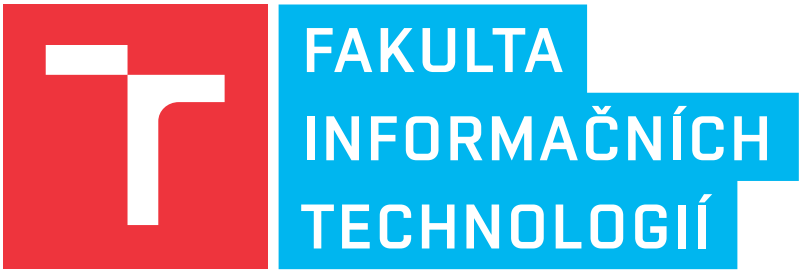
Vysoké učení technické v Brně
Fakulta informačních technologií



Simulátor P/T Petriho sítí

Dokumentácia k projektu do predmetu IMS

Jakub Kelečéni – xkelec00@stud.fit.vutbr.cz
Václav Bayer – xbayer05@stud.fit.vutbr.cz

8. decembra 2015
Brno

Obsah

1 Úvod

Obsahom práce je popis návrhu, riešenia a implementácie simulátora P/T Petriho sietí[5][IMS 123] v jeho stochastickej variante. Úlohou simulátora je vykonanie simulácie[5][IMS 8] abstraktného modelu[5][IMS 7], ktorý je zadaný na vstupe. Výstupom simulátora sú štatistiky o stave systému v určitých okamihoch modelového času[5][IMS 21]. Výstup obsahuje tiež celkové štatistiky, ktoré vyjadrujú extrémne a priemerné hodnoty.

1.1 Autori

Autormi tejo práce sú Jakub Kelečéni a Václav Bayer. K vytvoreniu práce boli potrebné teoretické znalosti nadobudnuté ma prednáškach[5] v predmete Modelování a simulace(ďalej IMS), ktoré viedol Dr. Ing Petr Peringer. Rovnako aj teoretické a praktické znalosti získané z demonštračných cvičení pod vedením Ing. Martina Hrubého, Ph.D.

Hlavným zdrojom informácií bol kurz IMS. Medzi ďalšie dôležité zdroje patrí referenčná príručka jazyka C++[2].

1.2 Overovanie funkčnosti

Testovanie simulátoru prebiehalo na školskom serveri merlin.fit.vutbr.cz (CentOS 6.7 - x64 Linux). Výstup bol porovnaný výsledkami získanými z knižnice SIMLIB[4], ktoré boli dostupné v prezentácií[3] ku druhému demonštračnému cvičeniu. Viac informácií v kapitole 5.

2 Rozbor témy a použitých metód/technológií

Cieľom je implementovať simulátor P/T Petriho sietí, ktorý je rozšírený o možné obmedzenie kapacity miest a hrán, pravdepodobnosť prechodov, časovanie prechodov a priority prechodov. Jedná sa o stochastickú Petriho sieť[5][IMS 135]. Simulátor bude spracovávať vstupný model Petriho siete, ktorý je potrebné do programu vložiť. Je potrebné si definovať z akých komponentov bude model pozostávať a za akých podmienok bude možná zmena stavu systému. Je potrebné určiť, že čo znamená zmena stavu systému a aké deje pri tom prebiehajú.

2.1 Popis použitých postupov

Vychádzalo sa zo všeobecnej definície 2.1 Petriho siete. Model Petriho siete vložený do simulátora bude tvoriť jeho základ.

Definícia 2.1 *P/T Petriho sieť[5][IMS 123] je:*

$$\sum = (P, T, F, W, C, M_0)$$

kde:

- P je množina miest(stavy)
- T je množina prechodov, $P \cap T = \emptyset$
- incidenčná relácia $F \subseteq (P \times T) \cup (T \times P)$
- váhová funkcia $W : F \to \{1, 2, \ldots\}$
- kapacity miest $C : P \to N$
- počiatočné značenie $M_0\ P \to N$

Všeobecná definícia Petriho siete nepopisuje jej stochastický variant, ktorý je rozšírený o prechody časované, prioritné a pravdepodobnostné. V implementácií je použitý stochastický variant, ktorý bol upravený vzhľadom k implementácii.

Vstupný model bude tvorený prvkami *prechod, miesto, hrana.* Každý z týchto prvkov má svoje vlastnosti. Nasleduje bližší popis uvedených prvkov:

- Prechod: Vlastnosťou prechodu je, že mení stav systému. To kedy je možné túto zmenu vykonať popisuje nasledujúca definícia:

Definícia 2.2 *Prechody je možné vykonať práve vtedy, keď je vo všetkých vstupných miestach dostatočný počet značiek, a zároveň je dostatočná kapacita v miestach výstupných[5][IMS 125].*

Zmena stavu systému je vykonaná takým spôsobom, že presúva značky zo vstupného miesta do miesta výstupného[1].

- Miesto: Obsahuje stavovú informáciu o počte značiek, ktoré sa v ňom nachádzajú. Môže mať obmedzenú kapacitu[1].
- Hrana: Spája prechod s miestom. Má svoju orientáciu a môže mať obmedzenú kapacitu. Nemôže spojiť dva rovnaké prvky (prechod s prechodom, miesto s miestom)[1].

2.2 Použité technológie

Zadanie povoľovalo použitie jazykov C a C++. Keďže jednotlivé prvky Petriho siete môžu byť chápané ako objekty, tak bolo vhodné použitie jazyka C++, ktorý podporuje objektový model. Vďaka tomu bolo možné rozdelenie implementácie do jednotlivých tried a príslušných metód/funkcií.

3 Koncepcia

Vzhľadom k zvolenému programovaciemu jazyku sme mohli jednotlivé časti rozdeliť na menšie celky. Každý z týchto celkov má jednoznačnú funkcionalitu a je uložený v dvoch súboroch. Jedným zo súborov je súbor s príponou *.h*, ktorý predstavuje hlavičkový súbor[2], v ktorom sú vytvorené triedy, deklarované funkcie a členské premenné. Druhý súbor má príponu *.cpp* a predstavuje zdrojový text napísaný v jazyku C++. V tomto súbore sú implementované funkcie deklarované v hlavičkovom súbore.

Ku spusteniu programu musí užívateľ poznať abstraktný model zakreslený v Petriho sieti, ktorý zadá do programu pomocou pravidiel popísaných v bode 4.2. Po tom čo je model zadaný, vloží sa počiatočná udalosť a spustí sa štart simulácie. Program v určitých časových intervaloch vypisuje priebežné informácie o štatistikách na štandardný výstup. Časové intervaly sú vždy vypočítané na základe doby behu programu. Prvý výstup prebieha vždy po 1/10 celkového času simulácie a ostatné vždy po približne 1/5.

Základom simulátora je vstupný model. Ten je tvorený prechodmi, miestami a hranami. Samotné riadenie simulácie je zabezpečené zvláštnym algoritmom, ktorý bude popísaný v bode 3.5. Koncepcia simulátoru je taká, že po vytvorení modelu a spustení simulácie je simulácia riadená určitým algoritmom, ktorý riadi zasielanie správ jednotlivým objektom. V nasledujúcej časti tejto kapitoly budú podrobnejšie popísané jednotlivé prvky(objekty) simulátora.

3.1 Miesto

Miesto je chápané ako objekt. Nesie informácie o svojom názve, kapacite a aktuálnom počte značiek. Značka predstavuje proces/transakciu, ktorá čaká na spracovanie. V systémoch hromadnej obsluhy[5][IMS 136] môže miesto predstavovať frontu. Tento objekt tiež nesie štatistické dáta o maximálnom a minimálnom počte a tiež informáciu pomocou ktorej je možné vypočítať priemerný počet značiek v mieste.

3.2 Hrana

Úlohou hrany je prepojenie miesta a prechodu. Jej vlastnosťou je kapacita, ktorá má implicitne hodnotu 1. Kapacita hrany udáva, že koľko značiek sa bude z miesta prenášať. V programe rozlišujeme dva druhy hrán – vstupné, výstupné. To že je hrana vstupná znamená, že do prechodu vstupuje, naopak výstupná z prechodu vystupuje.

3.3 Prechod

Prechod je prvkom, ktorý mení stav systému. Zmena stavu je realizovaná presunom značiek z miesta vstupného, do miesta výstupného. Prechod je vykonateľný za určitých podmienok, ktoré musia byť splnené. Pre rôzne typy prechodov sa tieto podmienky dopĺňajú inými. Stochastické Petriho siete využívajú dva základné typy prechodov – okamžité a časované. Časované prechody je možné rozdeliť do 2 skupín:

- s generovaným časom – čas čakania značky vo vstupnom mieste prechodu bude určený vygenerovaním pravdepodobnosti určitého rozdelenia s vopred známym stredom(frekvencia príchodu transakcie do systému)
- s konštantným časom – čas čakania je vždy konštantný a vopred známy

Časované prechody sú vykonateľné vtedy, ak značka vo vstupnom mieste čakala stanovenú dobu. Okamžité prechody sa ďalej delia na prechody:

- s implicitnou prioritou – ich priorita je vždy rovná nule, aby bol prechod vykonateľný nemôže byť súčasne vykonateľný prechod s vyššou prioritou, ktorý je spojený s rovnakým vstupným miestom
- prioritné – priorita je 1 a viac, vždy sa najprv vykonávajú prechody s vyššou prioritou a postupuje sa smerom k priorite nižšej
- pravdepodobnostné – majú rovnakú prioritu ako okamžité, sú vykonateľné s určitou pravdepodobnosťou

Prechod musí držať informáciu o každej hrane, ktorá do neho vstupuje a z neho vystupuje. Toto je možné implementovať dynamickou dátovou štruktúrou typu vektor. Ďalej je potrebné jednoznačne určiť akého je prechod typu. Tento objekt musí byť schopný definovať, že či je v aktuálnom stave/čase vykonateľný alebo nie a taktiež riadiť presun značiek zo vstupného miesta do výstupného.

3.4 Simulačný model

Simulačný model rovnako ako abstraktný model vychádza z nasledujúcich objektov: prechod, miesto, hrana. Ďalej obsahuje názov modelu a údaj o hodnote priority prechodu z najvyššou prioritou. Model prestavuje objekt, ktorý je naplnený informáciami o Petrho sieti. Obsahuje teda množinu všetkých miest a prechodov. Tieto sú uložené vo vektoroch[2]. Tým je zabezpečené to, že v každom okamihu je možné pristúpiť ku ktorémukoľvek prvku modelu a pracovať s ním – získavať informácie o stave, meniť stav prvku, atď. Prístup je ilustrovaný nasledujúcim algoritmom:

Data: prvok
for *i ← 0 to velkost vektoru* **do**
| prvok[i]→vykonaj operaciu
end

Ďalšou významnou časťou modelu je schopnosť aktualizovať stav systému. V praxi to znamená že sa vykonajú všetky okamžité prechody, ktoré spĺňajú podmienku pre vykonanie. V programe bolo potrebné zohľadniť prioritné prechody, prechody s pravdepodobnosťou a časované prechody. Časované prechody sú špeciálnym prípadom a v programe sú definované ako udalosť. Ich vykonávanie je riešené cez tzv. kalendár udalostí[5][IMS 173], ktorý je popísané v sekcii 3.5. Algoritmus aktualizácie stavu systému sa dá popísať nasledovne:

while *je vykonatelny necasovany prechod* **do**
| vykonaj prioritne
| vykonaj okamzite
| vykonaj pravdepodobnostné
end

Pre prioritné prechody platí, že ak sú vykonateľné oba prechody smerujúce z jedného miesta a jeden z prechodov má vyššiu prioritu tak sa vykoná ten, ktorého priorita je vyššia. Problémom je, že po každom vykonaní prechodu s nižšou prioritou sa systém môže dostať do stavu, kedy je vykonateľný nejaký prechod s vyššou prioritou. Riešením tohoto problému je riadenie spracovania prioritných prechodov takým spôsobom, že po každom vykonaní prechodu sa prechádzajú všetky s vyššou prioritou. Ilustruje to uvedený algoritmus:

```
Data: maximalna priorita
while maximalna priorita >0 do
    if su vykonatelne prioritne prechody then
        spracuj prioritne(maximalna priorita)
    end
    else
        maximalna priorita=maximalna priorita-1
    end
end
```

Keďže okamžité prechody majú implicitnú prioritu nulovú, tak pre ne platí podobné pravidlo ako pre prioritné. To znamená, že po vykonaní prechodu sa vždy vykonajú všetky prioritné a potom sa pokračuje ďalším okamžitým. Takto je zabezpečené to, že okamžitý prechod sa nikdy nevykoná pred prechodom prioritným.

Pri prechodoch s pravdepodobnosťou je tento prípad ošetrený taktiež rovnako. Avšak bolo potrebné navrhnúť to, aby sa medzi prechodmi s rovnakým identifikátorom prioritného prechodu rozhodlo o tom, že ktorý sa vykoná a to na základe pravdepodobnosti. Toto je docielené vygenerovaním náhodného čísla z intervalu 1 až 100 pomocou generátora pseudonáhodných čísiel rovnomerného rozdelenia. Na základe zadanej pravdepodobnosti jednotlivého prechodu je stanovený interval pre každý jeden prechod a v závislosti od toho, že do ktorého intervalu číslo spadá, sa vykoná príslušný prechod.

3.5 Riadenie simulácie

Keďže simulátor dovoľuje pracovať s časovanými prechodmi, tak je potrebné nejakým spôsobom určiť, kedy a čo sa má vykonať. Čas vykonania a miesto vykonania bude popisovať udalosť. Udalosť teda bude niesť informáciu o prechode, ktorý sa má vykonať a o čase v ktorom sa má vykonať. Jedna udalosť je chápaná ako blok neprerušiteľných inštrukcií. Aby sa udalosti do kalendára pridávali musíme zabezpečiť plánovanie udalostí. Plánovať sa budú iba časované prechody, okamžité sa budú vykonávať počas každého plánovaného. Spracovanie udalostí bude realizovať algoritmus kalendára:

```
while kalendár je neprázdny do
    if cas vykonania udalosti >cas ukončenia simulácie then
        koniec simulacie
    end
    vyber polozku z kalendara
    nastav cas simulacie na cas vykonania udalosti
    vykonaj naplanovanu udalost
    vykonaj vsetky okamzite prechody
    naplanuj nove udalosti
end
```

Kalendár musí všetky udalosti radiť vzostupne podľa času vykonania. Začiatok simulácie tvorí vstup do procedúry kalendára. Pred začiatkom simulácie musí algoritmus poznať čas ukončenia simulácie a model, ktorý bude simulovať.

4 Architektúra simulátoru

V tejto kapitole je popísaná implementácia simulátoru. Keďže simulátor využíva programovací jazyk C++, ktorý je objektovo orientovaný, v popise sa budeme zameriavať na popis objektovo orientovaného modelu a teda implementáciu jednotlivých tried.

4.1 Implementácia

Nasleduje zoznam jednotlivých tried s popisom implementácie:

- **Miesto:**
  Trieda miesto popisuje miesto v Petriho sieti. Deklaruje členské premenné *kapacita, pocetZnaciek, cakaju,*

id. Premenná pocetZnaciek reprezentuje aktuálny počet značiek v mieste. Implicitná hodnota tejto premennej je 0. Premenná kapacita označuje maximálnu kapacitu značiek v mieste, jej implicitná hodnota je 0 a znamená, že aký maximálny počet značiek môže byť v mieste. Význam premennej cakaju je taký, že v momente keď sa naplánuje časovaný prechod, odčíta sa určitý počet značiek z premennej pocetZnaciek a pričíta sa k premennej cakaju. Pri miestach, ktoré sú spojené s okamžitými prechodmi, premenná cakaj nezohráva žiadnu úlohu. Premenná s názvom id označuje názov miesta, jednoznačný identifikátor, podľa ktorého bude miestu priradená hrana. Ďalej sú deklarované verejné premenné, ktoré uchovávajú štatistické dáta.

V triede sú definované funkcie pre prácu s členskými premennými, pomocou ktorých je možné získať hodnoty premenných a tiež nastaviť hodnoty niektorých premenných.

- **Prechod:**
V tejto triede je definovaných niekoľko premenných. Medzi najhlavnejšie patria premenné s identifikátormi *vstupneHrany, vystupneHrany, id, typ*. Premenné vstupneHrany a vystupneHrany umožňujú prístup ku všetkým hranám prechodu. Pomocou nich je možné pristupovať ku miestam a vykonávať operáciu presunu značiek. Premenná je typu *vector* a uchováva zoznam ukazateľov na typ Hrana. Typom prechodu je celočíselná hodnota, ktorá jednoznačne špecifikuje typ prechodu – okamžitý (s prioritou, s pravdepodobnosťou, s implicitnou prioritou), časovaný(s generovaným/konštantným časom). Každý typ prechodu potom využíva premennú ktorá popisuje jeho špecifickú vlastnosť(priorita, čas čakania, pravdepodobnosť,. . .). V triede sú tiež premenné ktoré sú využívané pri výpočte štatistík.

  Ako už bolo uvedené vyššie, každý prechod musí mať pridelený názov – id a typ. Samotné vytvorenie inštancie tejto triedy je možné niekoľkými spôsobmi, v závislosti od typu prechodu. V triede sú preto implementované štyri rôzne konštruktori, ktoré sa od seba odlišujú počtom parametrov a typom parametrov. Napríklad pri vytvorení prioritného prechodu je potrebné uviesť údaj o priorite, pri časovanom je to údaj o čase, atď.

  Medzi podstatné funkcie definované v tejto triede možno zaradiť:

  – funkcie pre pridávanie vstupných a výstupných hrán
  – funkciu pre spracovanie značiek – vykonanie prechodu
  – funkciu pre zistenie toho, či je prechod vykonateľný
  – funkcie pre prístup k členským premenným

- **Hrana:**
Úlohou inštancií tejto triedy je prepojenie miesta a prechodu. V triede sa nachádzajú tri členské premenné: *kapacita, miesto, prechod*. Premennou kapacita je určená kapacita hrany. Premenné miesto a prechod sú referenciami na objekty, ktoré sú hranou prepojené.

- **Model:**
Trieda Model je jednou z najdôležitejších a najrozsiahlejších(implementácia) v programe. Obsahuje premenné s názvom *prechody* a *miesta*. Tieto premenné sú typu vektor, v ktorom sú uložené ukazatele na všetky miesta a prechody zadaného modelu. Vďaka tomu je možné pristúpiť ku každému prvku Petriho siete, čo je využité pre riadenie operácií s týmito prvkami. Medzi členské premenné patrí tiež premenná, uchovávajúca informáciu o najvyššej priorite okamžitého prioritného prechodu v modele, názov modelu a premenná, pomocou ktorej je riadený výpis priebežnej štatistiky.

  V triede sa nachádzajú rôzne metódy pre:

  – vytvorenie modelu – pridanie objektov(miesta, prechodu alebo hrany)
  – operácie s modelom – aktualizáciu stavu systému a spracovanie prechodov určitého typu
  – generovanie náhodných čísiel – exponenciálneho a rovnomerného rozdelenia
  – výpis informácií – štatistiky a model

  Funkcie pre pridanie objektov typu prechod a typu miesto do modelu, sú implementované ako pridanie ukazateľa do príslušného vektoru. Parametrami týchto funkcií sú ukazatele na objekt. Pridávanie hrán vyžaduje zadanie názvu prechodu a názvu miesta, ktoré má prepojiť. Je potrebné rozlišovať to, či hrana do prechodu vstupuje alebo z neho vystupuje. O tom o akú hranu ide rozhodne parameter funkcie *PridajHranu*, ktorý je typu *bool*. V prípade že je hrana vstupná, hodnota premennej bude *true*, v opačnom

prípade *false*. Implicitná hodnota pre túto premennú je nastavená na hodnotu nepravda. Ďalším parametrom, ktorý je možné špecifikovať pri pridávaní hrany je premenná typu celé číslo. Jej hodnota určuje kapacitu hrany.

Riadenie zmeny stavu systému zabezpečuje funkcia s názvom *AktualizujStavSystému*. Algoritmus funkcie je popísaný v bode 3.4. Úlohou funkcie je kontrolovať či je v systéme vykonateľný prechod. Po vykonaní funkcie sa v systéme nenachádza žiadny vykonateľný nečasovaný prechod. Obsluha vykonávania prechodov je realizovaná cez funkcie *SpracujOkamzite, SpracujPravdepod, SpracujPrioritne*. Každá z týchto funkcií obsluhuje jeden typ prechodu, prípadne volá funkciu pre vykonanie prioritných prechodov a to po každom vykonaní prechodu. V algoritme riadenia zmien stavu systému sa nenachádza funkcia, ktorá by vykonávala časované prechody. Postup vykonávania časovaných prechodov bude bližšie špecifikovaný v popise triedy *Kalendar*.

Funkcie, ktoré riešia výpis informácií sú: *VypisStatistiky, VypisStav, VypisModel*. Nachádza sa tu funkcia pre výpis modelu. V demonštračných príkladoch použitia simulátoru táto funkcia nie je volaná. Slúži pre vykreslenie modelu, konkrétne každého prechodu, hrany a miesta. Vo výpise je možné identifikovať prechod, miesto a hranu s príslušnou kapacitou a orientáciou, ktorá tieto dva objekty spája. Táto funkcia bola využitá pre overenie správnosti zadaného modelu.

Výstupom simulátoru sú štatistiky. Tie sú vypisované vo forme celkových štatistík po prebehnutí simulácie a vo forme priebežných štatistík, z ktorých je viditeľná evolúcia modelu. Priebežné štatistiky sú vždy vypisované po určitom čase. Bola zvolená taká frekvencia výpisu, aby bolo zjavné, ako sa menil stav systému. Avšak frekvencia nemusí byť dostatočná pre podrobnú analýzu modelu v určitých časových okamihoch. Ideálne by bolo vypisovať detailné informácie po každej zmene stavu systému, čo by ale pri rozsiahlejších modeloch viedlo k obrovskému množstvu dát, ktoré by eliminovalo prehľadnosť. Priebežné štatistiky sa teda vypisujú celkom päťkrát. Prvý výpis sa vo frekvencii odlišuje a vypíše sa už po prvej desatine celkovej doby behu simulácie. Nasledujúce výpisy sú realizované vždy v po pätinách celkového času. Informácie nemusia byť vypísané vždy v presnom časovom okamihu získaným výpočtom z celej simulácie, pretože v systéme bývajú väčšinou prvky, ktorých chovanie je ovplyvnené istým náhodným dejom a vzhľadom ku koncepcii simulátora je plánovanie v určitom okamihu simulačného času nemožné. Priebežný výpis je implementovaný vo funkcii *VypisPriebeznyStav*

- **Kalendar:**
  Triedu Kalendar si môžeme predstaviť ako kalendár, v ktorom sú naplánované udalosti. Pozostáva z členských premenných, ktoré uchovávajú informácie o čase začiatku simulácie(vždy 0), aktuálnom simulačnom čase a tiež o čase ukončenia. Pomocou týchto časov je riadený celý algoritmus(viď bod 3.5) a tiež udalosti, ktoré v ňom nastávajú. Ďalšími premennými sú ukazateľ na model a fronta. Premenná popisujúca model je vyžadovaná pri vytváraní novej inštancie, to isté platí aj pre premennú *endTime*. Keďže algoritmus kalendára využíva dátovú štruktúru zoradenú podľa času, vzostupne, bola použitá fronta typu *priority_ queue*[2], ktorá zoraďuje vkladané prvky podľa nejakého kritéria. Tým je zabezpečené, že pri každej operácií *front* bude vybraný prvok vždy s najnižším aktivačným časom. Fronta v sebe uchováva prvky typu ukazateľ na *Udalost*. Typ Udalost definuje jeho trieda. V tejto triede sú premenné, ktoré uchovávajú čas vykonania udalosti a prechod, ktorý sa má vykonať pri spracovaní udalosti. Trieda Udalost taktiež obsahuje funkcie pre prístup k premenným. Samotné spracovanie udalostí prebieha v jadre kalendára, jeho algoritme, ktorý je implementovaný vo funkcií *StartSimulacie*.

  V triede Kalendar sú definované funkcie pre prácu s frontou(*VlozUdalost, VymazUdalost*), riadenie simulácie *StartSimulacie* a tiež funkcie pre prístup k premenným. Najdôležitejšou funkciou je funkcia, ktorá zabezpečuje riadenie simulácie. Funkcia vychádza zo základného algoritmu kalendára, na ktorý bolo odkazované v úvodnej časti tejto kapitoly. Základom funkcie je cyklus, ktorý beží kým fronta nie je prázdna, alebo kým čas simulácie neprekročí zadanú hranicu. V každom cykle sa vyberá z fronty jedna udalosť, ktorá sa spracuje. Následne sa pomocou objektu modelu volajú funkcie pre aktualizovanie stavu systému, aktualizáciu štatistík, plánovanie ďalších udalosti a výpis priebežnej štatistiky.

4.2 Vloženie abstraktného modelu

V tejto kapitole bude popísaný postup mapovania abstraktného modelu Petriho siete do simulátora. Pre mapovanie jednotlivých prvkov je potrebné najprv vytvoriť inštanciu triedy *Model*. Po jej vytvorení je možné volanie funkcií, pomocou ktorých bude model vytvorený. Postup vytvárania jednotlivých komponentov musí prebiehať presne podľa nižšie špecifikovaných pravidiel(podkapitoly *Vkladanie prechodov, Vkladanie miest, vkladanie*

Hrán). V opačnom prípade by to mohlo viesť k nedefinovanému chovaniu, ktoré by mohlo znamenať predčasné ukončenie programu alebo jeho zacyklenie. Vytvorenie inštancie modelu môže byť naprogramované takto:

- *Model *ukazatelNaModel = new Model("NazovModelu", 3);*
  //číslo 3 označuje najvyššiu prioritu prechodu v modele

Pridávanie jednotlivých prvkov môže byť rôzne v závislosti od typu prvku a jeho vlastností. V nasledujúcich troch podkapitolách budú uvedené príklady vytvárania nových inštancií tried a vkladanie prvkov do modelu.

4.3 Vkladanie prechodov

Prechod v Petriho sieti môže byť dvoch základných typov(časovaný, okamžitý). Avšak v simulátore rozlišujeme medzi piatimi druhmi prechodov a síce prechod okamžitý(s implicitnou prioritou), okamžitý s prioritou(priorita väčšia ako 0), okamžitý s pravdepodobnosťou, časovaný s generovaným časom a časovaný prechod s konštantným časom. Každý typ prechodu má vlastný konštruktor s výnimkou časovaných. Pre určenie typu prechodu je potrebné použiť definované identifikátory: *OKAMZITY, OKAM_PRIO, PRAVDEPO, CAS_KONS, CAS_GENE*. Popis vytvorenia inštancie každého typu prechodu je nasledovný:

- okamžitý – *new Prechod("nazov", OKAMZITY);*
- okamžitý s prioritou – *new Prechod("nazov", OKAM_PRIO, (int)1);*
  //číslo 1 predstavuje hodnotu priority
- okamžitý s pravdepodobnosťou – *new Prechod("nazov", PRAVDEPO, (int)10, 1);*
  //predposledný parameter vyjadruje pravdepodobnosť v %, posledný parameter identifikuje skupinu pravdepodobnostných prechodov
- s konštantným časom – *new Prechod("nazov", CAS_KONS, 4.0));*
  //pri časovaných prechodoch je potrebné uvádzať číslo vždy v desatinnom tvare
- s generovaným časom – *new Prechod("nazov", CAS_GENE, 10.0, true);*
  //parameter true označuje tzv. vstupný prechod, znamená že modelujeme príchod transakcie

Samotné vloženie prechodu potom prebieha zavolaním funkcie pre pridanie prechodu a môže vyzerať nasledovne:

- *ukazatelNaModel→PridajPrechod(new Prechod("P3", OKAMZITY));*

4.4 Vkladanie miest

Pre vytvorenie miesta je potrebné zadať minimálne jeho názov. Konštruktor pre vytvorenie miesta zahŕňa parametre pre špecifikáciu jeho kapacity a počtu značiek. Keďže najčastejšie sa používajú miesta s nekonečnou kapacitou a nulovým počiatočným počtom značiek, sú tieto parametre inicializované implicitne na nulovú hodnotu. Nulová kapacita je v programe chápaná ako neobmedzená. Samotné vytváranie inštancie môžeme vykonať tromi spôsobmi:

- *new Miesto("nazov");*
- *new Miesto("nazov", 20);* //vytvorí nove miesto s kapacitou 20
- *new Miesto("nazov", 0, 5);* //vytvorí nove miesto v ktorom bude 5 značiek

Pre pridanie miesta je potrebné zavolať funkciu pre pridanie miesta:

- *m→PridajMiesto(new Miesto("nazovMiesta"));*

4.5 Vkladanie hrán

Správne pridanie hrany vyžaduje zadanie názvu miesta a prechodu. Treba tiež rozlíšiť, o aký druh hrany sa jedná. Toto je implementované premennou boolovského typu, ktorej logická hodnota 1 znamená, že vkladáme vstupnú hranu:

- *m→PridajHranu("miestoX", "prechodY", true);*
  //pridá hranu smerujúci z miestaX do miesta prechodY, pričom do prechodu vstupuje
- *m→PridajHranu("Kotvy", "P10");*
  //pridá hranu vystupujúcu z prechodu P10

4.6 Preklad aplikácie

Archív zo súbormi obsahuje zdrojové súbory simulátora a síce *hrana.cpp/.h, miesto.cpp/.h, prechod.cpp/.h, model.cpp/.h, kalendar.cpp/.h, headers.h*, a zdrojové súbory demonštračných príkladov, pričom pre každý príklad je vytvorený samostatný súbor, ktorý obsahuje hlavný program aplikácie. Obsahom príkladov je vytvorenie modelu, namapovanie abstraktného modelu do programu a zahájenie simulácie. Dôležitým súborom, ktorý je priložený je súbor *Makefile*. Pomocou tohoto súboru je možné preložiť a spustiť simulátor. Samotný preklad sa vykonáva príkazom *make*. Prekladom sú vygenerované tri binárne súbory, každý z nich demonštruje jeden príklad. Spustiteľné súbory majú názov *simulator_konkretnyPriklad*. Spustenie všetkých príkladov je realizované prílazom *make run*. Po prebehnutí simulácie sa z každého ukážkového programu vygeneruje súbor *vystup_konkretnyPriklad.txt*, ktorého obsahom je výstup simulátoru – štatistiky. Príkazom *make clean* je možné vymazať všetky súbory vytvorené týmto makefilom. Obsahom archívu s programom je tiež súbor s dokumentáciou vo formáte *.pdf*.

5 Experimenty

Hlavným cieľom experimentov bolo zistiť správnosť fungovania simulátora. Pri zisťovaní toho, že do akej miery je simulátor korektný, boli použité výsledky jednotlivých abstraktným modelov prezentované v rámci demonštračného cvičenia v predmete *IMS*. Samotné experimentovanie prebiehalo zmenou jednotlivých parametrov. Detailný popis je v príslušnej kapitole.

5.1 Demonštračné príklady

Pre demonštráciu základnej funkcionality sme zvolili tri príklady, ktoré boli prebrané v rámci demonštračných cvičení. Sú to príklady s názvami *učebňa, vlek, kravín*. Pre prvé dva z uvedených boli dostupné vzorové dáta z knižnice *simlib*. Z tohoto dôvodu boli vybrané práve tieto dva demonštračné príklady. Tretí príklad je uvádzaný len pre doplnenie. V rámci projektu sa neskúma validita zvolených modelov. Ako už bolo popísané v bode 4.6, jeden príklad predstavuje jeden spustiteľný súbor. Po jeho spustení sa vypisujú obsahuje štatistiky na štandardný výstup. Formát výstupu štatistiky je závislý od jej typu:

- Priebežná štatistika je vypisovaná niekoľkokrát, počas behu simulácie a vždy v inom časovom okamihu. Zahŕňa informáciu o čase, v ktorom bola vypísaná, ďalej informáciu o stave miest a prechodov v tomto čase. Dáta o miestach a prechodoch sú formátované do tabuliek, ktoré obsahujú stĺpce *nazov, act, avg* pre miesto. Pre prechody sú to stĺpce *nazov, count, avg*. Obe tabuľky obsahujú informáciu o názve prvku a o priebežnom priemere. Položka act v mieste znamená, že koľko značiek sa v tomto konkrétnom čase nachádza v danom mieste. Informácia count pri prechode vyjadruje celkový počet vykonaní daného prechodu za čas od začiatku simulácie. Výpis zobrazuje iba tie položky, ktorých hodnoty count(prechod) a act(miesto) sú väčšie ako nula. Je to z toho dôvodu, aby neboli uvádzané zbytočné informácie. S určitosťou môžeme povedať, že ak daný prechod nie je zahrnutý vo výpise, tak sa nevykonal ani raz, a že v mieste, ktoré nie je zahrnuté sa nenachádzajú žiadne značky. Položka avg pri prechode zobrazuje priemernú dobu čakania na vykonanie prechodu a vypisuje sa iba pre časované prechody s generovaným časom. Jedna iterácia výpisu priebežnej štatistiky má takúto podobu:

```
Priebezna statistika v case: 201.43

---> MIESTA (pocet znaciek)
-----------------------------
| nazov       act    avg    |
|---------------------------|
| M1          0      0.33   |
| PC          2      3.27   |
| NaLinke     0      6.73   |
| M_END       0      0.18   |
|___________________________|
```

```
---> PRECHODY
_______________________________
| nazov       count  avg     |
|-----------------------------|
| Prichod     18     12.49   |
| Obsluha_p   18     -       |
| Uvolnenie   10     35.59   |
| END         10     -       |
|_____________________________|
---------------------------------
```

- Celková štatistika je narozdiel od priebežnej vypísaná iba jeden raz a to po ukončení simulácie. Jej cieľom je podať informáciu o extrémnych a priemerných hodnotách, ktoré nadobúdali prvky modelu počas simulácie. V úvode celkovej štatistiky sa nachádza tabuľka informujúca o názve modelu a o časovom intervale, v ktorom prebehla simulácia a má takúto podobu:

```
_______________________________
| model: UCEBNA                |
|------------------------------|
| casovy interval: 0 - 1000.00 |
|______________________________|
```

Nasledujúca tabuľka informuje o miestach. Informácie, ktoré boli vypísané pri priebežných štatistikách dopĺňa údajmi o maximálnom a minimálnom počte značiek v mieste. Položka zobrazujúca priemer sa teraz vzťahuje k celej dobe behu simulácie. Tabuľka vyzerá nasledovne:

```
 ------> MIESTA (pocet znaciek)
_____________________________________________
| nazov       max     min     act     avg     |
|---------------------------------------------|
| PC          10      0       0       1.10    |
| Cakajuci    6       0       0       1.03    |
| NaLinke     10      0       0       8.90    |
| M_END       1       0       0       0.19    |
```

...

Ďalšia tabuľka informuje o prechodoch s generovaným časom. Obsahuje položky *nazov, count, max, min, avg*. Informuje o extrémoch časov čakania na vykonanie prechodu, ich priemere a tiež o počte vykonaní. Formát tejto štatistiky je:

```
 ------> PRECHODY - S GENEROVANYM CASOM
_____________________________________________
| nazov       count   max     min     avg     |
|---------------------------------------------|
| Prichod     98      59.48   0.58    13.64   |
| Uvolnenie   65      594.78  7.94    117.61  |
|_____________________________________________|
```

Posledná časť výpisu štatistík je venovaná ostatným typom prechodov. Vypovedá o tom, že koľkokrát bol ktorý prechod vykonaný. Má túto podobu:

```
 ------> PRECHODY - OSTATNE
-----------------------
| nazov       count   |
|---------------------|
| Obsluha_p   37      |
| END         65      |

...
```

Spustenie programu je možné aj bez použitia príkazu make run. Spustiť program je možné cez príkazový riadok. Vďaka tomu je možné zadať voliteľný parameter v podobe celého čísla, ktorý určí dobu behu simulácie(časový interval). Implicitne má každý demonštračný príklad interne nastavenú dobu behu a to nasledovne:

- učebna – 10000 časových jednotiek
- vlek – 1000 časových jednotiek
- kravín – 720 časových jednotiek

Každý údaj predstavuje hodnotu v nejakej časovej jednotke. Je potrebné dodržať rovnaké časové jednotky, aké boli použité pri mapovaní abstraktného modelu.

5.2 Experimenty

Po zadaní modelu sme najprv zisťovali, či sa model vytvoril správne. Využívala sa k tomu funkcia, ktorá vypísala daný model. Po kontrole, či bol model vytvorený správne(korektné prepojenie miest a prechodov hranami so správnou orientáciou) prebiehalo niekoľko experimentov:

1. opakované spustenie konkrétneho príkladu – pri tomto experimente bolo sledované chovanie systému v každom spustení a vzájomne medzi sebou porovnávané
2. spúšťanie príkladov s rôznou dobou behu programu
3. zmena parametrov niektorých prvkov modelu
4. porovnanie výsledkov s výsledkami získanými z knižnice *simlib*

5.3 Záver experimentov

Z jednotlivých experimentov bolo posúdené, či sa jednotlivé prvky správajú podľa očakávaní. Napríklad či sa prechody s pravdepodobnosťou vykonávajú v pomere rozložení pravdepodobností, alebo tiež či sa prechody s vyššou prioritou vykonávajú pred tými s nižšou prioritou atď.

Pri spúšťaní programu s rôznou dobou behu bolo zistené, že čím je kratšia celková doba behu simulácie, tým je počet vykonaní niektorých prechodov menší. Zistili sme že pri časovaných prechodoch s generovaným časom to môže spôsobovať priemerné hodnoty výrazne odlišné od stredu zadaného času. Záverom tohoto experimentu je, že dôveryhodné výsledky je možné získať pri väčších počtoch vzoriek. Vhodné je voliť takú dobu behu programu, aby trvala dlhšie ako tisíc časových jednotiek.

Experimentovaním so zmenou niektorých parametrov sme zisťovali, že či je návrh jednotlivých komponentov simulátora korektný. Týkalo sa to zmien parametrov čas, priorita, pravdepodobnosť v prvku prechod, kapacity v prvku miesto/hrana a počiatočný počet značiek v prvku miesto. Systém sa pri experimentovaní choval podľa očakávaní.

Porovnanie výsledkov s referenčnými výsledkami nebolo jednoznačné, keďže testované modeli zavádzajú určité prvky neurčitosti. Po vykonaní niekoľkých spustení s rovnakými parametrami môžeme usúdiť, že výsledky sú podobné.

6 Záver

Výsledný program je simulátor stochastických P/T Petriho sietí. Simulátor po vložení abstraktného modelu v Petriho sieti, ktoré prebieha podľa pravidiel popísaných v kapitole 4 vykoná simuláciu daného modelu a realizuje výpis štatistických informácií. Štatistiky sú vypisované priebežne, aj po skončení simulácie. Zo štatistík je viditeľný vývoj modelu v čase. Súčasťou simulátora sú ukážkové príklady, ktoré je možné spustiť a analyzovať. Vzorovými príkladmi je demonštrovaná funkčnosť všetkých prvkov a ich typov implementovaných v simulátore. Príklady zahŕňajú prvky – okamžité prechody(s nulovou prioritou), okamžité prechody s prioritou, okamžité prechody s pravdepodobnosťou, časované prechody s konštantným a generovaným(exponenciálne rozdelenie) časom.

Použitá literatúra

[1] PETRIHO SÍTE. www.fd.cvut.cz/department/k611/pedagog/K611THO_soubory/3_Petriho_sit.pdf.

[2] Referenčná príručka jazyka C++. http://www.cplusplus.com/reference/, 2015.

[3] Hrubý, M.: IMS Democvičení 2. http://perchta.fit.vutbr.cz/vyuka-ims/uploads/1/diskr2-2011.pdf, 2015.

[4] Peringer, P.: SIMulation LIBrary for C++. http://www.fit.vutbr.cz/ peringer/SIMLIB, 2014.

[5] Peringer, P.: Modelování a simulace - prednášky. www.fit.vutbr.cz/study/courses/IMS/public/prednasky/IMS.pdf, 2015.