LEAVING TOIL TO MACHINES - BUILDING SIMULATION KERNEL OF EDUCATIONAL SOFTWARE IN MODERN SOFTWARE ENVIRONMENTS

Jiří Kofránek, Marek Mateják, Pavol Privitzer

Laboratory of Biocybernetics, Dept.of Pathophysiology, 1st Faculty of Medicine, Charles University, Prague

Abstract

In past, simulation models had been created directly in the software development environment used for making the educational software itself. Recently, specialized development tools rather than universal programming languages have been used for making and verification of simulation models. Common choice is that of the block oriented environments (e.g. Simulink). In these environments, hierarchically connected blocks that process input information to output information are used and connected when creating a simulation model. Signals flow in connections between individual blocks, transmitting the values of individual variables from the output of one block to the input of other. The structure of the interconnected network preserves the algorithm of calculation, i.e. clarifies how the values of particular variables are calculated step by step. The whole process is known as causal modelling. The limitation of the causal approach to modelling is that it reflects more the calculation procedure than the actual structure of the modelled reality. The physical structure of the modelled world is captured only by the structure of calculation. Causal modelling is especially arduous for intricate, hierarchically organized models, for instance the models of complex physiological systems, which biomedical simulators are based on. At present, new simulation environments have become available. Fundamental innovation of these environments comprises of the possibility to describe individual parts of the model as a system of equations directly describing the behaviour of that part and not the algorithm of solving of these equations. Model description is declarative (the structure of mathematical relationships instead of algorithm of calculation is described) and notation is therefore acausal. Acausal modelling environments work with interconnected components (i.e. blocks) as well. A component represents an instance of class for which equations or parameters are defined. Components are linked through connectors that are defined more precisely then usual, as they themselves help to define the system of equations. Thus, the connections between components do not define the calculation procedure but rather the modelled reality. The exact algorithm and method of solving the equations is "left to the machines". When large and complex systems are modelled in acausal simulation environments (e.g. based upon modelling language Modelica), the whole process can be facilitated so substantially, as to make it the breakthrough innovation in building the simulation kernel of biomedical educational software. Advantages of the acausal approach are demonstrated using Modelica implementation of a large-scale physiological system model "Quantitative Human Physiology" as an example.

Keywords: Modeling, Simulation, Simulator

In place of an introduction – a web of physiological regulations

Thirty-six years ago the Annual Review of Physiology published an article (Gyuton et al. 1972) which at a glance was entirely different from the usual physiological articles of that time. It was introduced by a large diagram on an insertion. Full of lines and interconnected elements, the drawing vaguely resembled an electrical wiring diagram at first sight (Fig. 1). However, instead of vacuum tubes or other electrical components, it showed interconnected computation blocks (multipliers, dividers, adders, integrators, functional blocks) that symbolized mathematical operations performed on physiological variables (Fig. 2). Bundles of connecting wires between the blocks indicated the complex feedback interconnection of physiological variables at first glance. The blocks were arranged in eighteen groups that represented individual interconnected physiological subsystems. In the centre was a subsystem representing circulatory dynamics – linked through feedback links with other blocks: From the kidneys to tissue fluids and electrolytes to autonomic nervous control and hormonal control including ADH, angiotensin and aldosterone (Fig. 3).

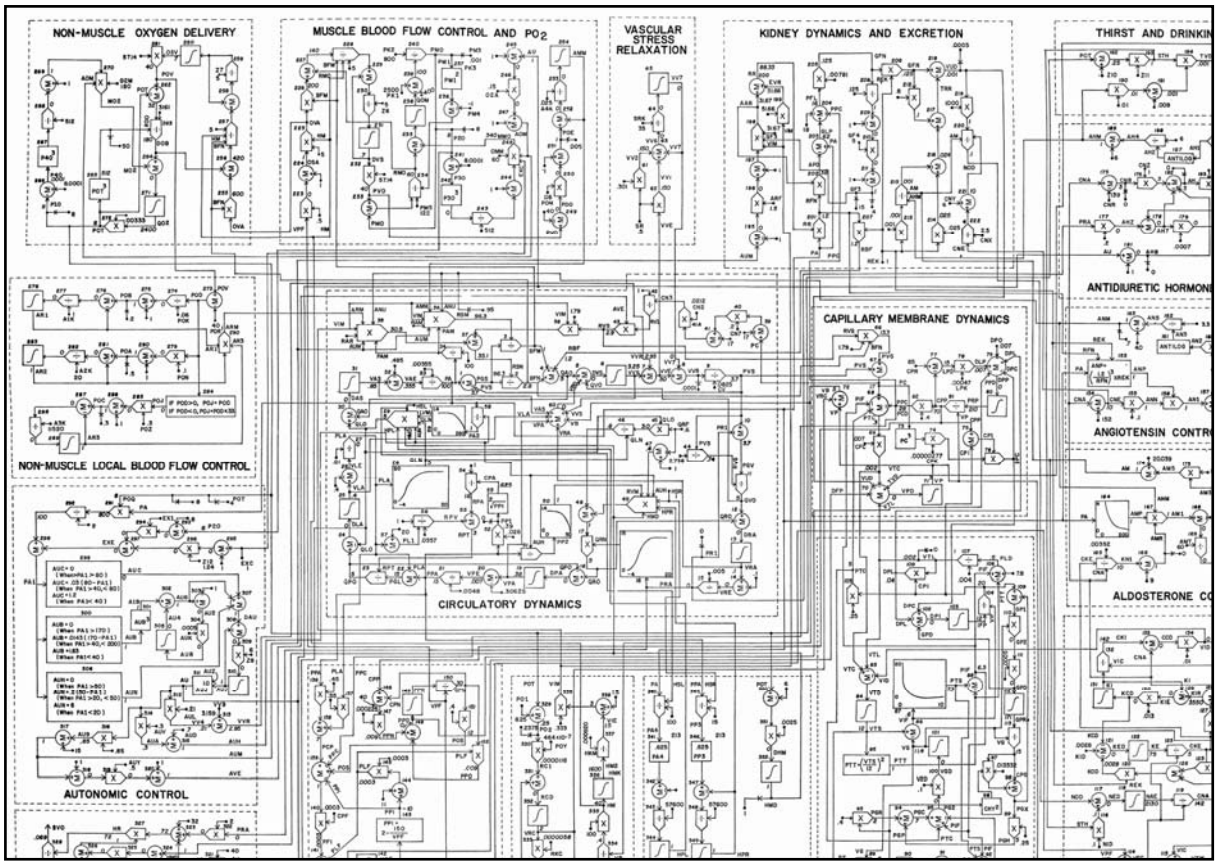


Figure 1: Guyton's blood circulation regulation diagram from 1972.

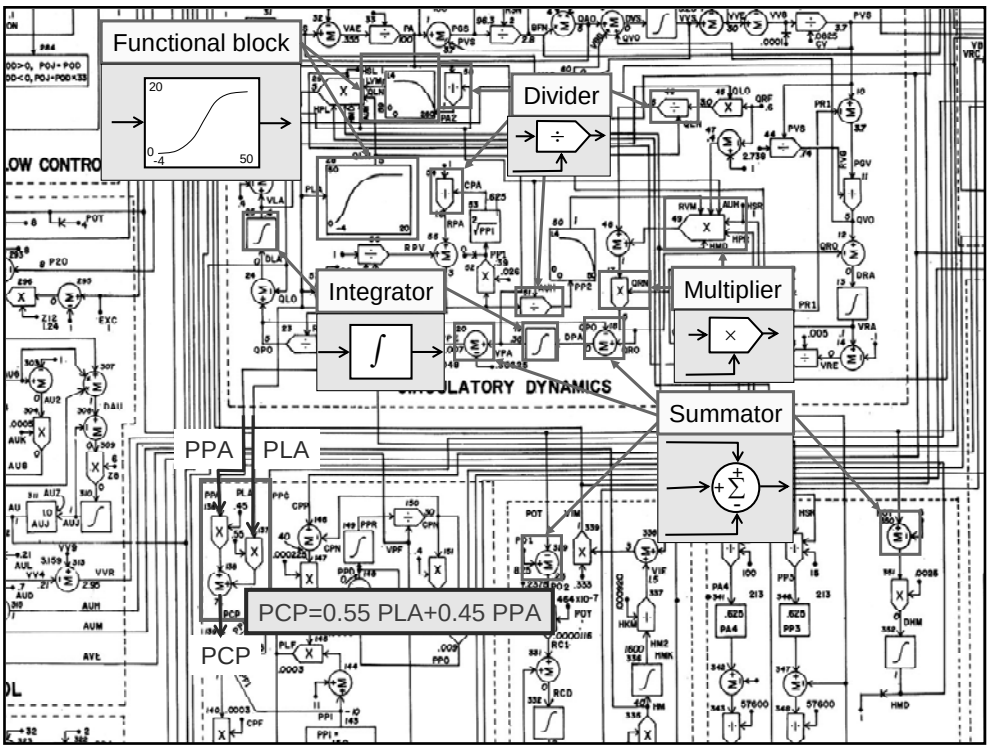


Figure 2: Individual elements in the block diagram of Guyton's model represent mathematical operations, the interconnection of elements represents equations in a graphically expressed mathematical model..

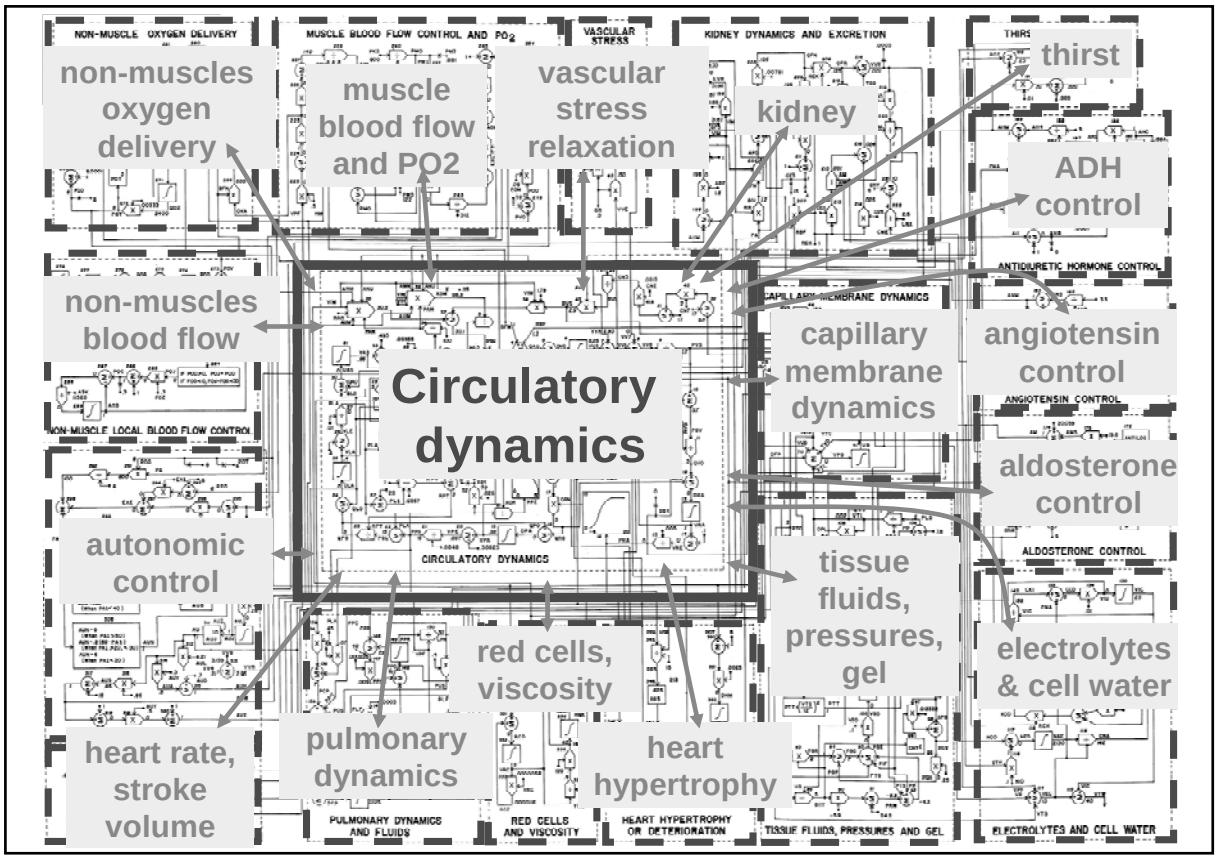


Figure 3: Individual interconnected subsystems in Guyton's model.

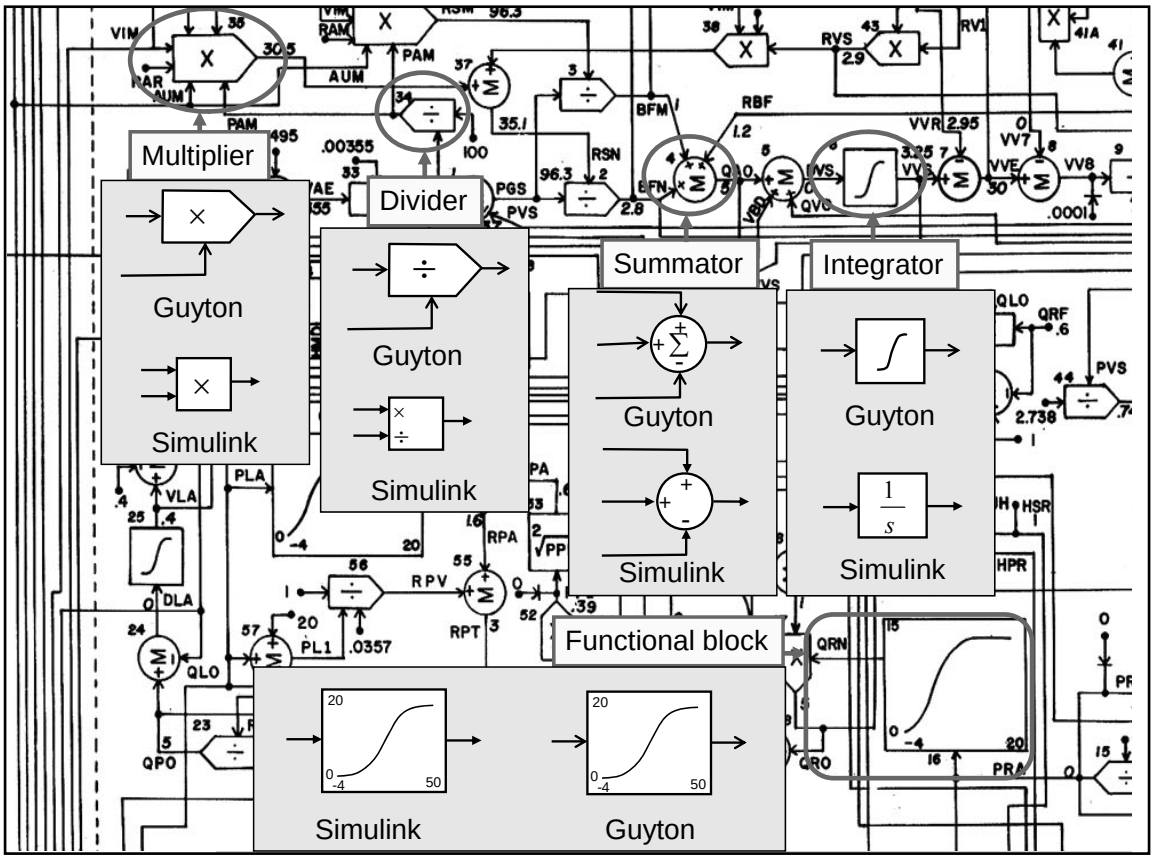


Figure 4: Appearance of blocks in Guyton's original graphical notation and in Simulink.

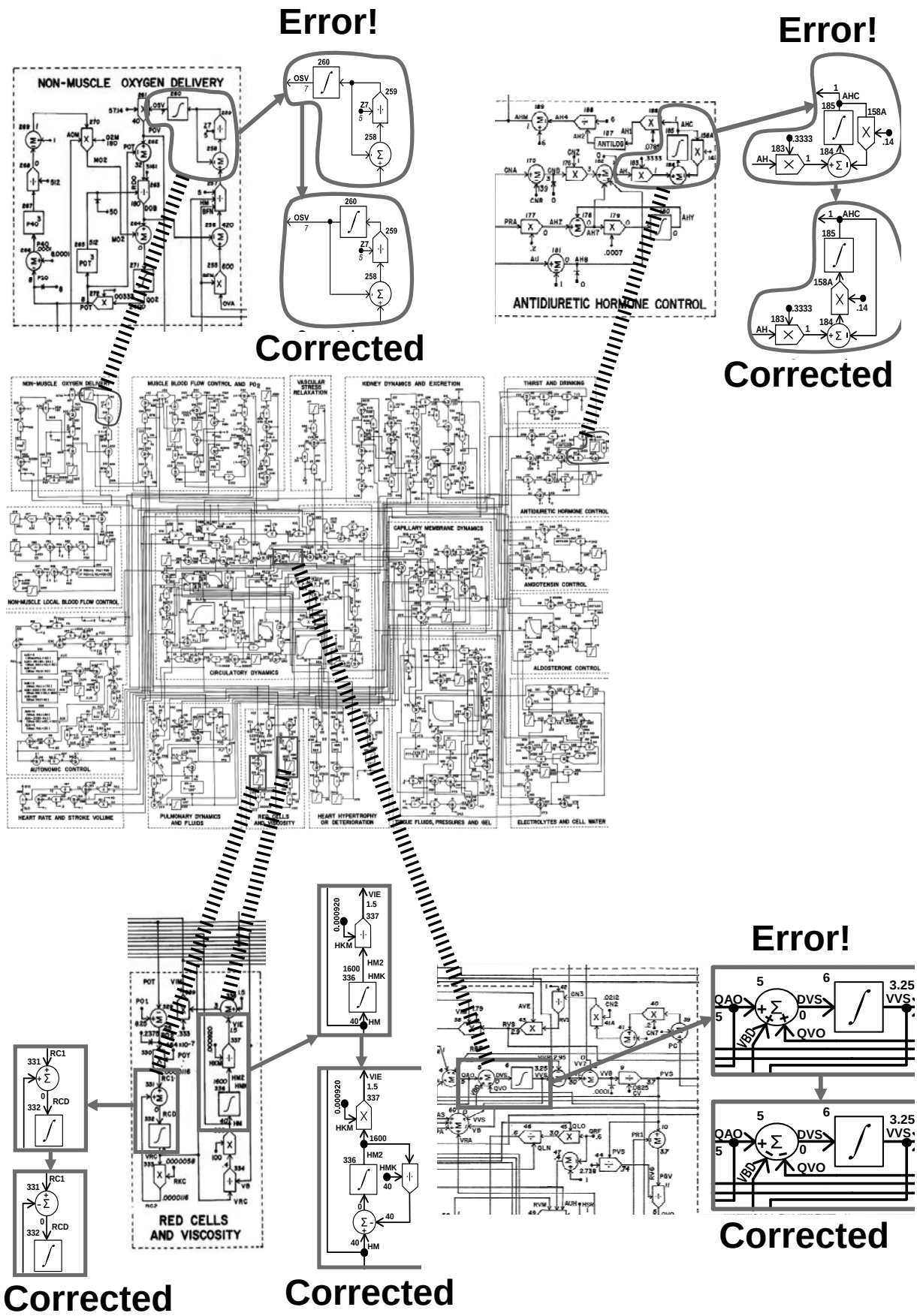


Figure 5: Correction of errors in Guyton's original diagram.

In this entirely new manner, using graphically represented mathematical symbols, the authors described 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, etc. Instead of an extensive set of mathematical equations, the article used a graphical representation of mathematical relations. This syntax allowed depicting relations between individual physiological variables graphically in the form of interconnected blocks representing mathematical operations. The whole diagram thus featured a formalized description of physiological relations in the circulatory system using a graphically represented mathematical model.

The actual description of the model in the article was mostly represented by the elementary (but fully illustrative) drawing. Comments on and reasons for the formulation of the mathematical relations were very brief, e.g.: "Blocks 266 through 270 calculate the effect of cell pO2, autonomic stimulation, and basic rate of oxygen consumption by the tissues on the actual rate of oxygen consumption by the tissues". This required exceptional concentration (and certain physiological and mathematical knowledge) from the reader to be able to understand the formalized relations between physiological variables.

A monograph (Guyton et al. 1973) published a year later, in 1973, explained a number of the adopted approaches in greater detail.

We do not see a mathematical representation of reality very often in biology and medicine. It should be noted that the process of formalization, i.e. the translation of a purely verbal description of a given network of relations into the formalized language of mathematics, is delayed in biological and medical sciences in comparison with engineering sciences, physics or chemistry. While the process of formalization in physics started as early as the seventeenth century, in medical and biological sciences it has been relatively delayed due to the complicacy and complexity of biological systems and has only advanced with cybernetics and computer technology. The methodological tool here is computer models built on a mathematical description of biological reality.

Formalized descriptions in physiology have been used since the late sixties (since the pioneering works of Grodins et al., 1967, describing respiration). Guyton's model 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. Just as theoretical physics tries to describe physical reality and explain the results of experimental research using formal means, "integrative physiology" strives to create a formalized description of the interconnection of physiological controls based on experimental results and explain their function in the development of various diseases.

From this point of view, Guyton's model was a milestone, trying to adopt a systematic view of physiological controls to capture the dynamics of relations between the regulation of the circulation, kidneys, the respiration and the volume and ionic composition of body fluids by means of a graphically represented network.

Guyton's graphical notation of a formalized description of physiological relations provides a very clear representation of mathematical correlations – the blocks in network nodes represent graphical symbols for individual mathematical operations and the wires represent individual variables. Guyton's graphical notation was soon adopted by other authors – such as Ikeda et al. (1979) in Japan and Amosov et al. (1977) in the former USSR.

However, the graphical notation of the mathematical model using a network of interconnected blocks was only visualization when created – Guyton's model and later modifications (as well as the models of other authors that adopted Guyton's representative notation) were originally implemented in Fortran and later in C++.

Today's situation is different.

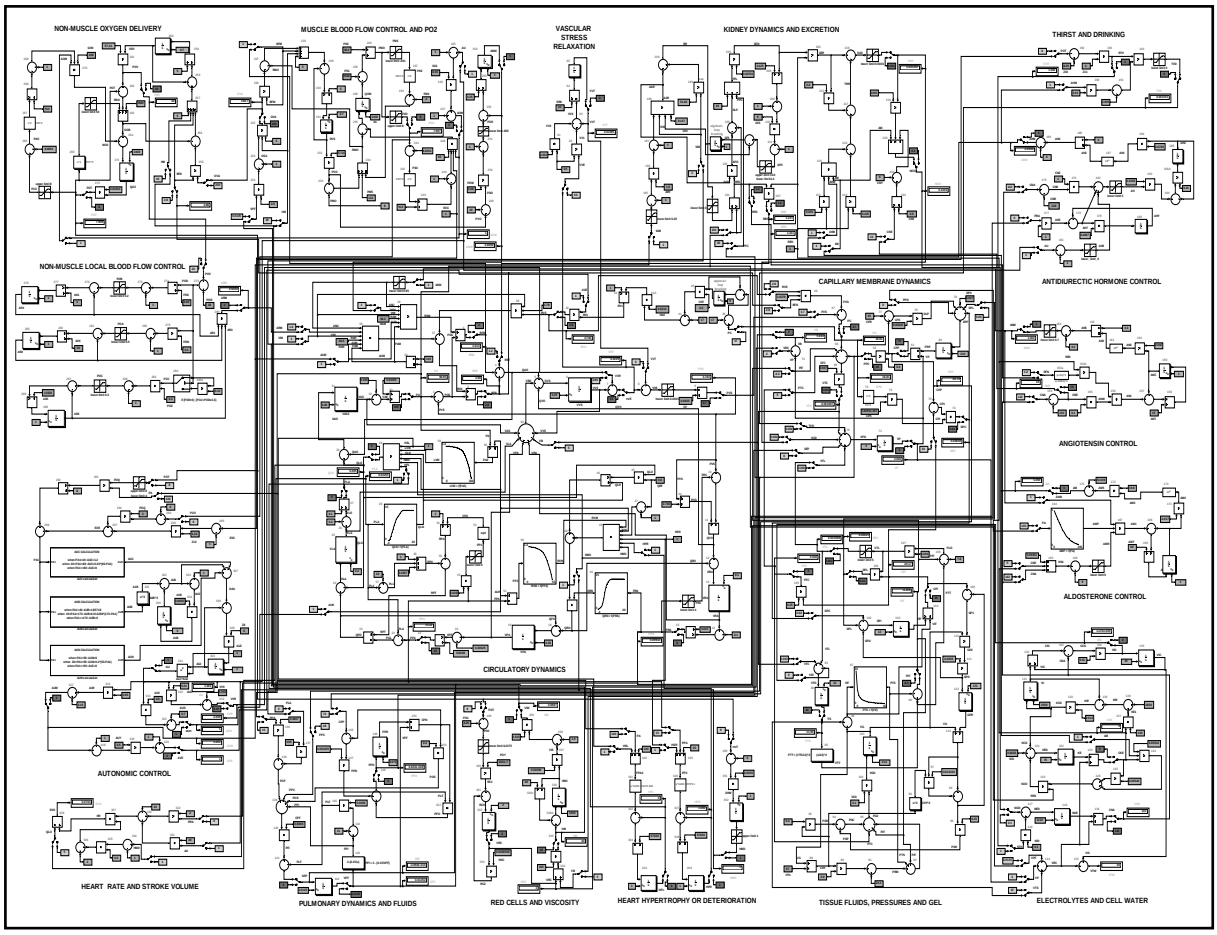


Figure 6: The implementation of Guyton's model in Simulink preserves the original arrangement of elements in Guyton's graphic diagram.

Today, there are specialized software simulation environments available for the development, debugging and verification of simulation models, which allow creating a model in graphical form and then testing its behaviour. One of these is the Matlab/Simulink development environment by Mathworks, which allows building a simulation model gradually from individual components – types of software simulation elements that are interconnected using a computer mouse to form simulation networks. Simulink blocks are very similar to the elements used by Guyton for the formalized representation of physiological relations. The only difference is in their graphical form (see Fig. 4).

This similarity inspired us to use Simulink to revive Guyton's good, classic diagram and transform it into a working simulation model. When implementing the model in Simulink, we used switches that allow us to connect and disconnect individual subsystems and control loops while the model is running. We strove to keep the appearance of the Simulink model identical to the original graphic diagram – the arrangement, wire location, variable names and block numbers are the same.

The simulation visualization of the old diagram was not without difficulties – there are errors in the original graphic diagram of the model! It does not matter in the hand-drawn illustration but if we try to bring it to life in Simulink, the model as a whole collapses immediately. There weren't too many errors – switched signs, a divider instead of a multiplier, mixed-up interconnections between blocks, a missing decimal point in a constant, etc. However, there were enough to prevent the model from working. Some of the errors could be seen at first sight (even with no knowledge of physiology) – it is obvious from the diagram that the value of some variables in some integrators would quickly grow to infinity in operation (because of incorrectly drawn feedback) and the model would collapse. With a knowledge of physiology and system analysis, however, all of the errors could be identified with some work (Fig. 5). A detailed description of the errors and their corrections is in Kofránek et al. 2007).

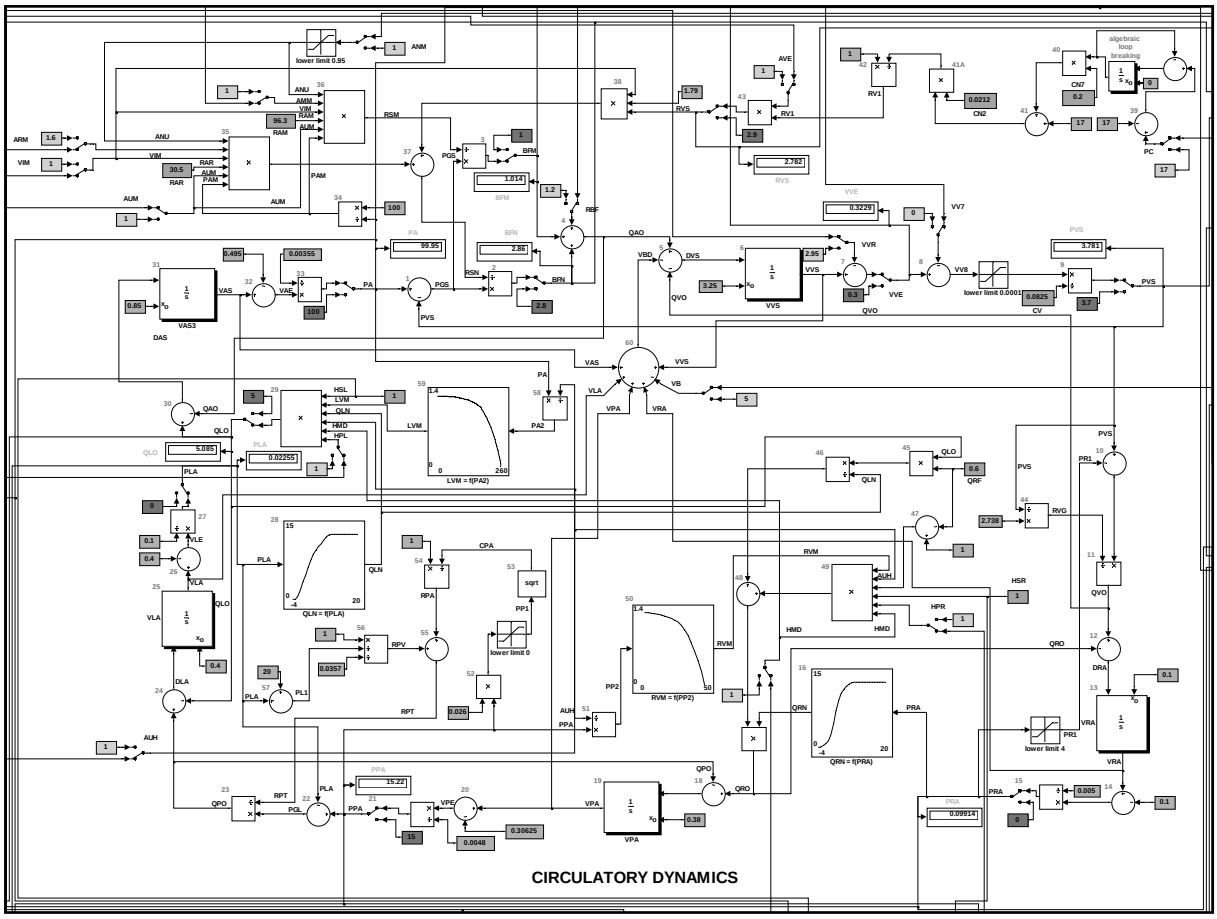


Figure7: More detailed central structures of the Simulink implementation of Guyton's model, representing flows through aggregated parts of the circulatory system and the activity of the heart as a pump.

It is interesting that Guyton's diagram as a complex drawing was reprinted many times in various publications (recently see e.g. Hall, 2004, Van Vliet and Montani, 2005). However, nobody mentioned the errors or made an effort to correct them. This was understandable at the time the diagram was created. Drawing software did not exist – the diagram was created as a complex drawing – and redrawing the complex diagram manually was not easy. It is also possible that the authors of the model did not wish to correct the errors – those who went to the trouble of analyzing the model could spot the drawing's "typos" and those who would just like to unthinkingly copy the diagram were out of luck. After all, the authors used to send out the source codes of Fortran programs for their model – so if anybody wished just to test the behaviour of the model, they did not have to program anything (at most, they had to routinely convert the program from Fortran to another programming language).

Our Simulink implementation of Guyton's (corrected) model (Figs. 6 and 7) is available for download at www.physiome.cz/guyton. Also available at that address is our Simulink implementation of a much more complex, later model from Guyton et al. There is also a very detailed description of all applied mathematical relations with an explanation.

Block-oriented simulation networks

Guyton's block diagram augured the rise of visual, block-oriented simulation languages. However, Guyton and his colleagues implemented the model in Fortran back in 1972 – Simulink version 1 was released eighteen years later (in 1990). Block-oriented simulation languages, of which Simulink is a typical example, allow assembling computer models from individual blocks with defined inputs and

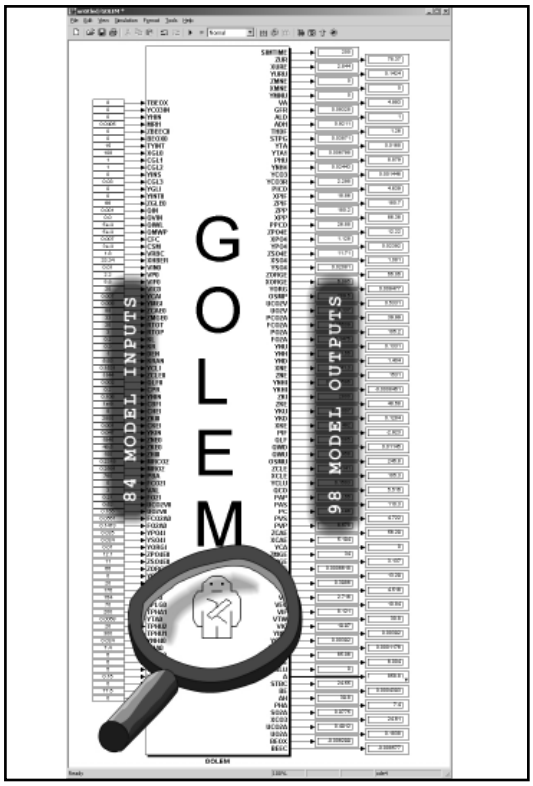


Figure 8: An example of a simulation chip (here representing the simulation model that is the basis for the GOLEM simulator (Kofránek et al., 2001). Its behaviour can be tested easily in the Simulink environment – input values (or value patterns) can be fed to the "input pins" and outputs or the time behaviour of outputs can be read out from the "output pins" by means of virtual displays or oscilloscopes. The next illustration shows the inside of this chip.

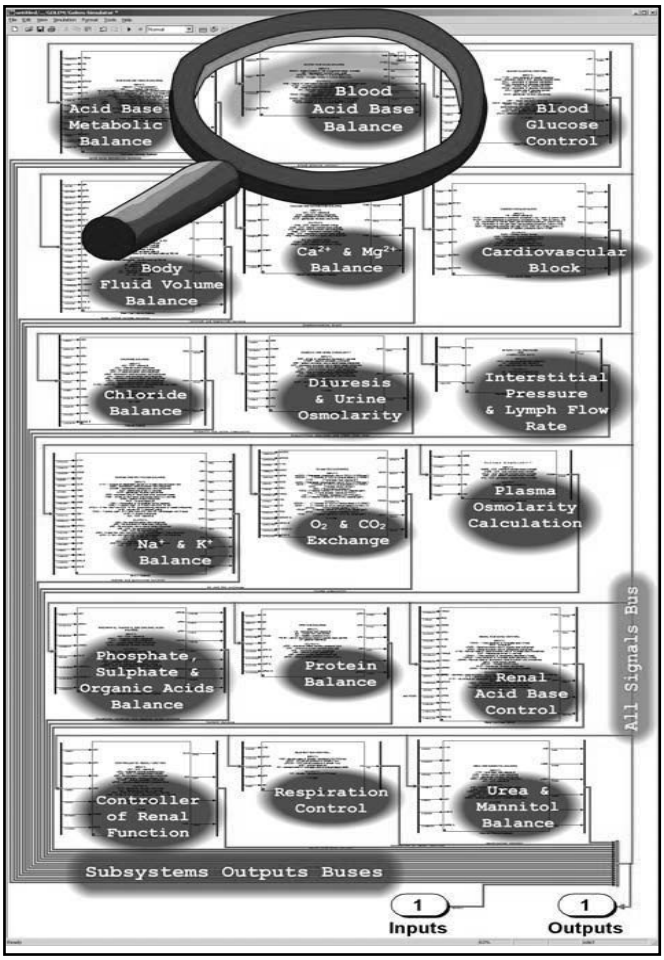


Figure 9: The "inside" of the simulation chip from the previous illustration. The structure resembles an electric network with interconnected integrated circuits, which represent simulation chips of a lower hierarchical level here. The next illustration shows the contents of the "Blood Acid Base Balance" chip.

outputs. The blocks are grouped in libraries; when building a model, a computer mouse is used to create individual block instances, with inputs and outputs connected through wires that "conduct" information.

A Simulink network can be arranged hierarchically. Blocks can be grouped into subsystems that communicate with their ambient environment through defined input and output "pins", making "simulation chips" of a sort. A simulation chip hides the simulation network structure from the user, much like an electronic chip hiding the interconnection of transistors and other electronic elements. Then the user can be concerned just with the behaviour of the chip and does not have to bother about the internal structure and calculation algorithm. The behaviour of a simulation chip can be tested by monitoring its outputs using virtual displays or virtual oscilloscopes connected to it. This is very useful especially for testing the behaviour of a model and expressing the mutual relations of variables.

The entire complex model can be then visualized as interconnected simulation chips and the structure of their interconnection clearly shows what effects are taken into account in the model, and how (Figs. 8–11).

This is very useful for interdisciplinary collaboration – especially in borderline fields such as biomedical system modelling (Kofránek et al., 2002). An experimental physiologist does not have to examine the details of mathematical relations hidden "inside" a simulation chip; however, from the

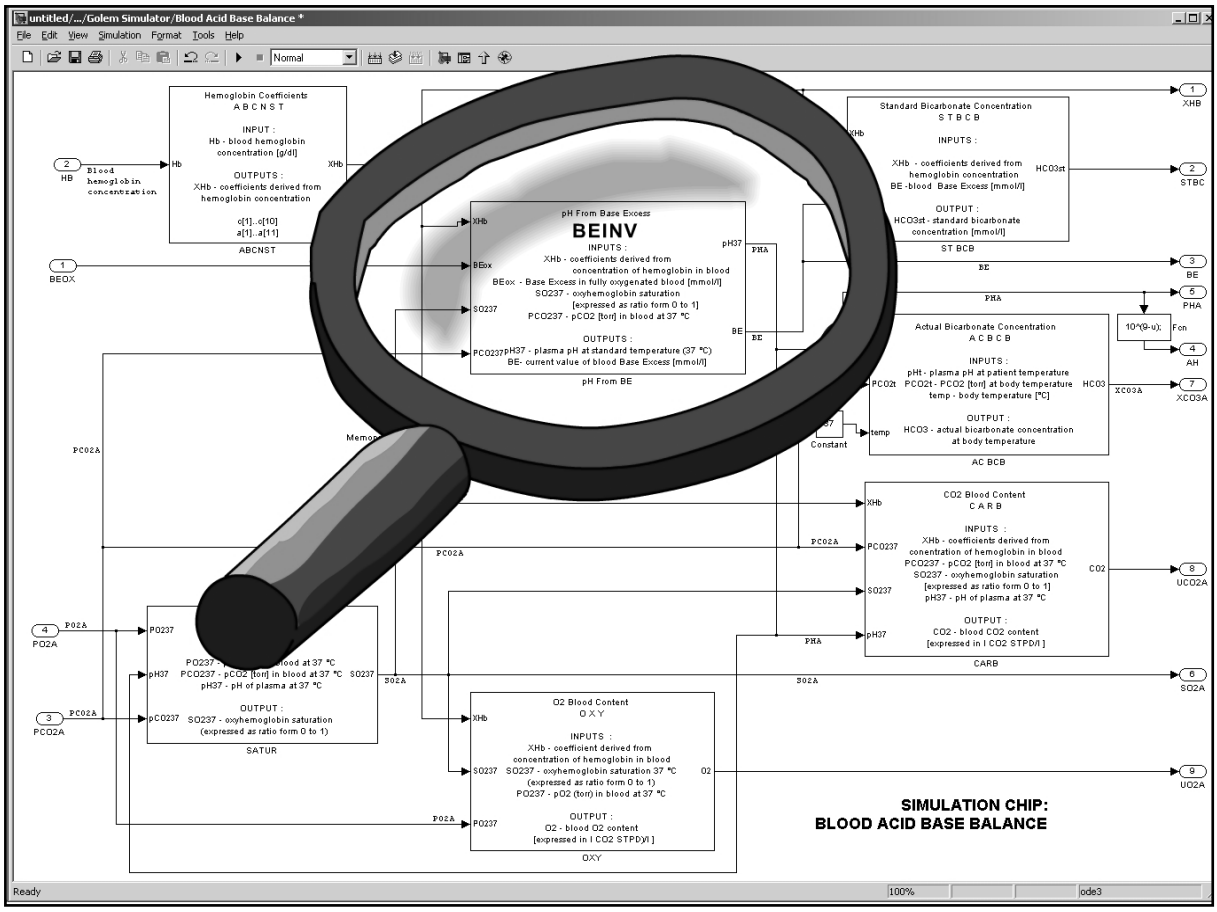


Figure 10: Simulation chips are arranged hierarchically. The illustration shows the "inside" of one of the simulation chips from Fig. 9. Since each simulation chip includes sufficiently detailed documentation of its inputs and outputs, the structure of relations inside a simulation chip (representing the physiological relations in a real organism) can be understood by physiologists. The next illustration shows the contents of the "BEINV" chip.

mutual interconnection of simulation chips they will understand the model structure and will be able to check its behaviour in the appropriate simulation visualization environment.

Simulation chips can be stored in libraries and users can create their instances for use in their models (Fig. 12). For example, we created a Physiolibrary for modelling physiological regulations (http://www.physiome.cz/simchips).

Hierarchical, block-oriented simulation tools are thus used advantageously in the description of the complex regulation systems that we have in physiology. A formalized description of physiological systems is the subject matter of PHYSIOME, an international project that is a successor to the GENOME project. The output of the GENOME project was a detailed description of the human genome; the goal of the PHYSIOME project is a formalized description of physiological functions. It uses computer models as its methodological tool (Bassingthwaighte, 2000; Hunter et al., 2002).

Several block-oriented simulation tools developed under the PHYSIOME project have been used as a reference database for a formalized description of the structure of complex physiological models. These include JSIM (http://www.physiome.org/model/doku.php) and CEllML (http://www.cellml.org/). Prof. Guyton's disciples and followers have expanded the original, extensive simulator of the circulatory system (Quantitative Circulatory Physiology (Abram et al. 2007) with an integrated connection of all important physiological systems. The latest result is the Quantitative Human Physiology simulator (Hester et al. 2008), now also distributed as "Digital Human", which represents today's most comprehensive

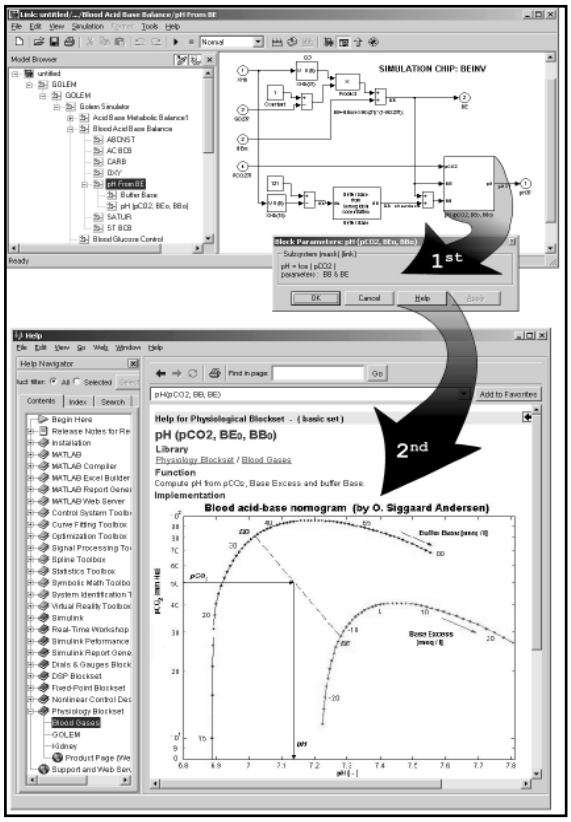


Figure 11: Structure of the BEINV simulation chip. Simulation chips at the lowest hierarchical level consist in interconnected elementary components of the Simulink development systems (adders, multipliers, integrators, etc.) and possibly simulation chips of further lower levels. These interconnected elements represent individual mathematical relations. Usually, each chip is dynamically linked with the appropriate documentation page including a factual description of the chip's function, including a description of the mathematical relations that constitute its basis.

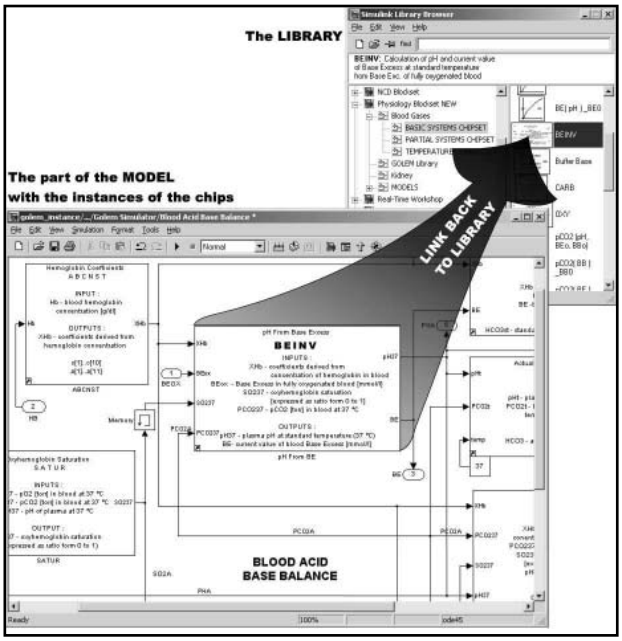


Figure 12: Simulation chips can be gathered in hierarchically arranged libraries in Simulink. It is then possible to use the mouse to "pick up" individual chips from the libraries (as from a tool palette), place them into the created application, interconnect them and create more complex models.

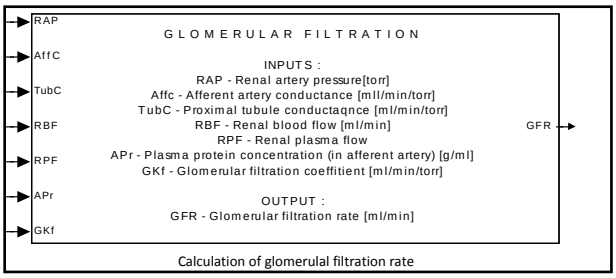


Figure 13 The "Glomerular Filtration" simulation chip calculates the glomerular filtration rate. The structure of calculation is hidden from the user. However, there must be no algebraic loops inside a chip.

and largest model of physiological functions. The model can be downloaded from http://physiology.umc.edu/themodelingworkshop/. The authors developed a special block-oriented simulation system to represent the complex model structure.

Causal and acausal approaches

Block-oriented tools work with hierarchically interconnected blocks. The connections between blocks "conduct" signals that transmit the values of individual variables from the output of a block to the inputs of other blocks. The blocks process input information into output information.

The hierarchically arranged block-oriented description clearly shows how the values of individual variables are calculated in the model – i.e. what the calculation algorithm is.

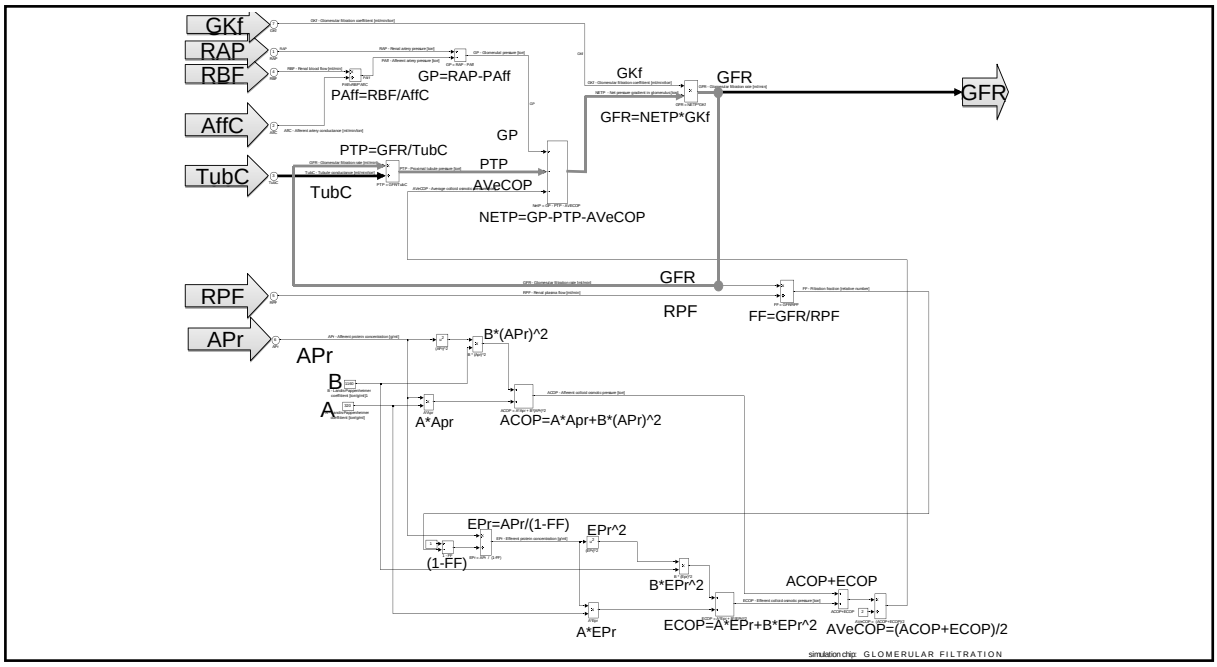


Figure 14: The interconnection of individual blocks inside the "Glomerular Filtration" simulation chip graphically represents individual mathematical relations for the calculation of the glomerular filtration rate. However, there is an algebraic loop. It is necessary to break the loop.

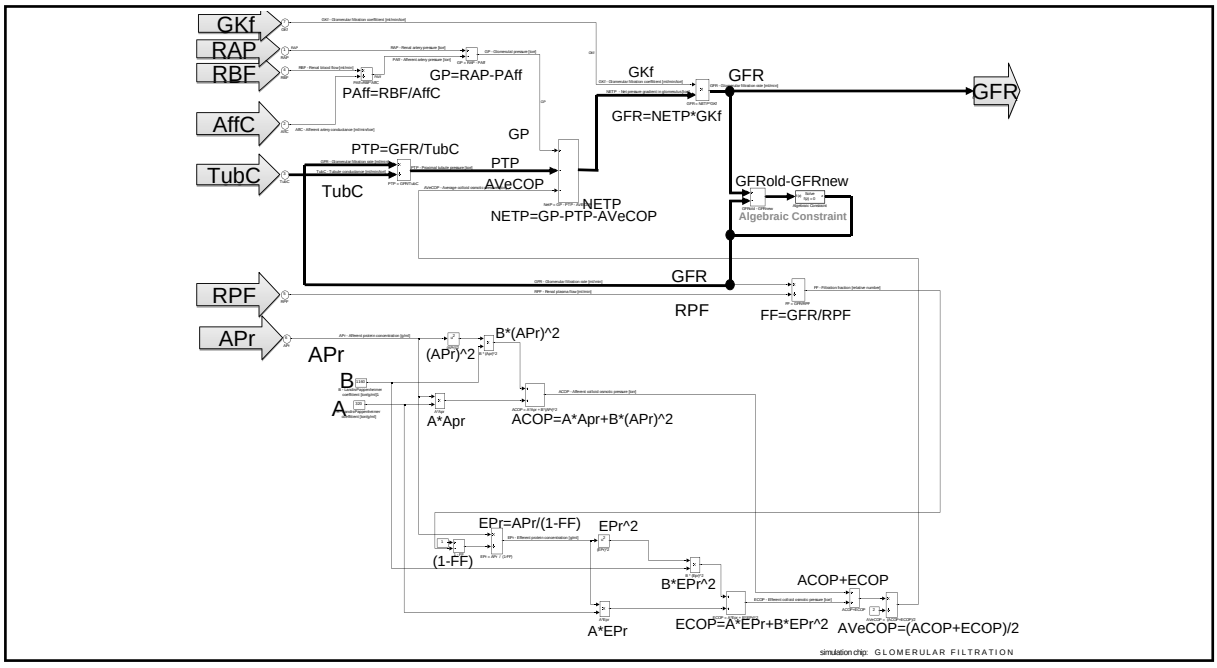


Figure 15: Breaking the algebraic loop in the calculation of the glomerular filtration rate. The interconnection of Simulink blocks reflects the calculation procedure rather than a graphical representation of mathematical relations.

However, the interconnection of blocks in a network of relations cannot be completely arbitrary. Interconnected elements **may not include any algebraic loops** – i.e. cyclic structures where an input value fed to the input of a calculation block depends (through several intermediate blocks) on the block's output value in the same time step.

For illustration, let us consider the small example of an algebraic loop in Simulink, a block-

oriented language.

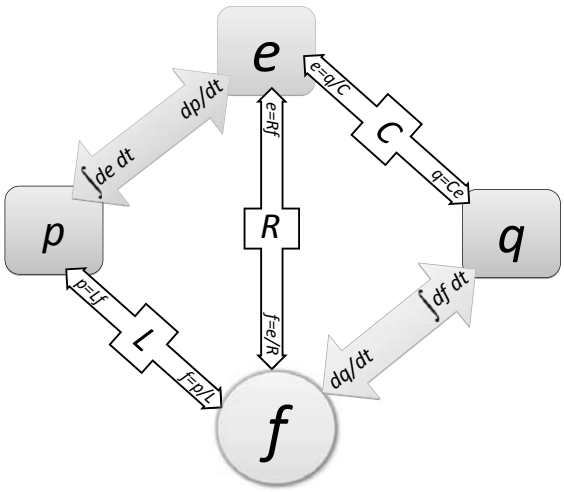
A model of the kidneys uses a simulation chip calculating the glomerular filtration rate. The individual inputs and outputs of that chip are shown in Fig. 13. The inside of the simulation chip consists in elementary blocks performing mathematical operations. The value of GFR, a variable representing the glomerular filtration rate, is calculated from the value of NETP; to calculate NETP it is necessary to know the value of PTP, which is however calculated as the quotient of GFR and TUBC (Fig. 14). Our Simulink diagram contains an algebraic loop that must be broken. Therefore we solve an implicit equation in the blocks identified as "Algebraic Constraint" in Fig. 15 to calculate GFR in each integration step.

Therefore, a Simulink network does not constitute the graphical representation of mathematical relations in a model; rather, it is the graphical representation of a chain of transformations from input values to output values through Simulink elements where loops are not allowed.

If we focus on the representation of a structure of mathematical relations rather than the algorithm of calculations when building a model in Simulink, we can easily introduce algebraic loops into the model (however, the compiler will warn us about this). There are methods that can be used to get rid of algebraic loops (e.g. see Dabney and Harman , 2004) – however, they lead to transformations that make the model structure even more complex and the model more difficult to understand. The need to have a fixed direction of connection from inputs to outputs with no algebraic loops also makes model building more difficult.

The interconnection of blocks in Simulink thus ***reflects the calculation procedure rather than the actual structure of the modelled reality.*** We call this ***causal modelling.***

In complex systems, the ***physical reality of the modelled system becomes somewhat lost under***



Generalized system properties:

- ***e*** means generalized effort – corresponding to force in mechanics, voltage in electrical diagrams, pressure in hydraulics, etc.
- ***f*** is generalized flow – corresponding to velocity in mechanics, current in electrical diagrams, flow rate in hydraulics, temperature flow in thermodynamics, etc.
- ***q*** is generalized accumulation or deflection, representing the integral of the generalized flow. It corresponds e.g. to the stretching of a spring in mechanics, fluid volume in hydraulics, charge in electrical diagrams, accumulated heat in thermodynamics, etc.
- ***p*** is generalized momentum (inertance) – the integral of the generalized effort, representing kinetic energy; in hydraulics it represents the change of the flow rate proportional to the pressure difference (flow momentum), in electrical circuits it is the potential needed to change an electric current (induction), etc.
- ***R***, ***C*** and ***L*** represent constants of proportionality between the generalized system properties. They correspond e.g. to resistance, capacitance or weight.

Figure 16: Relations between generalized system properties.

the structure of calculation with this approach.

New, ***"acausal" tools*** have recently been developed for the creation of simulation models. The major innovation brought about by acausal modelling tools is the possibility to describe the individual parts of a model directly ***as a system of equations rather than an algorithm for solving the equations***. The notation of models is declarative (we describe the structure and mathematical relations, not the calculation algorithm) – thus the notation is acausal.

Acausal modelling tools work with interconnected components that are instances of classes in which equations are directly defined.

The components (i.e. instances of classes with equations) can be interconnected by means of precisely defined interfaces – connectors; this defines a system of equations.

The latest version of Simulink provides certain options for using acausal tools as well. Mathworks, the producer of the Matlab/Simulink simulation tools, responded to the new trends by creating a special acausal Simulink library – Simscape – and related domain libraries such as SimElectronics, SimHydraulics, SimMechanics, etc.

A modern simulation language that is built directly on the acausal notation of models is ***Modelica*** (Fritzson, 2003). It was originally developed in Sweden and is now available both in an open-source version (developed under the auspices of Modelica Association, http://www.modelica.org/) and in two commercial implementations.

The first commercial implementation is made by Dynasim AB – which has been bought by Dassault Systemes, a multinational corporation (sold under the name of ***Dymola***, currently in version 7.1), and the other commercial implementation is made by MathCore (sold under the name of ***MathModelica***). Dynasim's Modelica has a good connection to the Matlab and Simulink simulation tools, while MathModelica can connect to the Mathematica environment made by WoModelica works with interconnected components that are instances of individual classes. Unlike the implementation of classes in other object-oriented languages (such as jw C# or Java), classes in Modelica have an additional special section in which ***equations*** are defined.

The equations do not mean assignment (i.e. storing the result of the calculation of an assigned command in a variable) but rather the definition of relations between variables (as is common in mathematics and physics). For example, the following notations of relations between variables expressing the resistance (R), flow (F) and pressure gradient (P) are equivalent:

$F=P/R$

$P/R=F$

$P=R*F$

$R*F=P$

$R=P/F$

$P/F=R$

Components (class instances) in Modelica can be interconnected by means of precisely defined interfaces – ***connectors***.

What is important is that the interconnection of components actually ***interconnects systems of equations in the individual components*** with one another. ***By interconnecting Modelica components, we do not define the calculation procedure but rather the modelled reality. The method of solving the equations is then "left to the machines".***

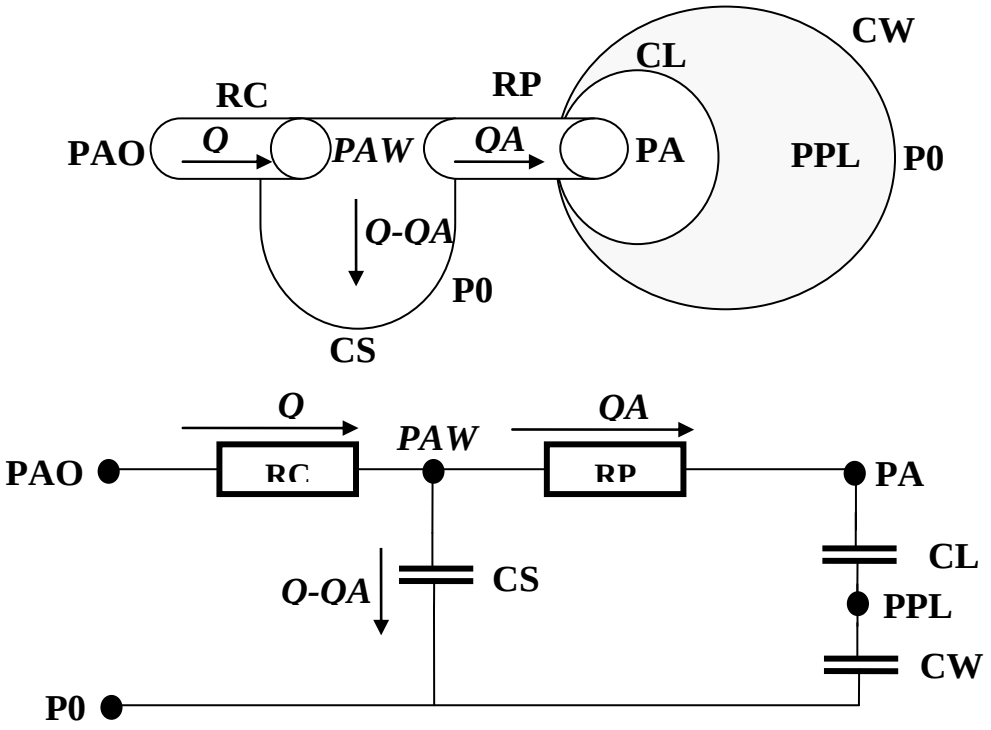


Figure 17: A simple pulmonary mechanics model (hydraulic and electrical analogy).

Generalized system properties

The representation of a model in an acausal simulation environment resembles the physical reality of the modelled world more than the standard interconnected block diagrams in causal modelling tools. This is associated with the generalized system properties of the real world (Fig. 16), where an important role is played by ***generalized effort*** (corresponding to force, pressure, voltage, etc. in the real world) and ***generalized flow*** (corresponding to current, flow rate, etc. in the real world). The integral of generalized flow is ***generalized accumulation or deflection*** (in the real world, this can be e.g. an electrical charge but also the volume of a liquid or gas, stretching of a spring, accumulated heat, etc.). The integral of generalized effort is ***generalized momentum*** (this represents flow momentum in hydraulics, induction in electrical circuits, etc.).

Also related to the generalized system properties is the fact that descriptions of models of biological or physiological processes often use electrical or hydraulic analogies for reasons of clarity.

Let us illustrate the utilization of generalized system properties and the difference between modelling in block-oriented simulation tools and in Modelica with a physiological reality modelling example – a model of simple pulmonary ventilation mechanics.

Let us consider a ***simple pulmonary mechanics model*** that is schematically shown in Fig. 17. With a high level of simplification, the lungs can be seen as three bags interconnected through two tubes. The lungs are connected to the fan of the artificial pulmonary ventilation equipment, which periodically drives air into the lungs with the pressure **PAO**. **P0** is the pressure of the ambient atmosphere. Airflow ***Q*** runs through the upper respiratory tract that has the resistance ***RC***. From the upper respiratory tract, air forces its way through the lower respiratory tract to the alveoli. The resistance of the lower respiratory tract is ***RP***, the pressure in the central parts of the respiratory tract (at the boundary between the upper and lower respiratory tracts) is ***PAW***, the pressure in the alveoli is ***PA***.

Air expands the pulmonary alveoli, whose compliance is ***CL*** (as the total compliance of the lungs). The interpleural cavity is in between the lungs and the rib cage. The pressure in it is ***PPL***. The

chest has to expand as well during artificial pulmonary ventilation when air is forced into the lungs – the chest compliance is **CW**. The small portion of air that does not reach the alveoli expands the respiratory tract instead – its compliance is ***CS***.

Now we can set up our equations. According to Ohm's law, it must be true that:

$$\begin{aligned} PAW - PA &= RP\,QA \\ PAO - PAW &= RC\,Q \end{aligned} \tag{1}$$

The relation between the compliance, pressure gradient and volume (expressed as the integral of the flow rate) is expressed by these equations:

$$\begin{aligned} PA - PPL &= \frac{1}{CL}\int QA\,dt \\ PPL - P0 &= \frac{1}{CW}\int QA\,dt \\ PAW - P0 &= \frac{1}{CS}\int (Q - QA)\,dt \end{aligned} \tag{2}$$

According to the generalized Kirchhoff's law, the sum of all pressures (voltages) along a closed loop must be equal to zero, i.e. the following must hold true for the loop along the node PAW and along the node PA0:

$$\begin{aligned} &(PAW - PA) + (PA - PPL) + (PPL - P0) + (P0 - PAW) = 0 \\ &(PAO - PAW) + (PAW - P0) + (P0 - PAO) = 0 \end{aligned} \tag{3}$$

Substituting from the equations for Ohm's law and compliances, we get:

$$\begin{cases} RP\,QA + \left(\frac{1}{CL} + \frac{1}{CW}\right)\int QA\,dt - \frac{1}{CS}\int (Q - QA)\,dt = 0 \\ Q\,RC + \frac{1}{CS}\int (Q - QA)\,dt + (P0 - PAO) = 0 \end{cases}$$

(4)

Causal approach – implementation of the pulmonary ventilation mechanics model in Simulink

When building a model in Simulink, we have to define precisely the ***procedure of calculation*** from input variables to output variables. If we wish to calculate the reaction of the air flow to/from the lungs (***Q***) to the input – i.e. to the changes in pressure at the beginning of the respiratory tract (***PAO***) caused by the artificial pulmonary ventilation apparatus – the Simulink model will look like Fig. 18.

We can also simplify the Simulink model. First we obtain a differential equation (input variable ***PAO***, output ***Q***) from equationc (4):

$$\frac{d^2PAO}{dt^2} + \frac{1}{RP\,CT}\frac{dPAO}{dt} = RC\frac{d^2Q}{dt^2} + \left(\frac{1}{CS} + \frac{RC}{RP\,CT}\right)\frac{dQ}{dt} + \frac{1}{RP\,CS}\left(\frac{1}{CL} + \frac{1}{CW}\right)Q \tag{5}$$

When we enter the numeric parameters of resistance (in units cm H_2O/L/sec) and compliance (in units L/cmH_2O) (Khoo, 2000):

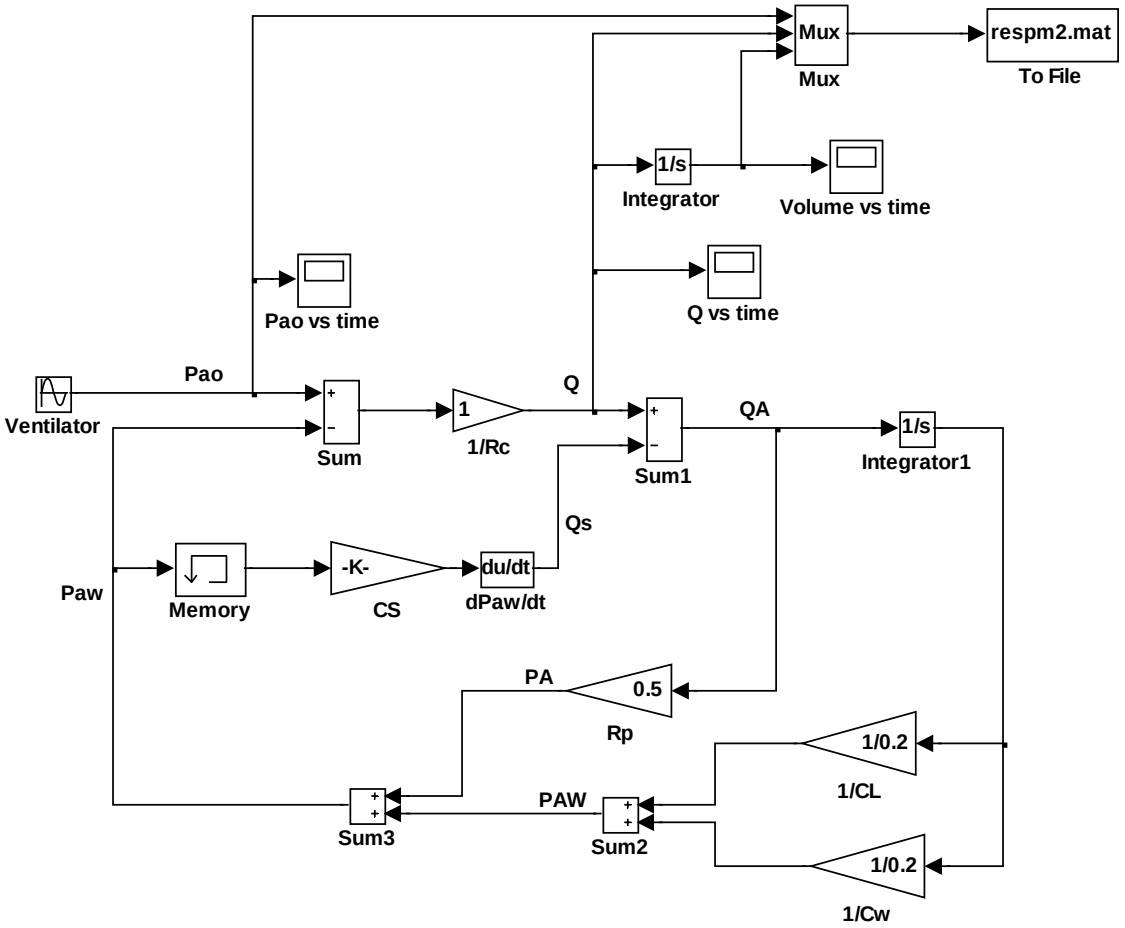


Figure 18: Simulink model implementation according to the equations (4).

$$RC = 1; \quad RP = 0,5; \quad CL = 0,2; \quad CW = 0,2; \quad CS = 0,005$$

(6)

the equation (5) simplifies:

$$\frac{d^2PAO}{dt^2} + 420\frac{dPAO}{dt} = \frac{d^2Q}{dt^2} + 620\frac{dQ}{dt} + 4000\,Q \tag{7}$$

In the Laplace transform of the equation (7), we get:

$$\frac{Q(s)}{PAO(s)} = \frac{s^2 + 420s}{s^2 + 620s + 4000} \tag{8}$$

This allows simplifying the Simulink model (Fig. 19):

However, when the values of the parameters change, the transform function (6) must be recalculated and the Simulink model will change.

Now we will make the model a little more complex by taking ***air inertia*** in the upper respiratory tract into account (Fig. 20).

In addition, we will now take into account the inertial element **LC**=0.01 cm H_2O s^2 L^{-1}:

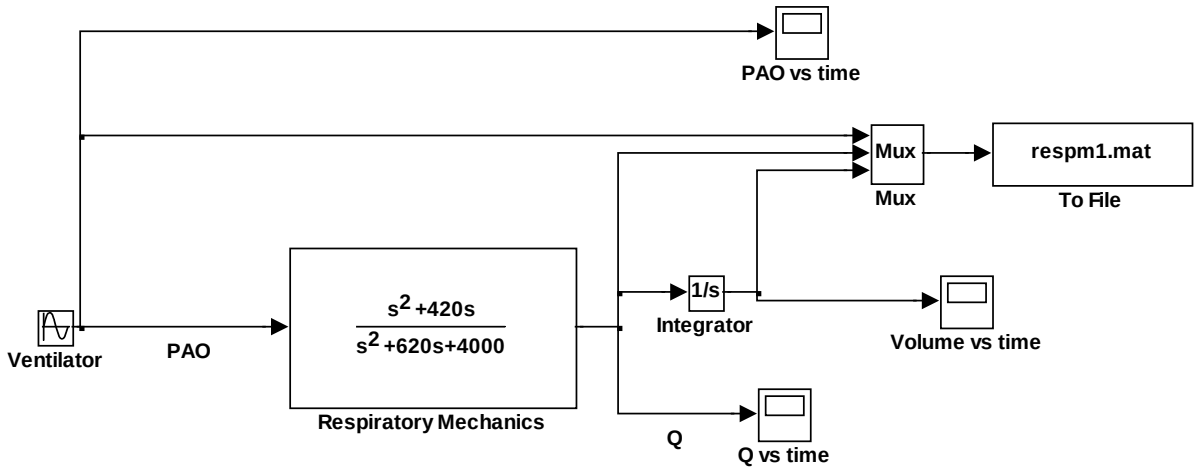


Figure 19 Simulink model implementation using the Laplace transform according to the equation (7).

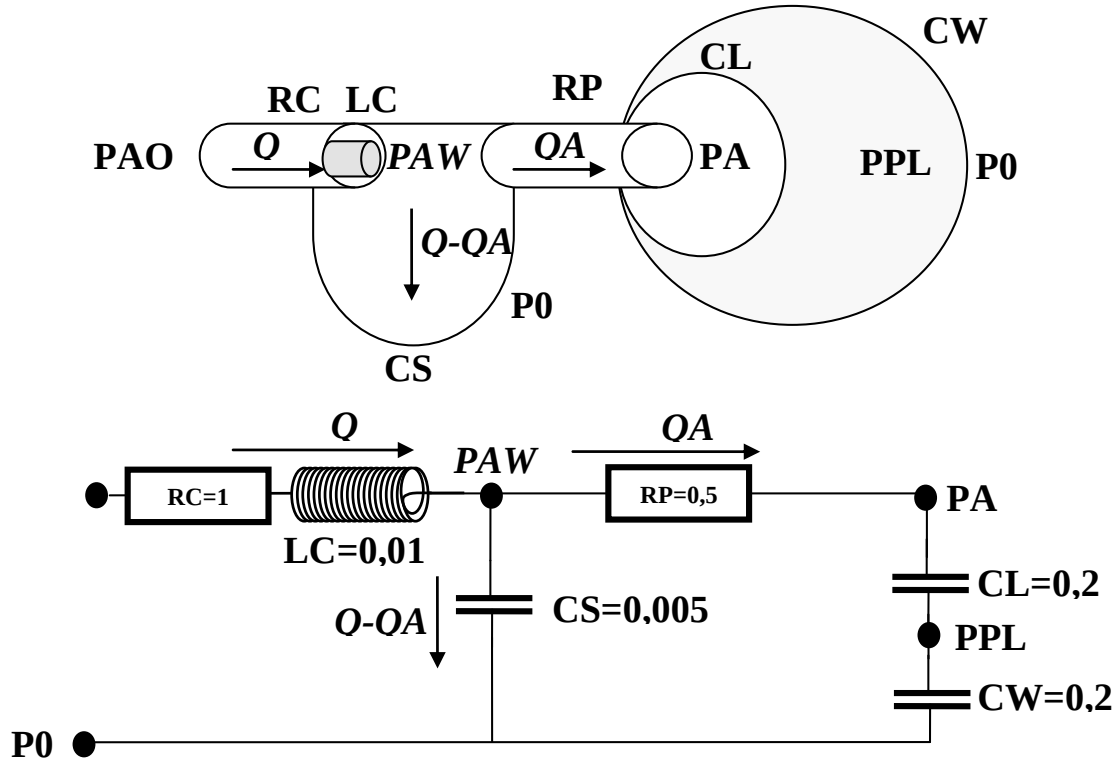


Figure 20: A simple pulmonary mechanics model taking inertia into account (hydraulic and electrical analogy).

$$LC = \frac{\Delta P}{\frac{dQ}{dt}} \tag{9}$$

where ***ΔP*** is the pressure gradient and **dQ/dt** is the flow acceleration, or:

$$\Delta P = LC \frac{dQ}{dt} \tag{10}$$

Then we get this instead of the system of equations (4):

$$\begin{cases} RP\dfrac{dQA}{dt}+\left(\dfrac{1}{CL}+\dfrac{1}{CW}\right)QA-\dfrac{1}{CS}(Q-QA)=0 \\ RC\dfrac{dQ}{dt}+LC\dfrac{d^2Q}{dt^2}+\dfrac{1}{CS}(Q-QA)+\dfrac{dP0}{dt}-\dfrac{dPAO}{dt}=0 \end{cases} \tag{11}$$

Instead of the equation (7), we get:

$$\frac{d^2PAO}{dt^2}+420\frac{dPAO}{dt}=0{,}01\frac{d^3Q}{dt^3}+5{,}2\frac{d^2Q}{dt^2}+620\frac{dQ}{dt}+4000Q \tag{12}$$

and in the Laplace transform, we get:

$$\frac{Q(s)}{PAO(s)}=\frac{s^2+420s}{0{,}01s^3+5{,}2s^2+620s+4000} \tag{10}$$

This Simulink model will change (Fig. 21):

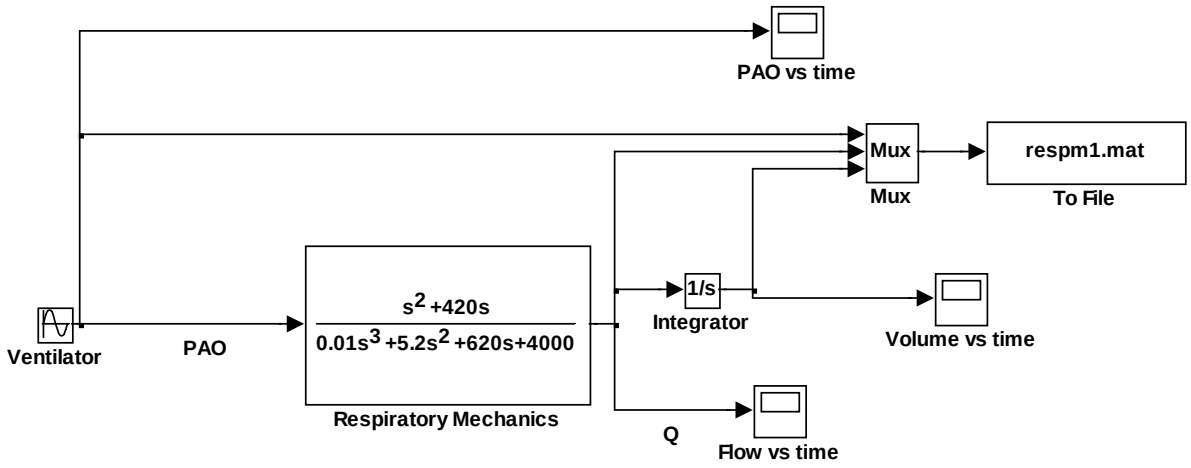


Figure 21: Simulink model implementation using the Laplace transform according to the equation (10).

Since we always have to take into account the direction of the calculation in Simulink, the actual Simulink diagram is rather dissimilar to the physical reality of the described system. Even a small change in the model, such as the inclusion of the inertial element, requires careful calculation and a change in the model structure. The model will change significantly even if we consider spontaneous breathing instead of artificial pulmonary ventilation. The model input will be not the pressure ***PAO*** generated by the artificial pulmonary ventilation respirator but e.g. the compliance of the thoracic wall ***CW*** (a cyclic variation in the compliance can be used to model the function of the respiratory muscles).

Acausal approach – implementation of the pulmonary ventilation mechanics model in Modelica

Comparing the model structure in Figs. 17 and 20, formulated by means of generalized state variables, to the implementation of the model in Simulink (Figs. 18, 19, 21), we can see that the interconnected blocks in Simulink express the structure of the calculation procedure rather than the

structure of the modelled reality.

In Modelica, this is different.

Acausal modelling tools, of which Modelica is a typical example, work with interconnected components that are instances of special classes in which ***equations*** are defined. When modelling in Modelica, the first task is to formally express the modelled reality by means of equations.

In our simple pulmonary mechanics model, we describe the resistances of the respiratory tract, expansible elastic bags, and we might take into account air flow inertia (see Figs. 17 and 20). The description of the air flow in the lungs belongs in the pneumatic domain. However, if we disregard the compressibility of gases, we can describe the model using the hydraulic domain. The same formal expression can be provided by an electrical analogy.

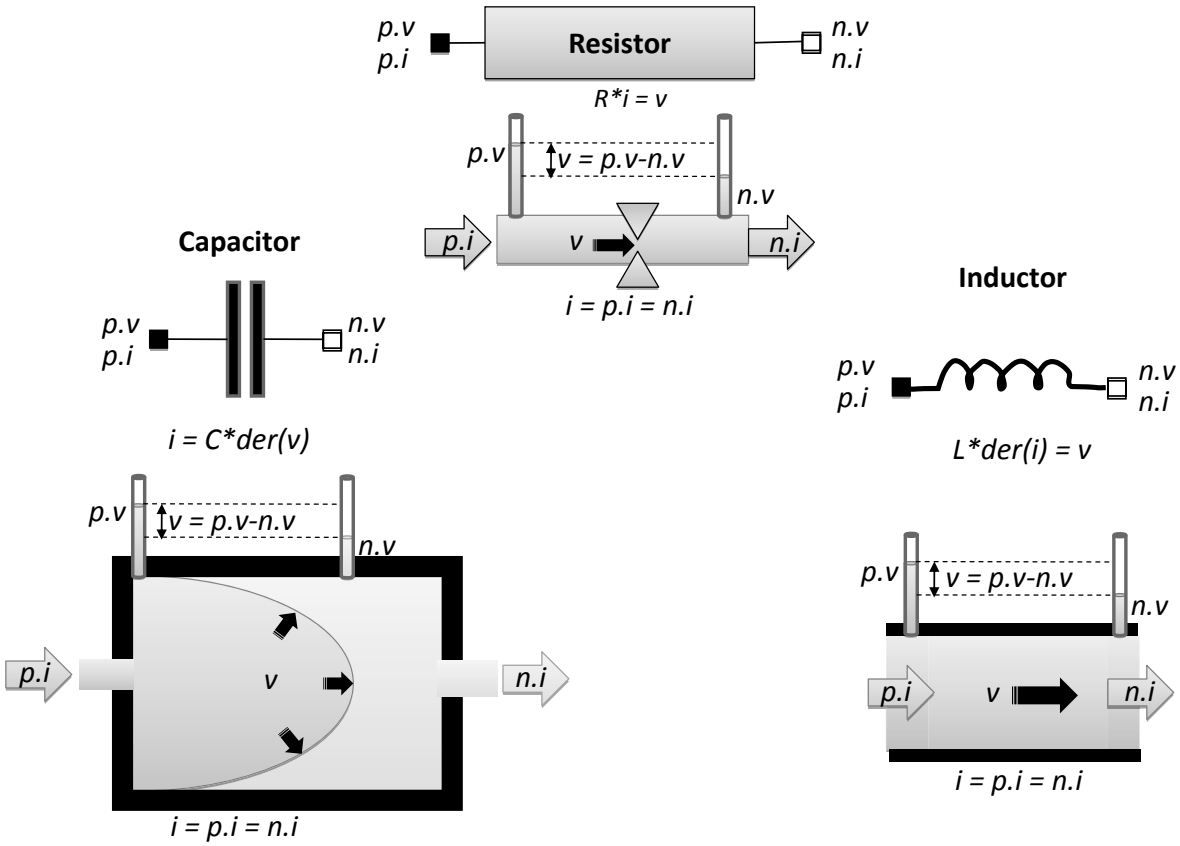


Figure 22: The hydraulic and electric elements are from different domains but have the same formal description. The analogy of voltage (v) in the hydraulic domain is pressure, the analogy of current (i) in the hydraulic domain is a stream of fluid (and a stream of gas in the pneumatic domain). Hydraulic resistance (R) follows Ohm's law in the same way as electric resistance (the voltage difference is just replaced with the pressure gradient and the current is replaced with the flow rate). The hydraulic analogy of a capacitor is an elastic bag expanded by the difference in pressures inside and outside the bag. The analogy of the electric capacity of a capacitor (C) is the compliance of the elastic bag wall. When we include inertia in a hydraulic system, the force that accelerates fluid flow is the pressure gradient. According to Newton's law, the acceleration of flow, i.e. the first derivative of flow der(i), is proportional to the pressure gradient (v) and inversely proportional to the weight of the selected fluid column, called inertance (L). In the electrical domain, inertance corresponds to coil inductance. Each element from the hydraulic or electric domain has two interconnecting connectors through which electric current or medium flow (p.i, n.i) flows in and out; as a rule, the running flow (i) never disappears in the element (i.e. i=p.i=n.i). Simultaneously, voltage or pressure (p.v, n.v) is connected to the connectors by interconnecting into a network, and a voltage gradient or a pressure gradient (v) builds up in the element.

It is interesting that the individual fundamental elements have the same formal expression (Fig. 22) in different domains (electrical, hydraulic or pneumatic). This is due to the general system properties of the real world, where voltage or pressure correspond to generalized effort and electric current or medium flow correspond to generalized flow, as the case may be.

To build the pulmonary mechanics model in Modelica, we will need to define the equations of three elementary classes, whose instances we will use in the model. To express the resistance of the respiratory tract, we will use an instance of the Resistor class. The elastic respiratory tract, alveoli and chest will be described as elastic bags using an instance of the Capacitor class and the air flow inertia will be expressed using an instance of the Inductor class.

The fragment with an equation notation in the "Resistor" class, describing the relation between variables expressing the resistance (R), pressure gradient (v) and flow (i) in Modelica, is simple, according to Ohm's law:

```
equation
  R*i = v;
end Resistor;
```

The "Capacitor" class is used to describe an elastic bag expanded by air flow at the input. The compliance (C) characterizes the level of "expansibility" of the bag wall due to the pressure difference (v) between the air pressure forcing air into the bag and the pressure outside the elastic bag. The flow rate of air coming to the bag (i) is then described by the following equation in the Modelica language (where *"der"* means derivative):

```
equation
  i = C*der(v);
end Capacitor;
```

The inertial element will be implemented in the model by means of the "Inductor" class. The force that accelerates air flow is the pressure gradient. According to Newton's law, the acceleration of flow, i.e. the first derivative of flow der(i), is proportional to the pressure gradient (v) and inversely proportional to the weight of the selected gas column, called inertance (L). We can thus describe the relation between a change in the flow rate (i) and the pressure gradient (v) depending on inertance (L) using a simple equation in the "Inductor" class:

```
equation
  L*der(i) = v;
end Inductor;
```

Instances of the above-mentioned fundamental elements are interconnected in a network by means of connectors – two interconnecting connectors, labelled "p" and "n", are defined for each of the elements. Voltage, or pressure for the hydraulic or pneumatic domain, is fed to each of them (p.v, n. v) when connected and an electric current or medium flow (p.i, n.i) can flow through the connectors.

Connectors are instances of special connector classes, in which the variables used for interconnection are defined. Components can be interconnected by means of connectors that are instances of the same connector classes (the "interconnection sockets" must be of the same type). In our case, connectors "p" and "n" are instances of the "Pin" connector class, which is able to interconnect voltages or pressures (p.v, n.v) and flows (p.i, n.i) with the environment. Values from the connectors are interconnected with the values of the variables (i) and (v) inside the individual fundamental elements. As a rule, flow does not disappear anywhere in any of the above-mentioned fundamental elements – what flows into an element also flows from it (i=p.i=n.i), and the appropriate gradient is calculated from the difference in voltages or pressures (v=p.v-n.v).

Implementing this requirement is simple – since Modelica is an object-oriented language, all three of the above-mentioned classes of fundamental elements will have a common ancestor (OnePort)

from which they will inherit connectors “p” and “n” as well as the following equations:

```
equation
  v= p.v-n.v;
  0= p.i-n.i;
  i= p.i;
end OnePort;
```

The equations will thus connect the values of the pressures or voltages fed from the environment to connectors “p” and “n” (p.v, n.v) with the pressure or voltage gradient (v) and express the same value of (electric or hydraulic) flow at both connectors (p.i, n.i) and inside the component (i).

Connector classes define the manner in which Modelica components communicate with one another. Figuratively speaking, by defining connector classes we define the types of “sockets”. In connectors we define individual variables that the connector will use to interconnect a component with its environment.

It is defined for each variable in a connector whether it represents a flow (then the variable is identified with a “flow” attribute) or not (“non-flow” variables). This differentiation is important for the correct interpretation of the interconnection of individual components (instances of element classes) through the appropriate connectors (see Fig. 23). For flow variables, it is obvious that we must make sure the entity in question (whose flow the variable characterizes) neither disappears nor accumulates anywhere in the interconnection. Therefore, the sum of all interconnected variables with the “flow” attribute must be zero (as according to Kirchhoff’s law in the electrical domain). For non-flow variables, an interconnection defines that their values must be the same for all interconnected connectors (according to Kirchhoff’s first law). By interconnecting the instances of individual fundamental elements through connectors, we express the requirement of the zero algebraic sum of the values of interconnected flow variables and the requirement of the equality of the values of interconnected non-flow variables.

Each Modelica class can have a graphical representation – this is important especially for depicting the interconnection of instances where components are interconnected to create a clear graphical structure of a model. That is why we can also define an icon for each class in Modelica. The icon can be animated.

We can then create a model graphically in Modelica, by interconnecting the instances of individual elements that we select from a library with the mouse and setting the values of the appropriate parameters in a dialogue box.

For the implementation of our pulmonary ventilation model, we need to interconnect instances of the “Resistor”, “Capacitor” and “Inductor” elements.

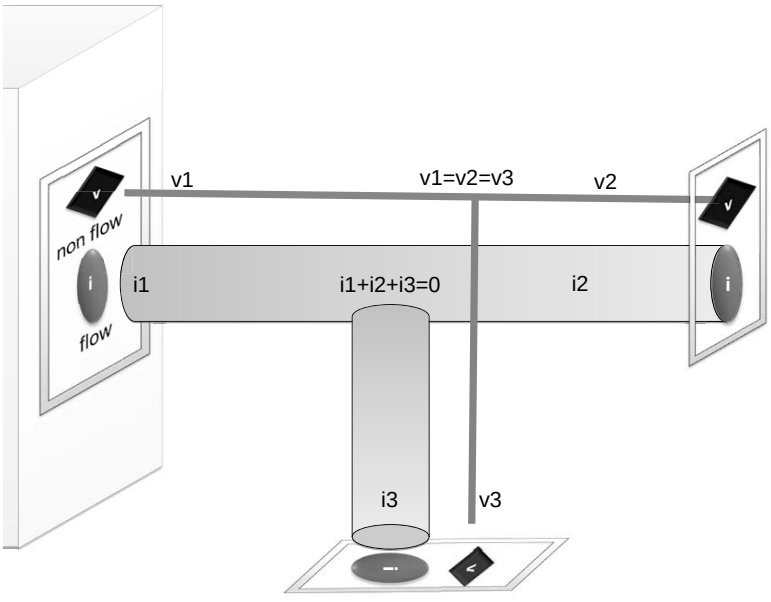


Figure 23: Interconnection of Modelica components by means of acausal connectors. The values of flow-type connector variables (here the variable i) will be set so that the algebraic sum of the values of all interconnected flows is zero. The values of other (non-flow) variables (here the value of v) will be set to the same value at all interconnected connectors.

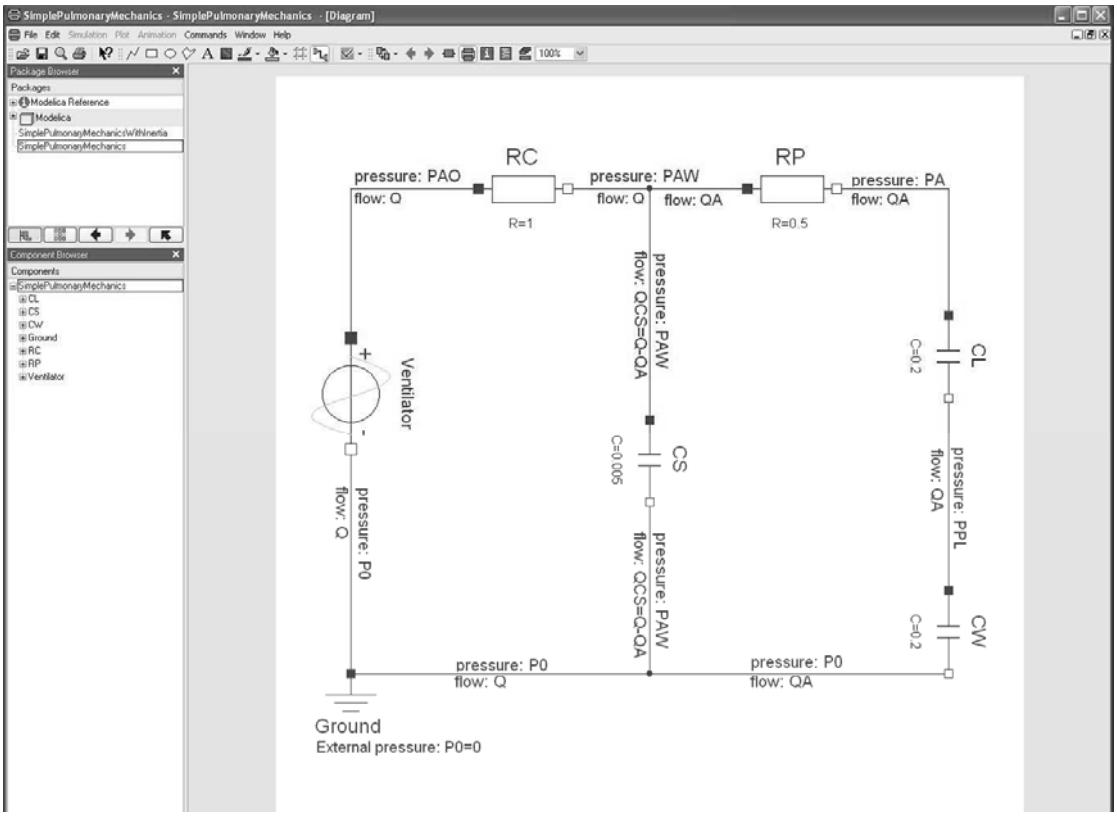


Figure 24: The implementation of the pulmonary mechanics model (according to Fig. 17) in Modelica resembles the modelled reality much more than the implementation in Simulink.

However, we do not have to program the fundamental elements we need from the very beginning – Modelica includes extensive libraries from various physical domains (electric, hydraulic, mechanic, etc.) where such elements can be found.

In our specific case, we can take advantage of e.g. the visual components of electrical circuits for a quick solution – we will create the individual instances (RC, RP, CL, CW and CS), enter the appropriate values of parameters (C and R) and interconnect the components with a connector.

The result is shown in Fig. 24. Comparing the model structure implemented in Modelica with the original schematic drawing showing the model structure (Fig. 17), we can see that the Modelica solution is straightforward and (unlike the Simulink implementation – see Figs. 18 and 19) the structure of the model corresponds to the structure of the modelled reality.

Increasing the complexity of the model by including an inertial element does not cause any significant trouble – we just pick up the appropriate inertial component (LC) from the library with the mouse, set the value of its parameter (L) and interconnect it in the model. The structure of the model implemented in Modelica, shown in Fig. 25, corresponds to the structure of the modelled reality (see Fig. 20), while the structure of the Simulink implementation (Fig. 19) corresponds more to the method of solution for the model's equations.

The fundamental elements of the simulated reality can have very trivial notation of relations between the variables in question. A resistor, capacitor or coil from the electrical physical domain or their hydraulic analogies are illustrative examples of this.

A complex system for calculation will ensue from interconnecting the fundamental elements in networks – a system of equations will result from their mutual interconnections. Their numerical solution in causal simulation tools may not be trivial at all – e.g. more complex R-C-L models of circulation or respiration implemented in Simulink are very complex (see e.g. circulation models in our Simulink

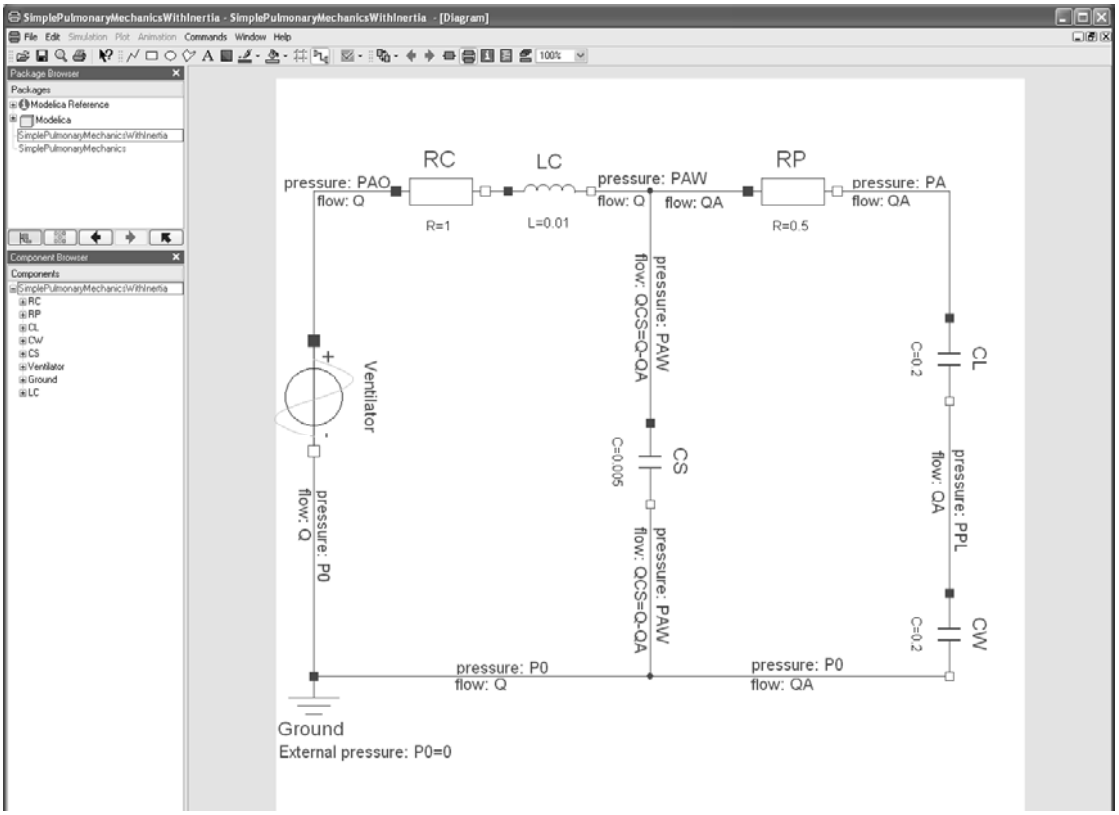


Figure 25: In Modelica, implementing the pulmonary mechanics model taking into account the inertial element (according to Fig. 20) merely requires adding the LC inertial component.

library, Physiolibrary – http://www.physiome.cz/simchips).

In Modelica, we do not have to bother with the method of solution for equations. Instead, more attention should be paid to the definition of equations in individual elements and interconnection of their instances (individual components).

In Modelica, the acausal tool itself will take care of the algorithm for solving the resulting system of equations and we can monitor the appropriate flows and pressures in various places in the simulated circuit when the simulation is launched.

Causal and acausal connectors

The acausal connector interconnection of components is implemented by means of two types of variables: one representing a flow – for this, it holds true that the sum of flow values in all connected nodes is zero (because no medium accumulates in the area of branching into connected nodes); and one whose value remains the same in all connected nodes. It is advisable that each variable with the flow attribute is accompanied by a non-flow variable representing the generalized effort in relation to the flow variable in the connector interconnection.

Unlike Simulink components (which have defined component inputs and component outputs), we do not define what is an input and what is an output in an acausal interconnection. An acausal Modelica component does not calculate output values from input values. The interconnection of Modelica components by means of acausal connectors interconnects the equations in individual components into systems of equations.

In addition to acausal linking connectors, Modelica classes may include causal input connectors that are used to feed actual input variables from the environment, as well as causal output connectors that serve to send output variables to the environment.

In addition to equations, Modelica classes may also include a precisely defined algorithm for the calculation of output values from input values (a typical example is the modelling of functional dependencies).

Modelica components are thus interconnected using both acausal links and causal, directional inputs and outputs.

Causal connectors usually distribute signals – e.g. in a blood circulation model, signal causal inputs may contain signals used to set resistance values in components representing the resistance of the circulatory system.

Consequently, a Modelica model is usually represented by a graphical set of components interconnected using both acausal and causal links. Components are instances of Modelica classes whose structure may also be represented as a network of interconnected instances.

An example of the definition and use of an acausal element – elastic compartment

Let us see a simple example of the definition and use of a Modelica class. When modelling the dynamics of blood vessels, we often need an elastic (inflatable) compartment.

Therefore we will define a class named *VascularElasticBloodCompartment* whose instances will be elastic, acausally interconnectable compartments that can be connected to the "distribution" of a fluid through an acausal connector – the fluid may flow to/from the compartment at a certain rate and under a certain pressure. We can assign a graphic icon to each class representing a model or connector in the programming environment. We can create an icon for our elastic compartment, too (Fig. 26).

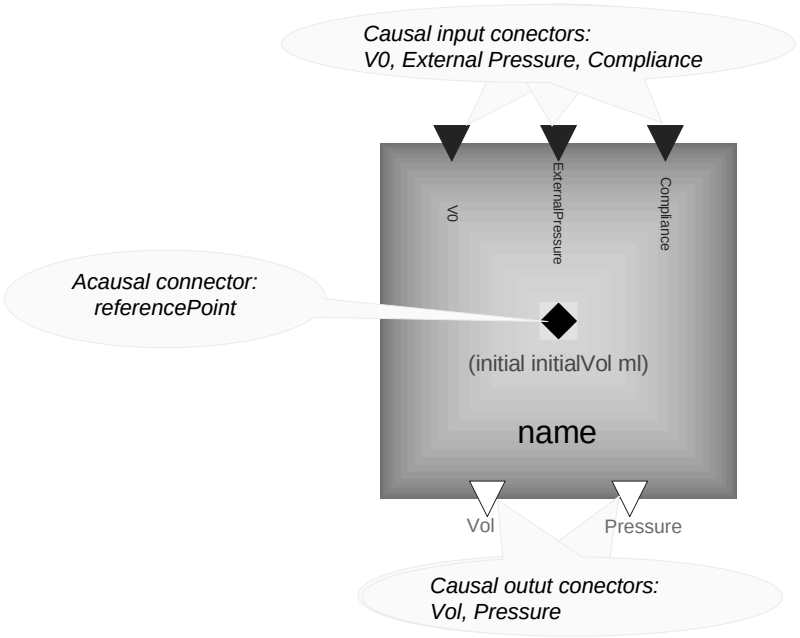


Figure 26: Modelica allows creating an icon for each created class representing a model or a connector, which will be used to interconnect instances of the class with other instances using graphic tools. The result is a model structure consisting in interconnected instances, very close to the modelled reality. Here we have created an icon for the elastic compartment, having one acausal connector (black diamond), three connectors for signal inputs and two connectors for signal outputs. Each instance of the elastic compartment will have this icon, displaying the actual value of the initial volume (specified as a parameter) instead of "*initialVol*" and the name of the instance instead of "*name*".

This is not just a school example – we take this compartment into account in our Modelica implementation of an extensive model of physiological functions, "Quantitative Human Physiology" (Abram 2007, Coleman et al, 2008). Fig. 28 shows an example of the use of instances of the elastic compartment in our implementation of this extensive model.

We can imagine the elastic vascular compartment (Fig. 27) as an inflatable bag with one ***acausal interconnecting connector*** (let us name it e.g. *"ReferencePoint"*) that we will use to connect to the environment – this connector will provide us with two variables:

- flow *"ReferencePoint.q",*
- pressure *"ReferencePoint. pressure"*.

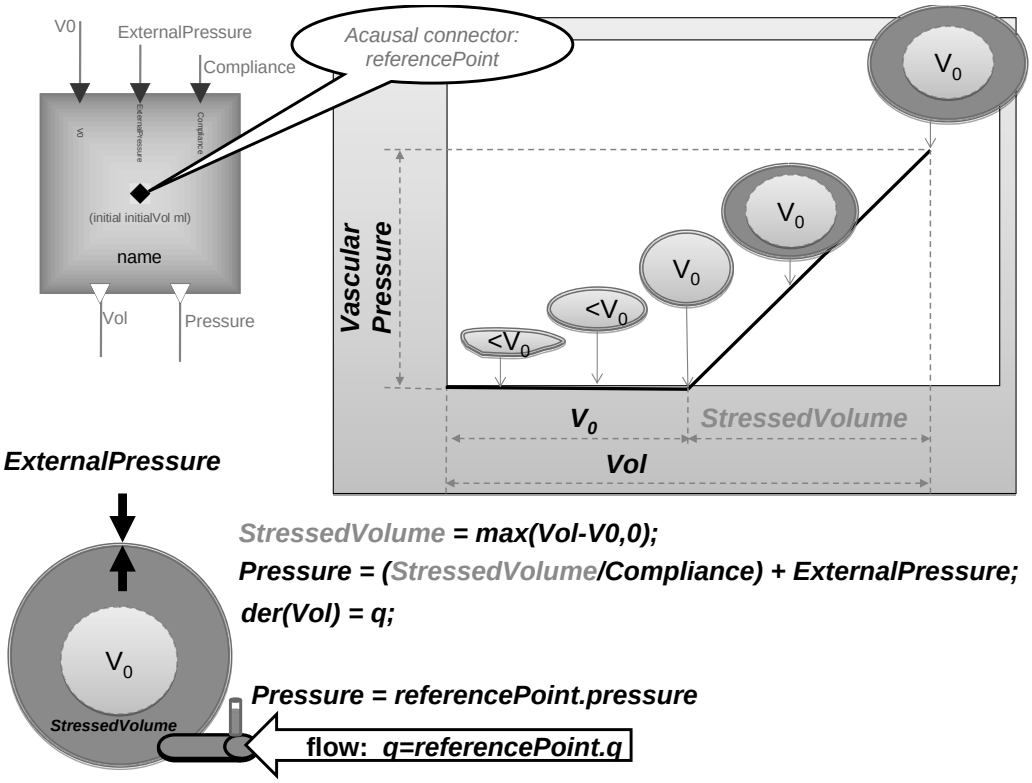


Figure 27: The concept of an elastic vascular compartment is based on the idea that when a blood vessel fills with blood, the pressure in the vessel is determined only by the external pressure on the vessel until a certain residual volume (*V0*) is achieved; the elastic and muscle fibres in the blood vessel will then start to tense and compress the volume of blood in the vessel with the *VascularPressure* pressure. If we label the volume of fluid in the blood vessel *Vol*, then the volume of blood stressing the vessel (*StressedVolume*) will determine the Pressure inside the vessel depending on its Compliance and on the external pressure on the vessel (*ExternalPressure*). The vascular compartment is connected to the system by means of the *ReferencePoint* connector, through which blood may flow into the compartment (at the rate of *referencePoint.q*) under pressure (*referencePoint.pressure*).

If the connector is connected to the environment through a connector, the pressure value will truly be the same in all nodes connected to the compartment, and the flow will be distributed to all connected nodes so that its algebraic sum will be zero (nothing ever accumulates in the area of branching) – see the example of the component connection in Fig. 28.

Three signal (causal) inputs will enter the compartment from the outside:

- Basic charge "*V0*" – the value of the volume that must be reached before the pressure in the elastic compartment starts increasing. If the volume is less than zero, the pressure in the compartment will be zero.
- Outer, external pressure "*ExternalPressure*" – the pressure of the ambient environment on the elastic compartment.
- "*Compliance*" of the elastic compartment – the pressure in the compartment will be inversely proportional to it if the compartment volume exceeds the basic charge.

Two (causal) signal outputs will go from the compartment to its environment:

- Information about the compartment's current volume, "*Vol*"
- Information about the pressure inside the compartment, "*Pressure*"

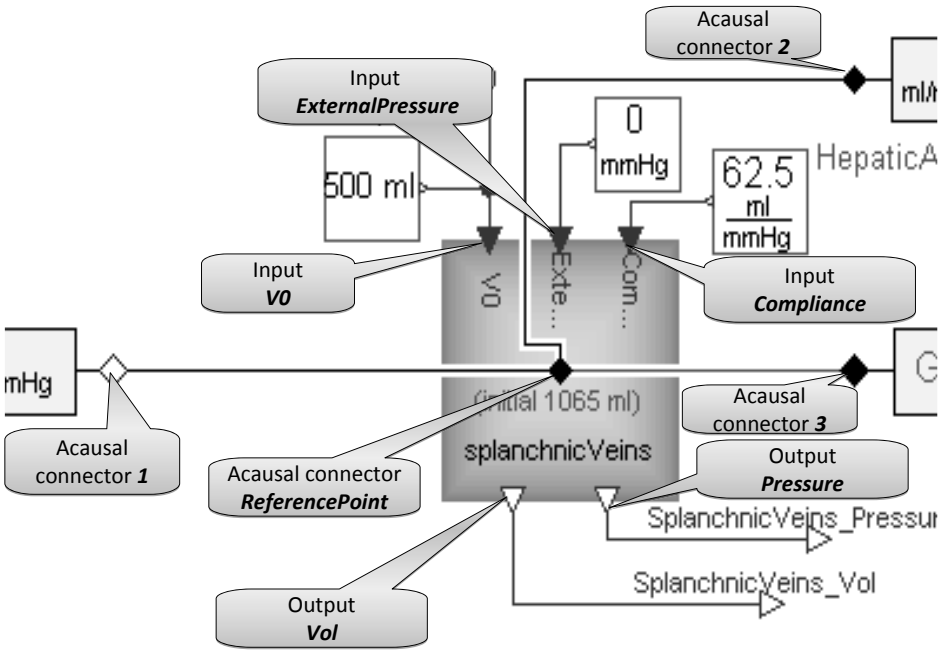


Figure 28: The instance "*splanchnicVeins*" of the elastic compartment "*VascularElasticBloodCompartment*". The acausal connection with the appropriate connectors at controllable resistors (labelled "*connector1*", "*connector2*" and "*connector3*" here) will interconnect the equations in the elastic compartment instance "*splanchnicVeins*" into a system of equations of all the interconnected elements. The pressure value will be the same at all interconnected connectors: *splanchnicVeins.ReferencePoint.pressure = connector1.pressure = connector2.pressure = connector3. pressure.*

The algebraic sum of all flows at the interconnected connectors must be zero: *splanchnicVeins.ReferencePoint.q + connector1.q + connector2.q + connector3.q = 0.*

It is useful to design another parameter for the compartment (whose value will be read before the start of simulation), which would specify its initial charge:

Initial compartment volume, "*initialVol*"

We can also design an icon to display the elastic component in the programming environment.

The actual fragment of code describing the behaviour of the elastic compartment looks like this in Modelica:

```
model VascularElacticBloodCompartment extends QHP.Library.Interfaces.BaseModel;
  Real StressedVolume ( final quantity="Volume", final unit="ml");
  parameter Real initialVol(  final quantity="Volume", final unit="ml")
                              „initial compartment blood volume";
  ...
  initial equation
      Vol = initialVol;
  equation
      der(Vol) =  referencePoint.q;
      StressedVolume = max(Vol-V0,0);
```

Pressure = (StressedVolume/Compliance) + ExternalPressure;

referencePoint.pressure = Pressure;

end *VascularElacticBloodCompartment;*

The first line declares the model class; in addition, there is the declaration of a real variable, "*StressedVolume*", whose physical units will be checked. This is not just a question of code clarity and readability. The check of unit compatibility will enable us to avoid a very hard-to-find error, when we exchange connectors in interconnections by mistake (if units are found to be incompatible, the check will not allow us to create the wrong interconnection at all).

Then there is the declaration of an "*InitialVol*" parameter, whose physical units will be checked as well. And then there is the equation section. The initialization of the compartment's initial volume, i.e. the variable "*Vol*", is declared first. The other lines in the equation section declare four equations. The first one is a differential equation – the derivative of the volume "*der(Vol)*" equals the inflow "*q*" from the connector "*referencePoint*".

The next equation declares that the value of the elastically stressed volume "*StressedVolume*" will be calculated as the difference between the compartment volume "*Vol*" and the value of its basic charge "*V0*" (which is an input); the equation also says that the value of the compartment volume may never drop down to negative values.

The third equation declares the relation between the "*Pressure*" in the compartment, the value of the "*StressedVolume*", the "*Compliance*" and the "*ExternalPressure*". We would like to repeat that these are not assignments but equations. The equation could also be written like this in Modelica:

Pressure - ExternalPressure= (StressedVolume/Compliance);

The last equation interconnects the value of "*Pressure*" in the compartment with the value of the pressure interconnected with its environment by the acausal connector through the "*referencePoint. pressure*".

The value of "Pressure" is also a signal output from the compartment – as a signal, it can be fed to other blocks – but it is a causal output (signal) variable and its value cannot be affected by what we connect it to. However, the situation is different with the interconnection from the acausal connector. When we interconnect an instance of the elastic compartment with other elements through the acausal connector, the four equations in the compartment will become part of the system of equations defined by the interconnection and the values of the variables in the elastic compartment instance will depend on the solution of the originated system of equations.

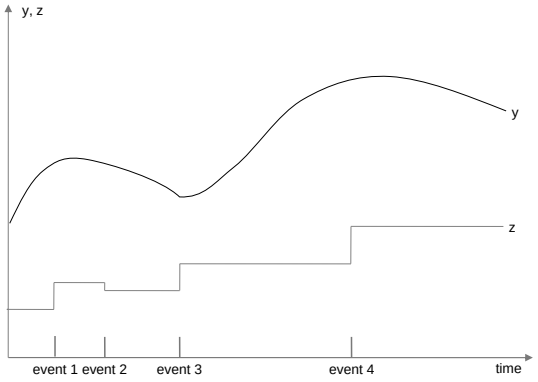


Figure 29: An example of the behaviour of real variables in hybrid systems. The continuous time variable "*y*" changes in time (its value does not have to be continuous – e.g. it may change discontinuously, perhaps in a jump, in response to an event). The real discrete-time variable "*z*" only changes its values at event instants.

Hybrid models

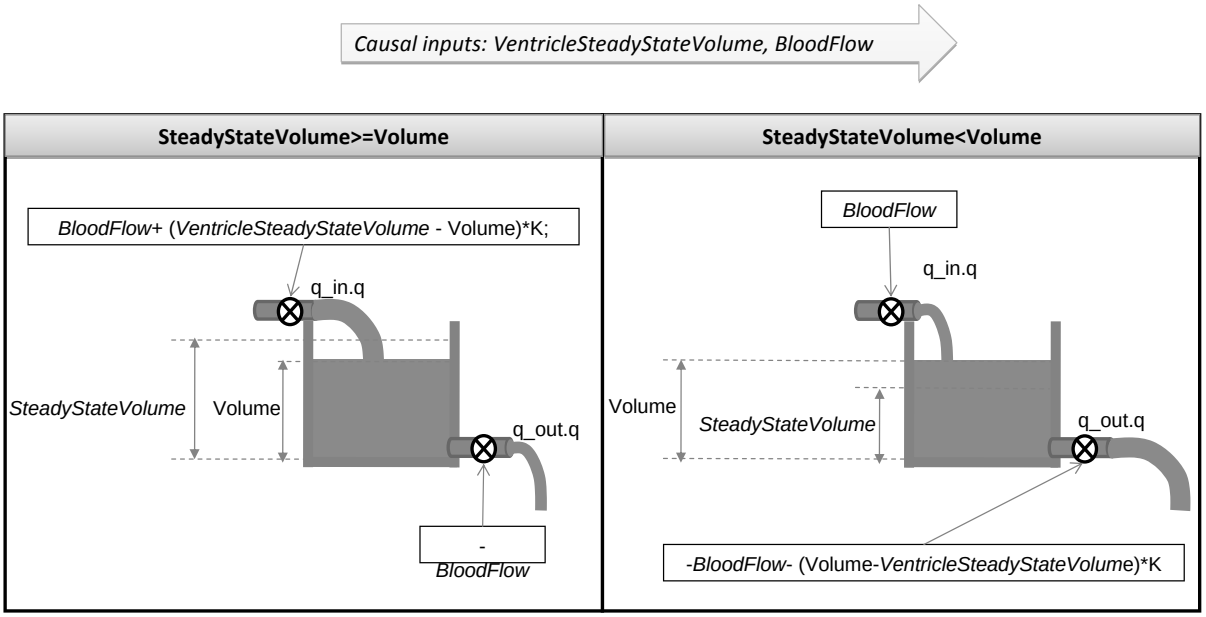
Continuous dynamics expressed by a system of algebraic differential equations is often enough for the mathematical description of real-world models. However, we frequently need to represent discontinuous, discrete behaviour (which is often an approximation of quick continuous processes in physical systems) and continuous dynamic systems themselves are not enough for the description of real-world processes – examples include the opening and closing of valves in the hydraulic and pneumatic

domain, the behaviour of diodes in the electrical domain or the switching on/off of genes, the creation and transmission of nerve impulses or the opening and closing of ion channels in the biological domain. Discrete event dynamic systems are frequent in the description of technical applications. Discrete hierarchical state automata are a very powerful tool for the formalized description of processes and their interactions (Harel, 1987).

When modelling large systems, it is often useful to combine discrete and continuous description to a lesser or greater extent. Such "hybrid" models can combine discrete and continuous time variables, and generate and react to various events (see Fig. 29).

Hybrid models are supported in modern development simulation environments. For example, a continuous dynamic system model in Simulink can be combined with hierarchical state automata created in a special modelling tool, Stateflow – the values of variables in Simulink can change the states of automata in Stateflow, and Stateflow can switch calculation blocks in Simulink by means of generated events, changing the calculation procedure.

However, acausal development tools can directly change the used equations (not just the method of solution). A small illustrative example can be the modelling of the average blood volume in a ventricle




```
der(Volume)=q_in.q+q_out.q
if (SteadyStateVolume >=Volume) then
    q_in.q=BloodFlow+ (VentricleSteadyStateVolume - Volume)*K;
    q_out.q=-BloodFlow;
else
    q_in.q=BloodFlow;
    q_out.q=-(BloodFlow+ (Volume-VentricleSteadyStateVolume)*K);
end if;
```

Figure 30: The Modelica acausal modelling tool allows changing the system of equations in use dynamically. The illustration shows the hydraulic analogy of a ventricle volume model represented as a continuous pump with variable internal volume. Two acausal connectors (*q_in*) and (*q_out*) interconnect the component with its environment; the component receives the value of blood flow (*BloodFlow*) and the required value of blood volume in the ventricle (*VentricleSteadyStateVolume*) as its causal inputs. Equations calculate the volume of blood in the ventricle (*Volume*) and blood inflow and outflow values (*q_in.q*, *q_out.q*). The equations used will vary depending on whether or not the required volume of blood is greater than the current volume of blood in the ventricle (i.e. whether *SteadyStateVolume >=Volume*).

(see Fig. 30).

A ventricle is modelled as a continuous pump with a variable internal volume.

The ventricle model is connected to the circulation by means of acausal connectors "*q_in*" and "*q_out*". These connectors interconnect the flow of blood into ("*q_in*") and out of ("*q_out*") the ventricle. The change of blood volume in the ventricle will be determined by the algebraic sum of flows in both acausal connectors. In Modelica, this will be written as follows:

```
der(Volume)=q_in.q+q_out.q;
```

The model has two causal inputs – one is the current flow in the ventricle ("*BloodFlow*") and the other is the required volume of blood in the ventricle in the steady state ("*VentricleSteadyStateVolume*"). If that volume is greater than the current ventricle volume ("*Volume*"), then inflow to the ventricle will be set to a larger value than the outflow, proportionally to the difference between the required value and the actual value:

```
q_in.q = BloodFlow + (VentricleSteadyStateVolume - Volume)*K;
```

Outflow from the ventricle ("*q_out.q*") will be set to the value of "BloodFlow" with a negative sign, because it flows out of the compartment:

```
q_out.q=-BloodFlow;
```

Alternatively, when the required value of blood volume in the ventricle ("VentricleSteadyStateVolume") is less than the actual value ("*Volume*"), the inflow of blood will be set to "*BloodFlow*" and the outflow of blood will be set to a larger value than the inflow, proportionally to the difference between the actual value and the required value. The equation notation fragment in Modelica then looks like this:

```
model VentricleVolumeAndPumping;
....
equation
der(Volume)=q_in.q+q_out.q
    if (SteadyStateVolume >=Volume) then
        q_in.q=BloodFlow+(VentricleSteadyStateVolume - Volume)*K;
        q_out.q=-BloodFlow;
    else
        q_in.q=BloodFlow;
       q_out.q=
      (BloodFlow+(VolumeVentricleSteadyStateVolume)*K);
    end if;
end VentricleVolumeAndPumping;
```

Two equations are then switched over in the model's system of equations, depending on the values of the variables "*Volume*" and "*VentricleSteadyStateVolume*". At first sight, the notation looks like an assignment (as in standard programming languages) but they are equations. An equivalent notation may look like this:

```
model VentricleVolumeAndPumping;
```

```
....
equation
        delta = (VentricleSteadyStateVolume - Volume)*K;
        der(Volume) = delta;
        q_in.q + q_out.q = delta;
        if (delta<0) then
            q_in.q=BloodFlow;
        else
            q_in.q=BloodFlow+delta;
        end if;
end VentricleVolumeAndPumping;
```

Because they are equations, their order does not matter; nor does it matter whether the value of the variable "*delta*" in the third equation is on the right or on the left.

The actual notation of the equations used in Modelica is even more compact:

```
model VentricleVolumeAndPumping;
....
equation
        delta = (VentricleSteadyStateVolume - Volume)*K;
        der(Volume) = delta;
        q_in.q + q_out.q = delta;
        q_in.q = if (delta<0) then BloodFlow else BloodFlow+delta;
end VentricleVolumeAndPumping;
```

Modelica allows describing discrete and continuous systems acausally, providing many possibilities of combining models with discrete and continuous parts. Details can be found in Fritzon, 2003.

Combining acausal and causal (signal) connections in hierarchically arranged models

Modelica makes modelling large systems easier and more controllable and supports their hierarchical decomposition.

Modelica's object-oriented architecture supports the structuring of models into suitable parts having a coherent meaning so that they can be examined separately under certain conditions or re-used (whether in a different place in the same model or in another model), greatly enhancing the clarity of the created models. That is why we create large, reusable libraries of Modelica "simulation chips" in Modelica and each model is usually accompanied by an extensive, hierarchically arranged library of elements. Hierarchical components can be clicked to expand, which will reveal their internal structure.

An example of the hierarchical structure of a Modelica program is the "*VascularCompartments*" class (see Fig. 31), which implements a part of the blood circulation subsystem and makes use of an instance of the above-mentioned class "*VascularElasticBloodCompartment*". Blood flows through acausal connectors between elastic compartment instances, resistances of individual parts of the vascular

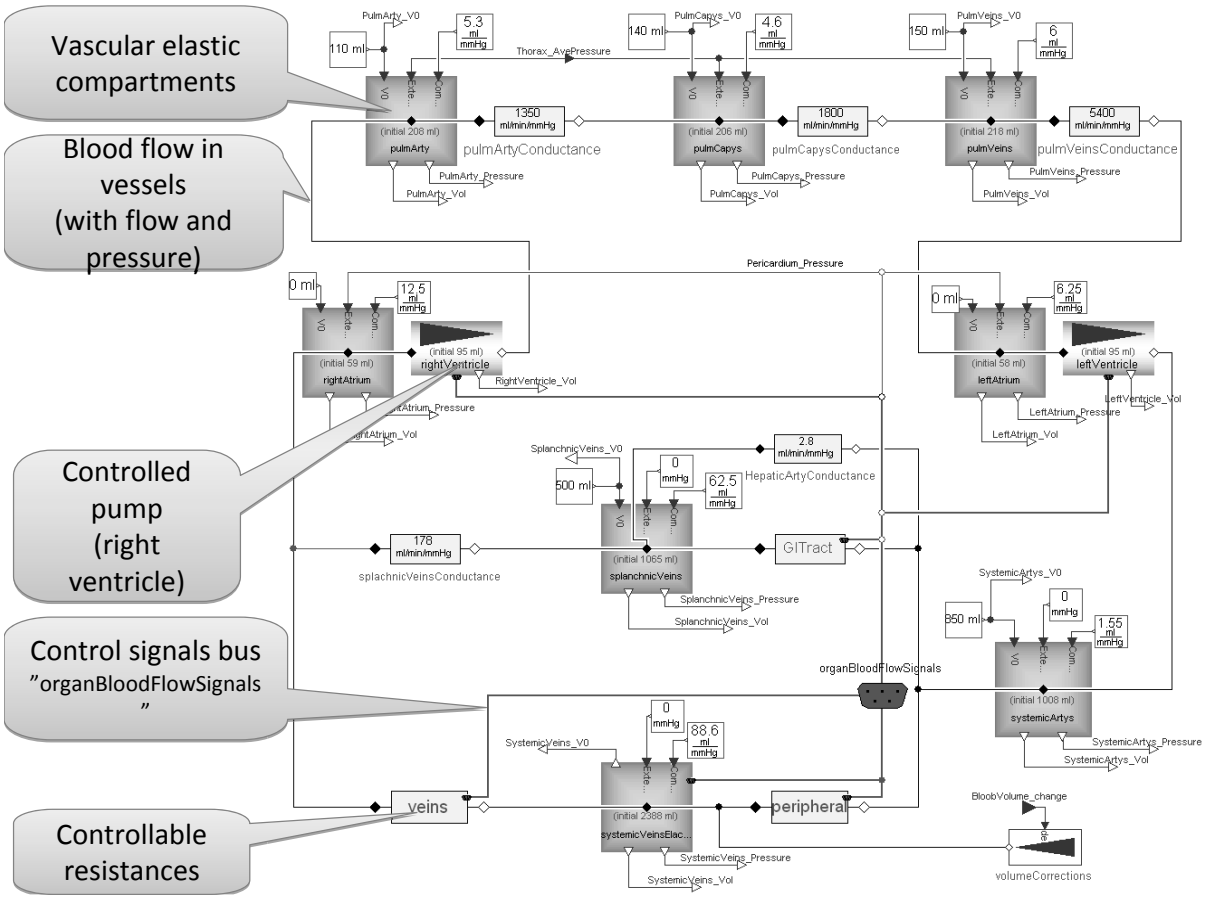


Figure 31: An example of a part of the blood circulation subsystem – an instance of the "*VascularCompartments*" class in Modelica (a part of the Modelica implementation of the large Quantitative Human Physiology model). The model combines acausal and causal (control, signal) connections. In this case, the interconnection by means of acausal connections models the distribution of blood flows and pressures among the individual interconnected components. The model is organized hierarchically; individual blocks can be clicked to expand and represent instances of classes in which equations are specified. The Modelica network thus represents the structure of the modelled system much better than networks in causal modelling tools, which rather represent the calculation procedure.

system and two pumps modelling the activity of the right and left ventricles. The component also uses causal signal connections. An entire set of signal connections (coming from outside the component) is distributed e.g. by the "OrganBloodFlowSignals" bus. Input signal connections control the value of the peripheral resistors and the pumping functions of the right and left ventricles. The structure of the model represents the structure of the modelled reality much better and much more clearly than models created in causal modelling environments. Just compare the Modelica model in Fig. 31 to the model shown in Fig. 6, implemented in Simulink. The two models represent roughly the same – the flow through an elastic vascular system and a heart pump (however, the Modelica model has more details). The Simulink model represents the calculation procedure rather than the structure of the modelled system. The advantage of acausal modelling tools is particularly evident in more complex models, where the possibility of hierarchical model decomposition is crucial for success, as it is important for the interconnection of components to always express in an aggregated manner the cardinal relations at a given hierarchical level while details can be obtained by digging deeper into the structure of individual components, which will reveal the aggregated structure of the modelled reality at a lower hierarchical level.

For instance, the component representing the pump of the right ventricle is connected to the elastic compartment of the right atrium and the elastic compartment of the pulmonary arteries by means

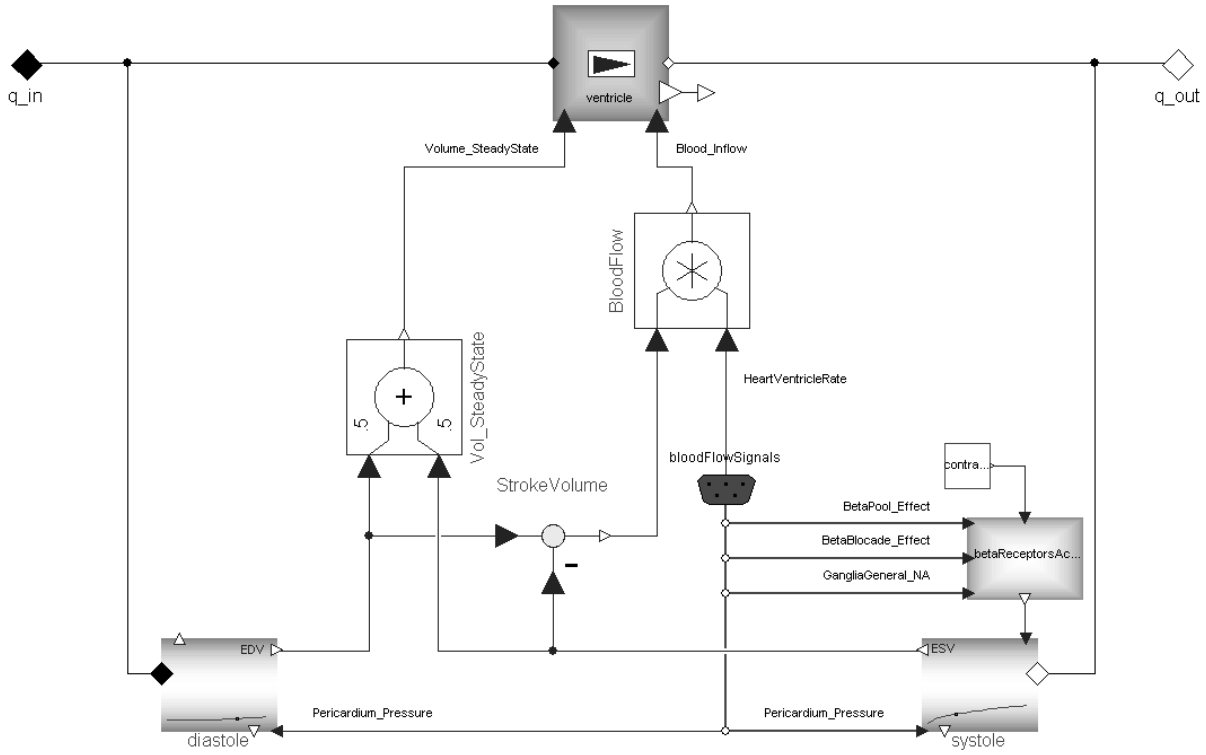


Figure 32: The "inside" of an instance of the right ventricle pump (component "*rightVentricle*") from Fig. 30. The ventricle is modelled as a continuous pump with variable internal volume.

of two causal connectors (distributing the blood flow and blood pressure). Causal signal control inputs are connected to it from the "organBloodFlowSignals" bus. The "inside" of the component is shown in Fig. 32.

The heart is a pulsation pump that first draws blood from the atria into the ventricles during "diastole" – at the end of diastole, the volume of blood in the ventricle equals the end-diastolic volume (EDV). After the end of diastole, the valves between the atrium and the ventricle close and the ventricle starts contracting during "systole". The appropriate valves open and the right ventricle starts pumping blood out to the pulmonary artery (the left ventricle to the aorta). At the end of systole, the valves between the ventricle and the pulmonary artery in the right ventricle (and between the ventricle and the aorta in the left ventricle) close – the volume of blood in the ventricle at the end of systole is called end-systolic volume (ESV). The ventricle muscles relax, the pressure gradient between the atrium and the ventricle opens the atrioventricular valves and diastole begins again.

In the ventricle model, the end-diastolic volume (EDV) is calculated in a "diastole" component and the end-systolic volume in a "systole" component. Modelica allows not only designing of the graphic form of icons representing the individual components but also animating the icons (to improve clarity). In the given example, both components have animated curves during simulation, which represent the relation between pressure in the ventricle and the values of ESV and EDV. A dot on the curves represents the current value of EDV/ESV. Blood pressure in the ventricle is derived from the value of the supply pressure in the atrium (this gets to the component from "q_in" by means of an acausal connector) and the value of the external pressure in the pericardium – this gets to the ventricle from the signal

bus “BloodFlowSignals” by means of a causal connector, “Pericardium_Pressure”. In the “systole” component, blood pressure in the right atrium at the end of systole is derived from the value of the counter-pressure in the pulmonary artery (or pressure in the aorta in the left ventricle) – by means of an acausal connector, “q_out”, and the value of the external pressure in the pericardium (by means of the causal connector “Pericardium_Pressure”). During systole, the dependency of the ESV value on the end-systolic pressure is also affected by the stimulation (or blocking) of “beta receptors”, which results in changes in the contractile power of the heart muscle. A detailed description of equations that describe this dependency is contained in “BetaReceptorsActivityFactor”, a component whose output is the causal input for the “systole” component.

The ventricle model in Fig. 32 is not expressed as a pulsation pump but rather as a continuous pump with variable internal volume. We do not model pumping “beat by beat” but by the average cardiac output per minute.

The systolic volume is calculated first (in the component “StrokeVolume”), as the difference between the end-diastolic (EDV) and end-systolic (ESV) volumes. The value of the blood flow per minute is calculated (in the “BloodFlow” multiplier) from the systolic volume multiplied by the heart rate. The value of the heart rate (“HeartVentricleRate”) comes from the outside, from the “bloodFlowSignals” bus.

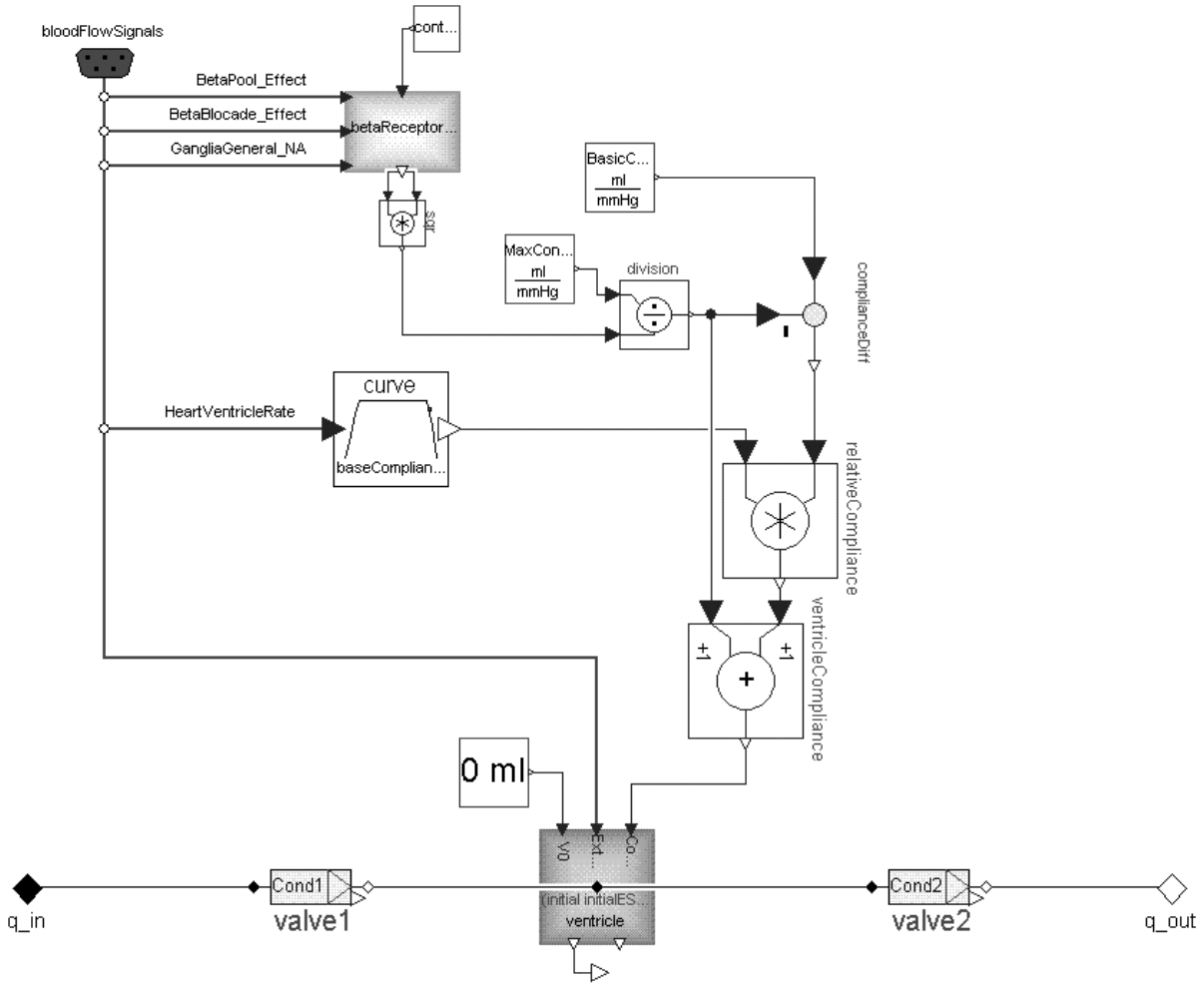


Figure 33: A ventricle model with valves, which generates a pulsating blood flow beat by beat. It has the same outer interface for interconnecting into the model of a higher hierarchical level as the pump model.

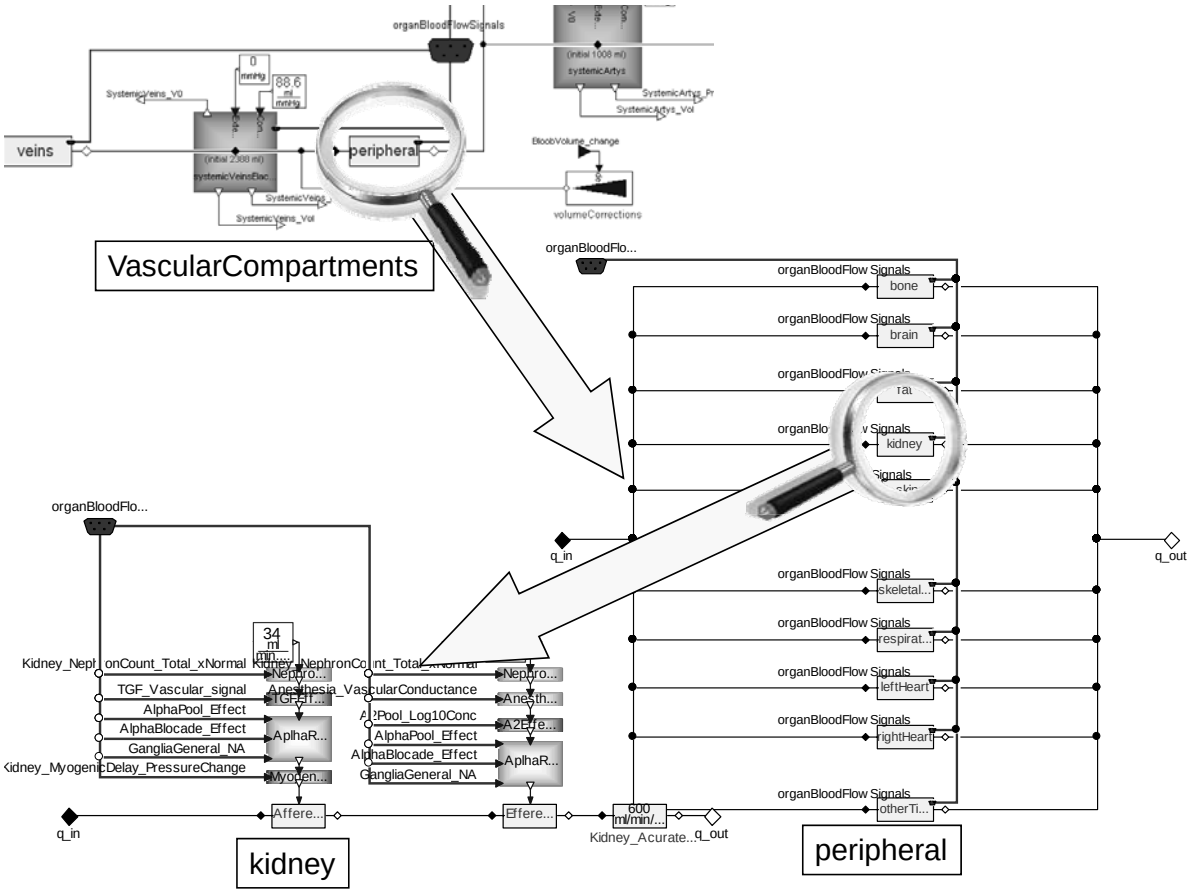


Figure 34: Hierarchical arrangement of models in Modelica. The "*VascularCompartments*" component (from Fig. 30) has one peripherally controlled resistor named "*peripheral*". When clicked, it will expand and show a number of controlled resistors connected in parallel. Clicking one of them – named "*kidney*" – will display complexly controlled resistors in the kidneys. The combination of acausal and (causal) signal connections and the wide range of graphical options for displaying the modelled relations allow the creation of hierarchically structured and "self-documenting" models.

The average volume of blood in the ventricle is estimated as the arithmetic mean ("Vol_ SteadyState") of the maximum heart charge in diastole (EDV) and heart volume at the end of systole (ESV).

The ventricle is represented (by the "ventricle" component) as a continuous pump that has variable internal volume (the component is an instance of the model from Fig. 30). The pump is connected to the blood circulation by means of acausal connectors ("q_in" and "q_out"). It receives the calculated value of the cardiac output (Blood_inflow) and the required average value of the pump's internal volume ("Volume_SteadyState") by means of two causal connectors.

The model of the heart approximated as a continuous pump is sufficient (and sufficiently quick) for a number of applications in medical simulators. However, if we wish to model e.g. various valve defects, we have to use a more detailed model, describing the behaviour of the ventricle beat by beat.

Replacing a simpler component with a more complex component does not have to mean reworking the entire model. Model notation in Modelica allows a very elegant exchange of components as different variants of classes with the same interface.

For example, it is possible to exchange the instances of the left and right ventricle models ("rightVentricle" and "leftVentricle" components) inside the blood circulation subsystem model (see Fig. 31): Instead of the continuous pump model of the ventricles (Fig. 32), we can insert instances of a more complex model into the diagram, generating blood flow beat by beat. We just have to cast the

instances of the left and right ventricles.

The basis of the ventricle model with valves that generates a pulsating blood flow beat by beat (Fig. 33) is an elastic compartment ("ventricle"), which has a generated oscillating value of compliance (unlike the elastic compartment used in the blood vessels). The frequency of the oscillations is determined by the number of heartbeats per minute. The shape of a single compliance change period (the "curve" component) expresses the properties of the heart muscle. The amplitude is affected by the stimulation and blocking of the beta receptors. At last, the direction and rate of blood flow in the ventricle is derived automatically from the properties of the valve components ("valve1" and "valve2" components) and from the pressure gradients.

A simple valve model can be represented as an analogy of a series connection of an ideal diode with a resistor. An alternative (more complex) model of the valves will allow the modelling of various valve defects.

By exchanging components of different complexity with the same interface, we can create model instances of different complexity as needed for their application use.

Modelica supports the possibility to exchange individual components by allowing defining interfaces with a variable number of control (input, causal) signals. Depending on the number of control signals, components may be more complex or, conversely, simplified, and their function may be tested when connected to a model of a higher hierarchical level. This major advantage can be used efficiently not only when debugging complex models but also when identifying a model from experimental data.

Making use of the hierarchy and component structure of models is very important in Modelica (see Fig. 34). For the model construction architecture, it is advisable to follow the rule stating that the structure of a component should always fit in a single screen. A complex tangle of connections is not the sign of a good design and calls for trouble.

The purpose of this chapter was not to describe the physiology of blood circulation. We just wanted to use the rather detailed description of the structure of some components to illustrate how acausal modelling tools allow the creation of richly hierarchically structured, easily modifiable, "self-documenting" models.

When modelling extensive systems, such as the models of interconnected physiological regulations as a basis for medical simulators, the acausal modelling environment of the Modelica language is a great help.

From simulation model to educational simulator

For many years we have been using the environment Mathworks Matlab/Simulink as development tool for building simulation kernel of educational simulators. Now, we use a very efficient environment, which utilizes the Modelica simulation language. By our experience, the simulation model development in Modelica is much more efficient.

Creation of the educational simulator is demanding programming work, linked to the results of the simulation model development and to the created elements of interactive graphics.

In accordance with the designed scenario, graphic elements of the user interface must be "knitted" together with the mathematical model programmed in the background. In order to make writing of the simulators easier (and not to have to program an already debugged simulation model "manually" in Visual Studio .NET), here, too, we have developed a special software tool to automatically generate the simulation model from Simulink in the form of a component for the .NET environment.

To facilitate conversion of mathematical models from the Modelica language environment into .NET, we are extending OpenModelica compiler (as part of the international project Open Modelica) to C# simulation code generation (see Fig. 35). Besides interconnection with the model creation tools,

easy connection to graphic components of the user interface under development is important, as well. Flash components can be incorporated into the simulator in the process of creation through an Active X component. The new .NET environment version also introduces entirely new possibilities of creating graphic components. Thanks to the new WPF (Windows Presentation Foundation) technology, komplex graphic components can be created directly in the .NET platform, which include animations, vector graphics, 3D elements etc. (similarly as in Adobe Flash or even with potentially greater possibilities). It is important that the graphic user interface under development is directly integrated with the .NET platform, which removes the need of bridging the heterogeneous worlds of .NET and Adobe Flash in the simulators development. Microsoft Expression Blend provides considerable support of cooperation of artists and programmers thanks to the interface, which separates (and connects) the work of an artists and programmer.

An artist can create komplex animations in this tool very comfortably (using a graphic user interface), and the animations can be controlled easily. The programmer specifies such control by connecting to relevant program modules (the animations can be thus controlled by the simulation model on the background similarly as puppets on strings).

Moreover, the new platform ***Microsoft Silverlight*** shall make it possible to develop simulators, which can run directly in the Internet browser (even on computers with different operating systems – it is only necessary to install the relevant plugin in the browser).

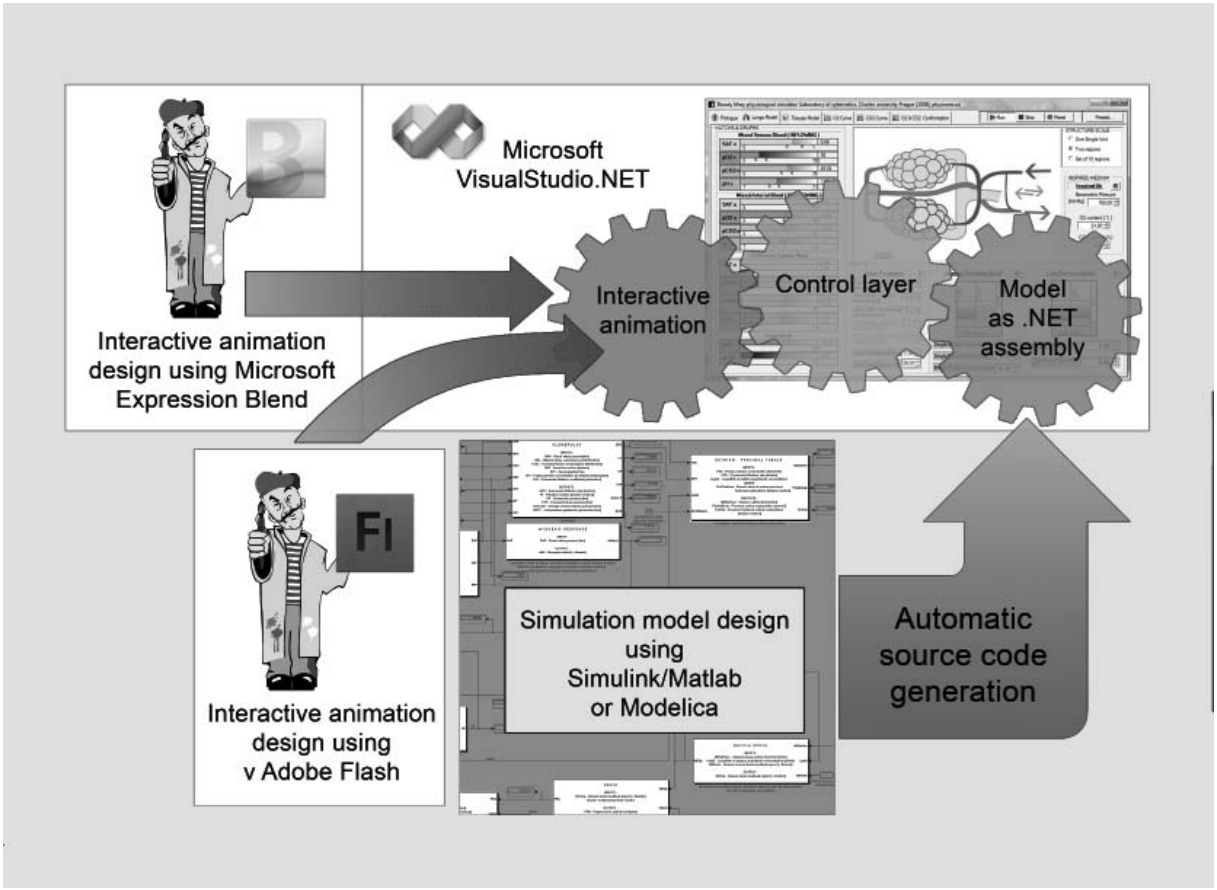


Figure 35: Making simulators in the .NET environment. The model is programmed as a component of the .NET environment (the so called .NET assembly) – preferably by means of automatic generation from models development tools (from the Matlab/Simulink or Modelica programming language environment). Graphic components are created in Adobe Flash or Microsoft Expression Blend. Creating animations in Expression Blend offers the advantage of creating both the animations as well as the simulator in the common .NET platform.

Conclusion

Almost four decades have passed since the publication of the extensive model of Guyton, Coleman and Grander (1972), mentioned at the beginning of this chapter.

Current integrative models of interconnected physiological systems, created as a basis for medical simulators, are much larger. One example is the current model of Guyton's colleagues and disciples – Quantitative Human Physiology, which can be downloaded from http://physiology.umc.edu/ themodelingworkshop. The authors (Hester et al., 2008) have separated the implementation of the simulator and the description of model equations to make the model structure obvious to the wider scientific community. The model equations are described (in a very complex manner) using a special XML-based language. However, the structure of the rather complicated model is rather hard to see from the notation at first sight. The description of the model is distributed over dozens of subfolders, each of which contains files (often more than ten thousand). Consequently, the overall model structure and individual relations are very unclear. That is why we have created a ***special software tool, QHPView,*** providing a well-arranged view of the mathematical relations from the QHP XML notation (see Fig. 36). However, the authors implemented the actual model by means of a custom, block-oriented solver. Since there are a large number of relations in models of such complexity (leading to the solution of implicit equations), block-oriented model implementation (where the outputs of one block are used as the inputs for other blocks) is very complicated and reflects the calculation algorithm rather than the structure of

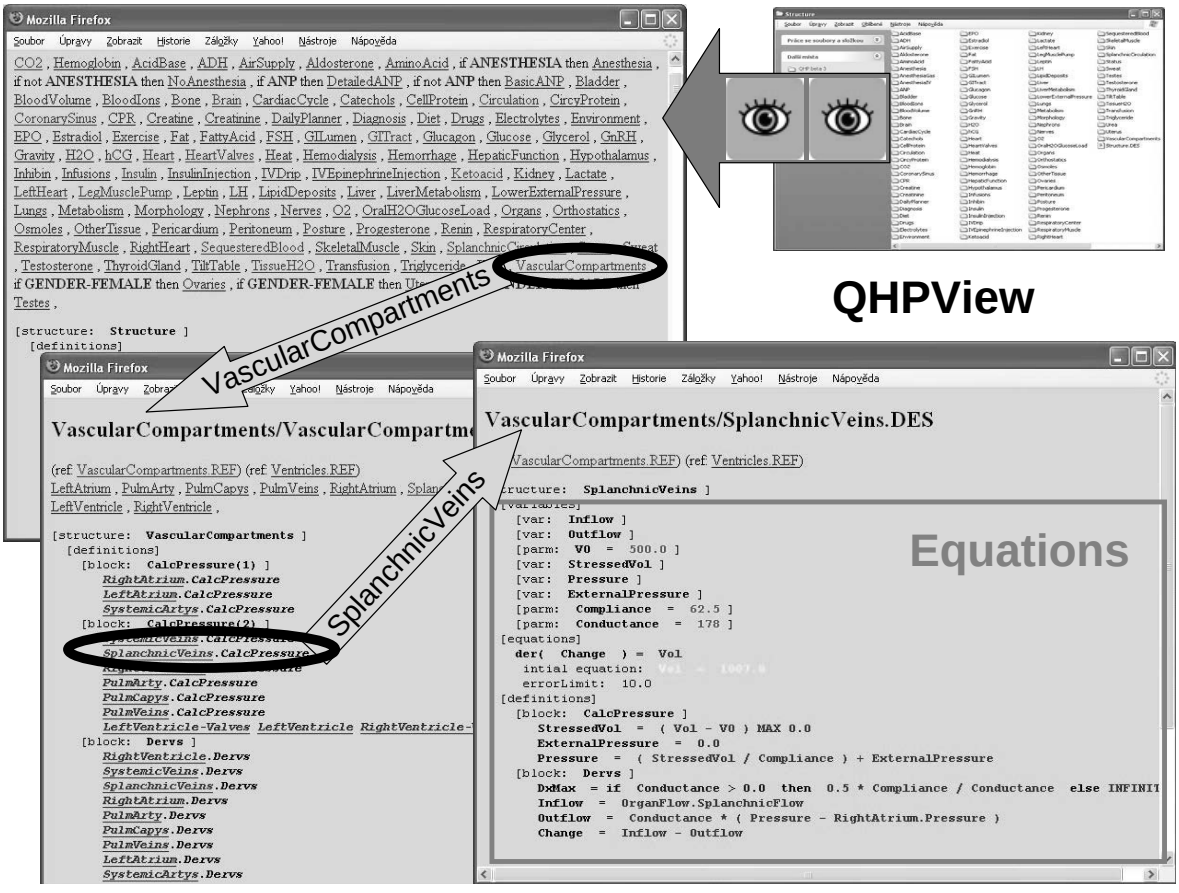


Figure 36: Our *QHPView* tool will allow clarifying the structure of the Quantitative Human Physiology (QHP) model, originally recorded in hundreds of XML files distributed over dozens of folders in which equations and individual relations were hard to identify at first sight. The implementation (and subsequent modification) of such a large model in a block-oriented environment is difficult. Acausal modelling tools, which enable a declarative model notation using the notation of the model's equations, are a great help, in particular for such extensive models.

the modelled relation. As the complexity of a model grows, its clarity decreases.

The use of modern simulation environments where the individual parts of a model can be described directly ***as a system of equations rather than an algorithm for solving the equations*** is very useful, especially for extensive models. The notation of models is declarative – we describe the structure and mathematical relations, not the calculation algorithm (therefore we speak of ***acausal*** modelling).

An acausal description is much better in capturing the fundamentals of the modelled reality and simulation models are much more readable and thus less prone to errors. Object-oriented architecture allows building models with a hierarchical structure, using a large library of reusable elements.

Acausal modelling tools work with interconnected components that are instances of classes in which equations are directly defined, and acausal interconnection defines systems of equations. By contrast with causal modelling approaches, we do not have to bother about the method of solving such equations. We leave it to the machines to find an algorithm for solving them.

These advantages are demonstrated in practice by our implementation of the large QHP model, which we ***implemented (modified and expanded) in the acausal environment of the Modelica language***. Comparing the complex structure of the model from Fig. 35 with the examples of the implementation in Modelica in the previous illustrations, we can see that the acausal implementation leads to a much clearer model structure and makes model modifications and adjustments much easier.

Today's acausal modelling tools are able to generate and numerically solve large systems of equations, which allows the creation of diagrams of physical, chemical or biological processes directly during the implementation of a model. Such diagrams then allow obtaining the results of simulations directly with a mouse click.

New technologies bring about new possibilities and new challenges for the creation of simulation models.

Acausal simulation environments and in particular ***Modelica***, the new object-oriented simulation language that will make modelling large and complex systems significantly easier, are among them.

Acknowledgement

Work on the development of medical simulators is supported under the National Research Programme, Project No. 2C06031, a development project of the Ministry of Education, C20/2009, and by Creative Connections s.r.o.

References

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

[2] Amosov, N.M, Palec, B.L., Agapov, G.T., Ermakova, I.I, Ljabach, E.G., Packina, S.A., Soloviev, V.P.(1977): Theoretical research of physiological systems (in Russian). Naukova Dumaka, Kiev, 1977.

[3] Bassingthwaighte J. B. Strategies for the Physiome Project. *Annals of Biomedical Engeneering* 28, 2000, 1043-1058.

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

[5] Dabney J.B., Harman T.L. (2004) *Mastering Simulink,* Prentice Hall, Houston, 2004, ISBN: 0-13-142477-7

[6] Grodins F.S., Buell J., Bart A.J. (1967): Mathematical analysis and digital simulation of the respiratory control system. J.Appl. Physiol. 22 (2),260-276

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

[8] Guyton A.C, Jones C.E and Coleman T.A. *Circulatory Physiology: Cardiac Output and Its Regulation.* Philadelphia: WB Saunders Company,1973.

[9] Fritzson P. (2003). *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1,* Wiley-IEEE Press.

[10] Hall J.E. (2004): The pioneering use of system analysis to study cardiac output regulation. *Am.J.Physiol.Regul.Integr.Comp.Physiol.* 287:R1009-R10011,2004,287: s. 1009-1001.

[11] Harel, David (1987). Statecharts: a visual formalism for complex systems. In *Science of Computer Programming* 8, 231-274.

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

[13] Hunter P.J., Robins, P., Noble D. (2002) The IUPS Physiome Project. *Pflugers Archive-European Journal of Physiology*, 445, 1–9

[14] Ikeda N, Marumo F and Shirsataka M. A Model of Overall Regulation of Body Fluids. *Ann. Biomed. Eng.* 1979:7, 135-166

[15] Khoo M.C. (2000) *Physiological Control Systems: Analysis, Simulation, and Estimation*,.IEE Press, New York 2000, ISBN 0-7808-3408-6

[16] Kofránek J. Anh Vu L. D., Snášelová H., Kerekeš R. and Velan T., 2001. GOLEM – Multimedia simulator for medical education. In MEDINFO 2001, Proceedings of the 10th World Congress on Medical Informatics. (London, UK, 2001), Patel, L., Rogers, R., Haux R. Eds. IOS Press, London, 1042-1046.

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

[18] Kofránek, J, Rusz, J., Matoušek S., (2007): *Guytons Diagram Brought to Life - from Graphic Chart to Simulation Model for Teaching Physiology.* In Technical Computing Prague 2007. Full paper CD-ROM proceedings. (P. Byron Ed.), Humusoft s.r.o. & Institute of Chemical Technology, Prague, ISBN 978-80-78-658-6, 1-13, 2007. Available at (including source code): http://www.humusoft.cz/akce/matlab07/sbor07.htm#k

[19] Raymond, G. M., Butterworth E, Bassingthwaighte J. B.: JSIM: Free Software Package for Teaching Physiological Modeling and Research. *Experimental Biology* 280, 2003, 102-107

[20] Van Vliet, B.N., Montani J.P. (2005):, Circulation and fluid volume control. In: *Integrative Physiology in the Proteomica and Post Genomics Age.* Humana Press, 2005, ISBN 918-1-58829-315-2, s. 43-66