MODELICA

Jiří Kofránek

Abstrakt

Modely vytvářené pomocí klasických simulinkových sítí přehledně graficky vyjadřují jednotlivé matematické vztahy. V propojkách mezi jednotlivými bloky tečou signály, které přenášejí hodnoty jednotlivých proměnných od výstupu z jednoho bloku ke vstupům do dalších bloků. V blocích dochází ke zpracování vstupních informací na výstupní. Propojení bloků v Simulinku pak odráží spíše postup výpočtu, než vlastní strukturu modelované reality. Hovoříme o tzv. kauzálním modelování. Při vytváření a hlavně při prezentování a popisu modelu je ale důležité, aby vlastní struktura modelu, spíše než vlastní algoritmus simulačního výpočtu, vystihovala především fyzikální podstatu modelované reality. Proto se v moderních simulačních prostředích začíná stále více uplatňovat deklarativní (akauzální) zápis modelů, kdy v jednotlivých komponentách modelu popisujeme přímo rovnice a nikoli algoritmus jejich řešení. Propojením jednotlivých komponent dochází k propojení soustav rovnic mezi sebou. Propojením komponent pak nedefinujeme postup výpočtu, ale modelovanou realitu. Způsob řešení rovnic pak "necháváme strojům". Moderním simulačním jazykem, který je přímo postaven na akauzálním zápisu modelů je Modelica. Jazyk se v poslední době velmi využívá v průmyslových aplikacích, v biomedicínských aplikacích však zatím málo. Praxe však ukazuje, že Modelica je pro modelování biomedicínských systémů velmi vhodným nástrojem, zvláště pro modelování rozsáhlých a komplexních systémů pro lékařské výukové trenažéry. V tomto přehledovém článku je uveden praktický příklad modelování hemodynamiky oběhového systému.

Klíčová slova

Akauzální modelování, Kauzální modelování, Modelica, Simulace

1 Úvod

Internetem zpřístupněné výukové simulační hry doplněné výkladem s multimediálním uživatelským rozhraním jsou novou perspektivní výukovou pomůckou, umožňující názorně ozřejmit vykládaný problém ve virtuální realitě. Jsou moderní realizací starého Komenského kréda "Schola Ludus" (škola hrou) [6]. Na našem pracovišti se léta zabýváme využitím interaktivních multimédií a simulačních her pro lékařskou výuku.

Tvorba výukových programů využívajících simulační hry není jednoduchá a vyžaduje vyřešit dva druhy problémů:

1. Vytvoření simulačního modelu.
2. Tvorba vlastního multimediálního simulátoru pro výukové simulační hry.

Zatímco vytvoření vlastního simulátoru je spíše vývojářskou a programátorskou prací, tvorba simulačního modelu není vývojářský, ale spíše (poměrně náročný)

výzkumný problém, jehož efektivní řešení vyžaduje použít adekvátní nástroje pro podporu tvorby simulačních modelů.

V minulosti se simulační modely vytvářely přímo ve stejném vývojovém prostředí jako i vlastní simulátor (např. v programovacím jazyku Fortran, C++ či Java). V současné době se pro tvorbu a testování simulačních modelů využívají specializované vývojové nástroje.

Jedním z nich jsou nástroje od firmy Mathwork – Maltab a Simulink, v nichž jsme dlouhá léta vyvíjeli a ladili matematické modely. V tomto prostředí jsme vytvořili speciální knihovnu formalizovaných fyziologických vztahů Physiology Blockset, volně dostupnou na našich webových stránkách (www.physiome.cz/simchips). V Simulinku jsme implementovali matematický model, který byl podkladem pro simulátor Golem [23, 31] v Simulinku jsme také implementovali rozsáhlé Guytonovy modely [30] a Simulink jsme také využívali při tvorbě webových simulátorů v našem Atlasu fyziologie a patofyziologie (*http://www.physiome.cz/atlas*) [28]

Simulink zpravidla pracuje s propojenými bloky. V propojkách mezi jednotlivými bloky tečou signály, které přenášejí hodnoty jednotlivých proměnných od výstupu z jednoho bloku ke vstupům do dalších bloků. V blocích dochází ke zpracování vstupních informací na výstupní. Simulink nabízí velkou sadu elementárních bloků (násobičky, děličky, integrátory atd.), realizujících přímo nějakou matematickou operaci, nebo i nějaký test, na jehož výsledku závisí řízení dalšího postupu výpočetního toku. Propojováním těchto elementů se dají vytvářet počítací sítě realizující i poměrně komplikované algoritmy. Propojení bloků v Simulinku proto odráží spíše postup výpočtu než vlastní strukturu modelované reality. Hovoříme o tzv. kauzálním modelování.

V poslední době došlo k vývoji nových tzv. „akauzálních" vývojovým nástrojům pro tvorbu simulačních modelů. Zásadní inovaci, kterou akauzální modelovací nástroje přinášejí je možnost popisovat jednotlivé části modelu přímo jako soustavu rovnic a nikoli jako algoritmus řešení těchto rovnic (to je úlohou pro příslušný překladač. Struktura modelů vytvořených v těchto nástrojích pak odráží strukturu modelované reality a nikoli pouze způsob výpočtu. Tyto nástroje nejčastěji využívají objektově orientovaný programovací jazyk Modelica.

Knih popisujících jazyk Modelica na trhu není mnoho [12, 13, 43], v české literatuře zatím neexistuje žádná. Tento příspěvek je skromným pokusem tuto mezeru v české odborné literatuře trochu zaplnit.

2 Modelica

Modelica není firemní proprietární firemní produkt, jakým je např. Simulink, vyvíjený společností Mathworks. Modelica je standardizovaný objektově orientovaný, deklarativní modelovací jazyk pro komponentové modelování komplexních systémů obsahujících komponenty z různých fyzikálních domén. Jazyk využívá akauzální popis modelované reality pomocí rovnic v jednotlivých modelicových třídách.

Iniciátorem vzniku jazyka byl Hilding Elmqvist z univerzity v Lundu. V roce

1978 vytvořil v rámci své disertační práce jazyk objektově orientovaný, na rovnicích založený jazyk Dymola, který implementoval v jazyce Simula 68 [8]. Později Dymolu reimplementoval v jazyce C++. V roce 1991 založil firmu Dynasim AB, kde pokračoval ve vývoji jazyka Dymola. V roce 1996 inicioval úsilí pro vytvoření standardizovaného objektově orientovaného na rovnicích založeného programovacího jazyka pro modelování technických systémů, který by umožňoval znovupoužitelnost a výměnu jednotlivých komponent dynamických modelů ve standardizovaném formátu. První specifikace jazyka byla zveřejněna v září 1997. Vycházela ze zkušeností nejen s jazykem Dymola, ale i s dalšími modelovacími jazyky jako např. s jazyky Allan [20], Omola [34], Smile [10], ObjectMath [45], SIDOPS+ [1] a jazykem NMF [40]. V roce 2000 bylo založeno nekomerční sdružení Modelica Association (*https://www.modelica.org*) pro standardizaci a další rozvoj jazyka Modelica a pro vývoj standardizované volně dostupné knihovny Standard Modelica Library.

Vznik jazyka Modelica je příkladem úspěšné spolupráce akademického a komerčního sektoru. Firma Dynasim, spolupracující s Lundskou univerzitou vytvořila první komerční implementaci jazyka Modelica pod názvem staronovým Dymola (což již nebyl jazyk, ale simulační prostředí pro tvorbu modelů ve standardizovaném jazyce Modelica). Záhy vznikla další komerční implementace jazyka ve firmě Mathcore která v úzké spolupráci s universitou v Linköpingu vyvinula implementaci Modeliky s názvem MathModelica.

Modelica, která původně vznikala jako akademický projekt ve spolupráci s malými vývojovými firmami při univerzitách v Lundu a v Linköpingu, se záhy ukázala jako velmi efektivní nástroj pro modelování složitých modelů uplatnitelných zejména ve strojírenství, automobilovém a leteckém průmyslu. Vývoj jazyka Modelica proto postupně získal podporu komerčního sektoru. Rozšiřovaly se standardizované knihovny a jazyk se vyvíjel. Modelica nyní existuje ve standardizované verzi 3.3, zveřejněné v květnu 2012 (*https://www.modelica.org/documents/ModelicaSpec33.pdf*). S každou novou verzí se okruh uživatelů Modeliky rozšiřoval a z původně spíše akademického a vědeckého modelovacího prostředku se Modelica stala efektivním průmyslovým nástrojem.

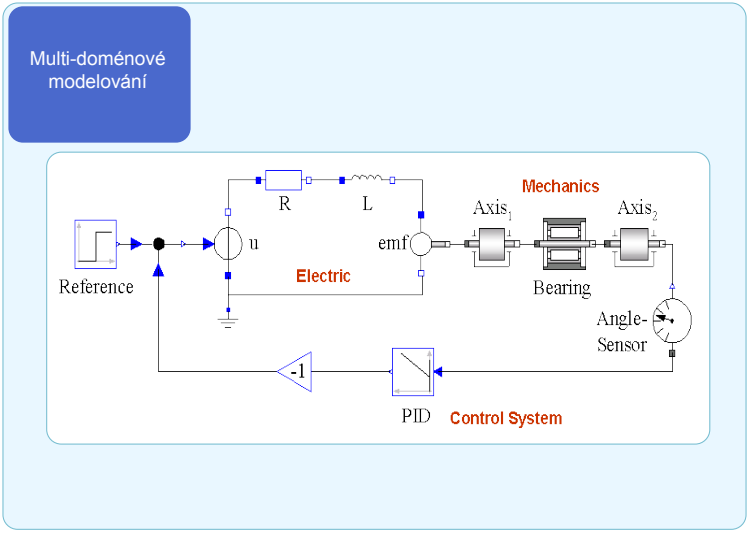
Rychlost, s jakou si nový simulační jazyk Modelica osvojila nejrůznější komerční vývojová prostředí je ohromující. Jestliže ještě před nedávnem existovaly pouze dvě komerční implementace tohoto jazyka (Dymola od švédské firmy Dynasim a MathModelica *od další švédské společnosti Mathcore), dnes již jazyk Modelica využívají také simulační prostředí LMS* Imagine.Lab AmeSim od firmy LMS (*http://www.lmsintl.com*), MapleSim od Maplesoftu (*http://www.maplesoft.com/*), Mosilab od firmy Frauenhofer (*http://www.fraunhofer.de*) a SimulationX od společnosti ITI (*http://www.iti.de*) a mnohé další. Původně malou vývojovou firmu Dynasim, která při univerzitě v Lundu vyvinula první implementaci Modeliky s názvem Dymola dnes vlastní významná společnost Dassault Systemes, (*http://www.3ds.com/products/catia/portfolio/dymola*), která Dymolu zainkorporovala do svých komplexních softwarových nástrojů pro inženýrské konstruování. Společnost Wolfram,

produkující proslulý nástroj pro vědecké a inženýrské výpočty s názvem Mathematica, koupila firmu Mathcore. Její MathModelicu (pod novým názvem System Modeler) zaintegrovala do svého nástroje Mathematica. Existují ale i volně dostupné nekomerční implementace Modeliky, z nichž nejrozšířenější je OpenModelica vyvíjená jako Open Source konsorciem 14 firem a 11 univerzit (viz *https://www.openmodelica.org/*).

Modelica nachází stále větší uplatnění v průmyslových aplikacích. Tento moderní simulační jazyk dnes využívají velké průmyslové korporace, jako Siemens, ABB a EDF. Firmy působící v automobilovém průmyslu, jako (AUDI, MBW, Daimler, Ford, Toyota, VW) používají Modelicu pro návrh energeticky úsporných automobilů a pro návrh klimatizačních jednotek. Rozvoj vývojových protředí a technologií využívajících jazyk Modelica i vývoj příslušných aplikačních knihoven je součástí celoevropských výzkumných projektů EUROSYSLIB, MODELISAR, OPENPROD a MODRIO financovaných v letech 2007-20015 celkovou částkou 75 milionů Euro (viz *http://www.modelica.org*).

V průmyslových simulačních aplikacích se často kombinují komponenty z různých domén – např. elektrické pohony propojené se složitými mechanickými komponenty, nejrůznějšími snímači, čidly, řídící elektronikou apod., často bývá nutnost propočítávat mechanické namáhání i odvod tepla aj. Modelica zde nachází uplatnění, protože umožňuje kombinovat komponenty z různých fyzikálních domén (viz Obrázek 1).

Krom toho, průmyslové technologie jsou složité a komplexní. Proto je

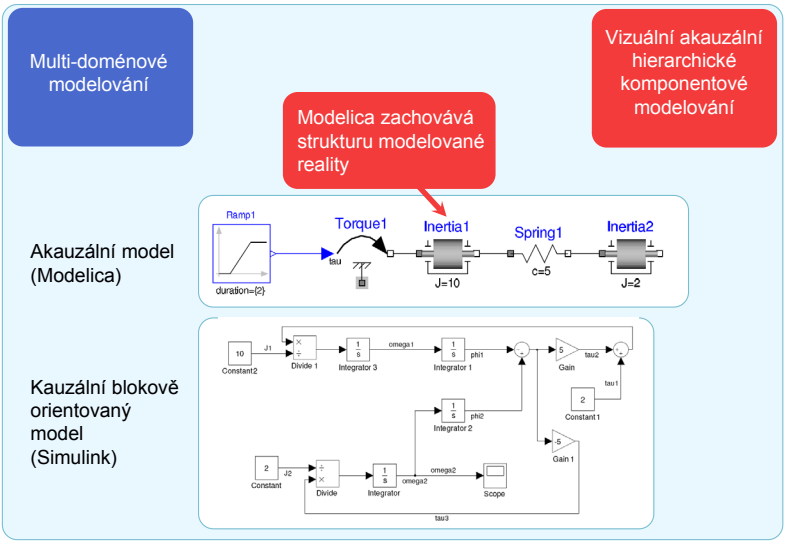


Obrázek 1 — Modelica umožňuje kombinovat modely z komponent z různých fyzikálních domén, například (jako na tomto příkladu) propojovat elektrické, mechanické a řídící prvky.

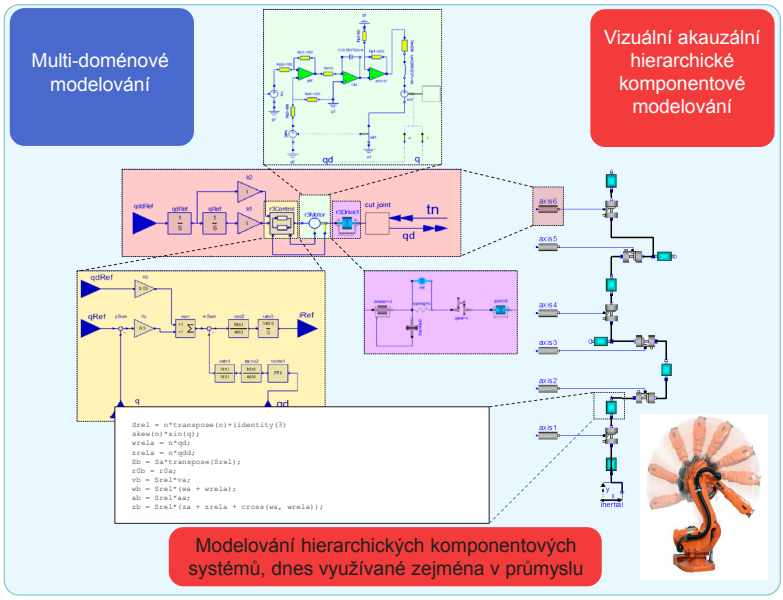
důležité, udržet v modelu přehlednost a neztratit se v množství propojovaných bloků v simulačních nástrojích. A zde je opět Modelica vhodným nástrojem protože umožňuje vizuální hierarchické komponentové zobrazení složitých modelů, jejichž struktura pak vyjadřuje strukturu modelované reality (na rozdíl od blokově orientovaných nástrojů, které spíše než strukturu modelovaného objektu zobrazují postup výpočtu) – viz Obrázek 2.

Velkou výhodou Modeliky pro její využití v průmyslu jsou standardizované knihovny pro různé fyzikální domény (elektrické, hydraulické, mechanické aj.) postupně vyvíjené v široké mezinárodní komunitě a prověřované dennodenním praktickým využíváním. Tyto knihovny modelikových komponent umožňují jejich propojováním sestavovat složité, ale přehledné hierarchické modely, která navenek vypadají jako schematické obrázky specifické pro každou část hierarchicky uspořádaného technologického celku (jinou pro mechanické části, a jinou pro elektrické či řídící komponenty apod.). Tyto obrázky jsou ovšem „živé" a představují funkční spustitelný simulační model (viz Obrázek 3).

Modelica je objektově orientovaný jazyk, kde každá třída může mít svou uživatelem definovanou vizuální podobu a vytvářet tak grafickou podobu



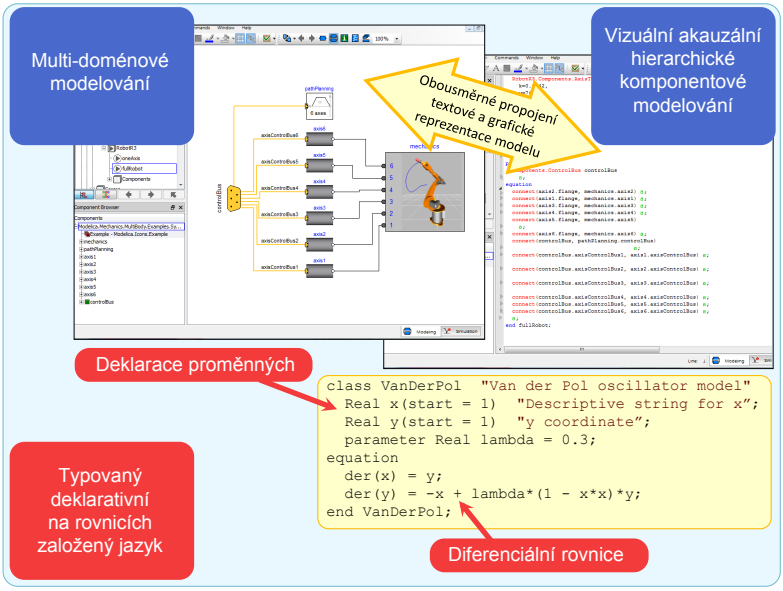
Obrázek 2 — Důležitou vlastností modelicového programu je to, že program skládáme z jednotlivých komponent vizuálně na obrazovce počítače. To ovšem Simulink umožňuje také. Každý blok v Simulinku počítá z hodnot definovaných vstupů příslušné hodnoty výstupů a propojování jednotlivých bloků mezi sebou pak vyjadřuje postup výpočtu. Proto zde hovoříme o kauzálním modelování. V Modelice jsou na pozadí jednotlivých bloků soustavy rovnic. Propojení vyjadřuje souvislosti mezi jednotlivými komponentami (např. cesty přenosu energie, řídící a informační vazby apod.), nikoli způsob výpočtu to je záležitost modelikového překladače). Proto se mluví o akauzálním modelování. Struktura modelu v Modelice pak vyjadřuje spíše strukturu modelované reality, než postup výpočtu.



Obrázek 3 — V Modelice mají modely vizuální hierarchické uspořádání. Při modelování vnitřní struktury jednotlivých bloků se využívají komponenty, přizpůsobené po vizuální i funkční stránce, dané doméně. Tak například vnitřní struktura elektrického motoru odpovídá elektrickému schématu, vnitřní strukturu řídící komponenty popisujeme v blokových schématech užívaných pro popis systémů automatického řízení, vnitřní struktury mechanických komponent vyjadřují struktury propojení mechanických částí, to vše se může kombinovat s komponentami, kde dáváme přednost popisu pouze soustavou rovnic apod. Tímto způsobem můžeme přehledně popisovat složité hierarchicky uspořádané technologické konstrukce a zároveň je tento popis "živým" funkčním simulačním modelem. To je také důvod, proč se Modelica začíná stále více používat při navrhování složitých průmyslových zařízení.

modelikových knihoven. Když přetáhneme v grafickém editoru jazyka Modelica nějakou ikonku z knihovní nabídky na editační plochu, tím zároveň vygenerujeme vytvoření instance komponenty, která je instancí příslušné třídy a navenek je vizuálně reprezentována ikonickým obrázkem. Takto vytvořené vizuální prvky můžeme spolu propojovat a vytvářet model (nebo novou komponentu) z jednotlivých knihovních komponent. Vývojová prostředí jazyka Modelica umožňují obousměrné propojení textové a grafické reprezentace modelu (Obrázek 4). Při propojování komponent v grafickém editoru se automaticky generují textové příkazy jazyka zobrazitelné v textovém editoru.

V Modelice ovšem nevytváříme modely a nové komponenty pouze propojováním komponent z bohaté nabídky modelikových knihoven. V Modelice tvoříme nové specifické komponenty i v textovém editoru. Na rozdíl od jiných objektově orientovaných jazyků, Modelica dává možnost ve zvláštní sekci uvozené klíčovým slovem "equation" psát přímo rovnice. Propojováním jednotlivých komponent tak vlastně definujeme soustavy rovnic modelu.

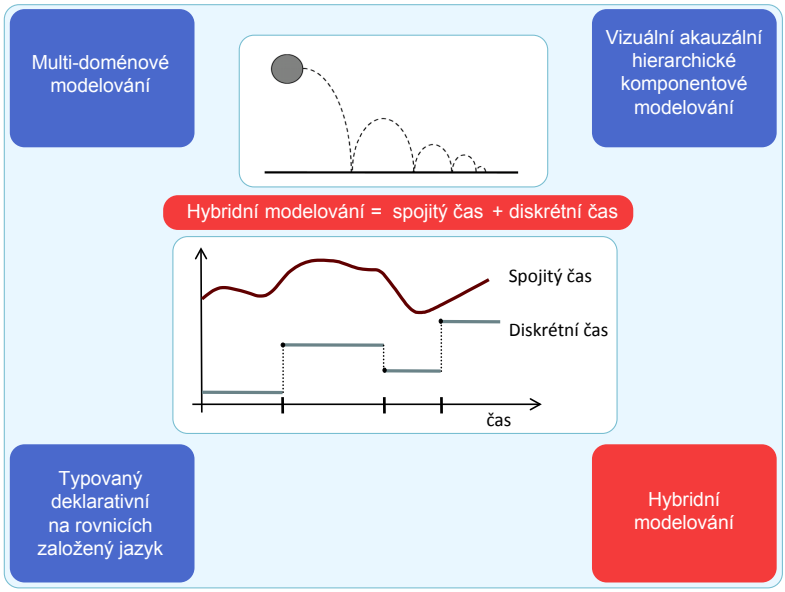


Obrázek 4 — Modelica je normalizovaný modelovací objektově orientovaný jazyk. Model skládáme z instancí tříd. Každá třída má uživatelem definovanou vizuální podobu. Instancí těchto tříd jsou pak vizuální komponenty, které můžeme využívat v grafickém editoru, kde jednotlivé komponenty můžeme mezi sebou propojovat (a model tak "skládáme" z komponent jako z "legových kostiček"). Při tomto skládání je na pozadí grafického editoru automaticky generován zdrojový text modelu, který můžeme prohlížet v textovém editoru. Toto propojení můžeme dělat i "ručně" v textovém editoru. Při vytváření nových "kostiček skládačky" - tj. nových tříd, můžeme také popisovat chování vytvářených komponent v textovém editoru pomocí algebrodiferenciálních rovnic. Textová a grafická reprezentace modelu jsou spolu obousměrně propojeny - vlastní zápis modelu, který je předkládán překladači je ovšem textový. V textové podobě objektově orientovaného jazyka ve zvláštní sekci uvozené klíčovým slovem "equation" píšeme přímo rovnice (rovnítko zde neznamená přiřazení, ale rovnost, takže klidně můžemei levou a pravou stranu rovnice prohodit). Propojením jednotlivých komponent vlastně přidáváme soustavy rovnic (obsažené v jednotlivých komponentách) do systému. Řešení rovnic je starostí překladače, a ne programátora vytvářejícího model.

Jejich řešení je starost překladače Modeliky.

V průmyslových aplikacích se často pracuje s událostmi a diskrétními hodnotami proměnných měněných nespojitě v diskrétním čase (viz např. celá oblast digitální elektroniky). Výhodou jazyka Modelica pro tyto aplikace je to, že umožňuje tzv. hybridní modelování, tj. kombinaci spojitých modelů s diskrétními (viz obr. Obrázek 5). To např. umožní pracovat s událostmi, reagovat na změnu hodnot některých proměnných, nebo na vnější diskrétní vstupy apod.

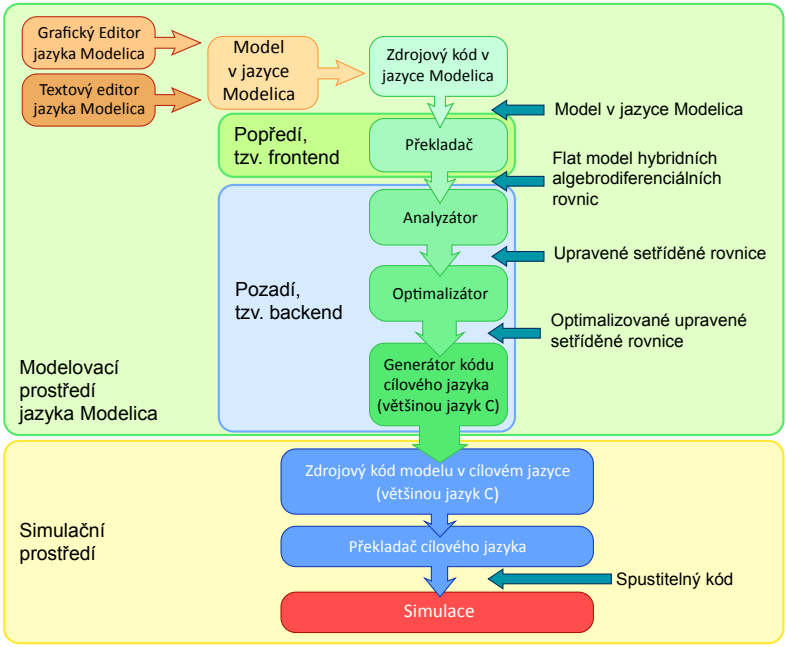
Součástí Modeliky jsou standardní knihovny. Řada dalších knihoven je vytvářena a prodávána samostatně, a protože jsou vytvořeny ve standardizovaném jazyce Modelica, mohou se využívat v různých simulačních



Obrázek 5 — Modelica dovoluje kombinovat spojité modely, vyjádřené soustavou algebrodiferenciálních rovnic, s diskrétními událostmi a s diskrétními proměnnými, jejichž hodnoty se skokově mění v čase. To např. umožní na základě nějaké události změnit soustavu momentálně platných algebrodiferenciálních rovnic - typickým příkladem je třeba modelování skákajícího míčku, kdy při dotyku s oporou se skokově změní směr pohybu. Hybridní modely, kombinující spojitý a diskrétní čas umožňují popsat reakci modelu na změny hodnot některých proměnných, reagovat na vnější události, generovat události a měnit chování modelu apod.

prostředích, které jsou schopny pracovat s tímto jazykem.

Pro vlastní využití Modeliky při tvorbě simulačních modelů potřebujeme modelovací prostředí, v němž model v jazyce Modelica vytváříme a překládáme (obr. Obrázek 6). Součástí těchto prostředí bývá propojený grafický a textový editor, v němž vytváříme a modelikový model. Zdrojový kód Modeliky je ale textový, tvar grafických prvků je definován standardním způsobem v textové podobě v tzv. anotacích, připojitelných ke každé třídě. Modelika je tzv. na rovnicích založený (tzv. „equation-based“) programovací jazyk. O způsob řešení těchto rovnic se stará modelikový překladač, který nejprve provádí symbolické manipulace s rovnicemi. Jeho výsledkem je nejprve tzv. „flat model“, kde jsou veškeré rovnice modelu setříděny tak, aby ze vstupů modelu šly vypočítat výstupy. S tímto „flat modelem“ pak pracuje analyzátor, který rovnice analyzuje, vyřeší veškeré konflikty a upraví. Výsledek předá optimalizátoru, který způsob řešení optimalizuje tak, aby byl co nejrychlejší a dostatečně přesný, zjednoduší rovnice tak, aby se nemusely dělat zbytečné numerické výpočty, a teprve po těchto symbolických manipulacích s rovnicemi jsou optimalizované upravené a setříděné rovnice předloženy generátoru kódu, který vytvoří



Obrázek 6 — Postup tvorby simulačních modelů v modelovacích nástrojích využívajících jazyk Modelica. Zdrojový kód modelu v jazyce Modelica je textový (tvary grafických prvků, s nimiž pracuje grafický editor jazyka Modelica, jsou standardizovaným způsobem popsány v textové formě tzv. anotacích připojených k jednotlivým třídám). Překladač jazyka Modelica provádí symbolickou manipulaci s rovnicemi, setřídí je, upraví je a optimalizuje způsob jejich řešení a nakonec podle nich vygeneruje zdrojový kód příslušného programovacího jazyka (většinou C). Z něj je po přeložení klasickým kompilátorem vytvořen spustitelný kód simulačního modelu.

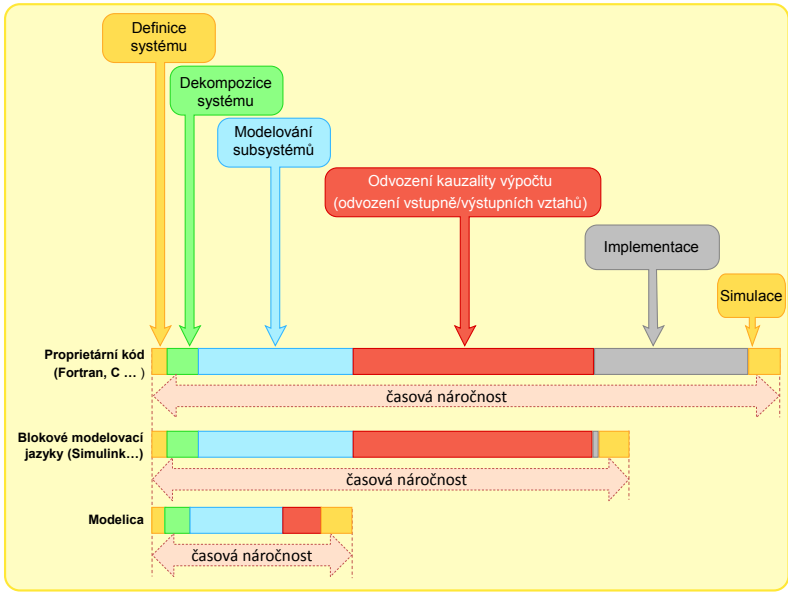
zdrojový kód modelu v cílovém jazyce (většinou C nebo C++) a propojí ho s příslušnými knihovnami numerického řešiče algebrodiferenciálních rovnic.

Výsledkem překladu je zdrojový kód modelu v cílovém programovacím jazyce, který je pak příslušným překladačem přeložen a spuštěn. Součástí simulačních prostředí pro jazyk Modelica jsou též nástroje, které umožňují spouštět model a sledovat průběhy hodnot jeho jednotlivých proměnných.

Součástí komerčních nástrojů je též možnost propojení na další vývojové prostředí (např. součástí modelovacího prostředí od Wolframu je napojení na prostředí Mathematica, společnost Dassault zase propojila Dymolu se svými nástroji pro počítačové konstruování a řízení výroby (CATIA, DELMIA, ENOVIA, SIMULIA a 3DVIA).

Modelica je úspěšná především proto, že podstatně snižuje čas, který je nutno věnovat tvorbě modelu (viz Obrázek 7).

Dříve se modely vytvářely s využitím klasických programovacích jazyků



Obrázek 7 — Blokově orientované jazyky přinesly především zkrácení implementační části modelu oproti tvorbě modelu s využitím klasických programovacích jazyků. Modelica přináší další časovou úsporu, protože odvození postupu, jak ze vstupních proměnných modelu vypočítávat výstupní proměnné, přenechává kompilátoru.

(Fortran, C, C++, Java apod.). V řadě případů modely vytvářejí v těchto jazycích dodnes, zejména v konkrétních aplikacích simulačních modelů (např. jako simulační jádro výukových trenažérů). Speciální simulační prostředí v blokově orientovaných modelovacích jazycích (např. v Simulinku aj.) přinesly podstatné zkrácení doby implementace modelu. Nicméně časová náročnost odvození způsobu výpočtu modelu (tj. nalezení způsobu jak ze vstupů modelu vypočítat výstupy, jak odstranit algebraické smyčky apod.) se mnoho nezměnila. Není to jednoduchá úloha, zejména u komplexních hierarchicky uspořádaných modelů. Čím složitější model, tím je úloha odvození kauzality výpočtu náročnější. A právě zde Modelica přináší velkou pomoc tím, že vyřešení této úlohy nechává převážně na kompilátoru. Díky bohatým knihovnám i vizuálnímu komponentovému modelování je v Modelice také jednodušší modelování subsystémů a dekompozice systému.

Časové úspory při tvorbě modelů jsou největší zejména u složitých, hierarchických multidoménových modelů, se kterými se často setkáváme při návrhu technologicky náročných celků a proto v průmyslu má Modelica rostoucí uplatnění.

Na rozdíl od průmyslových aplikací se však Modelica při tvorbě modelů v medicíně a biologii prozatím příliš neuplatnila.

Drtivá většina biomedicínských simulačních aplikací je dosud realizována

v kauzálních blokově orientovaných prostředích. Patří k nim například vývojové prostředí referenčních databází biomedicínských modelů (v jazycích JSIM http://physiome.org/model/doku.php nebo CEllML http://www.cellml.org/).

Zhusta využívaným prostředím v biologii a medicíně je Matlab/Simulink - monografie věnované biomedicínským modelům bývají často doprovázeny přídatným softwarem pro toto vývojové prostředí [např. 9, 17, 21, 32, 38, 41, 46]. I když v Simulinku byly v posledních letech implementovány akauzální knihovny Simscape a další, modely v biomedicínských aplikacích je zatím téměř nevyužívají. Tak např. tyto akauzální simulinkopvé knihovny nevyužívá ani poslední verze rozsáhlého modelu respiračního a cirkulačního systému PNEUMA, po léta vytvářeného Universitou Jižní Kalifornie v San Diegu. Model PNEUMA je využíván mimo jiné ke studiu řady respiračních poruch, např. spánkové apnoe [4, 5, 11, 19] a jeho zdrojový kód v Simulinku je volně stažitelný (viz http://bmsr.usc.edu/software/pneuma/).

Nicméně již v roce 2006 Cellier a Nebot [3] ukázali výhody, které Modelica přináší pro přehlednou implementaci popisu fyziologických systémů. V Modelice implementovali klasický model McLeodův cirkulačního systému PHYSBE (PHYSiological Simulation Benchmark Experiment) [35, 36, 37]. Tyto rozdíly zvláště vyniknou, porovnáme-li si Cellierovu a Nebotovu implementaci modelu s volně stažitelnou verzí implementace modelu PHYSBE v Simulinku http://www.mathworks.com/products/demos/simulink/physbe/.

Haas a Burnhan [16] ve své monografii upozornili na velké možnosti jazyka Modelica pro modelování adaptivních regulačních systémů v medicíně. Brugård a spol. v roce 2009 [2] referoval o práci na implementaci knihovny značkovacího jazyka SBLM (System Biology Markup Language), používaného jako jeden ze standardů pro popisování modelů biologických systémů (http://sbml.org/), do jazyka Modelica. To by v budoucnu umožnilo jednoduchým způsobem přímo spouštět modely, jejichž struktura je popsaná v jazyce SBLM, na vývojových platformách, založených na jazyce Modelica. Výsledkem jejich práce je mimo jiné první knihovna SBML v Modelice, která je dnes již standardní součástí Wolfram SystemModeler.

V naší laboratoři biokybernetiky na 1. Lékařské fakultě UK jsme v Matlabu a Simulinku po léta vyvíjeli modely fyziologických systémů a rozvíjeli příslušnou aplikační simulinkovou knihovnu Physiolibrary (http://physiome.cz/simchips). Vyvinuli jsme také příslušné softwarové nástroje usnadňující převod modelů implementovaných v Simulinku do vývojových prostředí (Control Web a Microsoft .NET), v nichž vytváříme vlastní výukové simulátory. Náš vývojový tým má dlouholetou praxí poměrně slušné zkušenosti v práci s vývojovým prostředím Matlab/Simulink od renomované firmy MathWorks. Na druhé straně nás ale lákaly nové možnosti vývojových prostředí využívající jazyk Modelica.

Stáli jsme proto před rozhodnutím, zda nadále pokračovat ve vývoji modelů fyziologických systémů v prostředí Simulink (s využitím nových akauzálních knihoven Simscape), nebo zda učinit radikálnější rozhodnutí a přejít na novou platformu jazyka Modelica.

Velmi rychle se ukázalo, že **implementace rozsáhlých modelů v Modelice**

je mnohem efektivnější než pouhé využívání akauzálních knihoven v Simulinku. Při porovnání simulinkové a modelicové implementace se projevil podstatný rozdíl, spočívající zřejmě v tom, že nové akauzální knihovny jsou pouhou akauzální nadstavbou Simulinku a nikoli objektově orientovaným na rovnicích postaveným modelovacím jazykem, jakým je Modelica.

Porovnáme-li tedy vývojová prostředí, založená na simulačním jazyce Modelica s vývojovým prostředím Matlab/Simulink od firmy Mathworks můžeme konstatovat že:

- na rozdíl od Simulinku model implementovaný v Modelice mnohem lépe vystihuje podstatu modelované reality a simulační modely jsou mnohem čitelnější i méně náchylné k chybám;
- objektová architektura Modeliky umožňuje postupně stavět a ladit modely s hierarchickým uspořádáním s využitím knihoven znovupoužitelných prvků;
- na rozdíl od Simulinku (který je průmyslovým standardem firmy Mathworks) je Modelica normalizovaný programovací jazyk, proto také mohou existovat různá komerční (i nekomerční) vzájemně si konkurující vývojová prostředí a pro řešení specifických problémů z různých aplikačních oblastí se v tomto jazyce vytvářejí (komerční i nekomerční) specializované knihovny;
- v Modelice je možné nenásilně kombinovat kauzální (většinou signálové) a akauzální vazby; na rozdíl od Simulinku v propojení kauzálních bloků je možné bez větších problémů vytvářet algebraické smyčky – součástí kompilátoru Modeliky jsou symbolické manipulace na pozadí a proto rozpojení algebraických smyček je starostí vývojového prostředí a nikoli programátora.

Výše uvedené důvody nás vedly k tomu, že jako hlavní implementační prostředek pro tvorbu modelů jsme v naši laboratoři zvolili Modelicu a postupně upustili od vývoje modelů v prostředí Matlab/Simulink [24, 25].

Do tvorby aplikačních knihoven a nástrojů vývojových prostředí pro jazyk Modelica jsme se zapojili i v rámci mezinárodní spolupráce.

Společným úsilím 14 firem a 11 univerzit sdružených v Open Modelica Source Consortium je společně vyvíjeno prostředí OpenModelica, šiřitelné jako open source (Open Modelica Source Consortium - viz *http://www.ida.liu.se/labs/pelab/modelica/OpenSourceModelicaConsortium.html*).

Tohoto vývoje se účastní i náš vývojový tým v rámci aktivit firmy Creative Connections s.r.o., která je členem tohoto konsorcia (viz *http://www.creativeconnections.cz/*). Pro toto konsorcium vyvíjíme nástroj, umožňující z modelu vyvinutého a odladěného v Modelice vygenerovat zdrojový text modelu v jazyce C#. Naším cílem je umožnit z Modeliky přímo generovat simulační jádro pro multimediální webové simulátory pro prostředí .NET. Ve všem pohodlí, které poskytuje jazyk Modelica bude možné vytvářet simulační modely a snadno je propojovat s animacemi, které, jako jakési grafické loutky budou řízené simulačním modelem [29].

Hlavním cílem naším cílem je ale usnadnit propojení složitého

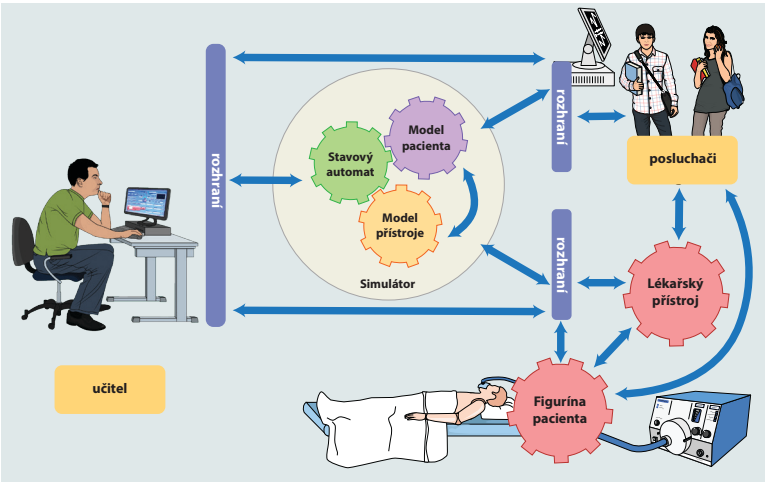
integrovaného modelu fyziologických systémů ***HumMod-Golem Edition***, který vytváříme v jazyce Modelica, s uživatelským rozhraním lékařského trenažéru realizovaného jako multimediální výuková aplikace a perspektivně i s využitím robotizované figuríny pacienta [26].

3 Vytváříme kostičky modelikové stavebnice simulátoru

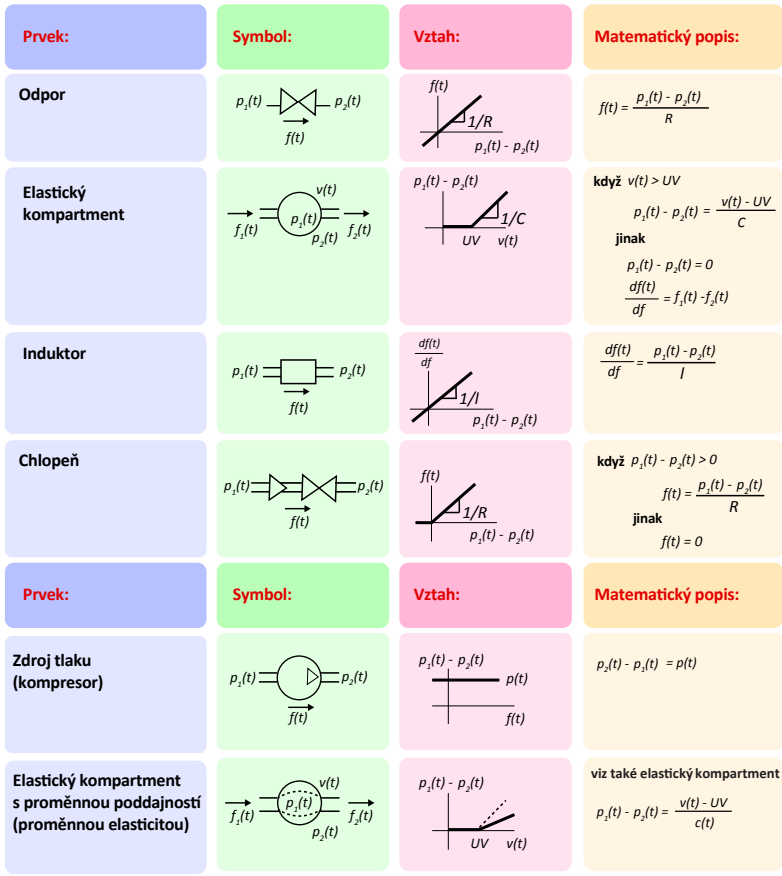
Nejlépe je vhodné ilustrovat výhody jazyka Modelica na nějakém konkrétním příkladě. Vybrali jsme jeden ze subsystémů lékařského trenažéru trenažéru HPS (Human Patient Simulator) *https://caehealthcare.com/home/eng/product_services/product_details/hps-human-patient-simulator*.

Simulátor HPS, patří k tzv. modelem řízeným (model-driven) trenažérům. Na rozdíl od scénářem řízených (patient-driven) simulátorům, které jsou řízeny algoritmem větveného scénáře, je základem modelem řízených pacientských simulátorů matematický model fyziologických systémů, propojený s modelem lékařských přístrojů (umělé plicní ventilace a dalších) – viz obr. Obrázek 8. Oba modely jsou propojeny se stavovým automatem, kde se podle hodnot vstupů a výstupů modelů přepínají jednotlivé stavy a na druhé straně stavový automat nastavuje příslušné parametry modelů zejména podle modelovaného scénáře. Simulátor má dvě uživatelská rozhraní – jedno pro studenty a druhé pro učitele. Učitelský terminál umožňuje řídit simulátor, připravovat simulační scénáře, sledovat akce studentů a dělat následný rozbor (tzv. debriefing). Výstupy jsou rovněž přes příslušná rozhraní propojeny na figurínu pacienta a na připojené lékařské přístroje.

Základním jádrem modelu pacienta v simulátoru HTS je model kardiorespiračního systému a přenosu krevních plynů. Tím se dynamicky provazují mezi sebou ventilační a cirkulační parametry spolu s hodnotami



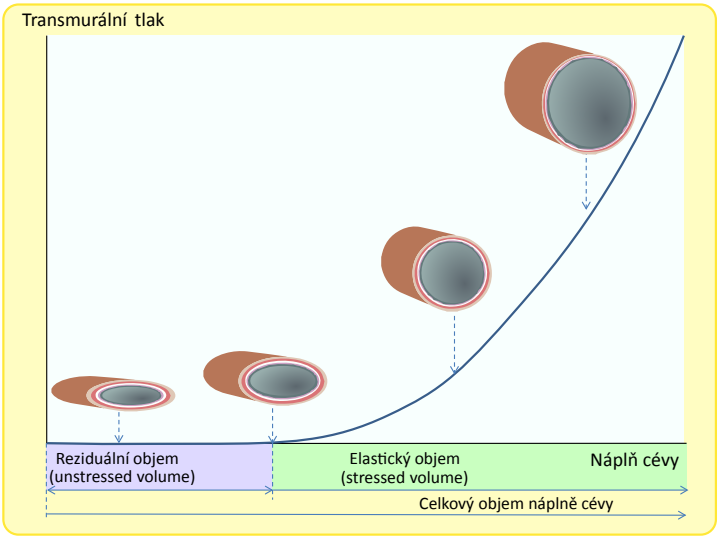
Obrázek 8 — Uplatnění modelu pacienta (přesněji řečeno modelu fyziologických systémů) v tzv. modelem řízeném (model driven) pacientském simulátoru.



Obrázek 9 — Komponenty modelu hemodynamiky podle Meurse (Meurs, 2011). Význam symbolů: R – odpor cévy, p(t) - tlak, v(t) – objem, C – poddajnost, I – induktance, UV – „unstressed volume", reziduální objem cévy (maximální objem náplně cévy který ještě elasticky neroztahuje cévu, pokud je náplň nižší, než tento objem, transmurální tlak je nulový), c(t) – proměnná poddajnost.

krevních plynů a umožňují tak realisticky simulovat dynamické změny při respiračních a cirkulačních onemocněních. Změnou parametrů kardio-respiračního modelu (nastavovanou propojením s farmakologickým modelem) se dá modelovat ovlivnění příslušnými terapeutickými zásahy. Rovněž se dá simulovat ovlivnění kardiorespiračních parametrů a hodnot krevních plynů při simulované umělé plicní ventilaci.

V dalším textu si ukážeme, jak lze implementovat v jazyce Modelica jeden ze subsystémů tohoto kardiorespiračního modelu, konkrétně subsystém hemodynamiky. Veškeré zdrojové texty modelu v jazyce Modelica jsou k dispozici na doprovodném CD ROM a na webových stránkách sborníku



Obrázek 9 — Závislost transmurálního tlaku na náplni cévy. Při zvětšování objemu náplně cévy transmurální tlak (tj. rozdílu tlaků mezi vnitřkem a vnějškem cévy) zůstává do určité hodnoty objemu náplně cévy nulový, a pak se začně (v závislosti na poddajnosti cévy zvyšovat.

MEDSOFT *http://creativeconnections.cz/medsoft* jako elektronická příloha.

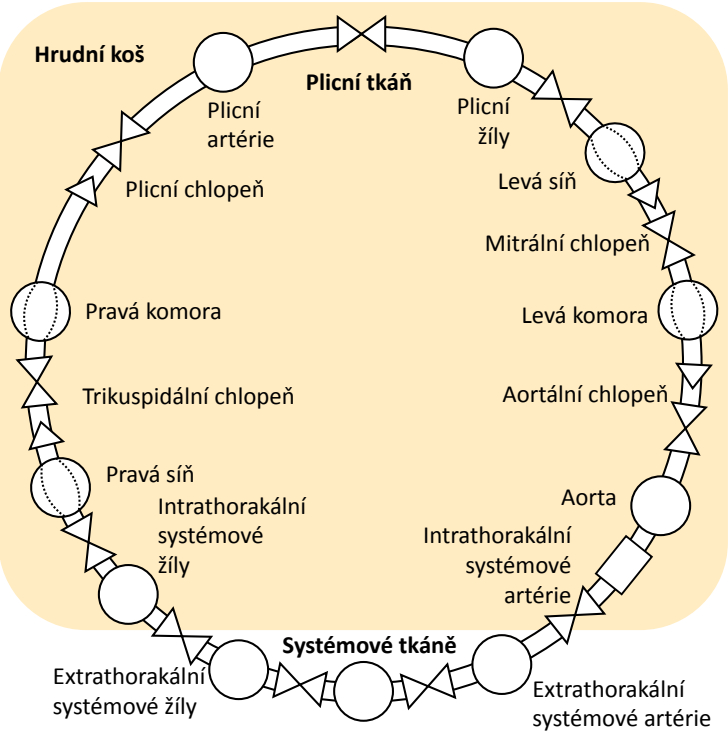
Matematický model, který je teoretickým podkladem funkčnosti simulátoru HPS, není veřejně dostupný. Nicméně nedávno van Meurs, jeden z autorů modelu použitého v simulátoru HPS (dříve prodávaného pod komerčním názvem METI), vydal knihu, v níž strukturu modelu částečně poodhalil [44]. Některé doplňující hodnoty, nutné pro realizaci modelu subsystému hemodynamiky, který v Modelice postavíme, jsme vzali z jiných publikací [7, 14, 39]. Na následujících stránkách si ukážeme, že postavení funkčního simulačního modelu v jazyce Modelica snadné a rychlé.

3.1 Tvorba konceptuálního modelu

Modelování vždy začíná tvorbou konceptuálního modelu. Při tvorbě konceptuálního modelu si nejprve dekomponujeme systém hemodynamiky na jednotlivé komponenty a popíšeme jejich vlastnosti. V modelu hemodynamiky si vystačíme s několika opakujícími se komponentami. Obrázek 9, vytvořený podle obrázku z van Meursovy knihy [44], zobrazuje komponenty, které budou při modelování hemodynamiky zapotřebí. Zároveň je u každé komponenty specifikováno její chování pomocí jednoduchých matematických vztahů.

Především budeme potřebovat komponentu ***odporu***, jejíž chování popisuje Ohmův zákon. Cévy jsou pružné trubice, jejichž chování je možné popsat pomocí ***elastického kompartmentu.*** V tomto kompartmentu rozdíl tlaků mezi vnitřní

a vnější stranou cévy (tzv. transmurální tlak) závisí na náplni cévy (viz Obrázek 10). Ve skutečnosti je závislost tlaku na náplni cévy nelineární, můžeme si ji však pro jednoduchost linearizovat. Čím je céva poddajnější, tím má směrnice křivky, zobrazující závislost transmurálního tlaku na náplni cévy, menší sklon. Krom toho, při postupném plnění cévy je transmurální tlak zpočátku nulový a stoupá teprve od překročení určitého reziduálního objemu („unstressed volume") v závislosti na poddajnosti. Další komponentou je ***induktor***, kterým modelujeme setrvačnost krve – změna rychlosti proudění je úměrná rozdílu tlaků a nepřímo úměrná induktanci. Dalším prvkem je ***chlopeň***, která (v elektrické analogii) funguje jako dioda propojená s odporem. Uvažovaným prvkem je ***zdroj konstantního tlaku (kompresor)***, generující konstantní tlak nezávisle na průtoku. Tuto komponentu ve vlastním modelu hemodynamiky nebudeme potřebovat, dá se však využít při modifikaci modelu (např. při modelování externího čerpadla namísto srdeční komory). Konečně, posledním prvkem je ***elastický kompartment s proměnnou poddajností***, který funguje obdobně jako elastický kompartment s tím rozdílem, že poddajnost jeho stěny se v čase mění. Pomocí elastického kompartmentu s časově proměnnou



Obrázek 11 — Propojení komponet v modelu hemodynamiky podle Meurse (Meurs, 2011).

poddajností je modelována srdeční pumpa (tj. síně a komory).

Propojení komponent znázorňuje Obrázek 11, rovněž vytvořený podle obrázku z monografie van Meurse [44]. V modelu se rozlišují cévy, které jsou uvnitř hrudníku (intrathorakální) a mimo hrudník (extrathorakální). Důvod tohoto dělení spočívá v tom, že na intrathorakální cévy působí zvnějšku tlak uvnitř hrudníku (který je negativní). Nitrohrudní tlak se ale mění s dýcháním – při nádechu se snižuje, při výdechu se zvyšuje. Může být také ovlivněn při umělé plicní ventilaci, kdy se plíce „nafukují" zvnějšku aparátem umělé plicní ventilace, což má pak vliv na hemodynamiku. Proto pro propojení subsystému hemodynamiky se subsystémem respirace je nutné rozlišovat elastické kompartmenty cév, které jsou vně a uvnitř hrudníku.

Levá komora je modelována jako elastický kompartment s časově proměnnou poddajností. Krev přes aortální chlopeň proudí do nitrohrudní aorty, modelované jako elastický kompartment. Dále krev proudí do induktoru, který modeluje setrvačné vlastnosti proudící krve. Pak přes odpor krev proudí do extrathorakálních systémových artérií a z nich pak přes další odpor do elastického kompartmentu systémových tkáňových cév (arteriol, kapilár a přiléhajících venul). Z něj pak krev proudí přes odpor venul do elastického kompartmentu extrathorakálních žil a odtud pak přes žilní odpor do elastického kompartmentu nitrohrudních systémových žil. Odtud se krev dostává, přes odpor centrálních žil, do pravé síně, modelované jako elastický kompartment s časově proměnnou poddajností. Z pravé síně krev proudí přes trojcípou (trikuspidální) chlopeň do pravé komory (modelované opět jako elastický kompartment s časově proměnnou poddajností). Z pravého srdce pak krev proudí přes plicní chlopeň do elastického kompartmentu plicních artérií. Z nich se krev přes odpor cév v plicních tkáních dostává do elastického kompartmentu plicních žil a odtud pak přes odpor plicních žil do levé síně, modelované jako elastický kompartment s časově proměnnou poddajností. Odtud se pak krev dostává přes mitrální chlopeň zpět do levé komory.

3.2 Komponenty a jejich instance

Na začátku implementace modelu v jazyce Modelica vytvoříme balíček (**package) s názvem Haemodynamics a v něm balíček Parts kam budeme ukládat jednotlivé vytvořené komponenty. Modelikový program vypadá zatím jednoduše:**

```
  package Haemodynamics „Hemodynamic model for Human
Patient Simulator"
   package Parts „Necessary components"
   end Parts;
  end Haemodynamics;
```

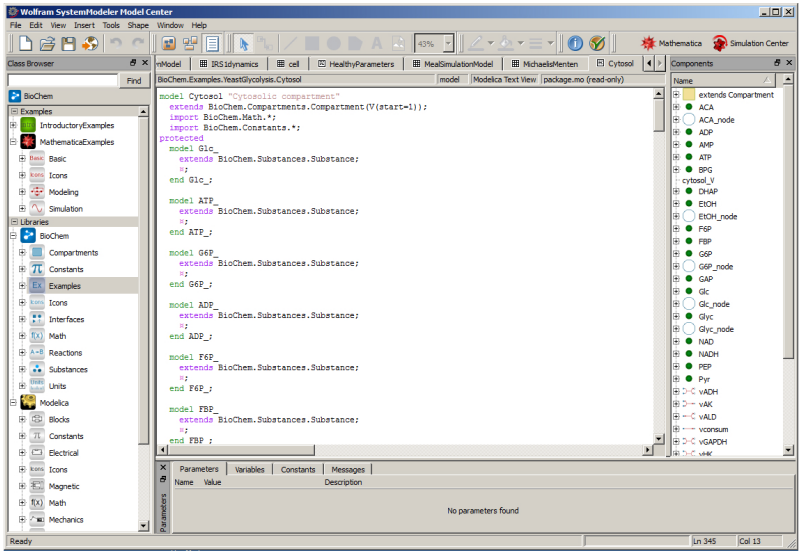
Komponenta je pojímána jako třída – komponentou je odpor či elastický kompartment. Instancí je konkrétní realizace této třídy – např. odpory v plicních cévách, žilní plicní odpor, odpor v mitrální chlopni, odpor v aortální chlopni,

odpor systémových artérií atd. Instancí jsou elastické kompartmenty aorty, extrathorakálních artérií, perifeních cév v tkáních, elastický kompartment extrathorakálních a intrathorakálních žil atd. V grafickém editoru vývojového nástroje jazyka Modelica instanci komponenty vytvoříme přenesením ikonky komponenty z knihovny na editační plochu.

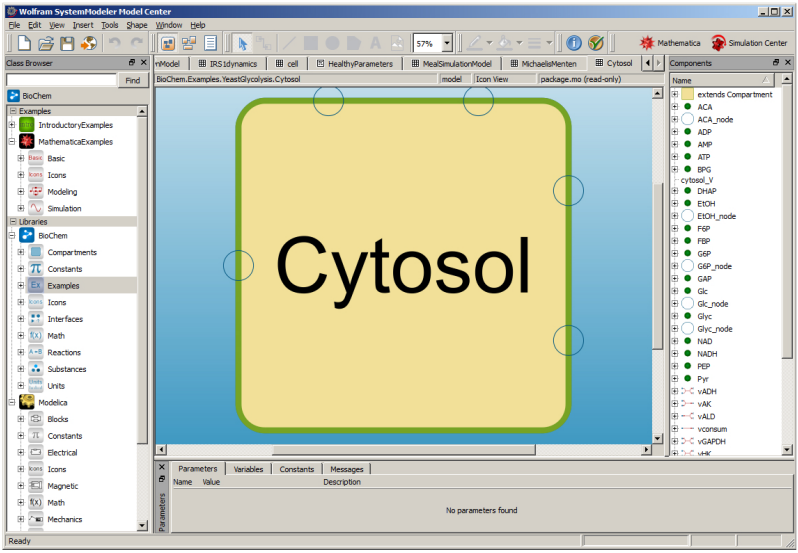
Tvorba modelu v Modelice spočívá v tvorbě nových komponent (tříd), a ve využívání instancí již hotových komponent.

Pro tvorbu uživatelského vzhledu ikonky nově vytvářené komponenty můžeme využít grafický editor pro tvorbu ikonek (viz Obrázek 12). Propojováním instancí již vytvořených komponent můžeme vizuálním způsobem vytvářet novou komponentu a využívat přitom grafický editor (viz Obrázek 13).

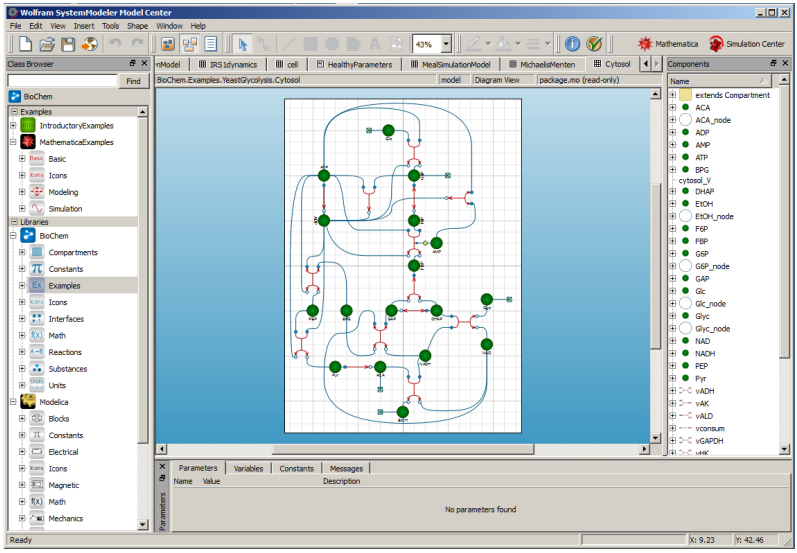
Protože Modelica je objektově orientovaný jazyk – můžeme využívat dědičnost a komponenty dále rozšiřovat. Na rozdíl od ostatních objektově orientovaných jazyků, třídy v Modelice mohou obsahovat rovnice. Ty vytváříme v textovém editoru (viz Obrázek 14). Propojené instance komponent dohromady vytvářejí soustavu algebrodiferenciálních rovnic. Základním požadavkem je, aby počet proměnných a počet rovnic v této soustavě byl vyrovnaný – rovnic by mělo být stejně jako proměnných. To se při tvorbě modelu neustále kontroluje. Proto je vhodné, aby každá komponenta obsahovala stejný počet proměnných a stejný počet rovnic. Pokud tomu tak není, tak taková komponenta nemůže vytvářet své instance – o takových komponentách se hovoří jako o částečných, parciálních



Obrázek 12 — Textový editor ve vývojovém prostředí Wolfram SystemModeler. Modelikový model je ale vytvářen jako text. Při začleňování instancí komponent do modelu a jejich propojování v grafickém editoru je automaticky generován zdrojový kód jazyka Modelica, který můžeme vidět v textovém editoru. Textový editor využíváme hlavně při vytváření nových komponent a definování příslušných rovnic.



Obrázek 13 — Grafický editor pro tvorbu vizuální podoby vnějšího ikonkového vzhledu nových modelikových komponent ve vývojovém prostředí Wolfram System Modeler.



Obrázek 14 — Grafický editor pro tvorbu vnitřní struktury nových komponent ve vývojovém prostředí Wolfram SystemModeler. Zde se s výhodou využívá vizuální modelování propojováním instancí jednotlivých komponent. Díky akauzálnímu propojování komponent struktura modelu vyjadřuje strukturu modelované reality a nikoli jen postup výpočtu. V daném příkladě je zobrazen model glykolýzy kvasinek a příslušné spojnice vyjadřují metabolické dráhy.

třídách. Parciální třídy nemohou vytvářet instance, ale mohou být podkladem (předkem) pro vytvoření řady dalších odvozených komponent (potomků), které budou sdílet stejné rovnice či proměnné, a zároveň komponentu rozšíří o potřebný počet rovnic či proměnných.

Vlastní matematické vztahy v jednotlivých komponentách jsou vcelku triviální.

Vstupem modelu mohou být změny odporů a poddajností jednotlivých částí cévního řečiště, objem cirkulující krve a samozřejmě čerpací činnost srdce, který bude modelována periodickou změnou poddajnosti komor a síní. V systole se poddajnost elastického kompartmentu, reprezentujícího srdeční komoru, sníží (tuhost stěn se zvýší) a tím se zvýší tlak uvnitř. To vede k okamžitému uzavření mitrální a trikuspidální chlopně. Pokud tlak přesáhne hodnotu protitlaku na druhé straně chlopně, chlopeň se otevře a krev je vypuzena do aorty a do plicní artérie. V diastole se poddajnost zvýší (tuhost sníží), tlak poklesne pod úroveň tlaku v aortě a v plicní artérii, aortální a plicní chlopně se uzavřou. Po dalším poklesu tlaku, když tlak v komorách poklesne pod úroveň tlaku v síních, otevře se mitrální a trikuspidální chlopeň a komory se začnou plnit. K plnění komor také do určité míry může přispět i systola síní.

V modelu bude také potřeba navrhnout bloky, které periodicky řídí změnu poddajnosti (resp. změnu tuhosti) elastických kompartmentů síní a komor.

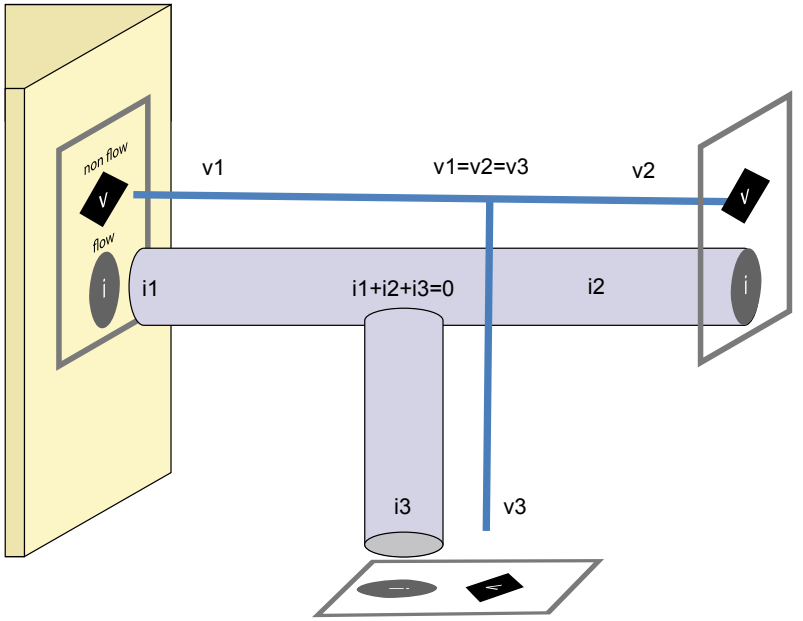
3.3 Konektory

Každá vytvářená komponenta bude potřebovat konektor, kterým se může propojit s další komponentou. Proto si v Modelice nejprve vytvoříme konektory,

Konektory jsou instance speciálních konektorových tříd, kde se definují proměnné, používané pro propojení. Propojovat se mohou prvky pomocí konektorů, které jsou instancemi stejných konektorových tříd („propojovací zásuvky" musí být stejného typu).

Konektorové třídy definují způsob komunikace jednotlivých modelicových komponent mezi sebou. Obrazně řečeno, definicí konektorových tříd definujeme typy „zásuvek". V konektorech se definují jednotlivé proměnné, kterými bude konektor propojovat příslušnou komponentu s okolím.

Pro každou proměnnou v konektoru se definuje, zda představuje nějaký tok – (proměnná je pak označena atributem „flow"), či nikoli (tzv. „non-flow" proměnné). Toto rozlišení je důležité pro správnou interpretaci propojení jednotlivých komponent (instancí tříd prvků) přes příslušné konektory. U tokových proměnných je zřejmé, že musí být zajištěno, aby se nikde v propojení příslušná entita (jejíž tok příslušná proměnná charakterizuje) neztrácela nebo neakumulovala. Proto součet hodnot všech propojených veličin s atributem „flow" musí být nulový (jako podle Kirchhoffova zákona v elektrické doméně). U netokových proměnných propojení definuje, že jejich hodnoty u všech propojených konektorů jsou sdílené a musí tedy být stejné. Propojením instancí jednotlivých elementárních prvků prostřednictvím konektorů se vyjádří požadavek nulovosti algebraického součtu hodnot propojených tokových proměnných a požadavek rovnosti hodnot propojených



Obrázek 14 — Propojení pomocí konektorů. Pokud v modelikovém grafickém editoru propojíme instance modelikových tříd zároveň se tím na pozadí generují rovnice. Hodnoty tokových proměnných (typu flow - v daném případě proměnné i) budou automaticky nastaveny tak, aby algebraický součet hodnot všech propojených toků byl nulový. Hodnoty ostatních (non-flow) proměnných (v daném příkladě hodnoty v) budou na všech propojených konektorech nastaveny na stejnou hodnotu.

netokových proměnných (obr. toky – Obrázek 14).

Každá modelicová třída může mít svoji grafickou reprezentaci – ta je důležitá zejména pro zobrazení propojení instancí, kdy propojováním komponent se vytváří názorná grafická struktura modelu. Proto pro každou třídu v Modelice můžeme definovat i příslušnou ikonu. Tato ikona může být i animovaná.

Model pak v Modelice můžeme vytvářet graficky propojováním instancí jednotlivých prvků, které pomocí myši vybíráme z knihovny a v dialogu pak nastavíme hodnoty příslušných parametrů. Při tomto propojování (jak uvidíme dále), se na pozadí automaticky generuje příslušný textový modelikový příkaz, kterým se v Modelice propojení popisuje.

Začneme tedy s vytvořením konektorové třídy, jejíž instance budou využity pro propojení jednotlivých komponent. Konektorová třída se v Modelice uvozuje klíčovým slovem **connector**. Za tímto klíčovým slovem uvedeme název třídy (zvolíme pro ni název BloodFlowConnector). Za deklarací třídy můžeme v uvozovkách uvést poznámku, která nám později může pomoci pochopit, co jsme chtěli vytvářet. Podstatné je v poznámkách uvádět význam proměnných a jednotky. Častá a zákeřná chyba při modelování je chybné použití jednotek

(místo milimolů, miligramy apod.), proto si na kompatibilitu jednotek musíme dávat pozor. Proud budeme tedy měřit v ml/sec a tlak v torrech. Modelica má speciální prostředky, které nám pomáhají hlídat správné přiřazování jednotek, ale ty zatím, pro jednoduchost, nebudeme používat.

Konektoru bude propojovat dva typy reálných proměnných – průtok a tlak. Průtok je proměnná toku, proto před její deklarací píšeme klíčové slovo **flow**. Způsob deklarace konektorové třídy BloodFlowConnector je zřejmý z následujícího fragmentu programu:

```
 connector BloodFlowConnector „Connector for blood
flow"
  flow Real Q „blood flow in ml/sec";
  Real Pressure „Pressure in torr";
 end BloodFlowConnector;
```

V grafickém editoru vytvoříme pro právě vytvořený konektor ikonku ve tvaru červeného čtverečku, kterou pak budou v grafickém editoru označeny všechny konkrétní instance této třídy. V textovém editoru se při tom zároveň vygeneruje anotace, která popisuje způsob vytvoření grafického symbolu. Anotace je v textovém editoru obvykle sbalená do jednoho znaku (na kliknutí se dá rozbalit, a můžeme vidět příkazy, které popisují vykreslení ikonky). Ve výpisech programů v této knize anotace grafických komponent nebudeme pro zjednodušení uvádět.

Vytvoříme si ještě další dva konektory pro vtok a výtok krve do nebo z komponenty (nazveme je **BloodFlowInflow** a **BloodFlowOutflow**. Jako ikonku vtokového konektoru zvolíme plný červený čtvereček a výtokový konektor znázorníme prázdným červeným čtverečkem. Tím pak v příslušných komponentách na první pohled odlišíme, kde jsme v komponentě označili vtok a kde výtok (a nezapojíme pak třeba srdeční pumpu obráceně). Náš program zatím vypadá velmi jednoduše:

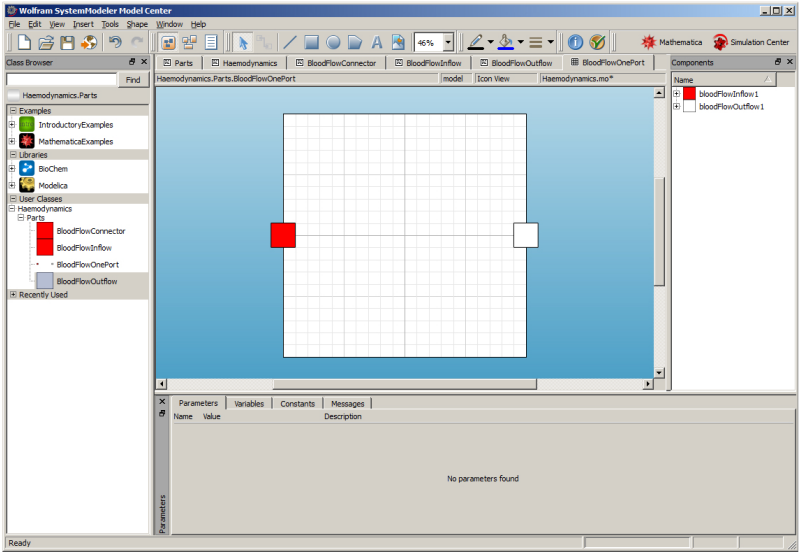
```
 package Haemodynamics „Hemodynamic model for Human
Patient Simulator"

 package Parts „Necessary components"

   connector BloodFlowConnector „Connector for blood
   flow"
    flow Real Q „blood flow in ml/sec";
    Real Pressure „Pressure in torr";
   end BloodFlowConnector;

   connector BloodFlowOutflow „Blood flow outflow"
    flow Real Q „blood flow outflow in ml/sec";
    Real Pressure „Pressure in torr";
   end BloodFlowOutflow;
   connector BloodFlowInflow „Blood flow inflow"
    flow Real Q „blood inflow in ml/sec";
```

```
    Real Pressure „Pressure in torr";
  end BloodFlowInflow;

end Parts;

end Haemodynamics;
```

3.4 Partial Model BloodFlowOnePort

Nyní vytvoříme první skutečnou komponentu – částečnou třídu. Klíčovým slovem Model se v Modelice označuje třída, která, na rozdíl od ostatních objektových jazyků, může obsahovat rovnice. Při vytváření modelové třídy je třeba vždy dbát na to, aby počet rovnic odpovídal počtu proměnných.

V Modelice existuje i dědění, je možné vytvářet další třídy rozšiřováním těch předchozích. Když se podíváme na komponenty, které chceme vytvářet, všimneme si, že některé z nich (odpor, induktor, chlopeň) mají společné vlastnosti. Mají bránu, kterou vtéká a vytéká krev (tedy budou obsahovat vtokový a výtokový konektor) a krev se v nich nehromadí, tedy algebraický součet hodnot toku na vtokovém a výtokovém konektoru bude nulový. Proto je vhodné si ulehčit práci tím, že tyto komponenty vytvoříme ze společného předka, který bude mít vtokový a výtokový konektor, algebraická suma toků bude nulová, a krom toho bude asi vhodné definovat proměnné, které



Obrázek 15 — Vytváříme partial Model BloodFlowOnePort v prostředí SystemModeller. Na levé straně je je okno s ikonkami komponent. Přetažením komponent BloodFlowInflow a BloodFlowOutflow z tohoto okna na editační plochu vytváření nové ikonky se aoutomaticky vytvoří instance těchto konektorových tříd v nově definované třídě BloodFlowOnePort.

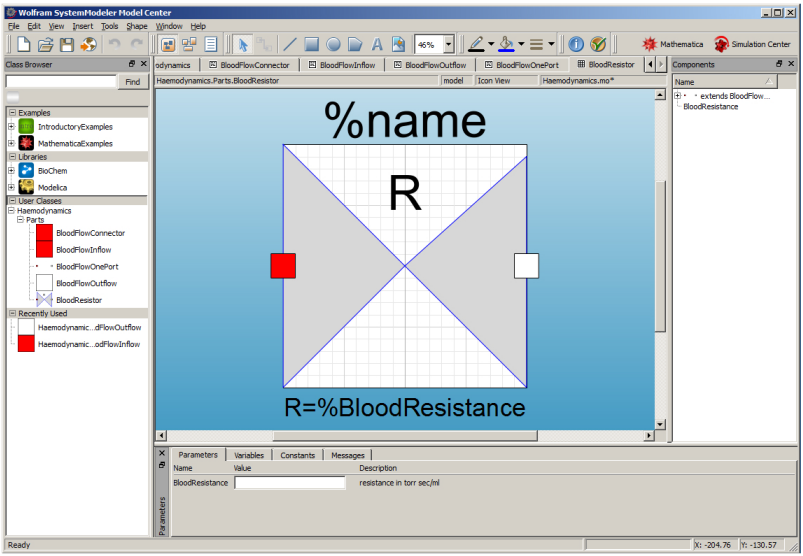
různým způsobem zužitkují všichni potomci. Těmito proměnnými bude hodnota tlakového spádu a hodnota průtoku krve komponentou. Tlakový spád vypočítáme z rozdílu hodnot tlaků ve vtokovém a výtokovém konektoru a průtok komponentou se bude rovnat hodnotě toku na vtokovém konektoru.

Tohoto předka nazveme BloodFlowOnePort. Tato třída bude jenom předkem pro další třídy a nebude mít vlastní samostatné instance (ani by je nemohla mít, protože počet rovnic je o jednu menší než počet proměnných). Proto před klíčové slovo Model, kterým se v Modelice označuje specifická třída obsahující rovnice, předřadíme klíčové slovo partial.

V grafickém editoru vytvoříme ikonku, kam přetažením z okna komponent přetáhneme komponenty BloodFlowInflow a BloodFlowOutflow, a v nově vytvářené komponentě tak vytvoříme jejich instanci (viz Obrázek 15). Zároveň se na pozadí vygenerovala část zdrojového textu, která definuje deklarace instancí těchto komponent v nově vytvářené třídě:

Přejmenujeme vygenerovaná jména bloodFlowOutflow1 a blloodFlowInflow1 na Inflow a Outflow, deklarujeme nové reálné proměnné PressureDrop a BloodFlow a v sekci equation napíšeme příslušné rovnice:

```
partial model BloodFlowOnePort
    Haemodynamics.Parts.BloodFlowOutflow Inflow;
```



Obrázek 16 — Vytváříme model odporu. V dialogovém okně se už objevila nabídka zadání hodnoty parametru BloodResistance, který jsme v této třídě definovali. V ikonce třídy jsme také definovali textové proměnné - ty jsou uvedeny se znakem % před jménem. To může být výhodné, když instance této komponenty využíváme při tvorbě nových komponent (můžeme mít několik instancí odporů, a je vhodné mít u ikonky vypsaný i název instance). Název instance vypíše %name, %BloodResistance vypíše hodnotu kterou má parametr BloodResistance.

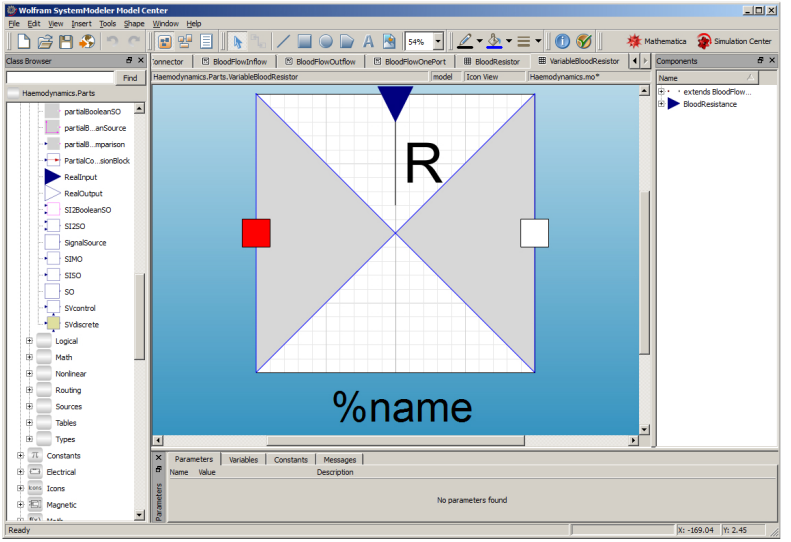
```
    Haemodynamics.Parts.BloodFlowInflow Outflow;
    Real PressureDrop;
    Real BloodFlow;
equation
    PressureDrop=Inflow.Pressure - Outflow.Pressure;
    Inflow.Q + Outflow.Q=0;
    BloodFlow=Inflow.Q;
end BloodFlowOnePort;
```

3.5 Komponenty odporů

Nyní vytvoříme třídu BloodResistor která je komponentou odporů v cévním řečišti jako rozšíření třídy BloodFlowOnePort (která je jejím předkem). V ikonkce třídy BloodResistor se automaticky objeví dvojice vtokových a výtokových konektorů – protože ty jsou již definovány pro třídu BloodFlowOnePort.

Ve třídě deklarujeme reálný parametr BloodResistance. Parametru přiřazujeme hodnoty na začátku simulace. Můžeme mu také přiřadit, jako v daném případě, implicitní hodnotu. Tu v deklaraci uvádíme v závorkách. V sekci rovnic pak deklarujeme jednu rovnici vyjadřující Ohmův zákon:

```
model BloodResistor
   parameter Real BloodResistance(start=1) „resistance
```



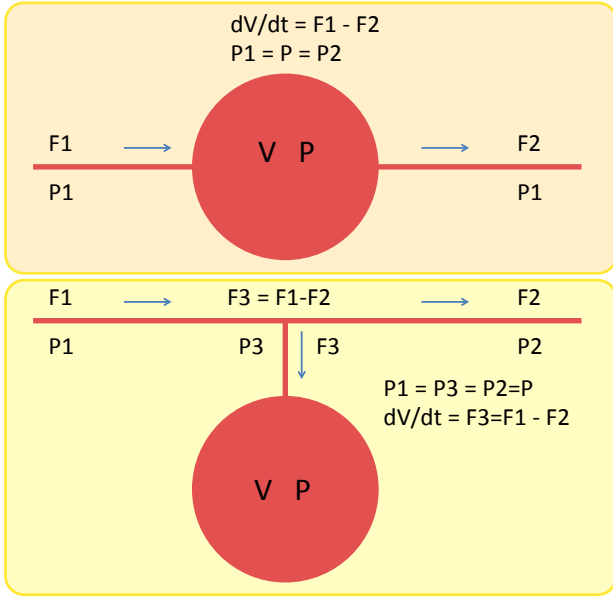
Obrázek 17 — Komponentu variabilního odporu, kde hodnota rezistence je zadávána zvnějšku, vytváříme s využitím komponenty RealInput ze standardní modelikové knihovny. Příslušnou ikonku této komponenty přetáhneme myší na editační plochu grafického editoru a v modelikovém programu se vytvoří její instance.

```
    in torr sec/ml";
    extends Haemodynamics.Parts.BloodFlowOnePort;
equation
    PressureDrop=BloodFlow*BloodResistance;
    end BloodResistor;
```

V grafickém editoru vytvoříme ještě této nové komponentě ikonku a doplníme ji textovými proměnnými, které budou vypisovat názvy instancí této komponenty a hodnoty parametru BloodResistance (viz Obrázek 16).

Hodnota odporu instancí této komponenty je zadávána parametrem, což znamená, že během simulace se tato hodnota nemůže měnit. Je proto užitečné vytvořit další komponentu, kde je možno zadávat hodnotu rezistence jejím instancím zvnějšku, pomocí reálného vstupu. Vytvoříme proto novou komponentu s řídícím signálovým vstupem zvnějšku. Pro vytvoření tohoto vstupu využijeme standardní modelikovou knihovnu (Obrázek 17). Zdrojový kód komponenty VariableBloodResistor vypadá takto:

```
model VariableBloodResistor
    extends Haemodynamics.Parts.BloodFlowOnePort;
    Modelica.Blocks.Interfaces.RealInput
    BloodResistance „in torr sec/ml";
```



Obrázek 18 — Na chování elastického kompartmentu (s objemem V a tlakem P) nemá žádný vliv, zda je propojen jendím nebo dvěma konektory. Proto k napojení elastického kompartmentu stačí jeden akauzální konektor.

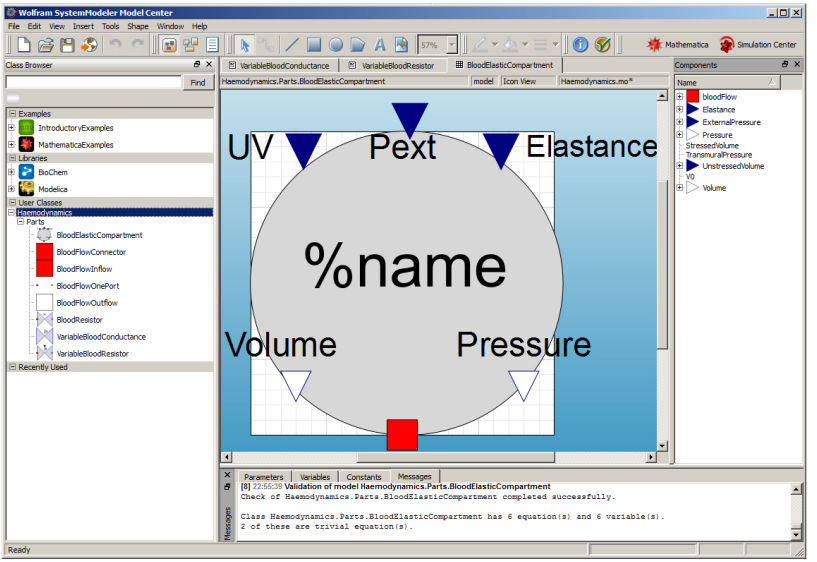
```
equation
   PressureDrop=BloodFlow*BloodResistance;
   end VariableBloodResistor;
```

Když bychom např. chtěli propojit komponenty tak, aby normálně mezi nimi byl tok zcela zastaven, můžeme nastavit hodnotu odporu na velmi vysokou hodnotu, nebo místo rezistence využít vodivost – nulová vodivost znamená žádný tok.

Může se nám proto hodit také komponenta, kde užíváme místo resistence vodivost. Vizuálně komponenta vypadá stejně, jen místo písmena R je na ikonce komponenty písmenko G. V textové podobě komponenta VariableBloodConductance vypadá následovně:

```
model VariableBloodConductance
   extends Haemodynamics.Parts.BloodFlowOnePort;
   Modelica.Blocks.Interfaces.RealInput
   BloodConductance „in torr ml/sec“;
equation
    PressureDrop*BloodConductance=BloodFlow;
   end VariableBloodConductance;
```

3.6 Elastický kompartment



Obrázek 19 — Vstupem pro komponentu elastického kompartmentu je reziduální objem (unstressed volume) - UV, vnější tlak - Pext a hodnota elastance (tuhosti) - Elastance (elastance je obrácenou hodnotou poddajnosti). Signálovým výstupem jsou hodnoty objemu a tlaku. Elastický kompartment není průtočný, k napojení stačí jeden akauzální konektor.

Elastický kompartment je pružný vak, který může mít jeden nebo dva konektory. Když ke dvěma konektorům připojíme dvoukonektorový elastický kompartment, je výsledek úplně stejný, jako když k nim pomocí „T" propojení připojíme jednokonektorový kompartment (viz Obrázek 18). Proto si vystačíme s jedním konektorem.

Pro realizaci řídících vstupů (reziduálního objemu - „unstressed volme", hodnoty tlaku na vnější straně kompartmentu a hodnoty Elastance) a signálových informačních výstupů (o tlaku a objemu) využijeme opět standardní modelikovou knihovnu, odkud vezmeme instance komponent RealInput a RealOutput (Obrázek 19)

Místo poddajnosti budeme pracovat s elastancí charakterizující tuhost (elastance = 1/poddajnost) .

Zdrojový text komponenty BloodElasticCompartment vypadá následovně:

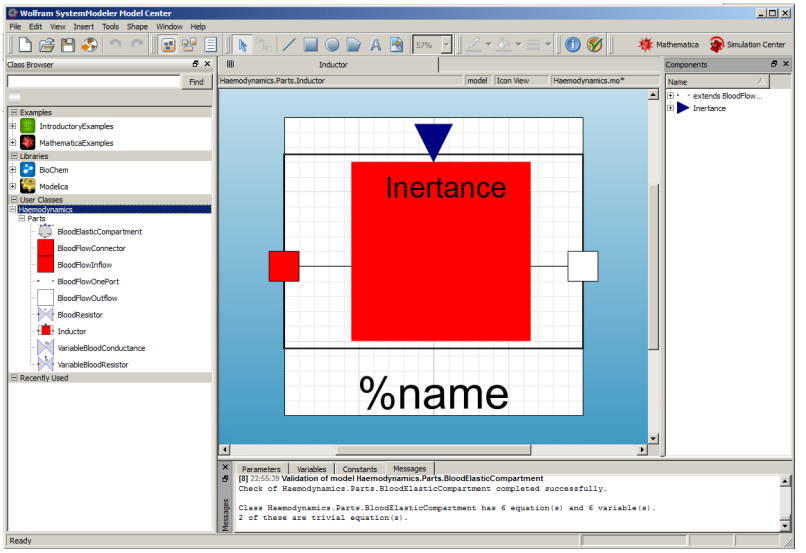
```
model BloodElasticCompartment „Elastic compartment with
unstressed volume"
 Modelica.Blocks.Interfaces.RealInput Elastance „\"in torr/
 ml\"";
 Modelica.Blocks.Interfaces.RealOutput Volume(start=V0);
 Modelica.Blocks.Interfaces.RealInput ExternalPressure „\"in
 torr\"";
 Modelica.Blocks.Interfaces.RealInput UnstressedVolume „in
 ml";
 parameter Real V0=1 „initial volume in ml";
 Real StressedVolume;
 Real TransmuralPressure;
 Modelica.Blocks.Interfaces.RealOutput Pressure „Blood
 pressure in torr";
 Haemodynamics.Parts.BloodFlowConnector bloodFlow;
equation
 bloodFlow.Pressure=Pressure;
 TransmuralPressure=Pressure - ExternalPressure;
 der(Volume)=bloodFlow.Q;
 StressedVolume=Volume - UnstressedVolume;
 if StressedVolume > 0 then
 TransmuralPressure=Elastance*StressedVolume;
 else
 TransmuralPressure=0;
 end if;
end BloodElasticCompartment;
```

3.7 Induktor

Další komponentou je induktor, charakterizující setrvačnost krve (Obrázek 20).

Induktor je průtočná komponenta se dvěma konektory a proto je rozšířením parciální komponenty BloodFlowOnePort. Řídící vstup je jediný – hodnota inertance, přiváděná prostřednictvím instance komponenty RealInput ze standardní modelikové knihovny.

Zdrojový kód komponenty je jednoduchý:



Obrázek 20 — Komponentou Inductor modelujeme setrvačné vlastnosti krve. Vstupem je hodnota inertance.

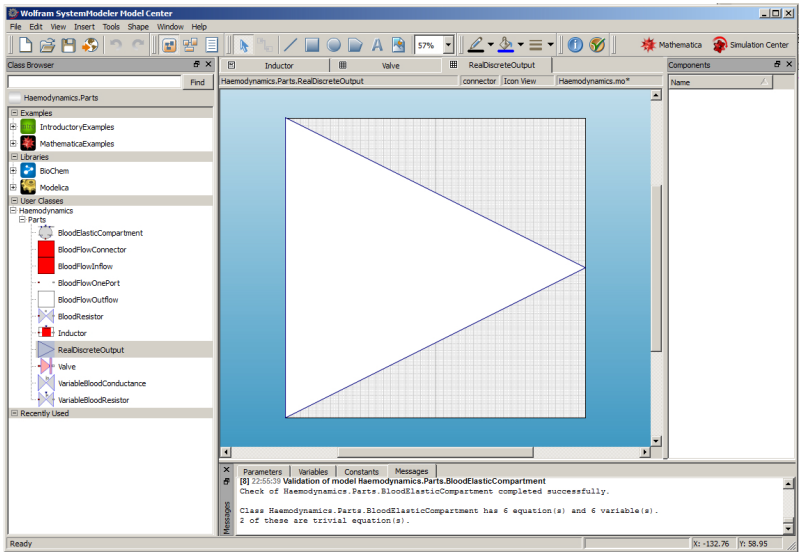
```
model Inductor
  extends Haemodynamics.Parts.BloodFlowOnePort;
  Modelica.Blocks.Interfaces.RealInput Inertance „in torr *
  sec^2/ml“;
equation
  PressureDrop=der(BloodFlow)*Inertance;
end Inductor;
```

3.8 Ventil

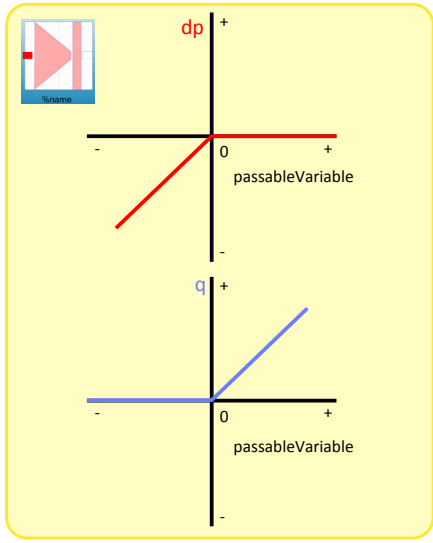
Další komponentou, kterou využijeme při modelování srdečních chlopní je ideální ventil (Obrázek 21) modelovaný komponentou Valve. Komponenta je průtočná a je proto opět odvozena od parciální třídy BloodFlowOnePort. Zvenčí nepřijímá žádný vstup a nevysílá žádný signálový výstup. Zajímavé je však její chování. Funguje jako ideální ventil – pokud je rozdíl tlaků dp mezi přítokovým a odtokovým konektorem záporný, pak se ventil zavírá a ventilem nic neteče – proud q, protékající ventilem je nulový. Jestliže tlakový gradient není záporný, pak ventilem protéká tekutina a tlakový gradient je vždy nulový.

Pro naprogramování této komponenty využijeme modelikový podmíněný příkaz if..then..else. Uvědomme si však, že tento příkaz funguje jinak, než při řešení algoritmu (kdy se v určitém okamžiku testuje hodnota nějaké proměnné a na základě výsledků se provede rozhodnutí). V Modelice podmíněným příkazem popisujeme podmínky, kdy platí, nebo neplatí určité sady rovnic, a kdy dojde k přepnutí z jedné sady rovnic na druhou.

Popsat chování ideálního ventilu pomocí rovnic není úplně triviální. Problém



Obrázek 21 — Komponenta ideálního ventilu.



Obrázek 22 — Parametrická charakteristika funkce ideálního ventilu - hodnoty tlakového rozdílu (dp) a průtoku (q) závisejí na parametrické proměnné passageValue. Pokud jje hodnota tohoto parametru kladná, pak se rovná průtoku, zatímco tlakový rozdíl mezi vstupním a výstupním konektorem je nulový. Pokud je hodnota parametrické proměnné záporná, pak se bude rovnat zápornému rozdílu tlaků mezi přítokovým a odtokovým konektorem.

tkví v tom, že tlakový gradient závisí na průtoku (pokud je ventil zavřený, tak změny tlaku na jedné straně se nemohou přes ventil přenášet na druhou stranu). Na druhé straně průtok závisí na tlakovém gradientu (je-li záporný, průtok se uzavírá).

Řešení je zavedení dalšího parametru passabeVariable a vyjádření chování ideálního ventilu pomocí parametrických rovnic (viz Obrázek 22).

Zdrojový text komponenty ideálního ventilu je následující:

```
model Valve
  Haemodynamics.Parts.BloodFlowInflow bloodFlowInflow;
  Haemodynamics.Parts.BloodFlowOutflow bloodFlowOutflow;
  Real q;
  Real dp;
  Boolean open(start=true);
  Real passableVariable;
equation
  bloodFlowInflow.Q + bloodFlowOutflow.Q=0;
  q=bloodFlowInflow.Q;
  dp=bloodFlowInflow.Pressure - bloodFlowOutflow.Pressure;
  open=passableVariable > 0;
  if open then
   dp=0;
   q=passableVariable;
  else
   dp=passableVariable;
   q=0;
  end if;
end Valve;
```

3.9 Délky srdečních intervalů

Nyní se začneme věnovat řídícím komponentám. Nejprve si vytvoříme komponentu počítající délku srdečních intervalů, tj. délku systoly síní, délku atrioventrikulárního intervalu – tj. doby, než se impuls pro zahájení depolarizace komor přenese převodním systémem srdce ze síní do komor, délku komorové systoly a čas zahájení srdečního cyklu.

Tyto intervaly se počítají pro každý srdeční stah zvlášť a jsou to diskrétní reálné proměnné, jejichž hodnoty se mění skokově (při výpočtu těchto intervalů během každého srdečního cyklu).

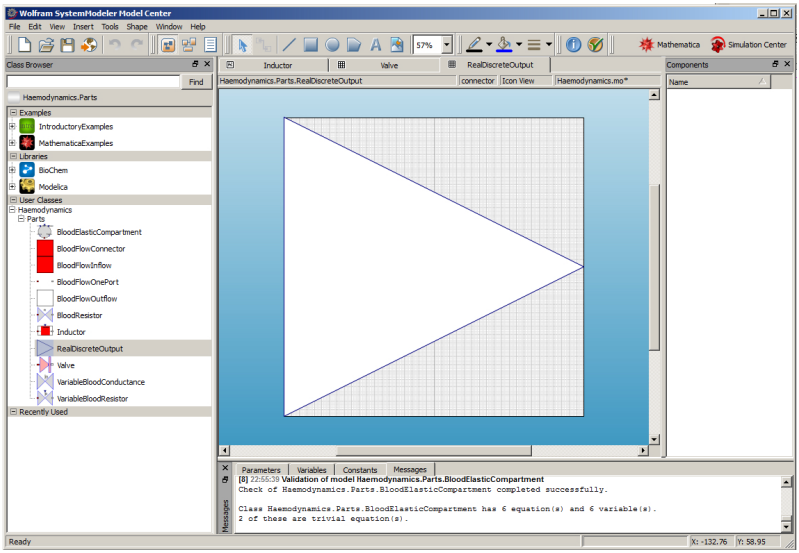
Pro pohodlnost si vytvoříme vlastní konektorovou komponentu pro výstup diskrétní reálné proměnné (Obrázek 23).

Její zdrojový kód má minimální velikost:

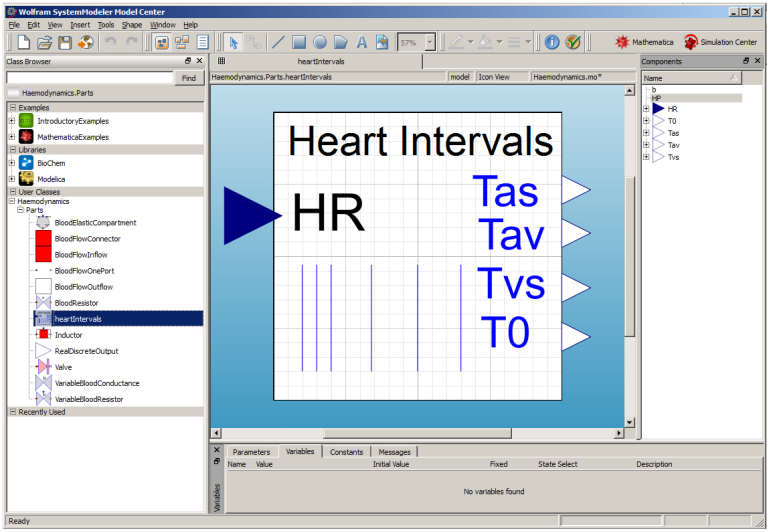
```
connector RealDiscreteOutput=discrete output Real
„'output Real' as connector";
```

Tento konektor hned využijeme při modelování komponenty heartIntervals počítající srdeční intervaly. Jediným vstupem do této komponenty je srdeční frekvence (heart rate) – hodnota je přiváděná instanci komponenty RealInput s názvem HR (Obrázek 24).

Zdrojový text komponenty ozřejmuje způsob počítání těchto intervalů.



Obrázek 23 — Konektorová komponenta RealDiscreteOutput pro výstup reálné diskrétní proměnné.



Obrázek 24 — Komponenta počítá délky srdečních intervalů, které jsou podkladem pro výpočet periodických změn elastance srdečních síní a komor.

Funkce „pre“ vrací předchozí spočítanou hodnotu dané diskrétní proměnné – v případě pre(HP) délku naposledy spočítaného, tedy stávajícího srdečního cyklu, a funkce pre(T0) vrací čas, kdy začala stávající srdeční cyklus. Jakmile bude délka spočítaného srdečního cyklu překročena – hodnota proměnné b se změní na true, budou malý okamžik platné rovnice uvnitř příkazu when. Na rozdíl od if, kde v závislosti na platností podmínek se přepínají soustavy rovnic, které platí po celou dobu, kdy je podmínka splněna, budou rovnice uvnitř klauzule when platné jen jeden „nekonečně malý“ okamžik, v němž se ale nastaví další hodnota diskrétní proměnné T0 a HP.

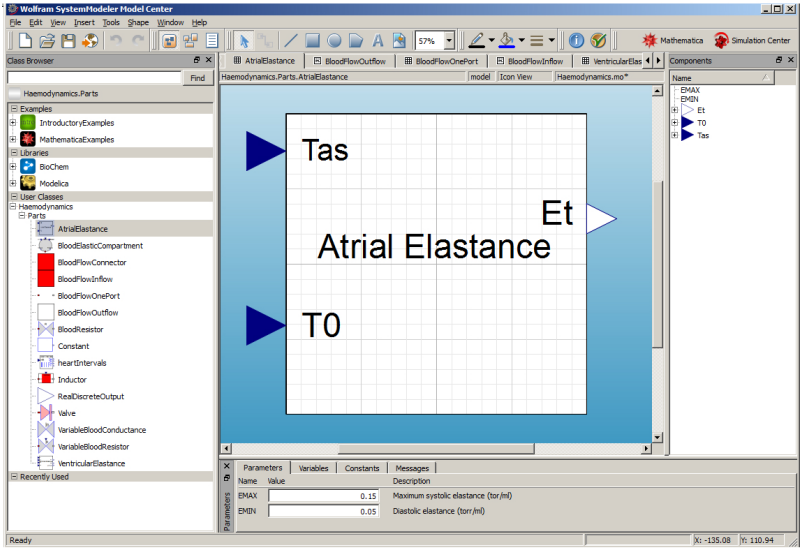
```
model heartIntervals
  Modelica.Blocks.Interfaces.RealInput HR;
  Haemodynamics.Parts.RealDiscreteOutput Tas „duration of
 atrial systole“;
  Haemodynamics.Parts.RealDiscreteOutput Tav
 „atrioventricular delay“;
  Haemodynamics.Parts.RealDiscreteOutput Tvs „duration of
 ventricular systole“;
  Haemodynamics.Parts.RealDiscreteOutput T0 „start time of
 cardiac cycle in sec“;
  discrete Real HP(start=0) „heart period - duration of
 cardiac cycle in sec“;
  Boolean b(start=false);
equation
  b=time - pre(T0) >= pre(HP);
  when b then
  T0=time;
  HP=60/HR;
  Tas=0.03 + 0.09*HP;
  Tav=0.01;
  Tvs=0.16 + 0.2*HP;
  end when;
end heartIntervals;
```

3.10 Elastance síní

V komponente AtrialElastance (Obrázek 25) se na základě délky srdečních atriální systoly (Tas) a času začátlu srdečního cyklu (T0) počítá hodnota elastance (tuhosti) srdečních síní (Et).

Způsob výpočtu je zřejmý ze zdrojového kódu implementace komponenty:

```
model AtrialElastance
 Modelica.Blocks.Interfaces.RealInput Tas „duration of
 atrial systole“;
 Modelica.Blocks.Interfaces.RealOutput Et „elasticity (torr/
 ml)“ ;
 Modelica.Blocks.Interfaces.RealInput T0 „time of start of
 cardiac cycle“;
 parameter Real EMIN=0.05 „Diastolic elastance (torr/ml)“;
 parameter Real EMAX=0.15 „Maximum systolic elastance (tor/
 ml)“;
equation
```



Obrázek 25 — Komponenta počítá periodickou zmenu elastance srdečních síní. Jako parametr se zadává maximální systolická elastance a diastolická elastance. Tímto způsobem se dá nastavit výkonnost síní a modelovat patologické stavy.

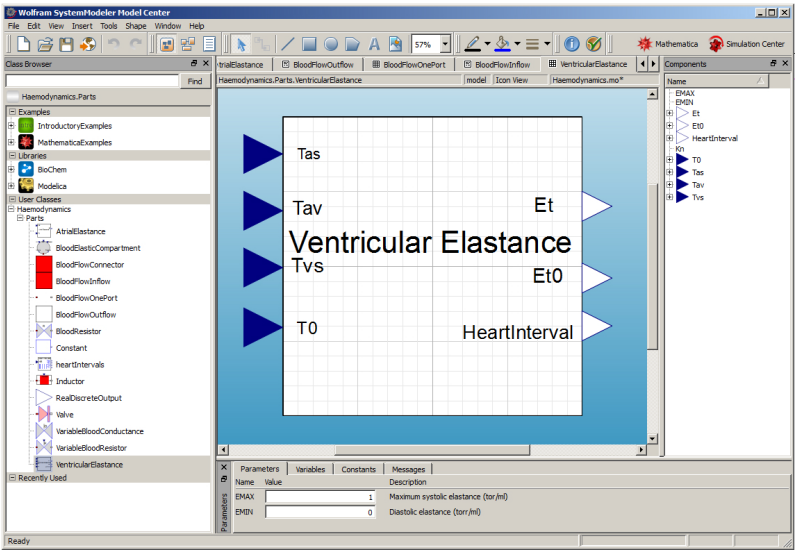
```
  if time - T0 < Tas then
   Et=EMIN + (EMAX - EMIN)*sin(Modelica.Constants.pi*(time -
   T0)/Tas);
  else
    Et=EMIN;
  end if
end AtrialElastance;
```

3.11 Elastance komor

Komponenta VentricularElastance (Obrázek 26) počítá na základě hodnot srdečních intervalů (délky systoly síní Tas, času zahájení srdečního cyklu T0, atrioventrikulárního intervalu Tav, a délky systoly komor Tvs) periodickou změnu elastance srdečních komor.

Způsob výpočtu je zřejmý ze zdrojového kódu komponenty:

```
model VentricularElastance
  Modelica.Blocks.Interfaces.RealInput Tas „duration of
  atrial systole“;
  Modelica.Blocks.Interfaces.RealOutput Et „elasticity (torr/
  ml)“;
  Modelica.Blocks.Interfaces.RealInput T0 „time of start of
  cardiac cycle“;
  Modelica.Blocks.Interfaces.RealInput Tav „atrioventricular
  delay“;
  Modelica.Blocks.Interfaces.RealInput Tvs „duration of
  ventricular systole“;
```

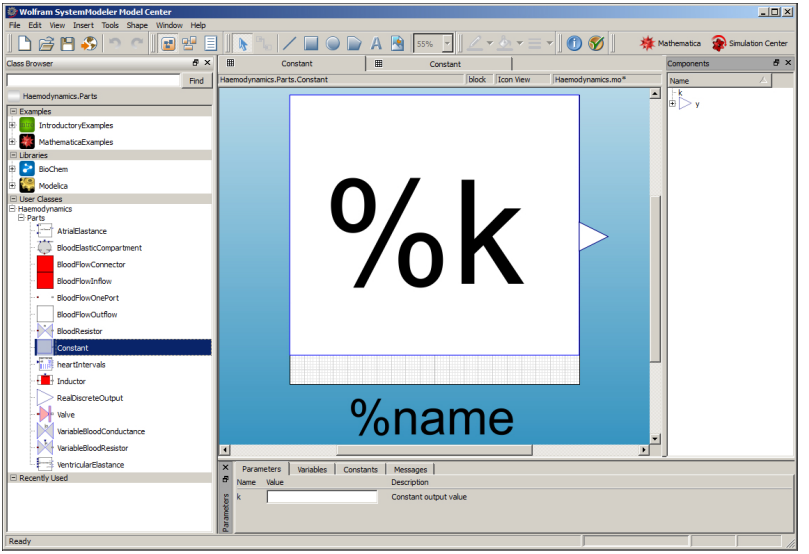


Obrázek 26 — Komponenta počítá periodickou zmenu elastance komor. Jako parametr se zadává maximální systolická elastance a diastolická elastance. Tímto způsobem se dá nastavit výkonnost komory a modelovat také i srdeční selhání.

```
  Modelica.Blocks.Interfaces.RealOutput Et0 „elasticity
  (torr/ml)“;
  Modelica.Blocks.Interfaces.RealOutput HeartInterval
  „elasticity (torr/ml)“;
  constant Real Kn=0.57923032735652;
  parameter Real EMIN=0 „Diastolic elastance (torr/ml)“;
  parameter Real EMAX=1 „Maximum systolic elastance (tor/
  ml)“;
equation
   HeartInterval=time - T0;
   Et=EMIN + (EMAX - EMIN)*Et0;
   if HeartInterval >= Tas + Tav and HeartInterval < Tas +
   Tav + Tvs then
   Et0=(HeartInterval - (Tas + Tav)) / Tvs*
   sin(Modelica.Constants.pi*(HeartInterval-(Tas+Tav))/Tvs)/
   Kn;
  else
   Et0=0;
  end if ;
end VentricularElastance;
```

3.11 Uživatelsky definovaná konstanta

Dosud jsme tvořili nové komponenty. Nyní již budeme model postupně skládat z instancí komponent jak stavebnici z kostiček Lega. Protože v řadě případů budeme potřebovat zadávat do propojovaných komponent nějaké konstanty a komponenta konstanty ze standardní modelikové knihovny nám



Obrázek 27 — Nově vytvořená komponenta konstanty.

nemusí vyhovovat – můžeme si snadno vytvořit vlastní. Chceme vytvořit takovou komponentu, aby se v celém okénku zobrazovala hodnota a pod ním jméno. Není složité si takovou komponentu vytvořit (Obrázek 27).

Její zdrojový text je velmi jednoduchý:

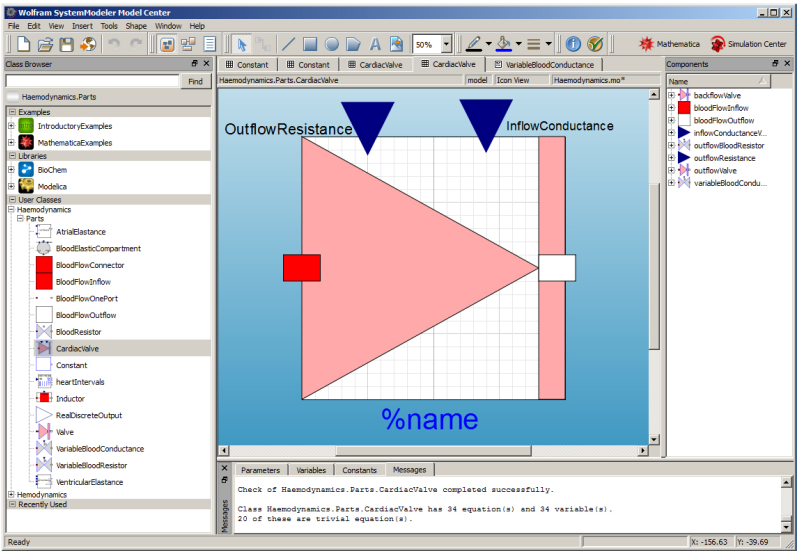
```
block Constant „Generate constant signal of type Real“
  parameter Real k(start=1) „Constant output value“;
  Modelica.Blocks.Interfaces.RealOutput y ;
equation
  y=k;
end Constant;
```

4. Skládáme model

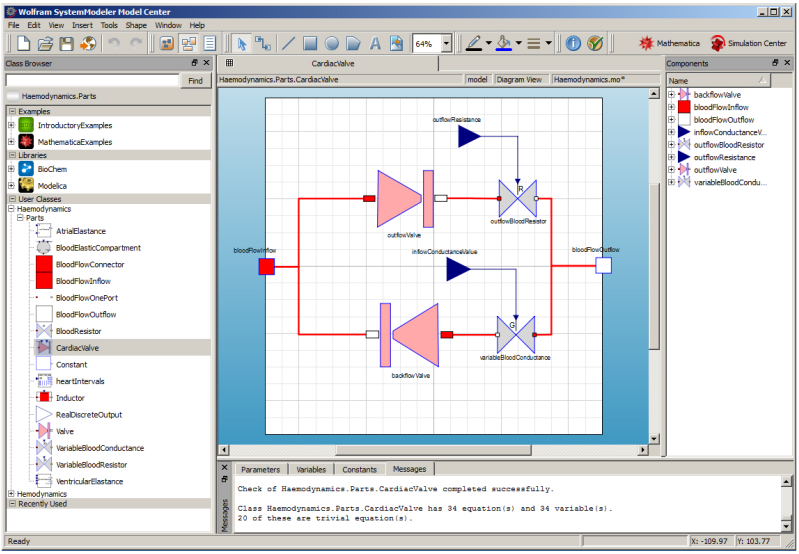
4.1 Srdeční chlopeň

Nyní již máme všechny potřebné „logové kostičky“ připraveny a můžeme postupně vytvářet model hemodynamiky.

Začneme vytvořením komponenty srdečních chlopní. Van Meurs [44] chlopeň vytváří z kombinace ventilu a odporu. Díky možnosti měnit odpor, můžeme modelovat chlopenní stenózu. A co když budeme chtít modelovat nedomykavost chlopní. Nejjednoduším řešením je poskládat si chlopeň ze dvou ventilů, jeden nastavit ve správném směru spolu s odporem, druhý nastavíme v obráceném směru, a zamkneme jej rezistorem s nulovou vodivostí. Pokud vodivost bude nenulová i skrz uzavřenou srdeční chlopeň může pronikat krev a můžeme tím simulovat nedomykavost chlopní. Jako vnější vstup do komponenty budeme zadávat rezistenci chlopně v jednom směru



Obrázek 28 — Vstupem do komponenty srdečních chlopní je výtoková rezistence a vtoková vodivosr (pro modelování nedomykavosti).



Obrázek 29 — Vnitřní struktura komponety srdečních chlopní..

a vodivost v druhém směru. Tento přístup umožní modelovat kombinované chlopenní poruchy stenózy a nedomykavosti. (viz Obrázek 28).

Vlastní komponentu jsme naprogramovali ve vizuálním grafickém editoru (Obrázek 29).

Zdrojový text komponenty pouze ilustruje, že komponentu srdečních chlopní jsme vytvořili skládáním:

```
model CardiacValve
   Haemodynamics.Parts.BloodFlowInflow bloodFlowInflow;
   Haemodynamics.Parts.BloodFlowOutflow bloodFlowOutflow;
   Haemodynamics.Parts.Valve outflowValve(open(fixed=true,
   start=true));
   Haemodynamics.Parts.Valve backflowValve(open(fixed=true,
   start=false));
   Haemodynamics.Parts.VariableBloodResistor
   outflowBloodResistor;
   Modelica.Blocks.Interfaces.RealInput outflowResistance „in
   torr sec / ml";
   Modelica.Blocks.Interfaces.RealInput
   inflowConductanceValue(start=0) „in ml/sec/torr";
   Haemodynamics.Parts.VariableBloodConductance
   variableBloodConductance;
equation
   connect(backflowValve.bloodFlowOutflow,bloodFlowInflow);
   connect(bloodFlowInflow,outflowValve.bloodFlowInflow);
   connect(outflowValve.bloodFlowOutflow,outflowBloodResistor.
   Inflow);
   connect(outflowBloodResistor.Outflow,bloodFlowOutflow);
   connect(outflowResistance,outflowBloodResistor.
   BloodResistance);
   connect(inflowConductanceValue,variableBloodConductance.
   BloodConductance);
   connect(backflowValve.
   bloodFlowInflow,variableBloodConductance.Outflow);
   connect(bloodFlowOutflow,variableBloodConductance.Inflow);
end CardiacValve;
```

4.2 Pravé srdce

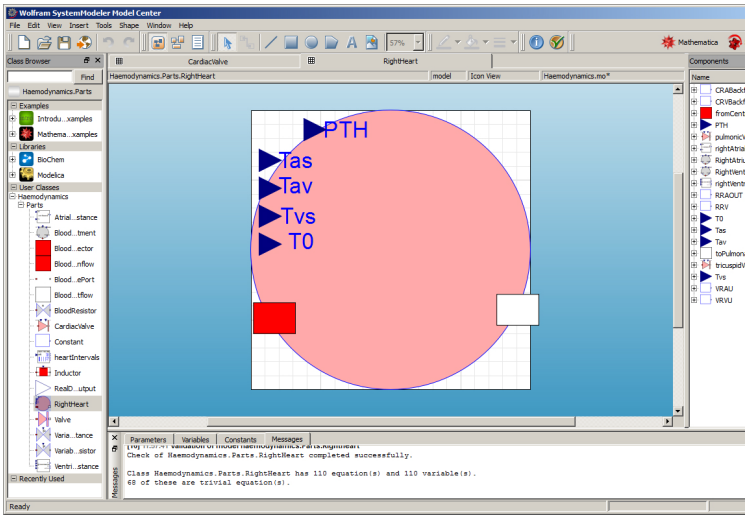
Nyní máme již připraveny všechny komponenty, z nichž můžeme složit model pravého srdce (Obrázek 30). Řídícími vstupy do komponenty budou hodnoty srdečních intervalů a nitrohrudní tlak (PTH).

Celý model vytváříme v grafickém editoru jako schematický obrázek (Obrázek 31).

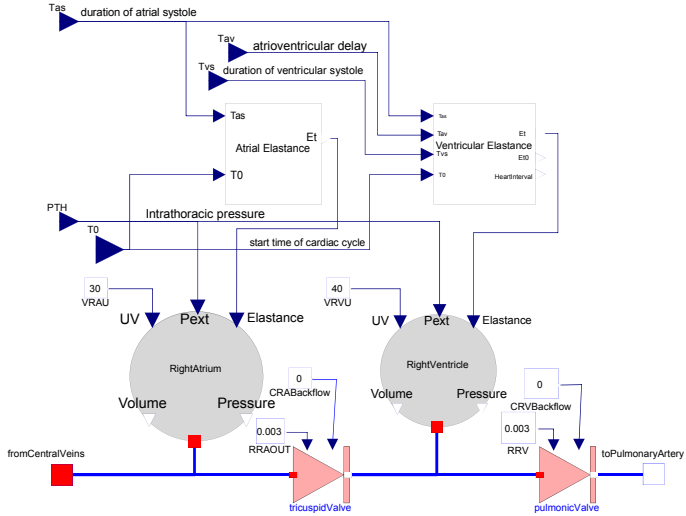
4.3 Levé srdce

Obdobně složíme model levého srdce (Obrázek 32).

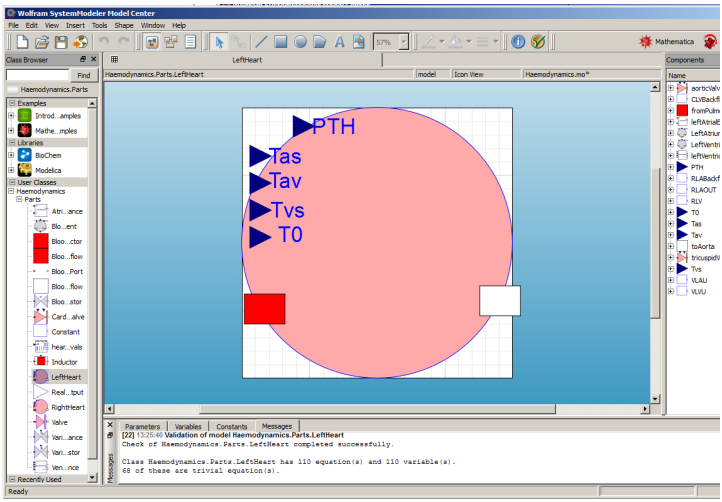
Struktura modelu vyjadřuje strukturu modelované reality (včetně všech zjednodušení, které jsme udělali) – Obrázek 33.



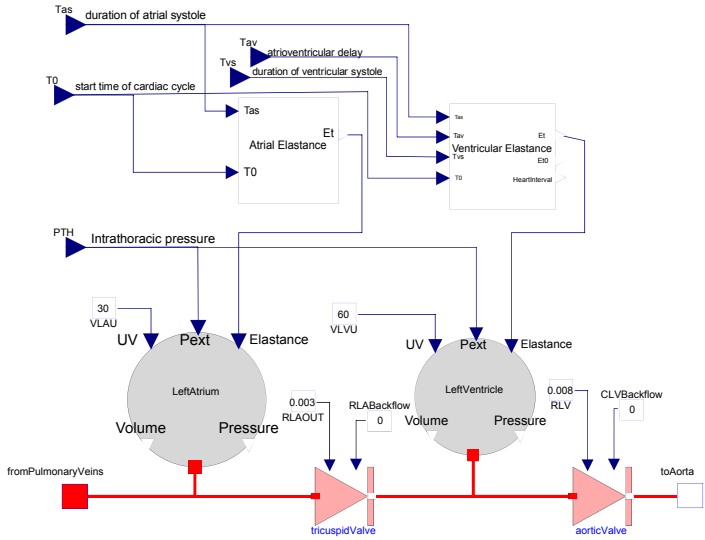
Obrázek 30 — Komponenta pravého srdce. Řídící vstupy: PTH - nitrohrudní tlak, Tas - délka systoly síní, Tav atrioventrikulární interval, Tvs - délka systoly komor, T0 - čas začátku srdečního cyklu.



Obrázek 31 — Vnitřní struktura komponenty pravého srdce. Vstupem do srdečních chlopní jsou odpory pro výtok (RRAOUT, RRV) a vodivosti pro zpětný vtok (CRABackflow a CRVBackflow). Změnou jejich hodnot lze modelovat chlopňové vady. VRAU a VRVU jsou hodnotx reziduálního (unstressed) objemu. Zmenou parametrů EMAX (maximální systolické elastance) a EMIN (diastolické elastance) v instancích AtrialElastance a VentricularElastance (parametry jsou dostupné na kliknutí) se dá měnit výkonnost myokardu komor a síní.



Obrázek 32 — Komponenta levého srdce. Řídící vstupy: PTH - nitrohrudní tlak, Tas - délka systoly síní, Tav atrioventrikulární interval, Tvs - délka systoly komor, T0 - čas začátku srdečního cyklu.



Obrázek 33 — Vnitřní struktura komponenty levého srdce. Vstupem do srdečních chlopní jsou odpory pro výtok (RLAOUT, RLV) a vodivosti pro zpětný vtok (CLABackflow a CLVBackflow). Změnou jejich hodnot lze modelovat chlopňové vady. VLAU a VLVU jsou hodnotx reziduálního (unstressed) objemu. Zmenou parametrů EMAX (maximální systolické elastance) a EMIN (diastolické elastance) v instancích AtrialElastance a VentricularElastance (parametry jsou dostupné na kliknutí) se dá měnit výkonnost myokardu komor a síní.

4.4 Systémové artérie

Z komponent složíme model systémových artérií – Obrázek 34.

Model vytváříme propojováním komponent v grafickém editoru – Obrázek 35.

4.5 Systémové periferní cévy

Vytvoříme komponentu systémových periferních cév, včetně vlastní ikonky - Obrázek 36

Model vytvořený skládáním komponent v grafickém editoru je samo-dokumentující – Obrázek 37.

4.6 Systémové žíly

Model systémových žil (Obrázek 38) také složíme z instancí již vytvořených komponent (Obrázek 39).

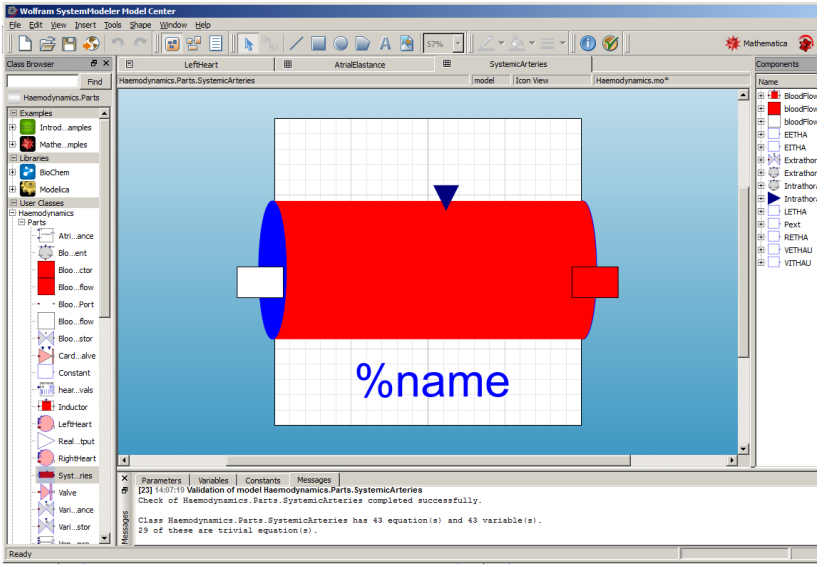
4.7 Plicní oběh

Poslední komponentou krevního oběhu, která nám scházela, je plicní cirkulace – Obrázek 40.

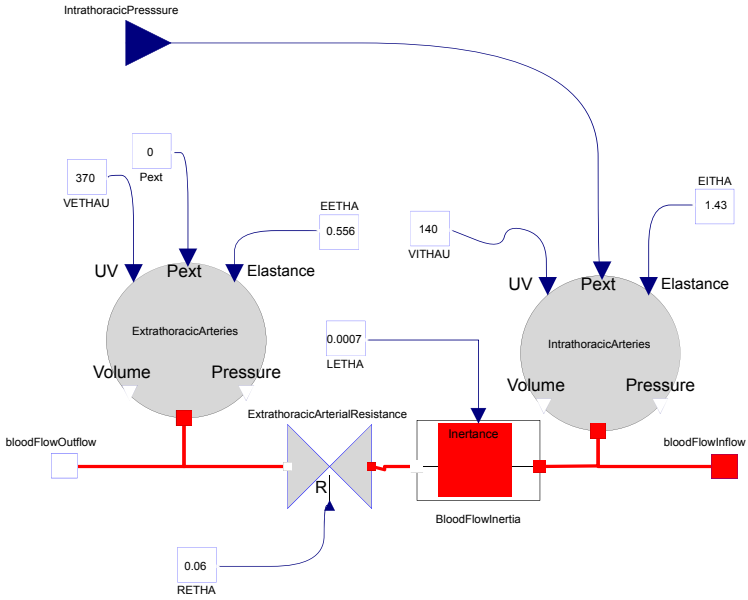
Plicní oběh také vytvoříme skládáním – Obrázek 41.

4.8 Model cirkulace jako celek

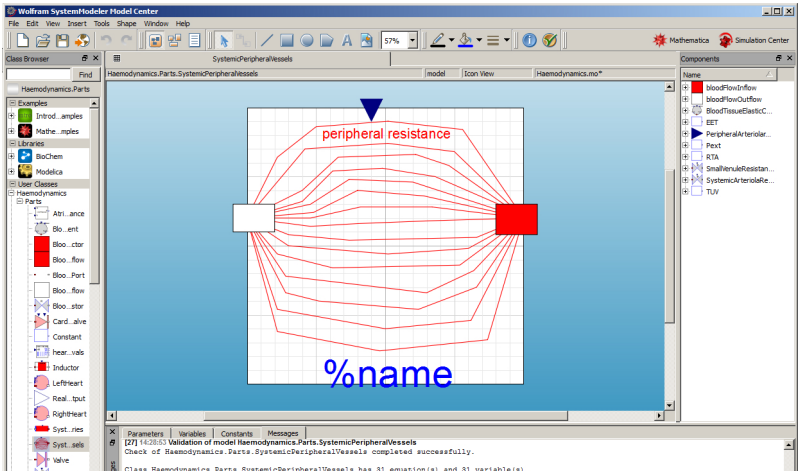
Nakonec složíme z vytvořených komponent celý model hemodynamiky krevního oběhu. Do ikonky vytvořeného modelu hemodynamiky můžeme umístit i obrázek – Obrázek 42.



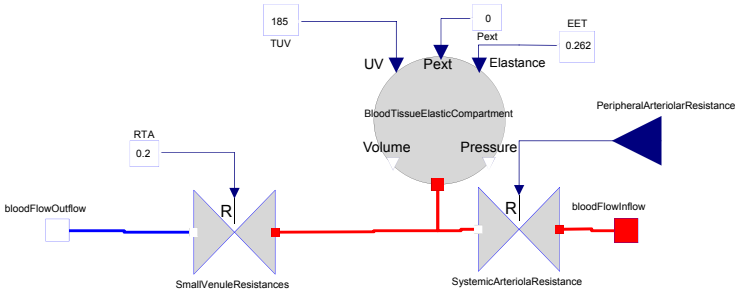
Obrázek 34 — Komponenta systémových artérií.



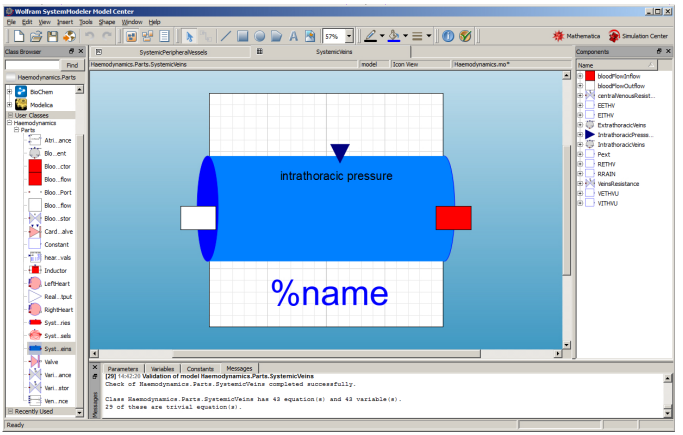
Obrázek 35— Komponenta systémových artérií. Jediným řídícím vstupem je zde nitrohrudní tlak.



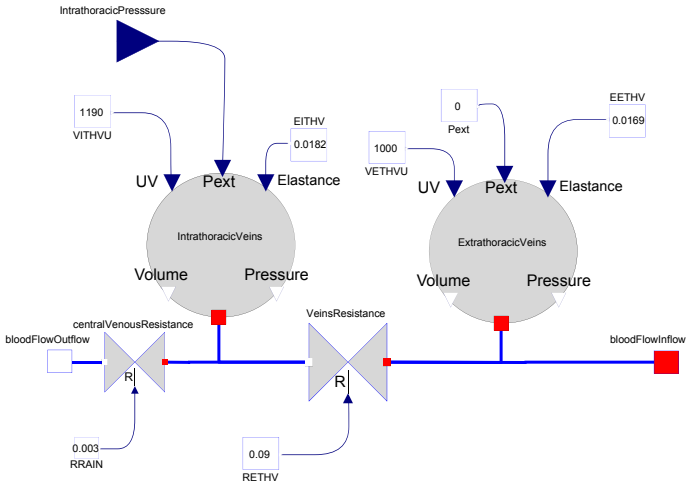
Obrázek 36 — Komponenta systémových artérií. Jediným řídícím vstupem je zde rezistence periferních arteriol.



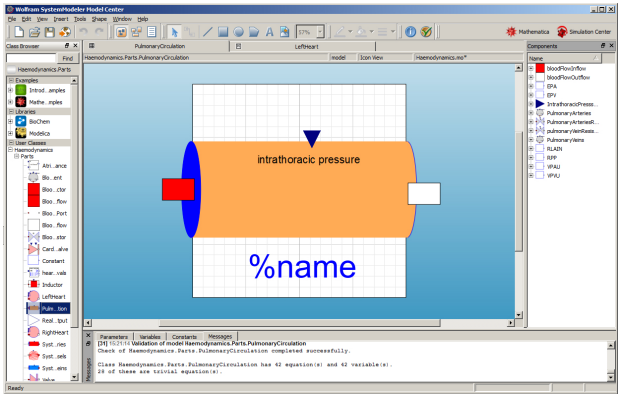
Obrázek 37— Komponenta systémových artérií. Jediným řídícím vstupem je zde rezistence systémových arteriol.



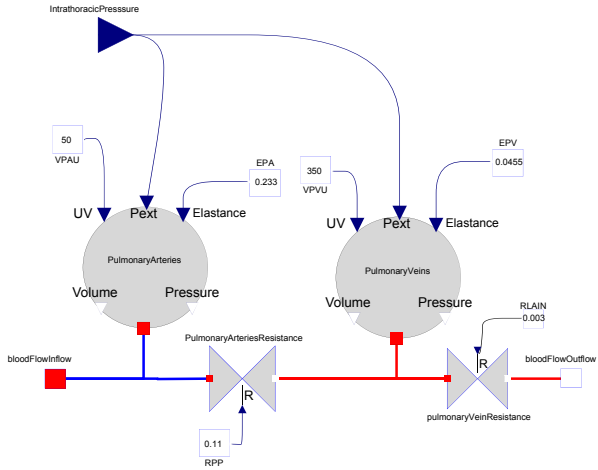
Obrázek 38 — Komponenta systémových žil. Jediným řídícím vstupem je zde nitrohrudní tlak



Obrázek 39 — Komponenta systémových žil. Jediným řídícím vstupem je zde nitrohrudní tlak



Obrázek 40 — Komponenta plicního oběhu, řídícím vstupem je zde nitrohrudní tlak



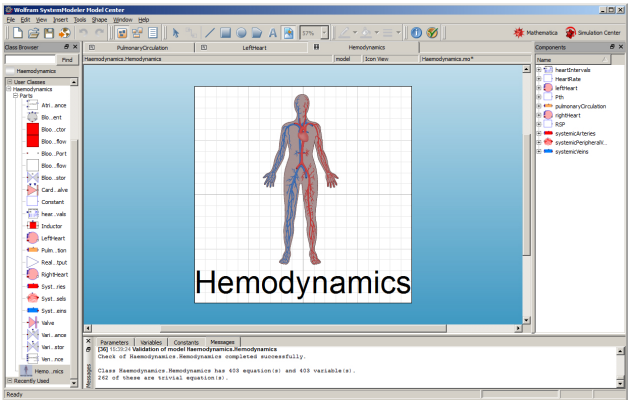
Obrázek 41 — Komponenta plicního oběhu, řídícím vstupem je zde nitrohrudní tlak

Vlastní model ovšem skládáme v grafickém editoru jako stavebnici z jednotlivých vytvořených komponent – Obrázek 43.

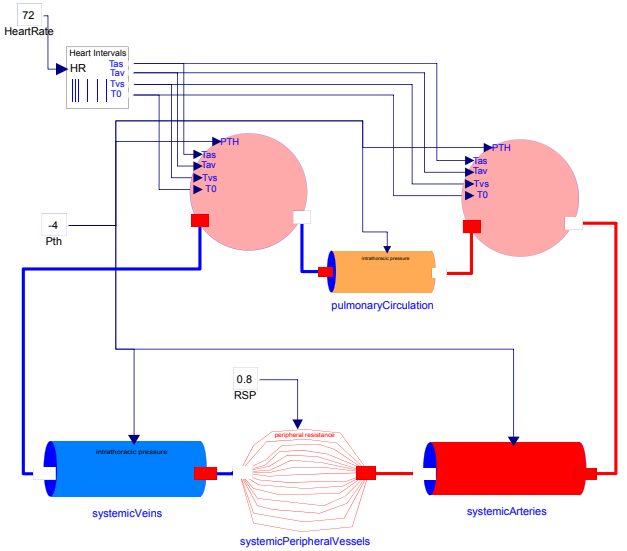
4.9 Testujeme chování modelu

Modeliková vývojová prostředí umožňují pohodlně testovat chování vytvářených modelů. Můžeme si snadno zobrazit průběhy libovolných proměnných. Tak např. můžeme sledovat pulzaci arterií, průtoky a tlaky v jednotlivých částech krevního řečiště apod. Např. na Obrázku 44 je uvedeno porovnání tlaků v aortě a v levé komoře.

Můžeme také zobrazovat různé závislosti – tak např. na Obrázku 45 je uveden vztah průběhu tlaků a objemu pravé a levé komoře.



Obrázek 42 — Model hemodynamiky krevního oběhu jako komponenta.

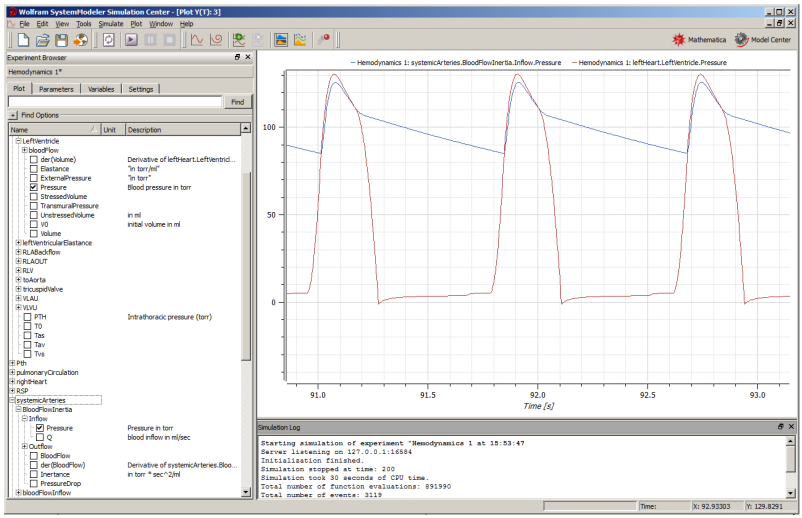


Obrázek 43 — Vnitřní struktura modelu hemodynamiky krevního oběhu.

4.10 Další směry rozvoje modelu

Model hemodynamiky krevního oběhu je možno dále rozvíjet v mnoha směrech. Bylo by třeba zajímavé pokusit se namodelovat různé chlopenní vady. Vhodné by bylo zavést do modelu oběhu řízení (baroreflex apod.). Dále by asi bylo vhodné rozčlenit průtok systémovou cirkulací na několik oblastí – protože různé části krevního řečiště jsou různě řízeny – již Guyton v roce 1972 uvažoval zvlášť svaly, ledviny a ostatní tkáně.

Především je asi vhodné propojit model cirkulace s modelem respirace a přenosu krevních plynů – úkolem cirkulace není jen rozvádět po organismu



Obrázek 44 — Ukázka výstupu modelu hemodynamiky - porovnání průběhů tlaku v aoorte a v levé komoře.

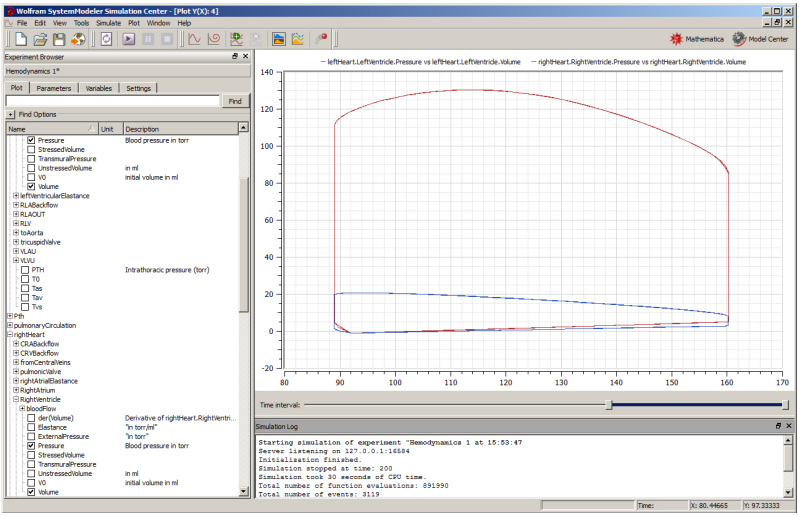
krev, ale především přivádět do tkání potřebné látky, zejména kyslík, a odvádět zplodiny, především oxid uhličitý. Přenos krevních plynů je velmi ovlivňován také respirací. Proto také model, který je implementován v simulátoru HTS, je rozpracován zejména v propojení modelu cirkulace a respirace.

Cílem této kapitoly ale nebyl podrobný popis a rozpracování složitějšího modelu fyziologických regulací. Cílem bylo na konkrétním příkladu ukázat, jak efektivním nástrojem pro modelování je Modelica, která si své místo v průmyslu již získala, ale v biomedicínských aplikacích na větší rozšíření teprve čeká.

5 HumMod – nerozsáhlejší model integrativní fyziologie

Nejrozsáhlejším modelem propojených fyziologických systémů je dnes patrně model **HumMod** vytvořený v mezinárodní kooperaci skupinou spolupracovníků a žáků A. Guytona, z Mississippi University Medical Center z USA. Model, včetně jeho zdrojového textu, je možné stáhnout z webových stránek modelu *http://hummod.org*. Na rozdíl od modelu, který je na pozadí simulátoru HPS, autoři se netají jeho strukturou, která je, patrně, mnohem složitější, než je model v simulátoru HPS. I když je zdrojový text simulátoru a celý matematický model na jeho pozadí nabízen jako „open source" (a uživatel si teoreticky může i model modifikovat), je orientace v matematických vztazích prohlížením tisícovek vzájemně provázaných XML souborů poměrně obtížná.

Uživatel proto může model upravovat i modifikovat. Potíž tkví ale v tom, že zdrojové XML texty celého modelu jsou napsány v celkem 3235 souborech umístěných v 1367 složkách! Díky tomu jsou rovnice modelu a jejich návaznosti obtížně srozumitelné a řada řešitelských týmů při vývoji lékařských simulátorů pro raději jako východisko pro další rozšíření raději sahá po starších



Obrázek 45 — Ukázka výstupu modelu hemodynamiky - tlakově-objemový diageam levé a pravé komory.

modelech komplexních fyziologických regulací - např. modelech Guytona z roku roku 1972 [15] a modelech Ikedy z roku 1979 [18]. Touto cestou se např. vydal mezinárodní výzkumný tým v projektu SAPHIR (System Approach for Physiological Integration of Renal, cardiac and repiratory control), když se zdrojové texty modelu QHP se účastníkům projektu zdály velmi špatně čitelné a obtížně srozumitelné [42]. Obdobně, nedávno Mangourova a spol. [33] implementovali v Simulinku raději starší Guytonův model z roku 1992, než poslední (pro ně, jak uvádějí, špatně čitelnou) verzi modelu QHP/Hummod týmu Guytonových spolupracovníků a žáků.

My jsme se toho nezalekli a s americkými autory jsme navázali spolupráci. Vytvořili jsme speciální softwarový nástroj **QHPView** [24, 25], který z tisícovek souborů zdrojových textů modelu vytvoří přehledné zobrazení použitých matematických vztahů a model jsme implementovali v jazyce Modelica (*http://www.3ds.com/products/catia/portfolio/dymola*).

Implementace modelu HumMod v Modelice podstatně zpřehlednila strukturu modelu [26] - a mimo jiné pomohla odhalit i některé chyby v původní americké implementaci modelu Hummod.

Model **Hummod** jsme modifikovali a rozšířili především v oblasti modelování přenosu krevních plynů a homeostázy vnitřního prostředí, zejména acidobazické rovnováhy. Při modifikaci jsme mimo jiné vycházeli z našeho původního modelu fyziologických regulací, který byl jádrem našeho staršího výukového simulátoru **Golem** [23, 31] a z našeho bilančního přístupu k acidobazické rovnováze [22, 27]

Strukturu našeho modelu, který jsme nazvali „Hummod-Golem edition" zveřejňujeme na webových stránkách projektu (*http://physiome.cz/Hummod*)

ve zdrojové formě s definicí všech proměnných a všech rovnic. Model je teoretický podkladem současně vyvíjeného lékařského výukového simulátoru. Našim cílem je zpřístupnit simulátor jako výukovou pomůcku dosažitelnou pomocí internetu. Při její tvorbě využijeme naši technologii tvorby webových simulátorů podrobněji popsanou v [25].

Naše zkušenost s modelem HumMod ukazuje že Modelice je velice efektivním prostředkem zejména pro vytváření rozsáhlých hierarchicky organizovaných modelů. V jiném prostředí bychom v našem malém týmu tak rozsáhlý model nebyli schopni implementovat.

6. Závěr

Nové technologie přinášejí pro tvorbu simulačních modelů nové možnosti i nové výzvy. Jednou z nich je nový objektově orientovaný jazyk Modelica, který podle našeho názoru podstatně zjednoduší modelování tak složitých a komplexních systémů s jakými se setkáváme ve fyziologii. Vzhledem k tomu, že se ve fyziologických systémech vyskytuje řada vztahů, které v blokově orientovaných jazycích (např. v Simulinku) vedou na řešení implicitních rovnic, je akauzální popis používaný v Modelice velkou výhodou. Akauzální popis mnohem lépe vystihuje podstatu modelované reality a simulační modely jsou mnohem přehlednější a tudíž i méně náchylné k chybám. Jak ukazují (nejen naše) zkušenosti, pro modelování fyziologických systémů je Modelica velmi vhodným prostředím.

Poděkování

Děkuji mým spolupracovníkům z Oddělení biokybernetiky a počítačové podpory výuky, kteří zavedli technologii využívání jazyka Modelica do námi vytvářených simulátorů, tuto technologii v denodenní praxi využívají a v rámci našeho výzkumu i naši účasti v Open Modelica Source Consortium ji dále rozvíjejí. Tento text vznikl také na základě zkušeností s výukou jazyka Modelica v rámci výuky studentů ČVUT i řady pracovních workshopů na kterých se rovněž moji polupracovníci podílejí. Můj dík patří zejména: MUDr. Mgr. Pavolovi Privitzerovi, Mgr. Tomáši Kulhánkovi, Mgr. Marku Matejákovi, ing. Janu Šilarovi, ing. Filipu Ježkovi, ing. Martinu Tribulovi a ing. Tomáši Kročkovi.

Děkuji také grafičce Veronice Sýkorové, DIS a Kláře Ulčové,DIS za vytvoření obrázků k tomuto modelikovému tutoriálu.

Práce na vývoji lékařských simulátorů je podporována projektem MPO FR-TI3/869 a společností Creative Connections s.r.o.

Literatura

[1.] Breunese, A. P. J., & Broenink, J. F. (1997). Modeling mechatronic systems using the SIDO-PS+ language. Simulation Series, vol. 29, str. 301-306.

[2.] Brugård, J., Hedberg, D., Cascante, M., Cedersund, G., Gómez-Garrido, À., Maier, D., Nyman, E., Selivanov, E. & Strålfors, E. (2009). Creating a Bridge between Modelica and the Systems Biology Community. In 7th International Modelica Conference, Como, Italy, str. 473-479.

[3.] Cellier, F. E., & Nebot, A. (2006). Object-oreiented modeling in the service of medicine. Proceedings of the 6tha Asia Conference, Bejing, China 2006, vol. 1, str. 33-40. Bejing: International Academic Publishers.

[4.] Cheng, L., Ivanova, O., Fan, H. H., & Khoo, M. C. (2010). An integrative model of respiratory and cardiovascular control in sleep-disordered breathing.Respiratory physiology & neurobiology, vol. 174(1), str. 4-28.

[5.] Cheng, L., & Khoo, M. C. (2011). Modeling the autonomic and metabolic effects of obstructive sleep apnea: a simulation study. Frontiers in Physiology, 2. 111. Published online 2012 January 4. doi: 10.3389/fphys.2011.00111, PMCID: PMC3250672

[6.] Comenius, J. A. (1656). Schola Ludus, seu Encyclopaedea Viva. Sarospatak.

[7.] Couto, C. D. S., van Meurs, W. L., Goodwin, J. A., & Andriessen, P. (2006). A model for educational simulation of neonatal cardiovascular pathophysiology.Simulation in Healthcare, 1 (Inaugural), str. 4-9.

[8.] Elmqvist H. (1978). A Structured Model Language for Large Continuous Systems. PhD Thesis. Lund University, Sweden, May, 1987. Dostupné z: http://www.control.lth.se/Publication/elm78dis.html

[9.] Enderle J. & Brozino, J. (2012). Introduction to Biomedical Engineering, Third Edition. Academic Press, 2012, 1253 str., ISBN 978-0-12-3749797-6.

[10.] Ernst, T., Jähnichen, S., & Klose, M. (1997). The architecture of the smile/m simulation environment. In Proc. 15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics (vol. 6, str. 653-658).

[11.] Fan, H. H., & Khoo, M. C. (2002). PNEUMA-a comprehensive cardiorespiratory model. In Engineering in Medicine and Biology, 2002. 24th Annual Conference and the Annual Fall Meeting of the Biomedical Engineering Society EMBS/BMES Conference, 2002. Proceedings of the Second Joint (vol. 2, str. 1533-1534). IEEE.

[12.] Fritzon, P. (2003). Principles of object-oriented modeling and simulation with Modelica 2.1. Wiley-IEE Press. ISBN 0-471-47163-1

[13.] Fritzon, P. (2012) Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica. Wiley-IEE Press, ISBN 978-1-118-0168-6

[14.] Goodwin, J. A., van Meurs, W. L., Couto, C. D. S., Beneken, J. E., & Graves, S. A. (2004). A model for educational simulation of infant cardiovascular physiology. Anesthesia & Analgesia, vol. 99 (6), 1655-1664.

[15.] Guyton, A. C., Coleman, T. G., & Grander, H. J. (1972). Circulation: Overall Regulation. Ann. Rev. Physiol. , vol. 41, str. 13-41.

[16.] Haas, O. C., & Burnham, K. J. (2008). Systems Modeling and Control Applied to Medicine. V O. C. Haas, & K. J. Burnham, Intelligent and Adaptive Systems in Medicine (str. 17-52). Boca Raton Fl, USA: CRC Press.

[17.] Hoppensteadt, F.,C. (2011) Mathematical Methods for Analysis of a Complex Disease (Courant Lecture Notes), New York, AMMS, 2012, 151 str., ISBN: 978-0-8218-7286-4.

[18.] Ikeda, N., Marumo, F., & Shirsataka, M. (1979). A Model of Overall Regulation of Body Fluids. Ann. Biomed. Eng. , vol. 7, str. 135-166.

[19.] Ivanova, O. V., & Khoo, M. C. (2004). Simulation of spontaneous cardiovascular variability using PNEUMA. In Engineering in Medicine and Biology Society, 2004. IEMBS'04. 26th Annual International Conference of the IEEE (vol. 2, str. 3901-3904). IEEE.

[20.] Jeandel, A., & Boudaud, F. (1997). Physical System Modelling Languages: from ALLAN to Modelica. GAZ DE FRANCE-Research and Development Division.

[21.]Khoo M. C. K. (1999). Physiological control systems. New York. IEE Press. ISBN 0-7803-3408

[22.]Kofránek, J. (2009a). Komplexní model acidobazické rovnováhy. (Anglická verze: Complex model of acid-base balance je dostupná na adrese http://www.physiome.cz/references/medsoft2009acidbase.pdf, model je na adrese http://www.physiome.cz/acidbase). V M. Zeithamlová (Editor), MEDSOFT 2009 (str. 23-60). Praha: Agentura Action M.

[23.]Kofránek, J., Anh Vu, L. D., Snášelová, H., Kerekeš, R., & Velan, T. (2001). GOLEM – Multimedia simulator for medical education. Studies in Health Technology and Informatics, vol. 84, str. 1042-1046. Práce je dostupná na adrese http://www.physiome.cz/references/MEDINFO2001.pdf.

[24.]Kofránek, J., Mateják, M., & Privitzer, P. (2009). Leaving toil to machines - building simulation kernel of educational software in modern software environments. CD ROM. V L. Dušek, D. Schwarz, & S. Štípek (Editor), Mefanet 2009, Conference Proceedings (str. kofranek.pdf: 1-39). Brno: Masarykova Univerzita. Práce je dostupná na adrese http://www.physiome.cz/references/MEFANET2009.pdf.

[25.]Kofránek, J., Mateják, M. & Privitzer, P. (2010). Web simulator creation technology. In MEFANET report, vol. 3 (Dušek, Vladimír Mihál, Stanislav Štípek, Jarmila Potomková, Daniel Schwarz, Lenka Šnaidrová, Eds.). Institute of Biostatistics and Analysis. Masaryk University, 2010, ISSN 1004-2961, str. 52-97. Práce je dostupná na adrese http://www.physiome.cz/references/mefanetreport3.pdf.

[26.]Kofránek, J., Mateják, M. & Privitzer, P. (2011) HumMod - large scale physiological model in Modelica. Proceedings of 8th. International Modelica conference, Dresden, Germany, March 20-22, 2011, Dresden, Linköping Electronic Conference Proceedings (ISSN: 1650-3686), str. 713-724, Dostupno na: http://www.ep.liu.se/ecp/063/079/ecp11063079.pdf.

[27.]Kofránek, J., Matoušek, S., & Andrlík, M. (2007). Border flux ballance approach towards modelling acid-base chemistry and blood gases transport. V B. Zupanic, S. Karba, & S. Blažič (Editor), Proceedings of the 6th EUROSIM Congress on Modeling and Simulation, Full Papers (CD) (str. TU-1-P7-4: 1-9). Ljubljana: University of Ljubljana. Práce je dostupná na adrese http://www.physiome.cz/references/ljubljana2007.pdf

[28.] Kofránek, J., Matoušek, S., Rusz, J., Stodulka, P., Privitzer, P., Mateják, M., & Tribula, M. (2011). The Atlas of Physiology and Pathophysiology: Web-based multimedia enabled interactive simulations. Computer methods and programs in biomedicine, vol. 104 (2), str. 143-153.

[29.]Kofránek, J. Privitzer, P. Mateják, M., Matoušek, S. (2011). Use of web multimedia simulation in biomedical teaching. In Proceedings of the 2011 International Conference on Frontiers in Education: Computer Science & Computer Engineering, Las Vegas, July 18-21, 2011, (H. R. Arabia, V. A. Cincy, L. Deligianidis, Eds.), ISBN 1-60132-180-5, CSREA Press, Las Vegas, Nevada, 2011, 282-288.

[30.]Kofránek, J & Rusz, J. (2010). Restoration of Guyton diagram for regulation of the circulation as a basis for quantitative physiological model development. Physiological Research, vol. 59, str. 897-908 Práce je dostupná na adrese http://www.biomed.cas.cz/physiolres/pdf/59/59_897.pdf .

[31.]Kofránek, J., Velan, T., & Kerekeš, R. (1997). Golem: a Computer Simulator of Physiological Functions as an Efficient Teaching Tool. V Y. M. Theo, W. C. Wong, & T. J. Okeu (Editor), Legacy for 21 Century. Proceedings of the World Congress on System Simulation. (str. 407-411). Singapore: IEE Singapore Section.

[32.]Logan, J. D., & Wolesensky, J. D. (2009). Mathematical methods in biology. Hoboken, NJ: John Wiley & Sons, Inc.

[33.]Mangourova, V., Ringwood, J., & Van Vliet, B. (2011). Graphical simulation environments for modelling and simulation of integrative physiology. Computer Methods and Programs in Biomedicine , vol. 102 (3), str. 295–304.

[34.] Mattsson, S.E, Andersson, M & Aström K.J. (1993). Object-oriented modeling and simulation. In: Linkens, ed., CAD for Control Systems (Marcel Dekker, 1993), str. 31-69.

[35.]McLeod, J. (1967). PHYSBE... a year later. Simulation ,vol. 10, str. 37-45.

[36.] McLeod, J. (1966). PHYSBE: A ophysiological simulation benchmark experiment. Simulation ,vol. 15, str. 324-329.

[37.] McLeod, J. (1970). Toward uniform documentation-PHYSBE and CSMP. Simulation , vol. 14, str. 215-220.

[38.] Oomnes, C., Breklemans, M., & Baaijens, F. (2009). Biomechanics: concepts and computation. Cambridge: Cambridge University Press.

[39.]Ottesen, M. S., Olufsen, M. J., & Larsen, J. K. (2004). Applied Mathematical Models in Human Physiology. Philadelphia: Society for Indistrial amd Applied Mathematics.

[40.]Sahlin, P., Bring, A., & Sowell, E. F. (1996). The Neutral Model Format for building simulation, Version 3.02. Report. Dept. of Building Sciences, KTH, Stockholm.

[41.] Schiff, S. J. (2012). Neural Control Engineering: The Emerging Intersection Between Control Theory and Neuroscience. MIT Press, 2012, 386 str. ISBN: 978-0-262-01537-0.

[42.]Thomas, R. S., Baconnier, P., Fontecave, J., Francoise, J., Guillaud, F., Hannaert, P., Hernándes, P., Hernándes,A., La Rolle, V., Maziere, P., Tahi, F. & White, R. J. (2008). SAPHIR: a physiome core model of body fluid homeostasis and blood pressure regulation. Philosophical Transactions of the Royal Society, vol. 366, str.. 3175-3197.

[43.]Tiller M.,M. (2001). Introdiction to physical modeling with Modelica. Kluver Academic Publishers, Boston, 2001. ISBN 978-9-7923-9367-4.

[44.]van Meurs, W. (2011): Modeling and Simulation in Biomedical Engineering: Applications in Cardio respiratory Physiology. Ed. 1. New York: McGraw Hill, 2011. 193 stran. ISBN 978-0-07-17-1445-7.

[45.]Viklund, L., & Fritzson, P. (1995). ObjectMath–An Object-Oriented Language and Environment for Symbolic and Numerical Processing in Scientific Computing. Scientific Programming, vol. 4 (4), str. 229-250.

[46.]Wallish, P., Lusignan, M., Benayoun, M., Baker, T. I., Dickey, A. S., & Hatsopoulos, N. G. (2008). MATLAB for Neuroscientists: An Introduction to Scientific Computing in MATLAB. Burlington, MA: Academic Press.

Kontakt:

Doc. MUDr. Jiří Kofránek, CSc.
Oddělení biokybernetiky a počítačové
podpory výuky
ÚPF 1. LF UK, Praha
U nemocnice 5,
128 53 Praha 2
tel: 777686868
e-mail: kofranek@gmail.com