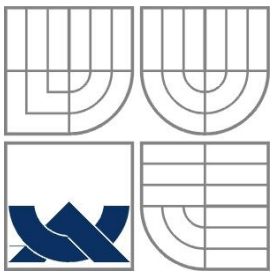
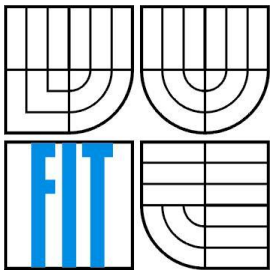


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

EDITOR OBJEKTOVĚ ORIENTOVANÝCH PETRIHO SÍTÍ

EDITOR OF OBJECT ORIENTED PETRI NETS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETER KARLUBÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK KOČÍ, Ph.D.

BRNO 2013

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací nástroje pro editaci a simulaci Objektově orientovaných Petriho sítí. Nástroj podporuje dva formáty zápisu Objektově orientovaných Petriho sítí, a to PNtalk a PNML. Výsledná aplikace v jazyku Java poskytuje dvě hlavní skupiny funkcionalit - vytvářet, načítat a ukládat sítě v těchto formátech a vzdáleně spolupracovat se simulátorem a s jeho využitím zjistit stav simulace a zobrazit ho. První část práce je věnovaná teorii, v které jsou objasněné koncepty spojené s Objektově orientovanými Petriho sítěmi. Ve druhé části se zabýváme popisem implementace a funkcionality implementovaného nástroje pro editaci OOPN a způsobem propojení nástroje se simulátorem.

Abstract

This bachelor thesis deals with the design and implementation of the tool, which allows to edit and simulate Object oriented Petri nets. It supports two formats of Object oriented Petri nets, PNtalk and Petri Net Markup Language. The final tool implemented in Java programming language allows to create, save and load Petri nets in these formats, as well as to remotely cooperate with the simulator and display the simulation state. The first part of this thesis is devoted to theory. Concepts associated with Object oriented Petri nets are explained in the first part. The second part describes the implementation and functionality of the implemented editing tool and the process of connecting the editor with the simulator.

Klíčová slova

Editor Petriho sítí, Objektově orientované Petriho sítě (OOPN), Editor Java, PNtalk, PNML, Petriho sítě

Keywords

Petri Net Editor, Object oriented Petri Nets (OOPN), Java Editor, PNtalk, Petri Net Markup Language (PNML) , Petri Nets

Citace

Karlubík Peter: Editor Objektově orientovaných Petriho sítí, bakalářská práce, Brno, FIT VUT v Brně, 2013

Editor Objektově orientovaných Petriho sítí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Radka Kočího, Ph.D. .
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Peter Karlubík
1.3.2012

Poděkování

Rád bych poděkoval vedoucímu mé práce, Ing. Radkovi Kočímu, Ph.D., za odborný přístup a vstřícnost při spolupráci.

Obsah

1 Úvod

Táto práca popisuje tvorbu nástroja, ktorý bude schopný manipulovať so špeciálnym typom Petriho sieti, Objektovo orientovanými Petriho sieťami(OOPN). Všeobecne sú v práci čitateľovi objasnené konvencie spojené ako aj s klasickými Petriho sieťami, tak aj s OOPN - ich syntax, sémantika a gramatika jazykov, ktoré sa využívajú pre ukladanie týchto sietí do textovej podoby. Tieto poznatky sú základným kameňom pre pochopenie významu nástroja pre úpravu a ukladanie Objektovo orientovaných Petriho sietí.

1.1 Cieľ práce

Prvým cieľom práce je pre kompletné pochopenie aj nezainteresovaných čitateľov objasniť problematiku OOPN. Druhým cieľom je popísať implementáciu a fungovanie nástroja na ich editáciu, ktorý je predmetom tejto práce.

Od nástroja sa požadujú dve základné skupiny funkcionalít a to:

- **editačná** - užívateľsky prívetivé zobrazenie sietí a možnosť ich úpravy, ukladania a načítavania
- **simulačná** - prepojením editora s poskytnutým simulátorom aplikácia umožňuje zobrazenie stavu simulácie daného modelu Petriho siete a ponúka možnosť riadenia simulácie rozhraním poskytnutým v editore.

1.2 Prehľad kapitol

Kapitola dva poskytuje náhľad na kompletné teoretické pozadie Petriho sietí, ich použitie a význam a postupné formovanie až k OOPN. Na úvod sú popísané náležitosti a koncepty Petriho sietí vo všeobecnosti a je uvedený stručný prehľad ďalších typov Petriho sietí. Podrobnejšie sú rozobraté Vysoko-úrovňové (High-Level) Petriho siete a Hierarchické Petriho siete, ktorých koncepty priamo ovplyvnili vývin, a postupný prechod k OOPN.

V tretej kapitole sú čitateľom priblížené formáty v ktorých je Petriho siete možné ukladať a to PNtalk, ktorý bol vyvinutý špeciálne pre popis OOPN a formát Petri Net Markup Language (PNML), ktorý sa používa pre popis všetkých typov Petriho sietí už niekoľko rokov. Kapitola je uzavretá príkladmi sietí uložených v týchto formátoch.

Štvrtá kapitola popisuje funkcionality, ktoré editor podporuje a návrh tried a implementačné postupy, ktoré boli využité pri tvorbe výslednej aplikácie. V záverečnej piatej kapitole môžeme nájsť porovnanie s inými nástrojmi podobného typu.

2 Petriho siete

V tejto kapitole bude pre pochopenie ďalších kapitol najskôr vysvetlené, čo to vlastne Petriho siete sú a postupne prejdeme viacerými typmi Petriho sietí, ktoré ovplyvnili vývin OOPN.

2.1 Základná definícia a typy Petriho sietí

Petriho siete sú najčastejšie používané vo forme P/T sietí a jedná sa o vizuálnu reprezentáciu štruktúry systémov či procesov a ich aktuálneho stavu.

Z grafického hľadiska ide o bipartitný graf, to znamená graf reprezentovaný hranami a dvoma množinami uzlov, v ktorom nemôžeme hranou spojiť dva uzly z tej istej množiny. Z toho vyplýva aj skratka P/T, tá značí Place/Transition alebo miesto/prechod a pomenúva spomínané dva typy uzlov tohto grafu. Z hľadiska logického ide o model diskrétneho distribuovaného systému. Konfigurácia alebo značenie všetkých uzlov typu miesto (place) v danom grafe reprezentuje stavy ktoré systém (jeho procesy) nadobúda/jú. Uzol typu miesto je uzol, v ktorom sa sústredia značky (tokeny) reprezentujúce stavovú informáciu daného miesta. V základných Petriho sieťach sú graficky reprezentované bodkou (dot). Prechod (transition) vyjadruje operáciu ktorá sa vykoná pokiaľ budú splnené vstupné podmienky, to znamená pokiaľ bude vo vstupných miestach dostatok tokenov, alebo podľa typu siete iných premenných či objektov. Táto operácia je spojená s odobraním tokenov zo vstupných miest a s vytvorením výstupu určitého typu (opäť závisí od typu siete, typu prechodu a výstupných podmienok), ktorý je zaznamenaný do výstupných miest. Všetky tieto zmeny sú atomické a diskrétne, to znamená, že vieme presne určiť či je prechod vykonateľný alebo nie a zmena stavu modelu je zaznamenaná skokovo. [1] [4]

Petriho siete sa využívajú najmä v systémovom inžinierstve ako nástroj modelovania rôznych reálnych systémov, ďalej napríklad pri modelovaní klasických aj paralelných procesov, modelovaní paralelného prístupu ku zdieľaným zdrojom a inde. [1] [4]

Formálne môžeme Petriho siete definovať nasledovne: Petriho sieť je štvorica

$$N = (P_N, T_N, PI_N, TI_N) \qquad 2.1$$

kde:

- P_N je konečná množina miest
- T_N je konečná množina prechodov, $P_N \cap T_N = \emptyset$
- $PI_N : P_N \rightarrow N$ je inicializačná funkcia (place initialization function).
- TI_N je popis prechodov (transition inscription function). Je to funkcia, definovaná na T_N, taká, že $\forall\ t \in T_N$:

  $TI_N(t) = (PRECOND_t^N, POSTCOND_t^N)$, kde
  a) $PRECOND_t^N : P_N \rightarrow N$ sú vstupné podmienky (vstupy) prechodu t.
  b) $POSTCOND_t^N : P_N \rightarrow N$ sú výstupné podmienky (výstupy) prechodu t. [1]

Základné Petriho siete podliehali viacerým rozšíreniam, napríklad rozšíreniu o špecifikáciu kapacít miest, rozšíreniu o inhibičné hrany[1], o časované prechody, o farbu určujúcu dátový typ miesta a iné. Vzniknuté varianty môžeme rozdeliť do viacerých kategórií, spomenieme:

[1] inhibičné hrany alebo inhibítory umožňujú testovať miesto na prítomnosť/neprítomnosť značky

- **Stochastické** - využitie pri diskrétnych systémoch, zahŕňajú priority prechodov, pravdepodobnosti a časovanie prechodov
- **Hierarchické** - do seba vnorené siete
- **Vysokoúrovňové Petriho siete** (High-Level Petri Nets - HLPN)
- **Symetrické Petriho siete**
- OOPN [1] [4]

2.2 Modelovanie Petriho sieťami

Mapovanie miest, ich značiek, prechodov a vzájomných väzieb medzi miestami a prechodmi je výsledkom abstrakcie reálneho systému, a teda býva rôzne v závislosti od typu, štruktúry, či účelu modelu. Treba brať do úvahy, že miesta a prechody modelujú statické prvky, zatiaľ čo značky dynamicky vznikajú a zanikajú pôsobením prechodov.

Je teda dôležité si uvedomovať, že miesta sú len pasívnymi prvkami modelu, ktoré sú schopné obsahovať iné objekty (typicky modelujú sklad, komunikačný kanál, databázu a iné), zatiaľ čo prechody reprezentujú aktívne prvky systému. To znamená, že ich aktivita je podmienená stavom systému a stav systému môžu meniť. Typicky modelujú procesor, funkčný blok apod. Ako už bolo spomenuté existencia značiek je podmienená pôsobením prechodov (prípadne inicializačnou funkciou) a ich pôsobením môžu aj migrovať, ide teda o mobilný pasívny objekt ukladaný v mieste. Značky typicky modelujú príznak stavu subsystému, dáta (v prípade Vysokoúrovňových Petriho sietí) apod.

Výhodou Petriho sietí je, že ich konceptuálna jednoduchosť a presný matematický základ umožňujú aplikovať formálne metódy pri validácií modelu, či pri verifikácií krokov pri jeho zjemňovaní. Sugestívna grafová reprezentácia uľahčuje chápanie systému ako celku a tým tiež prispieva k popularite tohto formalizmu. Jednou z nevýhod je, že Petriho sieťami je veľmi obtiažne modelovať systémy s premenlivou štruktúrou. Ďalšou nevýhodou ostáva, že sa jedná o plošný (neštruktúrovaný) model, a tým pádom modeluje systém ako celok, ktorého stav je štruktúrovaný do parciálnych stavov. Hierarchický aspekt tu nie je nijako vyjadriteľný. [1]

S touto bariérou sa vyrovnávajú Hierarchické Petriho siete, o ktorých sa v práci pojednáva v ďalších kapitolách.

2.3 Vysokoúrovňové Petriho siete (HLPN)

P/T siete predstavujú príliš nízkoúrovňový model, ktorý aj jednoduché skutočnosti modeluje príliš zložito, a teda nie sú vhodné pre popis podrobnejšieho modelu. Nie sú preto adekvátnym prostriedkom pre detailné modelovanie reálnych systémov.

Tento problém bola snaha prekonať viacerými modifikáciami (zavedenie globálnych premenných, rôzne špecializované varianty Petriho sietí s pozmenenou sémantikou prechodov slúžiace pre modelovanie systémov z určitej problémovej domény, predikátové Petriho siete[2] a iné).

Práve na predikátových Petriho sieťach stavajú z najväčšej časti vysokoúrovňové Petriho siete (HLPN), ktoré popisujú celý model prostriedkami sietí, bez nutnosti používať inhibítory, priority či globálne premenné. Významným prínosom HLPN je okrem možnosti modelovať obecné výpočty a manipulovať s dátami či dátovými štruktúrami taktiež zachovanie možnosti algebrickej analýzy, ktorá je však omnoho obtiažnejšia. Pre naše potreby budeme HLPN (túto variantu HLPN nazveme HL-sieť) definovať tak, aby štýlovo nadväzovali na definíciu 2.1 a umožňovali modifikácie vysvetľované v ďalších kapitolách. [1]

2.3.1 Základné koncepty HL-siete

Ďalej budú vysvetlené koncepty Vysokoúrovňových Petriho sietí, pretože na nich stavajú aj OOPN.

Ako už bolo načrtnuté, HLPN budú využívať koncepty pôvodných Petriho sietí (miesta, prechody hrany), a najvýznamnejším rozšírením je rozdielna reprezentácia dát v systéme. Už nie je používaný len token obecne, ale dáta sú reprezentované tzv. termami, ktorými rozumieme konštantu alebo premennú.

Hranové výrazy môžu okrem konštánt a premenných obsahovať aj funkcie (lambda-výrazy), ktorých účelom je špecifikovať podmienky naviazania premenných. Za účelom zjednodušenia výrazov na hranách a výpočtov, ktoré môžu značne skomplikovať vyhodnocovanie, sa do sémantiky prechodov zavádza prvok stráž. Stráž je booleovský výraz (predikát), ktorý vyjadruje dodatočnú podmienku pre naviazanie premenných, pre ktoré je prechod vykonateľný.

HL-sieť môžeme definovať tak, že každej hrane je priradená multimnožina[3] obecných výrazov a každému prechodu je priradený strážny výraz. Vykonanie prechodu je možné len pre určité premenné, ktoré sa naviažu na vstupných hranách a v stráži. Je pravidlom, že na hranách sa pre zjednodušenie vyhodnotenia vhodného naviazania premenných používajú čo najjednoduchšie výrazy. Zložitejšie vyhodnotenie vykonateľnosti je potom presunuté do stráže prechodu, kde sa môže použiť obecný výraz s booleovským výsledkom.

Podobne sa dajú zjednodušiť aj výrazy na výstupných hranách do akcie prechodu. Akcia prechodu opäť koncentruje zložitejšie výpočty do jedného uzlu, namiesto toho aby boli použité hranové výrazy. V skutočnosti sa dajú tieto výpočty vykonať aj v stráži, ale akcia ponúka sugestívnejší zápis, tým že odlišuje výpočty nutné k zisteniu vykonateľnosti prechodu od výpočtov, ktoré sa realizujú pri vykonávaní prechodu. Takáto forma prechodu (s akciou a strážou) môže uľahčiť

[2] V predikátových Petriho sieťach značky reprezentujú konkrétne dáta, ktoré sú prechodmi testované prípadne modifikované.
[3] Termy v HL-sieťach sú usporiadané do multimnožín. V podstate sa jedná o množinu umožňujúcu viacnásobný výskyt prvkov (v našom prípade termov).

implementáciu simulátoru a tiež umožňuje modifikáciu sémantiky prechodu spočívajúcu v možnosti jeho neatomického vykonania. Koncepty stráže a akcie sú využité aj v OOPN.

Samozrejmosťou ostáva, že do HL-sietí sa podobne ako do P/T Petriho sietí potenciálne dajú zakomponovať niektoré už spomínané rozšírenia ako napríklad kapacity miest, časovanie prechodov a iné. [1]

2.4 Hierarchické Petriho siete

Aj vo vysokoúrovňových sieťach ostáva prítomný problém plošnosti sietí. To odstraňujú Hierarchické Petriho siete. Tie riešia tento problém dekompozíciou systému na subsystémy, ktoré sa modelujú zvlášť a spoločne tvoria celkový model. Výsledné elementárne systémy potom popíšeme Petriho sieťami. Pri procese dekompozície je treba brať ohľad na potrebu komunikácie každého subsystému s okolím. Tu vzniká riešenie zdieľaním určitých miest a prechodov, ktoré budú súčasťou Objektovej siete, ktorá bude popísaná ďalej v texte.

Oddelenie a pomenovanie čiastkových sietí umožňuje ich viacnásobné použitie statickým inštanciovaním. Tieto čiastkové siete sa v hierarchických Petriho sieťach nazývajú stránky. Každá stránka obsahuje Petriho sieť a definuje uzly, ktorými sa prepája s inými sieťami daného systému. Aj u hierarchických Petriho sietí, napriek zavedeniu statického inštanciovania, platí, že umožňujú modelovať len systémy, ktorých štruktúra sa za behu nemení. Stále platí aj fakt, že jedinou dynamickou zložkou takéhoto systému je značenie v miestach. [1]

2.4.1 Invokácia sietí, invokačný prechod

Spôsob riešenia problematiky modelovania systémov s premenlivou štruktúrou ponúka koncept invokácie sietí. V sieťach sa zavádza tzv. invokačný prechod, ktorý je obdobou programového volania funkcie či procedúry. Úlohou invokačného prechodu je dynamicky inštanciovať určitú sieť, syntakticky podobnú stránke.

Invokačný prechod sa vykonáva neatomicky, dvoma atomickými akciami. Prvou je vytvorenie novej inštancie danej siete, čo zahŕňa odobratie značiek zo vstupných miest invokačného prechodu a umiestnenie týchto značiek do vstupných miest vytvorenej siete. Nasleduje "životný cyklus" invokovanej siete. Tá beží nezávisle od ostatných inštancií sietí v systéme, až kým nedôjde k vopred špecifikovanej ukončujúcej udalosti (vykonanie ukončujúceho prechodu alebo umiestnenie značky v ukončujúcom mieste). Druhou operáciou invokačného prechodu je potom dokončenie invokácie, ktoré spočíva vo vykonaní výstupnej časti invokačného prechodu. Značky zo zdieľaných ukončujúcich miest siete priradených výstupným miestam invokačného prechodu sú skopírované do výstupných miest invokačného prechodu a súčasne dôjde k zrušeniu inštancie invokovanej siete.

Keby sme nadviazali na analógiu invokačného prechodu s volaním funkcie, tak si môžeme vstupné značenie prechodu predstaviť ako parametre funkcie, beh invokovanej siete ako samostatný

bežiaci proces a značenie umiestnené do výstupných miest invokačného prechodu ako výsledok funkcie. [1]

2.5 Objektová orientácia

Keďže existuje viacero definícií princípov či vlastností objektovej orientácie, ktoré sa môžu líšiť, tak sú v tejto kapitole vysvetlené niektoré princípy objektovej orientácie, z ktorých vychádzame v našej práci. [1]

2.5.1 Objekty, operácie, metódy

Objekty predstavujú formálny pohľad na jednotky reálneho sveta. Objekt je charakterizovaný troma zložkami:

1) Stavom - Objekty nesú stavovú informáciu a objekt sa v každom čase nachádza v určitom stave, ktorý sa môže meniť. Stavy objektov sa navzájom neovplyvňujú .
2) Chovaním - Chovanie objektu definuje zmenu jeho stavu za určitých podmienok, či už vonkajších - komunikácia s iným objektom alebo vnútorných.
3) Identitou - Každý objekt má jedinečné meno (identifikáciu), ktorou sa líši od ostatných, a pomocou neho je identifikovateľný v rámci systému.

Rozhranie objektu je určené množinou operácií, ktoré dávajú informáciu o tom, čo objekt vie robiť. Predpis definujúci ako operáciu vykonať sa nazýva metóda, ktorá na rozdiel od operácie nie je súčasťou verejného rozhrania objektu, je teda skrytá vo vnútri objektu. [1]

2.5.2 Inštanciácia a triedy

Väčšina objektovo orientovaných jazykov klasifikuje objekty do tried. Triedy potom reprezentujú množinu objektov s rovnakou vnútornou štruktúrou. Môžeme rozlišovať tri pohľady na to čo je trieda:

1) Trieda je forma pre množinu štrukturálne identických prvkov a prvky vytvorené podľa tohto vzoru sú inštanciami danej triedy.
2) Trieda je objekt, obsahujúci vzor a mechanizmus pre tvorbu inštancií a tie vznikajú aplikáciou zmieneného mechanizmu.
3) Trieda je množina všetkých prvkov, ktoré je možné vytvoriť podľa určitého vzoru, teda je množinou všetkých inštancií tohto vzoru. [1]

Vo väčšine objektovo orientovaných jazykov trieda definuje množinu inštančných premenných alebo atribútov objektu, ktoré sú skryté a množinu viditeľných operácií, ktoré reprezentujú ponúkané služby. Tieto operácie implementujú skryté metódy triedy.

2.6 Objektový model Petriho sietí

Vychádzajúc z definícií uvedených v predošlej kapitole môžeme pokračovať k špecifikácií objektového modelu Petriho sietí.

2.6.1 Objekty, správy, metódy

Systém bude modelovať množinu objektov, ktoré budú počas jeho existencie dynamicky vznikať a zanikať. Každý objekt je možné kedykoľvek bez závislosti na čase identifikovať menom a tak odlíšiť od každého iného objektu, zároveň je možné získať informácie o stave objektu, ktoré sa v čase menia. Zmenu stavu vykonáva sám objekt, teda objekty majú svoj stav vo vlastnej réžií. Samozrejme platí, že objekt môže iný objekt, ktorého identifikáciu pozná, zaslaním správy požiadať o vykonanie nejakej služby, ale cieľový objekt sám na základe obsahu správy zareaguje (vyberie vhodnú metódu, ktorú vykoná) a potenciálne zmení svoj stav. Po dokončení metódy vráti výsledok (opäť identifikáciu objektu) ako odpoveď na správu. [1]

2.6.2 Triedy

Objekty sú popísané triedami, ktoré definujú spoločné vlastnosti príslušnej množiny objektov. Každý objekt je inštanciou určitej triedy a je charakterizovaný identifikáciou (menom) a stavom v danom okamžiku. Triedy poskytujú službu vytvárania inštancií. Každý objekt môže triedu požiadať o vytvorenie jej novej inštancie a týmto spôsobom môžu dynamicky vznikať nové objekty.

Triedy teda majú charakter konštánt - zatiaľ čo existujú staticky, ich inštancie sa podieľajú na dynamickom behu systému. [1]

2.6.3 Funkcionálne štruktúrovanie

Funkcionálne štruktúrovanie je akýmsi medzikrokom k objektovej orientácií, ktorého výsledkom je funkcionálne štruktúrovaná Petriho sieť (FPN).

V podstate ide o vytvorenie alebo zaistenie podobnosti medzi štruktúrou funkcie a štruktúrou siete, ktorej inštancia má byť vytvorená v invokačnom prechode. FPN je samostatne použiteľná, ale jej praktická použiteľnosť je diskutabilná. Jej význam je najmä v skúmaní problematiky a ustálení postupov pre dynamickú inštanciáciu Petriho sietí, analogicky uplatniteľných v objektovo orientovanom štruktúrovaní.

FPN tvorí množina funkcií, ktoré sú buď to primitívne - typicky funkcie zahŕňajúce matematické výpočty vykonané v prechode atomicky, alebo neprimitívne - funkcie definované Petriho sieťou.

Ak je funkcia definovaná Petriho sieťou, tak vykonanie prechodu prebehne z hľadiska činnosti prechodu v troch krokoch - atomickom vytvorení inštancie siete, existenciou/behom siete (ktorý je nezávislý od prechodu ktorý sieť vytvoril) a jej atomickým ukončením a priradením jej výstupov do výstupných miest prechodu - ako bolo popísané v kapitole 2.4.1. Za určitých obmedzujúcich podmienok sa dá pre FPN zostrojiť HL-sieť s ekvivalentným chovaním. Tento prevod je v mnohých smeroch komplikovaný, preto sa to prakticky nevykonáva. [1]

2.7 Prechod k Objektovej orientácií Petriho sietí

Význam funkcionálneho štruktúrovania spočíva v tom, že vnieslo do Petriho sietí znovupoužiteľnosť a dynamiku, samo o sebe je však ťažko použiteľné, preto je doplnené o ďalšie prvky, ktoré už vedú k objektovo orientovanému štruktúrovaniu. [1]

Na obrázku 2.1 je uvedený príklad Objektovo orientovanej Petriho siete a v nasledujúcej kapitole bude pridaný aj jeho popis vo formátoch PNtalk a PNML.

2.7.1 Sieť ako objekt, objektová sieť, sieť metódy

Prvá modifikácia slúžiaca k prechodu k OOPN spočíva v tom, že prechodom je umožnené ovplyvňovať obsah globálne dostupných miest, ktoré môžu byť zdieľané inštanciami všetkých funkcií.

Ďalšou modifikáciou je, že množinu funkcií a miest, ktoré sú funkciami zdieľané, nazveme objektom a umožníme inštanciovať takto vzniknutý model. Funkcie nazveme metódami a zdieľané miesta prehlásime za atribúty. Štruktúra obsahujúca množinu invokovatelných sietí a množinu zdieľaných miest sa tak stala triedou objektov, ktoré sú dynamicky inštanciovatelné. Vytvorené inštancie triedy sú nezávislé objekty s vlastnou identitou.

Keď objekt príjme správu, je dynamicky inštanciovaná sieť odpovedajúcej metódy (odpovedajúca zaslanej operácií). Pre každú inštanciu siete metódy platí, že má prístup k atribútom objektu, v rámci ktorého došlo k invokácií, ale atribúty iných objektov (aj tejto triedy) sú nedostupné. Ďalej bude platiť, že pri vytvorení inštancie siete prechod, ktorý ju vytváral nemusí čakať na dokončenie procesu invokácie ako tomu bolo u definície HL-sietí. Invokačný prechod sa uspokojí s tým, že vytvorí novú inštanciu a že má jej identifikáciu, ktorú môže vo forme značky umiestniť do výstupného miesta.

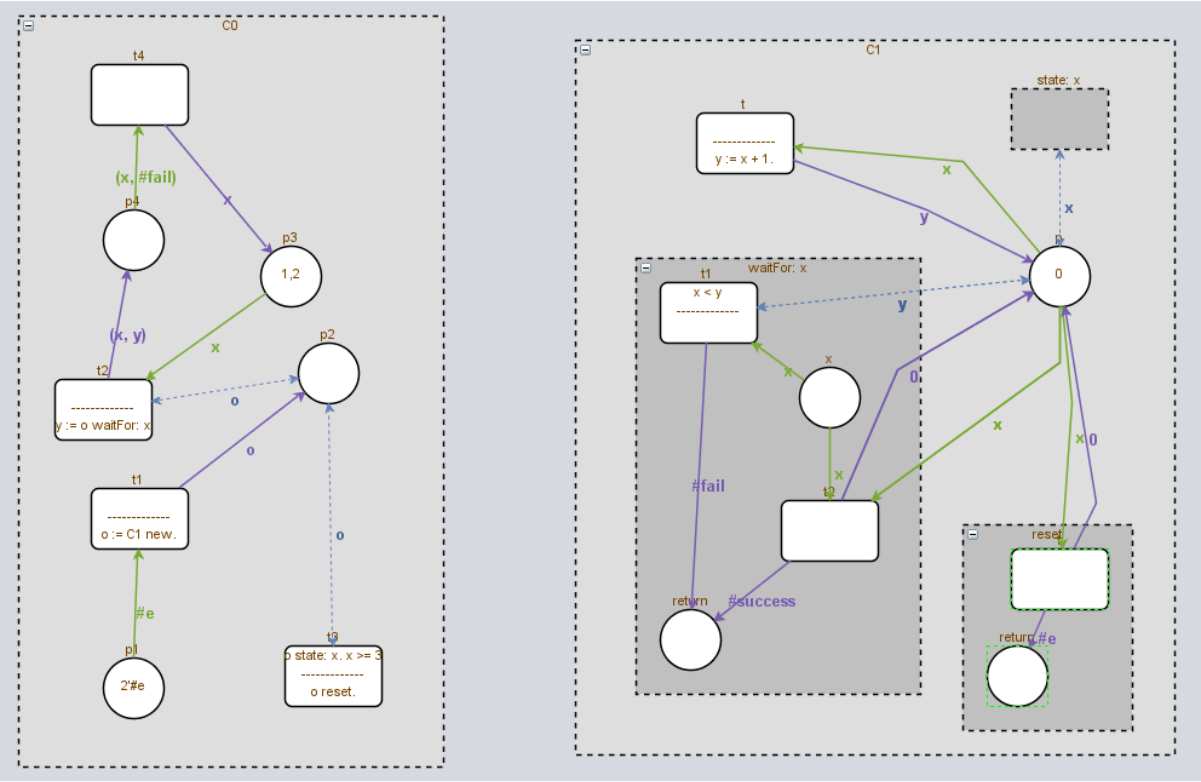
Ak do takejto reprezentácie objektu zakomponujeme aj aktívne prvky - prechody - dostaneme takzvanú sieť objektu, alebo objektovú sieť. Kedykoľvek je objekt (ako objekt uvažujeme inštanciu

siete) vytvorený, vytvorí sa inštancia objektovej siete, ktorá mu umožňuje vykonávať nejakú činnosť samému od seba. Keď objekt príjme správu je invokovaná sieť odpovedajúcej metódy. [1]

2.7.2 Testovacie hrany

Ďalším zavedeným konceptom sú testovacie hrany. Majme miesto **p** a prechod **t**, ktoré sú prepojené testovacou hranou. Sémantika testovacích hrán je v prípade atomického vykonania prechodu **t** ekvivalentná dvojici opačne orientovaných hrán ohodnotených rovnakým hranovým výrazom ako má testovacia hrana, kedy jedna hrana značku odoberie a druhá ju tam hneď vráti.

V prípade neatomického vykonania prechodu dôjde k situácií kedy vstupná hrana ako keby odoberie značku z **p**, ale výstupná ju tam nevráti až kým nie je ukončená invokácia. Podstatou testovacej hrany je, že značka "použitá" touto hranou nezanikne. [1]



Obr. 2.1 - príklad Objektovo orientovanej Petriho siete

3 Formáty uloženia OOPN

V tejto kapitole budú predstavené štandardné formáty OOPN, s ktorými bude výsledná aplikácia pracovať. Implementovaný editor má podporovať dva najčastejšie používané formáty pre ukladanie a načítavanie Objektovo orientovaných sietí: PNtalk a Petri Net Markup Language (PNML).

3.1 PNtalk

Meno jazyka - PNtalk - je odvodené zo spojenia "Petri Nets & Smalltalk". Tento názov má naznačovať, že PNtalk je konkrétnou implementáciou Objektovo orientovaných Petriho sietí a vychádza priamo z definícií uvedených v tomto dokumente, a navyše má niektoré črty jazyka Smalltalk. [1]

Jedným z hlavných rozdielov systému, či jazyka PNtalk a jazyka PNML je, že formát PNtalk je už vo svojej podstate založený na objektovo orientovaných sieťach. Z tohto dôvodu využíva mierne sugestívnejší a jednoduchší zápis konštrukcií spojených s objektovou orientáciou.

Ďalšou významnou odlišnosťou je, že formát PNtalk zanedbáva grafické vlastnosti objektov v graf. Pre nás to znamenalo, že pri načítaní súboru tohto typu bolo potrebné riešiť rozmiestnenie objektov, čo bude popísané v ďalších kapitolách. [1]

Okrem toho, že editor ponúka možnosť v tomto formáte ukladať siete rovnako ako aj z tohto formátu siete načítať, stojí za zmienku aj to že práve v tomto formáte sú siete predávané simulátoru. Jednak je to preto, že ten je implementovaný v jazyku Smalltalk a preto ponúka dobrý základ pre vytvorenie príslušnej štruktúry, na druhú stranu je to jasná voľba aj preto že je založený na objektových sieťach a drží menšie množstvo informácií o modeli (sieťach) - len tie ktoré sú podstatné pre simuláciu. [1]

Okrem inskripčného jazyka definuje aj syntax popisu štruktúry OOPN. Syntax popisu sietí, syntax štruktúry sietí a celého systému tried je formálne definovaná rozšírenou Backus-Naurovou formou.

3.1.1 Štruktúra PNtalk, analógia s objektovou orientáciou

Model v jazyku PNtalk je štruktúrovaný po triedach. Na úvod je označená hlavná trieda (**`main`**), ktorej inštancia sa vytvorí ako prvá pri spustení simulácie. Nasleduje ľubovoľný počet tried (uvedených slovom **`class`** a menom triedy), z ktorých každá obsahuje objektovú sieť (**`object`**) a ľubovoľný počet sietí metód (uvedené slovom **`method`** a názvom danej metódy). Ako už bolo spomenuté jazyk PNtalk už od počiatku počíta s objektovou orientáciou. Jednotlivé triedy preto prirovnať k triedam z definície objektovej orientácie, objektová sieť definuje atribúty triedy a siete

metód definujú metódy danej triedy, ktoré sú volané v prechodoch a vytvárajú inštancie príslušnej siete metódy. [1]

Ďalej ostáva v každej sieti, ako objektovej tak v sieťach metód definovať zoznam miest (kľúčové slovo **`place`**) a prechodov (**`transition`**). Element transition obsahuje zoznam stráží (**`guard`**), akcií (**`action`**) a zoznam vstupných (**`precondition`**), výstupných (**`postcondition`**) a testovacích (**`condition`**) podmienok prechodu. Vstupné, výstupné a testovacie podmienky prechodu vyjadrujú hrany siete náležiace danému prechodu. Element **`place`** obsahuje v zátvorkách uvedený zoznam značiek, ktoré miesto obsahuje po inicializácií. Pre podrobnejšie informácie je možné nahliadnuť do gramatiky jazyka [1].

3.2 Petri Net Markup Language

Jazyk PNML sa využíva k uchovávaniu Petriho sietí viacerých typov už niekoľko rokov. Jeho hlavnou črtou je, že má XML štruktúru, značky sa teda zanorujú a organizujú do úrovní a tie isté značenia (tagy) môžu podľa umiestnenia v dokumente popisovať odlišné veci. [2] [3]

3.2.1 Ukladanie grafických informácií

Jedná o semi-grafický jazyk, to znamená, že okrem vnútorných informácií o uzloch siete - ich inicializačný stav, vzájomné spojenia a iné, uchováva aj vybrané grafické informácie o uzloch. Je už potom na osobe implementujúcej nástroj na prácu so sieťami uloženými vo formáte PNML si vybrať, ktoré z grafických možností sa rozhodne vo svojej aplikácií podporovať. [2] [3]

V PNML sú ako grafické objekty chápané okrem uzlov aj ich menovky, hranové výrazy, menovky hrán a tokenov(značiek) a iné. Všetkým grafickým objektom je podľa definície PNML možné modifikovať určité grafické vlastnosti, ako napríklad definovať pevnú pozíciu, veľkosť, farbu, offset menovky od zvyšku objektu a iné.

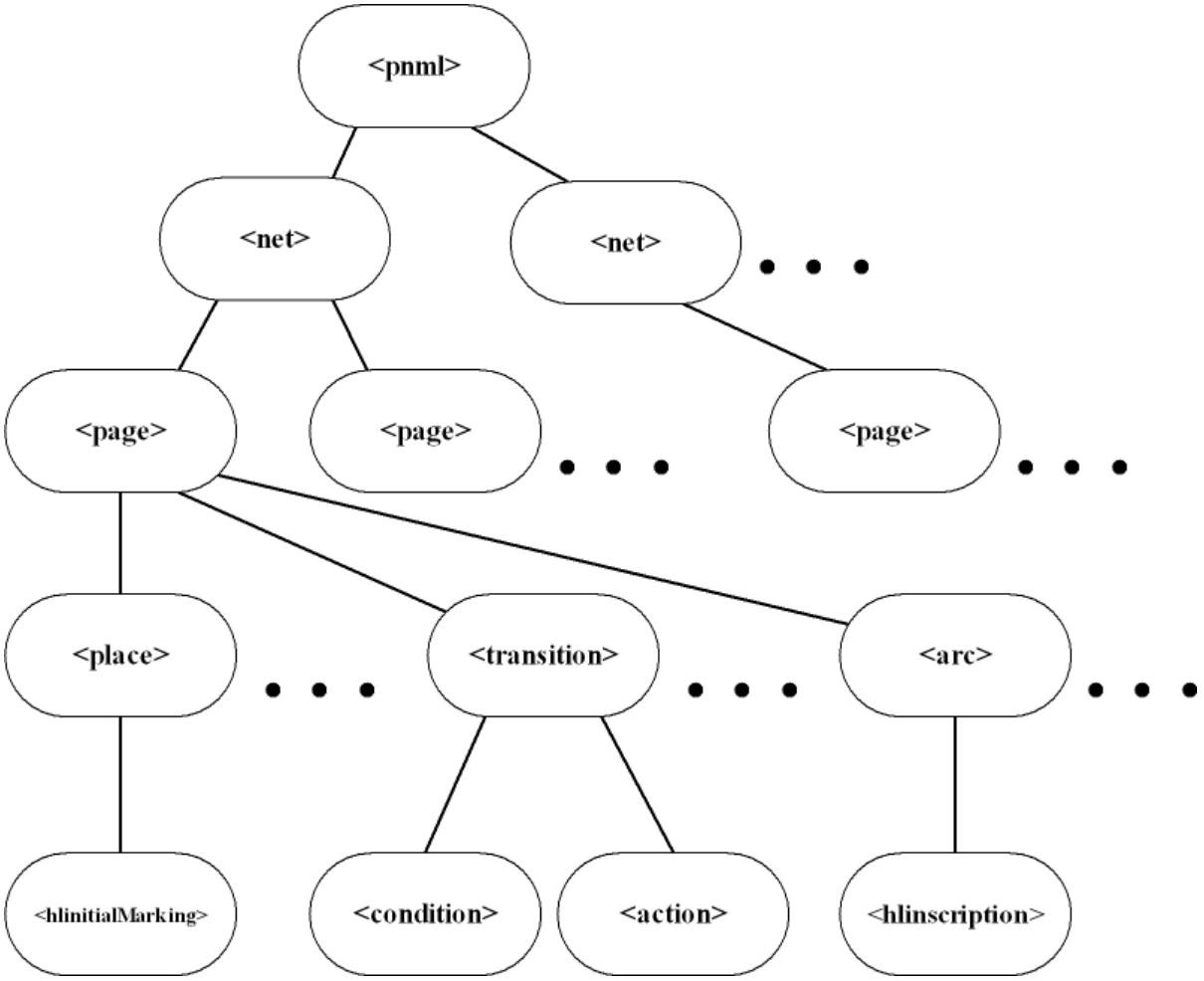
Zatiaľ budeme v našej aplikácie väčšinu z týchto informácií zanedbávať. Editor má svoj pevný grafický štýl, ktorý je momentálne nemenný, v tomto smere sa však naskytuje priestor rozšíreniam. V súčasnosti sú z grafických informácií pre potreby editora podstatné len informácie o pozícií uzlov siete (miest a prechodov) a metód - v PNML reprezentovaných stránkami. [2] [3]

3.2.2 Štruktúra PNML, analógia s PNtalk

Ďalej budú v skratke popísané základné elementy jazyka PNML, pre prehľadnejšie znázornenie je dostupné aj značne zostručnené grafové znázornenie najvýznamnejších elementov PNML dokumentu na obrázku 3.1.

Všetky statické prvky siete, ako aj metódy a siete samotné, majú v popise jazykom PNML svoj identifikátor **id** a potenciálne môžu mať aj grafickú definíciu uvedenú značkou **<graphics>**, meno **<name>** a textový popisok **<text>**. [2] [3]

Koreňový element **<pnml xmlns="url gramatiky pnml">** obsahuje ľubovoľný počet vnorených elementov označujúcich sieť **<net id=" " type="url gramatiky daného**



Obr. 3.1 - Grafové znázornenie PNML súboru

typu siete">. Tie sú ekvivalentom elementu **class** jazyka PNtalk, čiže popisujú danú triedu. Tak ako element **class** jazyka PNtalk, aj element **net** môže obsahovať ľubovoľný počet sietí metód, označených v PNML elementom **<page id=" ">**. Ten obsahuje zoznam miest **<place id=" ">**, prechodov **<transition id=" ">** a hrán medzi nimi **<arc id=" " source=" " target=" ">**. Vstupné a výstupné podmienky prechodov v PNML nie sú odlíšené špecificky ako tomu bolo u PNtalk, ale len na základe toho či je ako zdroj (**source**) hrany uvedený prechod a ako cieľ (**target**) miesto alebo naopak. [2] [3]

Týmto sa stráca možnosť jazykom vyjadriť testovaciu hranu, čo sme sa rozhodli vyriešiť, v jazyku PNML už existujúcim elementom **<text>,** s hodnotou "**condition**" pridaným k hrane ako jej popis. Toto spojenie určuje že sa jedná o hranu testovaciu.

Rovnako bol problém vyjadriť akciu prechodu, ktorá tiež nemá svoju presnú definíciu v jazyku PNML. Jazyk PNML bol preto pre potreby práce rozšírený o element obsahujúci rovnicu akcie prechodu **<action> rovnica </action>.** Tento element musí samozrejme byť vnorený do elementu transition. Toľko ku kľúčovým elementom jazyka PNML, pre podrobnosti je možné nahliadnuť do gramatiky jazyka [2]

3.3 Príklady formátov PNtalk a OOPN

Na záver kapitoly popisujúcej formáty uloženia OOPN si uvedieme príklad formátu PNtalk a PNML, a to konkrétne popis modelu uvedeného na obrázku 2.1.

3.3.1 PNtalk príklad

```
main C0
class C0 is_a PN
object
trans t4
precond p4((x, #fail))
postcond p3(x)
trans t2
cond p2(o)
precond p3(x)
action {y := o waitFor: x}
postcond p4((x, y))
trans t1
precond p1(#e)
action {o := C1 new.}
postcond p2(o)
trans t3
cond p2(o)
guard {o state: x. x >= 3}
action {o reset.}
place p1(2'#e)
place p2()
place p4()
place p3(1, 2)
```

```
class C1 is_a PN
object
place p(0)
trans t
precond p(x)
action {y := x + 1.}
postcond p(y)
method waitFor: x
place return()
place x()
trans t1
cond p(y)
precond x(x)
guard {x < y}
postcond return(#fail)
trans t2
precond x(x), p(x)
postcond return(#success), p(0)
method reset
place return()
trans t
precond p(x)
postcond return(#e), p(0)
sync state: x
cond p(x)
```

3.3.2 PNML príklad

Pre príliš veľký rozsah je uvedený iba popis triedy C0.

```
<?xml version="1.0" encoding="UTF-8" ?>
<pnml   xmlns="http://www.pnml.org/version-
2009/grammar/pnml">
<net id="C0"
type="http://www.pnml.org/version-
2009/grammar/pt-hlpng">
<name>
<text>C0</text>
</name>
<page id="object">
<graphics>
<position x="0" y="0"/>
</graphics>
<transition id="9 3">
<name>
<text>t4</text>
</name>
<graphics>
<position x="60" y="40"/>
</graphics>
</transition>
<transition id="10 3">
<name>
<text>t2</text>
</name>
<graphics>
<position x="30" y="300"/>
</graphics>
<action>
<text>y := o waitFor: x</text>
</action>
</transition>
<transition id="11 3">
<name>
<text>t1</text>
</name>
<graphics>
<position x="60" y="390"/>
</graphics>
<action>
<text>o := C1 new.</text>
</action>
</transition>
<transition id="12 3">
<name>
<text>t3</text>
</name>
<graphics>
<position x="220" y="520"/>
</graphics>
<action>
<text>o reset.</text>
</action>
<condition>
<text>o state: x. x >= 3</text>
</condition>
</transition>
<place id="1 3">
<name>
<text>p1</text>
</name>
<graphics>
<position x="70" y="530"/>
</graphics>
<hlinitialMarking>
<structure>
<numberof>
<subterm>
<numberconstant value="2">
<natural/>
</numberconstant>
</subterm>
<subterm>
<text>#e</text>
</subterm>
</numberof>
</structure>
</hlinitialMarking>
</place>
<place id="2 3">
<name>
<text>p2</text>
</name>
<graphics>
<position x="231" y="270"/>
</graphics>
</place>
<place id="3 3">
<name>
<text>p4</text>
</name>
<graphics>
<position x="70" y="160"/>
</graphics>
</place>
<place id="4 3">
<name>
<text>p3</text>
</name>
<graphics>
<position x="200" y="190"/>
</graphics>
<hlinitialMarking>
<structure>
<add>
<subterm>
<numberof>
<subterm>
<numberconstant value="1">
<natural/>
</numberconstant>
</subterm>
<subterm>
<numberconstant value="1">
<integer/>
</numberconstant>
</subterm>
</numberof>
</subterm>
<subterm>
<numberof>
<subterm>
<numberconstant value="1">
<natural/>
</numberconstant>
</subterm>
<subterm>
<numberconstant value="2">
<integer/>
</numberconstant>
</subterm>
</numberof>
</subterm>
</add>
</structure>
</hlinitialMarking>
</place>
<arc id="p4t4" source="3 3" target="9 3">
<hlinscription>
<structure>
<numberof>
<subterm>
```

```
<numberconstant value="1">
<natural/>
</numberconstant>
</subterm>
<subterm>
<tuple>
<subterm>
<variable refvariable="x"/>
</subterm>
<subterm>
<variable refvariable=" #fail"/>
</subterm>
</tuple>
</subterm>
</numberof>
</structure>
</hlinscription>
<arc id="t4p3" source="9 3" target="4 3">
<hlinscription>
<structure>
<numberof>
<subterm>
<numberconstant value="1">
<natural/>
</numberconstant>
</subterm>
<subterm>
<variable refvariable="x"/>
</subterm>
</numberof>
</structure>
</hlinscription>
</arc>
<arc id="p3t2" source="4 3" target="10 3">
<hlinscription>
<structure>
<numberof>
<subterm>
<numberconstant value="1">
<natural/>
</numberconstant>
</subterm>
<subterm>
<variable refvariable="x"/>
</subterm>
</numberof>
</structure>
</hlinscription>
<arc id="t2p4" source="10 3" target="3 3">
<hlinscription>
<structure>
<numberof>
<subterm>
<numberconstant value="1">
<natural/>
</numberconstant>
</subterm>
<subterm>
<tuple>
<subterm>
<variable refvariable="x"/>
</subterm>
<subterm>
<variable refvariable=" y"/>
</subterm>
</tuple>
</subterm>
</numberof>
</structure>
</hlinscription>
<arc id="t2p2" source="10 3" target="2 3">
<text> condition </text>
<hlinscription>
<structure>
<numberof>
<subterm>
<numberconstant value="1">
<natural/>
</numberconstant>
</subterm>
<subterm>
<variable refvariable="o"/>
</subterm>
</numberof>
</structure>
</hlinscription>
</arc>
<arc id="p1t1" source="1 3" target="11 3">
<hlinscription>
<structure>
<numberof>
<subterm>
<numberconstant value="1">
<natural/>
</numberconstant>
</subterm>
<subterm>
<variable refvariable="#e"/>
</subterm>
</numberof>
</structure>
</hlinscription>
<arc id="t1p2" source="11 3" target="2 3">
<hlinscription>
<structure>
<numberof>
<subterm>
<numberconstant value="1">
<natural/>
</numberconstant>
</subterm>
<subterm>
<variable refvariable="o"/>
</subterm>
</numberof>
</structure>
</hlinscription>
</arc>
<arc id="t3p2" source="12 3" target="2 3">
<text> condition </text>
<hlinscription>
<structure>
<numberof>
<subterm>
<numberconstant value="1">
<natural/>
</numberconstant>
</subterm>
<subterm>
<variable refvariable="o"/>
</subterm>
</numberof>
</structure>
</hlinscription>
</arc>
</page>
</net> // KONIEC C0
.... </pnml> // KONIEC SÚBORU
```

4 Analýza a návrh aplikácie

V tejto kapitole budú čitateľovi priblížené možnosti a funkcionality aplikácie, ďalej návrh tried a spôsob implementácie niektorých častí programu.

4.1 Funkcionality editora

Editor poskytuje dve skupiny funkcionalít. Prvá, označíme ju ako editačnú, zahŕňa užívateľské možnosti typické pre celú skupinu obdobných aplikácií tohto typu a druhá, špecifická práve pre náš nástroj, môže byť označená ako simulačná. Funkcionality sú popísané diagramom prípadov užitia uvedeným na obrázku 4.1.

Ako je viditeľné z diagramu prípadov užitia, hlavným aktérom je užívateľ, ktorý ma k dispozícií dve sady príkazov, a to editačné a simulačné.

Jedno z obmedzení použitia aplikácie je, že tieto sady príkazov nie sú použiteľné súčasne, to znamená pokiaľ je užívateľ v stave editácie modelu tak nemôže spustiť či krokovať simuláciu. Najskôr je nutné vybrať možnosť "začať simuláciu" čo znemožní použitie editačných príkazov a následne pre návrat do "editačného módu" je treba ukončiť simuláciu. Použitie je takto riešené preto, lebo editor posiela simulátoru model určený k simulácií len pri jej začatí, následne už informácie o zmenách v modely za jej behu odosiela len simulátor editoru, a nikdy nie naopak. Tým pádom akékoľvek editácie by počas simulácie nemali zmysel a zbytočne by zmiatli užívateľa. V tomto smere je možné rozšíriť aplikáciu o pridávanie značiek do miest počas simulácie, ktoré má asi najbežnejšie využitie z editácií modelu.

Väčšina editačných príkazov je dostupná prostredníctvom pracovnej lišty s tlačidlami, niektoré sú umiestnené v menu lište a modifikácia konkrétneho objektu je možná výberom akcie z menu, zobrazeného pri kliknutí pravým tlačidlom na daný objekt.

Všetky príkazy určené pre riadenie simulácie sú dostupné prostredníctvom pracovnej lišty s tlačidlami. Momentálne je nastavené pevné pripojenie na simulátor, neskôr môže byť nástroj rozšírený o možnosť modifikácie tohto pripojenia, dostupnú v menu lište.

4.2 Grafická reprezentácia

Rovnako ako pre štruktúru tak aj pre grafickú reprezentáciu Petriho sietí platia určité pravidlá, ktoré síce nie sú presne definované, no napriek tomu sa v nami implementovanej aplikácií snažíme ich dodržať.

Obr. 4.1 - Diagram prípadov užitia

Pre väčšinu typov Petriho sietí platí, že miesta sú graficky reprezentované kruhmi a prechody štvorcami, prípadne obdĺžnikmi. Rovnakú reprezentáciu sme vybrali aj pre náš nástroj.

Každé miesto obsahuje zoznam značiek v ňom aktuálne umiestnených, ktoré je tiež treba graficky zobraziť. Takmer vo všetkej literatúre a existujúcich aplikáciách sú značky zobrazené v textovej podobe vo vnútri miesta (kruhu, ktorý miesto graficky reprezentuje). Rovnaký prístup sme zvolili aj my, s tým že obmedzením pre množstvo zobrazených značiek sú hranice kruhu.

V prechodoch sú sústredené akcie a stráže, ktoré rovnako musia byť zobrazené užívateľovi a opäť platí, že pre prehľadnosť sú obmedzením pre plochu ich zobrazenia hranice prechodu.

Pre úplnosť uvedieme, že napriek tomu že editačná plocha má svoje obmedzenia, užívateľ má možnosť nahliadnuť na kompletný obsah miest a prechodov inými spôsobmi.

Hrany v Petriho sieťach sú orientované, aby bolo jasné, či ide o vstupnú alebo výstupnú podmienku. Toto je dodržané aj v nami implementovanom nástroji, to znamená hrany majú šípky a okrem toho je pre prehľadnosť použité farebné odlíšenie vstupných a výstupných hrán. Navyše, oproti bežným zápisom Petriho sietí, figurujú v OOPN a aj implementovanom nástroji testovacie hrany, ktoré sú reprezentované prerušovanými obojsmerne orientovanými šípkami (šípka na obidvoch koncoch hrany).

Dôležité je aj odlíšiť prvky siete podľa ich príslušnosti k metódam a k triedam. To je riešené ohraničením a farebným odlíšením pozadia tried a metód.

4.3 Návrh tried, kľúčové prvky

Nasledujúca kapitola popisuje návrh tried výslednej aplikácie implementovanej v jazyku Java. Pri implementácií sme sa držali diagramu tried uvedeného na obrázku 4.3. Pri tvorbe užívateľského rozhrania a aplikácie ako takej bol kladený dôraz na jej prenositeľnosť a jednoduchosť, z toho dôvodu boli pre tvorbu užívateľského rozhrania využité základné grafické nástroje zahrnuté v knižniciach **`Swing`** a **`AWT`**. Editačná časť užívateľského rozhrania, v ktorej je zobrazený výsledný model, využíva voľnú knižnicu **`JGraphX`**, ktorej niektoré špecifiká budú popísané ďalej.

4.3.1 Trieda EditorPN, rozhranie aplikácie

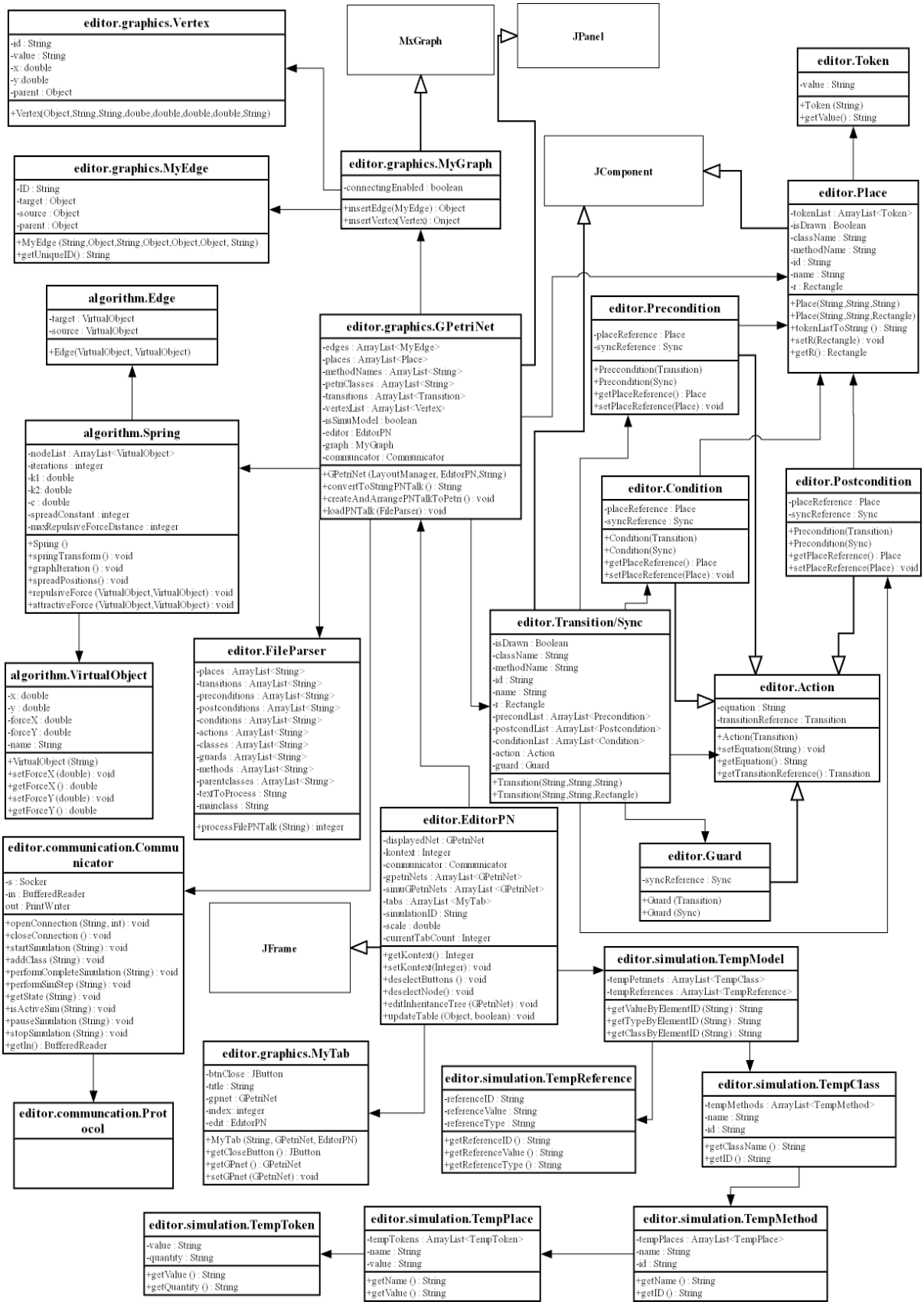
Hlavnou triedou celej aplikácie je trieda **`EditorPN`**, rozširujúca **`JFrame`**. Má v réžií kompletné ovládanie aplikácie a je v nej implementované užívateľské rozhranie. Schéma rozhrania je zobrazená na obrázku 4.2.

Obr. 4.2 - Schéma výslednej aplikácie

Z obrázku je viditeľné, že rozhranie editora má šesť hlavných zložiek:

- menu lištu obsahujúcu príkazy pre základne ovládanie aplikácie
- nástrojová lišta s tlačidlami na ovládanie aplikácie - editáciu a riadenie simulácie
- panel na ľavej strane hierarchicky zobrazujúci objekty aktuálne zvoleného modelu/aktuálne inštancie sietí vytvorené počas simulácie
- centrálny panel pre tvorbu/editáciu/zobrazenie modelu
- panel na pravej strane zobrazujúci informácie o aktuálne zvolenom objekte modelu
- textový panel v spodnej časti okna, ktorý poskytuje informáciu o aktuálne zvolenom príkaze, prípadne dodatočné informácie o označenom objekte

Pracovné lišty majú pomerne priamočiare použitie a ikony boli vybraté tak aby užívateľ aspoň z časti znalý problematiky vedel aplikáciu na prvý pohľad ovládať. Nie je preto nutné ich nejako detailne popisovať. Stačí pripomenúť, že pracovné lišty umožňujú užívateľovi vykonať všetky akcie



Obr. 4.3 - Návrh tried

popísané diagramom prípadov užitia. Je možné aplikáciu rozšíriť o súbor okien s nápoveďou zobrazovaných pri štarte (tooltips), pre efektívnu prácu s aplikáciou.

Panel na ľavej strane okna aplikácie pre prehľadnosť počas editácie užívateľovi hierarchicky zobrazuje zoznam objektov modelu, organizovaných podľa ich príslušnosti k metódam a ku triedam. Pri označení konkrétneho objektu siete je v modely pre rýchle nájdenie tento objekt zvýraznený. Počas simulácie panel zabezpečuje hierarchické zobrazenie aktuálne vytvorených inštancií modelu podľa ich príslušnosti ku triedam.

Panel na pravej strane prehľadne zobrazuje informácie o označenom objekte vo forme tabuľky, neumožňuje však tieto informácie meniť. To musí užívateľ spraviť zvolením príslušnej položky z **`JPopupMenu`** (viac v kapitole 4.3.3).

Centrálny panel pre zobrazovanie samotných modelov je implementovaný pomocou **`JTabbedPane`**, takže aktuálne otvorené modely sú vkladané do záložiek a vytváranie inštancií triedy **`GPetriNet`**, ktorá je popísaná v nasledujúcej kapitole, je v réžii triedy **`EditorPN`**. Keďže editor musí byť schopný zobraziť ktorýkoľvek z aktuálne načítaných modelov, je nutné si na ne uchovávať odkazy, a to je riešené zoznamom inštancií triedy **`GPetriNet`**.

Výsledná podoba aplikácie je zobrazená na obrázku 4.4.

4.3.2 Trieda GPetriNet

Prvou z hlavných úloh triedy **`GPetriNet`** je združovať informácie o uzloch daného modelu. Jej najdôležitejšími atribútmi sú tým pádom zoznam miest (trieda **`Place`**) a zoznam prechodov (trieda **`Transition`**) modelu. Treba spomenúť, že dve zmienené triedy **`Place`** a **`Transition`** zabezpečujú len logickú reprezentáciu uzlov siete.

Ďalšou úlohou triedy je informácie o objektoch modelu nejako užívateľsky prijateľne zobraziť. Prvotné riešenie bolo vytvoriť triedy pre grafické objekty siete. Tieto triedy by rozširovali triedu **`JComponent`** a reprezentovali by jednotlivé inštancie tried **`Place`** a **`Transition`**. Následne by inštancie týchto tried boli vykresľované pomocou nástrojov poskytovaných toolkitom **`Swing`**. Toto riešenie sa však neosvedčilo z viacerých dôvodov a skutočná grafická reprezentácia je zabezpečená prostredníctvom už spomínanej knižnice **`JGraphX`**.

Základnou triedou knižnice **`JGraphX`** je trieda **`MxGraph`**, ktorá je koreňovou triedou akéhokoľvek grafu vytvoreného použitím knižnice **`JGraphX`**. Trieda **`GPetriNet`** z toho dôvodu rozširuje triedu **`JPanel`**, čo umožňuje do nej vložiť inštanciu triedy **`MxGraph.`** V skutočnosti je vkladanou triedou nami implementovaná trieda **`MyGraph`** rozširujúca **`MxGraph`** o pár detailov, špecifických pre potreby aplikácie, umožňujúcich vhodnejšie narábanie s modelom.

Načítanie modelu funguje tak, že sa vytvorí inštancia **`GPetriNet`**, zo súboru sa načítajú potrebné údaje o uzloch, uložia sa v inštanciách tried **`Place`** a **`Transition`**, vytvorí sa inštancia

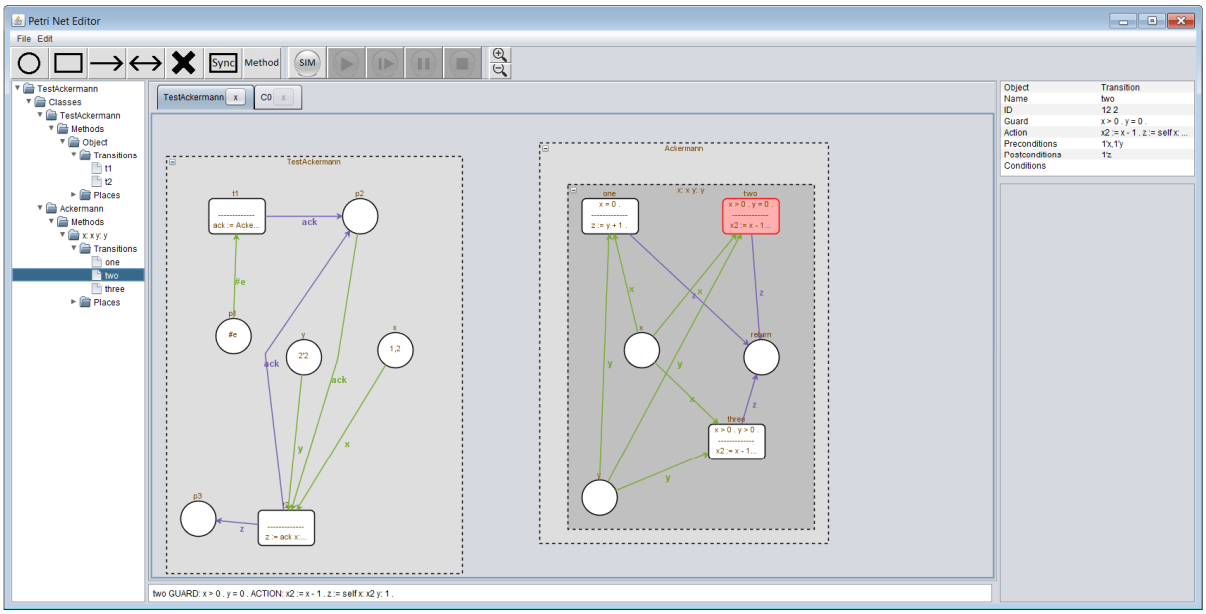
triedy **MyGraph** a zavolá sa metóda, ktorá inštancií **MyGraph** predá potrebné informácie pre vytvorenie grafickej reprezentácie modelu.

Pre vytvorenie odpovedajúcej grafickej reprezentácie modelu je samozrejme potrebné poznať súradnice jednotlivých objektov. V kapitole 3 boli predstavené formáty, v ktorých je možné modely ukladať a bolo spomenuté, že formát PNtalk zanedbáva súradnice objektov. Problém priradenia súradníc objektom, alebo náhodného rozloženia objektov takým spôsobom aby bol model prehľadný, riešia metódy implementované v triede **Spring** (súčasť balíka **algorithm**) viď 4.3.4. Predtým než bude rozobraná problematika rozmiestňovania objektov budú ešte pre celistvosť popísané už spomínané triedy **Place** a **Transition**.

4.3.3 Triedy Place a Transition

Užívateľ je slobodný presúvať a upravovať objekty grafu zobrazeného pomocou inštancie triedy **MyGraph**, je však nutné tieto zmeny zaznamenať a uložiť nové informácie o objektoch, pretože v budúcnosti užívateľ môže uložiť model a pri ukladaní sa v skutočnosti spracovávajú atribúty triedy **GPetriNet** a nie reálne vyobrazené objekty modelu. Pre účel uchovávania informácií o uzloch modelu, ako napríklad aktuálnej pozície v prípade načítania či ukladania modelu slúžia tieto dve triedy.

Samozrejme okrem pozície objektu obsahujú tieto triedy samotné informácie definované pri tvorbe siete. Čiže trieda **Transition** obsahuje zoznamy podmienok prechodu (**Precondition**, **Postcondition**, **Condition**), stráže (**Guard**) a akcie (**Action**) a trieda **Place** zoznam tokenov, reprezentovaných triedou **Token**. Tieto informácie je možné meniť pravým kliknutím na konkrétny objekt a výberom príslušnej vlastnosti, ktorú chceme modifikovať.



Obr. 4.4 - Ukážka výslednej aplikácie

4.3.4 Balík algorithm, rozloženie objektov bez známych súradníc

Jedným z náročnejších implementačných problémov, ktoré bolo treba riešiť, bol problém rozloženia objektov pri načítaní modelu uloženého vo formáte PNtalk.

4.3.5 Výber vykresľovacieho algoritmu

Všeobecne problematiku vykresľovania grafov rieši viacero typov algoritmov. Na základe predpokladaných parametrov výsledného grafu vytvoreného v našej aplikácií bolo nutné sa rozhodnúť pre niektorý z existujúcich vykresľovacích algoritmov. O grafe môžeme prehlásiť nasledovné tvrdenia:

- jedná sa o orientovaný graf
- pre funkčnosť uzlov je zvykom ich prepojiť do nejakého celku (napríklad do siete metódy), čiže samostatne existujúce uzly nemajú v Petriho sieťach príliš veľký zmysel ani časté použitie
- spravidla sa nejedná ani o cyklický ani acyklický graf

a sformulovať nasledovné požiadavky na graf:

- nezáleží, či je "vedený" nahor alebo nadol, väčšinou ani nie je možné ho definitívne preorientovať na niektorú z týchto variánt, kvôli násobným hranám medzi uzlami
- ideálne je dostať čo najmenší počet pretínajúcich sa hrán a rozložiť graf na dostatočne veľkú plochu kvôli prehľadnosti
- dobré je dosiahnuť približne rovnaké rozostupy medzi jednotlivými uzlami grafu - pri modifikácií aktuálne použitého vykresľovacieho algoritmu by bolo možné uplatniť jednotnú dĺžku hrán, v súčasnosti to nástroj nepodporuje

Vyhovieť všetkým požiadavkám na vzhľad grafu je pri komplexnejších grafoch veľmi náročné. Je však dôležité vybrať najvhodnejšiu alternatívu. Ako sme spomenuli existuje viacero prístupov na vykresľovanie a rozmiestňovanie uzlov grafu:

- topologický prístup - vytvára graf určitého tvaru, používa sa pravouhlá sieť hrán
- hierarchický prístup - graf organizovaný do stromovej štruktúry
- "viditeľnostný" prístup (visibility approach) - využíva nadväzujúce hrany alebo inak povedané hrany obsahujúce zlom, rozdeľuje graf na nepretínajúce sa "dlaždice" jednotlivých uzlov
- prírastkový prístup (augmentation approach) - postupne rozširuje graf o ďalšie uzly

- silami riadený prístup (force-directed approach) - priraďuje uzlom vnútornú energiu, na základe toho, či sa s okolitými priťahuje (uzly sú prepojené) alebo odpudzuje (navzájom neprepojené uzly). Hľadá sa bod rovnováhy, kedy bude vnútorná energia čo najmenšia, ideálne nulová.
- prístup rozdeľ a panuj (divide and conquer approach) - rozdelí graf na podgrafy, prístup vhodný pre stromové grafy [5]

Automaticky boli pri výbere vhodného prístupu pre vykresľovanie a rozloženie uzlov grafu s ohľadom na potreby editora zavrhnuté prístupy/algoritmy určené pre vykresľovanie stromových grafov. Pre graf reprezentujúci Petriho sieť tiež nie je vhodné využívať ortogonálne hrany a je žiaduce dosiahnuť relatívne pravidelné rozostupy medzi uzlami, s ohľadom na to, ktoré sú navzájom prepojené a ktoré nie. Po zhodnotení vyšiel ako najvhodnejší silami riadený prístup (force-directed approach), konkrétne algoritmus založený na strunách (spring-based). Jednak preto, že vo väčšine prípadov dostatočné spĺňa požiadavky na výsledný graf ako aj pre jeho jednoduchú implementáciu a prijateľnú rýchlosť. [5]

4.3.6 Silami riadený prístup vykresľovania grafu, strunový algoritmus

Algoritmy založené na silami riadenom prístupe majú analógiu vo fyzike. Na graf sa nahliada ako na systém vzájomne prepojených telies medzi ktorými pôsobia sily - príťažlivé a odpudivé. [5] [6]

Cieľom všetkých algoritmov pre rozmiestnenie uzlov grafu je nájsť ich optimálnu konfiguráciu. U strunového algoritmu, je toto dosiahnuté tak, že sa vo viacerých iteráciách prechádzajú všetky uzly grafu a spočítava sa ich vnútorná energia. Tá je ovplyvnená tuhosťou pružiny, ktorá priamo ovplyvňuje silu akou sa objekty priťahujú a vzájomným odporom častíc - uzlov grafu.[5] [6]

Môžeme si to predstaviť tak, že keď sú uzly pôsobením algoritmu pritlačené k sebe tak je pre ne prirodzené sa zase oddialiť pôsobením ich prirodzenej odpudivosti, ako napríklad u magnetov natočených na seba rovnakým pólom. Naopak medzi uzlami, ktoré sú prepojené hranou pôsobí pri ich prílišnom vzdialení opačný druh sily, príťažlivá, to znamená keď je pružina príliš natiahnutá tak sú jej konce priťahované k sebe. Výpočet týchto síl vždy závisí od aktuálnej vzdialenosti uzlov. [5] [6]

4.3.7 Trieda VirtualObject, trieda Spring

Pre potreby nájdenia vhodného rozmiestnenia uzlov grafu boli vytvorené triedy **`VirtualObject`** a **`Spring`**. Inštancie triedy **`VirtualObject`** sú virtuálnou reprezentáciu uzlov pre potreby pružinového algoritmu (drží napríklad informácie o vnútornej energií uzlu a aktuálnych

súradniciach) a v triede **`Spring`** sú implementované metódy potrebné pre výpočet jednotlivých síl pôsobiacich na uzly a metóda **`springTransform`** implementujúca pružinový algoritmus.

4.3.8 Zhrnutie priebehu načítania modelu bez udaných súradníc

Prvým krokom je načítanie súboru vo formáte PNtalk. Pri načítaní súboru sa najskôr vytvoria odpovedajúce logické uzly, reprezentované inštanciami tried **`Place`** a **`Transition`**.

Následne sú vytvorené virtuálne uzly grafu, pre ktoré sú počas behu pružinového algoritmu prepočítavané hodnoty ich vnútornej energie a upravované súradnice.

Po skončení behu algoritmu (po určitom počte iterácií výpočtu vnútornej energie), sú súradnice virtuálnych objektov predané odpovedajúcim logickým uzlom a grafickým uzlom grafu, ktoré sú súčasťou inštancie triedy **`MyGraph`**. Akékoľvek ďalšie zmeny vykonané v grafickom prostredí sú zaznamenané v logickej reprezentácií príslušného uzlu, takže napríklad pri uložení vo formáte PNML sú vždy uložené aktuálne súradnice všetkých objektov.

4.4 Simulátor, vzdialené riadenie simulácie

Editor sám o sebe je len čiastkovým nástrojom pre prácu s OOPN. Pre jeho plné využitie je potrebné spojenie so simulátorom OOPN. Aj bez neho má aplikácia svoje pole využiteľnosti, no značne obmedzené.

Pre potreby práce bol dodaný simulátor implementovaný v jazyku Smalltalk. Simulátor funguje ako samostatná aplikácia bežiaca v open-source Smalltalk prostredí, Pharo.

Pred spustením simulácie je najskôr nutné zaslať simulátoru jednotlivé triedy modelu a určiť hlavnú triedu. Simulátor si ich spracuje, uloží a následne už editor nemusí simulátoru posielať zo svojej strany žiadne údaje o stave v siet. Správy, ktoré je možné zaslať simulátoru zahŕňajú:

- už spomínané pridanie triedy a nastavenie hlavnej triedy
- začatie simulácie
- vykonanie kroku simulácie
- spustenie behu simulácie (simulátor dôjde až ku koncovému stavu siete)
- získanie aktuálneho stavu simulácie
- zastavenie simulácie
- zrušenie simulácie - z repozitára bežiacich simulácií držaných na strane simulátora
- dotaz, či je simulácia aktívna (použitie pri kompletnej simulácií modelu pre zistenie, či už simulácia dosiahla finálny stav)

Simulátor reaguje na správy zaslané druhou stranou vždy potvrdením úspechu/neúspechu vykonania požadovanej akcie, a prípadne zaslaním požadovaných údajov (napríklad stavu simulovaného modelu).

Aktuálne je prednastavené vytváranie spojenia so serverom bežiacim na rovnakom stroji ako editor, komunikácia prebieha cez port 12345. Do menu lišty editora je žiaduce v budúcnosti pridať možnosť pre nastavenie parametrov spojenia.

Komunikácia so simulátorom by sa v budúcnosti dala rozšíriť o možnosť vybrať si v editore z aktuálne vykonateľných prechodov ten ktorý má byť vykonaný. V tom prípade by simulátor zasielal editoru zoznam vykonateľných prechodov a editor simulátoru údaje o aktuálne zvolenom objekte.

4.4.1 Balík simulation

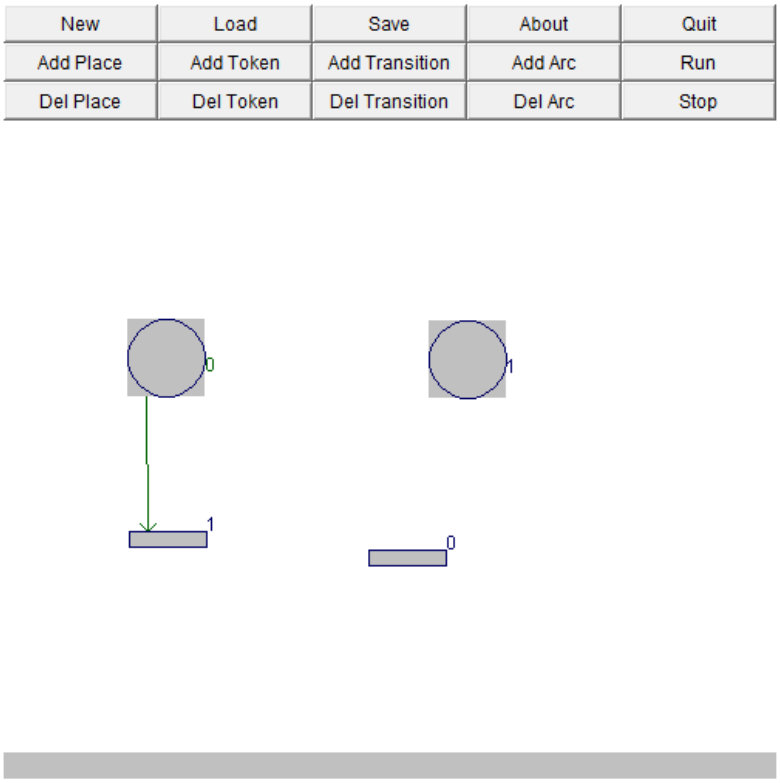
Zmeny v simulovanom modely sú editoru zasielané vo formáte sixx, využívanom v jazyku Smalltalk. Pre potrebu načítania modelu je v balíku **`simulation`** vytvorená ďalšia špeciálna reprezentácia siete, ktorej jediným účelom je poskytnúť štruktúru pre uloženie informácií o stave sietí počas simulácie, a tá je následne prevedená do klasickej štruktúry ako pri načítaní modelu. Prepojenie jednotlivých tried balíka je jasné z návrhu tried, uvedeného na obrázku 4.3.

5 Porovnanie s inými aplikáciami

Pri tvorbe editora sme sa jednak inšpirovali riešeniami z iných nástrojov a na druhú stranu sme sa snažili vyvarovať chýb alebo nedokonalostí, ktoré v nich boli viditeľné z užívateľského hľadiska. Podstatné bolo najmä nájsť vhodné riešenia pre užívateľské rozhranie a zabezpečiť pohodlné a intuitívne ovládanie aplikácie.

Väčšina podobných aplikácií, ktorými sme sa mohli inšpirovať je implementovaných v jazyku Java.

Najprv sme nahliadli na rôzne amatérske voľne dostupné aplikácie ako napríklad **Free online Petri Net Simulator**. Jedná sa o naozaj základný online nástroj pre tvorbu a simuláciu P/T Petriho sietí, ako je viditeľné aj z obrázku 5.1.



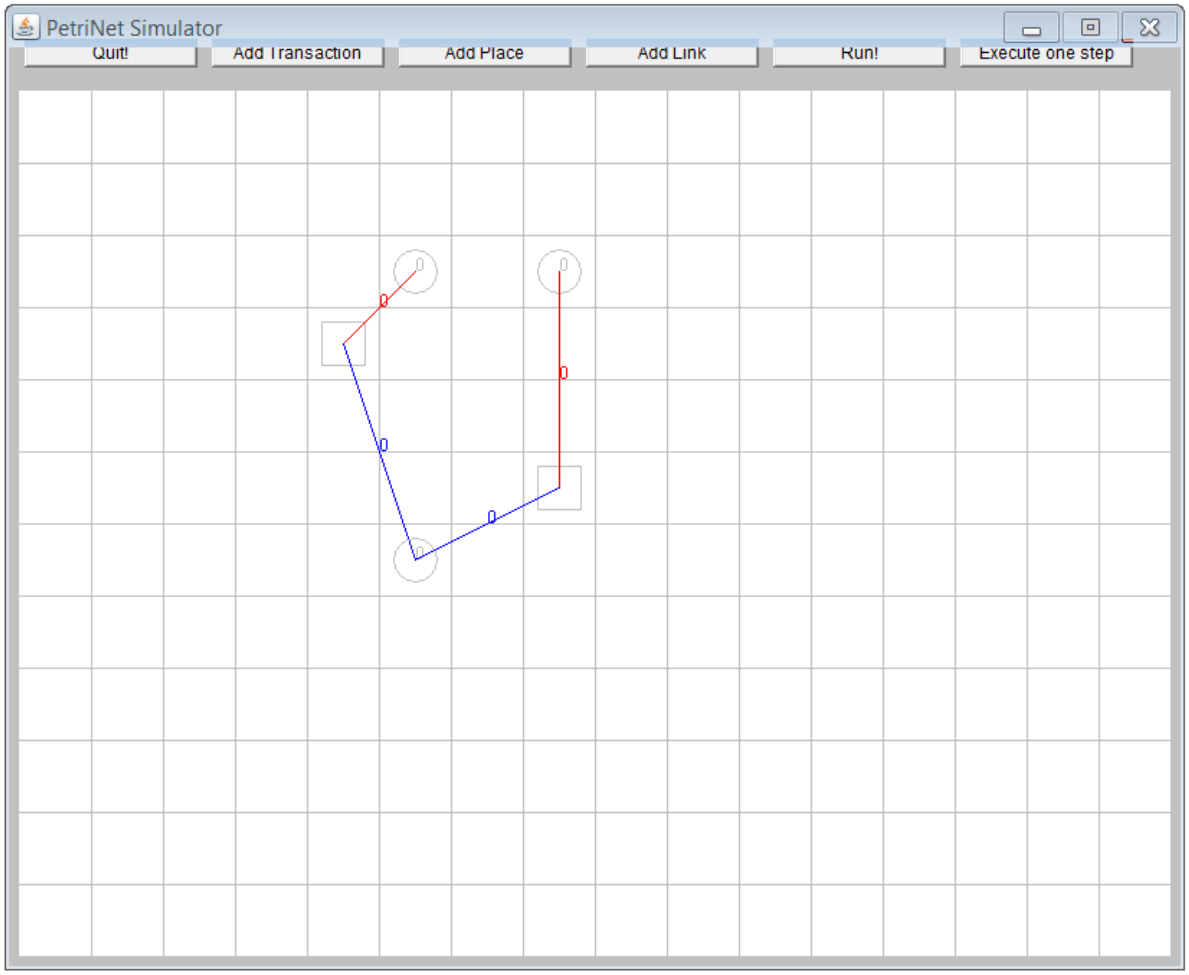
Obr. 5.1 - Free online Petri Net Simulator

Odhliadnuc od neprívetivej grafickej reprezentácie siete, je nie príliš vhodné prevedenie menu aplikácie. Príkazy pre prácu s aplikáciou (ako napríklad save, load či quit) a pre editáciu siete sú preto v našom nástroji oddelené.

V nástroji je nutné pri každom pridaní objektu znova stlačiť príslušné tlačidlo pre pridanie ďalšieho, čo bolo brané z našej strany ako nepohodlné (hlavne pri vytváraní novej siete). V nami vytvorenej aplikácií je preto funkcia označeného tlačidla v platnosti, až kým výber nie je zrušený výberom iného tlačidla alebo kliknutím pravého tlačidla myši. Je totiž jednoduchšie kliknúť pre

prípadné zrušenie funkcionality pravým tlačidlom myši ako sa znova presúvať s myšou do pracovnej lišty a vybrať ho v prípade, že chceme opäť vykonať tú istú akciu.

Ďalším z nástrojov nižšej úrovne je napríklad študentský projekt McGill univerzity v Kanade, ktorého ukážka je na obrázku 5.2.

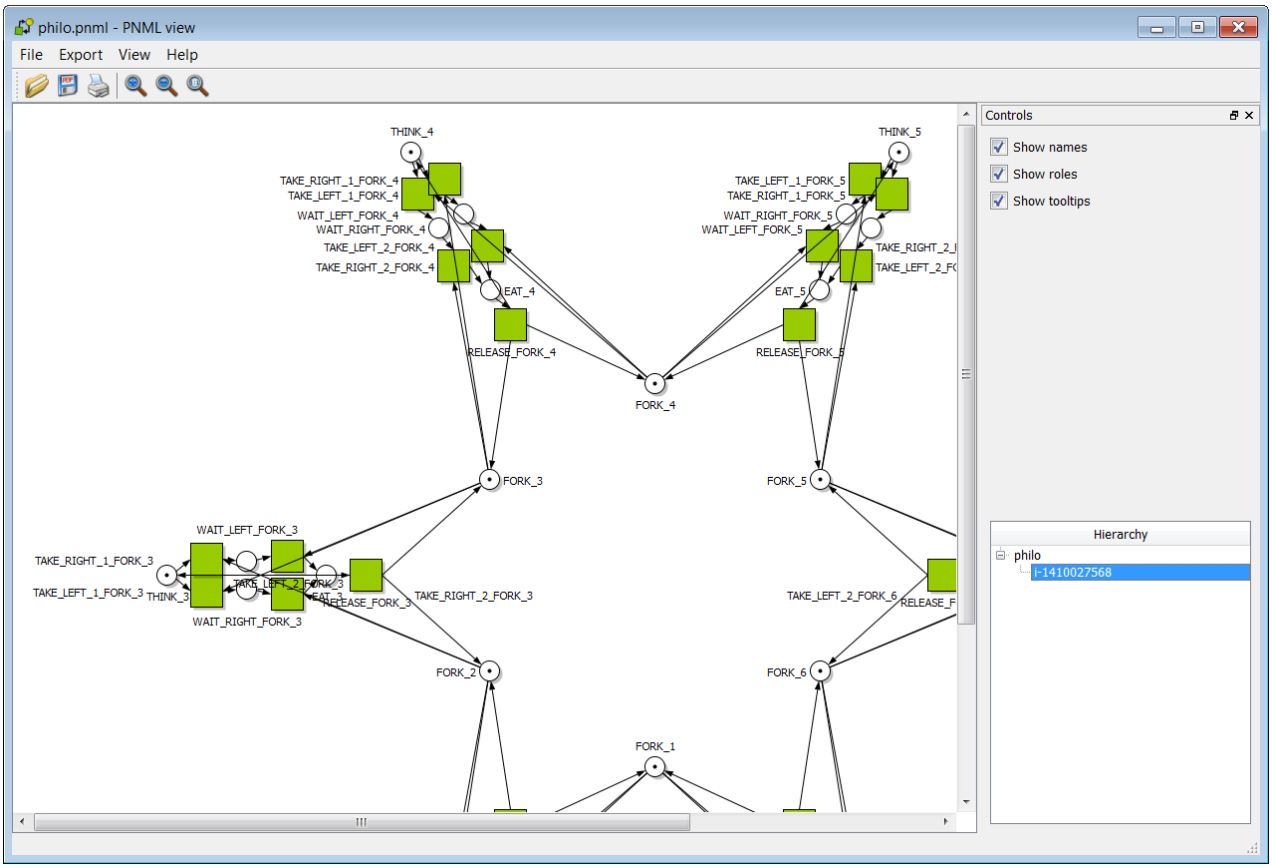


Obr. 5.2 - Študentský projekt z McGill univerzity v Kanade

Tento nástroj neponúka možnosť simulácie, no za zlepšenie oproti predošlému je možné považovať napríklad prívetivejšiu grafickú reprezentáciu siete. Farebné odlíšenie vstupných a výstupných hrán zvyšuje prehľadnosť a preto bolo použité aj v našej aplikácií.

Ďalej je zaujímavé využitie usporiadania objektov siete do mriežky. Toto bolo čiastočne implementované aj v našom nástroji, ktorý ponúka možnosť zarovnania, kedy uzly grafu, ktorých rozdiel x-ových alebo y-ových súradníc je menší ako určitá pevne stanovená hodnota, sú zarovnané podľa patričnej osi.

Z nástrojov vyššej úrovne bol testovaný napríklad nástroj **PNML view**, ktorého ukážka je na obrázku 5.3. Ide výlučne o prehliadač PNML sietí, ktorý podporuje aj niektoré grafické modifikácie siete formulované jazykom PNML, viď 3.2.1.



Obr. 5.3 - Ukážka prostredie aplikácie PNML view

Ako je viditeľné z menu lišty a pracovnej lišty aplikácie, PNML view neumožňuje akékoľvek modifikácie modelu, no umožňuje jeho tlač a export napríklad do formátu PDF, čo bol záujem implementovať aj v našej aplikácií, no pre krátkosť času sa na to nedostalo. Praktickou vymoženosťou je aj približovanie a odďaľovanie modelu (zoom in a zoom out), ktoré bolo implementované aj v našom nástroji.

Čo sa týka grafickej stránky jednak vyobrazenej siete a aj užívateľského rozhrania, tak je tento nástroj na neporovnateľne vyššej úrovni ako predošlé a v našej aplikácií je rovnako ako tu využitá reprezentácia ikon pomocou obrázkov, namiesto textu, keďže pôsobí na užívateľa prívetivejšie. Rovnako sú pri vyobrazení grafu napríklad používane orientované hrany, no farebné odlíšenie prechodov, či iné farebné odlíšenia, sa nezdali nutné.

Nevýhodou nástroja okrem obmedzenia jeho použitia je aj fakt, že neumožňuje prehliadanie viacerých sietí zároveň, čo naša aplikácia podporuje, organizovaním aktuálne otvorených modelov do záložiek.

Okrem spomínaných existuje samozrejme rada iných, aj spoplatnených aplikácií, či frameworkov, ktoré umožňujú tvorbu Petriho sietí no neboli v porovnaní spomenuté.

6 Záver

Cieľom práce bolo vytvoriť nástroj pre prácu s OOPN, s možnosťou ich načítania a ukladania vo formátoch PNtalk a PNML. Nástroj mal ďalej zabezpečiť prepojenie so simulátorom OOPN a umožňovať vzdialené riadenie simulácie. Základné požiadavky môžeme považovať za splnené, no treba povedať, že sa v aplikácií naskytuje ešte množstvo možností pre vylepšenia, jednak čo sa týka modifikácií zobrazenia grafu, riadenia simulácie, či prispôsobeniu sietí pre tlač.

Práve fakt, že nie je implementovaná možnosť tlače a možnosť pokročilo upraviť grafickú podobu siete možno považovať za najväčšie nedostatky aplikácie, ktoré by však nemal byť problém v budúcnosti vyriešiť.

Ako celok je však aplikácia plne funkčná a stanovené ciele môžeme z väčšej miery považovať za splnené.

7 Literatúra

[1] Janoušek, V.: *Modelování objektů Petriho sítěmi*, Brno, CZ, UIVT FEI VUT, 1998.

[2] PNML.org: *PNML grammar, version 2009* [online]. 2009 [cit. 2013-03-07]. Dostupné na URL: <http://www.pnml.org/version-2009/version-2009.php>

[3] ISO/IEC: *ISO/IEC 15909-2 (WD 1.1.5) Software and Systems Engineering - High Level Petri Nets - Part 2: Transfer Format*, 2007.

[4] Peringer, P.; Hrubý M.: *Prednášky z predmetu Modelování a simulace*, Brno, CZ, FIT VUT, 2012

[5] Di Battista, G.; Eades, P.; Tamassia, R.; Tollis, I.O.: *Graph Drawing: Algorithms for the Visualization of Graphs*, Prentice-Hall Inc., 1999, ISBN: 0-13-301615-3

[6] Fruchterman,T.M.J; Reingold, E.M: *Graph Drawing by Force-directed Placement*, Software-Practice and Experience vol. 21, November 1991.

Zoznam príloh

Príloha 1: CD

Priložené CD obsahuje nasledujúce materiály:

- Zdrojové kódy aplikácie - adresár *./source_codes*
- Vygenerovaná javadoc dokumentácia - adresár *./javadoc*
- Pharo obraz obsahujúci implementovaný simulátor - adresár *./simulator*
- Manuál, obsahujúci pokyny pre spustenie simulátora a pre obsluhu editora - súbor *manual.pdf*
- Technická správa v elektronickej podobe - súbor *sprava.pdf*
- Plagát - súbor *poster.pdf*

Príloha 2: Plagát

Vytlačený vo formáte A2 odovzdaný spolu s tlačenou verziou BP. Na nasledujúcej strane je dostupný zmenšený náhľad a na CD priloženom k tlačenej verzii je plagát v elektronickej podobe - súbor *poster.pdf*.

Príloha 2: náhľad plagátu