VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

GRAFICKÝ EDITOR OBJEKTOVĚ ORIENTOVANÝCH PETRIHO SÍTÍ

GRAFICAL EDITOR OF OBJECT ORIENTED PETRI NETWORKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Antonín Neužil
AUTHOR

VEDOUCÍ PRÁCE — Ing. Radek Kočí Ph.D.
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/19329/2016/xneuzi05

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Neužil Antonín**
Obor: Informační technologie
Téma: **Grafický editor Objektově orientovaných Petriho sítí OOPN Editor**
Kategorie: Softwarové inženýrství

Pokyny:

1. Prostudujte formalismus Objektově orientovaných Petriho sítí (OOPN) a jazyk PNtalk. Seznamte se s aktuální implementací simulačního serveru OOPN a komunikačním protokolem.
2. Navrhněte nástroj pro editaci a ladění modelů OOPN. Nástroj musí umožnit ukládání a načítání modelů ve vhodném formátu a ve formátu jazyka PNtalk. Dále musí umožnit vzdálenou práci se simulačním serverem, tj. získávat modely ze serveru, posílat modely na server a spustit, zastavit a krokovat simulaci. Nástroj umožňuje zobrazit aktuální stav simulace.
3. Implementujte nástroj v jazyce Java. Rozšiřte nástroj o možnost exportu modelů do formátu SVG (Scalable Vector Graphics).
4. Vytvořte manuál a sadu příkladů demonstrující možnosti nástroje, zejména komunikaci se simulačním serverem.
5. Diskutujte dosažené výsledky a navrhněte možná rozšíření nástroje. Výsledky také prezentujte formou posteru.

Literatura:

- Janoušek, V.: Modelování objektů Petriho sítěmi, Brno, CZ, UIVT FEI VUT, 1998

Pro udělení zápočtu za první semestr je požadováno:

- První dva body.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kočí Radek, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Tato bakalářská práce se zabývá Petriho sítěmi a jejich modelování. V první řadě práce popisuje formalismy Petriho sítě. Dále se také zaměří na objektovou orientovanost sítí. V druhé řadě se práce specializuje na vytvořenou aplikaci a tou je grafický editor objektově orientovaných Petriho sítí. Součástí práce je postup návrhu a implementace již zmíněné aplikace. Aplikace je vytvořena pomocí jazyku Java a je kompatibilní pouze s operačním systémem Windows. Pro tvorbu aplikace bylo použito vývojové prostředí NetBeans IDE 8.0.2. Výsledkem celé bakalářské práce je tedy aplikace pro tvorbu modelů Objektově orientovaných Petriho sítí a jejich simulací pomocí přiloženého serveru. Několik příkladů těchto modelů a jejich výsledky jsou součástí této práce. Tyto příklady slouží jako manuál pro použití aplikace.

Abstract

This bachelor thesis deals with Petri nets and their modeling. First of all, the thesis describes the formalism of the Petri network. It will also focus on object orientation of networks. In the second row, the thesis is specialized on the created application and it is a graphical editor of object-oriented Petri nets. Part of the thesis is the process of designing and implementing the already mentioned application. The application is created using programming language Java and is compatible only with operating system Windows. NetBeans IDE 8.0.2 development environment was used to create the application. The result of the whole bachelor thesis is an application for creation of models of object-oriented networks and their simulation with the attached server. Several examples of these models and their results are part of this thesis. These examples serve as an application manual.

Klíčová slova

Petriho sítě, Java, NetBeans, Objektově orientované Petriho sítě, PNtalk, Windows, IDE.

Keywords

Petri networks, Java, NetBeans, Object oriented Petri networks, PNtalk, Windows, IDE.

Citace

Neužil, Antonín. *Grafický editor Objektově orientovaných Petriho sítí*, bakalářská práce, Brno, FIT VUT v Brně, 2017

Grafický editor Objektově orientovaných Petriho sítí

Prohlášení

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

……
Antonín Neužil
17. května 2017

Poděkování

Rád bych poděkoval za odborné a velmi cenné rady poskytnuté vedoucím bakalářské práce Ing. Radkem Kočím Ph.D.

Obsah

1 Úvod

Petriho síť je účinný nástroj pro modelování a simulaci systému. Grafická náročnost Petriho sítí není velká a zároveň je snadno pochopitelná. Proto představuje vhodnou možnost pro experimentování s modely různých systémů. Existuje více typů Petriho sítí, které poskytují různé možnosti pro modelování. Petriho sítě se vyvíjí přes padesát let a za tuto dobu se jejich modelovací schopnost značně rozrostla. Následující kapitola popisuje nejběžnější typy Petriho sítí.

Cílem práce je vytvořit grafický editor pro práci s Objektově orientovanými Petriho sítěmi. Editor musí být schopen pracovat s jazykem PNtalk, který představuje konkrétní implementací Objektově orientovaných Petriho sítí. Tato implementace je podrobněji popsána v kapitole 2.1.9.

Třetí a čtvrtá kapitola popisují analýzu zadání projektu a grafický návrh editoru. Pátá kapitola se zabývá implementaci a strukturou editoru. Implementace popisuje závislost zdrojového kódu na jednotlivých prvcích. Struktura představuje hierarchii zdrojového kódu editoru.

Šestá kapitola zobrazuje příklady a postup k jejich vytvoření. Příklady jsou vizualizovány pomocí obrázku, které byly vystřiženy z vytvořeného editoru.

2 Petriho sítě

Tato kapitola popisuje Petriho sítě a jejich formalismus. Petriho sítě reprezentují matematický nástroj pro modelování a simulaci distribuovaných systémů. Petriho sítě taktéž mohou popisovat paralelismus a konflikty u moderních systémů. Tudíž je lze využít k návrhu, analýze a testování různých systémů.

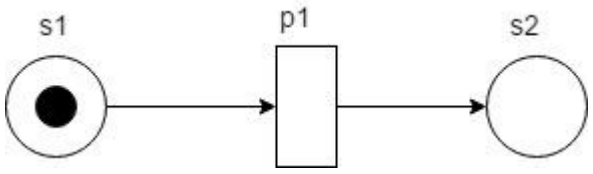
Příkladem vhodného systému pro návrh pomocí Petriho sítě může být např. Návrh na chod automatické montážní linky. V tomto případě lze chování linky poměrně snadno předvídat a podle parametrů jednotlivých strojů také navrhnout. Pomocí Petriho sítí lze také navrhnout speciální stavy linky jako např. havarijní stav. Po přihlédnutí na určitou abstrakci lze také navrhnout chování personálu, který drží linku v chodu.

2.1 Typy Petriho sítí

Základní kámen Petriho sítí položil v roce 1962 ve své disertační práci německý matematik Carl Adam Petri. Od té doby se Petriho sítě vyvíjejí a vznikly i odlišné typy. Tyto typy se od sebe většinou liší přidáním jiného prvku pro zvýšení modelovací schopnosti. Typy Petriho sítí:

- C/E (Condition / Evennt) Petriho sítě
- P/T (Place / Transition) Petriho sítě
- P/T Petriho sítě s inhibičními hranami
- P/T Petriho sítě s prioritami
- P/T Petriho sítě s pravděpodobností
- Časované Petriho sítě
- Barevné Petriho sítě
- Hierarchické Petriho sítě
- Objektově orientované Petriho sítě

[3]



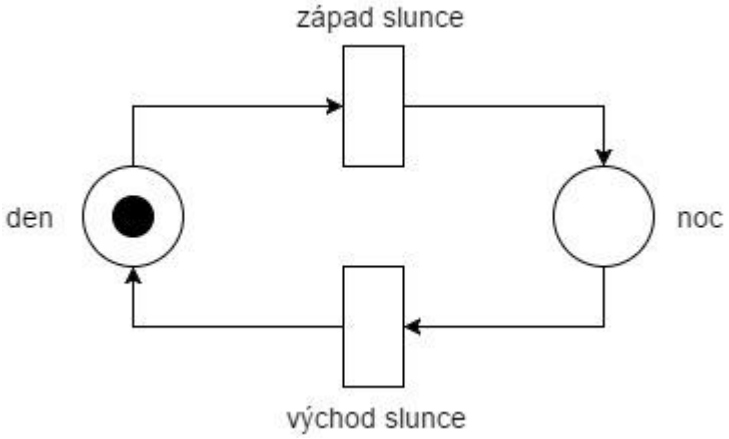
Obrázek 2.1: Základní koncept

Obrázek 2.1 je tvořen základními prvky, které jsou pro většinu typy Petriho sítí společné. I když jsou tyto prvky společné, jednotlivé typy mohou na ně nahlížet z jiného úhlu a také je jinak označovat. Největší rozdíl úhlu pohledu je mezi C/E a P/T sítěmi.

2.1.1 C/E Petriho sítě

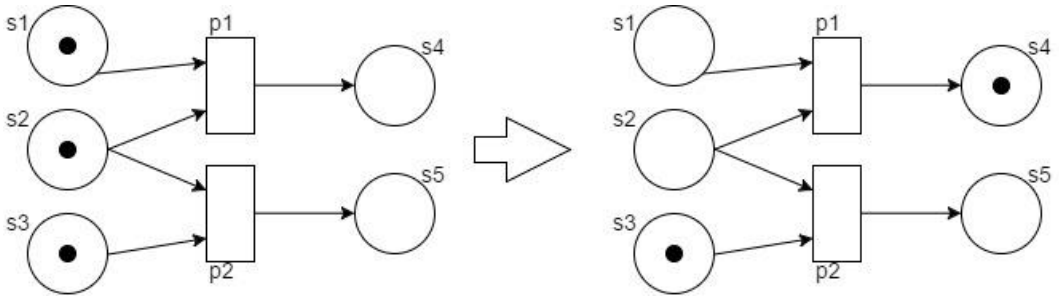
Koncept C/E Petriho síť se skládá z několika prvků, které tvoří grafické model. Prvním je podmínka, kterou znázorňuje kruh nebo elipsa. Příkladem jsou podmínky z obrázku 2.1, které jsou pojmenovány *s1* a *s2*. Druhým prvkem je událost. Událost obvykle je vyobrazena pomocí obdélníku, čtverce nebo svislé úsečky. V obrázku 2.1 událost znázorňuje obdélník *p1*. Dalším prvkem jsou orientované hrany. Hrany mohou směřovat pouze od podmínky do události nebo naopak. V žádném případě hrany nesmí spojovat stejné prvky. Čtvrtým prvkem je značka nebo také „žeton"[1]. Značky jsou zakreslovány jako tečky a vždy se nachází uvnitř podmínky. Na obrázku 2.1 se objevuje pouze jedna značka a to v podmínce *s1*. Výskyt značky uvnitř podmínky určuje pravdivost dané podmínky. Pokud se značka nachází v podmínce, pak je podmínka splněna. Pokud se v podmínce žádná značka nenachází, pak podmínka splněna není. Součástí C/E Petriho sítě je také počáteční značení. Tím se rozumí úvodní rozložení značek v síti.

Podmínky můžeme rozdělit do dvou typů vzhledem k jejich orientaci k určité události. Prvním typem je vstupní podmínka. Podmínka se stává vstupní, pokud je propojena hranou s událostí a zároveň propojující hrana je směrována z podmínky do události. V obrázku 2.1 je touto podmínkou *s1* ve vztahu k události *p1*. Druhým typem podmínky je výstupní. Výstupní podmínka je opačný případ vstupní, kdy hrana je orientována z události do podmínky. Tento případ znázorňuje vztah události *p1* a podmínky *s2* z obrázku 2.1. [2]



Obrázek 2.2: Denní cyklus

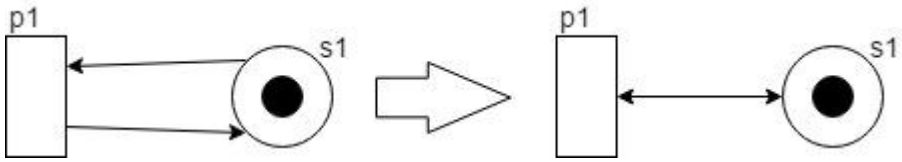
Obrázek 2.2 popisuje denní cyklus, kdy se střídá den a noc. Z aktuálního značení v grafu vyplývá, že momentální stav je v denní fázi. Po uskutečnění událost *západ slunce* graf změní značení v síti. Značka se přesune z podmínky den do podmínky noc a tím nastane noční fáze. Z toho vyplývá, že událost je proveditelná, jestliže všechny její vstupní podmínky jsou splněny. Na obrázku 2.2 toto vyjádření vysvětlují události *východ slunce* a *západ slunce*. V aktuální podobě je událost *západ slunce* proveditelná a to proto, že její vstupní podmínka *den* je pravdivá. Naopak událost *východ slunce* je neuskutečnitelná, protože její vstupní podmínka *noc* je nepravdivá a to z toho důvodu, že neobsahuje značku.



Obrázek 2.3: Konflikt událostí

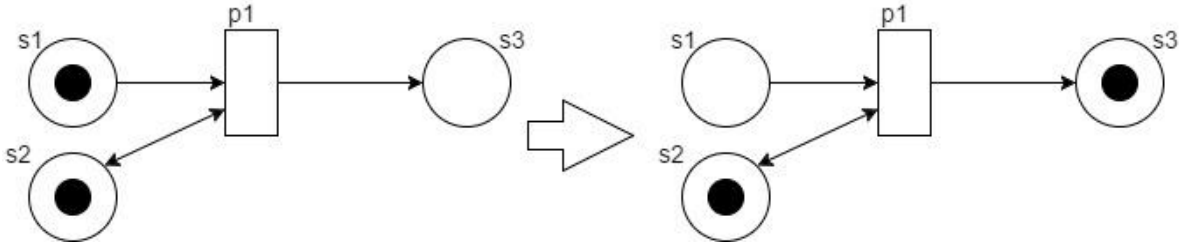
Vždy nejsou příklady C/E jednoduché tak, že každá událost obsahuje pouze jednu vstupní podmínku. Případ více vstupním podmínek demonstruje obrázek 2.3, kde obě události *p1* a *p2* mají vlastní vstupní podmínky *s1* a *s3*, ale také jednu společnou *s2*. Z obrázku vyplývá, že obě události mají splněné své vstupní podmínky, ale nelze je provést obě. Tím dochází ke konfliktu dvou událostí. Náhodně se provede jedna z událostí a výsledek lze nalézt na pravé straně obrázku 2.3. Výsledkem je tedy uskutečnění události *p1*, kde výstupní podmínkou je *s4*, ve které se objeví značka. Po této změně sítě již událost *p2* nelze provést a stav sítě se tedy nebude dále měnit.

Poslední zde ještě nezmíněný prvek se nazývá testovací hrana. Ve své podstatě se jedná o složeninu dvou typů hran spojující jednu událost a jednu podmínku. Sloučením těchto hran dostaneme uzavřenou smyčku mezi podmínkou a událostí.[2]



Obrázek 2.4: Testovací hrana

Obrázek 2.4 popisuje vznik testovací hrany. V levé části obrázku se nachází výše zmíněná smyčka, která je tvořena událostí *p1*, podmínkou *s1* a dvěma spojujícími hranami. Pro zjednodušení zápisu se testovací hrana kreslí jako šipka směřující do obou směrů, jak ukazuje pravá strana obrázku 2.5.



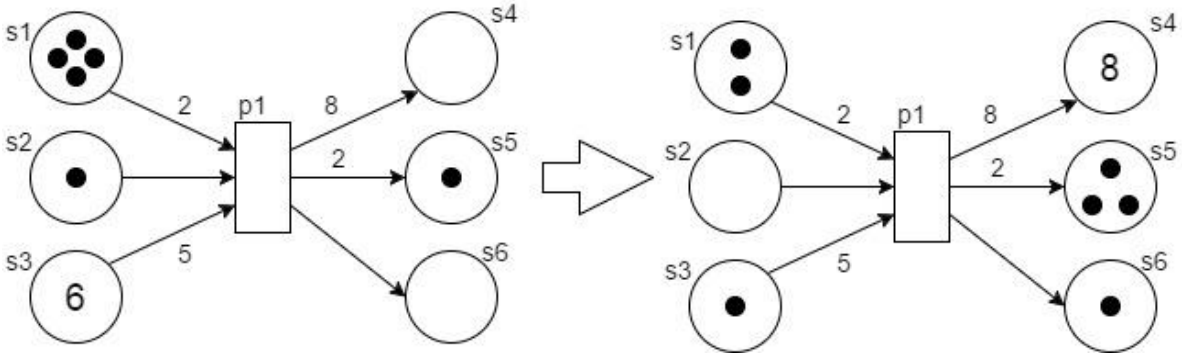
Obrázek 2.5: Příklad testovací hrany

Chování testovací hrany popisuje obrázek 2.5. Na něm se vyskytuje jediná událost *p1*, jejíž všechny vstupní podmínky jsou pravdivé, a proto je uskutečněna. Výsledek provedení události popisuje pravá strana obrázku. Důležité je všimnout si podmínky *s2*, jejíž pravdivost se nezměnila. Tento případ si lze představit, jako odebrání a okamžité navrácení značení zpět do podmínky *s2*.

2.1.2 P/T Petriho sítě

P/T Petriho sítě se po vizuální stránce od C/E Petriho sítěmi tolik neliší. Největší rozdíl se nachází v interpretaci jednotlivých prvků. P/T síť je tvořena následujícími prvky:

- Místa (stavy) jsou znázorněna pomocí kružnic nebo elips.
- Přechody jsou reprezentovány čtverci, obdélníky nebo krátkými úsečkami.
- Orientované hrany jsou vyobrazeny šipkami směřující od míst do přechodů, nebo naopak.
- Váha pro každou hranu sítě.
- Značení (žeton) je graficky reprezentovány tečkami nebo čísly uvnitř míst.
- Kapacity jednotlivých míst.
- Nastavení úvodního značení, které určuje počet žetonů uvnitř každé podmínky.



Obrázek 2.6: P/T sít s váhou hran

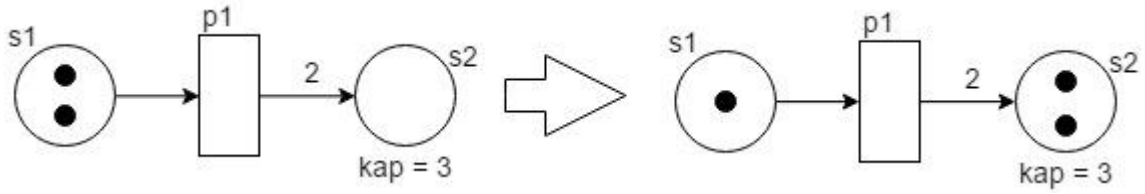
Obrázek číslo 2.6 znázorňuje model P/T Petriho sítí. Jsou v něm obsaženy všechny výše zmíněné prvky kromě kapacity, která je součástí obrázku 2.7. Místa (stavy) jsou označeny jako *s1* až *s6*. Přechod je v obrázku pouze jeden a je označen hodnotou *p1*. Orientované hrany jsou reprezentovány pomocí šipek vedoucích z míst do přechodu nebo z přechodů do míst.

Stavy mohou obsahovat tečku nebo číslo, které reprezentuje značku (žeton). Například stav *s1* obsahuje dva žetony, které jsou znázorněny dvěma tečkami. Druhým případem je znázornění pomocí čísla, které je graficky zobrazeno ve stavu *s3*. Pokud se ve stavu nachází větší počet žetonů, pak je vhodné použít číslo na místo značek. Značka představuje určitou jednotku, která může v konkrétních příkladech reprezentovat proměnnou. Například pokud by byla modelována automatická myčka aut, pak auta mohou představovat žetony, které prochází různými stavy v síti stejně jako auta určitými mycími procesy.

V obrázku 2.6 jsou vyobrazena čísla poblíž orientovaných hran. Tyto čísla udávají váhu jednotlivých hran. Pokud u hrany není explicitně zadána váha, pak výchozí hodnotou je číslo jedna. Váha hrany určuje počet žetonů, který má být odebrán nebo přidán do místa. Například stav *s1* v levé části obrázku 2.6 obsahuje čtyři žetony. K uskutečnění přechodu *p1* je potřeba odebrat dva žetony ze stavu *s1*. Pravá strana obrázku popisující výsledek uvádí počet zbylých žetonů obsažených ve stavu *s1* po provedení přechodu *p1*.

Místa spojená s přechodem pomocí hrany, lze dělit do dvou množin podle orientace vůči přechodu. Místo náleží vstupní množině přechodu, pokud z místa vychází hrana směřující do zmíněného přechodu. V opačném případě místo může náležet výstupní množině přechodu, pokud ze

zvoleného přechodu vede hrana do již zmíněného místa. V příkladu na obrázku 2.6 vstupní množinu přechodu *p1* tvoří stavy *s1*, *s2* a *s3*. Naopak výstupní množina přechodu *p1* obsahuje stavy *s4*, *s5* a s*6*. [2]



Obrázek 2.7: P/T síť s kapacitou

V obrázku 2.7 je u stavu *s2* znázorněna kapacita místa. Kapacita popisuje maximální počet žetonů, které se mohou nacházet uvnitř místa v jednom momentě. Pokud kapacitě není explicitně zadána u místa, pak výchozí hodnotou je nekonečno.

Aby přechod byl uskutečnitelný, musí splňovat dvě podmínky. První podmínka: všechny místa vstupní množiny přechodu musí obsahovat minimálně takový počet žetonů, kolik je váha hrany vedoucí z místa do přechodu. Konkrétní příklad této podmínky je přehledně znázorněn v obrázku 2.7, kde v levé části obrázku místa *s1*, *s2* a *s3* ze vstupní množiny přechodu *p1* splňují první podmínku. Po provedení přechodu *p1* místa *s2* a *s3* už nesplňují podmínku a proto nelze přechod již uskutečnit. Druhá podmínka: všechny místa výstupní množiny přechodu s explicitně zadanou kapacitou nesmí tuto kapacitu překročit po provedení přechodu. Pokud by počet žetonů obsažený ve výstupním místě sečtený s váhou hrany přesahoval kapacitu místa, pak je podmínka nesplněna a tudíž i přechod neproveditelný. Tato podmínka je demonstrována v příkladu na obrázku 2.7. V levé části obrázku je druhá podmínka splněna neboť po sečtení váhy hrany vedoucí z přechodu *p1* do stavu *s2* a aktuálního počtu žetonů ze stavu *s2* výsledek nepřesahuje kapacitu stavu *s2*. V pravé části obrázku je zobrazen výsledek po provedení přechodu. I když je přechod *p1* stále proveditelný podle první podmínky, tak druhá podmínka není splněna a proto nelze přechod *p1* opět uskutečnit.

Testovací hrany jsou stejně jako u C/E sítí součástí P/T sítí. Jejich grafická reprezentace je taktéž znázorněna šipkou směřující zároveň do místa i přechodu. Chování testovací hran je podrobně popsáno v kapitole: 2.1.1 C/E Petriho sítě.

Definice 2. 1. 2 P/T Petriho síť je šestice $N = (S, P, H, Z, V, K)$, [4] kde

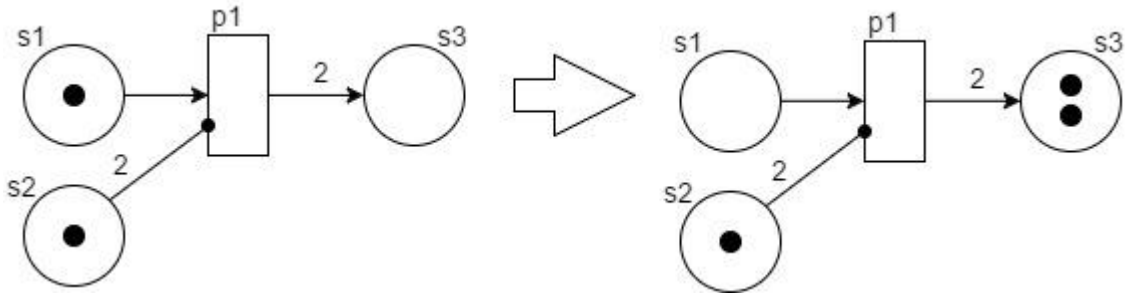
1. S je konečná množina stavů (míst)
2. P je konečná množina přechodů
3. H je konečná množina hran, kde $H \subseteq (S \times P) \cup (P \times S)$, nesmí spojovat dva přechody nebo dvě místa spolu.
4. $Z: S \to \mathbb{N}$ je úvodní rozložení značek, kde v každém stavu $s \in S$ je $n \in \mathbb{N}$ značek.
5. $V: H \to \mathbb{N}^+$ je množina vážených hran, kde každé hraně $h \in H$ náleží váha $v \in \mathbb{N}^+$
6. $K: S \to \mathbb{N}^+$ je množina kapacit stavů, kde každému stavu $s \in S$ náleží kapacita$k \in \mathbb{N}^+ \cup \{\infty\}$

2.1.3 P/T Petriho sítě s inhibiční hranou

P/T Petriho sítě s inhibiční hranou se liší od P/T sítí z předchozí kapitoly liší přidáním nového typu hrany. Tímto typem hrany je inhibiční. Inhibiční hrana může směřovat pouze ze stavu do přechodu nikoli naopak. Graficky se inhibiční hrana od ostatních liší záměnou šipky na konci za kolečko. Na

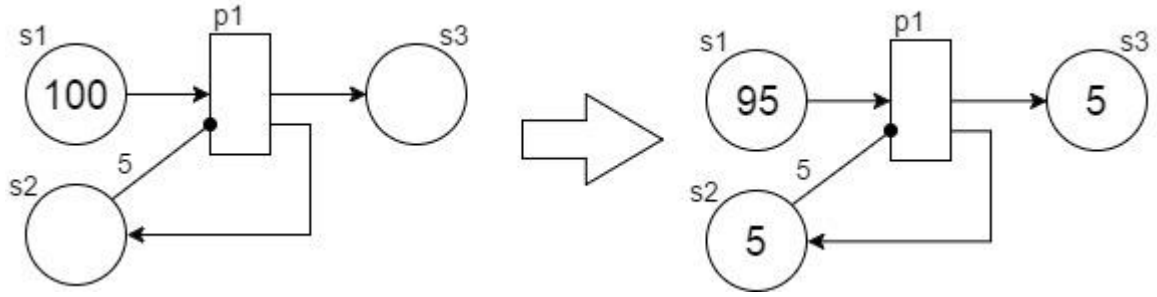
rozdíl od běžných typů hran se po inhibiční hraně nepřesouvají značky. Inhibiční hrana vytváří další podmínku proveditelnosti přechodu. Přechod s inhibiční hranou je proveditelný, pokud splňuje dvě následující podmínky: [1]

1. Přechod je uskutečnitelný v standardních P/T sítí.
2. Každé místo spojené s přechodem pomocí inhibiční hrany musí obsahovat méně žetonů, než činí váhové ohodnocení inhibiční hrany.



Obrázek 2.8: P/T síť s inhibitorem

Příklad inhibiční hrany je na obrázku 2.8, který ilustruje provedení přechodu *p1*. Inhibiční hrana na obrázku spojuje stav *s2* s přechodem *p1*. Levá strana obrázku představuje síť před provedením přechodu a pravá po provedení. Z obrázku vyplývá, že po uskutečnění přechodu *p1* nedojde k změně značení ve stavu *s2*. Zbytek sítě reaguje jako standardní P/T Petriho síť.



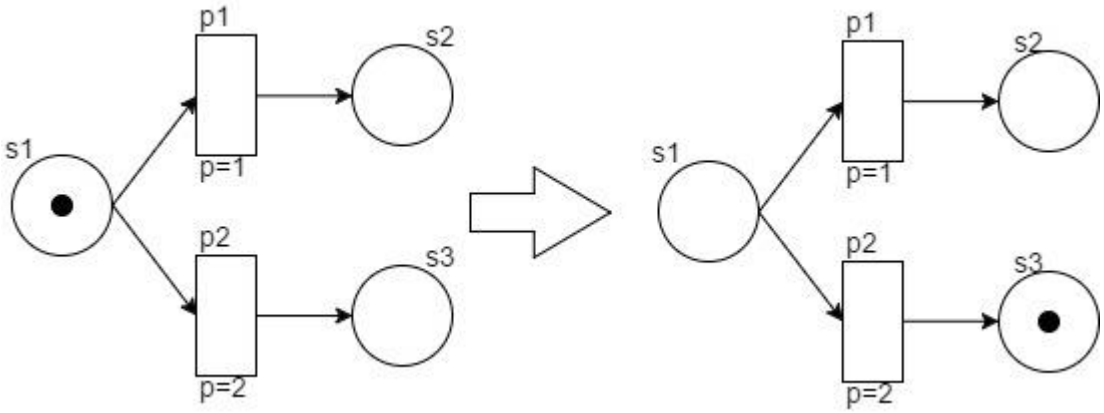
Obrázek 2.9: P/T síť s inhibitorem 2

Přidáním inhibičních hran do P/T Petriho sítí je podstatně navýšena modelovací síla. Obrázek 2.9 popisuje využití inhibiční hrany pro dávkování žetonů po pěti do stavu *s3*. Pravá strana obrázku popisuje stav modelu po první dávce žetonů. Pro uvolnění další dávky stačí odvést pět žetonů ze stavu *s2* pomocí jiného přechodu.

2.1.4 P/T Petriho sítě s prioritami

P/T Petriho sítě lze rozšířit o prioritu přechodů. Rozšířením P/T Petriho sítě o prioritu se zvyšuje její modelovací síla stejně jako u P/T Petriho sítě s inhibičními hranami. [2] Priorita pomáhá odstranit konflikty přechodů. Ke konfliktům dochází při rozhodování, když z jednoho místa vedou hrany do více přechodů, přičemž všechny přechody jsou proveditelný a model se musí rozhodnout, který přechod provede. Pro toto rozhodnutí jsou zavedeny priority, které mohou jednoznačně určit, který

přechod má byt uskutečněn. Priorita je graficky reprezentována výrazem $p =$ a celým nezáporným číslem.



Obrázek 2.10: P/T síť s prioritou

Velikost priority závisí na zadaném čísle. Čím větší číslo, tím roste i priorita daného přechodu. [5] Obrázek 2.10 znázorňuje konflikt dvou přechodů *p1* a *p2*. Přechod *p1* má zadanou prioritu jedna a přechod *p2* má prioritu dva. Pokud by nebyla priorita explicitně zadána, pak je výchozí hodnotou číslo nula. Kdyby v příkladu z obrázku 2.10 nebyla použita priorita nebo se hodnota priority shodovala, pak by byl proveden náhodně jeden z přechodů. Ale v tomto případě je směrodatná priorita, která zaručuje uskutečnění přechodu *p2*. Provedení přechodu *p2* se nijak neliší od provedení ve standardních Petriho sítí. Konečný stav sítě znázorňuje pravá strana obrázku. Z tohoto příkladu vyplývá, že priorita mění pouze podmínky uskutečnění přechodů. V P/T Petriho sítí s prioritou je přechod proveditelný, pokud splňuje následující dvě podmínky: [2]

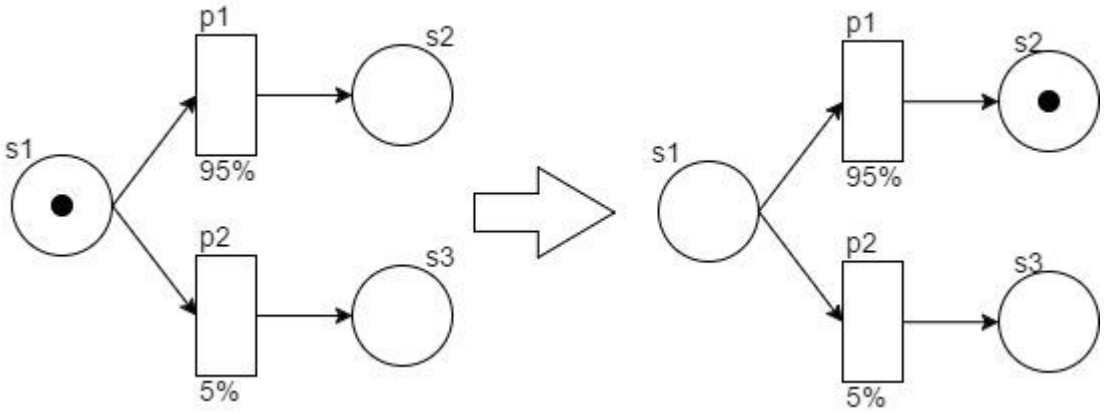
1. Přechod je proveditelný v standardních P/T Petriho sítí.

2. Ostatní proveditelné přechody, které jsou spojeny s výchozím místem, mají nižší prioritu než výchozí přechod.

2.1.5 P/T Petriho sítě s pravděpodobností

P/T Petriho sítě s pravděpodobností jsou podobným rozšířením jako sítě s prioritou. I zde pravděpodobnost přidaná přechodům vytváří parametr, který pomáhá odstranit konflikt přechodů. V žádném případě nesmí jeden přechod obsahovat více než jeden parametr. Příkladem této chyby může být libovolný přechod, kterému je nastavena pravděpodobnost a priorita současně.[5] Grafická reprezentace pravděpodobnosti je znázorněna u přechodů pomocí procent.

Pravděpodobnost je vhodná pro použití v případech, kdy může dojít například k výrobě vadného kusu. Obrázek 2.11 lze chápat jako část výrobního cyklu, kde dochází k tvorbě produktu. Na začátku je stav *s1*, ve kterém se nachází materiál pro tvorbu jednoho produktu. Z 95 % případu dochází k úspěšnému vytvoření produktu. Tento úspěch reprezentuje provedení přechodu *p1*, kterému je přidělena 95 % pravděpodobnost provedení. Výsledný produkt je pak uložen v stavu *s2*. V opačném případě, kdy se vytvoření produktu nezdaří, je proveden přechod *p2*. Vadný kus je poté uložen ve stavu *s3*. Obrázek 2.11 ilustruje úspěšné vytvoření produktu.



Obrázek 2.11: P/T Petriho sítě s pravděpodobností

Pro korektní užití pravděpodobnosti je nutné správné použití procent. Při rozhodování, kdy model určuje přechod, který má být proveden, musí součet procent všech vstupních přechodu činit 100 %. V opačném případě by byla porušena základní vlastnost pravděpodobnosti.

2.1.6 Časované Petriho sítě

Do této kapitoly modely Petriho sítí pracovali v diskrétním čase. Časované Petriho sítě přináší možnost pracovat s časem. Práce s časem snižuje úroveň abstrakce a model se pak jeví člověku reálnější. Pro vhodnější analýzu simulace Petriho sítě je zavedením časování výhodou. Časové rozšíření může být spojeno s těmito prvky P/T Petriho sítí:

- Přechody – Nejčastěji se rozšíření Petriho sítí o čas týká přechodů. Přechody v P/T Petriho sítí reprezentují akci, která má být provedena, a proto je přirozené jim přiřadit dobu trvání. Časové omezení přechodu určuje dobu, po kterou musí být přechod proveditelný. Pokud se podmínky pro uskutečnění nezmění, je po uplynutí doby přechod proveden.

- Místa – Časové omezení spojené s místem ovlivňuje příchozí žetony. Žeton, který přišel do časovaného místa, musí zde setrvat do uplynutí časového limitu. Teprve poté lze žeton použít k dalšímu zpracování.

- Hrany – Časové omezení na hraně určuje dobu, po kterou žeton přechází. Toto časového omezení s hranami není příliš běžné.

- Žetony – Spojení časového omezení s žetony není také velmi obvyklé. Žetony sebou nesou časové razítko. Razítko udává dobu, po kterou žeton nelze použít. Tato doba je aktivována po provedení přechodu.

[6]

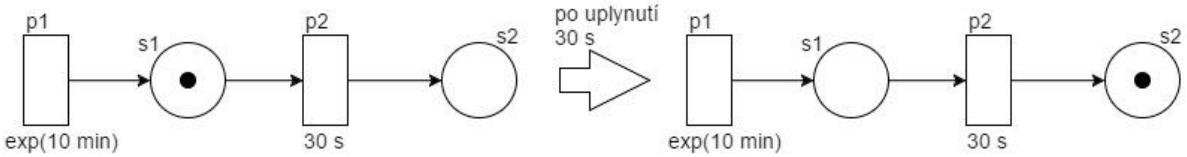
Časované Petriho sítě lze také dělit do tří skupin podle trvání dějů:

- Deterministické – Časové údaje v síti jsou konstanty.

- Stochastické – Časové údaje v síti jsou náhodné veličiny, nejčastěji s exponenciálním rozložením

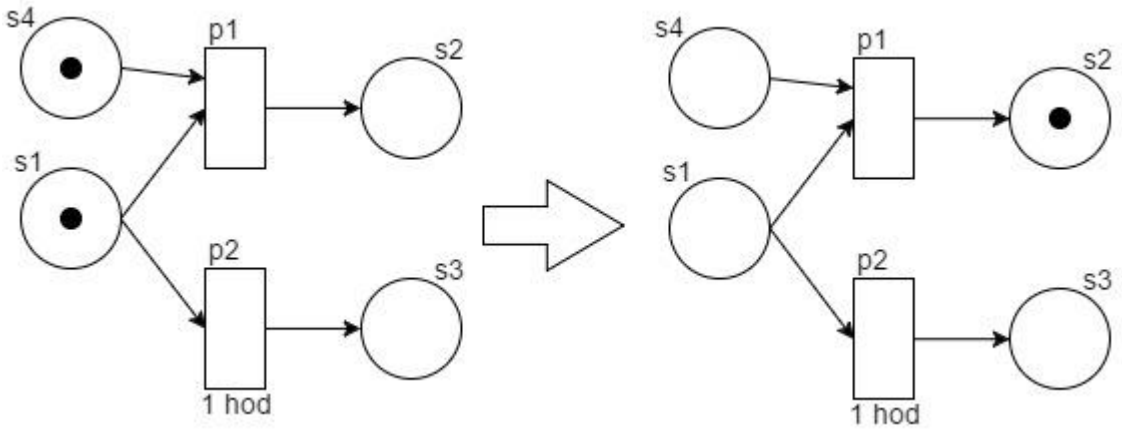
- Kombinované (Zobecněné stochastické Petriho sítě) – Časové údaje mohou být konstantní nebo náhodné. Jedná se o kombinaci deterministických a stochastických sítí.

[7]



Obrázek 2.12: Kombinované časové Petriho sítě

Příklad z obrázku 2.12 reprezentuje využití kombinovaného časového rozšíření pro P/T Petriho sítě. Příklad modeluje situaci, kdy zákazníci vstupují do pobočky banky. Zákazníci jsou reprezentování žetony. Levá strana obrázku představuje stav systému, kdy se v stavu *s1* ocitne první zákazník. Přechod *p1* reprezentuje vstup zákazníků do systému. Zákazníci vstupují do systému s exponenciálním rozložením deseti minut. Místo *s1* představuje stav zákazníka, který právě vstoupil do banky a na pořadníku si vybírá požadovanou službu. Výběr požadované služby trvá přibližně třicet vteřin. Tuto dobu výběru reprezentuje přechod *p2*. Po výběru služby se zákazník přesune do stavu *s2*. Tento stav může představovat setrvání zákazníka v čekárně. Pravá strana obrázku reprezentuje stav systému po uplynutí třiceti vteřin, kdy první zákazník postoupí do čekárny.



Obrázek 2.13: Kombinované časové Petriho sítě 2

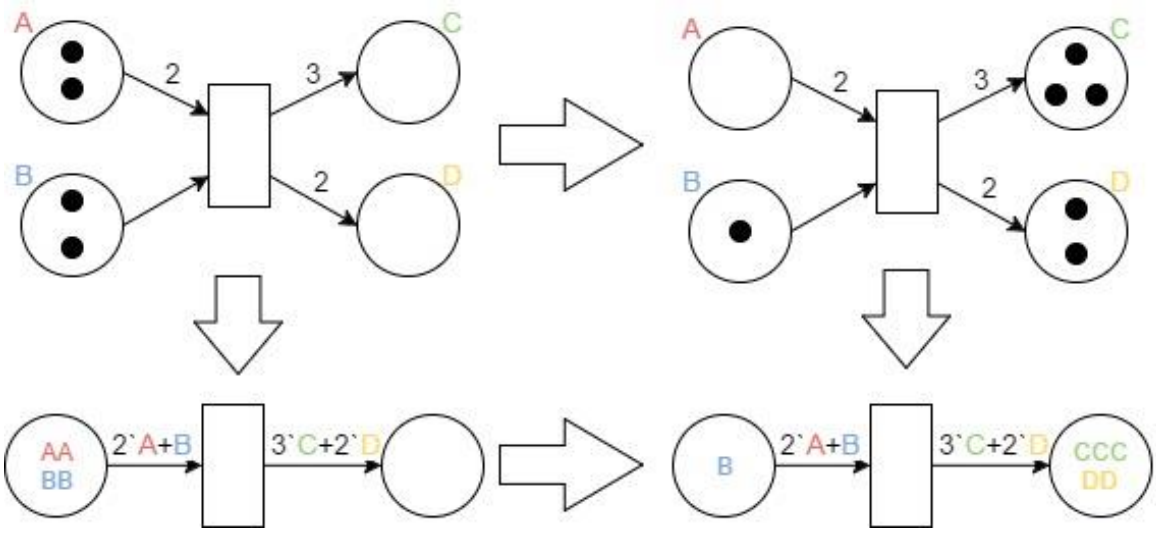
Obrázek 2.13 navazuje na předchozí příklad z obrázku 2.12. Lze vycházet z předpokladu, že místo *s2* z obrázku 2.12 je totožné s místem *s1* z obrázku 2.13. Zákazník tedy čeká ve stavu *s1* na zavolání k přepážce. Volné přepážky jsou zobrazeny ve stavu *s4*. Zároveň s čekáním na přepážku zákazníkovi ubývá trpělivost. Tato trpělivost je prezentována přechodem *p2*. Pokud by v průběhu jedné hodiny nebyla uvolněna ani jedna přepážka, pak je proveden přechod *p2*, který reprezentuje dojití trpělivosti zákazníkovi a přesunutí zákazníka do místa *s3*. Místo *s3* znázorňuje stav zákazníka opouštějícího pobočku banky. V příkladu z obrázku 2.13 je však přepážka volná a proto je proveden přechod *p1*. Z tohoto případu vyplývá, že okamžitě proveditelný přechody jsou upřednostněny oproti přechodům s časovým omezením.

2.1.7 Barevné Petriho sítě

Barevné Petriho sítě patří už mezi Petriho sítě vyšší úrovně. Petriho sítě vyšší úrovně obohacují strukturu modelů o další parametry. „Zavedení barev umožňuje někdy velmi výrazně stručnější zápis modelu. Na druhou stranu je ovšem použití CPN nutné zvažovat rovněž z hlediska možné analýzy a verifikace systémů pomocí nich popsaných. Analýza nad CPN, a to zejména formální analýza, je komplikovanější než nad klasickými P/T Petriho sítěmi.“ [8] Barevné Petriho sítě se od základních P/T Petriho sítí liší především těmito prvky:

- Žetony mohou být různého datového typu. V barevných Petriho sítí se pro rozlišení datových typů využívá různých barev.
- Každé místo obsahuje třídu žetonů (třídu barev). Třída žetonu určuje, které typy žetonů se v daném místě mohou nacházet.
- Přechody mohou obsahovat podmínku proveditelnosti (strážní podmínka). Podmínka je tvořena pomocí proměnných a konstant. Po vyhodnocení podmínka nabývá pravdivostních hodnot.
- Všechny hrany obsahují výraz. Tento hranový výraz je tvořen z proměnných a konstant. Tento výraz představuje množinu žetonů, které mají být z místa odebrány, pokud hrana vede z místa do přechodu. V opačném případě se jedná o množinu žetonů, které mají být do místa přivedeny.

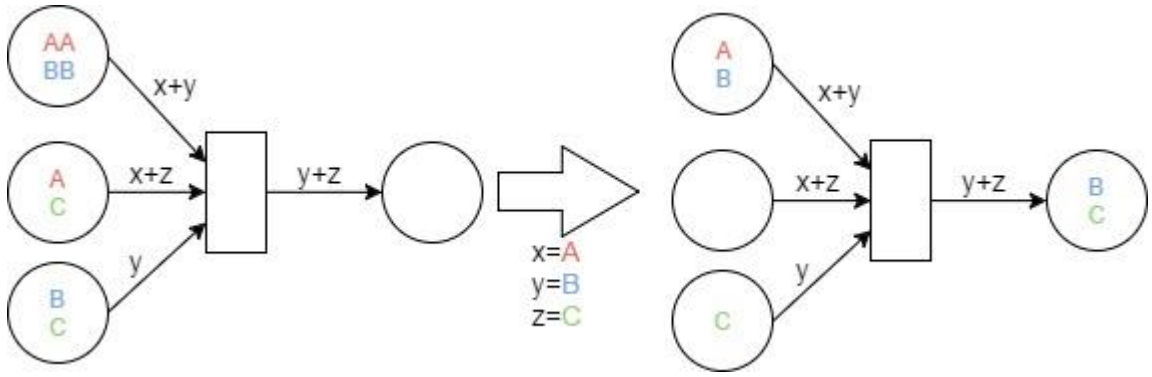
[2]



Obrázek 2.14: Barevné Petriho sítě

Příklad na obrázku 2.14 ilustruje rozdíl mezi základními P/T Petriho sítěmi a barevnými Petriho sítěmi. Horní část obrázku reprezentuje základní P/T Petriho sít a výsledek provedení jejího přechodu. Dolní část obrázku představuje barevné Petriho sítě. Obě sítě jsou ekvivalentní vzhledem k výsledku značení. Hlavním rozdílem těchto dvou síti je v úspoře stavů. Barevné Petriho sítě zredukovali počet stavů ze čtyř na dva. Tato úspora je způsobena zavedením datových typů žetonů.

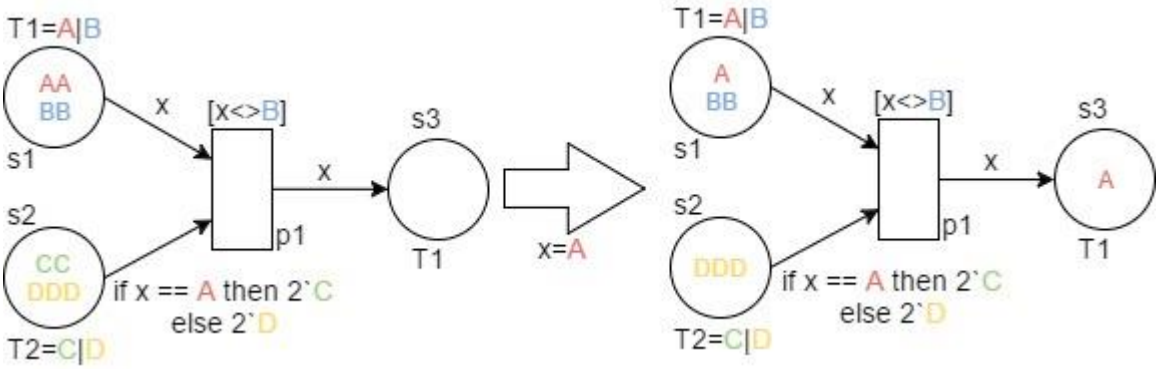
Během ověřování podmínek pro uskutečnění přechodu se na hranách mohou vyskytovat proměnné. Při jejich výskytu dochází k tzv. *navázání*, kdy dané proměnné je přiřazena určitá barva. Pokud se daná proměnná vyskytuje na více hranách spojené s totožným přechodem, musí na všech hranách být navázaná na stejnou barvu. [8]



Obrázek 2.15: Barevné Petriho sítě navázání

Příklad na obrázku 2.15 reprezentuje *navázání*. Proměnná *x* se navázala na barvu *A*, *y* na *B* a *z* na *C*. Finální stav sítě po provedení přechodu znázorňuje pravá strana.

V obou obrázcích barevných Petriho sítí doposud nebyla uvedena stráž přechodu a třída barev. Stráž přechodu bývá vyobrazena v hranatých závorkách. Třída barev je reprezentována identifikátorem zobrazeného poblíž místa. Pokud místu není explicitně místu přiřazena třída, pak místo může obsahovat všechny typy barev.



Obrázek 2.16: Barevné Petriho sítě se všemi prvky

Na obrázku 2.16 jsou definované dvě třídy *T1* a *T2*. Třída *T1* může obsahovat pouze barvy typu *A* nebo *B*. Třída je přiřazena stavům *s1* a *s3*. Třída *T2* je přiřazena pouze stavu *s2* a zahrnuje barvy typu *C* a *D*. Podmínka proveditelnosti přechodu je zobrazena nad ním v hranatých závorkách. Dalším speciálním prvkem v tomto případě je výraz hrany vedoucí ze stavu *s2* do přechodu *p1*. Tento výraz podmiňuje typ barvy, která má být užita pro uskutečnění přechodu *p1*. Přechod v barevných Petriho sítí lze uskutečnit, pokud jsou splněny následující dvě podmínky:

1. Každé vstupní místo přechodu musí obsahovat větší nebo rovnou multimnožinu[1], než je multimnožina, která je vyhodnocena pro danou hranu vedoucí z místa do přechodu.

[1] Multimnožina je zobecněním pojmu množina. Multimnožina umožnuje vícenásobný výskyt prvku v rámci jedné množiny.

2. Podmínka proveditelnosti přechodu (stráž přechodu) musí být splněna (nabývat logickou 1).

[2]

2.1.8 Hierarchické Petriho sítě

Hierarchické Petriho sítě jsou vhodným řešením pro modelování rozsáhlých systémů. Spolu se spojením s barevnými Petriho sítěmi patří k nejrozšířenějším Petriho sítím.[1][1] Dosavadní typy Petriho sítí neumožňovali propojení mezi jednotlivými modely. Hierarchické Petriho sítě nabízejí možnost členit síť na podsítě. Členění rozděluje model na sítě a jejich podsítě. Toto členění probíhá pomocí substituce. V hierarchických Petriho sítí lze substituovat místa a přechody.

Při substituci přechodu dochází k nahrazení přechody určitou podsítí. Pro realizaci substituce přechodu musí být splněny následující body:

- Ve výchozí síti je vybrán přechod, který bude substituován. Zároveň tento přechod je označen podle toho, jak se nazývá podsíť.[2]

- Všechny vstupní a výstupní místa substituovaného přechodu jsou označeny jako tzv. *porty*. Tyto porty zprostředkovávají komunikaci sítě s její podsítí.

- V podsíti jsou taktéž označený místa, které představují porty.

- Výchozí síť označíme jako tzv. *hlavní*. V hlavní síti začíná mechanismus substituce.

[8]

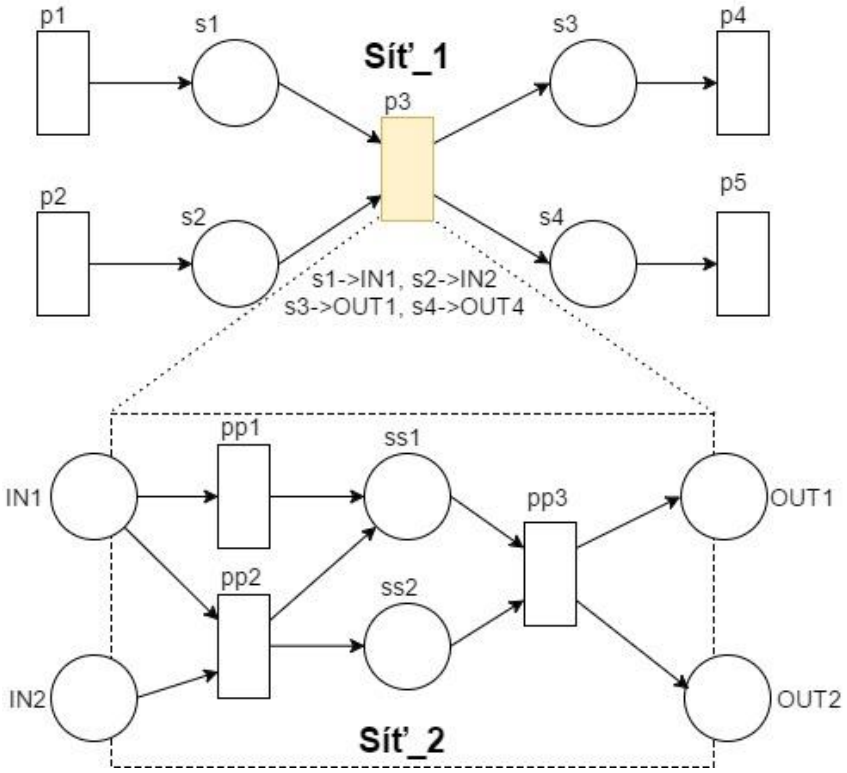
Substituce přechodu je demonstrována na obrázku 2.17. *Síť_2* se stává podsítí sítě *Síť_1*. Stavy *s1* a *s2* se stávají vstupními porty do podsítě a zároveň stavy *s3* a *s4* fungují jako výstupní porty podsítě.

Substituce míst je téměř identická se substitucí přechodu. Nahrazeným prvkem se zde stává místo. Vstupními a výstupními porty jsou přechody. Celý mechanismus probíhá obdobně, jak u substituce přechodu.

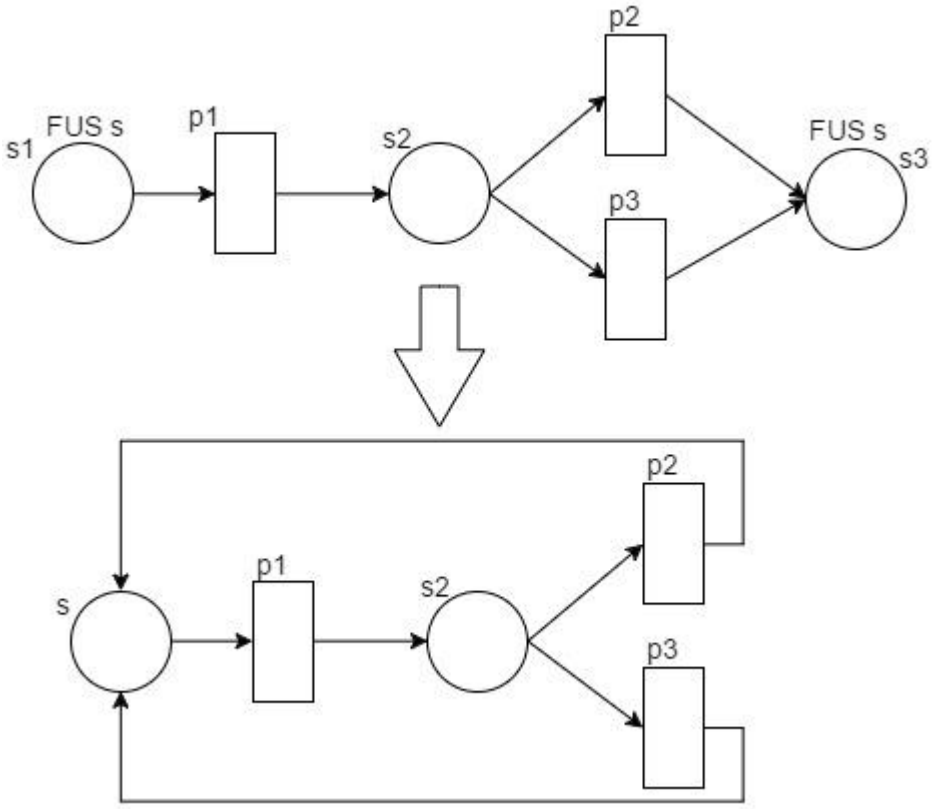
Další mechanismus, který hierarchické Petriho sítě umožňují, je fúze míst. Fúze dovoluje sloučit různá místa do jednoho. Tato výhoda může pomoct vytvořit přehlednější model obzvlášť, pokud model je velmi rozsáhlý. Graficky je fúze reprezentována stejným fúzním označením u všech míst, které mají být sloučeny.

Horní část obrázku 2.18 reprezentuje příklad fúze míst. Místa *s1* a *s3*, které se slučují, jsou označeny fúzním identifikátorem *FUS s*. Dolní část obrázku zobrazuje, jak by síť vypadala, pokud by nebyla použita fúze. Síť je obklopena hranami, a pokud by se měla rozšiřovat, tak by bylo těžké vyhnout se křížením hran. Právě toto křížení hran činí síť nepřehlednou.

[1] Spojení barevných a hierarchických Petriho sítí není součásti kapitoly 2.1.8.
[2] V hierarchických Petriho sítí se pracuje s více sítěmi najednou, proto v nich každá sít má své identifikační název.



Obrázek 2.17: Hierarchické Petriho sítě se substitucí přechodu



Obrázek 2.18: Hierarchické Petriho sítě s fúzí místa

2.1.9 Objektově orientované Petriho sítě

Důvod pro použití objektově orientovaných Petriho sítí je kombinace synchronizačních prostředků a objektové struktury. Tato kombinace poskytuje výhody objektové orientace jako je například dědičnost. Pro tvorbu Objektově orientovaných Petriho sítí existuje více konceptů, podle kterých lze modely vytvářet. Tato kapitola se zabývá objektově orientovanými Petriho sítěmi spojené s jazykem *PNtalk*[1] vyvinutého na Fakultě informačních technologii Vysokého učení technického v Brně. [8]

Stejně jak u jiných OO programovacích jazyků objekty jsou instancí tříd. Třídy jsou zde reprezentovány Petriho sítěmi. Síť je složena především z míst, přechodů a hran. Tyto prvky jsou oproti ostatním Petriho sítím rozšířeny o speciální parametry. Místa jsou standardně zobrazena pomocí kružnic nebo elips a rozlišena jednoznačným identifikátorem. Místa mohou obsahovat značky. Značení v OO Petriho sítí může být reprezentováno například pomocí čísla, symbolu nebo řetězce. Počáteční značení míst může obsahovat rovněž proměnné. Pro přiřazení hodnoty do proměnné se využívá *počáteční akce* místa. Počáteční akce místa je syntakticky shodná s akcí přechodu. V případě výskytu počáteční akce je místo rozděleno na dvě části. Horní část obsahuje značení a dolní počáteční akci místa. [6]

Přechod je graficky reprezentován čtvercem nebo obdélníkem. Přechody jsou pojmenovány jednoznačnými identifikátory. Stejně jak u barevných Petriho síti i zde přechod může obsahovat *stráž* (podmínka proveditelnosti). Stráž přechodu může být složena z více výrazů. Podmínka stráže je splněna, pokud všechny dílčí výrazy jsou rovněž splněny. Další parametr, který je součástí přechodu, je *akce*. Akce přechodu může obsahovat výraz přiřazení. [6] Výraz přiřazení je formulován například takto $A := B * C$.

Místa a přechody jsou spojeny pomocí hran. Hrana obsahuje tzv. *hranový výraz*. [6] Stejně jak váha hrany u standardních P/T Petriho sítí určuje, kolik žetonů po ní musí přejít, tak obdobně hranový výraz určuje nejen kolik, ale také co po hraně přechází. Hranový výraz se skládá z multimnožin. Příkladem hranového výrazu je $3`\#e$. Tento příklad definuje multimnožinu, ve které se třikrát vyskytuje symbol $\#e$. V jazyku PNtalk se využívá symbol $\#e$ jako náhražka, která v standardních P/T Petriho sítí je reprezentována žetonem. Podobně lze využít i literál *nil*, který je reprezentován symbolem •.

Hrany lze dělit do tří typů podle jejich orientace. První typem je vstupní hrana. Tato hrana směřuje z místa do přechodu. Opak jí je výstupní hrana, která směřuje z přechodu do místa. Poslední typem hrany je testovací. Testovací hrana ve své podstatně nepřesouvá značení z nebo do místa. Hrana pouze testuje přítomnost značení v připojeném místě.

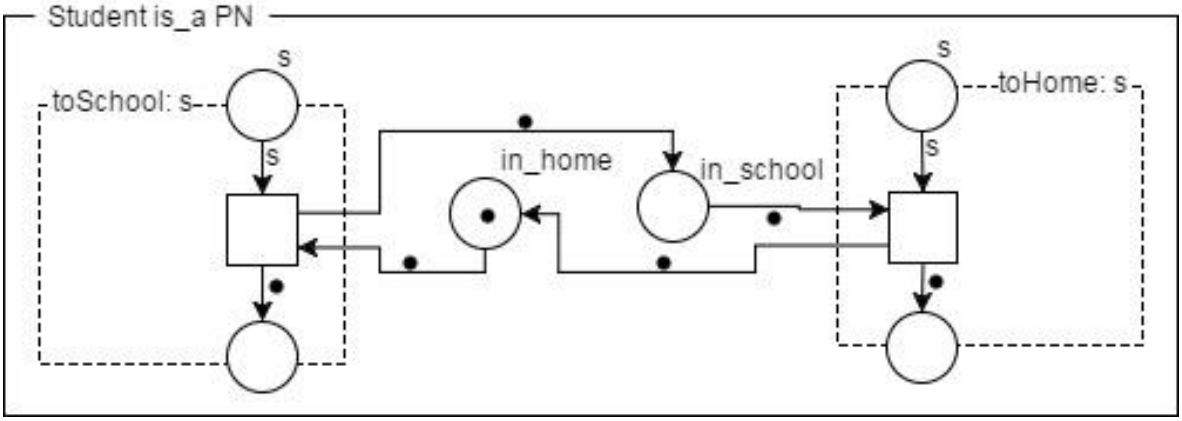
Jazyk PNtalk rozděluje sítě na dva typy. První typem je *síť objektu*. Tento typ představuje atributy objektu. Druhým typem je *síť metody*. Stejně jako síť objektu, tak i síť metody se skládá z míst, přechodů a hran. Každá síť metod vlastní tzv. *vzor zprávy*, který po zavolání slouží k vytvoření instance metody. Metodu lze vyvolat pouze objektem. Každá síť metod obsahuje výstupní místo nazvané *return*, které vrací výsledek metody. Síť metody může obsahovat množinu míst, které pomáhají předat parametry pro metodu při jejím vyvolání. Síť objektů může ovlivňovat síť metody pomocí hran propojených mezi místy objektové sítě a přechody sítě metody. Těmito hranami síť objektů může získávat data metody a zároveň je ovlivňovat. [6]

Speciálním prvkem třídy je *synchronní port*. V rámci třídy je na stejné úrovni jako je síť metod, ale o síť se nejedná, protože se neskládá z míst a přechodu. Charakteristicky je synchronní port podobný přechodu, protože pomocí hran je propojen s místy sítě objektu. Taktéž synchronní port může obsahovat stráž. Synchronní port obsahuje vzor zprávy podobně jako síť metody. Synchronní port může být volán pouze ze stráže přechodu. Synchronní port může být použit pro testování nebo

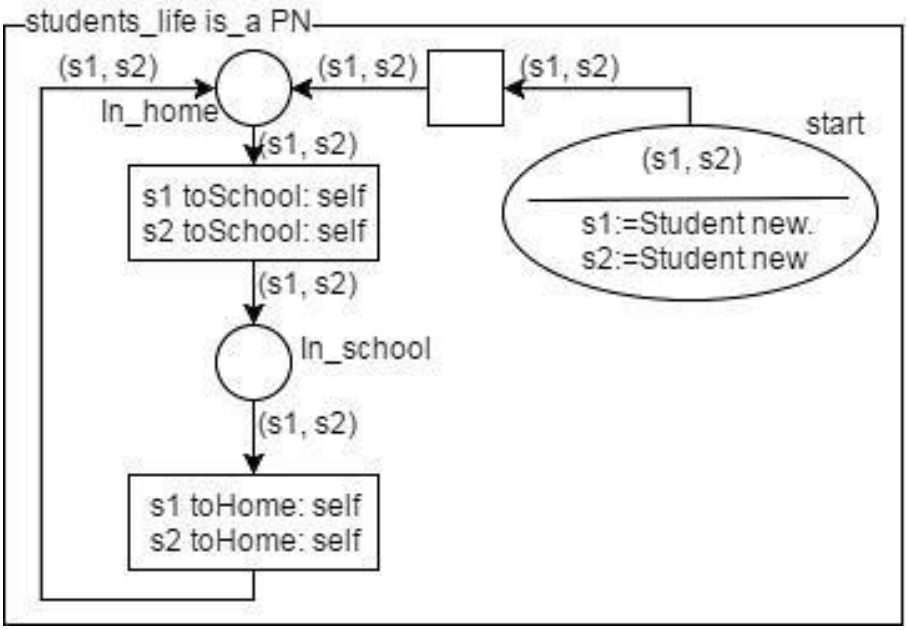
[1] Jazyk PNtalk je konkrétní implementací OO Petriho sítí. Jeho syntaxe vychází z jazyku Smalltalk.

ovlivňování sítě objektu. Pokud je synchronní port spojen pouze pomocí testovacích hran, pak se jedná o speciální případ. Tento případ synchronní portu se nazývá *predikát*, který neovlivňuje stav objektové sítě. [6]

Samotný model v jazyce PNtalk je složen z minimálně jedné třídy. Každý model musí určovat *počáteční třídu*, jejíž instance je provedena automaticky. V počáteční třídě model začíná výpočet modelu. Hierarchie tříd začíná nejvyšší nadtřídy zvané *PN*. Tato třída už není popsána Petriho sítí. Každá vytvořená třída může mít následníka jinou vytvořenou třídu nebo třídu *PN*. Zavedení tříd do OO Petriho sítí zpřístupňuje využití dědičnosti, která funguje stejně jako u ostatních OO programovacích jazyků.



Obrázek 2.19: Třída *Student*



Obrázek 2.20: Třída *students_life*

Příklad modelu OO Petriho sítí s jazykem PNtalk je zobrazen pomocí dvou tříd na obrázcích 2.19 a 2.20. Na obrázku je vyobrazena třída *Student*. Tato třída se skládá ze sítě objektu a dvou sítí metod. Síť objektu obsahuje pouze dvě místa (stavy) *in_home* a *in_school*. Metody jsou pojmenované *toSchool* a *toHome*. Voláním metod dochází k změně stavu sítě objektu. Druhá třída *students_life* obsahuje dvě instance třídy *Student* v proměnných *s1* a *s2*. Instance probíhá v počáteční akci místa

start. Poté jsou oba studenti přesunuti do cyklu, který představuje studentský život. V akcích přechodu dochází k volání metod třídy student, které mění stav studentů.

3 Analýza projektu

Analýza projektu proběhla v rámci přípravy před začátkem vývoje grafického editoru objektově orientovaných Petriho sítí. Analýza byla provedena v rámci prevence nejasností a chyb. Provedení analýzy bylo rozděleno do dvou úrovní. První úroveň zkoumala požadavky na funkcionalitu. Jednalo se o teoretické pochopení všech aspektů projektu. Druhá úroveň řešila nejvhodnější vývojový postup. I přes poměrně komplexní analýzu projektu došlo k zjištění nedostatků v projektu, které museli být v průběhu vývoje opraveny.

3.1 Analýza požadavků

Pro komplexní zpracování všech parametrů aplikace byla provedena analýza požadavků. Z velké části požadavky určují pravidla struktury OO Petriho sítí. OO Petriho sítě určují pouze funkcionální požadavky, které by měla aplikace zvládat. Jelikož OO Petriho sítě jsou v současné době poměrně abstraktní typ, musí dojít k rozšíření analýzy o další prvek, který by upřesňoval požadavky na funkcionalitu. Tímto prvkem je jazyk PNtalk. Jazyk PNtalk konkrétně určuje veškerá pravidla modelu, která platí u jeho vytváření a spouštění. Protože aplikace řeší pouze vytváření a zobrazení modelu, analýza se zaměřila na pravidla určující tvorbu modelu.

3.2 Analýza vývoje aplikace

Pro rychlý a korektní vývoj aplikace byla provedena analýza. Ta měla mimo jiné určit, pomocí jakého programovacího jazyku bude aplikace vytvářena. Jelikož se jedná graficky náročný projekt, byl vybrán jazyk Java. Zároveň byla provedena analýza pro výběr vhodného vývojového prostředí. Mezi vhodnými IDE[1] byli NetBeans a Eclipse. Konečné rozhodnutí určilo IDE NetBeans jako nejvhodnější rozhraní pro vývoj aplikace. Hlavní důvod pro výběr tohoto rozhraní byla jednoduchost manipulace s grafickou reprezentací projektu.

Posledním bodem analýzy bylo vybrat ideální model životního cyklu softwaru. Pro tento účel byl vybrán spirálový model. [9] Bylo tak učiněno z důvodu postupného vývoje aplikace, kdy bylo požadováno rychlé vytvoření prototypu. Po vytvoření prototypu, který byl později schválen jako vyhovující, byla provedena další analýza. Nová analýza se zaměřila už pouze na funkcionalitu, kterou aplikace musí splňovat. Prototyp splnil svůj účel a nová analýza měla za účel vytvořit postup jeho rozšíření.

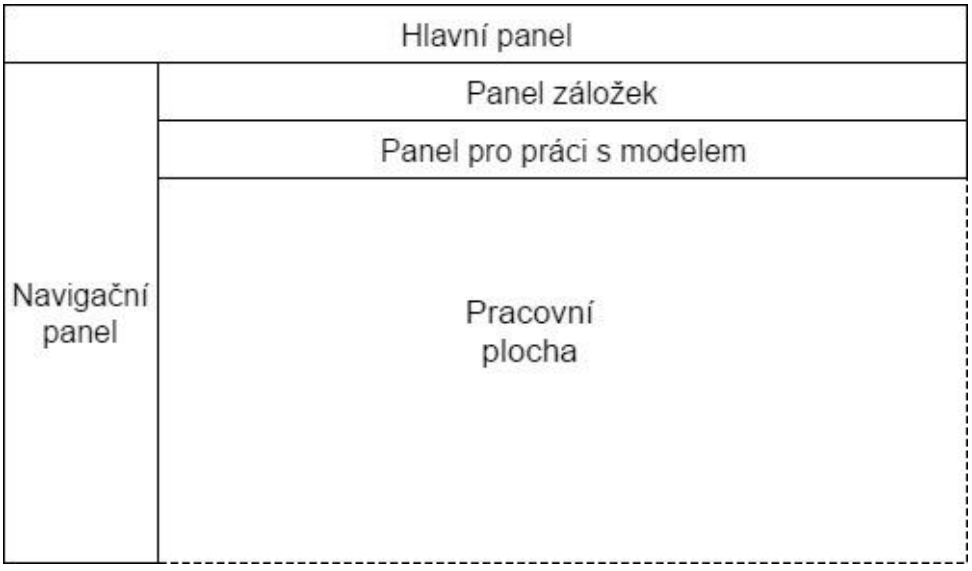
[1] IDE (Integrated Development Environment) je softwarové vývojové prostředí.

4 Návrh editoru

Návrh aplikace se zabýval především určení vhodné architektury aplikace. Hlavními požadavky na návrh bylo vytvoření takového grafického uživatelského rozhraní, které bude pro uživatele přehledné a srozumitelné. V tomto případě se jedná především o organizaci jednotlivých prvků systému, tak aby splňovali výše uvedené požadavky. Architektura se zabývala spojením požadavků s návrhem, tak aby se nevylučovali. Backend[1] aplikace nebyl součástí návrhu aplikace. Návrh vývoje backendu aplikace zůstal nepopsaný a k jeho tvorbě docházelo současně s grafickým rozhraním a tím pádem i neorganizovaně.

4.1 Návrhové vzory

Před samotnou tvorbou návrhu došlo k prozkoumání různých editorů, které pracují s Petriho sítěmi. První takovým editorem je PIPE2[2]. Tento editor stochastických Petriho síti dovoluje uživatelům velmi přehledné grafické uživatelské rozhraní. Touto důležitou vlastností se stal inspirací pro tvorbu návrhu grafického uživatelského rozhraní. Hlavní částí PIPE2 editoru, kterou se návrh snažil napodobit, je navigační panel. Navigační panel PIPE2 je velmi přehledný a je jeho klíčovou součástí. Pro návrh bylo tedy nezbytné vytvořit vhodnou podobu tohoto navigačního panelu, který bude pro uživatele vyhovujícím. Druhým editorem, který se stal inspirací pro návrh, je Petri Net Editor. Tento editor dovoluje jednoduchou práci s modely. Především jejich tvorba je přirozená. Kromě těchto vlastností, kterých se stali součástí návrhu aplikace, jsou spouštěcí prvky poměrně skromné, ale efektivní. Spouštěcí prvky se stali také součástí návrhu aplikace jako vhodná předloha. Obrázek 4.1 popisuje grafické uživatelské rozhraní před začátkem vývoje aplikace.

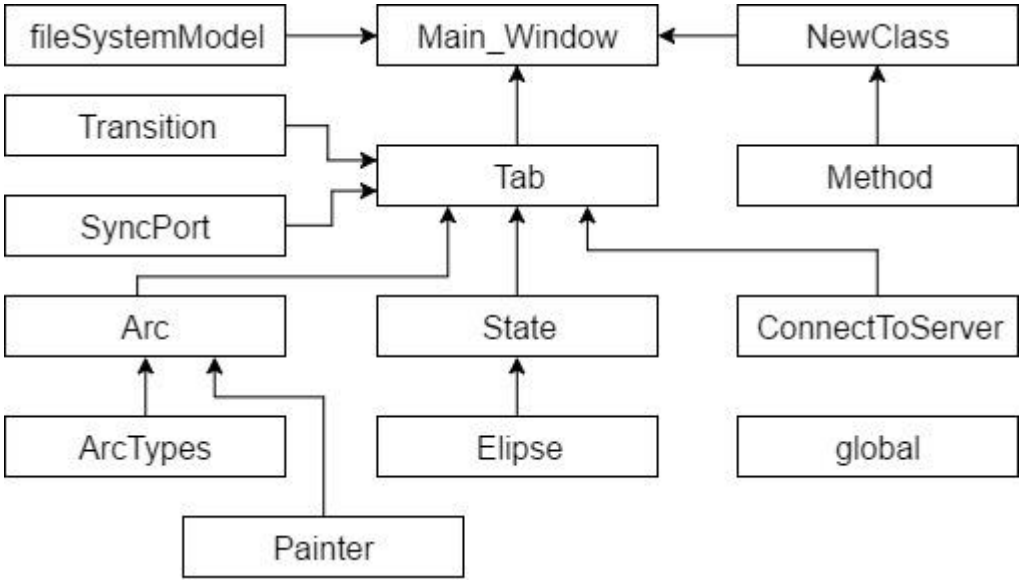


Obrázek 4.1: Návrh grafického uživatelského rozhraní

[1] Backend je vnitřní architektura aplikace se kterou se uživatel nedostane do styku.
[2] PIPE2 (Platform Independent Petri net Editor 2) je editor pro tvorbu stochastických Petriho sítí, který vyvinuli N. J. Dingle a W. J. Knottenbelt.

5 Implementace editoru

Implementace editoru vychází z návrhu popsaného v předchozí kapitole. Programování probíhalo v objektově orientovaném jazyce Java. Pro vývoj editoru bylo určeno programovací rozhraní *NetBeans*. Jelikož se jedná o OO programovací jazyk, je editor členěn do tříd. Hierarchie těchto tříd tvoří strukturu editoru. Kromě jednotlivých tříd je součástí také adresářová struktura, ve které jsou uložené modely. Obrázek 5.1 popisuje strukturu editoru. Jednotlivé prvky v obrázku reprezentují třídy. Spolu s navázáním jednotlivých tříd tvoří hierarchii editoru.



Obrázek 5.1: Hierarchie tříd implementace

5.1 Třída Main_Window

Třída *Main_Window* je tzv. *hlavní třída*[1]. Tato třída vytváří grafické uživatelské rozhraní, které se po celou dobu běhu programu nemění. Třída se od ostatních liší především svým typem. Jedná se o typ třídy *JFrame*, která vytváří okno programu se záhlavím a okraji.[11] Třída pomáhá zobrazit tři hlavní části programu. První částí je hlavní menu, které je zobrazeno na obrázku 5.2 v levém horním rohu. Hlavní menu obsahuje základní operace, které uživatel používá k řízení aplikace. Těmito operaci může být například vytváření nových tříd nebo metod. Druhým prvkem je zobrazovací prostor pro modely. Zde třída vytváří prostředí pro přepínání mezi jednotlivými modely pomocí záložek. Toto prostředí zobrazuje obrázek 5.2. Poslední částí je navigační panel. Tento panel zobrazuje strukturu sloužící k orientaci mezi modely. Zobrazení této struktury je delegováno do třídy *fileSystemModel*.

Třída *fileSystemModel* zobrazuje všechny dostupné modely v adresářové struktuře. Příklad této struktury je zobrazen na obrázku 5.2 pod nápisem *Navigate*. Struktura dovoluje uživateli vybírat modely, které mají být otevřeny. Pomocí této adresářové struktury si třída Main_Window vytváří ekvivalentní strukturu, která slouží pro vnitřní zpracování programu. Tato ekvivalentní struktura je

[1] Hlavní třída je počáteční třída, ve které se nachází metoda *main*, kde začíná výpočet programu.[10]

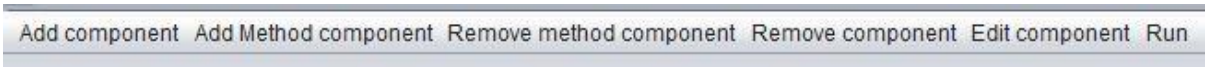
vytvořena pomocí tříd *NewClass* a *Method*, které si zachovávají informace o modelech především o jejich umístění ve struktuře. Poslední třída, která navazuje na hlavní, je *Tab*.

Obrázek 5.2: Programové okno

5.2 Třída Tab

Třída *Tab* (záložka) se dá považovat za stěžejní třídu. Zobrazení této třídy jde rozdělit na hlavičku a tělo. Tělo obsahuje prostor pro zobrazení sítě. Zobrazuje se vždy pouze jedna síť a to podle výběru mezi záložkami, jak je tomu na obrázku 5.2, kde vybranou záložkou je *Starting tab*. Hlavička obsahuje operace pro práci s aktuální sítí. Těmito operacemi rozumíme například přidávání a odebírání různých prvků Petriho sítí. Operace jsou situovány do několika nabídek podle jejich efektu na prvky. Existují dvě varianty této hlavičky, pro objektovou síť a pro metodu. Je to z důvodu provázání metod s objektovou sítí, když nejsou zobrazeny v jedné síti. První varianta pro objektovou síť je znázorněna v obrázku 5.3 a varianta pro metodu v obrázku 5.4.

Obrázek 5.3: Ovládací panel pro objektovou síť



Obrázek 5.4: Ovládací panel pro metodu

Na třídu Tab navazuje dalších pět tříd. Třídy *Transition, SyncPort, Arc* a *State* představují jednotlivé prvky, které se v sítích mohou vyskytovat. Výjimkou je třída *ConnectToServer*, která zprostředkovává komunikaci se serverem pro překlad.

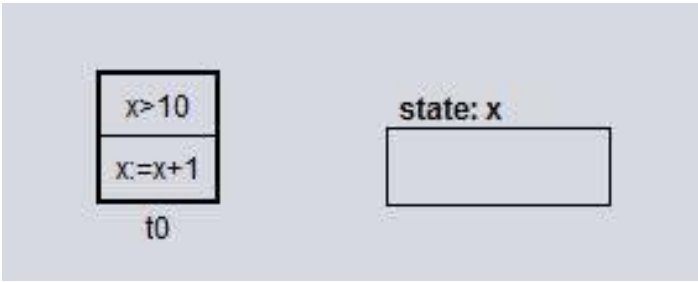
5.3 Třída State

Třída *State* reprezentuje prvek stav (místo). Stejně jak u P/T Petriho sítí je prvek místo graficky znázorněn jako kruh nebo elipsa. V případě tohoto editoru se jedná o elipsu. Tato elipsa může být půlená, pokud stav kromě značení obsahuje i počáteční akci. V případě najetí kurzoru na stav dojde k jeho obarvení obrysů na červenou barvu. V rámci přetížení aplikace je počet stavů v síti omezen na maximální počet sto. Se stavem lze hýbat pomocí kurzoru. Kliknutím a podržením levého tlačítka myše lze přesunout stav na jiného místo. Obrázek 5.5 reprezentuje zobrazení stavu v editoru. Stav *s0* neobsahuje počáteční akci a proto není půlen na rozdíl od stavu *s1*, který reprezentuje obarvení obrysů na červenou barvu. Pro zobrazení elipsy je použita třída *Elipse*. V této třídě jsou použity grafické nástroje pro vykreslení objektu.

Obrázek 5.5: Stavy v editoru

5.4 Třídy Transition a SyncPort

Třída *Transition* reprezentuje prvek přechod. Grafické znázornění prvku je pomocí obdélníku. Prvek se skládá ze dvou menších obdélníku vertikálně orientovaných. Horní obdélník obsahuje stráž přechodu. Dolní část obsahuje akci přechodu. Stejně jak je tomu u stavu, při najetí kurzorem na přechod dojde k jeho obarvení na červenou barvu. Rovněž je maximální počet přechodů omezen na sto. Pro zobrazení přechodu není použita žádná další třída. Obrázek 5.6 ilustruje přechod *t0* v editoru. Přechod obsahuje stráž $x > 10$ a akci $x := x + 1$.

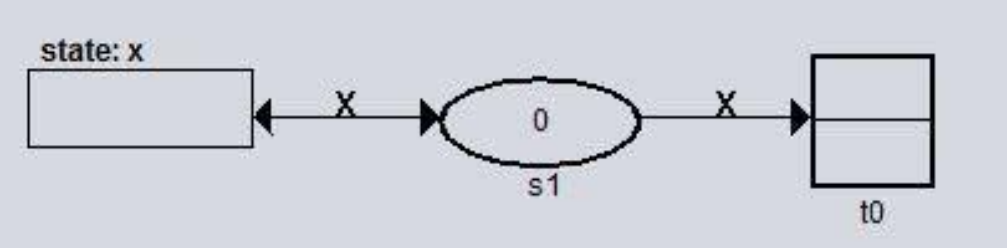


Obrázek 5.6: Přechod a synchronní port v editoru

Třída *SyncPort* reprezentuje synchronní port. Grafické znázornění je taktéž pomocí obdélníku. Na rozdíl od přechodu tělo synchronního portu složeno pouze z jednoho obdélníku. Zároveň vzor zprávy je zobrazen nad synchronním portem. Synchronní port může obsahovat stráž. V případě na obrázku 5.6 synchronní port *state: x* neobsahuje stráž. Pokud by obsahoval, byla by stráž znázorněna uvnitř obdélníku.

5.5 Třída Arc

Třída *Arc* zastupuje prvek hranu. Hrany nemohou existovat samostatně. Vždy jsou vázány ke dvěma rozdílným prvkům. Graficky je hrana reprezentována šipkou. Jelikož je třída Arc koncipována tak, aby si pamatovala zdroj, odkud šipka vychází a cíl kam přichází, tvoří různé typy hran, podle kombinace spojených prvků. Pro toto rozdělení do různých typů je zavedena třída *ArcTypes*. Tato třída obsahuje všechny možné typy kombinací, které hrany mohou nabývat. Pro grafické vyobrazení hrany je využit speciální grafický nastroj, který je obsažen v navazující třídě *Painter*. Třída Painter kromě vykreslení hrany, vypisuje rovněž hranový výraz, který je vždy vyobrazen přibližně v polovině délky hrany. Obrázek 5.7 graficky znázorňuje použití hran v přechodu. Oba přechody na obrázku vlastní hranový výraz *x*.



Obrázek 5.7: Hrany v editoru

5.6 Třída ConnectToServer

Tato třída navazuje spojení se serverem pro překlad a simulaci. Tento server se nazývá *PNtalk.v2.1*, který vznikl na Fakultě informačních technologii Vysokého učení technického v Brně. Tento server není součástí implementace editoru. Server komunikuje přes IP adrese tzv. *lokálního hosta*[1] a port 9999. Pro správnou komunikaci musí být splněna sekvence kroků. Třída ConnectToServer zařizuje tyto sekvenční kroky kromě posledního.

Prvním krokem je vytvoření vstupního textu pro server. Třída projde všechny prvky objektové sítě a z nich vytvoří textovou formu vstupní kód v jazyce PNtalk. Třída také projde všechny metody, které jsou součástí dané třídy Petriho sítě a z nich opět vytvoří vstupní kód. Druhým krokem je nahrání tohoto vstupního kódu na server. Pokud je server připravený pro práci, nahrání modelu na server by mělo proběhnout úspěšně. Třetím krokem je spuštění simulace na serveru. Zde již může dojít k zachycení chyby v modelu ze strany serveru. Pokud server spustí simulaci úspěšně, lze si z něho zažádat o zaslání aktuálního stavu sítě. Posledním krokem je získání výstupního kódu ze serveru a zpracováním ho. Po zpracování výstupního kódu dojde ke změně značení v síti.

5.7 Ostatní třídy a doplňující funkce

Výše uvedené třídy se dají považovat za velmi důležité. Pak existují další třídy, které se dají považovat jako doplňující. Třída *global* se mezi ně počítá. Tato třída uchovává globální proměnné, které ostatní třídy používají. Mezi další třídy patří všechny ze složky pojmenované *virtual*. Všechny třídy obsažené v této složce slouží pouze k dočasnému uložení dat pro jejich zpracování. Pro

[1] Lokální host je odkaz na speciální IP adresu 127.0.0.1.

usnadnění některých operací jsou v editoru implementovány klávesové zkratky. V průběhu práce s modely se uživatel setká s tzv. *modálními okny*[1], které usnadňují práci s editorem.

Editor obsahuje tři typy modálních oken. Prvním typem je informační okno, které podává informace uživateli o tom, jak dále postupovat. Druhým typem je dialogové okno, kdy editor čeká na zadání textu od uživatele. Posledním typem je chybové okno, které upozorňuje na chybu v editoru a také upřesňuje, o kterou chybu se jedná.

Ukládání modelů probíhá formou uložení potřebných dat v textových souborech. Data v textových souborech nejsou zabezpečena a je doporučeno do nich nezasahovat v rámci zachování jejich konzistence.

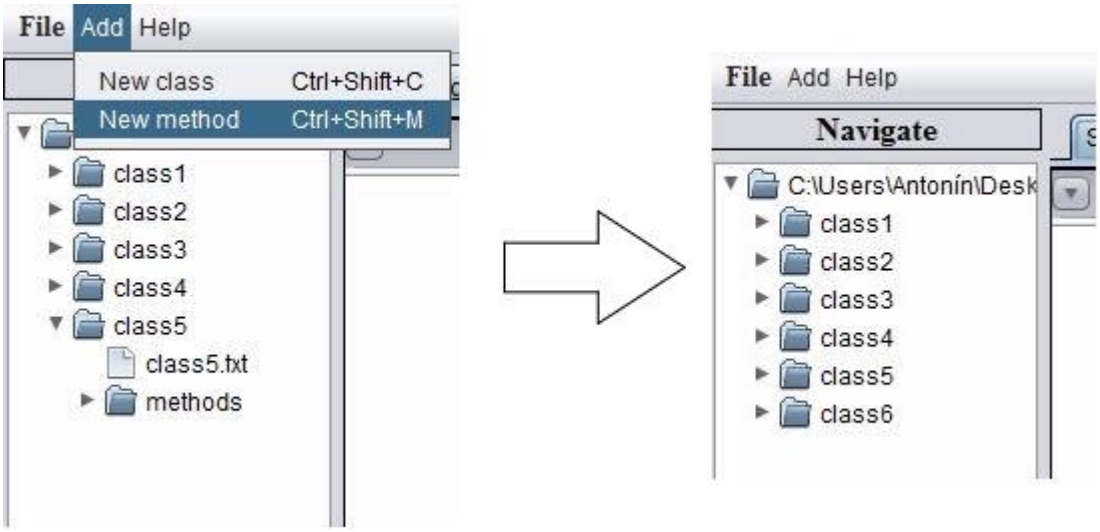
[1] Modální okno je ovládací prvek, který se používá v grafickém uživatelském rozhraní. Pod pojmem modální okno si lze představit vyskakovací okno, které například ohlašuje chybu aplikace.

6 Příklady použití editoru

Tato kapitola popisuje příklady modelů, které jsou vytvořeny v tomto editoru. Kapitola rovněž popisuje i postup vytvoření jednotlivých modelů. Součástí postup je i správné nahrání modelu na serveru a spuštění simulace. V prvním příkladu je postup popsán podrobněji pro přiblížení práce s editorem. Popis příkladu začíná v bodě po korektním spuštění editoru. Korektní spuštění je popsáno v manuálu, který je součástí přílohy této práce.

6.1 Úvodní příklad s postupem

Prvním krokem je otevření po případě vytvoření třídy. Pokud si uživatel přeje založit nový model je doporučeno vytvořit novou třídu. Vytvořit novou třídu lze pomocí operace *New class*. Tato operace se nachází v nabídce *Add*, která je součástí hlavního menu. Před použitím této operace je potřeba označit složku, kde se má třída vytvořit. Pokud uživatel nechce použít dědičnost z jiné třídy, stačí mu v navigačním panelu označit nejvýše uvedenou složku. V příkladu na obrázku 6.1 se jedná o složku *C:\User\Antonín\Desk…* Pokud uživatel chce využít dědičnost jiné třídy, stačí mu označit složku třídy, ze které chce dědit. Pro usnadnění práce lze tuto operaci vyvolat použitím klávesové zkratky *ctrl* + *shift* + „C“. Před samotným vytvořením se editor dotáže na pojmenování třídy. Po zadání příslušného jména třídy do modálního okna se aktualizuje adresářová struktura v navigačním panelu. Pokud vytvoření třídy proběhlo v pořádku, objeví se ve struktuře nová složka se zadaným názvem třídy. V opačném případě modální okno uvědomí uživatele, že vytvoření třídy nebylo možné. Obrázek 6.1 ilustruje stav navigačního panelu před a po přidání nové třídy *class6*.

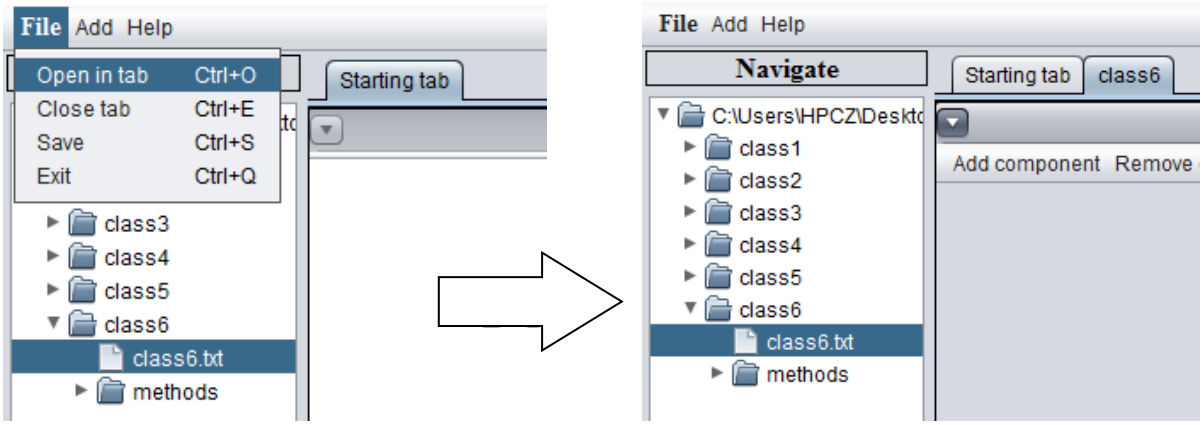


Obrázek 6.1: Vytvoření nové třídy

Pokud by si uživatel přál vytvořit ve třídě metodu, stačí klikem označit složku *methods* v dané třídě a použít operaci *New method*. Pro tuto operaci je implementována klávesová zkratka *ctrl* + *shift* + „M“. Průběh operace probíhá obdobně, jak u předchozího případu.

Po vytvoření nové třídy může uživatel otevřít soubor, který nese stejný název jako třída a nachází se uvnitř složky. Jedná se o textový soubor s příponu *.txt*. Před samotnou operací pro otevření je potřeba kliknutím označit soubor, se kterým chceme pracovat. Po označení příslušného souboru

stačí použit operaci *Open in tab*. Tuto operaci je možno nalézt v nabídce *File*, která je součástí hlavního panelu. Pro urychlení práce lze také využít klávesovou zkratku této operace *ctrl* + „O“. Pokud nedošlo k poškození souboru, otevření by mělo proběhnout správně. Správné otevření se projeví tak, že v záložkách nad pracovní plochou se zobrazí nová záložka s názvem otevřeného souboru bez koncovky. Pokud soubor obsahuje data, která korektně popisují síť, pak budou automaticky nahrána a zobrazena. Zobrazení dat se projeví vykreslením sítě na pracovní plochu. Obrázek 6.2 reprezentuje otevření objektové sítě třídy *class6*.

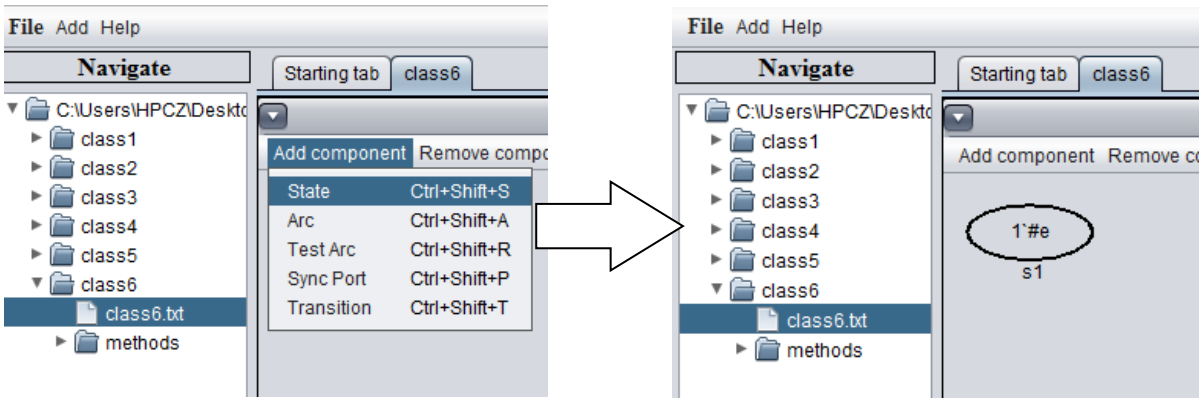


Obrázek 6.2: Otevření objektové sítě

Pro zavření záložky (sítě na pracovní ploše) slouží operace *Close tab*. Při použití této operace není potřeba označovat cokoliv v navigačním panelu. Operace uzavře aktuálně zobrazenou záložku. I u této operace lze využít klávesovou zkratku *ctrl* + „E“. Pokud před uzavření záložky není použita operace pro uložení, může dojít ke ztrátě vypracovaného modelu. Operace pro uložení je součástí nabídky *File*. Operace provede uložení aktuálně zobrazeného stavu sítě do souboru. Klávesová zkratka pro operaci je *ctrl* + „S“.

Pro práci se sítěmi jsou operace situovány v panelu nad pracovní plochou. Tento panel obsahuje několik nabídek, ve kterých se skrývají jednotlivé operace. V nabídce *Add component* jsou uloženy operace pro přidání prvků do sítě. Všechny operace v této nabídce mají přiřazené klávesové zkratky pro rychlejší práci s editorem. Přehled klávesových zkratek v dané nabídce je zobrazen na levé straně obrázku 6.3. Pro přidání místa (stavu) do sítě je v nabídce operace *State*. Při použití této operace dojde dvakrát k objevení modálního okna pro zadání parametrů nového místa. První modální okno se požaduje zadání počátečního značení. Druhé modální okno vyžaduje zadání počáteční akce stavu. Po zadání těchto parametrů dojde k vytvoření stavu, který se objeví v levém horním rohu pracovní plochy. Poté je stav hotový a lze s ním dále pracovat. Obrázek 6.3 zobrazuje vliv provedení operace na editor. Pro tento konkrétní příklad je v modelu vytvořeno více podobných stavů.

Pro přidání přechodu do sítě slouží operace *Transition*. Tato operace je rovněž součástí nabídky *Add component*. Před samotným vytvořením přechodu editor vyžaduje zadání dvou parametrů. Zadání parametrů probíhá opět pomocí modálních oken. První modální okno žádá o zadání stráže přechodu. Druhé modální okno požaduje zadání akce přechodu. Po dokončení zadávání parametrů je přechod vytvořen v levém horním rohu pracovní plochy. Součástí tohoto příkladu jsou dva přechody.



Obrázek 6.3: Přidaní nového stavu

I když synchronní port není součástí tohoto příkladu, tak jeho vytvoření je velmi podobné vytvoření přechodu. Obě vytvoření si liší v názvu operace, kdy pro přidání synchronního portu je zavedena operace *Sync Port*. Zároveň se obě operace od sebe liší zadáváním parametrů. Synchronní port vyžaduje jako první zadání vzoru zprávy, na který port reaguje.

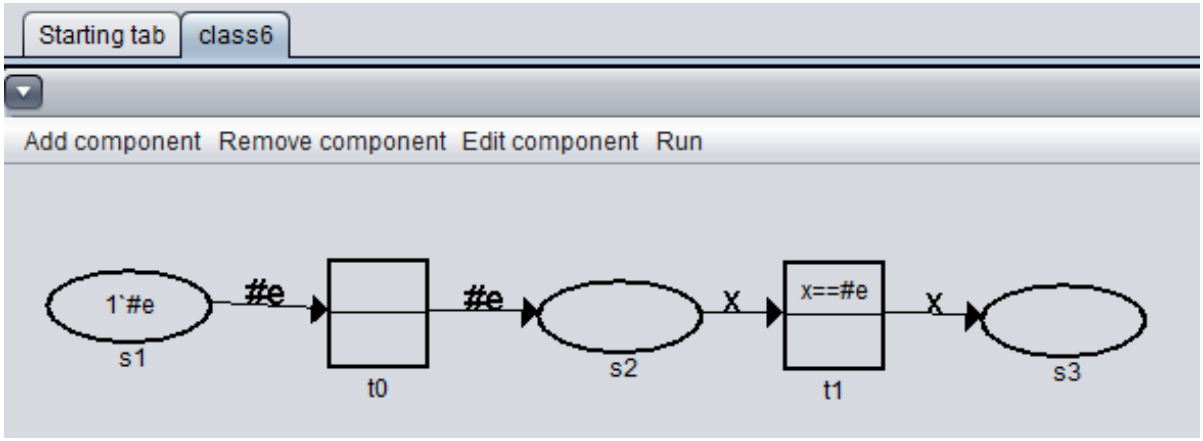
Pro přidání hrany spojující jednotlivé prvky je v nabídce *Add component* zavedena operace *Arc*. Vytvoření hrany se liší od ostatních prvků především tím, že se přímo váže na ostatní prvky. Samostatná existence hrany bez dalších prvků není možná, a proto tato operace vyžaduje určení, které prvky bude spojovat. Určení těchto prvků probíhá pomocí kliknutí kurzoru na daný prvek. Před prvním určení prvku je zobrazeno modální okno, které uživatele vybízí k označení zdroje hrany. Po kliknutí[1] na jeden z prvků se objeví nové modální okno, které uživateli sděluje, že právě určil zdroj hrany a nyní může určit cíl hrany. Kliknutím na cílový prvek dojde k vyvolání posledního modálního okna, které vyžaduje zadání hranového výrazu. Pokud by uživatel zvolil stejný typ prvku pro zdroj i cíl, pak editor zruší nedokončenou operaci a upozorní na to uživatele pomocí modálního okna oznamující chybu.

Poslední operací v nabídce je *Test arc*. Tato operace vytváří testovací hranu spojující dva různé prvky. Průběh operace je velmi podobný, jak je tomu u běžné hrany. Jediný rozdíl, který je pro uživatele důležitý, je ten, že se nemusí rozhodovat, který prvek zadá jako zdroj a který jako cíl. Ostatní aspekty této operace jsou totožné s operací *Arc*.

Pomocí všech operací je vytvořena síť, která je zobrazena na obrázku 6.4. Tento obrázek ukazuje stav sítě na počátku před zahájením simulace. Před samotnou simulaci je doporučeno uložit si stav modelu pomocí operace *Save*. Pokud při simulaci dojde ke změně značení v síti, nejrychlejší způsob obnovy sítě do původního stavu je zavření a opětovné otevření záložky. Druhou možností je editace stavů, ale to je zbytečně zdlouhavé.

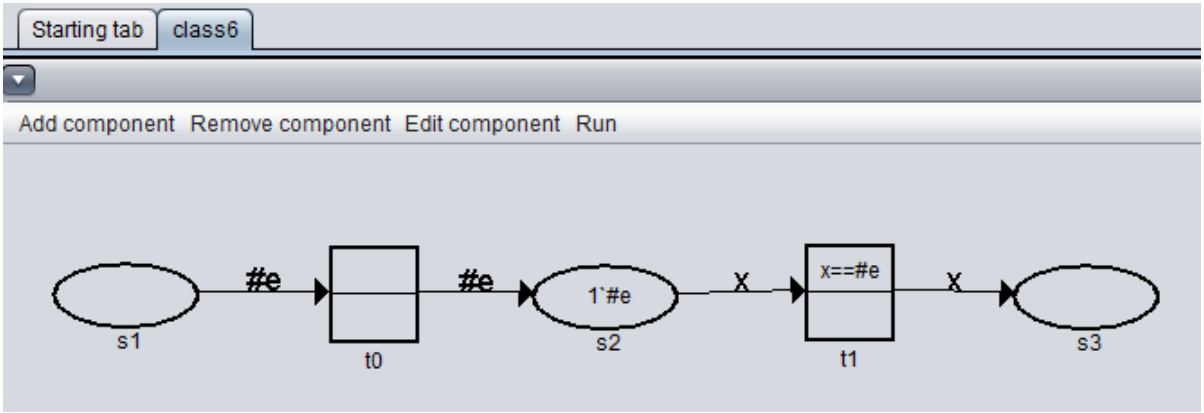
Pro zahájení simulace je potřeba model uložit na server. Pro tento úkol slouží operace *Load model to server*. Operaci lze nalézt v nabídce *Run*. Po spuštění operace se objeví modální okno. Pokud je model korektní, okno oznámí uživateli, že vše proběhlo v pořádku. Pokud by server odmítl nahrát model na server, pak modální okno oznámí uživateli, že server odmítl model, protože model je nekorektní. V tomto příkladu se samozřejmě jedná o korektní model a jeho nahrání na server proběhlo správně.

[1] Kliknutím na prvek nesmí dojít ke změně polohy prvku. Pokud k tomu dojde, pak se nejedná o kliknutí, ale o přetažení prvku na jinou pozici.



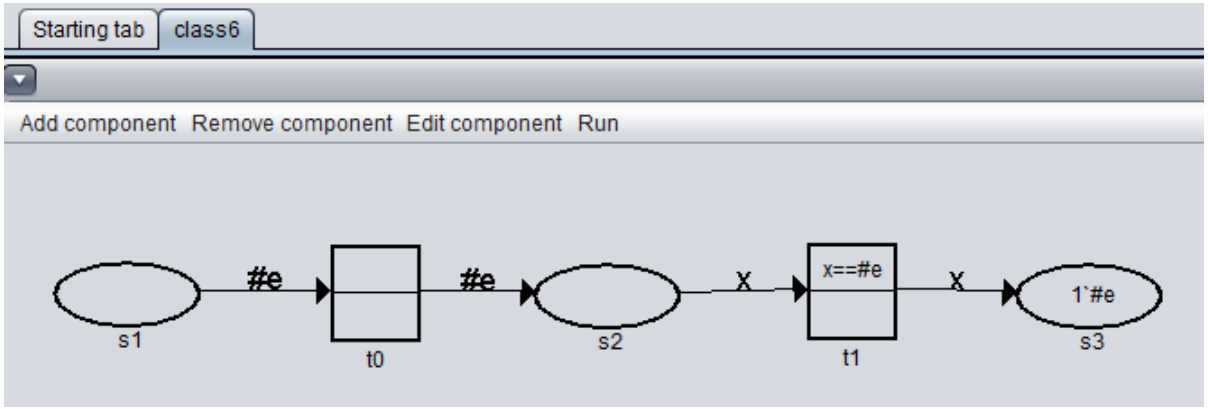
Obrázek 6.4: První příklad v čase 0

Po nahrání modelu na server lze experimentovat se simulací. Pro experimentování se simulací slouží dvě operace. První operací je *One step simulation*. Tato operace provede pouze jeden krok v simulaci. Provedení operace se projeví okamžitou změnou značení v síti. Obrázek 6.5 reprezentuje provedení této operace v tomto příkladu. Opětovné provedení této operace reprezentuje obrázek 6.6. Zároveň tento obrázek představuje konečný stav sítě. Druhou operací je *Run simulation*. Tato operace provede všechny možné kroky simulace až do konečného stavu. V tomto příkladu by výsledek vypadal jako skok stavu sítě z obrázku 6.4 rovnou do obrázku 6.6.



Obrázek 6.5: První příklad v čase 1

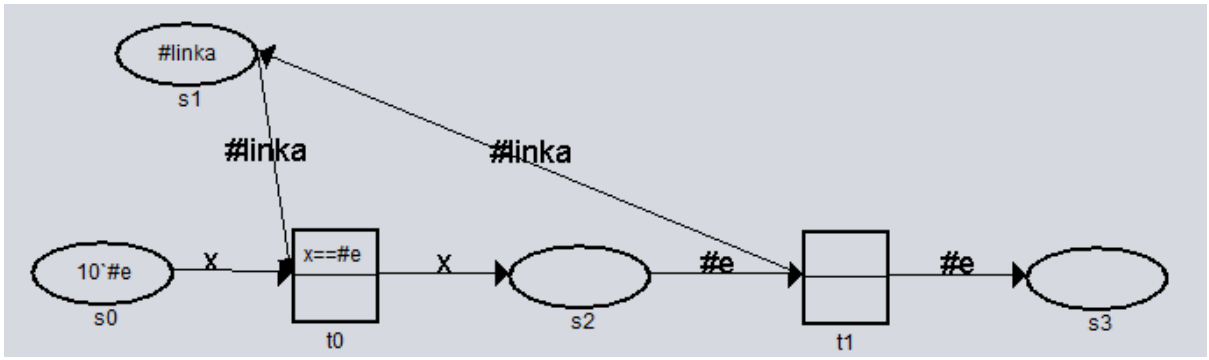
Po dokončení simulace je doporučeno opětovné načtení modelu do počátečního stavu. Poté je dále možno editovat síť dle potřeby uživatele. K této editaci slouží operace ze záložky *Edit component*. Jejich použití se téměř neliší od operací pro přidávání nových prvků. V podstatě stačí uživateli držet se postupu, který vytváří modální okna, aby tyto operace správně používal.



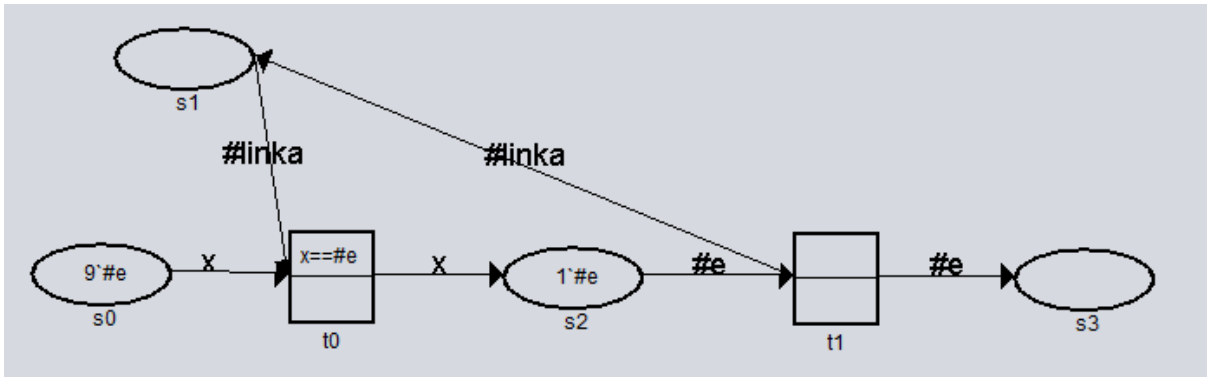
Obrázek 6.6: První příklad v čase 2

6.2 Obslužná linka

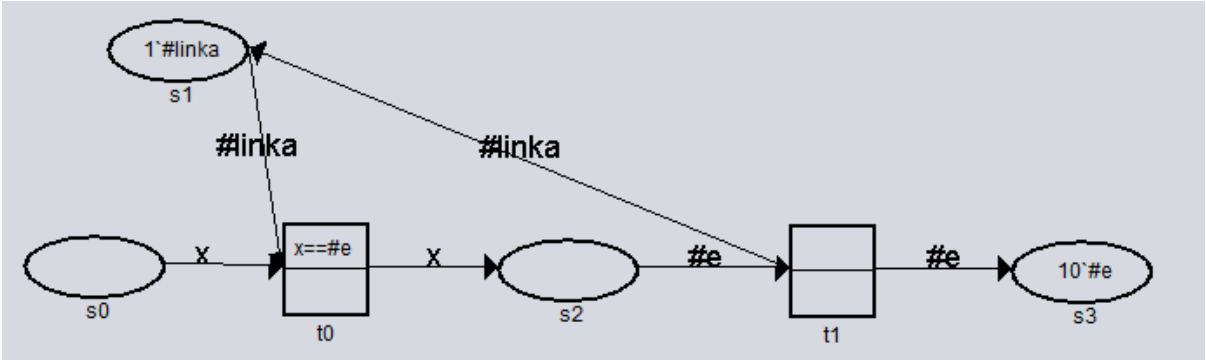
Druhý příklad modeluje obslužnou linku. Tato linka postupně převádí primitivní objekty ze stavu *s0* do *s3*. Na vstupu do linky je 10 objektů, které se po jednom přesouvají skrz linku. Při vstupu do linky je pomocí stráže přechodu kontrolováno, zda přichází pouze požadovaný objekty. Obrázek 6.7 reprezentuje stav linky, když není linka obsazena. Obrázek 6.8 zobrazuje stav linky, kdy je obsazena. Obrázek 6.9 představuje konečný stav sítě po dokončení simulace.



Obrázek 6.7: Druhý příklad, volná linka



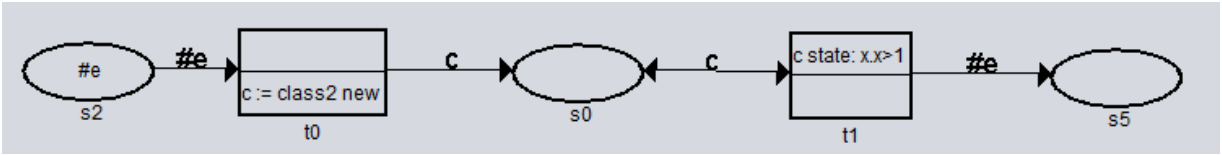
Obrázek 6.8: Druhý příklad, obsazená linka



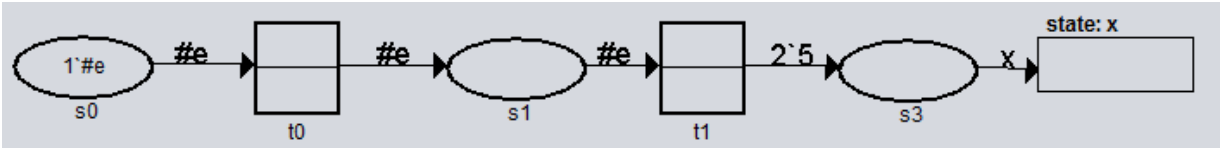
Obrázek 6.9: Druhý příklad, konečný stav

6.3 Příklad se dvěma třídami

Ve třetím příkladu je už využita objektová orientace Petriho sítí. Součástí příkladu jsou dvě objektové sítě dvou tříd. První třída se nazývá *class1* a jedná se o počáteční třidu modelu. Druhý třída se nazývá *class2*. Instance druhé třídy bude uložena v proměnné, která je součástí první třídy. Obrázek 6.10 zobrazuje třídu *class1*. Obrázek 6.11 reprezentuje třídu *class2*.



Obrázek 6.10: Třetí příklad, třída *class1* v čase 0



Obrázek 6.11: Třetí příklad, třída *class2* v čase 0

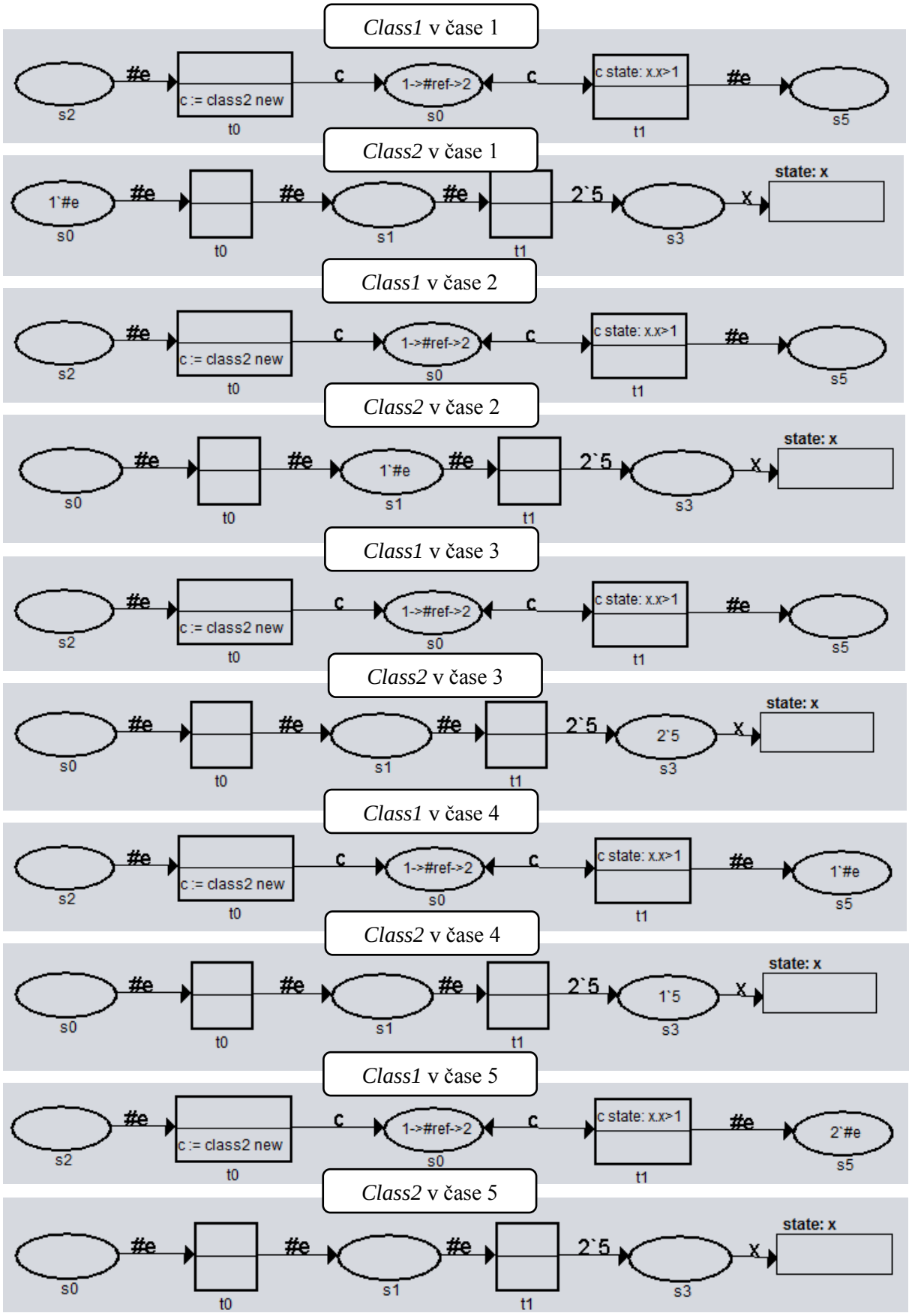
Bohužel editor nezvládá zobrazovat změny ve dvou sítích najednou. A proto bude zobrazovat změny vždy ve třídě, která je počáteční. Pro lepší pochopení příkladu, jsou jeho součástí zobrazeny i změny, které editor nezvládá zobrazit. Obrázek 6.12 představuje průběh simulace těchto dvou tříd v závislosti na diskrétním čase.

Ve třídě *class1* započne simulace modelu. Prvním krokem simulace je provedení přechodu *t0*. V přechodu dojde k vytvoření nového objektu a jeho uložení do stavu *s0*. Objekt je instancí třídy *class2*. Po vytvoření objektu se simulace přesouvá do druhé třídy. Zde jsou bez jakéhokoliv problému provedeny přechody *t0* a *t1*. Ve stavu *s3* jsou uloženy dvě čísla hodnoty 5. Na vstup synchronní port jsou přivedeny požadované hodnoty. Přechod *t1* ve třídě *class1* volající synchronní port načte do proměnné x hodnotu 5. Jelikož hodnota v proměnné x je větší než 1, je přechod proved. Tento přechod je proveden dvakrát. Výsledkem celé simulace je přivedení dvou symbolů *#e* do stavu *s5*.

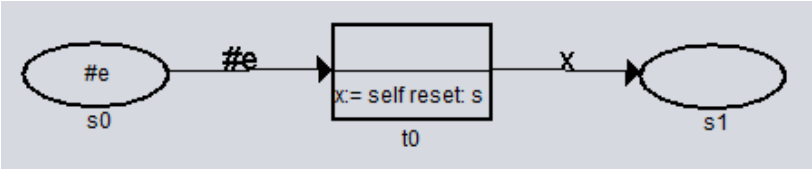
6.4 Příklad s metodou

Příklad se skládá z počáteční třídy *class3* a její metody. Metodě je přiřazen vzor zprávy *reset: s*. Tento vzor zprávy lze nalézt v názvu souboru, ve kterém jsou data metody uloženy. Jelikož operační systém Windows nedovoluje použití symbolu dvojtečky v názvu souboru, je tento znak nahrazen podtržítkem. Tato obměna na chod editoru nemá žádný vliv. Jedná se pouze o vizuální nepřesnost.

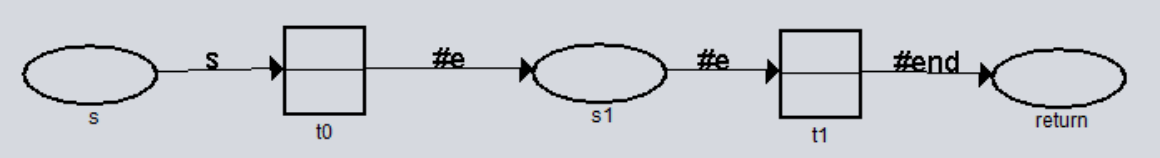
V akci přechodu *t0* třídy *class3* dojde k vyvolání metody. Do metody je poslán jako parametr symbol #e. Tento symbol přijde do vstupního stavu *s* metody. Symbol projde až k přechodu *t1*. Do stavu *return* je poté nahrán symbol *#end*. Tento stav je výstupním místem metody. Symbol je nahrán do proměnné *x* a poslán k uložení do stavu *s1* třídy *class3*. Obrázek 6.13 popisuje úvodní stav třídy *class3*. Obrázek 6.14 znázorňuje počáteční stav metody. Obrázek 6.15 představuje konečný stav sítě. Jelikož konečný stav metody se od úvodního neliší, není zde ilustrován stav metody na konci simulace.



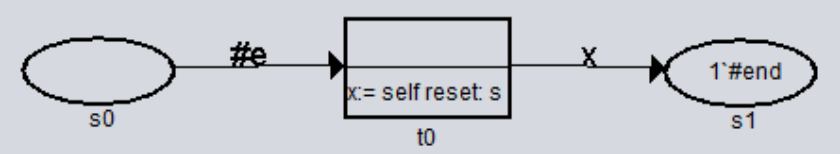
Obrázek 6.12: Třetí příklad, průběh simulace



Obrázek 6.13: Čtvrtý příklad, třída *class3* v čase 0



Obrázek 6.14: Čtvrtý příklad, metoda reset: s



Obrázek 6.15: Čtvrtý příklad, konečný stav třídy *class3*

7 Závěr

Hlavní výsledkem této práce je grafický editor. Tento editor se mi povedlo implementovat pro platformu Windows. Editor poskytuje vývojové prostředí pro tvorbu modelů Objektově orientovaných Petriho sítí implementovaných v jazyce PNtalk.

Osobně výsledný editor považuji za poměrně funkční. I když tato práce neobsahuje všechny testy, které na editoru byly provedeny, tak jeho celkovou funkčnost odhaduji zhruba na devadesát procent. Design editoru není nijak oslnivý a zasloužil by si lepší vzhled.

Pro budoucí rozvoj aplikace by bylo vhodné rozšířit editor o několik funkcí, které pomohou lépe zobrazit sítě. Touto funkcí může být rozšíření editoru o časovou osu, která bude zobrazovat aktuální čas simulace. Dalším rozšířením může být krokování zpět v simulaci.

Práce na tomto projektu mě přinesla hodně pozitivního. Od rozšíření znalostí o Petriho sítí až po samotné programování v jazyce Java. I když programování v jazyce Java pro mě nebyla novinka, tak tato bakalářská práce mi pomohla lépe koncipovat složité projekty.

Literatura

[1] VORÁČOVÁ, Šárka, Martin PĚNIČKA a Jaroslav VESELÝ. Úvod do modelování procesů Petriho sítěmi. Praha: Česká technika - nakladatelství ČVUT, 2008. ISBN 978-80-01-03979-3.

[2] DORDA, Michal. Úvod od Petriho sítí [online]. [cit. 2017-05-01]. Dostupné z: http://homel.vsb.cz/~dor028/Nekonvencni_metody_1.pdf

[3] BALDA, Pavel. Úvod do Petriho sítí [online]. ZČU Plzeň [cit. 2017-05-01]. Dostupné z: http://vendulka.zcu.cz/Download/Free/IRS1/IRS1-01_Petriho_site.pdf

[4] EHRIG, Hartmut. Unifying Petri nets: advances in Petri nets. New York: Springer, c2001. ISBN 978-3-540-43067-4.

[5] PERINGER, Petr. *Modelování a simulace* [online]. Vysoké učení technické v Brně, 2016, s. 332 [cit. 2017-05-01]. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FIMS-IT%2Flectures%2FIMS.pdf&cid=11453

[6] JANOUŠEK, Vladimír. Modelování objektů Petriho sítěmi. Brno, 1998. Disertační práce. Vysoké učení technické.

[7] VORÁČOVÁ, Šárka. Petriho sítě. FD ČVUT [online]. Praha [cit. 2017-05-01]. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FIMS-IT%2Flectures%2FIMS.pdf&cid=11453

[8] ČEŠKA, Milan, Vladimír MAREK, Petr NOVOSAD a Tomáš VOJNAR. Petriho sítě [online]. Brno, 2009 [cit. 2017-05-01]. Dostupné z: http://www.fit.vutbr.cz/study/courses/PES/public/Pomucky/PES_opora.pdf. Studijní opora. Vysoké učení technické.

[9] KŘENA, Bohuslav a Radek KOČÍ. Úvod do softwarového inženýrství [online]. Brno, 2010 [cit. 2017-05-10]. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/course-files-st.php?file=%2Fcourse%2FIUS-IT%2Ftexts%2FIUS_opora.pdf&cid=9410. Studijní opora. Vysoké učení technické.

[10] PAVLÍČKOVÁ, Jarmila a Luboš PAVLÍČEK. *Úvod do Javy*. Praha: Oeconomica, 2005. ISBN 80-245-0963-6.

[11] LOY, Marc. a Robert. ECKSTEIN. *Java Swing*. 2nd ed. Sebastopol, CA: O'Reilly, 2003. ISBN 05-960-0408-7.

Seznam příloh

Příloha 1.
Manuál pro spuštění editoru

Tento manuál je součástí i souboru README.txt na CD.

1. Stáhněte aplikaci *Pharo* verze 5.0 z http://pharo.org/web/download a nainstalujte.
2. Na CD je složka *PNtalkServer*, kde se nachází soubor *PNtalk-2.1-server.image*, který otevřete pomocí aplikace *Pharo*.
3. Po spuštění aplikace *Pharo* se objeví okno nazvané *Playground*, kde se nachází text *PNtalkServer start*. Tento text označte do bloku a klikněte na něj pravým tlačítkem. Objeví se nabídka, kde stisknete možnost *Do it*. Od této chvíle je server spuštěn.
4. Na CD se nachází složka *PN_Editor*. V této složce se nachází soubor *PN_Editor.jar*. Po spuštění tohoto souboru se objeví Grafický editor Objektově orientovaných sítí.
5. Adresářová struktura uložených souborů se nachází v této složce: PN_Editor\src\Editor\classes.

Příloha 2.

Obsah CD

1. Složka PN_Editor – obsahuje zdrojové kódy i samotný editor.
2. Složka PNtalkServer – obsahuje prostředky pro spuštění serveru.
3. Textový soubor README.txt s manuálem
4. PDF soubor BP_xneuzi05 s technickou zprávou