NOVÉ POSTUPY V TVORBĚ SIMULÁTORŮ – INTELIGENTNÍ PROPOJENÍ MATLAB SIMULINKU S PLATFORMOU .NET A TVORBA STAVOVÝCH AUTOMATŮ ŘÍDÍCÍCH VÝSLEDNOU APLIKACI.

Petr Stodulka, Pavol Privitzer, Jiří Kofránek, Jan Mašek
Ústav patologické fyziologie 1. LF UK Praha

Abstrakt

Technologii tvorby medicínských simulátorů jsme detailně popisovali v sérii příspěvků pro předchozí ročníky konference Medsoft. V podstatě se jedná o vytvoření matematického modelu v prostředí Matlab Simulink, návrh grafického rozhraní simulátoru a jejich propojení pomocí střední programové vrstvy. Tento přístup však s sebou nese řadu technických problémů, ať už se jedná o propojení Matlab s .NET, vytvoření adekvátního grafického výstupu v .NET nebo Macromedia Flash, řízení chování aplikace v závislosti na jejím aktuálním stavu apod. V roce 2005 jsme podstatně vylepšili technologii konverze fyziologických modelů z prostředí Matlab Simulink na komponenty platformy MS .NET. Nová technologie umožňuje použít v jedné aplikaci více instancí stejného simulačního modelu a zahrnuje programátorsky efektivní způsob jejich propojení s uživatelským prostředím simulátoru pomocí zvláštních objektů zajišťujících automatickou propagaci hodnot mezi vizuálními komponentami a modelem. Výsledný simulátor je samostatnou aplikací pro MS Windows, jejíž běh nevyžaduje instalaci programu Matlab. Velký vývoj doznala také tvorba hierarchických stavových automatů, které řídí vnitřní logiku simulátoru. Vyvinuli jsme vlastní nástroj pro vizuální návrh těchto automatů, jejich ladění a generování jejich zdrojového kódu, který se stane součástí výsledné aplikace.

Klíčová slova

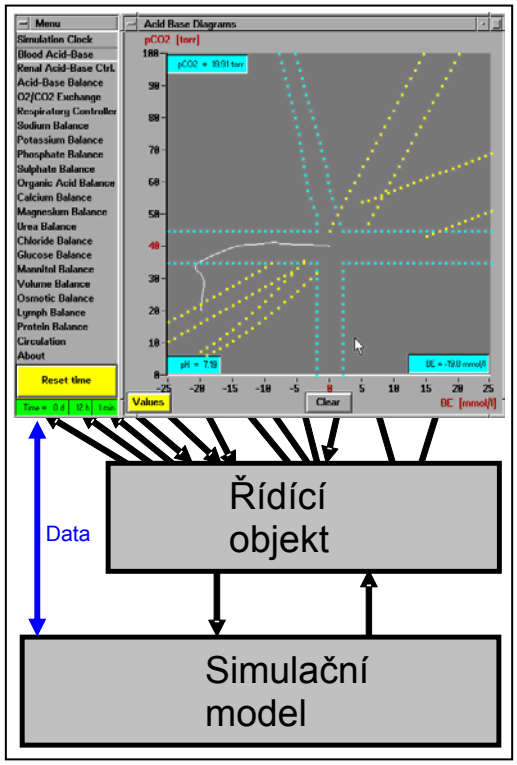
Výukové simulátory, MVC architektura, vizuální vývojové prostředí, propojeni Matlab s .NET

Úvod

Simulátor, jak mu budeme rozumět v tomto textu, je program, který se snaží věrně napodobit chování určitého systému. Typicky jde o znázornění fyziologických pochodů popsaných matematickým modelem.

Zabýváme se zde zejména simulátory, které mají interaktivní vizuální uživatelské rozhraní nebo se může jednat až o jakousi „fyziologickou simulační hru“. Účelem takových simulátorů je názorné a interaktivní zprostředkování netriviálních představ o dynamice fyziologických systémů a jejich regulaci. Použití simulátorů jsme podrobně popisovali v článku [1].

Samotná aplikace simulátoru se skládá ze tří základních částí – vrstev. Model, který má na starosti datový a funční základ simulátoru, uživatelské rozhraní, které musí vhodným způsobem prezentovat chování modelu a umožnit jeho ovládání, a střední vrstva, tzv. controller, který má na starosti propojení a vzájemnou komunikaci modelu s GUI (grafickým uživatelským rozhraním). V literatuře je tento přístup znám jako MVC architektura (*model-view-controller*) nebo také UCM architektura (*user interface-control object-model layer*). Bohužel, neexistuje univerzální prostředí, ve kterém by bylo možné dobře řešit jak problémy týkající se tvorby modelu, tak např. tvorby dostatečně graficky vyspělého uživatelského rozhraní. Jedním ze základních technických problémů, které je potřeba v technologii tvorby simulátorů vyřešit, je proto nalezení vhodných platforem pro jednotlivé vrstvy a jejich vzájemné propojení.



Základní platforma, na které pracujeme, je framework Microsoft .NET. Pro jednotlivé vrstvy výukových medicínských simulátorů používáme následující technologie:

- Model – typicky se jedná o fyziologický model vytvořený v prostředí Matlab Simulink. Modely jsou obvykle příliš složité pro implementaci přímo v některém z obvyklých programovacích jazyků, používáme proto prostředí speciálně k tomuto účelu určené. Modely vytvořené v Matlabu nelze přímo použít v prostředí .NET, nicméně lze je s pomocí speciálního nástroje do něj exportovat.
- View – prezentační vrstvu aplikace je možné vytvořit použitím vizuálních komponent .NET, a to ať už standardně dodávaných v distribuci nebo vlastních. Pro náročné vizualizace (např.

animované tepající srdce) používáme klipy vytvořené v prostředí Macromedia Flash. Zásadní vlastností těchto animací je, že je lze programově řídit a tím ovlivňovat prezentaci. Je však zároveň možné z klipu informace získávat, což umožňuje tvorbu animovaných ovládacích prvků. Klipy jsou do prostředí .NET vkládány pomocí technologie ActiveX.

- Controller – řídící vrstva aplikace, která má na starosti reakce na události vzniklé v GUI, řízení běhu modelu a zpětné ovlivnění uživatelského prostředí. Vyvinuli jsme vlastní framework pro programátorsky efektivní propojení uživatelského prostředí s modelem, který zajišťuje inteligentní automatickou propagaci hodnot mezi těmito vrstvami. Protože logika simulátoru může být velmi komplexní, je vhodné centralizovaně uchovávat informaci o stavu simulátoru a podle něj reagovat na vzniklé události. Řešením tohoto problému jsou tzv. hierarchické stavové automaty, o nichž je detailně pojednáno v [2]. Jejich aplikace v simulátorech je dále rozebrána v [3, 4]. Pro zjednodušení tvorby stavových automatů jsme vyvinuli vizuální prostředí umožňující graficky automaty navrhnout, vygenerovat jejich kód a také je ladit.

Model

Abychom mohli v prostředí .NET používat model vytvořený v Matlab Simulinku, je potřeba ho nejdřív vyexportovat do kódu C/C++, který je s .NET kompatibilní. To se dělá pomocí Real-Time Workshopu, což je nástroj komerčně dodávaný spolu s Matlabem. Ten je schopný vygenerovat zdrojový kód v jazyku C, který je funkčně ekvivalentní s původním modelem v Matlabu. Pro Real-Time Workshop jsme vytvořili speciální tzv. target do Visual Studia .NET, který generovaný kód „zaobalí“ podle našich potřeb do jazyka C++ pro .NET framework. Výsledkem překladu je potom tzv. .NET assembly (ve formě .dll souboru), ve které je implementovaná třída modelu z Matlabu přímo použitelná v prostředí Visual Studia .NET a každá její instance funguje jako samostatný model. Velkou výhodou je to, že je exportovaný model kompletně nezávislý na původní platformě, tj. Matlabu.

Vygenerovaná třída modelu má jako své členy tzv. vstupní a výstupní porty, kterými do modelu vstupují hodnoty, resp. vystupují požadované výstupní hodnoty. Běh instance modelu je pak možno řídit jeho metodami a vlastnostmi. Model nabízí možnost nadefinovat uživatelské obslužné metody, kterými se např. implementují počáteční vstupní hodnoty modelu či programová kontrola vstupních hodnot na jednotlivých portech. Třída modelu i jeho porty dále implementují

rozhraní, které slouží pro napojení propojovací vrstvy, o které pojednává následující kapitola.

Vizuální rozhraní a jeho propojení s modelem

Jak bylo napsáno v úvodu, tvoří prezentační vrstvu (View) vizuální komponenty .NET, a to buď standardně dodávané v distribuci nebo vlastní, které jsme vytvořili v naší laboratoři. Pro náročné vizualizace je pak možno použít klipy vytvořené v prostředí Macromedia Flash. Návrh prezentační vrstvy potom sestává z účelného rozmístění požadovaných vizuálních prvků na jednotlivé scény či obrazovky simulační aplikace.

Při tvorbě aplikace je potřeba zajistit jednak vstup hodnot zadaných uživatelem ve vizuálním rozhraní do modelu a naopak zobrazení hodnot, které počítá model, ve vizuálních komponentách. V prvém případě vyvstává problém, když je možné stejnou vstupní proměnnou modelu zadávat pomocí více druhů vstupu (např. pomocí „šoupátka“ a zároveň editačního textového pole). Změna v jednom vstupu se musí kromě propagace do modelu projevit také změnou zobrazené hodnoty v druhém vstupním prvku. Při zobrazování výstupů modelu je naopak potřeba automaticky aktualizovat zobrazení v uživatelském prostředí, kdykoliv dojde ke změně jejich hodnoty.

Bez prostřední propojovací vrstvy (součást vrstvy controller), o které se zmíníme dále, by pak další implementační krok spočíval v programové definici událostních metod vizuálních komponent, které by se staraly o zápis resp. čtení hodnot z a do modelu a dále pak o synchronizaci propagace vstupních hodnot na alternativní vizuální komponenty, které zapisují své hodnoty do stejných vstupních portů modelu. Také i kontrolu vstupních hodnot by bylo třeba implementovat na úrovni vizuálních komponent, čímž často vznikají duplicity v kódu aplikace. Takovýto přístup je poněkud pracný a pro programátora dosti ubíjející, protože se neustále opakuje podobná posloupnost příkazů. Vzniká tak velmi nepřehledný kód aplikace, což je potenciálním zdrojem programových chyb.

Velikou část zmíněné programátorské práce lze zautomatizovat vhodně navrženým propojovacím systémem, který se postará o inteligentní propagaci hodnot mezi vizuálními prvky a modelem. Takový systém jsme navrhli. Jeho hlavní funkce jsou implementované pomocí speciálních propojovacích objektů, tzv. PortLinků. Jde o objekty, které při jejich vytvoření navážou funkční spojení mezi daným vizuálním prvkem a

daným portem či porty modelu. Samotný model a jeho porty implementují rozhraní, které tento propojovací systém podporují.

S pomocí tohoto systému pak stačí jenom propojit žádaný vstupní/výstupní port modelu s daným vizuálním prvkem a o většinu zmíněné práce se už automaticky postará propojovací mezivrstva. Automatická propagace hodnot funguje podle očekávání i v případech, kdy je na jeden port modelu napojených více alternativních vizuálních prvků. Příslušná uživatelská změna vstupu či změna výstupní hodnoty modelu se synchronně projeví na všech připojených vizuálních komponentách.

Další velikou výhodou navrženého systému je, že programová kontrola vstupních uživatelských hodnot je posunuta logicky blíže k modelu. Kontrolu hodnot je totiž možné definovat přímo na úrovni modelu v obslužných metodách událostí na portech. V součinnosti s PortLinky je tak vlastně kontrola vstupů automaticky propagována k vizuálním prvkům. Pro určitou skupinu portů, které spolu souvisí, je přitom možné definovat společnou obslužnou metodu a implementovat tak třeba i složitější vztahy mezi vstupními hodnotami modelu (obslužná metoda kontroly vstupních hodnot při svém vyvolání může měnit i jiné porty modelu, jejichž změna se taky automaticky zpětně propaguje k vizuálním prvkům). Programátor aplikace se tedy může věnovat hlavně samotným vztahům či kontrole hodnot na portech a nemusí přitom včleňovat kód, který se stará o propagaci změn na vizuální prvky, což je ohromná úspora programátorského času a prevence možných chyb.

Vznikla tak poměrně vyspělá vývojová podpora, která umožňuje jednoduchým propojováním požadovaných portů s vizuálními prvky poskládat uživatelské rozhraní simulátoru do funkčního celku a přitom se dál soustředit jen na těžko automatizovatelné programování. Pro jednoduché simulátory je takováto podpora často i postačující. U složitějších simulátorů je však někdy dosti náročné implementovat celkovou logiku chování výsledné aplikace. Pro tento účel jsme vyvinuli další nástroj popsaný v následující kapitole.

Stavové automaty

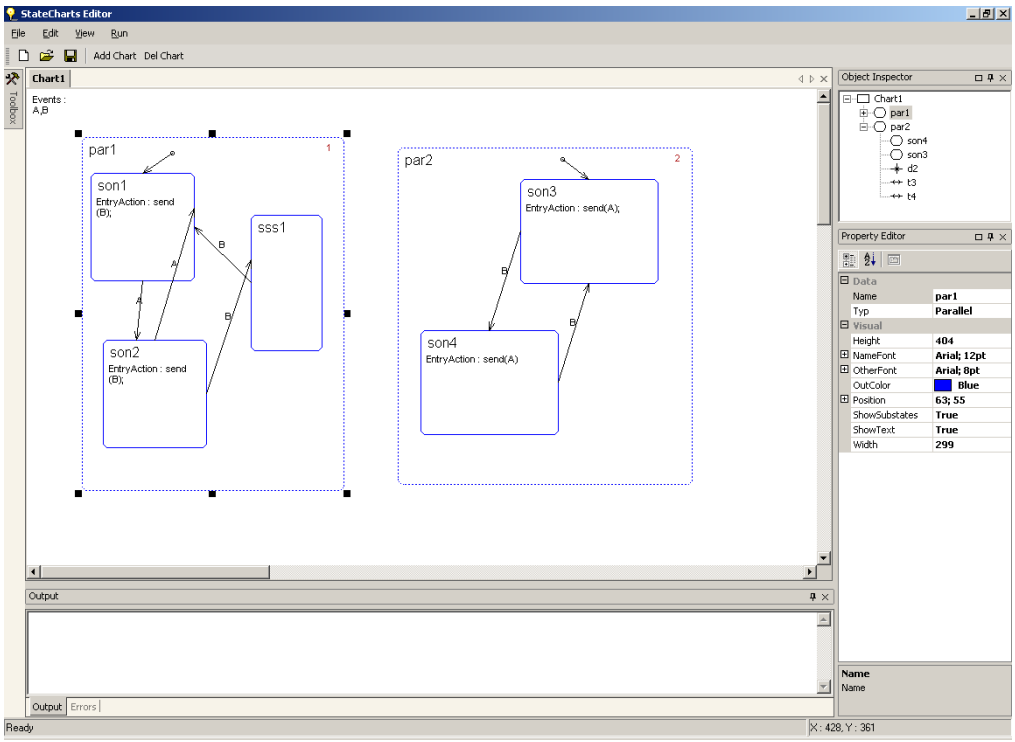
Stavové automaty jako způsob řízení aplikace v závislosti na jejím momentálním stavu byly podrobně popsány v [2, 3]. Jde o techniku uchování informace o aplikaci v tzv. hierarchickém stavovém automatu. Ten je přirozeným rozšířením běžného stavového automatu o hierarchii stavů, kdy platí, že každý stav, který je v hierarchii nadřazen aktivnímu

stavu, musí být také aktivní. Navíc je zaveden nový pojem tzv. paralelních stavů, které umožňují uchovávat informace o nezávislých subsystémech zároveň. Syntézou Moorova a Mealyho přístupu je možné definovat akce, kterými automat řídí aplikaci, jak na úrovní stavů, tak na přechodech mezi nimi.

Výhodou uchování stavové informace v automatu oproti použití sady stavových proměnných je především možnost graficky zobrazit logiku aplikace a jednoznačně a přehledně definovat přechody mezi stavy. Problémem je pochopitelně převod mezi grafickou podobou automatu a zdrojovým kódem, který vykonává požadovanou funkci. V roce 2004 jsme vyvinuli nástroj, který umožňoval generovat kód v jazyce C# z reprezentace hierarchického stavového automatu ve formátu XML [2]. Nyní se podařilo dokončit první verzi vizuálního editoru hierarchických stavových automatů, který ve spolupráci s výše zmíněným generátorem zajišťuje proces tvorby automatu až po jeho ladění a generování kódu. Podrobný popis je v [5].

Vizuální editor stavových automatů

Editor je samostatná aplikace napsaná v prostředí Microsoft .NET s použitím několika externích knihoven pro lepší vzhled uživatelského prostředí, které připomíná nejrozšířenější vývojový nástroj pro platformu .NET – Microsoft Visual Studio.



Editor umožňuje komfortně graficky tvořit stavové automaty a manipulovat s nimi pomocí drag&drop, měnit jejich vlastnosti v property editoru, případně pomocí kontextového menu atd. Řízení aplikace, jejíž součástí se automat stane, je prováděno v tzv. akcích ve stavech nebo na přechodech, které se zapisují přímo v jazyce C#.

Důležitou funkcí, kterou editor nabízí, je vygenerování zdrojového kódu automatu, jeho automatický překlad a následné **spuštění s možností ladění**. V tomto jsou aktivní stavy zvýrazněny a je možné sledovat změny, které v automatu probíhají v reakci na příchozí zprávy. Běh automatu je možné pozastavit, krokovat nebo zpomalit, je možné nastavit na některý stav zarážku, která pozastaví běh při dosažení tohoto stavu apod.

Editor je v první verzi a zatím neumožňuje zápis některých automatových konstrukcí, jako jsou např. tzv. *self-transitions* a vnitřní přechody. Stejně tak je potřeba doplnit ovládání o možnost zoomování, copy&paste a další.

Závěr

Podařilo se nám nalézt způsob, jak jednoduše používat modely vytvořené v prostředí Matlab Simulinku jako součást programů psaných na plaformě .NET. Tyto modely umíme jednoduše propojit s uživatelským rozhraním tak, že jsou automaticky propagovány hodnoty mezi vizuálními komponentami a vstupy nebo výstupy modelu. Chování aplikace řídíme pomocí stavových automatů, které je nyní možné tvořit ve vlastním vizuálním nástroji.

Další vývoj technologie tvorby výukových simulátorů budeme směřovat směrem k tvorbě vizuálních nástrojů pro propojení vrstev MVC architektury a jejich integraci do prostředí MS Visual Studia a snaze o automatizaci co největšího počtu úkonů spojených s implementací architektury simulátorů.

Literatura

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

[2] Stodulka, P.: *Vývojový nástroj pro návrh a testování komunikujících stavových automatů. Diplomová práce.* MFF UK, Praha 2004

[3] Kofránek J, Stodulka P.: *Hierarchické komunikující stavové automaty*; 2004; Objekty 2004; 2004

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

[5] Mašek J.: *Vývojové grafické prostředí pro návrh systémů pomocí paralelních komunikujících automatů. Bakalářská práce.* MFF UK, Praha 2006

Mgr. Petr Stodulka (petr.stodulka@lf1.cuni.cz)
Mgr. Pavol Privitzer (pavol.privitzer@lf1.cuni.cz)
Ústav patologické fyziologie
1. LF UK
U nemocnice 5
128 00 Praha-2