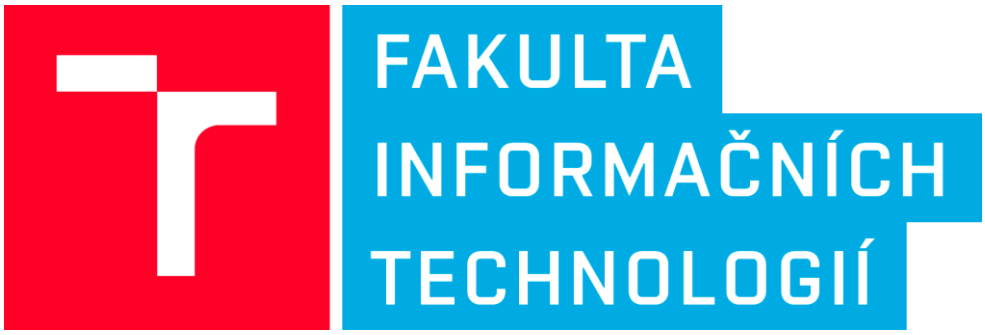
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií



Dokumentace k projektu do předmětu IMS

Simulátor P/T Petriho sítí

8. Prosince 2015

Autoři: Jakub Stejskal, xstejs24@stud.fit.vutbr.cz
Petr Staněk, xstane34@stud.fit.vutbr.cz
Fakulta informačních technologií
Vysoké Učení Technické v Brně

Obsah

1 Úvod

Tato technická zpráva se zaměřuje na implementaci simulátoru P/T Petriho sítí [IMS, slide 126] pomocí jazyka C++. Výsledný simulátor zpracovává libovolnou P/T Petriho síť na základě vložených míst, přechodů a hran a jejich parametrů. Nejprve ověří, zda je model[IMS, slide 7] validní a až poté provede simulaci situace, kterou popisuje zadaný model.

Jako ukázkový model byla zvolena Petriho síť Herna z demonstračního cvičení [pulsem09.ps, příklad č. 4], která byla upravena, aby lépe splňovala demonstraci rozsah všech možností Petriho sítí. Zajímavou část zvoleného příkladu tvoří obsluha přerušení probíhajících procesů[IMS, slide 121], kde všichni právě hrající hráči končí s hraním a odchází pryč.
Pro správnou implementaci simulátoru bylo nutné dopodrobna pochopit koncept Petriho sítí, k čemu výrazně pomohl Dr. Ing. Petr Peringer v přednáškách a Ing. Martin Hrubý, Ph.D. při demonstračních cvičeních. Dodatečné zdroje informací jsou zmíněny dále v dokumentu podle souvislostí s popisovanou částí simulátoru.

1.1 Autoři

Jakub Stejskal, xstejs24@stud.fit.vutbr.cz
Petr Staněk, xstane34@stud.fit.vutbr.cz
Jak již bylo uvedeno výše, ke správnému pochopení simulátoru přispěl Dr. Ing. Petr Peringer v přednáškách a Ing. Martin Hrubý, Ph.D. při demonstračních cvičeních.

1.2 Zadání projektu

Navrhněte a implementujte simulátor interpretující zadaný vstupní model ve formě P/T Petriho sítě, kde uvažujete možné omezení kapacity míst, kapacity hran, pravděpodobnosti přechodů, priority přechodů, časování přechodů s konstantním a generovaným časem (jedno rozložení dle výběru). Zadání modelu proveďte například programově vytvořením datových struktur (žádné specializované jazyky a překladače). Činnost simulátoru demonstrujte na příkladu, který bude obsahovat všechny zadané prvky sítě

1.3 Validita projektu

Simulátor byl implementován na operačních systémech Ubuntu 14.04 LTS a Linux Mint 17.2. Následně byl simulátor přeložen a otestován na školní serveru Merlin. Přeložení i testování proběhlo úspěšně. Výsledky simulátoru jsou rozebrány v bodě číslo 6.

2 Rozbor tématu a použitých metod/technologií

Vytvořený simulátor simuluje zadanou Petriho síť, která je formálně popsána níže. Formální popis se vztahuje k obecné Petriho síti.

2.1 Definice P/T Petriho sítě

Šestice $\Sigma = (P, T, F, W, C, M_0)$
kde
P je množina všech míst (stavy)
T je množina přechodů, $P \cap T = \emptyset$
Incidenční relace $F \subseteq (P \times T) \cup (T \times P)$
Váhová funkce W: $F \to \{1, 2, \dots\}$
Kapacity míst C: $P \to N$
Počáteční značení $M_0 = P \to N$ (M se nazývá značení Petriho sítě)

2.2 Prvky P/T Petriho sítě

Petriho síť obsahuje tyto prvky: místa (kružnice), přechody (obdélníky), hrany (šipky), značky (tečky nebo číslo.

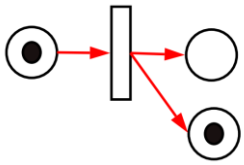
- Místo
  - V grafu značeno jako kružnice. Jedná se o prvek sítě, který uchovává značky během simulace. Každé místo má svoji vstupní a výstupní hranu. Každé místo může obsahovat údaj o maximální kapacitě značek, implicitně je nastavena na 0, což značí nekonečno.
- Přechod
  - V grafu značen jako obdélník. Jedná se o prvek, který v grafu simuluje určitou činnost, případně nějaké rozhodnutí. Díky přechodům je možné simulaci provést.

- Přechodů je hned několik:
  - časovaný (konstantní/s exponenciálním rozložením[IMS, slide 93])
  - prioritní
  - pravděpodobnostní
- Každý přechod může být pouze jednoho typu. Mezi přechody navíc platí jistá omezení, která zakazují kombinaci pravděpodobnostních přechodů s ostatními typy přechodů.

– Hrana
- V grafu značena jako orientovaná úsečka. Jedná se o prvek, který spojuje místo s přechodem a obráceně. Po hraně jsou přenášeny značky z jednoho místa přes přechod do druhého místa.
- Hrana může mít nastavenou kapacitu, která udává, kolik značek je nutno po hraně přenést při jednom vyhodnocení přechodu. Této vlastnosti se také říká váhová funkce[IMS, slide 124].

– Značka
- V grafu značena jako černá tečka, případně jako číslo v místě. Značka v grafu určuje počet procesů, kteří v nějakém místě čekají na možnost projít přechodem dále, dokud neopustí simulaci.

Petriho sítě jsou obvykle zadávány ve formě grafu.



Obrázek 1 – Příklad grafu [IMS, slide 124]

2.3 Popis použitých postupů

Použitý implementační jazyk C++ byl striktně vyžadován zadáním. Nehledě na požadavky zadání je však jazyk C++ nejvhodnější volbou, je totiž objektově orientovaný, a tak umožňuje velmi přehledně a optimálně implementovat požadované funkcionality. Vhodnou volbu implementačního jazyku je možné podložit také faktem, že v jazyce C++ je implementována simulační knihovna SIMLIB.

2.4 Popis původu použitých metod/technologií

Převažujícím zdrojem byly demonstrační cvičení z předmětu IMS, kde byly na úrovni dostačující pro implementaci vysvětleny všechny nutné principy. Tedy veškeré algoritmy byly naše vlastní nebo převzaty a naimplementovány podle studijních opor do předmětu IMS.

3 Koncepce – implementační téma

Celý simulátor je rozdělen na několik algoritmů, které zajišťují správnou práci s objekty Petriho sítě a správný chod celé simulace[IMS, slide 8]. Jelikož jazyk C++ je objektový, byly všechny části Petriho sítě implementovány jako samostatné objekty, což umožnilo jednodušší práci při vývoji simulátoru.

3.1 Rozbor Petriho sítě

Petriho síť je možné rozložit na několik prvků:

– Místo

V grafu značeno jako kružnice. Jedná se o prvek sítě, který uchovává značky během simulace. Každé místo má svoji vstupní a výstupní hranu. Naše implementace implementuje místo jako objekt, který si pamatuje svoje jméno, seznam vstupních a výstupních hran a aktuální počet značek.

– Přechod

V grafu značen jako obdélník. Jedná se o prvek, který v grafu simuluje určitou činnost, případně nějaké rozhodnutí. Díky přechodům je možné simulaci provést. Námi implementovaný přechod o sobě uchovává informace jako jsou jméno, seznam vstupních a výstupních hran, číselnou hodnotu, která značí typ přechodu a pravdivostní hodnotu, zda je možné přechod provést.

- Hrana

  V grafu značena jako orientovaná úsečka. Jedná se o prvek, který spojuje místo s přechodem a obráceně. Po hraně jsou přenášeny značky z jednoho místa přes přechod do druhého místa. Námi implementovaná hrana si o sobě uchovává informace o vstupu a výstupu (místo/přechod) a o maximální kapacitě.
- Značka

  V grafu značena jako černá tečka, případně jako číslo v místě. Značka v grafu určuje počet procesů, kteří v nějakém místě čekají na možnost projít přechodem dále, dokud neopustí simulaci.

Každý výše zmíněný prvek modelu dále obsahuje seznam, ve kterém jsou uloženy všechny prvky stejného typu v daném modelu.

3.2 Rozbor logiky

Středem celé logiky simulátoru je kalendář [IMS, slide 173]. Kalendář je plněn záznamy o událostech [IMS, slide 169], které značí, kdy se má vykonat který časovaný přechod. Jinými slovy je kalendář jednorozměrná kolekce událostí. Událost je jednorázová, nepřerušitelná akce, která skokově mění stav systému.
Jednotlivá událost si uchovává přechod, který se má vykonat, vygenerované zpoždění přechodu, čas, ve kterém má být přechod vykonán a seznam značek, které se během události přemístí.

3.2.1 Hlavní algoritmus simulace

Hlavní algoritmus simulace je jednoduchá smyčka, která prochází kalendář, vybere událost, která má nejnižší aktivační čas a poté ji provede. Před vstupem do cyklu se musí simulátor pokusit provést všechny nečasované přechody a načasovat všechny časované. Pokud by však nebyl žádný časovaný přechod nastaven, hlavní podmínka cyklu by nebyla splněna a simulace by neproběhla.

- Hlavní algoritmus simulace:

```
proveď všechny nečasované přechody a nastav časované přechody
inicializace simulačního času
while (kalendář je neprázdný){
    vyjmi první záznam kalendáře
    if (aktivační čas události > konečný čas simulace)
          konec simulace
    nastav simulační čas na aktivační čas události
    proveď přechod v události
    proveď všechny nečasované přechody a nastav časované přechody
}
```

Konec simulace nastane, když aktivační čas události je větší než čas definovaný jako konečný čas simulace.

3.2.2 Algoritmus pro vyhodnocení proveditelnosti přechodu

Pro vyhodnocení proveditelnosti přechodu je využíván algoritmus, který jednoduše prověří, zda má vstupní místo dostatek značek a výstupní místo, zda má dostatek volných pozic pro nové značky.

- Ověření proveditelnosti přechodu:

```
 for (projdi všechna vstupní místa)
     if(vstupní místa nemají dostatek značek pro naplnění kapacity hrany)
      přechod není proveditelný
 přechod je proveditelný

     if(přechod má výstupní místo)
 for (projdi všechna výstupní místa)
     if(výstupní místa nemají dostatek volných pozic)
      přechod není proveditelný (
 přechod je proveditelný
```

Část pseudokódu „`přechod je proveditelný`“ můžeme brát jako pravdivostní hodnotu, kterou musí oba cykly nastavit na *true*.

3.2.3 Algoritmus pro provedení přechodu

V Petriho síti se vyskytují až tři typy přechodů – časovaný, prioritní a pravděpodobnostní. Pro každý typ přechodu je algoritmus vykonávání trošku odlišný.

- Časovaný přechod:

```
if(jedná se o konstantní přechod)
     zjisti zpoždění z přechodu
else
     vypočítej exponenciální zpoždění se středem nastaveným u přechodu
naplánuj událost s daným přechodem
```

- Prioritní přechod:

```
Projdi všechny přechody vztahující se k vstupnímu místu
if(pokud je v okolí přechod s vyšší prioritou)
     neprováděj přechod
proveď přechod
```

- Pravděpodobnostní přechod:

```
vygeneruj číslo od 1 do 100
vyber přechod, do jehož rozsahu patří vygenerované číslo
proveď přechod
```

4 Architektura simulačního modelu/simulátoru

Simulátor byl napsán v jazyce C++. Celý kód je pečlivě okomentován tak, aby mohla být vygenerována dokumentace pomocí Doxygenu (příkaz *make doxygen*).

4.1 Návrh simulátoru

Jazyk C++ je objektově orientovaný, což je velmi využito při implementaci jednotlivých prvků simulátoru. Každý prvek simulátoru je samostatný objekt, který spolupracuje s ostatními pomocí rozhraní.

4.1.1 Třída PlaceTransition

Jedná se o třídu, která obsahuje společné atributy míst a přechodů z modelu.

- Jméno místa/přechodu
- Seznam vstupních hran
- Seznam výstupních hran
- Příznak, zda se jedná o místo

Jméno přechodu je typu `string`, seznam vstupních hran a seznam výstupních hran jsou datové typu `std::vector<Link*>` a příznak, zda se jedná o místo typu `boolean`.

Každé místo a přechod z této třídy dědí tyto vlastnosti a přidává další, které více specifikují další prvek. Krom metod pro získání jednotlivých atributů disponuje třída *PlaceTransition* metodami pro přidání hrany jako vstup `addInputLink()`, pro přidání hrany jako výstup `addOutputLink()` a dvěma metodami pro získání počtu vstupních a výstupních linek `getInputLinkCount()` a `getOutputLinkCount()`.

4.1.2 Třída Place

Jedná se o třídu, která implementuje místo z modelu Petriho sítě. Dědí atributy a metody ze třídy *PlaceTransition*. Dále je rozšířena o tyto atributy:

- Kapacita místa
- Seznam tokenů v místě
- Seznam všech míst v modelu (statický atribut)

Pro každé místo se počítají statistiky, kvůli čemuž musí každé místo obsahovat hodnoty pro výpočty:

- Minimální počet tokenů v místě během simulace
- Maximální počet tokenů v místě během simulace
- Počet změn v daném místě
- Průměrný počet značek v modelu

Kapacita místa je udávána jako celočíselná hodnota, která je implicitně nastavena na nekonečno (hodnota 0). Seznam tokenů v místě je datového typu `std::vector<Token*>`, kdežto seznam všech míst je typu

`std::map<Place*>`. Hodnoty statistik jsou celočíselné hodnoty kromě průměru, který může obsahovat desetinné číslo.
Třída Place je dále rozšířena o konstruktor `Place()`, destruktor `~Place()` a metody pro přidání tokenu do místa `addToken()` a smazání tokenu z místa `removeToken()`. Ostatní metody nastavují/vrací hodnotu některého z atributů.

4.1.3 Třída Transition

Třída *Transition* implementuje přechod z modelu Petriho sítě. Stejně jako třída *Place* dědí atributy ze třídy *PlaceTransition* a poté blíže rozšiřuje svoje atributy o:

- Seznam všech přechodů modelu (statický atribut)
- Typ přechodu
- Hodnota přechodu
- Příznak, zda je přechod načasován

Pro každý přechod jsou počítány statistiky, kvůli čemuž musí každý přechod obsahovat hodnotu pro výpočet:

- Počet vykonání přechodu

Časované přechody navíc využívají zbylé atributy pro svoje statistiky:

- Maximální hodnota provádění
- Minimální hodnota provádění
- Průměrná doba provádění
- Vygenerovaná hodnota

Pro pravděpodobností přechody je navíc vedena statistika, která určuje, v kolika procentech případů byl vybrán daný přechod.
Atribut „vygenerovaná hodnota“ je využit v případě, že událost spjata s daným přechodem je vymazána z kalendáře na základě přerušení.
Seznam všech přechodů modelu je datového typu `std::map<Transiton*>`, hodnota přechodu je celé číslo indikující buď prioritu (0,1,2…), pravděpodobnost (45,55, …) nebo hodnotu času – konstantní čas nebo střední hodnotu exponenciálního rozložení (10,15…).
Ve třídě je využit výčtový typ enumerace pro atribut „*typ přechodu*“. Jedná se o označení typu přechodu hodnotou `TIMED_EXP`, `TIMED_CONST`, `PRIORITY` a `PROBABILITY`.
Mezi důležité metody patři konstruktor `Transition()`, metoda pro kontrolu dostatečného počtu značek ve vstupním místě `checkPlaceOut()`, metoda pro kontrolu dostatečného počtu značek ve výstupním místě `checkPlaceOutput()` a metoda pro přepočítání statistiky v případě smazání záznamu o události `recomputeStatsWithDeleteEventWait()`. Tato metoda slouží k připočítání vygenerované hodnoty pro události, které jsou přerušeny ve vykonávání.

Ostatní metody nastavují/vrací hodnotu některého z atributů.

4.1.4 Třída Link

Třída *Link* je jedna z nejméně rozsáhlých tříd v celém simulátoru. V této třídě je implementována hrana modelu.
Mezi atributy patří:

- Ukazatel na vstup hrany
- Ukazatel na výstup hrany
- Kapacita hrany
- Seznam všech hran modelu (statický atribut)

Ukazatel na vstup nebo výstup hrany je datového typu *PlaceTransition*, protože ukazuje na určité místo nebo přechod modelu. Zda je to místo nebo přechod je pak možné zjistit pomocí metody ve třídě *PlaceTransition* `isPlace()`. Kapacita hrany je celočíselná hodnota a seznam všech hran modelu je typu `std::vector<Link*>`.
Mezi důležité metody patří konstruktor hrany `Link()`, `kterému` je předána kapacita a jména vstupu a výstupu, Ty se potom hledají v seznamech míst a přechodů a ukazatele na tyto místa/přechody jsou uloženy k dané hraně.

4.1.5 Třída Token

Třída *Token* implementuje značky v modelu. Objekt této třídy obsahuje následující atributy:

- Seznam všech značek modelu (statický atribut)
- Ukazatel na místo, ve kterém se nachází
- Příznak, který značí, že token čeká na přesun
- Ukazatel na přechod, se kterým je vložen v události

Ukazatel na místo je typu `Place`, protože víme, že objekt, ve kterém se značka nachází je za každých okolností místo. Ukazatel na přechod je naopak typu *`Transition`*. Seznam všech značek modelu je typu `std::vector<Token*>`.
Konstruktoru `Token()` se předá ukazatel na místo, ve kterém se značka nachází. Mezi zásadní metody této třídy patří deleteTokenFromList(), která vymaže značku ze seznamu všech značek v modelu. Metoda `isTokenProcessedByTransition()` zjistí, zda je ve značce nastaven ukazatel na přechod. Pomocí této metody se poté zjišťuje, která událost se má z kalendáře smazat, pokud nastane přerušení.
Ostatní metody nastavují/vrací hodnotu některého z atributů

4.1.6 Třída Model

Třída *Model* reprezentuje celý model jako jeden objekt. Tento objekt nemá žádné atributy, ale obsahuje zásadní metody pro vytvoření modelu, který je simulován.
Metoda `addPlace()` vytváří místo v modelu, tato metoda je přetížena a to tak, že umožňuje volání metody pouze se jménem místa, které se má vytvořit anebo se jménem a kapacitou (implicitní kapacita je nekonečno – 0).
Metoda `addTransition()` vytvoří v modelu přechod. Tato metoda je také přetížena tak, že umožní vytvořit přechod pouze na základě jména (vytvoří se implicitně prioritní přechod s prioritou 0) nebo na základě jména, typu a hodnoty.
Metoda `addToken()` vytváří značku v místě, které je metodě předáno jako parametr ve formě textu (jméno místa). Metoda je přetížena tak, že umožní vytvořit tolik tokenů, kolik udává druhý celočíselný parametr.
Metoda `addLink()` propojí místo s přechodem (nebo obráceně).

Další důležité metody budou zmíněny níže.

4.1.7 Třída Calendar

Třída *Calendar* implementuje kalendář událostí. Jediným jejím atributem je *seznam všech událostí*, který je typu `std::multiset<Event*, EventSort>`, kde `EventSort` je řadící funkce. Tento typ byl zvolen z důvodu, že je možné mu předat řadící funkci, která pak bude automaticky řadit vložené prvky podle této funkce při vložení. Tato třída disponuje metodou na vkládání událostí do kalendáře `addEvent()`, smazání události `deleteEvent()` a také metodou `getEvent()`, která vrátí první událost z kalendáře.

4.1.8 Třída Event

Třída *Event* implementuje záznam o události. Objekt této třídy obsahuje následující atributy:

- Aktivační čas události
- Vygenerované zpoždění přechodu
- Ukazatel na přechod, který je s událostí spjatý
- Seznam značek události, které se mají přesunout

Aktivační část události je desetinné číslo, stejně jako vygenerované zpoždění přechodu. Ukazatel na přechod je typu `Transition` a seznam značek události je typu `std::vector<Token*>`.
Konstruktor události `Event()` vytvoří novou událost, která je následně vložena do kalendáře. Konstruktor potřebuje tedy aktivační čas, zpoždění přechodu a ukazatel na přechod. Důležitou metodou je také `addTokenToEvent()`. Tato metoda vloží do seznamu tokenů, které se během událostí mají přesunout, zadaný token. Tím je token spjat nejen s přechodem, ale přímo i s událostí v kalendáři.

Ostatní metody nastavují/vrací hodnotu některého z atributů.

4.1.9 Třída Simulator

Třída *Simulator* je srdce celé simulace. V této třídě jsou definovány všechny potřebné funkce, které se starají o logický chod simulátoru. Obsahuje také atributy, které dále obsahují vše potřebné k běhu simulace:

- Ukazatel na model
- Ukazatel na kalendář
- Aktuální simulační čas
- Konečný čas simulace

Ukazatel na model je typu `Model`. Jedná se o ukazatel na celý vytvořený model, který obsahuje všechna místa, přechody a hrany. Ukazatel na kalendář je typu `Calendar` a obsahuje všechny naplánované události. Atributy aktuální simulační čas a konečný čas simulace obsahují desetinná čísla. Konečný čas udává, kdy má simulace skončit.
Konstruktor simulace `Simulator()` vytvoří nové objekty kalendáře a modelu. Další metody budou popsány níže jako popis funkčnosti simulátoru.

4.2 Tvorba modelu

Model simulace je tvořen pomocí metody `createModel()` uvnitř programu v části Simulator. Tato metoda využívá metody ze třídy *Model*, které slouží pro vytvoření jednotlivých prvků modelu. Důležitým aspektem tvorby modelu je fakt, že nejprve je nutné vytvořit místa a přechody, než budou do míst vkládány tokeny, případně budou jednotlivé prvky spojovány mezi sebou. Pokud by toto pravidlo nebylo dodrženo, program se ukončí a vypíše chybovou hlášku s popisem chyby.

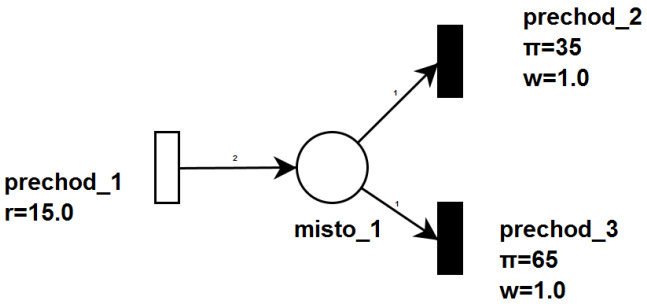
```
void Simulator::createModel()
{
    // přechody
    model->addTransition("prechod_1", 15,Transition::TIMED_EXP);
    model->addTransition("prechod_2, 35, Transition::PROBABILISTIC");
    model->addTransition("prechod_3", 65, Transition::PROBABILISTIC");

    //místa
    Model->addPlace("místo_1");

    //hrany
    Model->addLink("prechod_1 ","místo_1", 2);
    Model->addLink("místo _1 "," prechod _2", 1);
    Model->addLink("místo _1 "," prechod _3", 1);

    // tokeny
    Model->addToken("misto _1", 3);
}
```

Výše uvedený kód by například vytvořil tento model:



Obrázek 2 - jednoduchý model Petriho sítě

4.3 Validace modelu

K validaci modelu slouží metoda `ValidateModel()`, která je součástí třídy *Model*. V této metodě jsou nejprve získány ukazatele na seznamy všech míst a přechodů. Tyto seznamy jsou postupně procházeny a jsou u nich ověřovány následující vlastnosti:
Přechody:

- Každý přechody musí mít alespoň jednu vstupní nebo výstupní hranu
- Nesmí se kombinovat časovaný nebo prioritní přechod s pravděpodobnostním
- Na vstupu pravděpodobnostního přechodu smí být pouze jedno místo
- Na vstupu pravděpodobnostního přechodu nesmí být hrana s kapacitou (pouze implicitní hodnota 1)
- Na výstupu pravděpodobnostního přechodu nesmí být místo s kapacitou (pouze implicitní hodnota 0)

- Součet pravděpodobnostních přechodů musí být roven 100
- Smí se kombinovat pouze časované s časovanými přechody, prioritní s prioritními, pravděpodobnostní s pravděpodobnostními nebo časované s prioritními

Místa:

- Každé místo musí mít alespoň jednu vstupní nebo výstupní hranu
- Počet tokenů v místě nesmí překročit jeho kapacitu

Při procházení přechodů se ověří pouze první bod. Ostatní body se ověřují z místa takovým způsobem, že se pomocí hran dostaneme k přechodu, který je s místem spojen. Takto ověříme celé okolí místa.

4.4 Vyhodnocování a časování přechodů, tvorba kalendáře

Vyhodnocením přechodů začíná celá simulace. Jak již bylo uvedeno výše, před vstupem do hlavního algoritmu simulátoru je nutné provést nečasované přechody a načasovat časované. K tomu slouží hned několik metod, které jsou součástí třídy *Simulator*:

- `performTransitions()`
- `performTransition()`
- `performTransitionFromEvent()`
- `planTransition()`
- `planEvents()`

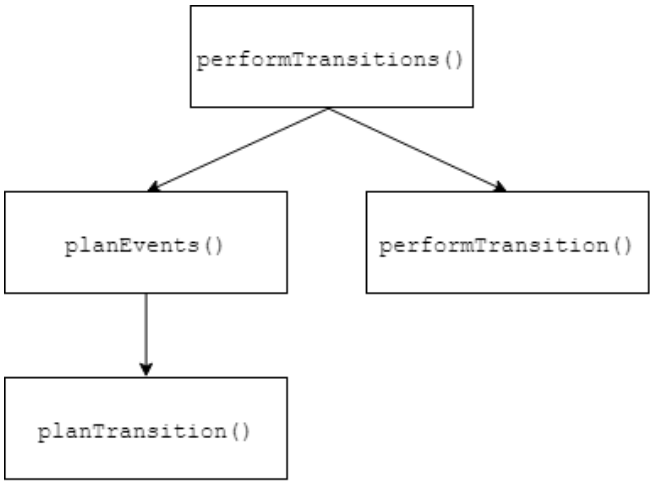
Metoda `performTransitions()` si vytvoří pomocný seznam přechodů, který zamíchá. Následně je vybrán přechod, který se bude zpracovávat. Podle typu přechodu se volá další metoda:

Časovaný přechod: volá se metoda `planEvents()`, ve které se určí počet naplánovaných událostí. Pokud přechod nemá vstupní hrany tak se jedná o generátor, který vygeneruje pouze jednu značku. Pokud má přechod vstupní i výstupní hrany, je naplánována pro každou značku určenou k přesunu jedna událost. Uvažujeme tedy fakt, že dva hráči nehrají stejnou dobu (pokud vstupní a výstupní hrana mají vyšší kapacitu než 1, tak se do kalendáře vloží jeden záznam o události pro více značek). Po rozhodnutí, o který druh přechodu se jedná je volána funkce `planTransition()`, která naplánuje událost s patřičnými parametry.

Nečasovaný přechod: volá se metoda `performTransition()`, která hned vykoná daný přechod, pokud je proveditelný.

Metoda `performTransitionFromEvent()` je volána v hlavním algoritmu simulace, kde vykoná první událost z kalendáře (událost s nejnižším simulačním časem). V této události je řečeno, který přechod se vykoná, a které značky se přesunou (případně vytvoří). Po vykonání přechodu je nastaven aktuální simulační čas na aktivační hodnotu události. Událost je následně smazána z kalendáře.

Na *Obrázku 2* je k dispozici graf volání, který znázorňuje návaznost funkcí jedna na druhé.
w



Obrázek 3- Graf volání metod při zpracování přechodu

Všechna vybraná místa/přechody/hrany atd. v celém simulátoru jsou vždy vybrána náhodně.

Časování přechodů probíhá na základě vygenerovaného zpoždění (pokud je přechod konstantní vezme se konstantní zpoždění). Při tvorbě události se vezme aktuální simulační čas, přičte se k němu zpoždění a vloží se do kalendáře. Značka, která má být událostí přenesena je pro opětovné časování tímto zablokována. V případě, že nastane přerušení události (v našem případě např. výpadek proudu v herně) je značka přenesena jiným přechodem a událost je z kalendáře smazána.

4.5 Náhodné generování pomocí funkce rand a srand

Pro generování pseudonáhodných čísel je využívána funkce `rand()`. Tato funkce vrací náhodné číslo v rozmezí 0 až RAND_MAX. Problémem je, že bez bližší specifikace generuje funkce `rand()` čísla stále ve stejném pořadí. Aby se zabránilo tomuto problému, využije se funkce `srand()`, která jako parametry očekává číslo typu `unsigned int`, kterému se říká semínko.
Jako semínko se často nastavuje aktuální čas systému z důvodu, že nikdy při generování čísla není stejný a tudíž funkce vygeneruje pseudonáhodnou posloupnost. Pokud by funkce `srand()` s náhodným semínkem nebyla použita, generátor by vracel stále stejná čísla. V našem případě je použito semínko s aktuálním systémovým časem.

4.6 Statistiky

Statistiky jsou uživateli předávány ve dvou částech. První část jsou informace o každém přesunu jakékoliv značky v modelu a přehled celkového počtu značek v každém místě. Podobu této statistiky je možné vidět v bodě 5.2.
Druhou částí statistiky je uložení některých informací u každého místa a přechodu.

Místo:

- Minimální počet značek během simulace
- Maximální počet značek během simulace
- Aktuální počet značek
- Počet změn (jedná se o počet vložení a přesunu značky)
- Výpočet průměrné hodnoty značky

Přechod:

- Počet provedení přechodu

Časovaný přechod:

- Počet provedení přechodu
- Minimální vygenerované zpoždění
- Maximální vygenerované zpoždění
- Průměrné vygenerování zpoždění

Pravděpodobnostní přechod:

- Procentuální vyjádření, v kolika případech byl přechod proveden

Podobu této statistiky je také možné vidět v bodě 5.2.

5 Podstata simulačních experimentů a jejich průběh

Cílem experimentování bylo zjistit, zda vytvořený simulátor správně zpracovává zadané Petriho sítě jak z pohledu validace modelu, tak z pohledu simulačního. Dalším cílem bylo zajištění ucelených informací o provádění simulace tak, aby byly snadno rozpoznatelné a srozumitelné.

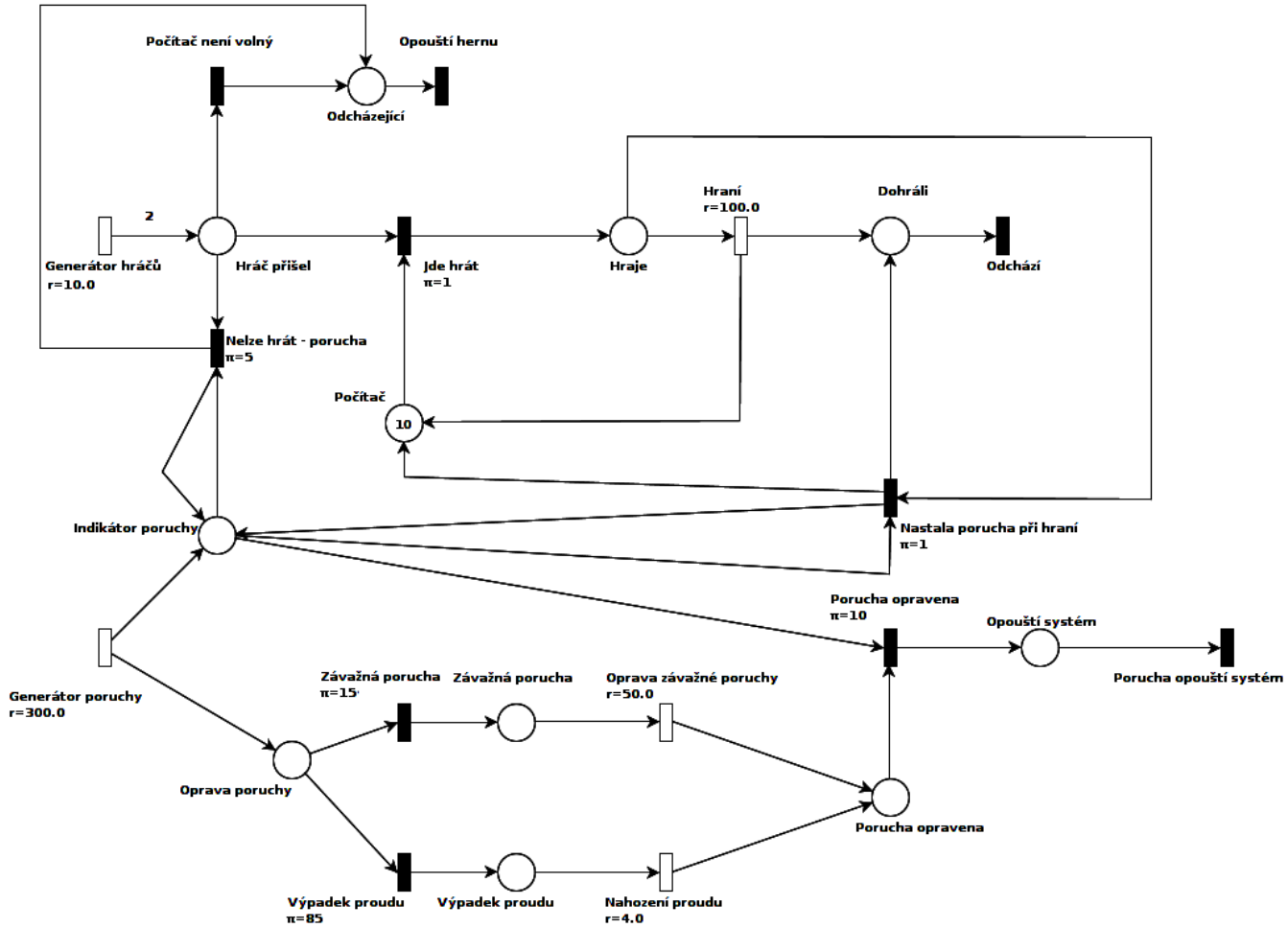
5.1 Použití programu

./PN-simulator {-p} {-s čas}
Přepínač –p značí přepnutí mezi vstupním modelem simulace. Implicitně je nastaven model Herna, použití přepínače –p spustí model Překladiště.
Přepínač –s nastaví maximální simulační čas. Implicitně je maximální simulační čas nastaven na 10000.
Pokud je program spuštěn bez přepínačů je spuštěn s implicitními hodnotami.

Pro spuštění programu je možné využít také příkaz *make run*.

5.2 Postup experimentu

Pro účely experimentování se simulátorem byl vybrán model Herna (*Obrázek č. 3*) z demonstračního cvičení, který byl následně upraven do podoby, která zahrnuje většinu možností, kterými může Petriho síť disponovat.



Obrázek 4 - Model herna

Hráči do herny přichází po dvou v intervalu s exponenciálním středem 10 minut. Hráč po příchodu zabere počítač a hraje po vygenerovanou dobu s exponenciálním středem 100 minut. Každých 300 minut nastane v herně porucha. V 85 % případů se jedná pouze o výpadek proudu, který je opraven za konstantní dobu 4 minut, v 15 % přídech se jedná o závažnější chybu, která se opravuje konstantní dobu 50 minut. Během poruchy přestanou všichni hráčí hrát a opouští hernu a nikdo další v herně hrát nemůže. Kdokoli přijde během poruchy, nebo když není volný počítač, odchází.

5.3 Přehled statistiky

První část statistiky znázorňuje vygenerování dvou hráčů v čase 2.09673 a průběh poruchy, kde nastane přerušení hraní, hráči odchází a počítače se vrací v čase 27.3612. Počet počítačů je od začátku na hodnotě 10.

1. Část statistik

```
Začátek provádění simulace v čase: 0
Proveden přechod: p_generator_hracu v čase: 2.09673
    Vložení tokenu do místa: m_hrac_prisel
    Vložení tokenu do místa: m_hrac_prisel
Počet tokenů v místech:
|m_dohrali(0)|m_hrac_prisel(2)|m_hraje(0)|m_indikátor_poruchy(0)|m_odchazejici
(0)|m_oprava_normalni_poruchy (0)|m_oprava_poruchy(0)|m_oprava_zavazne_poruchy
(0)|m_pocitace(10)|m_porucha_opousti_system(0)|m_porucha_opravena(0)|

Proveden přechod: p_nastala_porucha_pri_hrani v čase: 27.3612
    Přesun tokenu z místa: m_indikátor_poruchy
    Přesun tokenu z místa: m_hraje
```

```
    Vložení tokenu do místa: m_indikátor_poruchy
    Vložení tokenu do místa: m_dohrali
    Vložení tokenu do místa: m_pocitace
Počet tokenů v místech:
|m_dohrali(1)|m_hrac_prisel(0)|m_hraje(6)|m_indikátor_poruchy(1)|m_odchazejici(0)|
m_oprava_normalni_poruchy(1)|m_oprava_poruchy(0)|m_oprava_zavazne_poruchy(0)|m_poc
itace(4)|m_porucha_opousti_system(0)|m_porucha_opravena(0)|
```

U každého přechodu je napsán čas, ve kterém byl proveden. Pokud se jedná o nečasovaný přechod, je vypsán současný stav simulace.

Druhá část statistik je celkový přehled míst a přechodů. Zatímco se první druh statistiky vypisuje po dobu celého běhu simulace, tento druh se vypisuje pouze na konci simulace.

2. Část statistik

```
######################
Místo: m_odchazejici
Počet změn: 756
Minimální počet značek: 0
Maximální počet značek: 2
Průměrný počet značek: 0.685185
Aktuální počet značek: 0
######################
```

```
######################
Přechod: p_hraní
Celkový počet provedení: 447
Nejkratší doba provedení: 0.0140891
Nejdelší doba provedení: 573.018
Průměrná doba provedení: 75.6389
Průměrná vygenerovaná doba: 100.235
######################
```

5.4 Dokumentace jednotlivých experimentů

Po naimplementování jednotlivých funkcí simulátoru přišlo na řadu experimentování s model. Hlavním důvodem experimentování bylo nalezení vhodných hodnot pro model tak, aby byly rovnoměrně prověřeny všechny jeho části.

Hodnoty, které bylo neúčinnější měnit:

- Doba hraní
- Doba generování hráčů
- Doba opravy poruchy
- Počet počítačů

Pro menší rozsáhlost dokumentace byly zvoleny tři provedené experimenty.

5.4.1 První experiment

Jako první byl vyzkoušet experiment s hodnotami, které byly uvedeny na demonstračním cvičení. Spočítaná statistika je uvedena níže.

```
######################
Přechod: p_generator_hracu
Celkový počet provedení: 959
Nejkratší doba provedení: 0.014348
Nejdelší doba provedení: 77.7827
Průměrná doba provedení: 10.4062
######################

######################
Přechod: p_dohral_odchazi
Celkový počet provedení: 943
######################
```

```
######################
Přechod: p_pocitac_neni_volny
Celkový počet provedení: 0
######################


######################
Přechod: p_nelze_hrat_porucha
Celkový počet provedení: 16
######################
```

Z výpisu statistiky bylo možné zjistit, že přechod „p_pocitac_neni_volny“ se neprovedl ani jednou během celé simulace. Počítačů tedy bylo dostatek pro všechny příchozí hráče. Ze statistiky je dále patrné, že během poruchy přišlo a hned zase odešlo 16 hráčů.

5.4.2 Druhý experiment

V dalším experimentu byl změněn typ přechodu „p_hraní“ z časovaného konstantního na časovaný exponenciální se středem 30. Generátor hráčů byl upraven tak, aby každých 15 minut vygeneroval dva hráče. Bylo toho dosaženo nastavením hodnoty na 15 a kapacita odchozí hrany byla navýšena na 2. Poslední úpravou, spíše jen tak ze zvědavosti, bylo změněno generování přerušení z 300 minut na 600 minut.

```
######################
Přechod: p_generator_hracu
Celkový počet provedení: 708
Nejkratší doba provedení: 0.008228
Nejdelší doba provedení: 96.1031
Průměrná doba provedení: 14.1039
######################
```

```
######################
Přechod: p_hraní
Celkový počet provedení: 1352
Nejkratší doba provedení: 0.0331
Nejdelší doba provedení: 220.875
Průměrná doba provedení: 29.8584
######################
```

```
######################
Přechod: p_nelze_hrat_porucha
Celkový počet provedení: 14
######################
```

```
######################
Přechod: p_pocitac_neni_volny
celkový počet provedení: 0
######################
```

```
######################
Přechod: p_dohral_odchazi
Celkový počet provedení: 1399
######################

Proveden přechod: p_dohral_odchazi v čase: 9997.1
   Přesun tokenu z místa: m_dohrali
   Počet tokenů v místech:
|m_dohrali(0)|m_hrac_prisel(0)|m_hraje(3)|m_indikátor_poruchy(0)|
m_odchazejici(0)|m_oprava_normalni_poruchy(0)|m_oprava_poruchy(0)|
m_oprava_zavazne_poruchy(0)|m_pocitace(17)|m_porucha_opousti_system(0)|
m_porucha_opravena(0)|
```

Ze statistiky je vidět, že přechod „p_pocitac_neni_volny“ se opět neprovedl ani jednou. Toto nastavení tedy není pro nás vyhovující. Pro kontrolu je vidět, že bylo vygenerováno 708 × 2 = 1416 hráčů. 1399 hráčů hrálo, dohrálo a odešlo. Dalších 14 přišlo a odešlo během poruchy. 3 hráči ještě hrají -1399 + 3+14 = 1416. Je tedy patrné, že všechny značky, které byly vytvořeny, opustili systém.

5.4.3 Třetí experiment

Nastavení pro třetí popsaný experiment je stejné, které v současnosti simuluje vytvořený simulátor. Ve vytvořené herně je k dispozici 10 počítačů. Hráči přichází po dvojicích v čase s exponenciálním středem 15 minut. V herně nastane porucha každých 300 minut. V 85 % nastane pouze výpadek proudu, který je za 4 minuty obnoven. Ve zbylých 15 % nastane závažnější porucha, která vyžaduje delší opravu a to 50 minut. Doba hraní je generována exponenciálním přechodem se středem 100.

```
######################
Přechod: p_generator_hracu
Celkový počet provedení: 657
Nejkratší doba provedení: 0.0103244
Nejdelší doba provedení: 148.961
Průměrná doba provedení: 15.2071
######################
```

```
######################
Přechod: p_hraní
Celkový počet provedení: 648
Nejkratší doba provedení: 0.0813336
Nejdelší doba provedení: 484.628
Průměrná doba provedení: 75.6389
Průměrná vygenerovaná doba: 102.627
######################
```

```
######################
Přechod: p_generator_poruch
Celkový počet provedení: 33
Nejkratší doba provedení: 4.85269
Nejdelší doba provedení: 822.918
Průměrná doba provedení: 298.898
######################
```

```
######################
Přechod: p_automat_neni_volny
Celkový počet provedení: 384
######################
```

```
######################
Přechod: p_dohral_odchazi
Celkový počet provedení: 876
######################
```

```
######################
Přechod: p_porucha_opousti_system
Celkový počet provedení: 33
######################
```

```
######################
Přechod:
p_opravovani_normalni_poruchy
Celkový počet provedení: 26
Nejkratší doba provedení: 4
Nejdelší doba provedení: 4
Průměrná doba provedení: 4
######################
```

```
######################
Přechod: p_opravovani_zavazne_poruchy
Celkový počet provedení: 7
Nejkratší doba provedení: 20
Nejdelší doba provedení: 20
Průměrná doba provedení: 20
######################
```

```
Proveden přechod: p_nehral_odchazi v čase: 9991.05
    Přesun tokenu z místa: m_odchazejici
    Přesun tokenu z místa: m_odchazejici
    Počet tokenů v místech:
|m_dohrali(0)|m_hrac_prisel(0)|m_hraje(10)|m_indikátor_poruchy(0)|
m_odchazejici(0)|m_oprava_normalni_poruchy(0)|m_oprava_poruchy(0)|
m_oprava_zavazne_poruchy(0)|m_pocitace(0)|m_porucha_opousti_system(0)|
m_porucha_opravena(0)|
```

Ze statistik už je patrné, že byly provedeny všechny přechod. Třetina příchozích hráčů nemá volný počítač, a proto hned odchází. Generování hráčů průměrně trvá 15.2071 minuty, což se liší od středové hodnoty o 1,38 %. Porucha je průměrně vygenerována každý 298.898 minut, což se od středové hodnoty liší o 0,3674%. Hraní průměrně trvá 102.627 minut, což se liší o 2,627 %. Naměřené průměrné hodnoty generátorů odpovídají požadavkům. Průměrné hodnoty se zlepšují s větší délkou simulace.
Odlišnosti z důvodu menšího počtu provedení přechodu jsou patrné u přechodů „p_opravovani_normalni_poruchy“ a „p_opravovani_zavazne_poruchy“, které mají nastavené hodnoty 85% a 15 %. Z počtu poruch a provedeních jednotlivých přechodů bylo zjištěno, první z přechodů byl vybrán zhruba v 78,78 % namísto 85 % a druhý z přechodů byl zvolen v 21,22 % místo nastavených 15 %. Tyto hodnoty by se ovšem zlepšovali s větším počtem vygenerovaných poruch.

6 Shrnutí simulačních experimentů a závěr

Provedenými experimenty bylo zjištěno, že naimplementovaný simulátor P/T Petriho sítí splňuje všechny potřebné náležitosti, a je tedy schopen simulovat libovolnou Petriho sít.
Díky experimentům bylo také ověřeno, že generování exponenciálních čísel funguje náhodně, a průměrné hodnoty se blíží nastaveným exponenciálním středům víc v závislosti na délce simulace.

Projekt byl přínosný hned v několika směrech:

- Přiblížení simulační profese
- Zvýšení zkušeností s programovacím jazykem C++
- Zvýšení znalostí pro předmět IMS

Všechny tyto nabyté vědomosti určitě zúročíme v následujícím studiu i v praxi.

7 Reference

[1] PERINGER, Petr. *Modelování a simulace* [online]. [cit. 2015-12-07]. Dostupné z WWW: <https://www.fit.vutbr.cz/study/courses/IMS/public/prednasky/IMS.pdf>
[2] HRUBÝ, Martin. *IMS democvičení #2* [online]. [cit. 2015-12-07]. Dostupné z WWW: <http://perchta.fit.vutbr.cz:8000/vyuka-ims/uploads/1/diskr2-2011.pdf>