VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

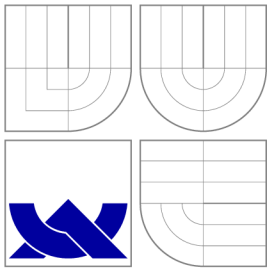
APLIKACE PRO VZDÁLENOU EDITACI DEVS MODELŮ A ŘÍZENÍ SIMULACE NA SIMULAČNÍM SERVERU

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

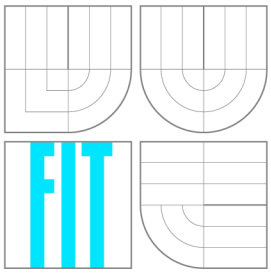
AUTOR PRÁCE
AUTHOR

Bc. JAN KOLAŘÍK

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

APLIKACE PRO VZDÁLENOU EDITACI DEVS MODELŮ A ŘÍZENÍ SIMULACE NA SIMULAČNÍM SERVERU

APPLICATION FOR REMOTE DEVS MODELLING AND SIMULATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JAN KOLAŘÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. Ing. VLADIMÍR JANOUŠEK, Ph.D.

BRNO 2013

Abstrakt

Práce se zabývá návrhem a samotnou implementací klient-server aplikace pro vzdálený přístup k modelům systémů uloženým na serveru. Aplikace umožňuje tyto modely také editovat a provádět nad nimi simulace. Součástí práce je i navržení samotného komunikačního protokolu mezi klientem a serverem. Klient je implementován pomocí knihovny Qt, stejně jako prototyp serveru. Server je realizován jako součást existujícího simulačního jádra (SmallDEVS), které je implementováno v jazyce SmallTalk.

Abstract

This thesis describes the design and implementation of an client-server application. This application is used to remote access to models of systems, which are saved on the server. Application also provides editation of the models and their simulation. In the thesis there is a design of Communication Protocol between the client and server too. For the implementation of the client and prototype of the server was used Qt library. Server is realized as a part of existing simulation core (SmallDEVS), which is implemented by SmallTalk.

Klíčová slova

DEVS, simulace, Qt, C++, Vysokoúrovňové Petriho sítě, Smalltalk

Keywords

DEVS, simulation, Qt, C++, High-Level Petri's nets, Smalltalk

Citace

Jan Kolařík: Aplikace pro vzdálenou editaci DEVS modelů a řízení simulace na simulačním serveru, diplomová práce, Brno, FIT VUT v Brně, 2013

Aplikace pro vzdálenou editaci DEVS modelů a řízení simulace na simulačním serveru

Prohlášení

Prohlašuji, že jsem tuto semestrální práci vypracoval samostatně pod vedením pana Doc. Ing. Vladimíra Janouška, Ph.D.

.....
Jan Kolařík
21. května 2013

Poděkování

Děkuji svému vedoucímu Doc. Ing. Vladimíru Janouškovi, Ph.D. za trpělivé a příkladné vedení při řešení problematiky, kterou se tato práce zabývá.

Obsah

Kapitola 1

Úvod

V dnešní době je progresivní pokrok vidět ve všech oblastech lidského snažení. Z celkového vývoje se však k běžným uživatelům většina novinek dostane až po určité, občas i několik let trvající, prodlevě. To je dáno jednak tím, že spousta oblastí je regulována státem prostřednictvím různých právních předpisů, které musí všechny produkty dodržovat, ale také tím, že není vždy jednoduché sehnat investora, který je ochoten financovat rychlé uvedení produktů, které zatím existují v teoretické rovině, na trh. Velkým úskalím je ale i to, že teoretické předpoklady nemusí vždy plně fungovat i v praxi. A toto je přesně ta fáze vývoje, kdy je správný čas přistoupit k simulaci systému. Velkou výhodou simulací je, že lze bez větších nákladů ověřit, zda systém funguje, aniž by bylo vůbec nutné investovat do vytvoření prototypu produktu. Simulace jako taková ale nenabízí pouze ověření toho, že bude systém fungovat, ale při jejím běhu jsou sbírána i jinak užitečná data, která na poli teoretickém jsou buďto těžko zjistitelná, nebo dokonce i nezjistitelná. Na základě těchto dat pak dochází i k postupným úpravám v modelu tak, aby co nejlépe odpovídal praktickým potřebám.

Tradiční pohled na software a vše, co s jeho vývojem souvisí, je obsahem softwarového inženýrství. Tento pohled na věc však není vždy dostačující. Softwarové inženýrství se zabývá převážně návrhem systému, jeho rozdělením do modulů, ale méně bere v potaz situace, které mohou nastat po spuštění systému. Příkladem může být řízení nezastavitelných technologických procesů, jako je třeba elektrárna, přehrada... Tyto systémy není možné zastavit jenom kvůli výměně některého prvku systému. Výměnu je nutné řešit přímo za běhu. Proto je nutné nový prvek systému nechat běžet až od stavu, ve kterém byl v systému vyměněn. Takové systémy jsou většinou realizovány jako distribuované a na každý prvek je nahlíženo jako na uzel. Distribuovaný systém je schopen se s výpadkem, změnou a následonou rekonfigurací určitého uzlu vyrovnat za běhu.

Tato práce se zabývá vytvořením aplikace typu klient-server, kde server zpřístupňuje simulační zdroje (modely, samotnou simulaci) pro klienta skrze síť. Uvažována je klasická komunikace založená na TCP. Jádro serveru je napsáno v jazyce Smalltalk a simulaci provádí autonomně. Klientská aplikace funguje jako rozhraní pro přístup do simulací a zobrazuje jejich aktuální stav. U klienta se dále předpokládá možnost komunikace s více servery současně a také možnost přesunu jednotlivých modelů mezi servery. Samotná simulace a popis jednotlivých modelů jsou založeny na DEVS a vysokoúrovňových Petriho sítích.

Kapitola 2

Systémy s diskrétními událostmi

V této kapitole jsou nastíněny principy systémů s diskrétními událostmi, které jsou také vhodnou reprezentací počítačových systémů. Každý dynamický systém (tj. systém, u kterého má smysl zkoumat jeho chování) je možné modelovat pomocí systému diskrétních událostí.

2.1 Znalosti systému

Klir[4] definuje rámec, na němž bývají systémy studovány, na základě čtyř hlavních úrovní znalosti daného systému. Každá úroveň zahrnuje i znalosti úrovní nižších.

- *Úroveň zdrojová* - jde o nejnižší úroveň, kde je zkoumána pouze množina proměnných, které jsou pro daný systém podstatné.
- *Úroveň datová* - zahrnuje temporální vývoj proměnných reprezentovaných pomocí časových řad.
- *Úroveň chování* - obsahuje informace o vztazích mezi historiemi jednotlivých proměnných. Tyto systémy jsou schopné generovat časové řady a proto jsou také známé jako generativní systémy.
- *Úroveň struktury* - obsahuje znalosti o subsystémech systému a struktuře jejich vzájemného propojení.
- Tato hierarchie je uzavřena pomocí páté úrovně, a to *úrovně metasystémů*. Tato úroveň obsahuje informace o tom, jak se datové, generativní a strukturální systémy vyvíjejí v čase.

2.2 Systém s diskrétními událostmi

Neformálně lze systém s diskrétními událostmi popsat následovně[3]:

- Systém má vstupy a výstupy pozorovatelné jako události.
- Na některé vstupy reaguje výstupy, a to buď okamžitě, nebo se zpožděním.
- Může dojít ke generování výstupu bez vnější příčiny.
- Uvnitř systém přechází mezi vnitřními stavy, a to když:

  - uplyne určitý čas, který stráví ve vnitřním stavu,
  - nebo když reaguje na přijetí vnější události.

- Výstup závisí pouze na aktuálním stavu systému a je pozorován jako svévolný přechod do stavu následujícího.

2.3 Čas v diskrétním systému

Základním pojmem spojeným s dynamickým systémem je čas, který je chápán jako nezávislá veličina. Čas systému je definován jako[12]:

$$time = (T, <)$$

- T - množina času;
- $<$ - antisymetrická, ireflexivní a tranzitivní relace uspořádání nad množinou T.

Relace $<$ je typicky definována jako lineární uspořádání. Jsou ale případy, kdy je vhodné pracovat i s částečným uspořádáním, kvůli modelování neurčitosti v systémech. Časová množina T je zpravidla definována jako $T = \mathbb{R}_0^+$, což určuje systém se spojitými událostmi. Další možností je T definovat jako $T = \mathbb{N}$, v tomto případě jde o diskrétní čas. Nad T definujeme následující intervaly:

$$(t_1, t_2) = \{\tau | t_1 < \tau < t_2, \tau \in T\}$$

$$< t_1, t_2 >= \{\tau | t_1 \leq \tau \leq t_2, \tau \in T\}$$

$$(t_1, t_2 >= \{\tau | t_1 < \tau \leq t_2, \tau \in T\}$$

Zápisem $T_{<t_1,t_2>}$ potom označujeme libovolný čas z intervalu $< t_1, t_2 >$.

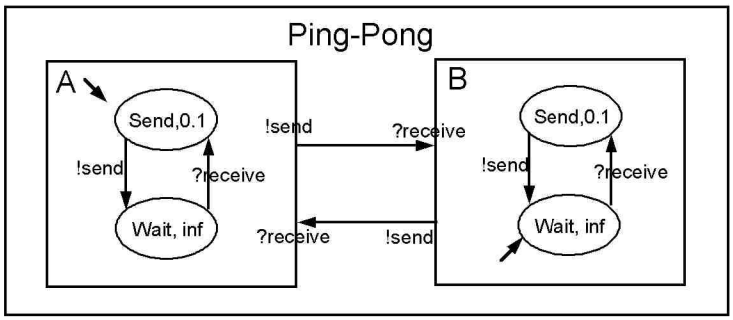
Kapitola 3

DEVS

DEVS (*Discrete Event System Specification*) je formalismus pro modelování a analýzu systémů s diskrétními událostmi. DEVS vymyslel Dr. Bernard P. Zeigler na Arizonské univerzitě. DEVS může být chápán jako rozšíření Moorova konečného automatu, kde je konečný počet stavů, a výstup je určen na základě předchozích stavů. Výstup tedy nezávisí na množině aktuálních vstupů.

3.1 Formalismus

DEVS definuje jak chování systému, tak i strukturu systému samotného. Chování systému je určeno událostmi vstup/výstup. Následující obrázek je modelem systému hry ping-pong:



Obrázek 3.1: Ping pong vyjádřený pomocí DEVS [10].

Vstupní událost je *?receive* a výstupní je *!send*. Oba hráči se mohou nacházet v těchto stavech:

- *Send* - V tomto stavu hráč setrvá 0,1 sekundu, než odpálí míček zpět přes síťku. Odpálení je reprezentováno událostí *!send*.
- *Wait* - Hráč setrvává v tomto stavu dokud k němu nedoletí míček od protihráče. Příjem míčku je reprezentován událostí *?recieve*.

Strukturu ping-pongu tvoří propojení dvou hráčů, A a B, přičemž výstupní údálost *!send* hráče A je přivedena a převedena na vstupní událost hráče B *?recieve*, a naopak. Samozřejmě je nezbytné, aby se hráči A a B na začátku simulace nacházeli v opačných stavech (jeden Send, druhý Wait). Jinak by došlo k uváznutí (deadlock).

3.2 Atomický DEVS

Atomický DEVS popisuje na základě vstupů a výstupů chování systému. Atomický DEVS je definován jako sedmice[12]:

$$M = (X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta)$$

X je *množina vstupních událostí,*

S je *množina stavů,*

Y je *množina výstupních událostí,*

$\delta_{int} : S \longrightarrow S$ je *interní přechodová funkce,*

$\delta_{ext} : Q \times X \longrightarrow S$ je *externí přechodová funkce,*

$Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$ je *množina úplných stavů,*
e je *čas uplynulý od poslední události,*

$\lambda : S \longrightarrow Y$ je *výstupní funkce,*

$ta : S \longrightarrow R_0^+ \cup \{\infty\}$ je *funkce posunu času.*

3.2.1 Výklad formalismu

Systém se nachází v daném čase ve stavu $s \in S$. Maximální čas setrvání v tomto stavu určuje funkce $ta(s)$. Zvláštním případem je $ta(s) = \infty$, což značí, že je systém pasivní a nikdy tento stav neopustí[3].

- Pokud nepřijde žádná externí událost, systém setrvává v tomto stavu s po dobu $ta(s)$. Jakmile je splněno, že uplynulý čas $e = ta(s)$, provede se zápis $\lambda(s)$ na výstup, a stav systému se změní na $\delta_{int}(s)$.
- Pokud se vyskytne externí událost $x \in X$ v čase $e \leq ta(s)$, systém se přepne do stavu $\delta_{ext}(s, e, x)$. Systém generuje výstup pouze ve chvíli, kdy platí $e = ta(s)$.

Při časovém konfliktu interního a externího přechodu se provádí pouze přechod externí.

3.3 Složený model

Jak již bylo naznačeno, tak DEVS spojuje více atomických modelů (Atomic DEVS) do komplexnějších modelů. Tyto modely jsou označovány jako složené modely (Coupled DEVS). Jedná se tedy o hierarchické spojení několika atomických modelů a pro komunikaci s okolím jsou stanoveny nové vstupy a výstupy. Stav složeného modelu (systému) je determinován na základě stavů všech jeho subsystémů. Složením a propojením systémů znovu vzniká systém. Složený model je definován následovně[12]:

$$N_{self} = (X, Y, D, \{M_d\}, \{I_d\}, \{Z_{i,d}\}, select)$$

X je *množina vstupních událostí,*

Y je *množina výstupních událostí,*

D je *množina jmen submodelů,*

$\{M_d | d \in D\}$ je *množina submodelů,*

$\{I_d | d \in D \cup \{selft\}\}$ je *specifikace propojení,*

$$\forall d \in D \cup self : I_d \subseteq D \cup \{self\}, d \notin I_d$$

$\{Z_{i,d} | i \ \in I_d, d \in D \cup \{selft\}\}$ je *specifikace překladu událostí*

$$Z_{i,d} : X \longrightarrow X_d \text{ pro } i = selft,$$
$$Z_{i,d} : Y_i \longrightarrow Y \text{ pro } d = self,$$
$$Z_{i,d} : Y_i \longrightarrow X_d \text{ pro } i \neq self \wedge d \neq selft,$$

$select : 2^D - \{\} \longrightarrow D$ je *preferenční funkce.*

3.3.1 Výklad formalismu

I_d je pro každý model d množina jmen všech modelů, pro které platí, že jejich výstupy jsou spojeny se vstupem modelu d. $Selft$ označuje složený model N. $Z_{i,d}$ pro každý model i, který je připojen na vstup modelu d, definuje překlad výstupních událostí modelu i na vstupní události modelu d. Funkce *select* slouží k určení toho, který submodel se bude vykonávat v případě konfliktu interních přechodů (pokud je více submodelů připraveno provést interní přechod)[12].

3.4 Porty a stavové proměnné

Množiny interních stavů a množiny vstupních a výstupních událostí se obvykle specifikují jako strukturované množiny. Tato skutečnost nám dovoluje používat libovolný, ale konečný počet vstupních a výstupních portů, a stavových proměnných. Strukturovaná množina

$$S = (V, S_1 \times S_2 \times \ldots \times S_n)$$

je definována pomocí množiny proměnných V, kde $|V| = n$, a kartézským součinem množin hodnot jednotlivých proměnných[3].

Kapitola 4

Petriho sítě

Petriho sítě (Petri's nets, také PN) je označení pro širokou skupinu matematických diskrétních modelů. Ty umožňují popisování informačních závislostí a řídících toků uvnitř modelovaného systému pomocí speciálních prostředků. Petriho sítě byly poprvé zveřejněny německým matematikem C. A. Petri v roce 1962. [13]

4.1 Definice Petriho sítě

Petriho síť je definována jako pětice[13]:

$$N = (P, T, F, W, M_0)$$

- P je konečná množina *míst*,
- T je konečná množina *přechodů*,
- F je konečná množina *hran*: $F \subseteq (T \times S) \cup (S \times T)$,
- $W : F \longrightarrow \mathbb{N} \setminus \{0\}$ je *ohodnocení* hran grafu určující kladnou váhu každé hrany sítě,
- $M_0 : P \longrightarrow \mathbb{N}$ je *počáteční značení* míst Petriho sítě.

4.2 Vysokoúrovňové Petriho sítě

Z teoretického hlediska je možné pomocí klasických Petriho sítí prezentovat jakýkoli algoritmický problém, ale pouze na nízké úrovni abstrakce. Existují ovšem momenty, kdy je nutno vytvořit podrobnější model, a nelze tedy využít jen hrubou abstrakci. I jednoduché věci, jako například provádění aritmetických operací, modeluje nízkoúrovňový model, který je vytvořený prostřednictvím nízkého stupně abstrakce, příliš podrobně. Analogií k tomuto může být představa, že bychom běžné konstrukce programovacích jazyků programovali přímo pomocí instrukcí procesoru. A právě proto se tyto sítě nehodí pro detailní modelování reálných systémů[3].

4.2.1 Multimnožina

Pro vysvětlení a pochopení vysokoúrovňových Petriho sítí (High-Level Petri Nets, také HL-sítě) musíme představit pojem multimnožina. Jedná se o zobecnění množiny. Nebo je také

možné chápat množinu jako speciální případ multimnožiny. Rozdíl mezi množinou a multimnožinou je v tom, že multimnožina připouští vícenásobný výskyt jednoho prvku.[13]

Význam symbolů $\in$, $\subset$ a $\subseteq$ je obdobný jako u množin:

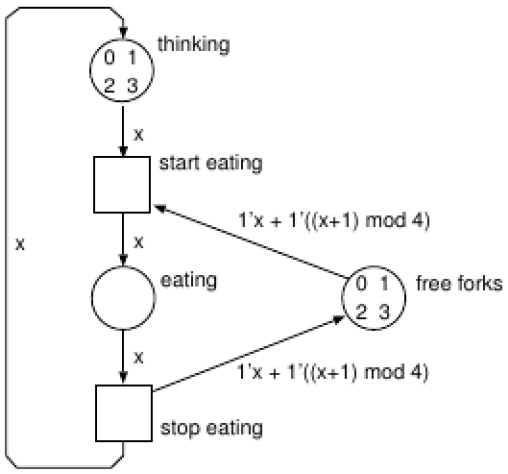
- $a \in A$ značí, že v multimnožině A je prvek a obsažen alespoň jednou.
- $A \subset B$ značí totéž, jako v případě množin, tedy $A \subseteq B \wedge A \neq B$.
- $A \subseteq B$ značí, že všechny prvky multimnožiny A jsou obsažené v multimnožině B a to ve stejném anebo větším počtu.

4.3 Popis vysokoúrovňové Petriho sítě

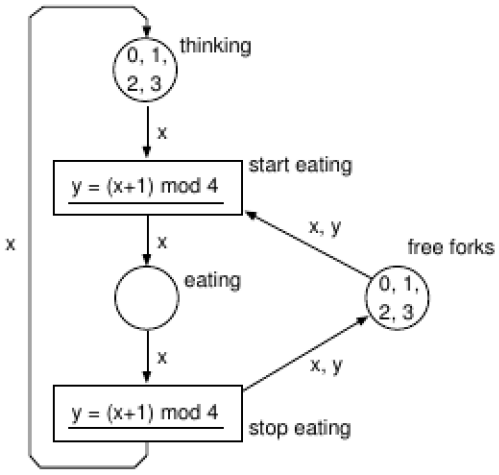
Vysokoúrovňovou Petriho síť je možné charakterizovat takto:

- Jednotlivá místa sítě obsahují značky.
- Každé hraně sítě je přiřazena multimnožina, která obsahuje obecné výrazy. Těmito výrazy mohou být značky, které chce přechod odebrat ze vstupních míst nebo je chce umístit do výstupních míst, nebo libovolná funkce. Příklad zápisu funkce je na obrázku 4.1, který modeluje problém večeřících filosofů. Každý filosof a vidlička jsou určeni přirozeným číslem. Jestliže filosof je definován jako x, potom používá vidličky x a *(x+1) mod 4*. Zápis *1'x + 1'((x + 1) mod 4* určuje multimnožinu, která obsahuje právě jeden prvek x a jeden prvek *(x + 1) mod 4* (vidlička).
- Jednotlivé přechody mohou obsahovat strážní výrazy (guards). Jsou to výrazy Boolovského typu (predikáty), které určují dodatečnou podmínku pro provedení přechodu.

Na obrázku 4.2 je zjednodušená verze zápisu vysokoúrovňové sítě. Je zde dovoleno nahradit + ,. Výraz x, y pak značí multimnožinu s jedním prvkem x a jedním prvkem y. Dalším zjednodušením je možnost přesunutí složitějších výpočtů přímo do stráže přechodu. [3]



Obrázek 4.1: Čtyři večeřící filosofové.[3]



Obrázek 4.2: Čtyři večeřící filosofové po zjednodušení.[3]

Kapitola 5

Vývojové prostředí

V této kapitole bude představen framework Qt a jazyk Smalltalk, jejich stručná historie a základní principy.

5.1 Qt

5.1.1 Historie

Na první verzi Qt se začalo pracovat v roce 1991 a od té doby neustále dochází k jeho zlepšování. Asi o tři roky později si dva z vývojářů založili společnost Trolltech, která ve vývoji pokračovala. V roce 2008 firmu odkoupila společnost Nokia a krátce poté byla přejmenována na Qt software. Nokia se sporadickými úspěchy nasadila Qt do svých nových OS Maemo a Meego. V dnešní době se jedná o mrtvé platformy[2].

Qt bylo od svého vzniku používáno také pro opensource aplikace, a je třeba využito v linuxovém grafickém prostředí KDE, nebo ve webovém prohlížeči Opera.

5.1.2 Popis

Knihovna je vystavěna nad C++ a tudíž se jedná o zcela multiplatformní knihovnu. Není tedy problém přenášet programy mezi Linuxem, Macem, Windows, Unixem a dalšími. Program pouze stačí pro danou platformu zkompilovat a vše funguje[2].

Knihovna je opravdu velice komplexní a poskytuje programátorovi veškerou funkcionalitu, jakou v dnešní době může potřebovat. Ke knihovně jsou dodávány i pomocné programy:

- QtCreator - Vývojové prostředí pro programování v této knihovně. Jedná se o intuitivní program, který oproti jiným vývojovým prostředím (NetBeans, Eclipse) využívá zlomek systémových zdrojů. Jediné, co se dá nástroji vytknout, je horší podpora refaktorizace.
- QtDesigner - Nástroj pro vizuální tvorbu grafického prostředí.
- QtLinguist - Program pro usnadnění jazykových lokalizací vytvořených programů.

5.2 Smalltalk

5.2.1 Historie

Smalltalk[7] byl vyvinut ve výzkumném centru Xerox Palo Alto Research Center (dále PARC) skupinou výzkumníků okolo Alana Kaye. Návrh systému Alana Kaye implementoval Dan Ingalls a tato první verze byla posléze pojmenována jako Smalltalk-71. Syntaxe i běh této a následující verze Smalltalk-72 byly stále velmi odlišné od verze současné.

První verzí, která opustila PARC byl Smalltalk-80 Version 1, kterou dostali k otestování vybrané společnosti jako Hewlett-Packard, Apple Computer... Smalltalk-80 Version 2 už byla vydaná jako tzv. image soubor a specifikace virtuálního stroje.

V současnosti jsou velmi populární komerční VisualAge Smalltalk od firmy IBM a Squeak pod licencí MIT. Squeak byl použit i v této práci.

5.2.2 Popis

Smalltalk[5] patří do kategorie jazyků, které jsou překládány do bytecodu a poté jsou pomocí virutálního stroje interpretovány. Jazyk je postaven na zásobníkové architektuře. Smalltak je čistě objektově orientovaný jazyk a všechno v něm jsou objekty. Díky tomu může být vše ukládáno stejným způsobem, a to v objektové paměti. Objekty uložené v paměti jsou definované pomocí ukazatele na sebe a seznamem hodnot proměnných, které obsahují. Většinou se jedná o ukazatele na jiné objekty. O odstranění nepotřebných objektů se garbage collector. Nad pamětí je prováděno pět operací, které využívá interpret. Jedná se o následující operace. Přístup k instančním proměnným, změna jejich hodnot, zjištění počtu instančních proměnných, přístup k ukazateli na třídu daného objektu a vytvoření nového objektu.

Jazyk smalltalk se skládá ze dvou částí. První je virtuální stroj a druhou je obraz (image), což je objektová paměť. Smalltalk je napsán sám v sobě. Editor, ve kterém se editují kódy, je napsán ve Smalltalku, a celé grafické prostředí, ve kterém se pracuje, také. Přestože Smalltalk běží jenom v jednom vlákně, umožňuje multitasking. Tato vlastnost je zajištěna díky tomu, že Smalltalk implementuje vlastní plánovač procesů.

Díky svým základním vlastnostem, jako je překlad do bytecodu a jeho interpretace, Smalltalk umožňuje snadnou přenositelnost. Stačí pouze přenést obraz objektové paměti na cílovou platformu a nainstalovat interpret bytecodu pro cílovou platformu.

Další výhodou jazyka je to, že programátor může nahlížet do všech zdrojových kódů, ze kterých je Smalltalk sestaven. Tyto kódy může libovolně modifikovat, dědit, nebo i smazat. Veškeré změny se provádí za běhu, bez nutnosti restartování.

Kapitola 6

Návrh aplikace

V této kapitole bude popsána specifikace aplikace, návrh aplikace a protokolu, na kterém se realizuje komunikace mezi klientem a serverem. Dále bude představen současný simulační systém SmallDEVS.

6.1 Specifikace aplikace

Účelem aplikace je zpřístupňovat modely a simulace ze vzdálených serverů a umožňovat uživateli manipulovat s nimi. Aplikace je navržena na principu komunikace klient-server. Server běží na počítači, a po připojení klienta mu umožní stáhnout si potřebná data (seznam modelů, obsah ostatních složek ...). Všechny změny na klientské straně se na server odesílají buď po ručním uložení uživatelem (tlačítko v menu, nebo tlačítko v dialogu), nebo okamžitě při provedení akce. Takovou akcí je například vytvoření nového byť i prázdného modelu, nebo přejmenování či smazání položky stromu.

Klient aplikace je napsán v Qt a proto je bez problému přenositelný na jiné platformy. Stačí jej pouze překompilovat pro cílovou platformu. Server je implementován v jazyce Smalltalk, stejně jako SmallDEVS. Přenos je taktéž bezproblémový. Program splňuje následující podmínky:

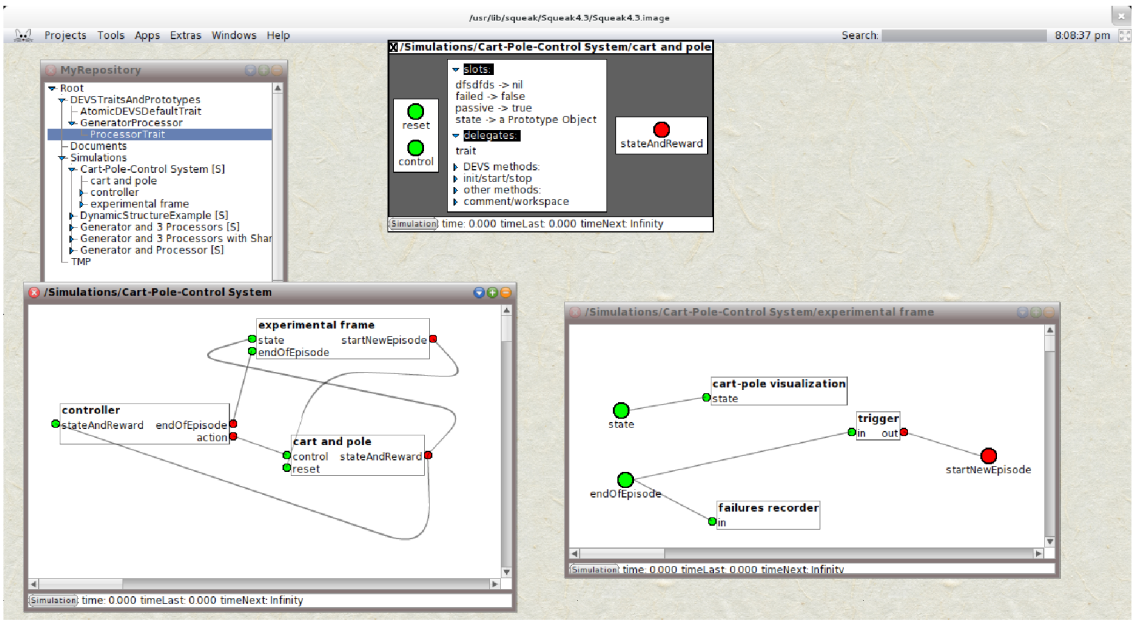
- Klient je multiplatformní.
- Klientská aplikace umožňuje kombinovat modely z více serverů.
- Klient umožňuje definovat atomy pomocí DEVS, nebo vysokoúrovňové Petriho sítě.
- Server zasílá klientovi na vyžádání stavy běžících simulací.
- Server je konkurentní.
- Server prosazuje politiku zámků pro zamezení nechtěného přepisování položek.

6.2 SmallDEVS

SmallDEVS je nová a odlehčená implementace DEVS formalismu ve Smalltalku a slouží pro výukové a výzkumné účely. Dovoluje experimentovat s:

- prototypově zaměřenou a objektově orientovanou konstrukcí modelů,

- interaktivním modelováním a simulacemi,
- vícenásobnými simulacemi a reflektivními simulacemi.



Obrázek 6.1: SmallDEVS verze 2012.

6.3 Protokol

Protokol komunikace je navržen na základně vnitřní struktury SmallDEVS[1] a potřeb pro realizaci spojení mezi klientem a serverem. Protokol také implementuje zámky, aby nedocházelo ke dvojímu zápisu do téhož prvku stromu. Struktura zprávy je následující:

příkaz data

Seznam příkazů je uveden v 6.3.2. Za příkazem musí být mezera, která slouží jako oddělovač příkazu a dat. Ne u všech příkazů jsou data povinná. Až na výjimku jsou data XML struktura, která reprezentuje stromovou strukturu uloženou na serveru. V případě změny stavu, nebo dotazování na daný prvek, vypadají data určující cestu k prvku například takto:

```
<myRepository>
  <root name="Root">
    <folder name="Simulations">
      <Simulation name="Cart-Pole-Control System" />
    </folder>
  </root>
</myRepository>
```

Pomocí tohoto XML dotazu říkáme, že chceme ve stromu přistoupit k prvku typu simulace, který se nachází v cestě */Root/Simulations/Cart-Pole-Control System*.

6.3.1 Specifikace XML souboru

Formální popis posloupnosti značek v XML[9] souboru je popsán pomocí EBNF[1]. Popis je omezen pouze na zanoření jednotlivých značek a nikoli na popis přesných atributů. Používané atributy u jednotlivých značek jsou popsány v průběhu popisování jednotlivých modelů a elementů:

```
document = "<myRepository>", rootElem, "</myRepository>" ;
rootElem = "<root>", {folderElem}, prototypeFolderTopElem "</root>" ;
folderElem = "<folder>",
               {folderElem} | {simulationElem} | {fileElem},
             "</folder>" ;
prototypeFolderTopElem = "<folder>",
                           {prototypeFolderElem},
                         "</folder>";
prototypeFolderElem = "<folder>",
                        {prototypeElem} | {prototypeFolderElem},
                      "</folder>";
prototypeElem = "<prototypeObject>"
                  slotsElem,
                  delegatesElem,
                  methodsElem,
                "</prototypeObject>" ;
simulatinElem = "<simulation>",
                  "<settings>",
                    simulationSettings,
                    coupledDEVSSettings
                  "</settings>", {modelElem},"</simulation>" ;

modelElem = PNAtomElem | atomicDEVSElem | coupledDEVSElem ;
PNAtomElem = "<PNAtom>",
               placesElem,
               transitionsElem,
             "</PNAtom>" ;
placesElem = "<places>",
               {placeElem},
             "</places>";
placeElem = "<place />" ;
placesElem = "<transitions>",
               {transitionElem},
             "</transitions>";
transitionElem = "<transition>",
                   "<guard>",String,"</guard>",
                   "<action>"String,"</action>",
                   "<preConds>",{condElem},"</preConds>",
                   "<postConds>",{condElem},"</postConds>",
                   "<conds>",{condElem},"</conds>",
                 "</transition>" ;
condElem = "<cond />";
atomicDEVSElem = "<atomicDEVS>",
                   inputPortsElem,
                   outputPortsElem,
                   settingsAtomicDEVSElem,
```

[1] Extended Backus-Naur Form - jedná se o matematický popis syntaxe programovacích jazyků

```
                    "</atomicDEVS>" ;
slotsElem = "<slots>",
                  {slotElem},
                "</slots>" ;
delegatesElem = "<delegates>",
                    {delegateElem},
                  "</delegates>" ;
methodsElem = "<methods>",
                  {method},
                "</methods>";
settingsAtomicDEVSElem = "<settings>",
                              slotsElem,
                              delegateselem,
                              "<initStartStop>",
                                {methodElem},
                              "</initStartStop>",
                                {methodElem},
                              "<DEVSMethods>",
                                {methodElem},
                              "</DEVSMethods>",
                              "<otherMethods>",
                                {methodElem},
                              "</otherMethods>",
                              "<comment>",
                                 String
                              "</comment>"
                            ,"</settings>" ;
slotElem = "<slot />" ;
delegateElem = "<delegate />" ;
methodElem = "<method>",methodBodyElem,"</method>" ;
methodBodyElem = String ;
coupledDEVSElem = "<settings>",
                     coupledDEVSSettings,
                   "</settings>", {modelElem} ;
coupledDEVSSettings = inputPortsElem, outputPortsElem,
                      modelsElem, interconnectionsElem;
modelsElem = "<models>",{modelElem},"</models>" ;
modelElem = "<model/>" ;
interconnectionsElem = "<interconnections>",
                         {connectinoElem},
                       "</interconnections>" ;
connectinoElem = "<connection />" ;
inputPortsElem = "<inputPorts>",{portElem},"</intputPorts>" ;
simulationSettings = "<rtFactor/>","<isRunning/>",
                     "<stopTime/>","<timeLast/>",
                     "<timeNext/>","<time/>","<log/>"
outputPortsElem = "<outputPorts>",{portElem},"</outputPorts>" ;
portElem = "<port/>" ;
```

6.3.2 Seznam příkazů

- **SEED** *náhodné hexadecimální číslo*
  Po připojení klienta k serveru zasílá server klientovi tento příkaz. Parametrem příkazu

je náhodně vygenerovaný řetězec pro dané spojení. Tento řetězec se využívá při šifrování přihlašovacích údajů. Více bude popsáno v implementaci.

- **AUTH** *xml*
  Na příkaz **SEED** odpovídá klient tímto voláním, parametrem jsou šifrované přihlašovací údaje.

- **AUTH_FAIL**
  Na příkaz *AUTH* může server odpovědět tímto příkazem, který značí, že uživatel zadal chybné údaje. Po odeslání tohoto příkazu je spojení uzavřeno.

- **AUTH_OK**
  Autentizace klienta proběhla v pořádku.

- **GET_MY_REPOSITORY**
  Po úspěšném přihlášení zasílá klient serveru tuto zprávu. Její bližší význam je popsán v 6.3.3.

- **LOCK** *XML*
  Žádost od klienta na uzamknutí položky stromu. Parametrem je XML soubor, obsahující cestu k zamykanému elementu.

- **UNLOCK** *XML*
  Žádost o odemčení položky určené parametrem.

- **LOCKED**
  Potvrzení od serveru, že uzamčení proběhlo v pořádku.

- **UNLOCKED**
  Potvrzení, že odemčení bylo úspěšné.

- **IS_LOCKED** *XML*
  Informace o tom, že položka určená parametrem byla odemčena klientem, který jí zmkl, nebo že došlo ke zrušení zámku (ukončení spojení s vlastníkem zámku).

- **IS_UNLOCKED** *XML*
  Pokud byla položka určená parametrem úspěšně uzamčena, jsou o tom informováni ostatní klienti.

- **UPDATE_MY_REPOSITORY** *XML*
  Příkaz pro změnu obsahu podstromu na straně klienta. Příkaz je generován serverem pro zrušení (odemčení) zámku. Zpráva je zaslána všem klientům kromě toho, který vlastnil zámek.

- **GET_UPDATE_FROM** *XML*
  Požadavek klienta na zaslání daného podstromu.

- **SIMULATION_IS_RUNNING** *XML*
  Příkaz informující klienty o tom, že simulace určená parametrem byla spuštěna.

- **SIMULATION_IS_STOPPED** *XML*
  Simulace určená parametrem byla zastavena.

- **SIMULATION_UPDATE** *XML*
  Příkaz informující o změně stavových informací o simulaci určené parametrem. Zpráva je zasílána všem klientům, kteří jsou na serveru registrováni k příjmu změn.
- **START_SIMULATION** *XML*
  Požadavek klienta na spuštění simulace.
- **RESTART_SIMULATION** *XML*
  Požadavek klienta na restartování dané simulace.
- **STOP_SIMULATION** *XML*
  Požadavek klienta na zastavení běhu simulace.
- **ADD_TO_LISTENERS** *XML*
  Požadavek klienta na zasílání změn stavu simulace určené parametrem.
- **REMOVE_FROM_LISTENERS** *XML*
  Požadavek klienta na zrušení zasílání změn stavu simulace určené parametrem.
- **CREATE_ATOMIC_DEVS** *XML*
  Příkaz na vytvoření nového AtomicDEVS určeného cestou v parametru.
- **CREATE_PN_ATOM** *XML*
  Požadavek klienta na vytvoření nového atomu definovaného pomocí vysokoúrovňové Petriho sítě.
- **CREATE_COUPLED_DEVS** *XML*
  Příkaz pro vytvoření nového spojovaného modelu (Coupled DEVS).
- **CREATE_SIMULATION** *XML*
  Požadavek na vytvoření nové simulace určené parametrem.
- **CREATE_PROTOTYPE** *XML*
  Příkaz pro vytvoření nového rysu používaného u AtomicDEVS.
- **CREATE_FOLDER** *XML*
  Požadavek na vytvoření nové složky určené parametrem.
- **CREATE_FILE** *XML*
  Požadavek na vytvoření nového souboru určeného parametrem.
- **UPDATE_ATOMIC_DEVS** *XML*
  Příkaz pro upravení atomického DEVS určeného parametrem. Parametr také obsahuje požadované změny. Elementy používané pro popis atomického DEVSu jsou uvedeny v 6.2
- **UPDATE_PN_ATOM** *XML*
  Příkaz pro upravení atomu definovaného pomocí vysokoúrovňové Petriho sítě. Elementy používané pro popis Petriho sítě jsou uvedeny v 6.3.5
- **UPDATE_COUPLED_DEVS** *XML*
  Požadavek na upravení spojovaného modelu určeného parametrem. Jednotlivé elementy sloužící k jeho popisu jsou uvedeny v 6.3.6.

- **UPDATE_SIMULATION** *XML*
  Požadavek na upravení nastavení simulace. Elementy sloužící k popisu nastavení jsou v 6.3.7.

- **UPDATE_PROTOTYPE** *XML*
  Požadavek na upravení rysu atomického DEVS.

- **UPDATE_TREE** *XML*
  Příkaz pro přejmenování daného prvku stromu. Pod přejmenovávaný prvek je vložena značka *rename* s atributem *name*, který určuje nové jméno prvku.

```
UPDATE_TREE <myRepository>
  <root name="Root">
    <folder name="Simulations">
      <Simulation name="Cart-Pole-Control System">
        <CoupledDEVS name="controller>
          <rename name="newController" />
        </CoupledDEVS>
      </Simulation>
    </folder>
  </root>
</myRepository>
```

  Například tento požadavek značí, že se má přejmenovat spojovaný model jménem *controller* na *newController*

- **DELETE** *XML*
  Požadavek na smazání prvku určeného parametrem.

6.3.3 Získání stromu modelů

Celá struktura modelů a ostatních objektů uložených na serveru je realizována pomocí stromu. Po připojení klienta k serveru zasílá klient serveru požadavek ***GET_MY_REPOSITORY***, na který server odpovídá XML souborem, který obsahuje strukturu stromu a obsah jednotlivých modelů. Značky, které se mohou vyskytnout v inicializačním souboru jsou uvedené v tabulce 6.1.

6.3.4 Popis atomic DEVS

Samotná definice atomic DEVS je uvozena, jak bylo uvedeno výše, značkou *atomicDEVS*. Definice vychází z implementace atomic DEVS dle SmallDEVS. Výčet jednotlivých značek souboru je uveden v tabulce 6.2.

Provádění změn atomic DEVS

Při dokončení změn modelu je vygenerován příslušnými třídami XML soubor, který obsahuje popis jednotlivých změn. Na server je poslán příkaz ve formátu ***UPDATE_ATOMIC_DEVS***, mezera, XML soubor se změnami.

Změna konfigurace vstupních nebo výstupních portů se provádí pomocí elementů:

- *add* - pro přidání nového portu, jako argument má *name*, určující jméno nového portu.

Jméno značky	Popis
root	Jedná se o kořenový prvek celého dokumentu, musí obsahovat atribut *name*, který udává jméno kořene.
folder	Označuje složku, která může obsahovat buď samotné modely, nebo další složky. Povinný argument je *name*, značící jméno složky.
file	Označuje složku, která reprezentuje textový soubor. Povinný argument je *name*, značící jméno souboru. Obsahem značky je obsah souboru.
simulation	Značka určuje, že se jedná o simulaci. Simulace se do sebe nemohou zanořovat. Simulace je vlastně spojovaný model DEVS, jenom může navíc provádět simulaci samotnou.
atomicDEVS	Uvozuje blok definující jeden atomic DEVS model. Jako atribut má *name*, určující jméno modelu. Značky popisující model budou popsány dále.
PNAtom	Uvozuje sekci definující atomic DEVS popsaný pomocí vysokoúrovňové Petriho sítě.
coupledDEVS	Uvozuje sekci, která definuje spojovaný model. Značky, které může tento element obsahovat budou popsány dále.
prototypeObject	Uvozuje sekci, která definuje *Trait* používaný u atomicDEVS. Značky, které může tento element obsahovat budou popsány dále.

Tabulka 6.1: Popis položek pro získání stromu modelů

- *rename* - přejmenování portu. Má dva argumenty: *oldName* obsahuje staré jméno portu, *newName* pak nové jméno portu.
- *delete* - smaže port podle jména určeného argumentem *name*.

Změna nastavení metod se provádí také pomocí *add* a *delete*. Při změně obsahu se použije element *update* obsahující atribut *name* se jménem metody. Tělo metody je uloženo ve značce *update*. Změny ostatních nastavení, jako je nastavení slotů a delegací, se provádí analogicky k tomu postupu.

6.3.5 Popis vysokoúrovňové Petriho sítě

Samotná specifikace sítě je uvozena značkou *PNAtom*. Definice vychází z formální definice popsané výše. Každý prvek sítě (místo či přechod) je identifikován na základě čísla (id), které je v rámci sítě unikátní. Popis jednotlivých značek je popsán níže:

- *places*
  Uvozuje sekci s jednotlivými místy sítě. Každé místo je potom definováno značkou

Jméno značky	Popis
inputPorts	Uvozuje sekci se vstupními porty. Jednotlivé porty jsou definovány značkou *port* s atributem *name* pro určení jména portu. Porty se nemohou dále zanořovat do sebe.
outputPorts	Je podobná jako *inputPorts*, jen definuje výstupní porty.
settings	Uvozuje sekci se samotnou konfigurací modelu. Všechny další značky v této tabulce musí být zanořeny v tomto jednom elementu.
slots	V tomto elementu mohou být zanořeny značky *slot*, určující jednotlivé sloty. Atributy elementu musí obsahovat *name* (jméno slotu) a *value* (hodnotu slotu).
delegates	Obsahuje delegace, kdy každá delegace *delegate* obsahuje parametr *name*, určující její jméno.
DEVSMethods	Seznam předem definovaných jmen metod, se kterými pracuje smallDEVS. Jedná se o: *extTransition*, *outputFnc*, *intTransition*, *timeAdvance*. Jednotlivé metody jsou zanořeny v tomto elementu a uvozeny značkou *method*, která má jako povinný atribut *name*, které určuje jméno metody. Tělo metody je uloženo jako text uvnitř elementu *method*.
initStartStop	Obsahuje metody související s inicializací, spuštěním a zastavením modelu. Jsou to tyto metody: *initModel*,*prepareToStart*, *prepareToStop*. Zápis těchto metod je stejný jako u *DEVSMethods*.
otherMethods	Obsahuje ostatní metody modelu. Jednotlivé metody jsou pak definovány stejně jako bylo popsáno u *DEVSMethods*.
comment	Tento element obsahuje text komentáře daného modelu.

Tabulka 6.2: Seznam položek v elementu *settings*

place. Značka obsahuje povinný atribut *id*, který slouží k jednoznačné identifikaci místa a nepovinný *name*, obsahující jméno místa. Poslední dva atributy jsou x a y a slouží k určení pozice místa. Obsahem značky *place* je výchozí hodnota.

- *transitions*
  Tato značka uvozuje sekci s přechody. Jednotlivé přechody jsou potom uvozeny značkou *transition*, která obsahuje stejně jako značka *place* atributy *id, name, x, y*. Každý přechod potom obsahuje značky:
  - *guard* - stráž přechodu.
  - *action* - akce přechodu.
  - *preConds* - množina podmínek vstupů do přechodu (znázorněno šipkou od místa k přechodu).
  - *postConds* - množina podmínek výstupů z přechodu (znázorněno šipkou od přechodu k místu).
  - *conds* - množina podmínek pro vstup a výstup z přechodu (znázorněno oboustrannou šipkou mezi místem a přechodem).

  Jednotlivé podmínky obsahují značku *cond*, která definuje podmínku samotnou. Atributy podmínky jsou *id*, které určuje místo, ke kterému se podmínka vztahuje, a atribut *positions*, který obsahuje body spojnice. Značka *cond* obsahuje podmínku pro přechod.

Následující příklad znamená, že existuje síť jménem *PNAtom*, která obsahuje místo jménem *in* s identifikačním číslem 1. Toto místo se nachází na pozici $[10, 10]$ a obsahuje dvě značky *x*. Síť dále obsahuje přechod s číslem 2, který se jmenuje *transition*, a je na pozici $[50, 10]$. Akce přechodu je $x := x + 1.$, stráž přechodu je $x > 0.$. Přechod obsahuje jednu podmínku před přechodem z místa s číslem 1. Spojnice mezi tímto místem a přechodem prochází body $[15, 10], [20, 50], [50, 50]$.

```
<PNAtom name="PNAtom>
  <places>
    <place id="1" name="in" x="10" y="10">x x</place>
  </places>
  <transitions>
    <transition id="2" name="transition" x="50" y="10">
      <action>x := x + 1.</action>
      <guard>x > 0.</guard>
      <preConds>
        <cond id="1" positions="15@10 20@50 50@20">2‘x</cond>
      </preConds>
    </transition>
  </transitions>
</PNAtom>
```

Provádění změn vysokoúrovňové Petriho sítě

Provádění změn v sítích se velice podobá prováděním změn v atomic DEVS. Pro přidání místa nebo přechodu se používá značka *add*, kde je místo klíčového atributu *name* použit atribut *id*. Analogicky se provádí i ostatní operace. Oproti atomic DEVS je zde na úrovni jednotlivých přechodů a míst přidána značka *move* obsahující atribut *id* položky a atributy *x, y* pro změnu polohy prvku. Vytvořený XML soubor je poté zaslán na server ve tvaru *UPDATE_PN_ATOM XML*.

6.3.6 Popis spojovaného modelu

V popisu spojovaného modelu je nutné uvést jména modelů, ze kterých se skládá, a také uvést schéma propojení výstupů na vstupy. Aby mohl model komunikovat s jinými modely, je třeba zavést nové vstupní a výstupní porty nového (spojovaného) modelu. Samotný spojovaný model je uvozen značkou *coupledDEVS*, v níž jsou vloženy jednotlivé modely. Vlastnosti modelu jsou obsaženy ve značce *settings*. Popis značek reprezentujících spojovaný model je uveden v následujícím seznamu:

- *inputPorts*
  Obsahuje výčet vstupních portů. Každý port je poté specifikován značkou *inputPort*. Značka pak obsahuje atribut *name*, který označuje jméno portu a atributy x a y určující polohu portu.
- *outputPorts*
  Je analogická k *inputPorts*, jenom se záměnou "input" za "output".
- *models*
  Obsahuje seznam modelů, které náleží danému spojovanému modelu. Jednotlivé modely jsou uvozeny značkou *model* s atributem *name* obsahující jméno modelu, a atributy x a y určující pozici modelu.
- *interconnections*
  Uvozuje seznam jednotlivých propojení mezi modely. Značkou *connection* je potom uvozeno spojení. Atributy této značky jsou:
  - *fromModel* - určuje model ze kterého vychází propojení. Pokud je tento atribut prázdný, znamená to, že propojení pochází z aktuálního modelu.
  - *fromPort* - obsahuje jméno zdrojového portu.
  - *toModel* - určuje cílový model propojení. Pokud není jméno modelu zadáno, tak se jedná o aktuální model.
  - *toPort* - jméno cílového portu propojení.

Provádění změn spojovaného modelu

Příkaz pro provedení změny ve spojovaném modelu je `UPDATE_COUPLED_DEVS`. Změny ve vstupních a výstupních portech jsou stejné jako u atomic DEVS a změny v nastavení modelů se provádí na stejném principu. Úpravy jednotlivých propojení mezi modely jsou prováděny pomocí značek *add* a *delete*, přičemž u každé z nich musí být nastaveny všechny jejich atributy, které obsahuje značka *connection*.

6.3.7 Popis simulace

Základem simulace je spojovaný model. Do jeho značky *settings* jsou vloženy informace o simulaci. Níže je uveden popis značek simulace:

- *isRunning*
  Obsahuje atribut *value*, nabývající hodnoty true/false podle toho, zda simulace běží.
- *stopTime*
  V atributu *value* je uložen koncový čas simulace, nebo řetězec *Infinity*, značící, že simulace je nekonečná.

- *rtFactor*
  Atribut *value* obsahuje číslo určující rychlost simulace. V případě, že je atribut nulový, to znamená, že simulace běží jak nejrychleji může. Čím vyšších hodnot dosahuje, tím pomalejší je simulace.

- *timeLast*
  V atributu *value* je uložen čas předchozího přechodu.

- *timeNext*
  Atribut *value* obsahuje čas dalšího interního přechodu.

- *time*
  Atributem *value* je určen aktuální čas simulace.

- *log*
  Obsahem této značky je výpis simulace.

Změny v simulacích se provádí nastavením hodnot značek popisujících vlastnosti simulace. Příkaz pro změnu nastavení simulace je *UPDATE_SIMULATION*.

6.4 Návrh serveru

Server po připojení klienta provádí ověření, zda může uživatel vůbec přistupovat k modelům uloženým na serveru. Komunikace mezi klientem a serverem je realizována pomocí příkazů a XML souborů. Pomocí XML souborů se pak určuje cesta v rámci stromu modelů. V případě chybné autentizace je spojení ukončeno. Samotný server pracuje nad síťovým protokolem TCP a používá komunikační protokol popsaný v části 6.3.

Server také musí obsahovat mechanismus, který je schopen spravovat uživatelské zámky jednotlivých položek stromu. Tento mechanismus se musí vyrovnat i s tím, že klient se odhlásí (ukončí se spojení), aniž by daný zámek zrušil. Zároveň je nutné provádět zápisy klíčů bezpečně (atomicky), aby byl přístup k nim vůči uživatelům výlučný. Podobným způsobem je nutné řešit i seznam klientů, kteří chtějí dostávat informace o běhu simulace. Životní cyklus serveru a spojení je ukázán na obrázku 6.2.

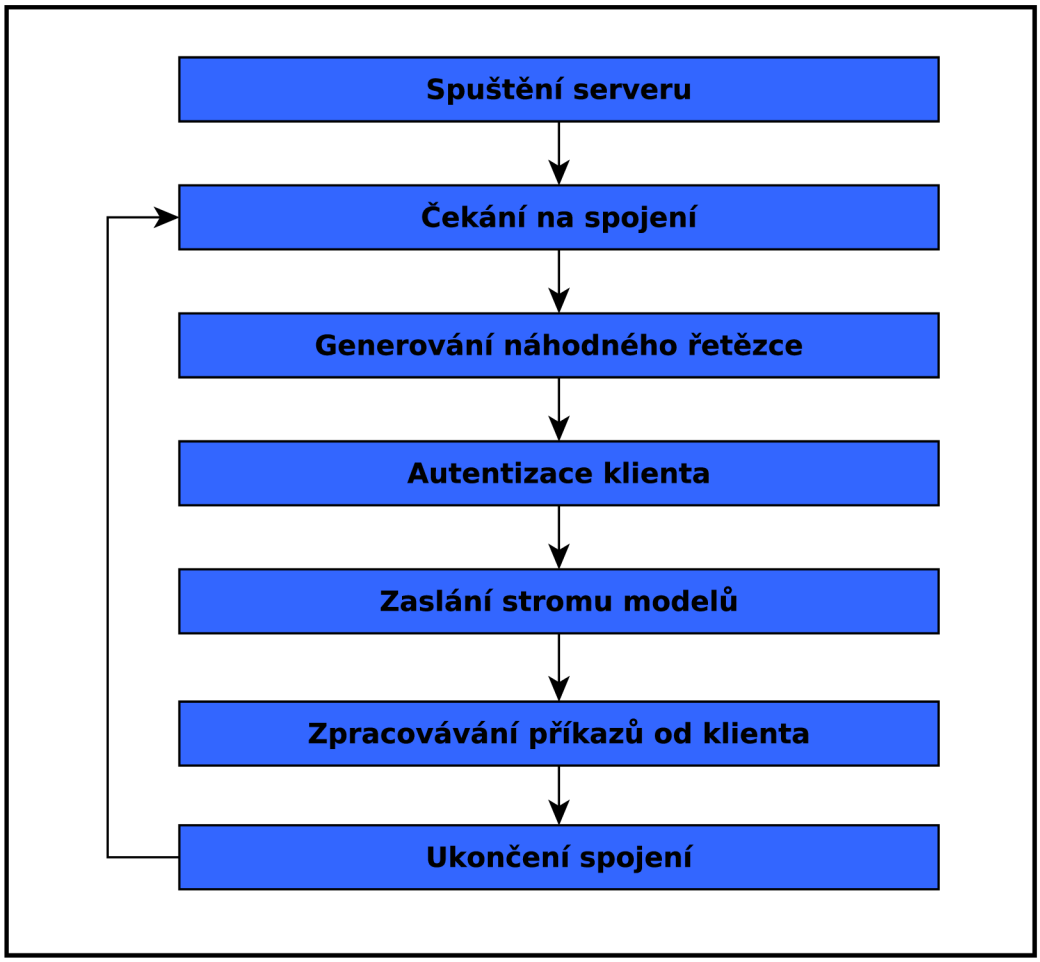
6.5 Návrh klienta

Klientská aplikace umožňuje pracovat s modely na vzdáleném serveru. Samozřejmostí je, že může být otevřeno více spojovaných modelů současně. Aplikace také musí perzistentně ukládat přihlašovací údaje k jednotlivým serverům. Životní cyklus klientské aplikace je zobrazen na obrázku 6.3

6.5.1 Návrh uživatelského rozhraní

Uživatelské rozhraní musí příjemné a efektivně zobrazovat a upravovat modely atomického a spojovaného DEVS a vysokoúrovňových Petriho sítí. Také musí přehledně zobrazovat modely z více serverů a spravovat zámky položek stromu. Schéma návrhu uživatelského rozhraní je vyobrazeno na obrázku 6.4.

(1) Hlavní nabídka slouží k ovládání celé aplikace a je pomocí ní možné vyvolat dialogy na nastavení parametrů pro přístup ke vzdáleným serverům. Z nabídky je možné také provádět připojení a odpojení vůči jednotlivým serverům.



Obrázek 6.2: Životní cyklus serveru.

(2) Lišta nástrojů obsahuje vybrané funkce, které jsou dostupné z hlavní nabídky. Jejím hlavním úkolem je často používané funkce aplikace podsunout uživateli více k ruce.

(3) Přepínání mezi servery je realizováno pomocí záložek a umožňuje uživateli přistupovat na více serverů současně. Při přepnutí na jiný server dojde k upravení obsahu ve **(4) Stromu modelů**. Pomocí této možnosti se také mohou jednotlivé modely přesouvat mezi servery či klonovat. Uživatel je tedy schopen z několika distribuovaných modelů sestavit jeden model.

(4) Strom modelů současně umožňuje otevírání existujících atomicDEVS modelů, které se otevírají v dialogu. Při otevření simulace (dvojklikem, nebo přes kontextové menu), spojovaného modelu, nebo vysokoúrovňové Petriho sítě, se požadovaný prvek otevře v nové záložce **(5)** (pokud již není otevřen). Z části **(4)** se ovládají i další klíčové funkce pro manipulaci a vytváření nových entit v programu.

(5) Záložky simulací, spojovaných modelů a vysokoúrovňových Petriho sítí umožňují přepínání mezi jednotlivými **(6) Pracovními plochami** a je tedy možné souběžně provádět několik různých úkonů. V části **(6)** probíhá samotná editace simulací, vysokoúrovňových Petriho sítí či spojovaných modelů.

(7) Záložka logování a záložky simulací slouží k přepínání mezi logovacími výpisy aplikace, které jsou zobrazovány v **(8) Textovém výpisu**.

(8) Textový výpis zobrazuje v první záložce, která nejde zavřít, obecné informace o běhu aplikace. Například že určitá položka byla zamčena/odemčena, nebo že byla spuštěna simulace. Ostatní záložky jsou otevírány na základě uživatelské registrace příjmu informací ze simulace. Pro každou simulaci jsou zobrazovány následující položky:

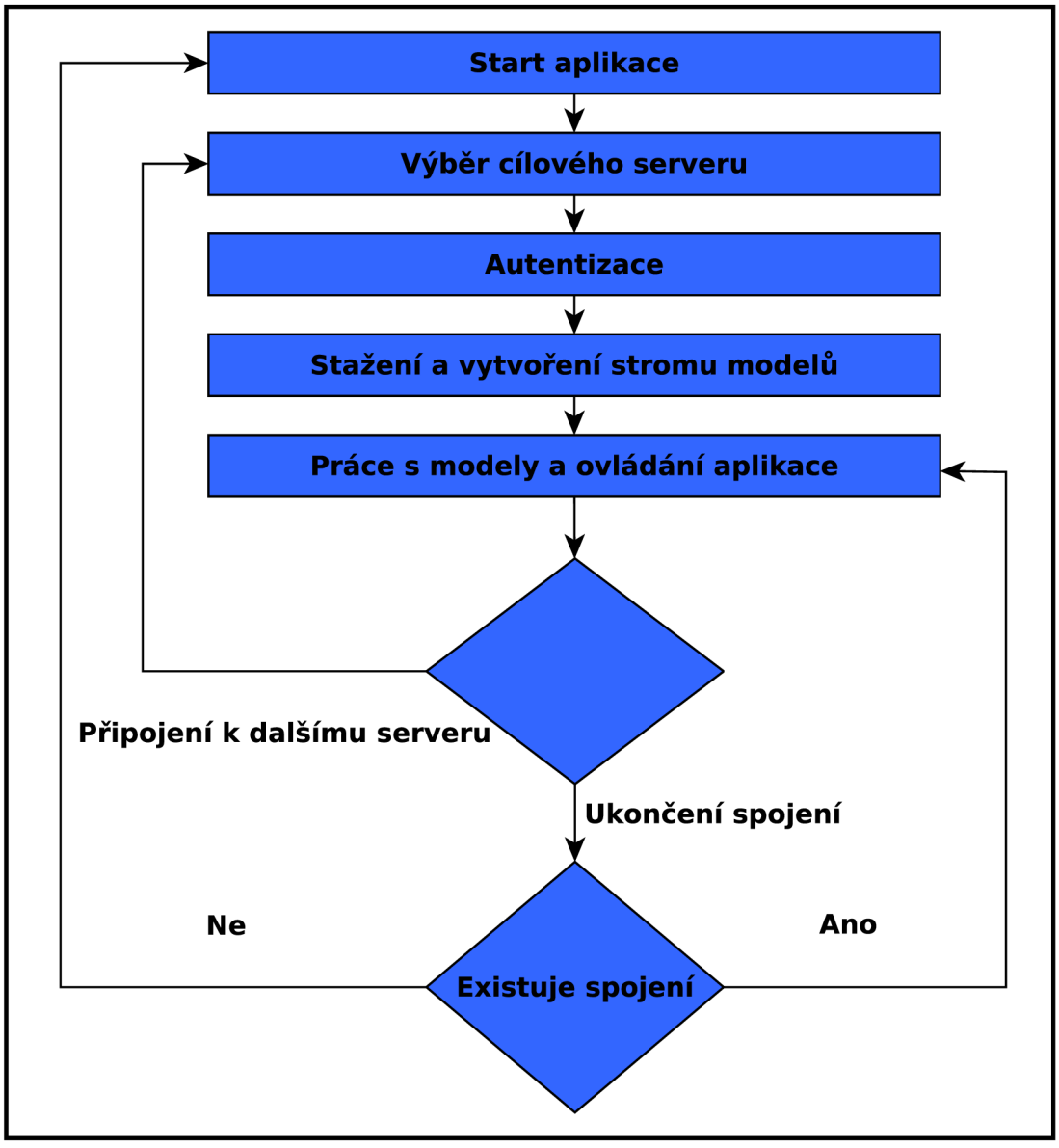
- Aktuální čas simulace.
- Čas minulého přechodu.
- Čas následujícího přechodu.
- Části logovacího výstupu simulace.

Posledním prvkem rozhraní je **(9)Stavová lišta**, která slouží k zobrazení informací o stavu aplikace, jako je například informace o tom, zda je aplikace připojena k serveru nebo ne.

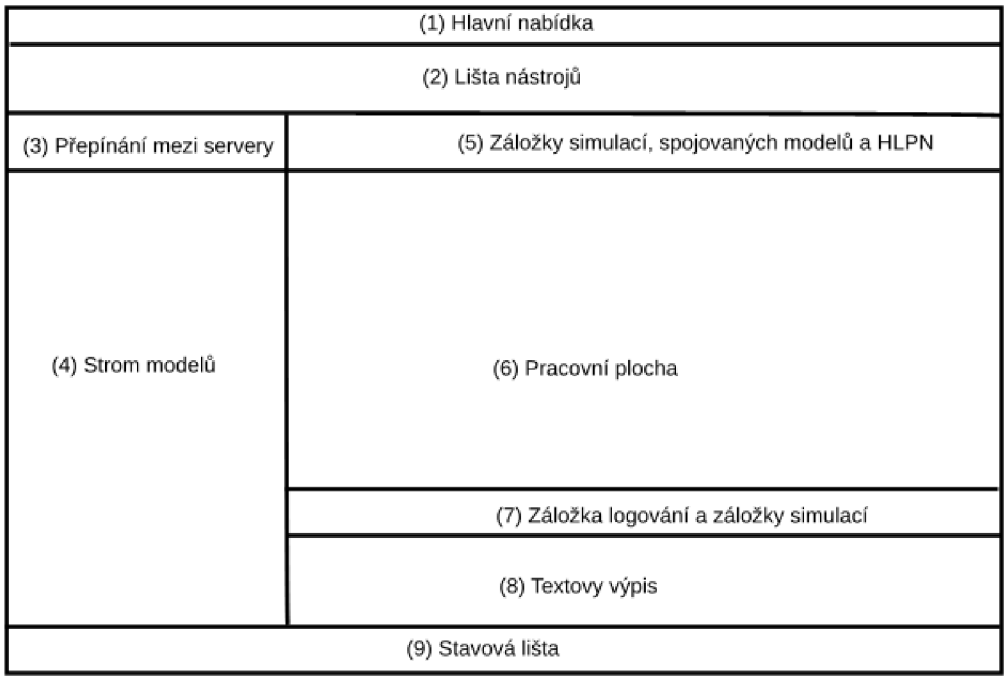
Dalším významnou částí návrhu uživatelského rozhraní je vytvoření dialogu pro editování atomického DEVS modelu. Návrh dialogu je na obrázku 6.5.

Dialog se skládá ze tří částí, a to:

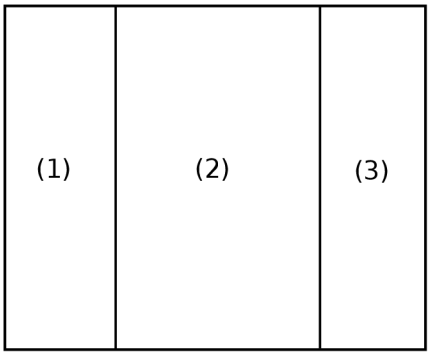
- **(1)** - Grafická reprezentace množiny vstupních portů i s jejich jmény.
- **(2)** - Strom jednotlivých vlastností, které jsou u modelu definovatelné.
- **(3)** - Grafická reprezentace množiny výstupních portů i s jejich jmény.



Obrázek 6.3: Životní cyklus klientské aplikace



Obrázek 6.4: Návrh uživatelského rozhraní klienta



Obrázek 6.5: Návrh dialogu pro editaci atomického DEVS

Kapitola 7

Implementace

7.1 Server

Jak již bylo zmíněno výše, server je implementován ve Smalltalku, přesněji v jeho implementaci jménem Squeak. Nabízí klientským aplikacím služby nástroje SmallDEVS. Server jako takový je pouze v textové formě, nemá žádné grafické rozhraní. Samotná implementace je rozdělena do dvou základním balíčků:

- **SmallDEVS-Server**
  V tomto balíčku jsou definované třídy a jejich instance sloužící pro běh a obsluhu klientských požadavků. Jednotlivé třídy budou zevrubně popsány níže.
- **SmallDEVS-PN**
  V tomto balíčku se nachází definice objektů pro práci s Petriho sítěmi. Implementace je ale pouze na abstraktní úrovni, nejedná se tedy o plnohodnotnou simulací schopnou implementaci.

7.1.1 SmallDEVS-Server

TCPServer

Jak již napovídá název, v této třídě je implementováno samotné jádro serveru. Pro obsluhu příchozích spojení se používá objekt *ConnectionQueue*. V hlavní nekončené smyčce serveru (po zavolání metody *start*), která je vytvořena jako samostatný proces, probíhá periodické volání metody *getConnectionOrNil* objektu *ConnectionQueue*. Pokud je ve frontě neobsloužené příchozí spojení, tak *ConnectionQueue* vrací *Socket* tohoto spojení. V opačném případě vrací hodnotu *nil*. Tento způsob práce není sám o sobě příliš dobrý, protože proces se pořád dotazuje, zda není nové příchozí spojení, a zatěžuje tím nadmíru procesor. Z tohoto důvodu je mezi jednotlivé testování fronty vloženo volání *(Delay forMilliseconds: 500) wait.*, které částečně eliminuje nežádoucí nadměrnou aktivitu na procesoru.

Po detekci příchozího spojení vytvoří server nový objekt *TCPConnection* a nastaví mu potřebné objekty, které budou popsány dále. Poté objekt spustí voláním metody *start*. Od této chvíle objekt autonomně zpracovává příkazy od klienta.

Vytvoření a spuštění serveru v prostředí Squeak se provádí následovně:

```
mServer := TCPServer new.
mServer start.
```

Poté se chod serveru zastavuje voláním *mServer stop.*, které provede jak zrušení hlavní smyčky, tak i zavření všech otevřených spojení.

TCPConnection

Jak již bylo řečeno, zde probíhá obsluha celého jednoho spojení mezi klientem a serverem. Při inicializaci objektu proběhne vygenerování náhodného hexadecimálního řetězce, který posléze slouží pro autentizaci klienta. Komunikace s klientem probíhá v samostatném procesu. Při spuštění se provede vytvoření proudu dat ze socketu a zaslání náhodného řetězce klientovi. Po obdržení odpovědi provede server prohledání uživatelských účtů ve snaze najít shodu. Pro ověření identity se používá kryptografický hash sha1, který je implementován třídou *SecureHashAlgorithm*. Data jsou testována následovně:

SHA1(jméno;heslo;náhodný_řetězec)

V případě, že bude nalezena shoda, nastaví se příznak *isAuth* a server čeká na požadavky od klienta. V opačném případě se zašle klientovi zpráva, že autentizace selhala a ukončí se spojení. Po přijetí dat dojde k jejich rekonstrukci do původního XML souboru pomocí detekce crlfcrlf (\n\r\n\r)[1]. Poté jsou přijatá data rozdělena podle výskytu první mezery na příkaz a na data. Tyto informace jsou předány do metody *doCommand:data:*, kde probíhá provedení příslušných operací dle příkazu. Jednotlivé operace jsou vykonávány ve třídě *MyRepositoryHolder*.

MyRepositoryHolder

Účelem této třídy je zajistit exkluzivní přístup při editaci modelů. Výlučný přístup je zajištěn pomocí semaforu *myRepositorySem*, který ohraničuje kritickou sekci, v níž se editují položky stromu. Třída MyRepositoryHolder obsahuje také proces, který periodicky zasílá změny v běžících simulacích klientům, kteří jsou k odběru zaregistrováni. Tato činnost se děje jednou za sekundu. Získávání logovacích informací z běhu simulace je prováděno pomocí nastavení objektu *MyReadWriteStream* do položky *reportStream* u požadované simulace. Informace o čase jsou získávány přímo z objektu simulace.

QueryXMLParser

Parsování jednotlivých klientských příkazů je prováděno zde. Třída obsahuje metody pro vyhledávání položek ve stromě podle XML, metody pro změnu nastavení, vytvoření nové položky...

MyReadWriteStream

Tato třída je potomkem *ReadWriteStream* a jsou v ní upraveny metody:

- **on:** - zde je navíc oproti předkovi vytvořen semafor pro výlučný přístup.
- **nextPutAll:** - toto volání je používáno v simulaci pro tvorbu logu. Volání předka je v kritické sekci.
- **popData** - vrátí všechna data v bufferu a vyprázdní ho. Zkopírování bufferu a jeho vyprázdnění se děje v kritické sekci kvůli vyloučení souběžného čtení a zápisu.

[1] Tato sekvence je způsob ukončování řádků v systému Windows, který je používán také v ostatních textových protokolech, jako je například HTTP.

7.1.2 SmallDEVS-PN

Tento balíček obsahuje implementaci pro reprezentaci Petriho sítí.

PetriNetPrototype

Jedná se pouze o neaktivní část simulace, která je postavena na *AtomicDEVSPrototype*, ale negeneruje žádné vstupy ani výstupy. Stanovení vstupních a výstupních portů modelu je zde řešno automaticky a to tak, že každé místo, které je v *PRECONDS* a není v *POSTCONDS* ani v *CONDS* je automaticky vstupní port. Analogicky k tomu se určují i výstupní porty, jenom s tím rozdílem, že místo se nachází v *POSTCONDS* a ne v *PRECONDS* ani *CONDS*.

PetriNetAbstractItem

Jedná se o abstraktní třídu, která slouží jako základ pro definice tříd *PetriNetPlace* a *PetriNetTransition*. Jsou zde definovány společné vlastnosti těchto dvou tříd, jako jsou jméno, jedinečné id v rámci sítě a pozice. Také obsahuje dvě abstraktní metody *isTransition* a *isPlace*.

PetriNetPlace

Tato třída slouží pro reprezentaci místa v Petriho sítích. Implementuje abstraktní metody svého předka *isTransition* a *isPlace*, a navíc obsahuje multimnožinu značek.

PetriNetTransition

Tato třída je reprezentací přechodu v Petriho sítích. Implementuje metody *isTransition* a *isPlace*. Dále obsahuje podmínky, metody pro práci s nimi, a další vlastnosti přechodu.

7.2 Klient

V této části bude rozebráno, jak je klientská aplikace strukturována, jak je implemtováno editování modelů, a ostatní uživatelské možnosti. Program je implementován v programovacím jazyce C++ s maximálním použitím objektového návrhu. Pro grafické rozhraní je použita knihovna Qt, která nahrazuje vlastní implementací většinu standardních knihoven jazyka C++.

7.2.1 Síťová komunikace

Síťová komunikace je implementována ve třídě *Connection* pomocí interní třídy Qt *QTcpSocket*. Třída *Connection* je implementována podle návrhového vzoru *Singleton* a to z důvodu, aby se k ní dalo přistupovat z různých částí aplikace. Pokud by se nejednalo o *Singleton*, vznikaly by problémy s předáváním jednotlivých spojení se servery.

Pro sestavení autentizační zprávy je použito volání *QCryptographicHash::hash(text)*, které provádí kryptografický hash pomocí *sha1*.

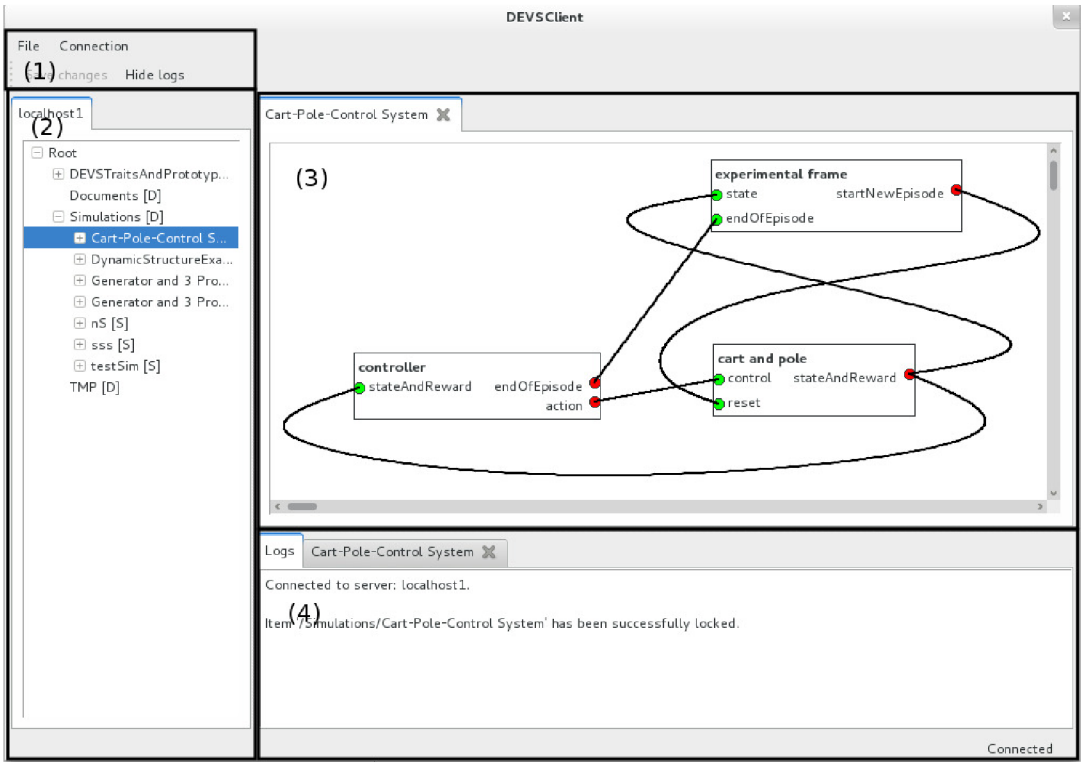
Zpracovávání přijatých dat ze serveru probíhá následovně:

1. Čekání na příjem dat.
2. Přidání přijatých dat do místního statického bufferu.

3. Zjištění, zda je zpráva celá (obsahuje *crlfcrlf* a značku $< /myRepository >$).
4. Pokud jsou data kompletní, dojde k přechodu na další bod. Pokud nejsou kompletní, nastane přechod do bodu **(1)**.
5. Rozdělení dat na příkaz a případný XML soubor a provedení příkazu.

7.2.2 Popis grafického rozhraní

Výsledné grafické rozhraní je na obrázku 7.1.



Obrázek 7.1: Výsledné grafické rozhraní.

Celé uživatelské rozhraní lze rozdělit do čtyř částí:

- **(1)** Hlavní menu a nástrojová lišta, které slouží k základnímu ovládání aplikace.
- **(2)** Záložky se stromy, které zobrazují modely uložené na jednotlivých serverech, ke kterým je klient připojen.
- **(3)** Pracovní plocha, kde uživatel pracuje se spojovanými modely a Petriho sítěmi.
- **(4)** Výsuvný panel, sloužící k zobrazování logovacích informací celé aplikace a dat z příslušných simulací.

Jednotlivé elementy aplikace budou detailněji popsány níže.

7.2.3 Hlavní menu a nástrojová lišta

V hlavním menu se nachází nejdůležitější nastavení a to nastavení informací o serverech. Správce všech spojení je vyobrazen na 7.2(a), a na 7.2(b) je správce jednotlivých spojení.

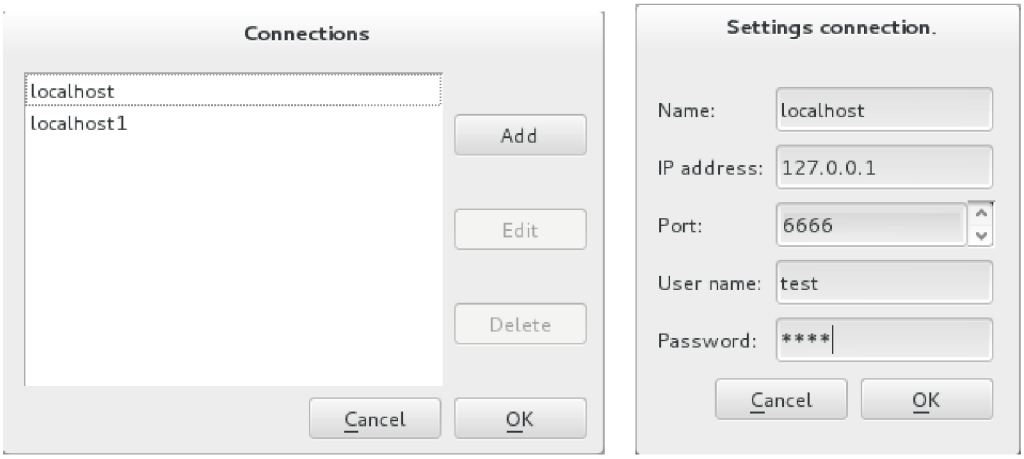
Nastavení spojení je ukládáno do profilu uživatele. V Linuxu se například jedná o adresář *$HOME/.config/DEVSClient*, kde poslední položka cesty je jméno aplikace/organizace. Samotná konfigurace se nachází v souboru *DEVSClient.conf*. S tímto souborem se pracuje pomocí Qt knihovny *QSettings*. Práce s ní je velmi intuitivní.

Jednotlivá spojení jsou identifikována pomocí jména, které musí být jedinečné. Rozsah pro zadávání portů je stanoven v intervalu $< 1, 65535 >$. Jakákoli jiná hodnota nebude dialogem akceptována. Heslo se v dialogu z bezpečnostních důvodů nezobrazuje. Pokud bylo heslo již někdy zadáno a uživatel upravuje spojení a položku heslo nevyplní, zůstane heslo nezměněno.

V programu jsou přihlašovací informace zabaleny do třídy *ConnectionInformation*, která pomocí serializace a deserializace zapisuje jednotlivá spojení skrze *QSettings* přímo do konfiguračního systému.

Další položkou menu je podmenu s výpisem všech zadaných spojení. V případě že je spojení aktivní, je u jména spojení zatržíko. Pokud se na aktivní spojení klikne, dojde k odpojení.

Poslední položkou v hlavním menu je uložení nastavení, které bude okomentováno níže.

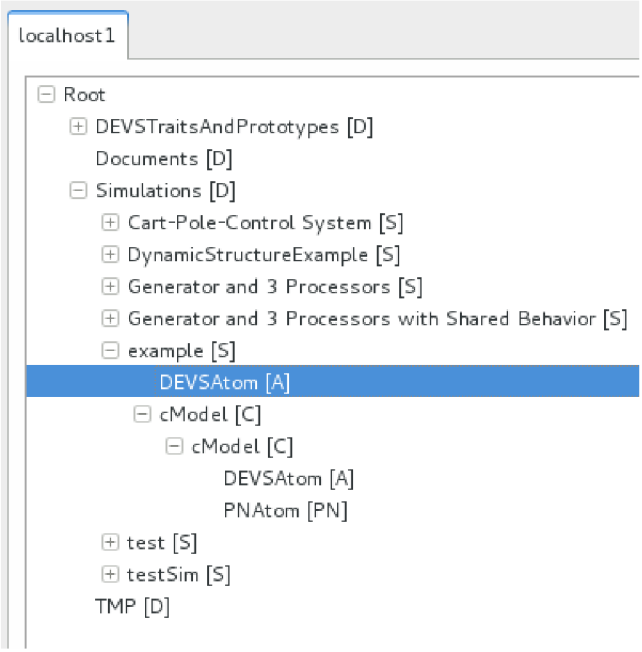


(a) Správce všech spojení. (b) Správce jednoho spojení.

Obrázek 7.2: Nastavení spojení.

7.2.4 Strom modelů

Po připojení k serveru je vytvořena nová záložka s prázdným stromem. V případě úspěšného přihlášení zažádá klient o strom modelů, který je poté zobrazen v příslušné záložce místo prázdného stromu. Struktura modelů je realizována pomocí *QTreeView*. Pomocí této komponenty je ovládána celá struktura modelů a simulací. Zde je možné modely přesouvat, klonovat, vytvářet nové... Při kliknutí pravým tlačítkem na jednotlivé položky ve stromu je možné vyvolat kontextové menu obsahující editační akce. Na obrázku 7.3 je vyobrazen strom modelů.

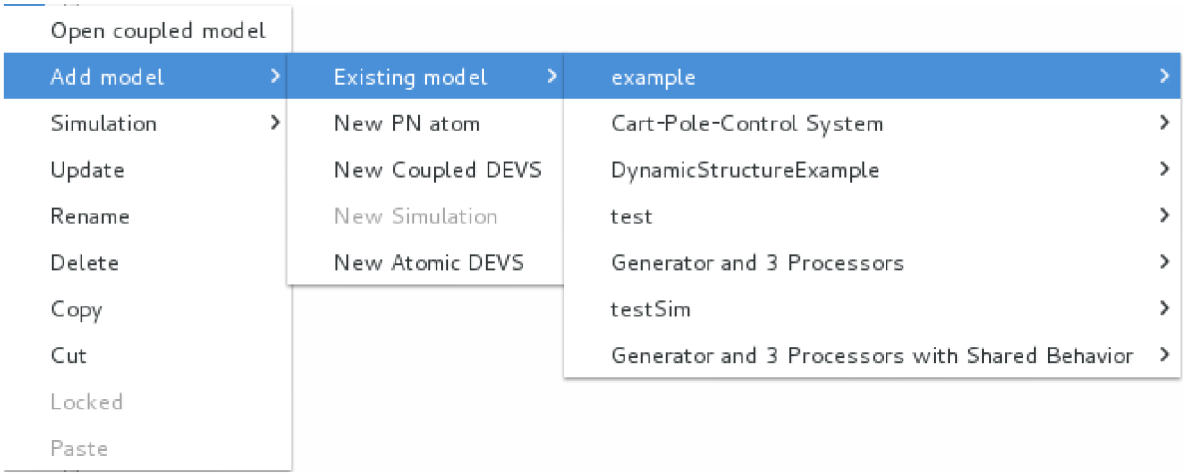


Obrázek 7.3: Strom modelů.

Nejvýše ve stromu je kořen stromu (Root). Ten nelze smazat, ani přejmenovat. Pod touto položkou se nachází další složky stromu. Ty obsahují jednotlivé simulace. V simulacích může být vložen spojovaný model, atomický DEVS nebo atom specifikovaný pomocí Petriho sítě. Pro zvýšení přehlednosti jsou za jmény položek uvedeny v hranatých závorkách písmenka, která značí, o jaký prvek se jedná. Použitá písmena a jejich význam je uveden níže:

- **D** - značí, že položka je složkou.
- **S** - označuje simulaci. Pokud je simulace spuštěna, změní se **S** na **R**.
- **A** - označuje model specifikovaný pomocí atomického DEVS.
- **P** - určuje model specifikovaný pomocí Petriho sítě.
- **C** - označuje spojovaný model (Coupled DEVS).
- **T** - značí rys (trait), který je používán u specifikace atomického DEVS.
- **F** - označuje obyčejný soubor.

Při kliknutí pravým tlačítkem na položku menu je vyvoláno kontextové menu. Obsah menu záleží na typu položky. Každé menu ale obsahuje položku **Lock**, která slouží k požádání serveru o zamčení položky. Pokud je položka uzamčena, umožňuje to uživateli editovat podstrom této položky. Přidat model do zamčeného podstromu lze také z kontextového menu. Je možné buď vytvořit model nový, nebo pomocí nabídky **Add Model - Existing Model** vybrat podstrom existující simulace a pomocí zanořených *QMenu* vybrat potřebný model. Po potvrzení dojde ke sklonování vybraného modelu. Další možností pro přidání modelu je zkopírovat pomocí kontextového menu daný model. Na obrázku 7.4 je vyobrazeno kontextové menu pro simulaci, ve kterém je rozbaleno podmenu pro přidání existujícího modelu.



Obrázek 7.4: Kontextové menu simulace.

7.2.5 Uložení modelů

Modely jsou uloženy ve třídách a vztahy mezi nimi jsou vidět na diagramu 7.5. Níže následuje podrobnější popis těchto tříd:

- **SimulationBaseObject**
  Základní třída, která se používá pro sestavení stromu, aby bylo možné se všemi položkami pracovat stejně. K identifikaci, o jakou položku se přesně jedná, slouží vlastnost třídy *type*. Tyto typy jsou až na *PrintableBoxModel* stejné, jak bylo popsáno u zkratek za jmény položek. Tato třída obsahuje i odkaz na předka ve stromu a seznam (*QList*) potomků prvního řádu (ten se nachází přímo pod danou položkou). Pokud je položka typu *Root*, odkaz na předka je samozřejmě *NULL*. Pro rozlišení, ke kterému serveru (spojení) daný prvek patří, obsahuje objekt typu *QTcpSocket*, pomocí kterého se komunikuje se serverem.

- **File**
  Tato třída reprezentuje obyčejný textový soubor.

- **Folder**
  Reprezentuje složky stromu. Složky mohou obsahovat další složky, nebo jednotlivé modely.

- **PrototypeObject**
  Třída sloužící k uložení rysu (trait) pro definici atomického DEVS. Podrobněji bude probrána v 7.2.6.

- **PrintableBoxModel**
  Tuto třídu dědí všechny prvky, které mohou byt začleněny do spojovaného modelu. V této třídě jsou definovány množiny vstupních a výstupních portů modelů. A právě díky tomuto lze se všemi modely pracovat graficky stejně.

- **AtomicDEVS**
  Obsahuje parametry popisující atomicDEVS.

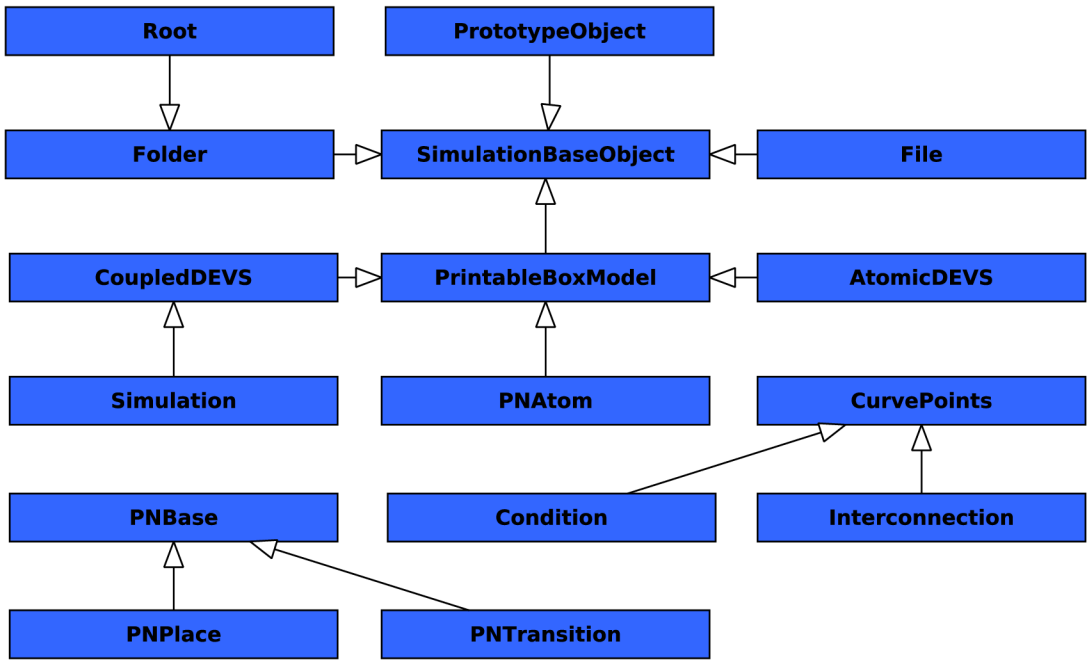
- **CurvePoints**
  Třída sloužící k uložení bodů křivky. její součástí jsou i metody pro parsování bodů ze zápisu *x@y* do *QPoint* a naopak.
- **CoupledDEVS**
  Uchovává definici spojovaného modelu. Jednotlivá propojení mezi modely jsou realizována seznamem objektů typu *Interconnection*.
- **Interconnection**
  Popisuje propojení mezi dvěma modely.
- **Simulation**
  Tato třída do definice *CoupledDEVS* přidává informace o simulaci (časy, log...).
- **PNAtom**
  Třída popisuje vysokoúrovňovou Petriho síť (místa, přechody) a udržuje informace o vstupních a výstupních portech.
- **PNBase**
  Třída obsahující společné parametry pro místa a přechody (id, jméno, pozice).
- **PNPlace**
  Popisuje místa sítě a implementuje abstraktní metody *isPlace, isTransition*.
- **Condition**
  Popisuje jednotlivé podmínky, které jsou dále používány v přechodech.
- **PNTransition**
  Třída popisuje přechod v Petriho síti. Obsahuje tři množiny podmínek (preConds, postConds, conds).

7.2.6 Editace atomic DEVS modelu

Pro editaci a vytváření nového atomic modelu, popsaného pomocí DEVS, slouží dialogové okno vyobrazené na 7.6. V levém a pravém sloupci dialogu jsou zobrazeny vstupní (vlevo) a výstupní (vpravo) porty. Pro přidání dalšího portu stačí kliknout pravým tlačítkem do příslušného sloupce, a to mimo již existující porty, a z menu vybrat *Add port*. Při kliknutí pravým tlačítkem na existující port se zobrazí možnosti jeho editace. Samotné ovládání vlastností modelu je zobrazeno pomocí *QTreeView* a je provedeno v souladu se specifikací atomického DEVS. Při vyvolání kontextového menu je možné upravovat vlastnosti modelu. Na 7.6 je ukázáno menu pro přidávání rysu modelu. Ve stromě existuje fixně definovaná položka *DEVSTraitsAndPrototypes*, ve které jsou uloženy rysy. A právě od této položky je budován strom rysů, který obsahuje také předdefinovaná a povolená interní jména metod pro řízení simulace.

Těla metod jsou v současné implementaci pouze textová a nejsou nijak validována. Klient neimplementuje žádnou formu překladače/validátoru Smalltalk. Chyba implementace metody se tedy projeví až za běhu simulace. Zkušenému uživateli jazyka Smalltalk by to však nemuselo dělat velké problémy.

Dialog pro editaci obsahuje také možnost přijímání chybových zpráv editovaného modelu od serveru. Při potvrzení dialogu jsou serveru zaslány změny a dialog čeká na signál o výsledku změn. Pokud není stavový příkaz prázdný, dialog není uzavřen. V případě výskytu chyby a zrušení dialogu dojde k návratu do posledního konzistentního stavu.



Obrázek 7.5: Diagram dědičností mezi objekty.

Editor rysů

Na obrázku 7.7 je vyobrazen editor rysů. Rysy slouží pro předdefinování vlastností modelů. Samotný rys může obsahovat i rys již existující. Dále rys obsahuje popis slotů a metod prováděných modelem.

7.2.7 Grafický editor

Grafický editor slouží pro editování spojovaných modelů a Petriho sítí. Jednotlivé editory jsou umisťovány do záložek, mezi kterými může uživatel přepínat. Záložky samotné jsou implementovány oddděním *QWidget* třídy *QWidgetTab*.

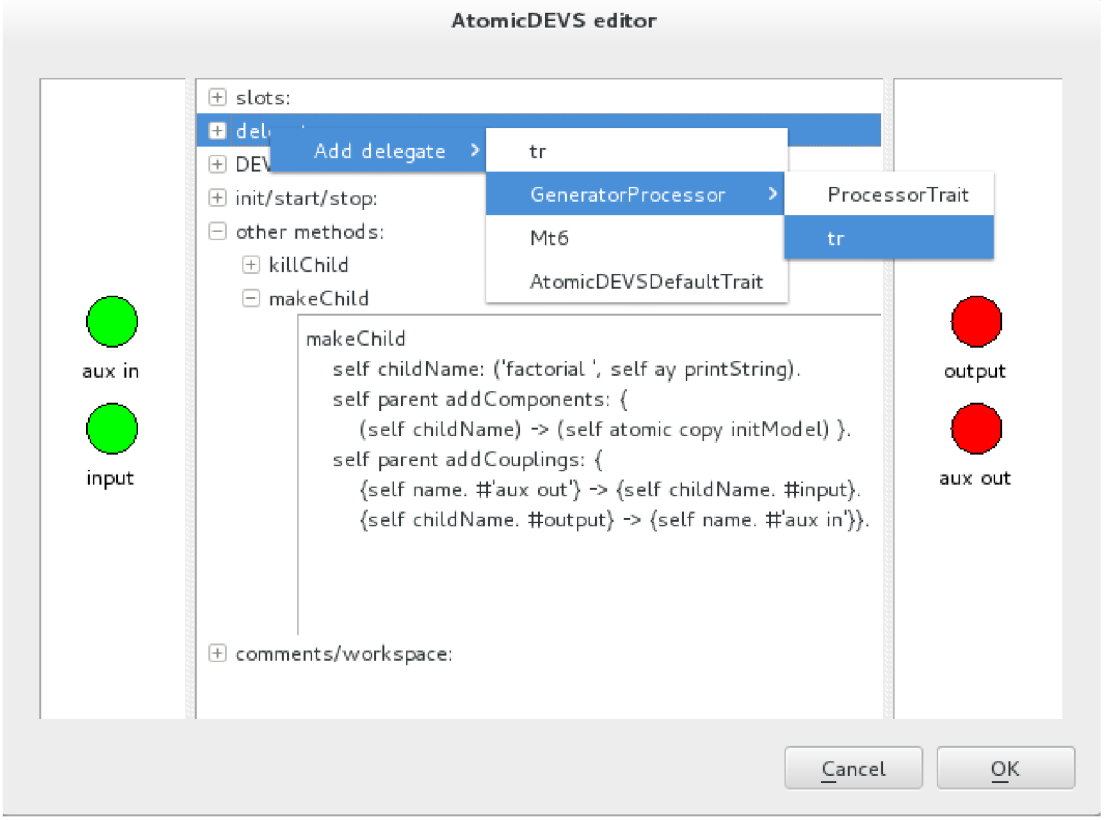
Záložka obsahuje editovaný model, který je dále předán do grafické scény (viz níže). V případě změny modelu o tom aplikace vizuálně informuje uživatele, jak je ukázáno na obrázku 7.9 v **(6) Záložka modelu**. V záložce je zobrazeno z důvodu úspory místa pouze jméno modelu a stav. Při najetí myší na záložku se zobrazí bublinová nápověda ve tvaru:

Jméno spojení:Pozice ve stromu

Pozice ve stromu je ve stejném tvaru jako v Linuxovém systému. Položka *Root* je nahrazena pomocí /.

V záložce je zobrazen i stav modelu, jestli byl nebo nebyl modifikován. V případě modifikace je za jeho jménem zobrazena *.

Záložka obsahuje křížek pro zavření. Pokud byl model změněn, je uživatel dotázán, zda si přeje změny uložit, nebo zrušit zavření. Pro zobrazování spojovaných modelů a Petriho sítí jsou použity dva různé editory. Na diagramu 7.8 je znázorněna dědičnost editorů. Pro práci s grafickými prvky, odvozenými od třídy *QGraphicsItem*, jsou zapotřebí dvě třídy jiné. *QGraphicsScene* slouží pro ukládání prvků. *QGraphicsView* vizualizuje prvky



Obrázek 7.6: Dialog pro upravování atomic DEVS modelu.

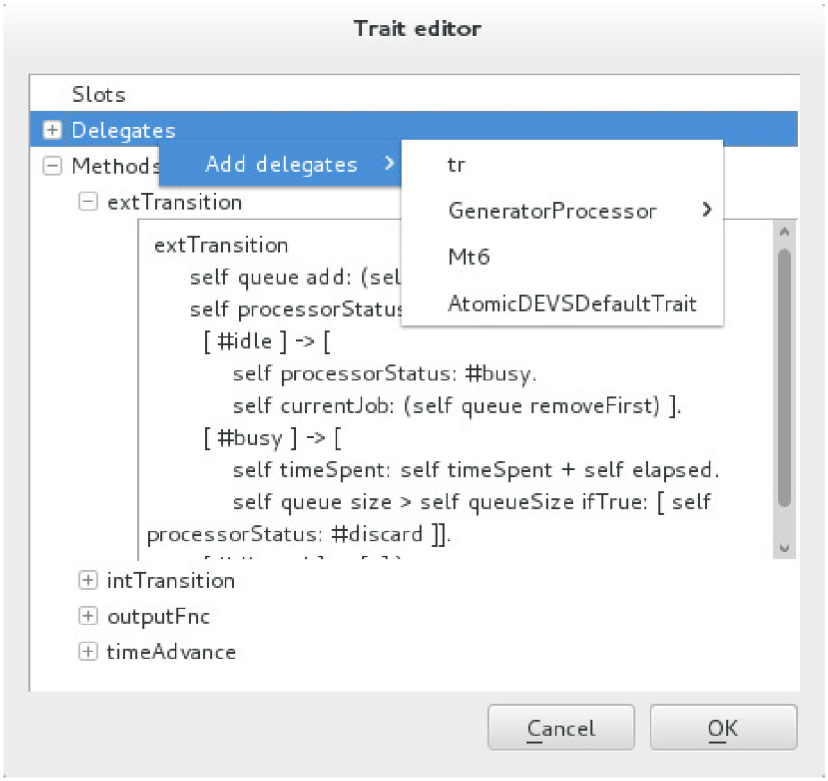
uložené v *QGraphicsScene*. Pro vytvoření editorů byla oděděna právě třída *QGraphicsView*. *QGraphicsViewEditorBase* umožňuje pracovat s editory pro spojovaný model a Petriho sítě stejným způsobem a definuje základní metody pro nastavování modelů a detekci změn. Jednotlivé modely jsou na úrovni *QGraphicsViewEditorBase* reprezentovány třídou *PrintableModelBox*. Postup pro vytvoření souboru změn bude popsán v 7.2.13.

7.2.8 Editace spojovaného DEVS modelu

Editor spojovaného modelu je vyobrazen na 7.9. Na obrázku je vyznačeno pět níže popsaných značek:

- **(1) Vstupní port** a **(2) výstupní port**
  Vstupní (zelené), respektive výstupní(červené), porty slouží pro začlenění tohoto spojovaného modelu do struktury jiného modelu. Pro vytvoření nového propojení stačí kliknout pravým tlačítkem na port a vybrat *Connection*. Připojení k určenému portu se opět provádí pravým tlačítkem. Propojovat jde jen ve směru od výstupního portu ke vstupnímu. Výjimku ovšem tvoří propojení vstupního portu aktuálního modelu na vstup jiného. Analogicky k tomu i výstup jiného modelu na výstup aktuálního modelu.

- **(3) Model**
  Všechny modely umístěné do scény jsou odděděny od třídy *PrintableBoxMox*. Při kliknutí pravým tlačítkem na model dojde k odeslání signálu, který přijde do stromu modelů, a na pozici kurzoru zobrazí dané kontextové menu.



Obrázek 7.7: Dialog pro upravování rysů.

- **(4) Vybrané propojení**
  Takto vypadá aktivní (vybraná) spojnice mezi modely. Blíže bude princip použitých křivek popsán níže v 7.2.9.

- **(5) Nevybrané propojení**
  Takto je vykreslena křivka, pokud není vybrána myší.

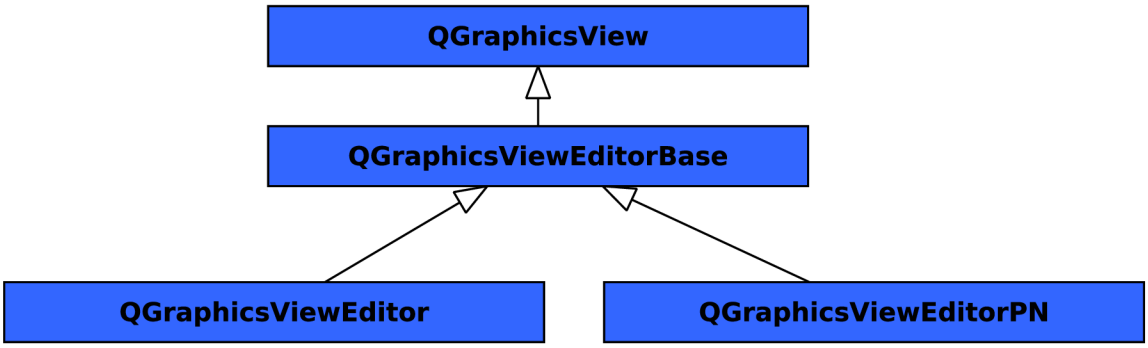
7.2.9 Implementace křivek

Knihovna Qt nabízí pouze aproximační Bézierovy křivky (na obrázku 7.10), ale SmallDEVS používá interpolační křivky. Z důvodu zachování kompatibility bylo tedy nutné použít tyto křivky.

Pro Qt sice existuje knihovna Qwt[8], která obsahuje interpolační křivky, ta je ale primárně určena pro kreslení grafů. Problém při implementaci nastal v okamžiku, kdy bylo třeba zjistit, zda daný bod (bod křivky, ne řídící) leží na křivce. Problém by bylo možné vyřešit procházením vykreslené rastrované křivky, tento způsob se ale při implementaci ukázal být jako nevhodný a neefektivní.

Bylo tedy nutné vytvořit vlastní knihovnu pro práci s přírodními kubickými interpolačními křivkami. Křivka je determinována následovně [6]:

- $n+1$ opěrných bodů $P_0, P_1 \ldots, P_n$ a $n+1$ reálných parametrů $u_0 < u_1 < u_n$,
- křivka $P(u)$ je složena z kubických polynomů $P_i(u)$, které musí splňovat tyto pod-



Obrázek 7.8: Diagram dědičností grafických scén editoru.

mínky:

$$P_i(u_i) = P_{i-1}(u_i), \forall i \ \in \{1, \ldots, n-1\}$$
$$P'_i(u_i) = P'_{i-1}(u_i), \forall i \ \in \{1, \ldots, n-1\}$$
$$P''_i(u_i) = P''_{i-1}(u_i), \forall i \ \in \{1, \ldots, n-1\}$$

- Volbou čísel $u_0, \ldots, u_n$ se určuje parametrizace křivky. Obvykle se volí jako $u_i = i$ (Uniformní parametrizace).
- Pro výpočet kubické spline křivky lze použít kubické polynomy ve tvaru:

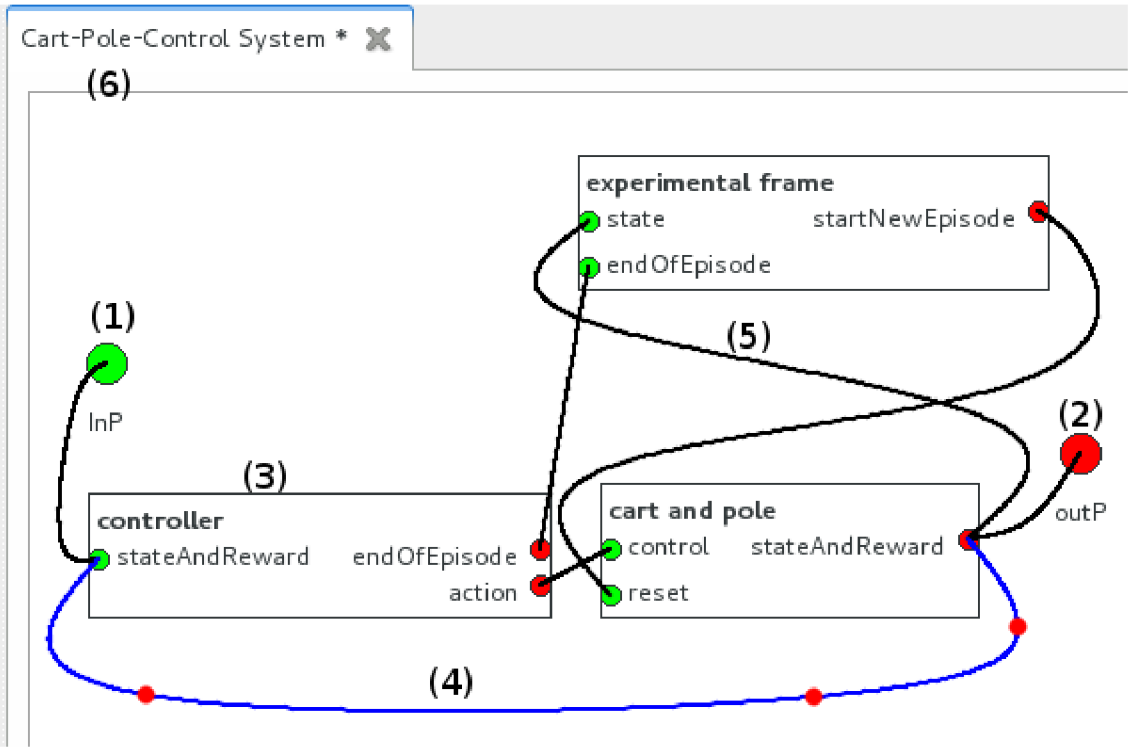
$$P_i(u) = a_i + b_i(u - u_i) + c_i(u - u_i)^2 + d_i(u - u_i)^3, \forall i \ \in \{1, \ldots, n-1\}$$

- Tyto polynomy dosadíme do podmínek pro křivku a vyřešíme soustavu $4 \times n$ rovnic.
- Před vyřešením matice musíme předpočítat derivace pro spline ze vztahu:

$$\begin{bmatrix} 1 & 2 & & & & \\ 1 & 4 & 1 & & & \\ & 1 & 4 & 1 & & \\ & & \cdot & \cdot & & \\ & & & 1 & 4 & 1 \\ & & & & 1 & 2 \end{bmatrix} \begin{bmatrix} D[0] \\ D[1] \\ D[2] \\ \cdot \\ D[n-1] \\ D[n] \end{bmatrix} = \begin{bmatrix} 3(x[1] - x[0]) \\ 3(x[2] - x[0]) \\ 3(x[3] - x[1]) \\ \cdot \\ 3(x[n] - x[n-2]) \\ 3(x[n] - x[n-1]) \end{bmatrix}$$

Samotná implementace křivek je rozdělena do dvou tříd:

- **CurveControl**
  V této třídě je řízena práce s křivkami. Jednotlivé body jsou uloženy do *QPolygon* a v případě vložení nového bodu se provede jeho zařazení na správné místo vzhledem k průběhu křivky. V opačném případě by docházelo k nechtěnému propojení řídících bodů. Při změně množiny řídících bodů dojde k přepočítání tvaru křivky. Před vykreslením se musí pokaždé předpočítat derivace pro jednotlivé body.
- **Cubic**
  Tato třída provádí řešení polynomu na základě předpočítaných derivací.



Obrázek 7.9: Pracovní plocha pro editaci spojovaného modelu.

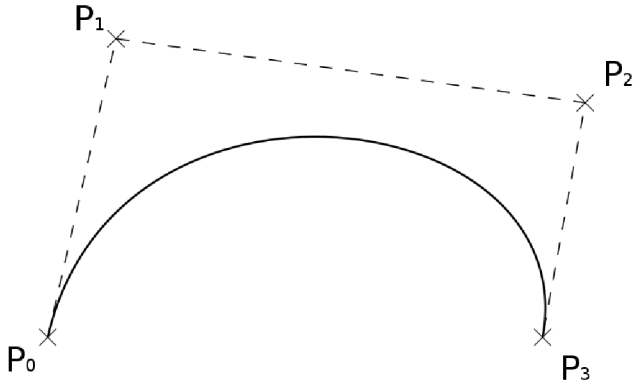
Samotné spojnice a orientované hrany u Petriho sítí poté tyto dvě třídy využívají pro kreslení. Na diagramu 7.11 je znázorněna dědičnost grafických prvků zobrazujících křivky.

Jako základní třída je použita *QGraphicsLineItem*, protože křivka může obsahovat dva body, a bylo tedy výhodné použít ji jako předka. Třída *QGraphicsCurveItemBase* slouží k práci s koncovými body a k práci s křivkou samotnou (přidat, respektive odebrat řídící body). V této třídě jsou odchytávány signály o stisknutí, uvolnění a změně polohy kurzoru, a na jejich základě dochází k ovlivňování vykreslené křivky. Také se zde řeší zamezení editace křivky v případě, že není nastaven zámek modelu. Třída *QGraphicsCurveItem* je používána pro vizualizaci propojení jednotlivých portů modelů. Poslední třídou je *QGraphicsCurveItemWithArrow* používaná pro propojování položek v Petriho sítích. Více o této třídě bude v 7.2.10.

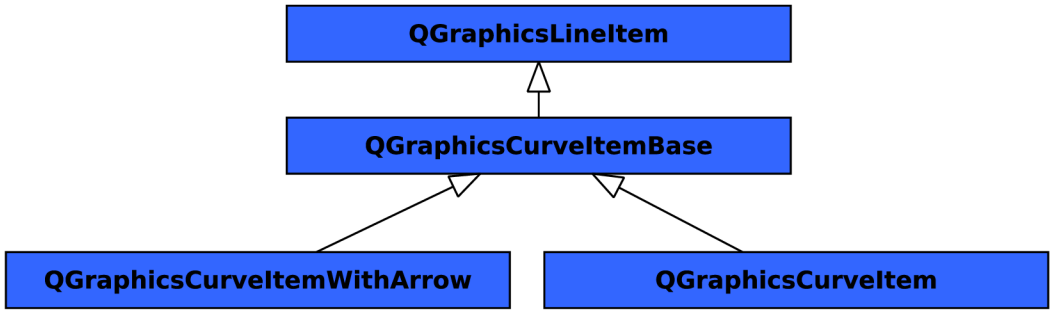
7.2.10 Editace modelu určeného Petriho sítěmi

Na obrázku 7.12 je zobrazen editor Petriho sítě. V obrázku jsou umístěny značky, které jsou rozebrány níže:

- **(1) Místo**
  Místo Petriho sítě je implementováno třídou *QGraphicsItemPNPlace*, která je potomkem *QGraphicsItem*.
- **(2) Přechod**
  Přechod je implementován třídou *QGraphicsItemTransition*, která je potomkem *QGraphicsItemGroup*. Výhodou této třídy je, že je schopná sdružovat jiné grafické prvky scény. Například texty jsou třídy *QGraphicsItemText*.



Obrázek 7.10: Beziérova křivka. [11]



Obrázek 7.11: Diagram dědičnosti grafických objektů křivek.

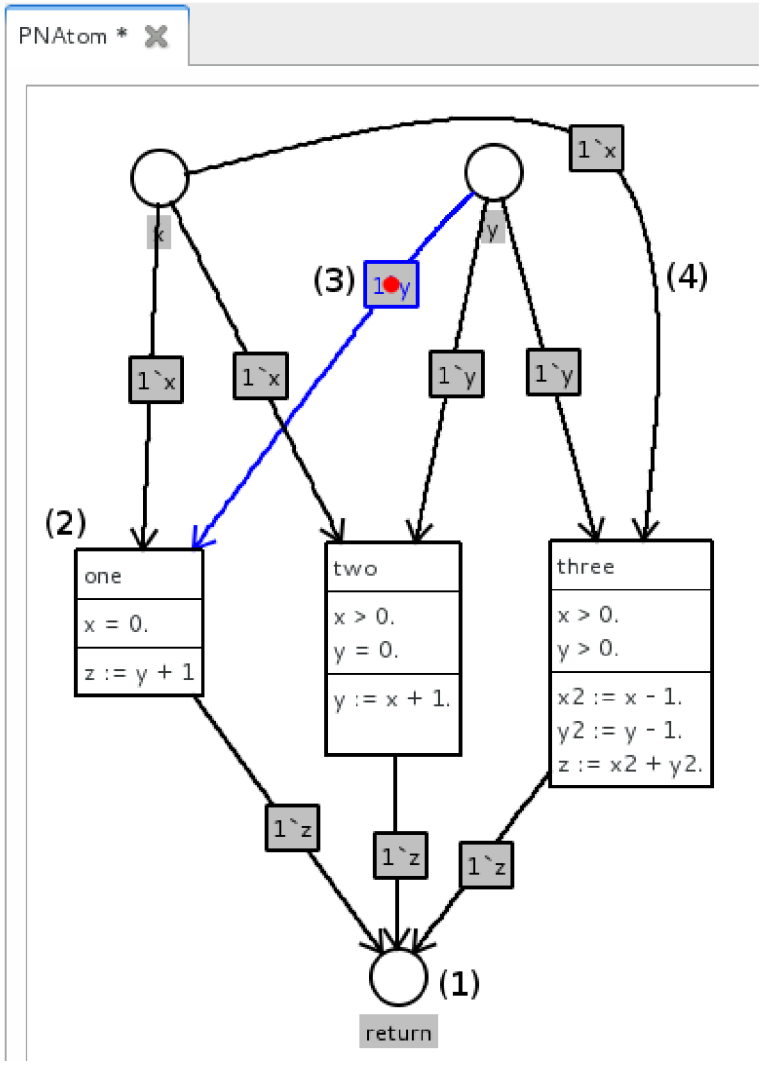
- **(3) Aktivní podmínka**
  Obdélník, ve kterém je vypsána podmínka hrany, je umístěn na prostřední bod křivky (pokud jsou jen dva body, tak se pozicuje na střed spojnice).

- **(4) Podmínka**
  Oproti propojení popsaného výše, je zde navíc ještě přidána šipka, určující zda se jedná o *PRECOND*, *POSTCOND* nebo *COND*. V případě *COND* se jedná o šipku oboustrannou. Pro správné zobrazení úhlu šipek vůči křivce je nutný zásah uživatele, aby vhodně umístil bod před šipku.

7.2.11 Práce se simulacemi

Simulace je rozšíření třídy *CoupledDEVS*. Na obrázku 7.13 je vyobrazena kontextová nabídka pro řízení simulace. Stejně jako ostatní editující operace nad prvky stromu i obsluhování (spuštění, zastavení, restartování) simulace vyžaduje zámek. Pro přihlášení se k odběru stavu simulace slouží volba *Show log*. Po kliknutí na tuto položku je zaslán požadavek serveru o zařazení do odběratelů stavu a otevře se nová záložka v logovacím okně, zobrazená na obrázku 7.14. Pro zobrazení/skrytí logovacího panelu je nutné kliknout na položku *Show log* / *Hide log* v hlavní nabídce aplikace.

Pomocí kontextového menu simulace se provádí také nastavování parametrů *rtFactor*



Obrázek 7.12: Model vytvořený pomocí Petriho sítě.

a *stopTime*. Význam těchto parametrů byl popsán v návrhu aplikace . Parametry jsou nastavovány pomocí dialogů, které hlídají validitu zadaných hodnot. Při nastavování *stopTime* parametru je možné, spíše nezbytné, umožnit zadat i textovou hodnotu. *StopTime* totiž bývá většinou definován nekonečnem. Pro toto nastavení se do dialogu musí napsat *inf*, nebo *Infinity*. Při restartování simulace dojde k jejímu zastavení a simulační čas je nastaven na 0. Je proto nutné simulaci opět znovu spustit. Tento fakt není způsoben implementací klienta, ale implementací simulačního jádra SmallDEVS.

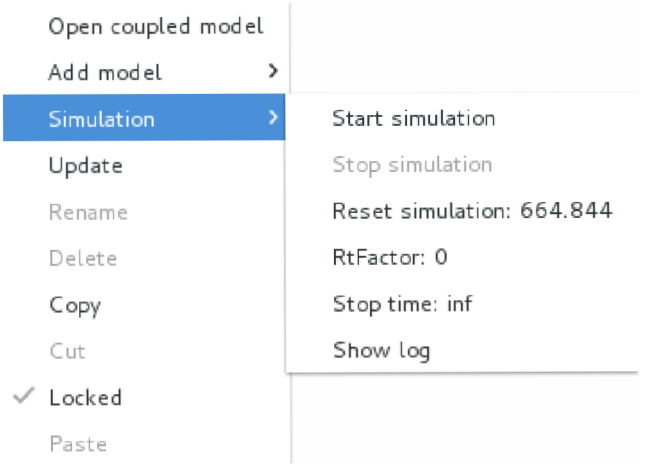
Dále budou popsány jednotlivé prvky logovacího dialogu na obrázku 7.14:

- **(1) Jméno simulace**
  Vizuální vlastnosti záložky simulace jsou stejné jako u záložek editoru popsaného výše. V případě zavření záložky dojde i k odhlášení uživatele z odběru stavových informací o simulaci. Pokud je pomocí kontextového menu simulace zrušeno *Show log*, dojde pouze k odhlášení odběru informací, nikoliv k uzavření této záložky.

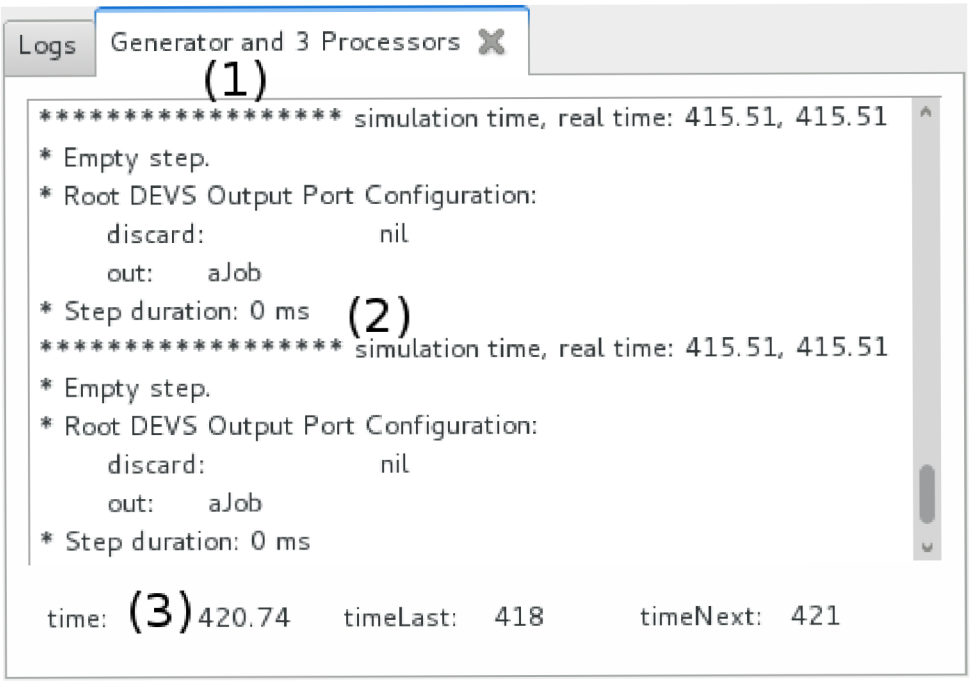
- **(2) Výpis logu**

V této oblasti jsou zobrazovány informace získané z běhu simulace. Nejnovější zprávy jsou přidávány pod již existující a posuvník je nastaven tak, aby byl neustále na posledním, tedy nejnovějším, řádku záznamu.

- **(3) Stavové informace simulace**
  Zde jsou zobrazovány časové informace o přechodech a o aktuálním čase simulace.



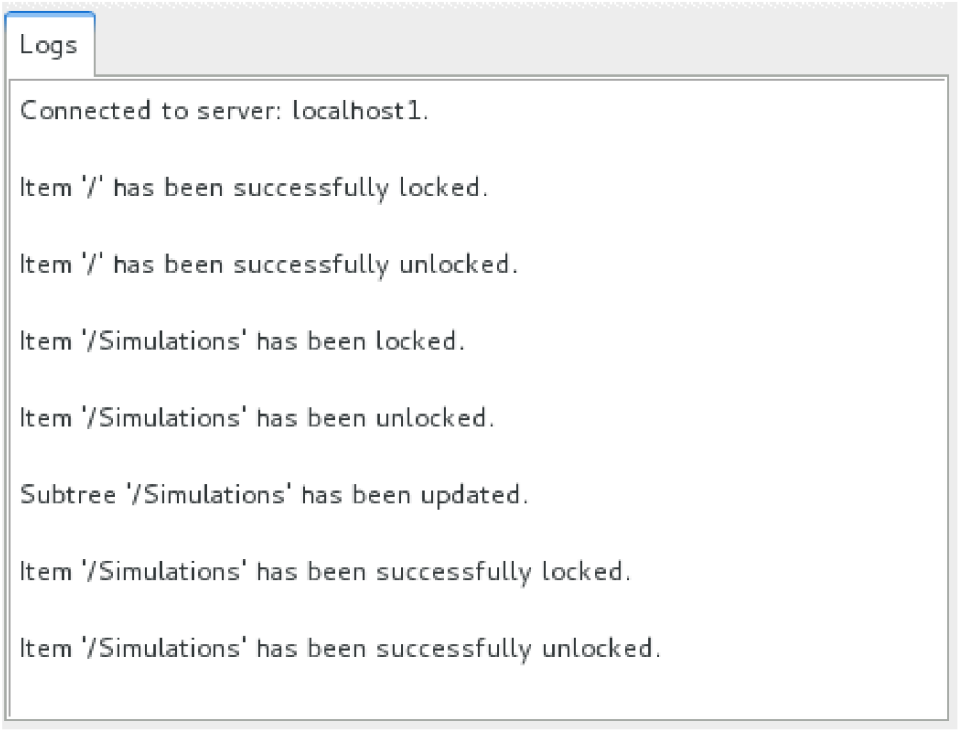
Obrázek 7.13: Menu pro řízení simulace.



Obrázek 7.14: Panel zobrazující stav simulace.

7.2.12 Logování běhu aplikace

Na obrázku 7.15 se nachází panel, v němž jsou zobrazovány logovací informace o běhu aplikace. Například pokud je prvek stromu zamčen jakýmkoli uživatelem, jsou o tom informováni i ostatní přihlášení klienti. Tato záložka nedisponuje možností zavřít, a zobrazuje informace od spuštění do ukončení aplikace. V případě potřeby je možné označit všechny, nebo jen vybrané výpisy, a pomocí klávesy *delete* nebo *backspace* je smazat.



Obrázek 7.15: Panel zobrazující stav simulace.

7.2.13 Provádění změn v modelech

XML soubory se změnami v modelu jsou generovány jednotlivými objekty. Před započetím editace je originální model sklonován. Každý z těchto modelů implementuje operátor == pro určení, zda originální a sklonovaný model obsahuje stejné hodnoty. Každý model obsahuje seznam změn, a to v pořadí, v němž byly vykonány. Konkrétně se tento seznam vztahuje na přejmenovávání, vytváření a mazání modelů nebo portů. Informace o změnách se uchovávají proto, že v případě přejmenování modelu či portu je nutné na klientské straně upravit jméno v nadřazených modelech, které provedou upravení nebo smazání propojení mezi modely.

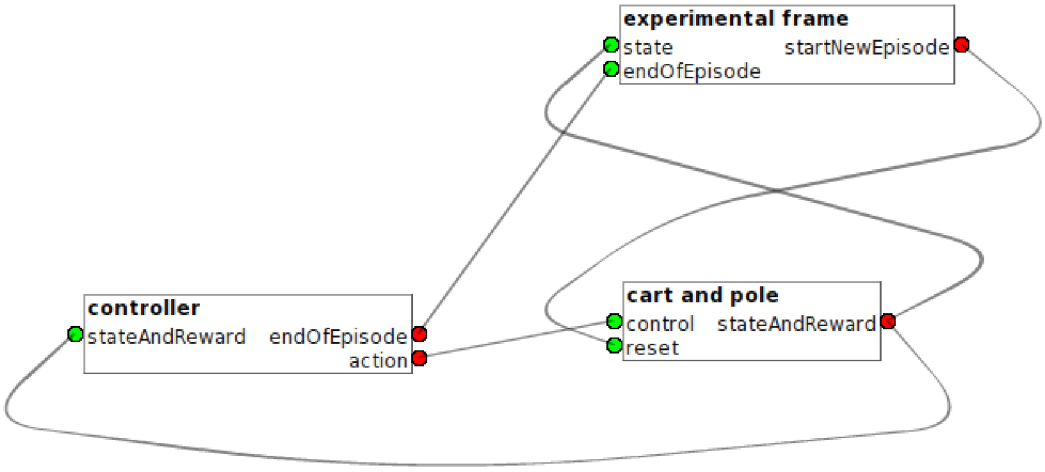
V případě, že je přejmenován model, jež je otevřený v záložce, provede se i příslušná změna titulku záložky.

Kapitola 8

Testování

Testování se provádělo na existujících modelech obsažených ve SmallDEVS. Vytvořená aplikace je schopná vytvářet a upravovat všechny modely, které splňují kritéria Petriho sítí a DEVS modelů. Pří testování byly použity následující modely:

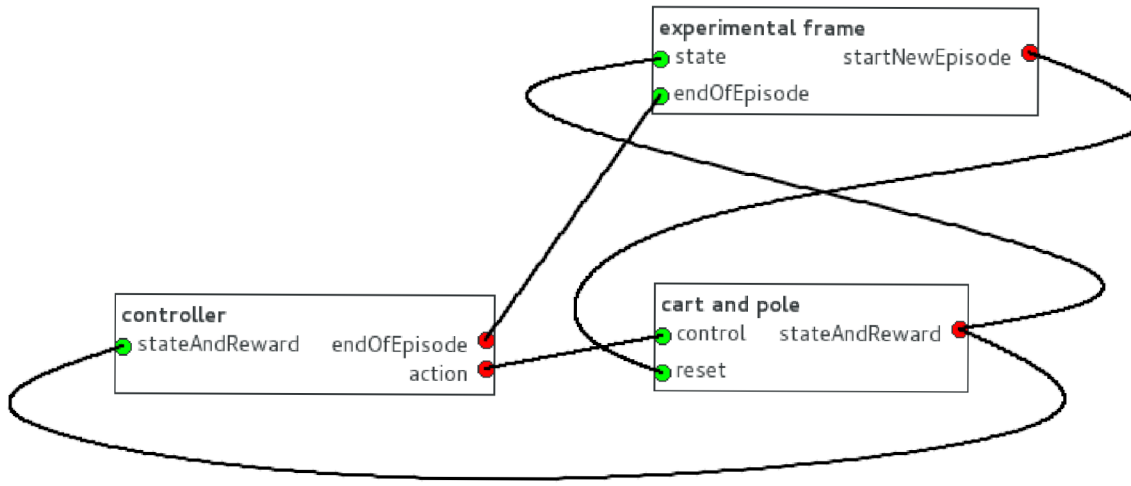
- **Cart-Pole-Control System**
  Tento model řeší problém inverzního kyvadla. Na obrázku 8.1 je ukázán náhled modelu ze SmallDEVS. Na obrázku 8.2 je náhled z klientské aplikace. Oba obrázky jsou pořízeny ze stejného modelu. Je tedy patrné, že byla zachována kompatibilita zobrazení spojnic i pozic modelů. Model (simulace) je definován pomocí spojovaných a atomických modelů.



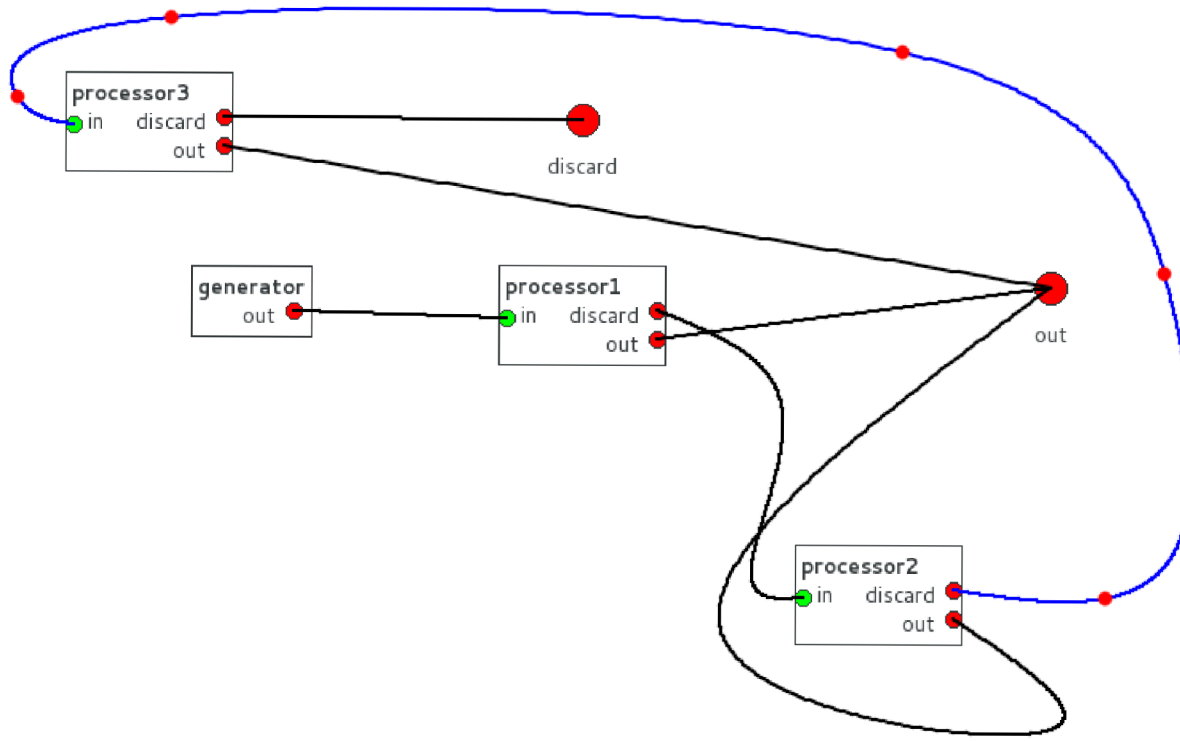
Obrázek 8.1: Model inverzního kyvadla ve SmallDEVS.

- **Generator and 3 processors**
  K testování byly použity i ostatní modifikace tohoto modelu jako je například *Generator and 3 Processors with Shared Behavior*. Na obrázku 8.3 je náhled modelu z klientské aplikace. Model je definován pomocí atomických DEVS.

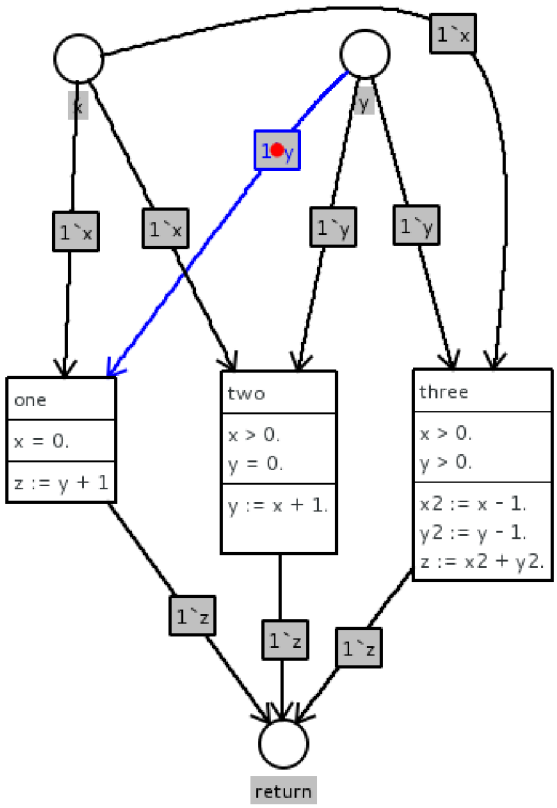
- **Atom definovaný pomocí Petriho sítě**
  Na obrázku 8.4 je ukázán model definovaný pomocí vysokoúrovňové Petriho sítě, který je vytvořen klientskou aplikací.



Obrázek 8.2: Model inverzního kyvadla v klientské aplikaci.



Obrázek 8.3: Model s generátorem a třemi procesory.



Obrázek 8.4: Atom definovaný pomocí vysokoúrovňové Petriho sítě.

Kapitola 9

Závěr

Cílem této práce bylo vytvořit aplikaci, umožňující vzdálený přístup k modelům uloženým v simulačním prostředí SmallDEVS. Aplikace je mimoto schopna ovládat simulace, vytvářet a upravovat nové modely či simulace. Práce je rozdělena do tří částí, které se zabývají návrhem a implementací serveru, protokolu a klienta.

Serverová část je implementována v rámci existujícího prostředí SmallDEVS, které je vytvořeno v jazyce Smalltalk (Squeak). Skrze TCP/IP komunikační protokol nabízí služby simulačního jádra. Server je konkurentní, což znamená, že umožňuje více souběžných spojení. Před umožněním přístupu k uloženým modelům musí proběhnout autentizace uživatele. V případě neúspěchu dojde k uzavření spojení. Oproti původní implementaci SmallDEVS je zde přidána možnost specifikovat atomický model pomocí vysokoúrovňové Petriho sítě. Tento model je abstraktní a umožňuje jen ukládání sítí a neumí danou síť simulovat.

Komunikace mezi klientem a serverem probíhá pomocí textového protokolu, který se skládá ze dvou částí. První částí je příkaz, který neobsahuje mezery. Druhou částí je nepovinný parametr, který je ve většině případů specifikován pomocí XML souboru. Výjimku tvoří příkaz pro určení náhodného řetězce pro autentizaci a stavový příkaz determinující chybu na serveru.

Klientská část aplikace je implementována v jazyce C++ s využitím knihovny Qt. Aplikace umožňuje přistupovat na více serverů souběžně a spravovat jednotlivé modely na nich uložené. Součástí spravování modelů je i možnost kopírovat modely mezi servery. Editace modelů probíhá graficky a to buď pomocí dialogu, v případě atomického DEVS, nebo pomocí pracovní plochy u ostatních, tedy spojovaných modelů, simulací či atomů definovaných pomocí vysokoúrovňových Petriho sítí. V současné implementaci aplikace nepoužívá žádný Smalltalk validátor kódu. V případě, že by validace probíhala na straně serveru, disponuje aplikace předpřipraveným komunikačním rozhraním pro detekci a zobrazování chybových hlášení uživateli. Aplikace dále obsahuje výsuvný panel, ve kterém jsou zobrazovány informace o běhu simulací na serverech a informace z běhu aplikace samotné.

Celá aplikace je přenositelná i na jiné operační systémy (implementace probíhala v Linuxu). Pro klientskou část stačí pouze aplikaci zkompilovat pro cílovou platformu, která samozřejmě musí podporovat knihovnu Qt. Jelikož je server implementován ve Smalltalku, jeho přenos na jinou platformu obnáší pouze změnu virtuálního stroje.

Další rozvoj aplikace by se měl zaměřit na plnohodnotnou implementaci simulací pro vysokoúrovňové Petriho sítě. Dalším rozšířením by mohlo být detekování výjimek z běhu simulací, které jsou v simulačním jádru realizovány ve zvláštním vláknu. Toto ovšem vyžaduje zásah do samotného simulačního jádra.

Literatura

[1] SmallDEVS Kernel API [online cit. 20. 12. 2012]. 2012.
URL http://perchta.fit.vutbr.cz:8000/projekty/25

[2] Digia: Qt official site. 2012.
URL http://qt.digia.com

[3] Janoušek, V.: *Simulace a návrh vyvíjejících se systémů*. Department of Computer Science and Engineering, 2011, 123 s., iSBM 978-80-214-4414-0.

[4] Klir, G.: *Architecture of Systems Problem Solving*. New Yourk: Plenum Press, 1985.

[5] Křivánek, P.: Seriál Squeak: návrat do budoucnosti - Root.cz. 2004.
URL http://www.root.cz/serialy/squeak-navrat-do-budoucnosti/

[6] McLeod, R.; Baart, L.: *Geometry and Interpolation of Curves and Surfaces*. Cambridge University Press, 1998, ISBN 9780521321532.

[7] Smalltalk.org™: Smalltalk.org™| smalltalk | history.html. 2010.
URL http://www.smalltalk.org/smalltalk/history.html

[8] Uwe Rathmann, J. W.: Qwt User's Guide: Qwt - Qt Widgets for Technical Applications. 2013.
URL http://qwt.sourceforge.net/

[9] W3C: Extensible Markup Language (XML) 1.0 (Fifth Edition)[online cit. 25. 12. 2012]. 2008.
URL http://www.w3.org/TR/xml/

[10] Wikipedia: DEVS — Wikipedia, The Free Encyclopedia. 2012, [Online; accessed 23-December-2012].
URL http://en.wikipedia.org/w/index.php?title=DEVS&oldid=528492554

[11] Wikipedia: Bézier curve — Wikipedia, The Free Encyclopedia. 2013, [Online; accessed 3-May-2013].
URL http://en.wikipedia.org/w/index.php?title=B%C3%A9zier_curve&oldid=553237915

[12] ZEIGLER, B.; Prähofer, H.; Kim, T.: *Theory of Modeling [Modelling] and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press, 2000, ISBN 9780127784557.

[13] Češka, M.: *Petriho Sítě*. Akademické nakladatelství CERM Brno, 1994, ISBN 80-85867-35-4.