid education of the next generation. With this vision we are developing our tools and frameworks.

In the modeling area we are extending our approach to object-oriented component based modeling. It gives the advantage of direct implicit formulation of mathematical equations instead of the need of defined causality in signal based block diagrams. Component based modeling seems more convenient and it is more relevant in physical modeling. An OOP modeling language is probably the way how we will design a high-level visual system language for the creational part of our future simulation games.

In the software technology, the compatibility and portability is a big problem to solve. The .NET world is promising a solution. Seamless interlanguage communication and binary portability will be the properties of the .NET to make it suitable for implementation of our tools and frameworks. Recently Microsoft announced its new Silverlight technology which is similar to Flash technology but it is based on the .NET platform.

ISBN 978-3-901608-32-2

This would give us the possibility to create, for example, an in-browser educational simulator based on a very complex model with a rich graphical interface. This simulator will run both on Windows and on Mac OS platforms without a change.

Our next work in the field of hierarchical state automata is to develop a plug-in for Visual Studio to more integrate statecharts into our simulator development process.

5 Conclusion

We have shown our motivations for using simulators as simulation games in medical education. Our main improvement is usage of rich graphical user interface as a gaming element in simulators based on complex models.

We use Matlab Simulink for physiological modeling, Adobe Flash for controllable animations and the .NET or Control Web as the base environment. We have developed a unique way of exporting the Simulink model into the .NET framework and into the Control Web platform. We have created an original integrated environment for a design of statecharts and their use in the process of controlling the simulator.

6 References

[1] Kofránek J, Andrlík M, Kripner T, Stodulka P.: From Art to Industry: Development of Biomedical Simulators. *The IPSI BgD Transactions on Advanced Research* 1 #2[Special Issue on the Research with Elements of Multidisciplinary, Interdisciplinary, and Transdisciplinary: The Best Paper Selection for 2005], 63-68. 2005. New York, Frankfurt, Tokyo, Belgrad. 2005. Journal (Full)

[2] Kofránek J, Andrlík M, Kripner T.: Multimedia educational simulators in pathophysiology - how to design and why to use them. In: Gamal Attiya and Yskandar Hamam editors. *Proceedings of the 5th EUROSIM Congress on Modeling and Simulation. Full Papers CD Volume.*; Marne la Vallee, Paris, France: Eurosim - Francosim - Argesim; 2004. p. Simulation in Education 22-27.

[3] Comenius Johann Amos (1592-1670): The Great Didactic. ed. by M. W. Keatinge (PDF files at Roehampton), available on: http://onlinebooks.library.upenn.edu/webbin/book/lookupid?key=olbp34684

[4] Clark Aldrich's web page: http://clarkaldrich.blogspot.com/2007/03/immersive-learning-simulation_14.html

[5] Harel, D.: Statecharts (1987): A visual formalism for complex systems. *Science of Computer Programming*, vol. 8, 1987, 231-274.

[6] http://physiome.cz/atlas

7 Acknowledgement

This research was supported by aid grant MŠMT 2C06031 and BAJT servis s.r.o company.

