České vysoké učení technické v Praze

FAKULTA ELEKTROTECHNICKÁ

Katedra řídící techniky

Diplomová práce

Editor Petriho sítí

Luboš Svoboda

Vedoucí práce: Ing. Martina Svádová Leden 2003

Abstrakt

Tato diplomová práce se zabývá úkolem návrhu a implementace editoru k tvorbě a editaci Petriho sítí. Editor podporuje diskrétní, spojité i hybridní Petriho sítě a jazyk PNML, který je používán jako formát pro ukládání dat popisujících Petriho sítě. Editor úzce spolupracuje s Petri Net Matlab Toolboxem. Pro řešení byl použit programovací jazyk Java, jeho grafická knihovna Swing a rozhraní pro zpracování XML dokumentů Java API for XML Processing.

Abstract

The aim of the thesis is to design and implement an editor for creating and editing of the Petri nets. Editor supports discrete, continous and hybrid Petri nets and PNML, which is used as a format for storing of data describing Petri nets. Editor is cooperating with Petri Net Matlab Toolbox. The realization is based on programming language Java, its graphics library Swing and on interface for processing XML documents Java API for XML Processing.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č.121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne ....

podpis

Poděkování

Na tomto místě bych rád poděkoval vedoucí mé diplomové práce Ing. Martině Svádové za laskavé přečtení textu a cenné připomínky a podněty. Dále bych pak chtěl poděkovat své rodině za její trpělivost a podporu při studiu.

Obsah

Prohlášení iv
Poděkování v

Úvod **3**
Seznam použitých zkratek 4

1 Petriho sítě **5**
1.1 Základní koncept 5
1.2 Autonomní Petriho sítě 6
1.2.1 Zobecněné Petriho sít 6
1.2.2 Barevné Petriho sítě 6
1.2.3 Petriho sítě s inhibičními hranami 6
1.2.4 Prioritní Petriho sítě 7
1.3 Neautonomní Petriho sítě 8
1.3.1 Synchronizované sítě 8
1.3.2 T - časované sítě 8
1.3.3 Spojité sítě 8
1.3.4 Hybridní sítě 10
1.4 Vlastnosti Petriho sítí 10
1.4.1 Ohraničená, bezpečná síť 10
1.4.2 Živá, kvaziživá síť 11
1.4.3 Reverzibilní síť 11
1.4.4 Graf dosažitelných stavů Petriho sítě 11

2 Programovací jazyk Java **12**
2.1 Základní rysy jazyka Java 12
2.2 Java platforma 13
2.3 Kompilace a běh programu 13

3 Jazyk PNML **15**
3.1 Jazyk XML 15
3.2 Koncept a terminologie PNML 16
3.3 Syntaxe PNML 16
3.3.1 Místa 17
3.3.2 Přechody 17
3.3.3 Hrany 17
3.3.4 Stránky 18
3.3.5 Síť 18

4 Realizace editoru **20**
4.1 Řešení hlavních částí editoru . . . 20
4.1.1 Jádro editoru . . . 21
4.1.2 Návrh grafických komponent . . . 22
4.1.3 Událostní model . . . 24
4.2 Vstupy a výstupy editoru . . . 27
4.2.1 Uložení sítě v PNML formátu . . . 27
4.2.2 Načtení sítě z PNML souboru . . . 31
4.2.3 Export sítě do obrázku JPEG . . . 32
4.2.4 Tisk . . . 32
4.3 Implementace ostatních funkcí . . . 33
4.4 Systém pluginů . . . 35
4.4.1 Realizace systému pluginů . . . 36
4.4.2 Plugin zobrazující graf dosažitelnosti . . . 38
4.5 Vytvoření instalačního souboru . . . 39

5 Uživatelský manuál programu PN Editor **41**
5.1 Co je PN Editor . . . 41
5.2 Instalace a spuštění editoru . . . 41
5.3 První kroky s editorem . . . 42
5.3.1 Vytvoření nové Petriho sítě . . . 42
5.3.2 Kopírování a vložení prvků sítě ze schránky . . . 43
5.4 Pokročilé funkce . . . 44
5.4.1 Editace vlastností sítě . . . 44
5.4.2 Přizpůsobení editoru požadavkům uživatele (menu "Options") . . . 47
5.4.3 Načtení a uložení sítě, tisk (menu "File") . . . 48
5.5 Systém pluginů (menu "Plugin") . . . 49
5.5.1 Vytvoření nového pluginu . . . 50
5.6 Technická data . . . 51
5.7 Seznam klávesových zkratek . . . 52

6 Závěr **53**

7 Příloha **55**
7.1 Obsah přiloženého CD . . . 55
7.2 M-file - uložení matic grafu dosažitelnosti . . . 56
7.3 Skript pro sestavení instalátoru v programu Inno Setup . . . 57

Úvod

Rozvoj systémů používajících Petriho sítě si vyžádal vytvoření specializovaných výpočetních prostředků pro jejich návrh a simulaci. Jedním z těchto prostředků je Matlab Toolbox for Petri Nets (MTPN) umožňující jejich analýzu, simulaci a visualizaci výsledků. Vstupem toolboxu pro všechny výpočty je popis Petriho sítě ve formě matic popisujících strukturu a stav sítě. Pro běžného uživatele je tvorba matic poměrně náročná a je pro něj mnohem příjemnější, když si může Petriho sít jednoduše nakreslit ve visuálním nástroji a poté ji přenést do toolboxu. Cílem této práce je vytvořit visuální editor, který by umožnil v uživatelsky přívětivém prostředí nakreslit Petriho síť a uložit ji ve formátu vhodném pro zpracování v MTPN. Tímto formátem je jazyk PNML (Petri Net Markup Language), který určuje formát pro ukládání dat popisujících Petriho sítě. Prostředí Matlab a s ním i MTPN jsou dostupné i na počítačích s operačním systémem UNIX a Macintosh. Z tohoto důvodu musí být editor naprogramován tak, aby se stal dostupným i na těchto operačních systémech. Toto lze nejjednodušeji splnit použitím programovacího jazyka Java.

V době, kdy byla zadána tato diplomová práce, neexistoval editor Petriho sítí, který by plně vyhovoval představám autorů MTPN. Existovalo však několik editorů, které byly určeny pro komplexnější analýzu a simulaci Petriho sítí. Tyto editory však většinou nepodporovali standard PNML, nebyly napsány v jazyku Java nebo neuměli pracovat s hybridními typy sítí. Dnes se situace poněkud zlepšila, stále však neexistuje editor, který by měl podobné vlastnosti jako *PN Editor* (editor navržený na základě této práce) a vyhovoval autorům MTPN. K zajímavým editorům patří *JARP Analyzer* [1], který je napsán v jazyku Java a částečně podporuje standard PNML (generovaný výstup ve formátu PNML ne zcela odpovídá standardu). Umožňuje nakreslit diskrétní Petriho sít a provést jednoduché simulace a analýzy. Zásadním nedostatkem je dost nepochopitelné ovládání a vzhled editoru. Dalším zajímavým editorem je *Income Process Designer* [2]. Jde o profesionální nástroj nabízející širokou škálu funkcí nejen pro Petriho sítě. Není napsán v jazyku Java, má příjemné uživatelské prostředí a podporuje formát PNML. Jeho nevýhodou je licenční politika, kdy jedna licence pro normální uživatele stojí téměř 1900$. Mnohem dostupnějším nástrojem je *PNK* (Petri Net Kernel) [3], který je zdarma ke stažení. Nejedná se však o plnohodnotný editor, nýbrž jen o jakési jádro, na kterém je vytvořené velice jednoduché grafické rozhraní. Ke stažení je k dispozici kompletní dokumentace včetně zdrojových textů a proto je toto jádro vhodné i k dalšímu vývoji. Tento editor je napsán v jazyku Java a podporuje formát PNML. Jeho širšímu použití brání nedokončené visuální rozhraní. Přehled o dalších editorech pro Petriho sítě lze nalézt v [4].

Celý dokument je rozdělen na šest částí. První kapitola stručně seznamuje se základními typy Petriho sítí a jejich vlastnostmi. Druhá kapitola začíná popisem jazyka XML a pokračuje standardem PNML odvozeným z XML. Ve třetí kapitole je vysvětlena koncepce jazyka Java a její základní rysy. Čtvrtá kapitola pojednává o způsobu návrhu programu *PN Editor* a jeho vnitřní struktuře. V páté kapitole je k dispozici uživatelský manuál programu *PN Editor* s popisem jeho funkcí a ovládání. Poslední kapitola obsahuje zhodnocení navrženého řešení a možné způsoby vylepšení do budoucna.

Seznam použitých zkratek

API	Application Programing Interface
LTM	levé tlačítko myši
JDK	Java Development Kit
JVM	Java Virtual Machine
OS	operační systém
PNML	Petri Net Markup Language
MTPN	Matlab Toolbox for Petri Net
PTM	pravé tlačítko myši

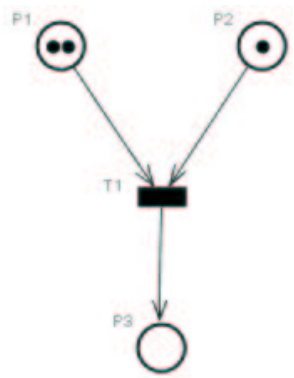
Kapitola 1

Petriho sítě

Petriho sítě jsou formálním a grafickým jazykem vhodným k modelování systémů se souběžností. Byly vyvinuty v 60. letech 20. století Adamem C. Petri a brzy se začaly používat jako jazyk pro popis, analýzu, synchronizaci, komunikaci a sdílení zdrojů mezi současně běžícími procesy. V praxi se ukázalo, že narážejí na dva závažné nedostatky. První problémem byl chybějící datový koncept. Modely byly velmi velké, protože všechny operace s daty musely být prováděny přímo na struktuře sítě. Druhým problémem byl chybějící hiearchický koncept - nebylo možné postavit velký model pomocí několika menších modelů s přesně definovaným rozhraním. Proto v 70.letech začal vývoj vysokoúrovňových Petriho sítí a hiearchických sítí odstranující tyto problémy. Výsledkem vývoje jsou barevné Petriho sítě spojující strukturování dat a hiearchický model bez kompromisů na kvalitu s původními Petriho sítěmi. V této kapitole popíšu základní koncept Petriho sítí, jejich typy a vlastnosti. Vycházel jsem přitom z [5].

1.1 Základní koncept

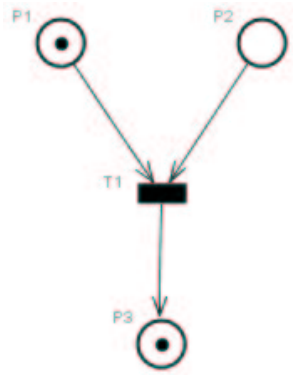
Petriho sítě mají dva typy uzlů: místa (places, P_i) a přechody (transitions, T_j). Místa a přechody jsou vzájemně spojeny pomocí hran (arcs). Hrany jsou orientované a spojují místo a přechod nebo přechod a místo. Každé místo obsahuje kladné číslo, které udává počet značek (tokenů) v daném místě. Tomuto číslu říkáme marking a značí se M(P_i) nebo zkráceně m_i.



Obrázek 1.1: Petriho síť s místy P_1, P_2, P_3 a přechodem T_1. Místo P_1 má marking $m_1 = 2$, P_2 má marking $m_2 = 1$ a marking celé sítě je M = (2, 1, 0)

Místo, ze kterého vede do přechodu hrana T_j, se nazývá vstupní místo přechodu T_j. Naopak

každé místo, do kterého vede hrana z přechodu T_j, se nazývá výstupní místo přechodu T_j. Přechod může být přeskočen, pokud každé vstupní místo obsahuje alespoň jeden token. Tento přechod se nazývá povolený. Síť na *obrázku 1.1* tuto podmínku splňuje a přeskočením přechodu T_1 vznikne síť na *obrázku 1.2*.



Obrázek 1.2: Síť vzniklá přeskočením přechodu T_1 sítě na *obrázku 1.1*

Přeskočení přechodu T_1 lze zobecnit pro jakýkoliv přechod T_j: při přeskočení přechodu T_j dojde ke snížení markingu o jeden token v každém vstupním místě a zároveň se zvýší marking o jeden token v každém výstupním místě. Pokud přechod může být přeskočen, neznamená to, že se to okamžitě stane. Je to pouze možnost. Ačkoliv nevíme, kdy bude přechod přeskočen, víme, jak bude vypadat příští vývoj markingu sítě. Pokud nemluvíme o časovaných ani synchronizovaných sítí (viz. 1.3.1), trvá přeskočení nulovou dobu a je dále nedělitelné.

1.2 Autonomní Petriho sítě

1.2.1 Zobecněné Petriho sít

Zobecněná Petriho síť má ke každé hraně přidruženou další vlastnost zvanou váha (násobnost). Pokud hrana $P_i \rightarrow T_j$ má váhu p, potom přechod T_j bude možné přeskočit, jen pokud vstupní místo P_i přechodu T_j bude obsahovat alespoň p tokenů. V okamžiku přeskočení se tedy sníží marking místa P_i o p tokenů. Zároveň se u všech výstupních míst přechodu T_j zvýší marking o počet tokenů odpovídající váze hrany směřující z T_j do daného výstupního místa.

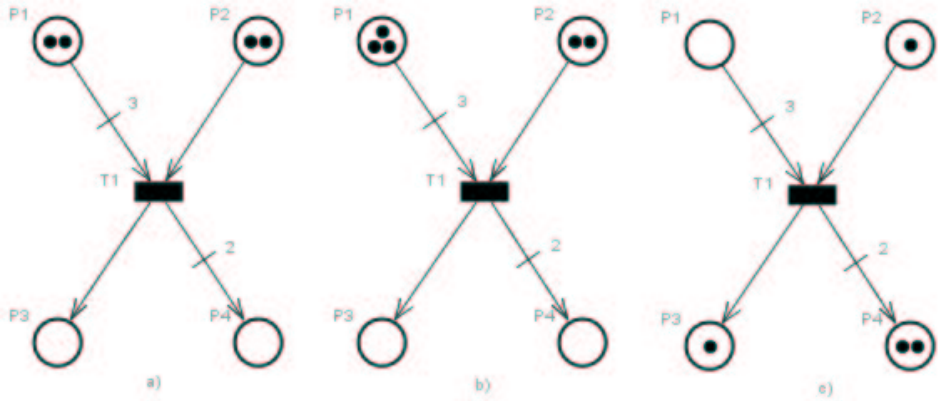
Obrázek 1.3.a ukazuje zobecněnou Petriho síť s hranou $P_1 \rightarrow T_1$ s váhou $p = 3$ a hranou $T_1 \rightarrow P_4$ s váhou $p = 2$. Přechod T_1 na *obrázku 1.3.a* nemůže být přeskočen, protože přechod P_1 obsahuje méně než tři tokeny. Na *obrázku 1.3.b* je síť před přeskočením T_1 (místo P_1 obsahuje tři tokeny) a na *obrázku 1.3.c* je výsledná síť po přeskočení přechodu T_1.

1.2.2 Barevné Petriho sítě

U těchto sítí je každý token určitého typu (má svou barvu). Přechody jsou ohodnoceny boolovskými výrazy tvořenými z proměnných a konstant, jejichž typy jsou podmnožinou množiny, ze které jsou tvořeny typy tokenů. Splnění podmínky přechodu je nutnou podmínkou jeho provedení.

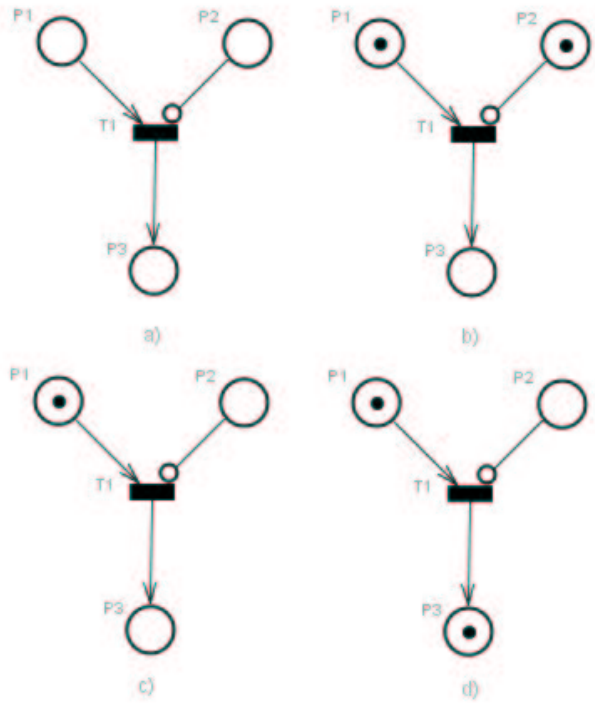
1.2.3 Petriho sítě s inhibičními hranami

Tyto sítě navíc obsahují inhibiční hrany směřující vždy od místa P_i do přechodu T_j. Inhibiční hranou neprocházejí tokeny a její význam je pouze informační. Říká, že přechod T_j může být



Obrázek 1.3: Zobecněná Petriho síť a možnosti přeskočení

přeskočen, pouze když vstupní místa P_i neobsahují žádné tokeny. Inhibiční hrana je zakončena malým kroužkem.



Obrázek 1.4: Petriho síť s inhibiční hranou. a. přechod T_1 nemůže být přeskočen, místo P_1 neobsahuje alespoň 1 token, b. přechod T_1 nemůže být přeskočen, protože v místě P_2 je token c. přechod T_1 může být přeskočen, d. síť po přeskočení přechodu T_1

1.2.4 Prioritní Petriho sítě

Někdy je vhodné určit přechod, který bude přeskočen při současném povolení několika přechodů. U prioritních sítí je proto ke každému přechodu přiřazena číselná hodnota (priorita), která udává

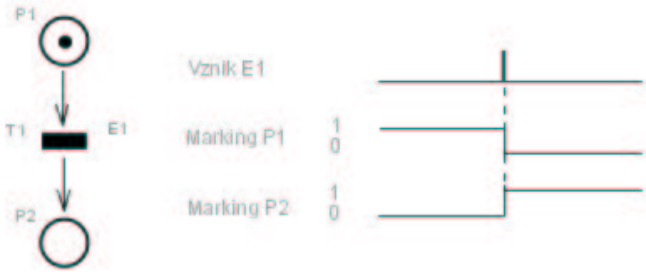
pořadí přeskočení přechodů při jejich současném povolení.

1.3 Neautonomní Petriho sítě

V této sekci jsou uvedeny základní druhy neautonomních sítí, kdy přeskočení přechodu je synchronizováno externí událostí, jejíž vývoj je časově závislý.

1.3.1 Synchronizované sítě

U těchto sítí je ke každému přechodu přiřazena vnější událost. Přeskočení přechodu nastane jen v případě, kdy přechod je povolený a současně nastane přidružená událost.



Obrázek 1.5: Princip přeskočení synchronizovaného přechodu

1.3.2 T - časované sítě

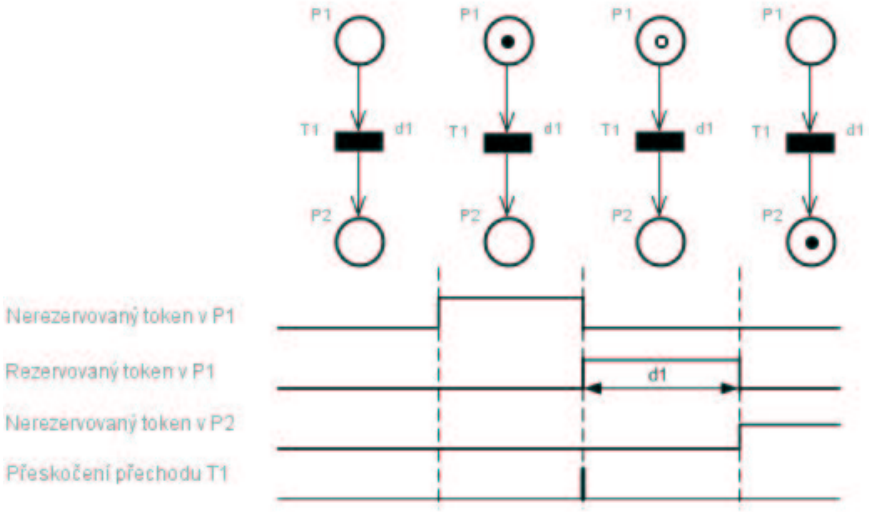
Časované sítě umožňují popis systémů, jejichž funkce je časově závislá (např. pokud určitá operace trvá nějaký čas). Existují dva modely popisů časovaných sítí: první, kde časování je asociováno s místy (P - časované) a druhý, kde je časování asociováno s přechody (T - časované). Dále popíšu pouze T - časované sítě, druhý model je podobný.

U časovaných sítí má token dva stavy. Může se stát rezervovaný pro přeskočení přechodu Tj nebo může být nerezervovaný. Tuto situaci ilustruje *obrázek 1.6*. V okamžiku, kdy má dojít k přeskočení přechodu T_1, je token na dobu d_1 rezervován v místě P_1 a po tuto dobu s ním nemůže být manipulováno. Po uplynutí doby d_1 je přechod efektivně přeskočen a rezervovaný token přeskočí do místa P_2, kde se opět stane nerezervovaným tokenem.

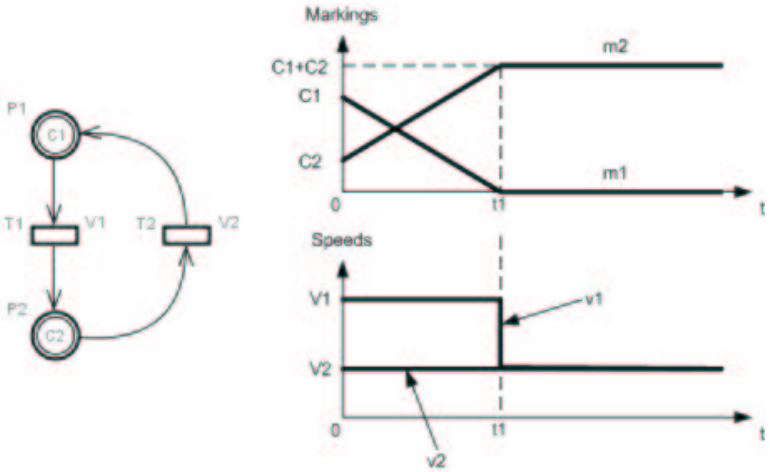
1.3.3 Spojité sítě

Ve spojitých Petriho sítích je, narozdíl od diskrétních Petriho sítí, počet značek (marks) a váha hran modelována reálným číslem. Ke každému přechodu T_j je navíc přiřazena jeho maximální rychlost V_j udávající, kolik značek může přechodem projít za jednotku času. Pokud jsou rychlosti všech přechodů konstantní, mluvíme o Petriho sítích s konstantní rychlostí (CCPN). Pokud rychlost přechodu závisí na markingu sítě, mluvíme o Petriho sítích s proměnnou rychlostí (VCPN).

Základní koncept spojitých Petriho sítí je vysvětlen na *obrázku 1.7*. Na tomto obrázku je Petriho síť představující výrobní systém obsahující $C_1 + C_2$ palet, které jsou ve dvou skladech. Oba tyto sklady jsou modelovány místy P_1 a P_2. Přechody T_1 a T_2 jsou pracovní stanice s maximální rychlostí V_1 a V_2 palet za jednotku času. Předpokládejme $V_1 > V_2$. Spuštěním výrobních strojů maximální rychlostí dojde v čase t_1 k vyprázdnění skladu P_1 a poklesu rychlosti $v_1(t) = V_2$, protože všechny



Obrázek 1.6: Rezervace tokenu v T - časovaných sítích



Obrázek 1.7: Spojitá Petriho síť

palety čekají na pomalejší stroj. Tento stav sítě se již nemění a říkáme, že byl dosažen stav se stabilní rychlostí. Z příkladu tak plynou dvě možnosti, kdy je přechod povolený:

1. přechod T_2 je silně povolený, dokud $m_2 > 0$
2. přechod T_1 je slabě povolený, protože $v1 > 0$ a $m_1 = 0$

Pro přesnou definici, kdy je spojitý přechod povolený, je třeba se seznámit s pojmem zásobené místo: místo P_i je zásobené v čase t jen tehdy, pokud existuje alespoň jeden přechod T_j, který je povolený a jeho výstupním místem je P_i. Potom přechod T_j je povolený v čase t, pokud všechny vstupní místa P_i přechodu T_j splňují alespoň jednu z následujících podmínek:

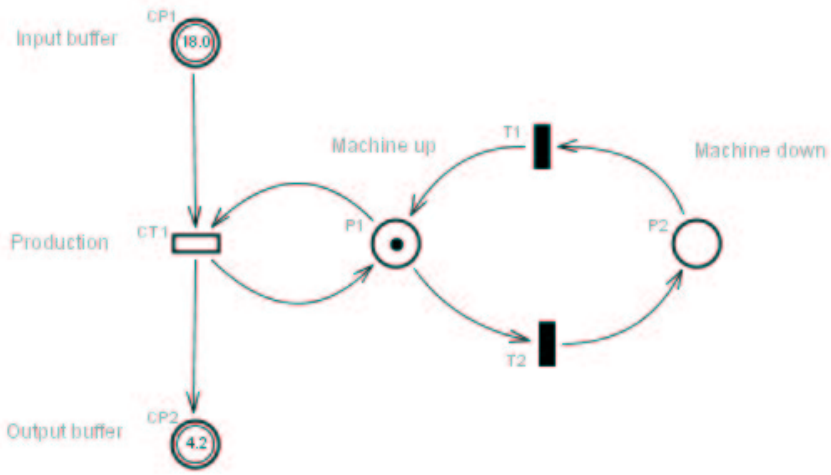
1. $m_i(t) > 0$.
2. P_i je zásobené.

Pokud je přechod T_j silně povolený, všechny vstupní místa přechodu T_j splňují první podmínku, pokud ne, přechod je povolený slabě.

1.3.4 Hybridní sítě

Hybridní sítě obsahují diskrétní i spojité prvky Petriho sítě popsané v předchozích odstavcích (*obrázek 1.8*). Používají se například pro modelování výrobních procesů, kde samotný výrobní proces je reprezentován spojitou částí a ovládání strojů je modelováno diskrétní části.

Obrázek 1.8: Typy prvků používaných v hybridních sítích



Obrázek 1.9: Použití hybridní sítě při modelování výrobního procesu (dvě vodní nádrže s ventilem)

Na *obrázku 1.9* je ukázáno použití hybridní sítě při modelování výrobního procesu. Předpokládejme, že CT_1 modeluje produkční stroj. Spojité přeskakování přechodu CT_1 odpovídá spojité produkci rychlostí V_1, když P_1 není prázdný. Pokud v místě P_1 není token, přechod CT_1 není povolený a stroj stojí.

1.4 Vlastnosti Petriho sítí

V tomto odstavci jsou vysvětleny některé vlastnosti Petriho sítí. Další informace o vlastnostech Petriho sítí lze získat v [5].

1.4.1 Ohraničená, bezpečná síť

Místo P_i je ohraničené pro počáteční marking M_0, pokud existuje přirozené číslo k takové, že pro všechny dosažitelné markingy ze stavu M_0 je počet tokenů v P_i roven maximálně k (P_i je potom k - ohraničené). Potom Petriho síť je ohraničená pro počáteční marking M_0, pokud jsou také všechny

místa sítě ohraničené. Bezpečná síť je speciálním případem ohraničené sítě, kdy každé místo všech dosažitelných markingů sítě může obsahovat maximálně jeden token.

1.4.2 Živá, kvaživá síť

Přechod T_j je živý pro počáteční marking M_0, pokud pro každý dosažitelný marking $M_i \in M_0$ existuje přeskoková sekvence z M_i obsahující T_j. Potom Petriho síť je živá pro počáteční marking M_0, pokud všechny její přechody jsou živé pro M_0.

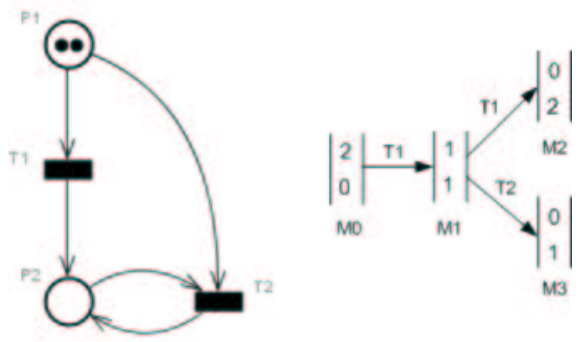
Přechod T_j je kvaživý pro počáteční marking M_0, pokud existuje přeskoková sekvence z M_0 obsahující alespoň jednou T_j. Potom Petriho síť je kvaživá pro počáteční marking M_0, pokud alespoň jeden přechod je kvaživý pro M_0.

1.4.3 Reverzibilní síť

Síť je pro počáteční marking M_0 reverzibilní, pokud existuje taková přeskoková sekvence S, která vrátí marking sítě z jakéhokoliv dosažitelného markingu $M_i \in M_0$ zpět do počátečního stavu M_0.

1.4.4 Graf dosažitelných stavů Petriho sítě

Jedna z metod analýzy vlastností Petriho sítě spočívá v konstrukci grafu dosažitelných stavů. Tento graf je složen z uzlů, kterým odpovídá dosažený stav sítě (marking). Hrany mezi uzly představují přeskočení přechodů, jejichž výsledkem je změna markingu sítě od jednoho k druhému.



Obrázek 1.10: Ukázka grafu dosažitelných stavů Petriho sítě

Kapitola 2

Programovací jazyk Java

Java je programovací jazyk vyvinutý společností Sun na začátku 90. let minulého století, který měl původně sloužit k programování domácích elektrospotřebičů. S nástupem internetu se ukázal jako ideální programovací jazyk pro internet. Jedná se o interpretovaný objektově orientovaný jazyk, který je nezávislý na platformě (a to jak z hlediska softwaru tak z hlediska hardwaru - aplikace napsané v Javě poběží ve Windows, Linuxu, ale třeba i na mobilním telefonu nebo jiném zařízení). Předpokladem je, že pro toto zařízení existuje tzv. virtuální stroj (Java Virtual Machine - viz. 2.2). Java je zvláště vhodná pro tvorbu internetových a klientských aplikací, protože podporuje širokou škálu internetových technologií. Naopak se nehodí pro systémové programování (psaní operačních systémů, driverů apod.) z důvodu, že tyto druhy aplikací jsou úzce svázány s jediným operačním systémem a jsou na ně kladeny vysoké požadavky na rychlost, které Java nesplňuje.

2.1 Základní rysy jazyka Java

Syntaxe jazyku Java je odvozena od jazyka C a C++. Byly vypuštěny některé složité konstrukce, které činí při programování nejčastější chyby a jsou přidány některé nové. Zde jsou některé základní rysy:

- **Plně objektově orientovaný jazyk.** Bez znalostí objektově orientovaného programování se v Javě nelze obejít. Neexistují tu žádné procedury či funkce. Ty jsou nahrazeny objekty a všechny operace jsou potom definovány jako operace s nimi.
- **Platformově nezávislý jazyk.** Program napsaný v Javě je možné spustit na jakémkoliv zařízení, pro které existuje virtuální stroj. Přenositelnost javovských programů je možná jak ve zdrojovém kódu, tak na binární úrovni v tzv. bytovém kódu (viz. 2.3).
- **Podpora internetu.** Java podporuje protokol TCP/IP, grafiku ve formátu GIF a JPG a zvuk ve formátu AU. Abecedy neanglických jazyků jsou podporovány 16-ti bitovým znakovým kódem Unicode.
- **Bezpečný jazyk.** Jedním z hlavních požadavků při vývoji Javy byla bezpečnost. Java má vestavěnou třídu SecurityManager, která kontroluje všechny nebezpečné akce.
- **Multithreading.** Java umožňuje provádění několika částí programů (vláken) v jednom okamžiku a poskytuje prostředky pro jejich synchronizaci. To zvyšuje výkon aplikací.

- **Jednodušší než C.** Oproti jazyku C má Java některá vylepšení. Není třeba dealokovat nepotřebné objekty - Java to dělá automaticky. Byly odstraněny ukazatele, preprocesor, chybí také příkaz goto a některé další rysy, které vedou k zjednodušení jazyka.
- **Rozsáhlá kontrola chyb a robustnost.** Při kompilaci i vlastním běhu programu Java kontroluje vznik možných chyb. Řídí využití aplikační paměti, čímž vylučuje možnost poškození obsahu operační paměti chybou v aplikačním programu. Vestavěný mechanismus provádí kontrolu výjimek, mezních hodnot polí a nulových ukazatelů. Aritmetika je definována podle normy IEEE, která zaručuje stejný výsledek na rozdílných systémech.
- **Distribuované aplikace.** Java podporuje vytváření dynamických a distribuovaných aplikací i jejich automatické šíření po síti. Za běhu aplikace lze zajistit upgrade softwaru.
- **Možnost psaní apletů.** Aplet je program napsaný v jazyku java určený pro zobrazení na WWW stránce. Na počítači nemusí být nainstalován virtuální stroj, protože je součástí prohlížeče. V apletech je možné využít celou sílu jazyka Java pro oživení stránek nebo k vytvoření komerční aplikace.

2.2 Java platforma

Nejdůležitější vlastností Javy je nezávislost na platformě. Tento fakt je důležitý hlavně pro internetové programování - na Internetu se setkávají uživatelé různých systémů a v blízké budoucnosti také různých zařízení. Tato multiplatformnost je umožněna tím, že Java není překládaný jazyk (jako např. C++ nebo Pascal), ale jazyk interpretovaný. To znamená, že příkazy pro procesor jsou překládány do strojového kódu při každém běhu programu. To umožňuje spouštět program na různých zařízeních s různými procesory. Hardwarové rozdíly mezi jednotlivými platformami zastřešuje Java platforma, která se dělí na dvě části:

- **Virtuální stroj (JVM - Java Virtual Machine).** Ten se skládá z runtime systému (realizuje vazbu na hardware) a interpretu (vykonává bytový kód). Interpret může být nahrazen JIT (Just In Time) kompilátorem, který při běhu programu nejprve překládá bytový kód do strojového, čímž se dosahuje zvýšení rychlosti.
- **Aplikační programové rozhraní (Java API).** Skládá se z knihoven pro psaní programů. Tyto knihovny není nutné distribuovat spolu s programem - jsou součástí Java platformy.

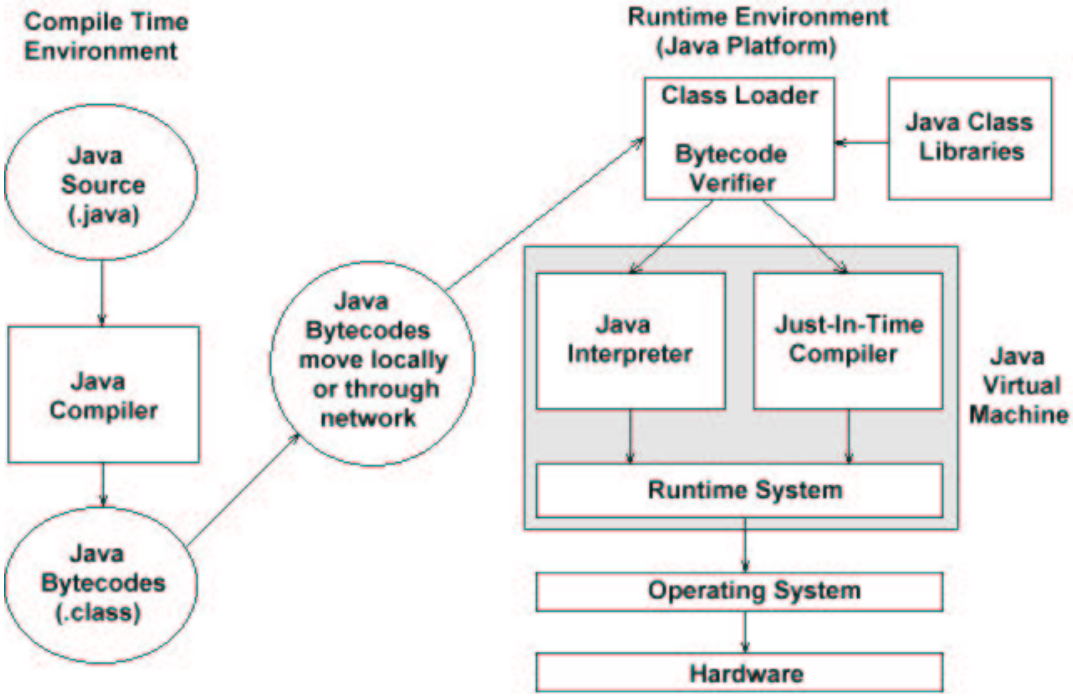
Protože spektrum aplikací a zařízení, pro které se Java používá, je velmi různorodé, rozdělila se Java platforma na další tři části:

- **Standart Edition (J2SE).** Standardní edice Javy, která obsahuje většinu potřebných funkcí pro vývoj běžných aplikací.
- **Enterprise Edition (J2EE).** Edice Javy rozšířená o technologie použitelné pro vývoj rozsáhlých podnikových a internetových aplikací.
- **Micro Edition (J2ME).** Platforma používaná k vývoji aplikací určených zejména pro mobilní zařízení a spotřební elektroniku.

2.3 Kompilace a běh programu

Zdrojový kód napsaný v Javě se ukládá do textového souboru s příponou .java. Výstupními soubory kompilátoru jsou binární soubory .class, které obsahují instrukce pro virtuální stroj (těmto

instrukcím se říká bytový kód - bytecode). Běh programu se u aplikace a appletu liší. U appletu řídí vykonávání programu internetový prohlížeč. Aplikace se po jejím spuštění nahraje do paměti, zkontroluje se její binární kód a poté se předá virtuálnímu stroji, který překládá instrukce bytového kódu do instrukcí konkrétního procesoru a předává tyto instrukce systému.



Obrázek 2.1: Průběh kompilace a běhu javovského programu

Kapitola 3

Jazyk PNML

Petri Net Markup Langugage (PNML) je jazyk, který určuje formát pro ukládání dat popisujících Petriho sítě. Je založen na jazyku XML a v současné době je stále ještě rozšiřován. Aktuální verzi lze nalézt na [6]. Než bude popsána syntaxe jazyka PNML, bude zde uveden koncept a terminologie PNML a krátký úvod do jazyka XML.

3.1 Jazyk XML

Jazyk XML (eXtensible Markup Language) [7] patří mezi značkovací jazyky (markup languages). Důležité části dokumentu se označují pomocí značek. V terminologii XML se jednotlivým označeným částem dokumentu říká elementy a jsou základním stavebním kamenem každého dokumentu. Elementy do sebe mohou být navzájem vnořené a tím dle potřeby mohou zachycovat strukturu informací uložených v dokumentu. Elementy se v textu vyznačují pomocí tzv. tagů. Většině elementů odpovídají dva tagy - počáteční a ukončovací.

```
<para>Toto je obsah elementu para</para>
```

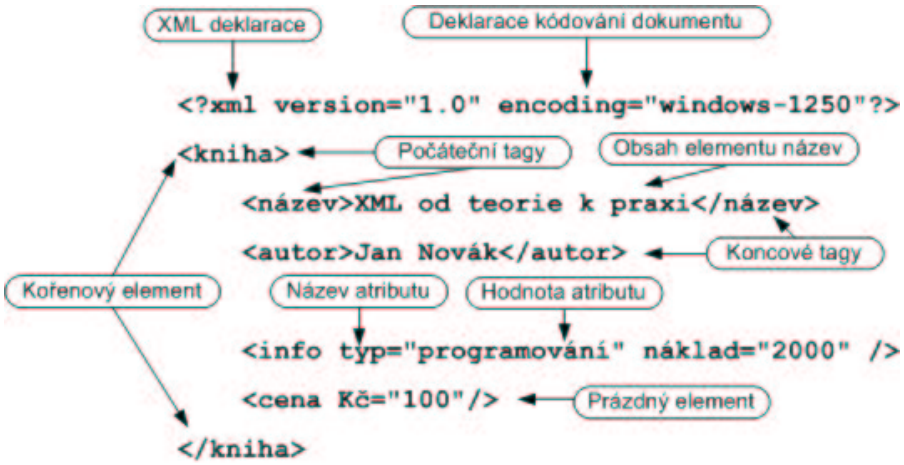
Ukázka obsahuje jeden element para. Jeho obsah je vyznačen pomocí tagů `<para>` (počáteční tag) a `</para>` (ukončovací tag). U každého počátečního tagu můžeme použít ještě atributy. Atributy se používají k upřesnění významu elementu nebo k přidání dalších důležitých informací.

```
<para zabezpečení="důvěrné">Tajná informace</para>
```

V tomto případě jsme atributu `zabezpečení` přiřadili hodnotu "důvěrné". Hodnotu atributu je nutné vždy uzavřít do uvozovek nebo do apostrofů. Každý XML dokument musí obsahovat informace o použitém kódování dokumentu a musí také obsahovat jeden kořenový element.

Výhody použití XML pro popis Petriho sítí lze shrnout do několika bodů:

- XML je otevřený a uznávaný standard
- není úzce svázán s nějakou platformou nebo proprietární technologií
- možnost použití různých znakových sad
- snadná konverze do dalších formátů
- možnost automatické kontroly správnosti dokumentu



Obrázek 3.1: Ukázka XML dokumentu

3.2 Koncept a terminologie PNML

Dokument napsaný v jazyku PNML může obsahovat několik Petriho sítí. Každá Petriho síť se skládá z objektů reprezentujících grafovou strukturu sítě. Tyto objekty jsou pojmenovány place, transition a arc. Pro popis struktury sítě existují další tři objekty - pages, reference places a reference transitions. Pro další výklad se bude pro reference places, reference transitions, place a transition používat označení uzly a pro jejich podmnožinu tvořenou z reference places a reference transitions označení referenční uzly. Každý objekt má unikátní identifikátor, který může být použit k odkazu na objekt. Pro popis dalších vlastností objektu se používají popisky (labels) objektu. Těchto popisků může mít objekt několik. Obvykle reprezentují jméno objektu nebo další vlastnosti jako marking místa nebo váhu hran. Správné popisky objektů a jejich kombinace se odvíjejí od typu sítě, kterou popisujeme. Popisky rozlišujeme na dva druhy: komentáře (annonations) a atributy (attributes). Komentář je obvykle použit pro jména objektů, marking apod. Atributy jsou popisky s konečnou množinou možných hodnot. Používají se například pro popis typu hrany, která může být normální nebo inhibitorová.

Každý objekt a každý komentář má svou grafickou reprezentaci. Pro místo je to jeho pozice a pro hranu seznam bodů definující prostřední body této hrany. Objekt i komentář má tzv. referenční bod, ke kterému se pozice prvku vztahuje. Standardně je to střed grafické reprezentace objektu, u komentáře je to levý spodní roh a u hrany je to střed prvního segmentu hrany.

Petriho síť může být rozdělena do několika částí. Tyto části se v PNML nazývají stránky (pages). Stránka je objekt, který se skládá z několika dalších objektů (places, transition, atd). Pro propojení mezi stránkami se používají referenční uzly. Referenční uzel odkazuje na jakýkoliv uzel v síti, tedy i mezi různými stránkami.

Pro některé nástroje používající PNML může být užitečné uložení některých interních informací, které nejsou podporovány ostatními nástroji. Vzhledem k tomuto požadavku může být každý objekt a popisek vybaven částí zvanou "tool specific information", jejíž interní formát si daný nástroj může libovolně definovat.

3.3 Syntaxe PNML

V PNML jsou sítě, objekty sítě a popisky reprezentovány jako XML elementy a jejich atributy. Tagy elementů jsou pojmenovány stejně jako typy objektů v Petriho síti podle odstavce 3.2 (kromě

popisků). Tagy popisků jsou pojmenovány podle jejich významu. Pokud se tedy objeví neznámý XML element v některé části dokumentu, je indikován jako popisek sítě nebo objektu.

3.3.1 Místa

```
<place id="p1">
  <graphics>
    <position x="10" y="20"/>
  </graphics>
  <name>
    <value>ready to produce</value>
    <graphics>
      <offset x="-5" y="2"/>
    </graphics>
  </name>
  <initialMarking>
    <value>3</value>
    <graphics>
      <offset x="0" y="0"/>
    </graphics>
  <initialMarking>
</place>
```

Příklad ukazuje reprezentaci místa s unikátním identifikátorem `p1`. Místo má dva popisky: první reprezentuje jméno místa (`name`) a druhé jeho počáteční marking (`initialMarking`). Popisek se skládá z jeho hodnoty (`value`) a jeho grafické reprezentace (`graphics`). Místo má kromě tagů s popisky také samostatný element `graphics` popisující umístění místa na pracovní ploše. Obsah tohoto elementu záleží na kontextu, ve kterém se objevuje. Místo (a obecně všechny objekty) má v tomto elementu vnořený element `position`, udávající absolutní pozici, kdežto popisek má místo toho element `offset` udávající relativní umístění vůči pozici objektu, ke kterému patří.

3.3.2 Přechody

```
<transition id="t1">
  ...
  <toolspecific tool="PNEditor" version="1.0">
    <color r="12" g="45" b="25"/>
  </toolspecific>
</transition>
```

Přechod je reprezentován podobně jako místo, chybí mu pouze popisek s počátečním markingem. Tento přechod navíc obsahuju specifické informace editoru PNEditor o barvě přechodu, které jsou obsažené v elementu `toolSpecific`. Tagy použité uvnitř tohoto elementu záleží pouze na uvážení tvůrce daného nástroje; samotný element `toolSpecific` by však měl obsahovat dva atributy: první je atribut `tool` říkající, ke kterému editoru se vztahují tyto specifické informace a druhým atributem je `version` udávající číslo verze daného nástroje. Kromě těchto dvou atributů nejsou ve standardu PNML žádná jiná doporučení ohledně atributu `toolSpecific`.

3.3.3 Hrany

```
<arc id="a1" source="p1" target="t1">
```

```
    <graphics>
      <position x="10" y="20"/>
      <position x="2" y="5"/>
    </graphics>
    <inscription>
      <value>10</value>
      <graphics>
        <offset x="-3" y="5"/>
      </graphics>
    </inscription>
    <type value="normal"/>
  </arc>
```

Tato hrana jde z uzlu s číslem id označeným v atributu `source` do uzlu s číslem id v atributu `target`. Grafická informace představuje seznam prostředních bodů, kterými hrana prochází. Element `inscription` udává váhu hrany. Navíc má hrana další atribut `type` označující o jaký typ hrany jde (normal, inhibitor).

3.3.4 Stránky

```
  <page id ="pg1">
    <name>
      <value>Example page of the net</value>
    </name>
    <referencePlace id="rp1" ref="p1">
      <name>...</name>
      <graphics>
        <position x="20" y="20"/>
      </graphics>
    </referencePlace>
    <referenceTransition id="rt1" ref="t1">
      ...
    </referenceTransition>
    <place id="p2">...</place>
    <transition id="t2">...</transition>
    <arc id="a2" source="rp1" target="t2">
      ...
    </arc>
    ...
  </page>
```

Tento výpis ukazuje reprezentaci stránky a referenčních uzlů sítě používaných pro jejich propojení. Referenční uzel odkazuje na uzel pomocí atributu `ref`, jehož hodnota identifikuje uzel v této síti.

3.3.5 Síť

```
  <net id="n1" type="hlNet">
    <name>
      <value>Example high-level net</value>
```

```
      <graphics pageref="pg1"/>
    </name>
    <place id="p1">
    ...
    </place>
    <page id="pg1">
      ...
    </page>
    ...
  </net>
```

Celá Petriho síť obsahuje stránky a objekty sítě popsané v předchozích odstavcích. Kromě toho má popisek `name`, který má dva elementy: v prvním je obsaženo jméno sítě (element `value`), ve druhém je informace udávající, ve které stránce se má jméno sítě zobrazit (element `graphics`). Element `net` definující síť má kromě atributu s číslem id také atribut `type`, udávající o jaký typ sítě se jedná. V současné době nejsou ve standardu PNML zahrnuty žádné předdefinované typy Petriho sítí.

Kapitola 4

Realizace editoru

V této části je popsán postup návrhu a způsob realizace editoru. Před samotným návrhem editoru byla provedena analýza, jejímž výsledkem je pět bodů, které by měl editor splňovat:

1. **Podpora PNML.** Toto je klíčová vlastnost celého editoru. Editor by měl umožnit import a export souborů ve formátu PNML se zachováním maximální možné kompatibility s ostatními editory a zároveň důsledně dodržet standard PNML.

2. **Spolupráce s Matlab Toolbox for Petri Nets.** Editor je primárně určen pro spolupráci s MTPN a proto bude poskytovat některé funkčnosti pracující pouze s MTPN.

3. **Platformová nezávislost.** Editor by mělo být možné provozovat na různých operačních systémech, čímž získá ještě širší použitelnost. [1]

4. **Otevřenost a rozšiřitelnost.** Je zřejmé, že editor nepokrývá všechny funkčnosti, které souvisí s Petriho sítěmi. Proto by mělo být možné editor dále rozšiřovat podle individuálních požadavků uživatelů. S těmito rysy je třeba počítat již při návrhu a vytvořit otevřený projekt připravený k dalším rozšířením.

5. **Uživatelsky přívětivé prostředí.** Pro skutečně širokou použitelnost editoru by se měl visuální návrh aplikace držet určitých konvencí, na které jsou uživatelé zvyklí. Nebude potom pro ně těžké začít editor okamžitě používat.

4.1 Řešení hlavních částí editoru

Pro realizaci editoru jsem použil programovací jazyk Java poslední specifikace (vývojový kit JDK 1.4.1) [9] od společnosti Sun a vývojové prostředí IntelliJ IDEA 3.0 [8]. Pro přehlednost zdrojových kódů, správný objektově orientovaný přístup a znovupoužitelnost tříd jsem rozdělil všechny třídy a rozhraní editoru do pěti balíků. Všechny tyto balíky jsou potom obsaženy v balíku `cz.cvut.k335.pneditor`.

Vlastní vývoj editoru probíhal postupně v několika krocích:

1. Analýza problému a navržení řešení (objektový návrh, rozvržení tříd do balíků, použité technologie, programovací techniky, atd.).

2. Vytvoření jádra editoru (implementace tříd popisující Petriho síť a její prvky).

[1] I když by tento rys měl být implicitně splněn použitím programovacího jazyka Java, lze i v tomto jazyku napsat konstrukce fungující pouze pod jedním operačním systémem.

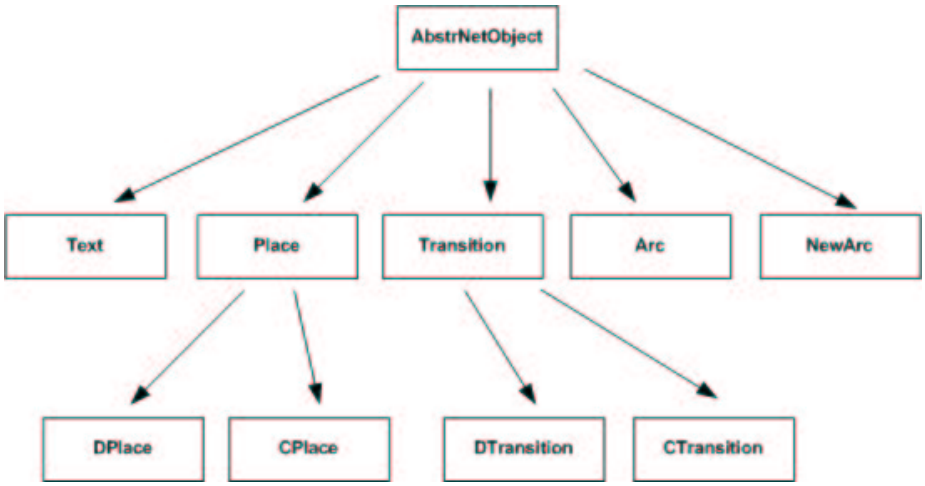
Obrázek 4.1: Rozdělení programu do balíků

3. Navržení visuálního vzhledu editoru a jeho naprogramování.
4. Propojení visuálních komponent s jádrem editoru.
5. Implementace vstupu a výstupu ve formátu PNML.
6. Implementace ostatních funkcí (schránka, historie, systém pluginů, atd.).
7. Testování a opravy chyb.
8. Vytvoření nápovědy a dokumentace.
9. Vytvoření instalátoru editoru.

4.1.1 Jádro editoru

Jádro editoru tvoří balík `netobject`, ve kterém jsou třídy reprezentující jednotlivé prvky sítě. Je zřejmé, že prvky Petriho sítě mají některé vlastnosti společné. Jedná se především o umístění, jméno, popis, barvu a určitou grafickou podobu. Těchto vlastností jsem využil při implementaci do jazyka Java a vytvořil pro prvky sítě společného předka s názvem `AbstractNetObject`. V této abstraktní třídě jsou všechny společné metody a atributy, které by měly mít všechny prvky sítě. Obdobnou úvahu lze použít i pro prvky diskrétních a spojitých sítí, kde např. diskrétní a spojité místo májí řadu společných vlastností. Proto je vhodné vytvořit třídu s obecným typem místa Place (děděnou z třídy `AbstractNetObject`) a z ní teprve vytvořit konkrétní třídy pro diskrétní nebo spojité místa (`DPlace`, `CPlace`). Aplikací tohoto pravidla i na přechody vznikne dědičná struktura na *obrázku 4.2*. Kromě tříd reprezentující prvky sítě jsem do balíku `netobject` navíc umístil třídy `NewArc` a `Text`. Třída `NewArc` popisuje hranu, která je uživatelem právě kreslena, třída `Text` je používána pro zobrazení popisků sítě a jejich prvků.

Prvky sítě a její popisky tvoří síť. Pro její reprezentaci jsem navrhl třídu `Net`, která popisuje strukturu nakreslené sítě a poskytuje metody pro manipulaci se sítí (přidání prvků, mazání, vyhledávání). Jednotlivé prvky sítě jsou uloženy v datové struktuře `java.util.Vector`, do které je možné vkládat a vybírat různé typy objektů a její velikost je omezena pouze dostupnou pamětí počítače. Použitím společného předka `AbstractNetObject` pro všechny prvky, které se ukládají do sítě, lze využít polymorfismu (konstrukce objektového programování) k vyvolání stejných operací nad různými třídami.



Obrázek 4.2: Obsah balíku `netobject` se stromem dědičnosti prvků

```
...
Vector net = new Vector();      //vytvoření struktury Vector
net.add(new DTransition(...));  //přidat prvky sítě
net.add(new DPlace(...));
net.add(new Arc(...));
...
AbstractNetObject ano;          //deklarace předka
for (int i = 0; i < net.size(); i++){    //projít všechny prvky v síti
  //vyjmutí i-tého prvku a přetypování na společného předka
  ano = (AbstractNetObject) net.getElementAt(i);
  ano.rotate();   //vyvolání metody rotate() ve správném následníkovi
}
```

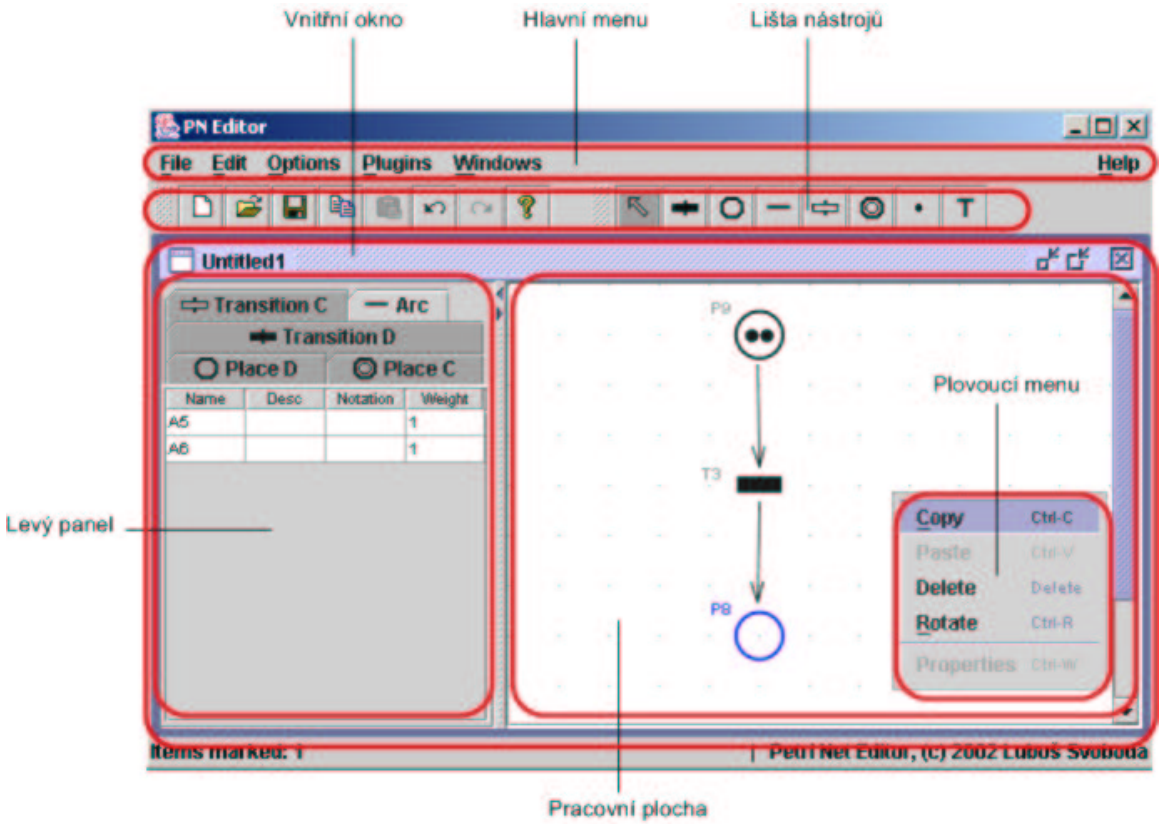
Výpis 1: Ukázka využití polymorfismu při ukládání a výběru prvků sítě ze struktury `Vector`

Třída `Net` spolu s balíkem `netobject` tvoří jádro editoru. Po jejím naprogramování jsem přistoupil k návrhu grafického prostředí, které toto jádro bude používat.

4.1.2 Návrh grafických komponent

I když v současné době existuje několik visuálních editorů pro navržení grafického vzhledu aplikace zcela jednoduchým způsobem (přetažení požadovaných komponent do okna tvořené aplikace a automatické vygenerování zdrojového kódu), dal jsem přednost klasickému návrhu napsáním kódu do zdrojových souborů. Výhodou je větší kontrola a přehlednost nad zdrojovým kódem, navíc ne všechny dostupné komponenty jsou visuálními editory podporovány. Vzhled editoru jsem navrhl s ohledem na maximální uživatelskou přívětivost. Má proto standardní rozvržení menu, jeho položek a pracovní plochy známé z prostředí operačních systémů Windows, stejně jako obrázky ikon (*obrázek 4.3*).

V horní části editoru je standardní menu s lištou nástrojů, dole je potom vnitřní okno (frame), které je rozděleno na pracovní plochu (vpravo) a tabulku na překlapávacích listech (vlevo) s jednotlivými prvky sítě (slouží k rychlé editací vlastností prvků sítě). Vnitřních oken s pracovní plochou je možné otevřít několik najednou a pracovat tak na několika sítích současně. Pro úpravu sítě a jejích vlastností je možné použít plovoucí (pop-up) menu, které se aktivuje stiskem pravého tlačítka myši. Zde je možné vybrat z několika dalších akcí pro úpravu vlastností sítě.



Obrázek 4.3: Vzhled programu PN Editor

Pro visualizaci jsem použil grafické komponenty z knihovny Swing, která obsahuje všechny základní grafické komponenty jako tlačítka, vstupní pole, posuvníky, rámy apod. Alterativně jsem mohl použít i komponenty ze starší grafické knihovny AWT, která je obsažena i ve starších distribucích Javy, její možnosti však dnes nejsou zdaleka takové jako knihovny Swing. Navíc její vývoj již byl zastaven a pokračuje se jen ve vylepšování knihovny Swing.

Samotná visuální část editoru je rozdělena do čtyř hlavních částí, kterým odpovídá rozdělení do těchto tříd:

`Menus`. Třída zděděná z komponenty `JFrame` obsahuje hlavní okno editoru s horním menu, lištou nástrojů a vnitřními rámy (`MyInternalFrame`). Jsou zde zaregistrované a implementované posluchače k jednotlivýn položkám menu a nástrojům lišty (více o posluchačích v 4.1.3).

`MyInternalFrame`. Třída zděděná z komponenty `JInternalFrame` představuje vnitřní okno editoru, které je rozděleno pomocí komponenty `JTabbedPane` na levou část s tabulkami zapouzdřené ve třídě `Tables` a pravou část s třídou `WorkSpace`.

`Tables`. Tato třída obsahuje tabulky a jejich datové modely, které se zobrazují v levém panelu.

`WorkSpace`. Třída zděděná z komponenty `JPanel` představuje vlastní pracovní plochu, ve které se kreslí Petriho síť. Jsou zde registrovány a implementovány posluchače od myši a tabulek v levém panelu. Obsahuje hlavní vykreslovací smyčku reagující na požadavky po překreslení pracovní plochy, plovoucí menu a třídy `Net, IOTransfer, ClipBoard a History` (popis těchto tříd v 4.2 a 4.3).

Pro interakci editoru s uživatelem jsem dále navrhl několik dialogů, které uživateli vypisují různé zprávy nebo umožňují editaci zobrazených hodnot. Všechny tyto dialogy jsou obsaženy v balíku `dialogs`. Tvoří ho tyto třídy:

Obrázek 4.4: Třídy s dialogy jsou v balíku `dialogs`

`AboutBox`. Zobrazí dialog s informacemi o autorovi a verzi programu.

`ColorDialog`. Zobrazí standardní dialog k výběru barvy (využívá komponentu `JColorChooser`)

`ExceptionDialog`. Pokud nastane v programu nečekaná chyba, vygeneruje se výjimka, která se zobrazí v tomto dialogu. Navíc nabízí možnost pokračovat nebo skončit program.

`LookAndFeelDialog`. Knihovna Swing nepoužívá grafické komponenty, které ji nabízí operační systém, ale používá vlastní komponenty vytvořené vývojáři Javy. Proto je možné, aby uživatel měnil vzhled své aplikace podle toho, který mu vyhovuje. Distribuce Javy do OS Windows je standardně dodávána se vzhledem aplikací pro Windows, Unix a vlastním vzhledem vytvořeným firmou Sun. Ve třídě `LookAndFeelDialog` je dialog umožňující měnit vzhled aplikace.

`OptionsDialog`. Tento dialog se zobrazí pokaždé, kdy chce uživatel upravit vlastnosti nějakého prvku sítě. Protože jsem nechtěl pro každý typ prvku sítě vytvářet různé dialogy s vlastnostmi prvku (u každého typu prvku jsou různé), vytvořil jsem univerzální dialog, kterému se pomocí parametrů předají vlastnosti, které má zobrazit. Využil jsem toho, že každou vlastnost (popisek) si daný prvek ukládá do třídy typu `Text` a uložil tyto třídy do struktury `Vector`. Potom při inicializaci dialogu pouze předám tuto strukturu se všemi popisky a dialog už zobrazí odpovídající komponenty podle typu editovaného prvku. Pro kontrolu údajů vkládaných do polí jsem použil komponentu `JFormattedTextField`, která funguje jako vstupní textové pole, navíc však lze nastavit masku, podle které probíhá kontrola vkládaných údajů.

`PluginInstallDialog`. Tento dialog umožňuje přidat do editoru nový plugin.

`TextDialog`. Dialog je používaný ve třídě `Text` pro změnu vlastností zobrazeného textu (text, velikost, font, barva).

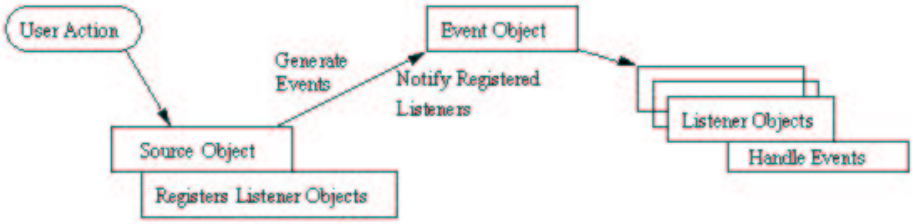
`WelcomeScreen`. Toto okno je zobrazeno při načítání a inicializaci po spuštění editoru. Pomocí funkce `setUndecorated(true)` je toto okno zobrazeno bez rámu.

Pro dialogy k otevření a uložení souboru jsem využil standardní dialogy, které poskytuje knihovna Swing, konkrétně komponentu `JFileChooser`. K filtrování obsahu aktuálního adresáře podle typu požadovaných souboru, bylo třeba vytvořit instance třídy implementující rozhraní `FileFilter` a předat je komponentě `JFileChooser` pomocí metody `addChoosableFileFilter(FileFilter filter)`. V editoru jsem použil filtr pro soubory typu *.pnml, *.jpeg a *.jar. Těmto filtrům odpovídají třídy `PnmlFileFilter`, `JpegFileFilter` a `JarFileFilter` tvořící obsah balíku `filefilter`.

4.1.3 Událostní model

K pochopení vnitřní funkce editoru a důvodů, které vedly k jeho návrhu tímto způsobem, je třeba se seznámit s událostním modelem jazyka Java, resp. jeho knihovny Swing.

Knihovna Swing spolupracuje s uživatelskými objekty na základě událostí. Pokud uživatel např. stiskne klávesu, vznikne událost (event). Aby byl uživatelský objekt informován o stavu nějaké komponenty, musí se v systému pro každý zdroj událostí (event source) registrovat svůj posluchač



Obrázek 4.5: Událostní model knihovny Swing

událostí (event listener). Systém potom tomuto posluchači bude posílat oznámení (event object) pokaždé, kdykoliv dojde k nějaké změně ve zdroji událostí.

Přehled o všech registrovaných posluchačích editoru je na obrázku 4.6. Hlavním zdrojem událostí je myš, jejíž posluchač je registrován ve třídě `WorkSpace`. Zpracovává přitom tyto akce:

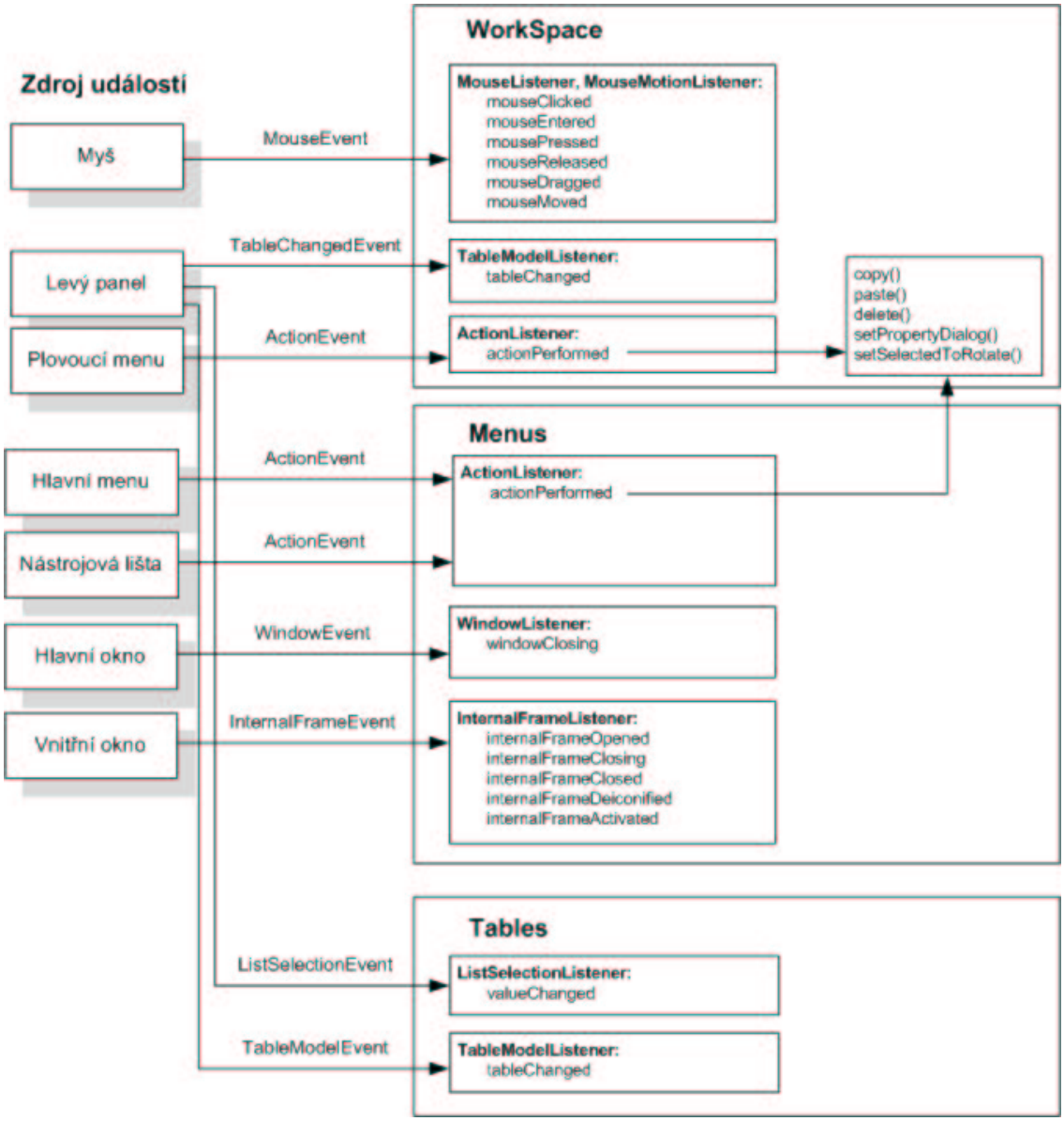
- pohyb myši - probíhá kontrola, zda se kurzor myši nachází nad nějakým prvkem sítě. Pokud ano, barevně ho označí.
- stisknutí levého tlačítka myši - dle typu vybraného nástroje se provede daná akce (vytvoření nového prvku sítě, přidání tokenu, kreslení nové hrany, zrušení označení prvků).
- stisknutí pravého tlačítka myši - inicializace a zobrazení plovoucího menu
- kliknutí (dvojité stisknutí) levého tlačítka myši - pokud je kurzor myši nad nějakým prvkem sítě, vyvolá se dialog pro nastavení jeho vlastností
- stisknutí levého tlačítka a současně pohyb myši (drag) - podle stavu sítě se provede jedna z těchto operací: kreslení obdelníku pro označení prvků, přesun prvku, přesun skupiny prvků
- uvolnění tlačítka myši - ukončení kreslení obdelníku pro označení prvků (levé tlačítko), zrušení označených prvků (pravé tlačítko)

Editor má dále zaregistrovány posluchače od položek v horním menu, nástrojové lišty a plovoucího menu na pracovní ploše. Je zřejmé, že některé funkce lze vyvolat několika způsoby. Například pro kopírování označených prvků existují celkem čtyři možné způsoby, jak vyvolat tuto funčnost:

- aktivací položky "Copy" v menu "Edit"
- stiskem ikony "Copy" na nástrojové liště
- klávesovou zkratkou <CTRL+C>
- aktivaci položky "Copy" na plovoucím menu

Pro obsluhu takových událostí je vždy napsána pouze jedna funkce, která provede danou akci. Z tohoto důvodu všechny posluchače nakonec vyvolají stejnou funkci `copy()` ve třídě `WorkSpace`. Jak ukazuje *obrázek 4.6*, události od nástrojové lišty a hlavního menu obsluhuje stejný posluchač. Pro akci kopírování označených prvků dojde v posluchači ke dvěma krokům:

1. získání aktuálního zobrazeného vnitřního okna a tím získání instance třídy `WorkSpace`



Obrázek 4.6: Registrované posluchače, jejich implementované akce, třídy, ve kterých jsou implementované a jejich zdroje událostí

2. vyvolání metody `copy()` v instanci třídy `WorkSpace`

Události generované plovoucím menu jsou obslouženy vlastním posluchačem ve třídě `WorkSpace`, který pouze zavolá odpovídající metodu, v našem případě metodu `copy()` (použije tedy stejnou metodu jako posluchač pro horní menu a nástrojovou lištu). Klávesová zkratka <CTRL+C> vyvolá stejného posluchače jako akce v hlavním menu, protože tato zkratka je svázána s položkou menu a není tedy nutné registrovat další posluchač od klávesnice.

Události od lévého panelu jsou zpracovány ve třídě `Tables`. První událostí, která se zpracovává, je událost vzniklá označením položek v tabulce. Při přijmutí této události (`ListSelectionEvent`) posluchač nejdříve najde, ve které záložce došlo k označení a poté označí stav daných prvků v kreslené síti. Na závěr posluchač vyvolá událost `TableChangedEvent`, která je zpracována ve třídě

`WorkSpace`. Tím dojde k oznámení, že v tabulce došlo ke změně (a tím i v kreslené síti) a proto se vyvolá metoda `repaint()`, která zajistí překreslení pracovní plochy. Druhou událostí, která se zpracovává ve třídě `Tables`, je událost `TableModelEvent`. Tato událost je vyvolána, pokud uživatel změní data zobrazená v tabulkách. Posluchač nejdříve zkontroluje platnost zadaných dat a poté tyto změny promítne i do kreslené sítě. Se změnou dat je zároveň generována událost `TableChangedEvent`, která zajistí překreslení sítě na pracovní ploše.

Pro položku hlavního menu "Windows" jsou dále sledovány události od vnitřního okna (událost `InternalFrameEvent`). Při otevření nového okna musí být do menu "Windows" přidána další položka s názvem okna, stejně tak jako musí být odebrána při zavření okna. Editor sleduje akce, které jsou v daném okamžiku povolené a podle toho tyto akce zpřístupňuje zvýrazněním v menu (například kopírování lze provést, jen pokud je označena nějaká část sítě). Ke správné funkci zvýraznění povolených akcí proto editor sleduje události vzniklé minimalizací a maximalizací vnitřního okna, kdy se mění aktuální editovaná síť (a spolu s ní povolené akce).

Poslední posluchač je registrován pro hlavní okno, kde sleduje jeho zavření (zasílá se událost `WindowEvent`). Při aktivaci zobrazí posluchač pro každé interní okno dialog umožňující uložení nakreslené sítě na disk.

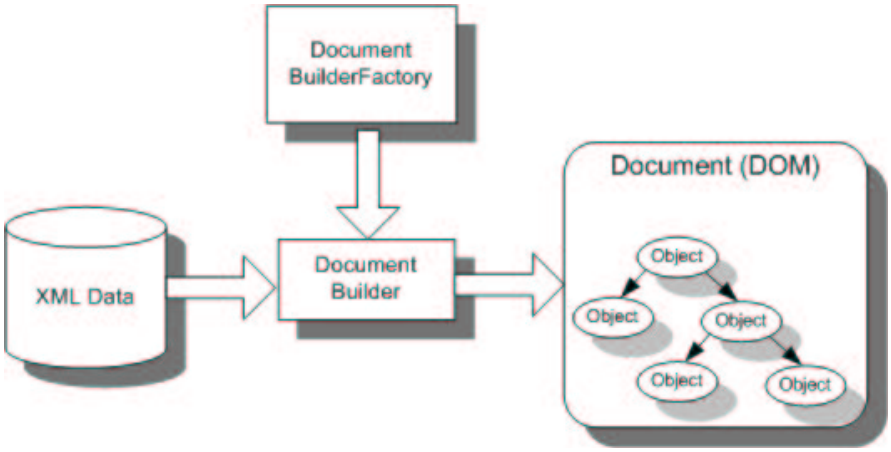
4.2 Vstupy a výstupy editoru

V této sekci popíšu úroveň implementace standardu PNML a postupy, které jsem při programování použil. Vycházel jsem přitom ze standardu popsaném v dokumentu [6], který je rozebrán v kapitole 3. Kromě vstupu a výstupu v PNML dále popíšu export nakreslené sítě do obrázku typu JPEG a také postup tisku nakreslené sítě na tiskárně.

4.2.1 Uložení sítě v PNML formátu

Pro vytvoření PNML výstupu lze použít standardní nástroje, které Java používá k práci s XML dokumenty. Tyto nástroje se nacházejí v balíku Java^{TM} API for XML Processing, který je standardní součástí Java platformy. Tento balík poskytuje dvě různá aplikační rozhraní (API) pro zpracování XML dokumentu. První z nich je Simple API for XML (SAX), které poskytuje událostmi řízené rozhraní pro analýzu XML dokumentu. Druhé aplikační rozhraní se nazývá Document Object Model (DOM) a převádí XML dokument na stromovou strukturu objektů, kterou lze procházet a prohledávat. Pro použití v PN Editoru jsem se rozhodl pro druhé rozhraní, které je vhodnější pro jednodušší XML dokumenty, není tolik programátorsky náročné a lze v něm snadněji provádět úpravy.

Obrázek 4.7 ukazuje vytvoření DOM dokumentu při zpracování XML dat. Nejdříve se vytvoří rozhraní `DocumentBuilderFactory`, které umožňuje aplikaci získat analyzátor (`DocumentBuilder`). Tento analyzátor poté zpracuje XML data a vytvoří strom objektů DOM dokumentu. Při vytváření nového XML dokumentu se postup liší pouze v tom, že se analyzátoru nepředává hotový XML dokument, ale vytvoří se s jeho pomocí nový. Tento postup ukazuje *výpis 2*.



Obrázek 4.7: Postup vytvoření DOM dokumentu při zpracování XML dat

```
//1. create DocumentBuilderFactory and set properties
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setValidating(false);
dbf.setIgnoringComments(true);
dbf.setIgnoringElementContentWhitespace(true);

//2. get a new DocumentBuilder
DocumentBuilder db = null;
try {
    db = dbf.newDocumentBuilder();
} catch (ParserConfigurationException pce) {...}

//3. create a new plain XML document
Document doc = db.newDocument();

//4. create PNML document
Element elem_pnml = doc.createElement("pnml"); //root element
doc.appendChild(elem_pnml);
... (save net)

//5. save to output stream
XmlDocument xmlDoc = (XmlDocument) doc;
ByteArrayOutputStream result = new ByteArrayOutputStream();
try {
    xmlDoc.write(result);
} catch (Exception e) {...}
```

Výpis 2: Vytvoření nového PNML dokumentu

PNML standard popsaný v kapitole 3 ukazuje, jak uložit standardní Petriho síť do PNML formátu. Zapomíná ovšem na další vlastnosti Petriho sítí, které už nejsou zcela běžné, jako je např. zpoždění a priorita u přechodů. Zmiňuje se pouze o pravidlu, které říká, že každý neznámý tag, který je v PNML dokumentu nalezen, je popisek. Tohoto pravidla jsem tedy použil pro uložení informací o prvcích sítě, které nejsou přímo ve standardu PNML a ke každému objektu přidal další popisky reprezentující jeho specifické vlastnosti. Všechny popisky pro každý prvek sítě jsou shrnuty v tabulce 4.1.

Dalším problémem standardu PNML je, že jeho autoři například vůbec nepočítali s možností

Prvek sítě	Jméno objektu	Popisek	Význam	Ukázka reprezentace v PNML

diskrétní místo
spojité místo
	place	name
		description
		notation
		initialMarking
			jméno prvku
			bližší popis prvku
			bližší popis prvku
			počáteční marking
				`<place id="P0">`
				`<graphics>...</graphics>`
				`<name>...</name>`
				`<description>...</description>`
				`<notation>...</notation>`
				`<initialMarking>...</initialMarking>`
				`</place>`

diskrétní přechod	transition	name
		description
		notation
		delay
		priority
			jméno prvku
			bližší popis prvku
			bližší popis prvku
			zpoždění
			priorita
				`<transition id="T0">`
				`<graphics>...</graphics>`
				`<name>...</name>`
				`<description>...</description>`
				`<notation>...</notation>`
				`<delay>...</delay>`
				`<priority>...</priority>`
				`</transition>`

spojitý přechod	transition	name
		description
		notation
		speed
			jméno prvku
			bližší popis prvku
			bližší popis prvku
			rychlost
				`<transition id="CT0">`
				`<graphics>...</graphics>`
				`<name>...</name>`
				`<description>...</description>`
				`<notation>...</notation>`
				`<speed>...</speed>`
				`</transition>`

hrana	arc	name
		description
		notation
		inscription
			jméno prvku
			bližší popis prvku
			bližší popis prvku
			váha hrany
				`<arc id="A0" source="P0" target="T0">`
				`<graphics>...</graphics>`
				`<name>...</name>`
				`<description>...</description>`
				`<notation>...</notation>`
				`<inscription>...</inscription>`
				`</arc>`

Tabulka 4.1: Popisky prvků sítě, jejich význam a reprezentace v PNML

hybridních sítí, možností změnit barvy prvku, použití písem různých druhů, velikostí, barev a některé další drobnosti. Ve standardu je sice zahrnut element `toolSpecific`, kterým lze výše popsané nedostatky odstranit, je to však za cenu funkčnosti pouze v jednom konkrétním nástroji. Pro správnou funkci editoru jsem tedy musel dovnitř elementu `toolSpecific` přidat několik vlastních tagů, které poskytují další informace. Zde je přehled změn:

1. **Popisky**. Každý popisek je vytvořen tak, jak je popsáno v kapitole 3, navíc obsahuje element `toolSpecific`, ve kterém jsou informace o barvě textu, použitém fontu, jeho velikosti a informaci o tom, zda tento popisek má být v editoru viditelný. Podrobnosti ukazuje *výpis 3*.

```
<name>
    <value>A1</value>
    <graphics>
        <offset x="-16" y="-16" />
    </graphics>
    <toolSpecific tool="PNEditor" version="1.0">
        <color r="128" g="128" b="128" />
```

```
            <font name="Ariel" size="10" />
            <show value="true" />
        </toolSpecific>
    </name>
```

Výpis 3: Přidání elementu `toolSpecific` do popisku name.

2. **Objekty**. Podobně jako u popisků byl navíc vložen element `toolSpecific` s barvou objektu, navíc je zde další tag `type` rozhodující o tom, jestli je daný prvek diskrétní nebo spojitý. Je-li hodnota jeho atributu `value="D"`, jedná se o diskrétní prvek, hodnota `value="C"` indikuje spojitý prvek. Některé prvky nesou v elementu `toolSpecific` ještě další informace:

   (a) Transition. Přechod má další element `rotate` udávající otočení o 90 stupňů. Pokud je jeho atribut `value="true"`, je přechod otočen o 90 stupňů.

   (b) Arc. K hraně jsem do elementu `toolSpecific` přidal další element `centre_moved`. Tento element říká, jak se má daná hrana vykreslit. Pokud je hodnota atributu `value="false"`, vykreslí se jako rovná přímka (bod uprostřed hrany leží na nejkratší spojnici prvků, kterých se hrana dotýká). Pokud je hodnota atributu `value="true"`, vykreslí se hrana tak, aby procházela bodem definovaném v elementu graphics.

Všechny výše popsané úpravy v generovaném PNML kódu ilustruje následující příklad obrázku Petriho sítě s vygenerovaným PNML zápisem. Pro úsporu místa jsem uvedl obsah popisku jen pro jméno prvního místa, ostatní popisky se liší pouze hodnotami, struktura je stejná.

Obrázek 4.8: Síť pro ukázku vygenerovaného PNML (*výpis 4*)

```
<?xml version="1.0" encoding="UTF-8"?>
<pnml>
  <net id="n1">
    <name>Test net</name>
    <place id="CP1">
      <graphics>
        <position x="349" y="58" />
      </graphics>
      <name>
        <value>CP1</value>
        <graphics>
          <offset x="-1" y="16" />
        </graphics>
        <toolSpecific tool="PNEditor" version="1.0">
          <color r="128" g="128" b="128" />
          <show value="true" />
          <font name="Ariel" size="10" />
```

```
        </toolSpecific>
      </name>
      <description>...</description>
      <notation>...</notation>
      <initialMarking>...</initialMarking>
      <toolSpecific tool="PNEditor" version="1.0">
        <color r="0" g="0" b="0" />
        <type value="C" />
      </toolSpecific>
    </place>
    <transition id="CT1">
      <graphics>
        <position x="186" y="65" />
      </graphics>
      <name>...</name>
      <description>...</description>
      <notation>...</notation>
      <speed>...</speed>
      <toolSpecific tool="PNEditor" version="1.0">
        <color r="0" g="0" b="0" />
        <type value="C" />
        <rotate value="false" />
      </toolSpecific>
    </transition>
    <arc id="A1" source="CT1" target="CP1">
      <graphics>
        <position x="284" y="114" />
      </graphics>
      <name>...</name>
      <description>...</description>
      <notation>...</notation>
      <inscription>...</inscription>
      <toolSpecific tool="PNEditor" version="1.0">
        <color r="0" g="0" b="0" />
        <centre_moved value="true" />
      </toolSpecific>
    </arc>
  </net>
</pnml>
```

Výpis 4: Vygenerovaný PNML popis odpovídající síti na *obrázku 4.8*.

4.2.2 Načtení sítě z PNML souboru

Pro načtení uložené sítě jsem použil postup popsaný v kapitole 4.2.1. Jak jsem také ukázal, musel jsem k objektům a popiskům přidat elementy `toolSpecific` s vlastním obsahem k uložení některých specifických vlastností sítě. U jiných editorů bude obsah těchto tagů jiný nebo budou zcela chybět. Proto pro načtení a správné zpracování souboru PNML vytvořeném v jiném editoru, jsem zpracování ošetřil tak, aby chybějící elementy `toolSpecific` nezpůsobily chyby zabraňující

načtení sítě. Místo chybějících elementů `toolSpecific` se použijí defaultní hodnoty (stejné hodnoty jako při vložení nového prvku sítě).

4.2.3 Export sítě do obrázku JPEG

K exportu nakreslené sítě do obrázku typu JPEG jsem použil bálík `com.sun.image.codec.jpeg` dodávaný k Java2D API, konkrétně třídu `JPEGImageEncoder` poskytující funkce pro převod obrázku (`BufferedImage`) nakresleného pomocí Java2D API do výstupního proudu (`OutputStream`). Postup exportu ukazuje *výpis 5*.

```
public static ByteArrayOutputStream getJPEGstream(Net net, Dimension dim) {
    ByteArrayOutputStream out = null;
    try {
        //1. create clear image
        BufferedImage bi =
            new BufferedImage(dim.width, dim.height, BufferedImage.TYPE_INT_RGB);
        Graphics2D g2 = (Graphics2D) bi.getGraphics();
        g2.setBackground(Color.WHITE);
        g2.clearRect(0, 0, dim.width, dim.height);
        g2.setColor(Color.BLACK);

        //2. render net to image
        for (int i = 0; i < net.size(); i++) {
            AbstractNetObject agi = net.getANO(i);
            agi.draw(g2);
        }

        //3. convert image to JPEG
        out = new ByteArrayOutputStream();
        JPEGImageEncoder encoder =
            JPEGCodec.createJPEGEncoder(out); //create encoder
        JPEGEncodeParam param = encoder.getDefaultJPEGEncodeParam(bi);
        param.setQuality(1.0f, false);
        encoder.setJPEGEncodeParam(param);  //set parameters
        encoder.encode(bi);     //encode
        out.flush();
        out.close();

    } catch (Exception e) {...}
    return out;
}
```

Výpis 5: Metoda převádějící nakreslenou sít do obrázku ve formátu JPEG. Parametry: síť popsaná ve třídě `Net` a velikost výsledného obrázku (`Dimension`). Výsledek: obrázek JPEG uložený v bytovém výstupním proudu (`ByteArrayOutputStream`).

4.2.4 Tisk

Pro tisk nakreslené sítě jsem použil aplikační rozhraní Java^{TM} Print Service API. Využil jsem toho, že v poslední verzi Javy (JDK 1.4) je možné tisknout obrázky typu JPEG a proto nejdříve celou síť vykreslím do obrázku typu JPEG a poté vytisknu podle následujícího postupu:

1. Vybrat formát, ve kterém jsou tisknutá data (nastavit `DocFlavor` na obrázek JPEG)
2. Vytvořit množinu atributů k nastavení tisku (`PrintRequestAttributeSet`)

3. Najít službu, která může vytisknout úlohu ve specifikovaném formátu (`DocFlavor`) a množinou atributů (`PrintRequestAttributeSet`)
4. Vytvořit objekt (`Doc`) zapouzdřující data k tisku, množinu atributů a formát dat
5. Získat úlohu tisku (`DocPrintJob`) ze služby, která může dokument vytisknout
6. Zavolat metodu `print()`

4.3 Implementace ostatních funkcí

Každý editor by měl kromě vkládání nových prvků také podporovat operace kopírování a vkládání. Tuto funkčnost jsem implementoval do třídy `ClipBoard`. Operace kopírování a vkládání fungují podle následujícího postupu:

1. kopírování:
   (a) posluchač vyvolá metodu `copyMarked(Net net)` instance třídy `ClipBoard`. Tato funkce nalezne označené prvky sítě a zavolá jejich metodu `copyProperty` (abstraktní metoda zděděná z `AbstractNetObject`)
   (b) metoda `copyProperty` zkopíruje do nově vytvořeného prvku sítě vlastnosti kopírovaného prvku, zároveň do proměnné `ID_old` zaznamená číslo ID kopírovaného prvku
   (c) nově vytvořený prvek je vložen do sítě (třídy `Net`), která je součástí schránky
2. vkládání:
   (a) posluchač vyvolá metodu `pasteCliped(Net net)`, která překopíruje obsah schránky do aktuálně používané sítě (parametr `net`). Postup je stejný jako při kopírování prvku.
   (b) protože se vložením prvku do nové pracovní plochy změnilo jeho ID, které není stejné jako u původně kopírovaného prvku, musí se u hran provést aktualizace z jakého ID a do jakého ID hrana směřuje (využije se ID původního prvku uložené v proměnné `ID_old`.

Dále jsem pro příjemnější práci s editorem implementoval do třídy `History` funkce zpět(undo) a znovu(redo). Využil jsem toho, že celou síť lze uložit do formátu PNML a poté ji znovu načíst. Proto po každé změně na pracovní ploše nebo levém panelu se celý stav pracovní plochy uloží ve formátu PNML do datové struktury `Vector`. Potom operace zpět a znovu znamenají pouze načtení sítě ze správné pozice ze struktury `Vector` a použití stejných metod pro konverzi z PNML jako při načítání ze souborů. Protože uložení nakreslené sítě do PNML je časově náročnější a při každém ukládání stavu sítě by docházelo k nepříjemnému čekání na dokončení této operace, přesunul jsem ukládání stavu sítě do samostatného vlákna. Délka historie je nastavena na 30 stavů.

K editoru jsem vytvořil stručnou nápovědu, kterou lze vyvolat klávesou F1 nebo položkou menu "Contents" v menu "Help". K vytvoření nápovědy jsou použil balík JavaHelp^{TM} 1.0, který prozatím není součástí Java platformy, lze ho stáhnou zde [12]. Umožňuje vytvořit nápovědu se zobrazením jejího obsahu, indexem slov, vyhledáváním a odkazy mezi jednotlivými stránkami (i hypertextovými). Samotná nápověda je vytvořena z HTML dokumentů a při její tvorbě jsem postupoval podle následujícího postupu.

1. vytvořit obsah samotné nápovědy ve formátu HTML (např. pro jednostránkovou nápovědu vytvořit soubor se jménem `hello_world_help.html`):

```
<html>
    <body>hello world</body>
</html>
```

2. vytvořit mapovací soubor propojující samotné soubory nápovědy se zobrazenými položkami nápovědy a uložit ho jako `help.jhm`:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
    <!DOCTYPE map
        PUBLIC "-//Sun Microsystems Inc.//DTD JavaHelp Map Version 1.0//EN"
               "http://java.sun.com/products/javahelp/map_1_0.dtd">
<map version="1.0">
    <mapID target="item1" url="hello_world_help.html" />
</map>
```

3. vytvořit soubor popisující položky nápovědy a uložit ho jako `helpTOC.xml`:

```
<?xml version="1.0" encoding="ISO-8859-1"  ?>
<!DOCTYPE toc
  PUBLIC "-//Sun Microsystems Inc.//DTD JavaHelp TOC Version 1.0//EN"
         "http://java.sun.com/products/javahelp/toc_1_0.dtd">
<toc version="1.0">
    <tocitem text="First Help chapter" image="chapter" >
    <tocitem text="Hello world item" target="item1" image="topic" />
    </tocitem>
</toc>
```

4. vytvořit soubor popisující celkový vzhled nápovědy a rozvržení jednotlivých položek pod témata nápovědy a uložit ho jako `help.hs`:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
    <!DOCTYPE helpset
    PUBLIC "-//Sun Microsystems Inc.//DTD JavaHelp HelpSet Version 1.0//EN"
           "http://java.sun.com/products/javahelp/helpset_1_0.dtd">
<helpset version="1.0">
    <!-- title -->
    <title>Help</title>
    <!-- maps -->
    <maps>
        <homeID>item1</homeID>
        <mapref location="help.jhm" />
    </maps>
    <!-- views -->
    <view>
        <name>TOC</name>
        <label>Table Of Contents</label>
        <type>javax.help.TOCView</type>
        <data>helpTOC.xml</data>
    </view>
</helpset>
```

5. pro zobrazení nápovědy se v programu vytvoří nová položka menu a zaregistruje se její posluchač, který vyvolá nápovědu:

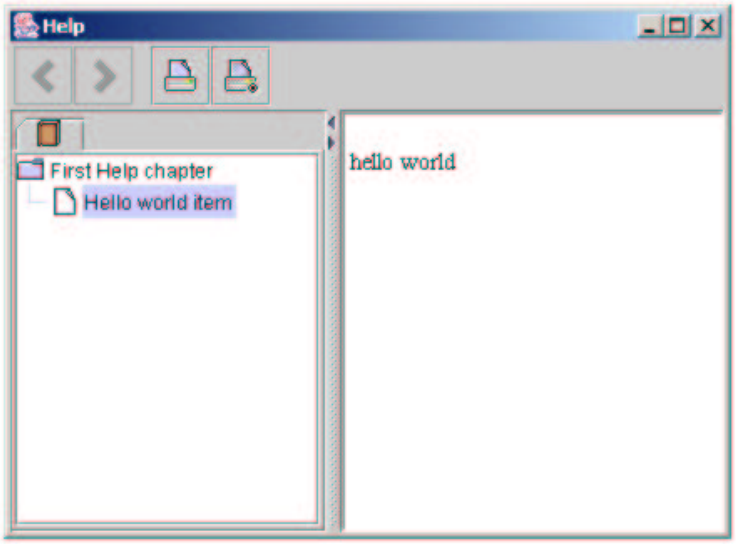
```
import javax.help.*;
...
JMenuItem miHelp;
...
private registerJavaHelp();
    try {
        miHelp = new JMenuItem("Contents");     //create new menu item

        URL url = new URL("file:help.hs");      //load help file
        HelpSet hs = new HelpSet(null, url);
        HelpBroker hb = hs.createHelpBroker();

        miInd.addActionListener(                //register listener
            new CSH.DisplayHelpFromSource(hb)   //show help
        );

    } catch (Exception ex) {...};
}
...
```

6. Výsledkem popsaného postupu je jednoduchá nápověda na *obrázku 4.9*.



Obrázek 4.9: Jednoduchá nápověda vytvořená postupem v 4.3

4.4 Systém pluginů

Jak jsem uvedl v úvodu, editor je určen zejména pro práci s MTPN. Jako výsledek jednoho z výpočtů toolboxu je graf dosažitelnosti určený maticí dosažitelných stavů a maticí popisující propojení stavů hranami. Protože visualizační schopnosti Matlabu pro kreslení grafů jsou velmi omezené, vytvořil jsem krátký program v jazyku Java, který dokáže graf dosažitelnosti vykreslit.

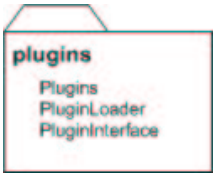
Tento program jsem integroval do editoru pomocí systému pluginů (zásuvných modulů), který popíšu v této části.

Plugin si lze představit jako program, který nějakým způsobem využívá nakreslenou síť nebo má nějakou jinou souvislost s Petriho sítěmi. Tento plugin nemusí být s editorem implicitně dodáván a lze jej do editoru později doinstalovat a také odinstalovat. Výhody, které z toho plynou lze shrnout do několika bodů:

- editor může být rozšiřován o další funkce
- pokud uživateli chybí nějaká funkce, může si ji sám vytvořit (dokumentace k editoru a jeho rozhraní je dostupná) a nemusí tak spoléhat na autora programu
- využitím pluginů vznikne otevřený projekt použitelný pro velkou skupinu uživatelů

4.4.1 Realizace systému pluginů

Realizace systému pluginů se opírá o balík `plugin`, kde jsou uloženy třídy pro práci s pluginy(`Plugins, PluginLoader, PluginInterface`). Nejdůležitější je rozhraní `PluginInterface`, pomocí něhož je možné plugin nahrát za běhu programu a potom s ním komunikovat. O nahrání pluginu se stará třída `PluginLoader`, která plugin načte a vrátí instanci rozhraní `PluginInterface`. Třída `Plugins` se poté postará o inicializaci a spuštění pluginu.



Obrázek 4.10: Obsah balíku `plugins`

Popis rozhraní `PluginInterface` a postup vytvoření nového pluginu je popsán v uživatelské dokumentaci editoru v části 5.5.1. Pro další výklad jsou důležité tyto body:

- plugin musí implementovat rozhraní `PluginInterface`
- celý plugin musí být zabalen pomocí programu *jar* a musí k němu být přibalen manifestační soubor udávající název třídy implentující rozhranní `PluginInterface` (více o *jar* a manifestačním souboru v části 4.5)

Instalace pluginu probíhá pomocí dialogu `PluginInstallDialog` v balíku `dialogs`. Po načtení souboru *.jar probíhá kontrola, zda daný archív obsahuje třídu implementující rozhraní `PluginInterface`. Pokud ano, vytvoří se nová instance třídy obsahující `PluginInterface` a zjistí se jméno pluginu. Při načítaní pluginu jsem využil funkcí z balíku `java.net` a použitý postup je ve *výpisu 6*.

```
...
public class PluginLoader {

    private URL url;
    private String namePluginInterfaceClass = null;

    //return plugin from file specified by jarFilePath
    public PluginInterface getPluginInterface(String jarFilePath)
        throws Exception{

        //1. load jar file
        url = new URL("jar", "", jarFilePath + "!/");
        //2. get name of class implementing PluginInterface
        namePluginInterfaceClass = getNamePluginInterfaceClass();
        //3. make new instance of PluginInterface
        PluginInterface pluginIF =
            (PluginInterface)getPluginObject(namePluginInterfaceClass);
        return pluginIF;
    }

    private String getNamePluginInterfaceClass() throws IOException {
        //open jar file
        JarURLConnection uc = (JarURLConnection)url.openConnection();
        Manifest mani = uc.getManifest();           //get manifest file
        Attributes atrib = mani.getMainAttributes();    //get attributes
        String name = atrib.getValue("Main-Class");   //get class name
        return name;
    }

    private Object getPluginObject(String namePluginInterfaceClass)
        throws Exception{

        URLClassLoader ucl =
            new URLClassLoader(new URL[] { url }); //load jar file
        Class cl  =
            ucl.loadClass(namePluginInterfaceClass);  //get class from jar
        Object plugin = cl.newInstance();   //make new plugin
        return plugin;
    }
}
```

Výpis 6: Načtení pluginu z jar archívu

Pokud uživatel souhlasí s nalezeným pluginem, uzavře dialog (tlačítko "OK") a plugin se předá třídě `Plugins`. Ta přidá jméno pluginu do nabídky menu "Plugins", zaregistruje jeho posluchače a přidá záznam do konfiguračního souboru pluginů, aby při příštím načtení byl plugin k dispozici. Konfigurační soubor pluginů je XML dokument popisující momentálně instalované pluginy a je vytvořen pomocí stejného postupu, jako PNML výstup v 4.2. Soubor je pojmenovaný `plugins_config.xml` a je uložen v aktuálním pracovním adresáři editoru. Jeho strukturu pro jeden instalovaný plugin ukazuje *výpis 7*. Obsahuje jméno pluginu, cestu odkud lze plugin načíst a třídu implementující rozhraní `PluginInterface`.

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <plugin>
        <name>Graf of reachable marking</name>
        <jar-path>file:plugins\reach_graph.jar</jar-path>
        <main-class>graph.Main</main-class>
```

```
    </plugin>
</config>
```

Výpis 7: Struktura konfiguračního souboru pro pluginy

Samotné spuštění pluginu probíhá zavoláním metody `executePlugin` ve třídě `Plugins`. V této metodě se nejdříve vytvoří nová instance pluginu (stejný postup jako při instalaci pluginu) a poté se provede inicializace a následně spuštění pluginu (*výpis 8*).

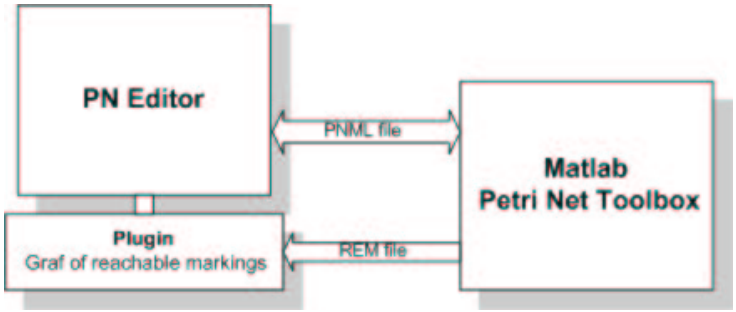
```
public void executePlugin(String name, Net net){
    Plugin plugin = getPlugin(name);
    try {
        PluginLoader pl = new PluginLoader();
        PluginInterface m = pl.getPluginInterface(plugin.classpath);
        m.initialize(net);
        m.runPlugin(net);
    } catch (Exception e) {...};
}
```

Výpis 8: Inicializace a spuštění pluginu

Odinstalování pluginu probíhá zobrazením dialogu `PluginUninstallDialog` v balíku `netobject`. Uživatel si vybere daný plugin a stiskne tlačítko OK. Poté je plugin odstraněn z menu Plugins, vnitřního seznamu pluginů a je také aktualizován konfigurační soubor `plugins_config.xml`.

4.4.2 Plugin zobrazující graf dosažitelnosti

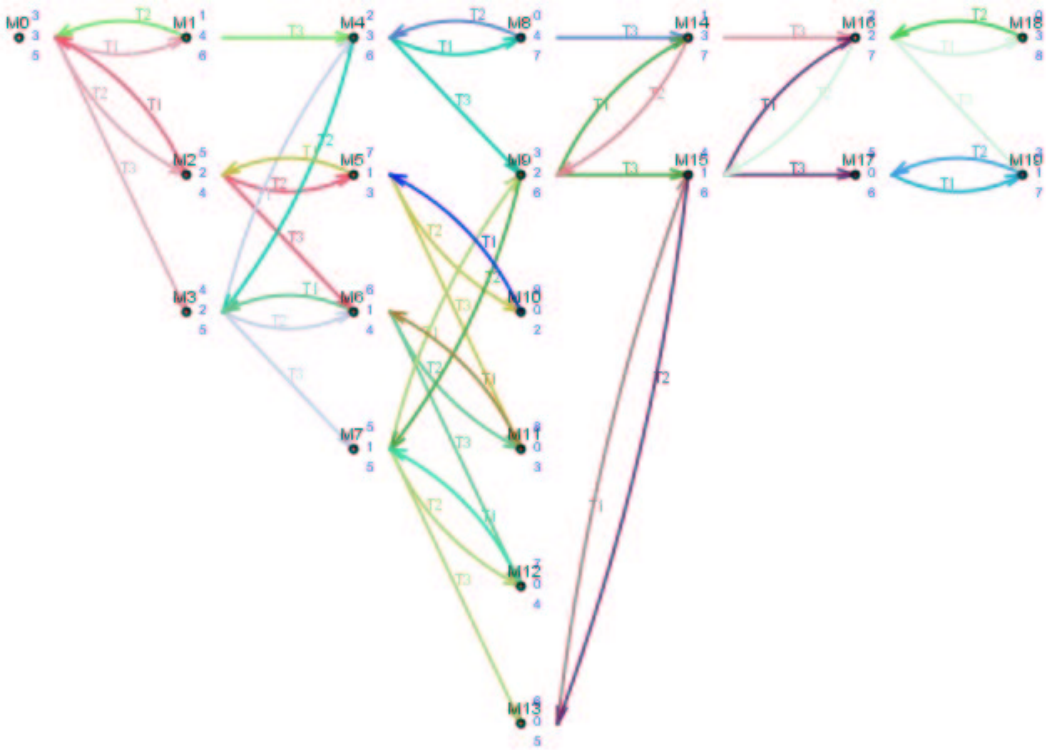
Pro demonstraci systému pluginů jsem vytvořil plugin pro zobrazení grafu dosažitelných stavů Petriho sítě (popis grafu dosažitelnosti je v kapitole 1.4.4). Vstupem pro plugin je soubor s koncovkou *.rem popisující strukturu grafu, výstupem je okno s vykresleným grafem dosažitelnosti.



Obrázek 4.11: Tok dat mezi editorem a toolboxem

Pro vytvoření souboru popisující strukturu grafu jsem vytvořil funkci matlabu `graph2rem(A, RM, filename)`, která do souboru s názvem `filename` zapíše obsah matic **A** a **RM** popisující graf dosažitelnosti. Prvek matice $A(i,j) = t$ říká, že ze stavu i směřuje do stavu j hrana, přičemž stav j vznikl přeskočením přechodu T_t. Matice **RM** má rozměr m x n a popisuje jednotlivé stavy sítě. Každému sloupci odpovídá jeden stav sítě, síť tedy bude mít n stavů a m míst. Prvek matice $RM(i,j) = p$ říká, že marking místa P_i ve stavu j je roven p. Formát souboru `filename` dokumentuje výpis funkce `graph2rem` v příloze 7.2. První čtyři bajty zabírá hlavička, kde je vždy po dvou bajtech zapsán rozměr matic (v pořadí **A**, **RM**). Následuje datový blok skládající se z matic **A**, **RM** (zapsané postupně za sebou po sloupcích).

Samotný plugin je vytvořen v balíku `graph`. Graficky je síť je reprezentována třídami `Arc` a `State`, ve kterých jsou metody pro zobrazení hran a stavů. Celá síť je potom uložena ve třídě `Net` pomocí seznamu hran a stavů (ve struktuře `Vector`). O komunikaci s editorem se stará třída `Main`, která implementuje potřebné rozhraní `PluginInterface`. Po spuštění načte soubor *.rem, který předá třídě `Net`. Ta ho nejdříve převede do dvou dvourozměrných polí odpovídající maticím **A** a **RM** a poté z těchto polí vytvoří seznam hran a stavů včetně jejich grafického rozmístění. Po inicializaci je síť předána třídě `Workspace`, která zajistí vykreslení sítě na obrazovku. Kromě výše popsaných tříd ještě balík `graph` obsahuje třídu `RemFileFilter`, která se používá k filtrování obsahu adresáře při načítání *.rem souboru. Výsledkem je graf dosažitelnosti na *obrázku 4.12*.



Obrázek 4.12: Graf dosažitelnosti zobrazený pluginem

4.5 Vytvoření instalačního souboru

Překladem zdrojových souborů vznikne více než 100 souborů, které se musí distribuovat. Pro snadnější distribuci a zmenšení velikosti výsledného produktu jsem použil utilitu *jar* (Java Archive Tool) dodávanou k java platformě. Tato utilita zkomprimuje obsah zadaného adresáře a vytvoří archív s příponou *.jar. Archív `pneditor.jar` použitý pro distribuci editoru jsem vytvořil komprimací adresáře `cz` (obsahuje přeložené třídy editoru) pomocí následujícího zápisu na příkazové řádce:

```
jar cmf Manifest.mf pneditor.jar cz
```

Takto vytvořený archív lze předat virtuálnímu stroji, který ho spustí. Aby virtuální stroj našel spustitelnou třídu, musí být do archívu přiložen tzv. manifestační soubor (`manifest.mf`), ve kterém musí být specifikována spustitelná třída.

```
Manifest-Version: 1.0
Created-By: 1.4.1_01 (Sun Microsystems Inc.)
Main-Class: cz.cvut.k335.pneditor.Main
```

Výpis 9: Obsah manifestačního souboru (`manifest.mf`)

Editor pro svůj běh vyžaduje, aby na hostitelském počítači byl k dispozici javovský virtuální stroj (JVM). Protože ne každý počítač je takto vybaven a pokud ano, nemusí mít aktuální verzi 1.4 nebo novější, na které funguje, rozhodl jsem distribuovat editor spolu s funkčním virtuálním strojem (pro OS Windows). Samotný virtuální stroj lze stáhnout na [13]. Do instalace editoru jsem zahrnul celý adresář, který se po instalaci JVM vytvoří. Protože balík `JavaHelp` není standartní součásti java platformy, musel jsem pro správnou funkci nápovědy nakopírovat do adresáře `../jre/lib/ext` soubor `jhall.jar`, který obsahuje rozhraní `JavaHelp`. Zahrnutím všech potřebných souborů pro běh editoru vznikne tato adresářová struktura:

```
..
help            <adr>
images          <adr>
jre             <adr>
plugins         <adr>
pneditor.jar
plugins_config.xml
```

Adresář `help` obsahuje soubory potřebné pro zobrazení nápovědy, `images` obsahuje obrázky, které editor potřebuje pro svůj běh (ikony, kursory), `jre` obsahuje virtuální stroj a v adresáři `plugins` jsou distribuovány pluginy. Soubory `pneditor.jar` a `plugins_config.xml` byly popsány výše. Aby se editor nemusel spouštět přes příkazovou řádku nebo dávkový soubor, vytvořil jsem v jazyku C++ krátký program (*výpis 10*), který zajistí spuštění editoru pomocí spustitelného souboru s příponou exe. Tento soubor (PNEditor.exe) jsem umístil do stejného adresáře jako jar archív s editorem.

```
#include <vcl.h>
WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevIns, LPSTR lpCmdLin, int nCmdShow)
{
  ShellExecute(NULL, "open", "jre\\bin\\java.exe", "-jar pneditor.jar",
    NULL, SW_SHOWNORMAL);
  return 0;
}
```

Výpis 10: Zdrojový kód napsaný v jazyku C++ pro spuštění editoru

Z důvodu zmenšení velikosti souborů nutných pro distribuci a snadnější instalaci jsem použil program *Inno Setup* [14]. Tento volně šiřitelný program vytvoří jeden spustitelný soubor pro instalaci aplikace, ve kterém je zkomprivána aplikace a visuální průvodce instalací. Zároveň přidá do startovací nabídky OS Windows ikony pro rychlé spuštění aplikace a její odinstalaci. Skript použitý pro sestavení instalačního souboru PN Editoru je v příloze 7.3.

Pro ostatní OS je editor distribuován bez virtuálního stroje v archívu typu GZIP (soubor pneditor.tgz). Struktura archívu je obdobná jako ve verzi pro Windows, neobsahuje však adresář `jre` s virtuálním strojem a soubor `jhall.jar` (balík `JavaHelp`) je přesunut do hlavního adresáře.

Kapitola 5

Uživatelský manuál programu PN Editor

5.1 Co je PN Editor

PN Editor je program určený pro tvorbu a editaci Petriho sítí v uživatelsky přívětivém prostředí. K základním rysům editoru patří:

- visuální tvorba a editace Petriho sítí
- podpora diskrétních, spojitých a hybridních Petriho sítí
- podpora časovaných Petriho sítí
- podpora prioritních Petriho sítí
- import a export sítě ve formátu PNML
- export sítě jako obrázku ve formátu JPEG
- systém rozšiřujících modulů (pluginů)
- platformová nezávislost

Editor byl vyvinut jako diplomová práce na katedře řídící techniky elektrotechnické fakulty ČVUT v Praze pro podporu Petri Net Matlab Toolboxu vyvíjeného pracovníky katedry.

5.2 Instalace a spuštění editoru

Pro operační systém Windows 9x, 2000 a XP je editor distribuován v jednom spustitelném souboru (`setup.exe`). Spuštěním souboru `setup.exe` se zobrazí instalátor umožňující nainstalovat editor do uživatelem specifikovaného adresáře a vytvořit ikony pro spuštění a odistalování editoru do nabídky menu Start. Na disku je třeba mít k dispozici přibližně 40MB volného místa. Samotný editor lze spustit kliknutím na soubor `pneditor.exe` nebo na ikonu PN Editor v menu Start.

Pro ostatní operační systémy je editor distribuován v souboru *.tgz, jenž uživatel rozbalí do požadovaného umístění editoru. Na disku je třeba mít k dispozici přibližně 1MB volného místa. Podmínkou pro spuštění je nainstalovaný virtuální stroj (JVM), a to ve verzi 1.4 nebo vyšší (poslední verzi lze stáhnout na [13]). Editor se spouští v pracovním adresáři příkazem:

```
java -classpath jhall.jar;pneditor.jar cz.cvut.k335.pneditor.Main
```

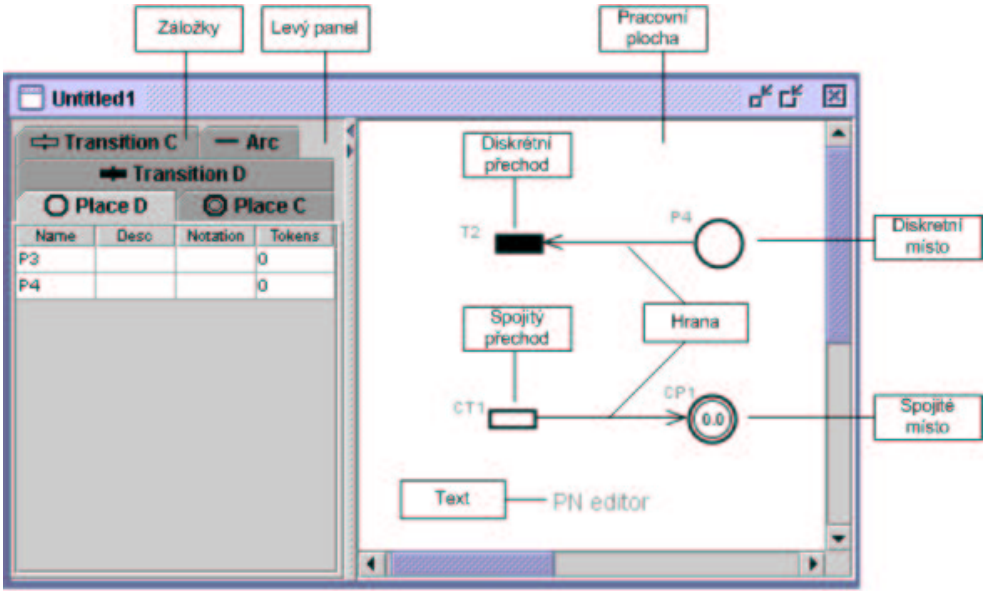
5.3 První kroky s editorem

Prostředí editoru je rozděleno na čtyři části:

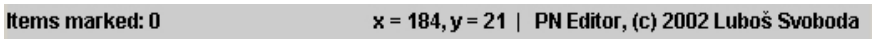
1. hlavní menu v horní části

2. lištu nástrojů pod hlavním menu

3. vnitřní okno s levým panelem a pracovní plochou



4. informační lištu ve spodní části editoru



5.3.1 Vytvoření nové Petriho sítě

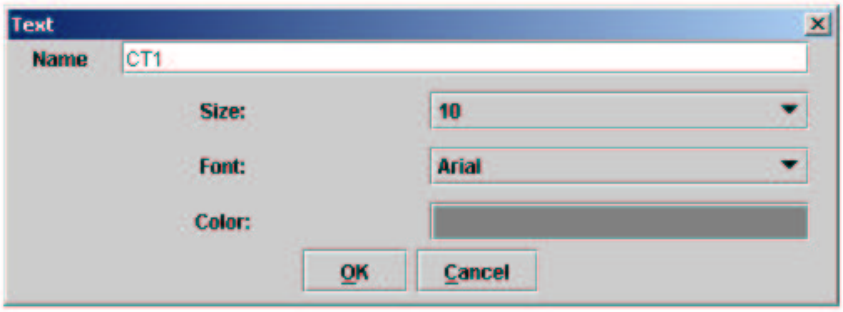
1. Pro nakreslení nové Petriho sítě je třeba nejprve otevřít nové okno s pracovní plochou. Nové okno se vyvolá v menu "File" poklepáním na položku "New" nebo aktivací klávesové zkratky <CTRL+N> nebo stiskem ikony "Nové okno" na nástrojové liště.

2. Síť se kreslí pomocí nástrojů zobrazených v pravé části nástrojové lišty. K dispozici jsou nástroje pro kreslení diskrétní i spojitých sítí (místa, přechody). Vyberte požadovaný prvek sítě a stiskněte levé tlačítkem myši (LTM) na pracovní ploše. Na pracovní ploše se zobrazí značka odpovídajícího prvku.

3. K propojení prvků sítě pomocí hran slouží nástroj s ikonou "Hrana". Zvolte tento nástroj, přesuňte kurzor myši nad prvek, kde má hrana začínat a stiskněte LTM. Nyní přesuňte myš nad prvek, kde má hrana končit a opět stiskněte LTM. Pro zrušení kreslení hrany stiskněte pravé tlačítko myši (PTM).

4. Pro zvětšení počtu tokenů v diskrétním místě přepněte na nástroj s ikonou "Vložit token" a přesuňte myš nad diskrétní místo. Stiskem levého tlačítka myši se ve zvoleném místě zvětší počet tokenů o jeden token. Pro zmenšení počtu tokenů uvedený postup opakujte se stisknutou klávesou Shift.

5. Pro vložení textu pro popis kreslené sítě přepněte na nástroj s ikonou "Text", přesuňte kurzor do požadovaného umístění textu na pracovní ploše a stiskněte LTM. Zobrazí se dialog na *obrázku 5.1*. Do vstupního pole "Text" vložte požadovaný text, vyberte velikost, typ fontu, barvu textu a stiskněte tlačítko "OK".



Obrázek 5.1: Dialog při vložení textu

5.3.2 Kopírování a vložení prvků sítě ze schránky

1. Pro kopírování sítě do schránky je třeba nejdříve dané prvky označit. Zvolte nástroj s ikonou "Označit" a přejděte na pracovní plochu. Pohybem myši se současně stisknutým LTM označte kopírovanou síť a uvolněte LTM. Označení se zruší stiskem LTM v místě mimo kopírované prvky sítě. Pokud chcete kopírovat jen několik nesouvislých částí nakreslené sítě, stiskněte klávesu <CTRL> a označte výše popsaným postupem požadované části.

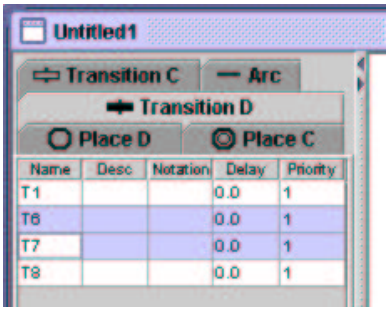
2. Akce kopírování do schránky se vyvolá v menu "Edit" položkou "Copy", klávesovou zkratkou <CTRL+C> nebo stiskem ikony "Kopírovat" na nástrojové liště. Označené položky se uloží do schránky.

3. Akce vložení ze schránky se provede v menu "Edit" položkou "Paste", klávesovou zkratkou <CTRL+V> nebo stiskem ikony "Vložit" na nástrojové liště. Aktivací se na pracovní plochu vloží obsah schránky, přičemž vložené prvky jsou označené. Nyní stačí prvky jen přesunout na požadované umístění podržením LTM a posunem na požadované umístění.

5.4 Pokročilé funkce

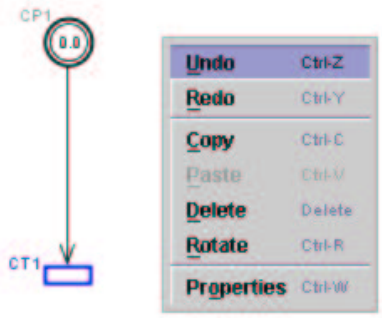
Pro snadnější a rychlejší práci s editorem uživatel může využít některých dalších funkcí:

1. **Levý panel**. Uživatel může pro snadnější orientaci v nakreslené síti použít levý panel (obrázek 5.2), který přehledně zobrazuje všechny dosud nakreslené prvky sítě v záložkách pojmenovaných podle typů prvků sítě. Označením některého řádku v tabulce se označí také prvek v nakreslené síti a zase naopak. Kliknutím na buňku tabulky lze přímo editovat vlastnosti daného prvku.



Obrázek 5.2: V levém panelu se v záložkách zobrazují všechny nakreslené prvky sítě

2. **Plovoucí menu**. Další užitečnou funkcí je plovoucí menu (obrázek 5.3), které obsahuje všechny často užívané akce. Aktivuje se stiskem PTM na pracovní ploše a položky menu, které jsou funkční, závisí na aktuálním stavu editované sítě a pozici kurzoru.



Obrázek 5.3: Plovoucí okno se aktivuje stiskem pravého tlačítka myši

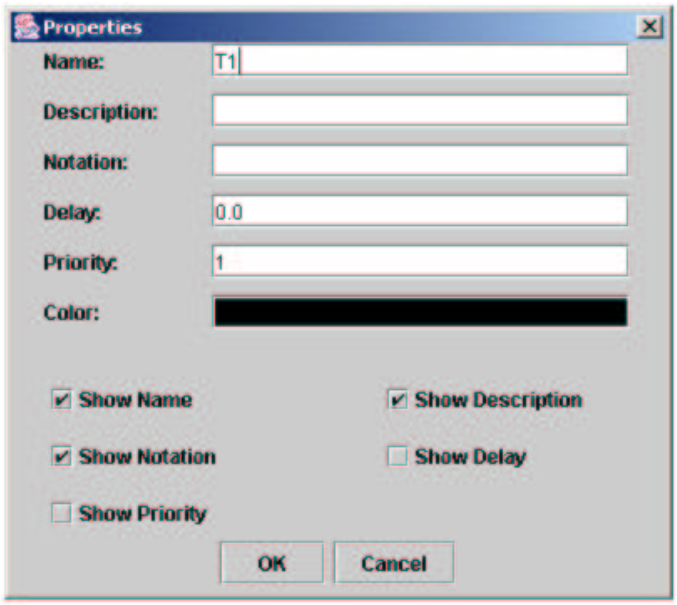
3. **Zpět a znovu**. Operace zpět a znovu umožňují pohybovat se v historii kreslené sítě. Operace zpět vrací stav sítě o jeden krok vzad, operace znovu naopak posune stav sítě o jeden krok vpřed. Tyto funkce se aktivují v menu "Edit" položkami "Undo" a "Redo", klávesovými zkratkami <CTRL+Z> a <CTRL+Y> a ikonamy "Zpět" a "Znovu" na nástrojové liště.

5.4.1 Editace vlastností sítě

U každého objektu na pracovní ploše lze nastavit určité vlastnosti. Některé vlastnosti lze editovat přímo v levém panelu dvojitým kliknutím LTM v dané buňce, všechny vlastnosti se potom nastavují v dialogu, který se zobrazí při dvojitém kliknutí LTM nad daným prvkem. Tento dialog lze také vyvolat označením daného prvku a stisknutím klávesové zkratky <CTRL+W>, v menu "Edit"

položkou "Properties" nebo v plovoucím okně položkou "Properties". Pro objekty na pracovní ploše se zobrazují tyto dialogy:

1. U diskrétního přechodu se zobrazí dialog na *obrázku 5.4*. V horní části jsou vstupní pole nastavující jednotlivé vlastnosti prvku, pod nimi jsou rozmístěny zaškrtávací pole rozhodující o tom, zda se daná vlastnost má zobrazit na pracovní ploše. Zcela dole je tlačítko "OK" potvrzující provedenou změnu, tlačítko "Cancel" naopak obnoví původní vlastnosti prvku před zobrazením dialogu.



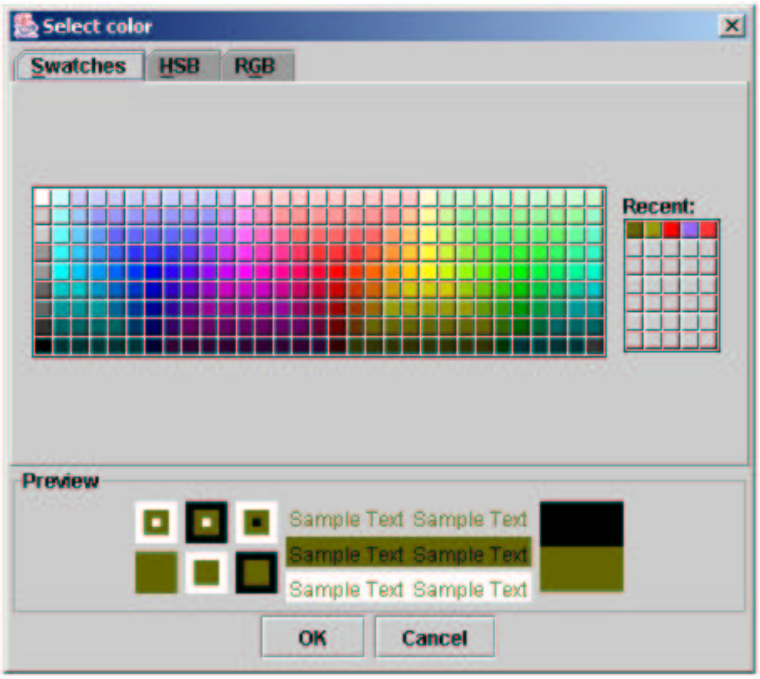
Obrázek 5.4: Okno pro nastavení vlastností diskrétního místa

Význam zobrazených vlastností je tento:

Name	jméno přechodu
Description	bližší popis přechodu
Notation	bližší popis přechodu
Delay	zpoždění přechodu
Priority	priorita přechodu
Color	barva přechodu

Pro nastavení barvy přechodu je třeba kliknout na tlačítko vedle popisku "Color". Zobrazí se dialog pro výběr a míchání vlastních barev na *obrázku 5.5*. Barvu lze zvolit v záložce "Swatches" klinutím na jednu z předdefinovaných barev. Vlastní barvu lze namíchat v záložkách "HSB" a "RGB".

U spojitého přechodu se zobrazí podobný dialog jako pro diskrétní místo, rozdíl spočívá v chybějících položkách "Priority" a "Delay", místo kterých je položka "Speed" popisující rychlost spojitého přechodu:



Obrázek 5.5: Dialog pro výběr vlastních barev

Name	jméno přechodu
Description	bližší popis přechodu
Notation	bližší popis přechodu
Speed	rychlost přechodu
Color	barva přechodu

2. Diskretní místo má podobný dialog pro nastavení vlastností jako přechod. Rozdíl je pouze v zobrazených položkách a jejich významu:

Name	jméno místa
Description	bližší popis místa
Notation	bližší popis místa
Tokens	počet tokenů v místě
Color	barva místa

Obdobný dialog se zobrazuje i pro spojité místo, položka Tokens je však nahrazena položkou Marks udávající počet značek.

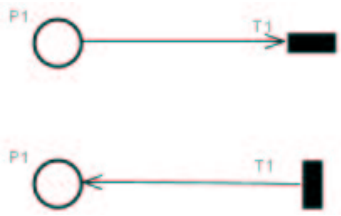
3. U hrany se opět zobrazí podobný dialog jako u přechodu, změna je v možnosti nastavení váhy hrany:

Name	jméno hrany
Description	bližší popis hrany
Notation	bližší popis hrany
Weight	váha hrany
Color	barva hrany

4. Pro každý text zobrazený na pracovní ploše (tedy i pro vlastnosti prvků) lze také vyvolat

dialogové okno pro nastavení jeho vlastností. Toto okno je stejné jako při vložení nového textu v sekci 5.3.1 (je na *obrázku 5.1*).

Kromě nastavení konkrétních vlastností prvků sítě editor ještě umožňuje změnit vzhled některých prvků. Jedná se především o funkci rotace, která otočí přechod o 90 stupňů (kratší stranou dolů) a u hrany otočí směr (obrázek 5.6). Funkce je dostupná v menu "Edit" pod položkou "Rotate", klávesou zkratkou <CTRL+R> nebo v plovoucím menu pod položkou "Rotate".

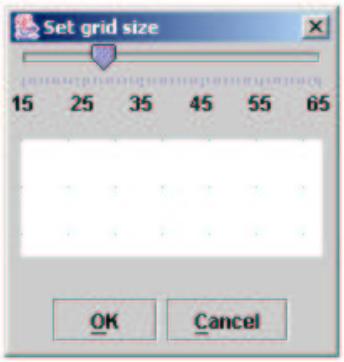


Obrázek 5.6: Rotace přechodu a hrany. Nahoře původní síť, dole síť vzniklá rotací.

Dále je možné změnit vzhled hrany posunem bodu v jejím středu (tento bod je při označení hrany zvýrazněný) nebo jednoduše uchycením a posunem samotné hrany. Vznikne tak hrana, která je zakřivená ve směru posunu bodu, který byl původně ve středu hrany. Narovnání zakřivené hrany se provede stiskem klávesy "Shift" a posunem hrany jakýmkoliv směrem. Stejnou funkci má klávesa "Shift" při přesunu označené skupiny prvků, kdy po jejím stisku dojde k napřímení označených hran.

5.4.2 Přizpůsobení editoru požadavkům uživatele (menu "Options")

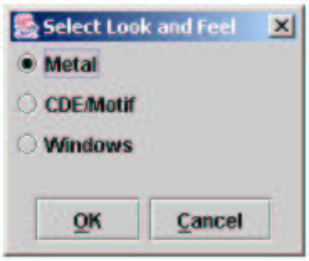
Při kreslení sítě je někdy výhodné mít zapnutou mřížku, na kterou se zachytávají vkládané objekty. Tuto mřížku lze zapnout v menu "Options" položkou "Grid On", případně klávesovou zkratkou <CTRL+G>. Nastavení velikosti mřížky lze vyvolat v menu "Options" položkou "Set grid size". Zobrazí se dialog na *obrázku 5.7*, kde se pohybem posuvníku doleva nebo doprava nastaví velikost mřížky.



Obrázek 5.7: Nastavení velikosti mřížky pro uchycení objektů

Editor při vykreslování objektů používá vyhlazování hran (antialiasing) k vylepšení celkového vzhledu kreslené sítě. Na některých počítačích může dojít k výraznému snížení rychlosti překreslování sítě a tím k celkovému snížení výkonu editoru. Potlačení vyhlazování hran lze zapnout v menu "Options" položkou "AntiAliasing" nebo klávesovou zkratkou <CTRL+A>.

Protože editor může být používán na různých operačních systémech s rozdílným vzhledem, uživatel si může vybrat vzhled editoru z dostupných vzhledů, které jsou v distribuci Javy pro daný operační systém dostupné. Dialog pro výběr vzhledu editoru (obrázek 5.8) uživatel vyvolá v menu "Options" položkou "Select Look&Feel" nebo klávesovou zkratkou <CTRL+E>.



Obrázek 5.8: Výběr vzhledu editoru ze všech dostupných vzhledů (OS Windows, JDK 1.4.1)

Uživatel si dále může zvolit způsob zobrazení otevřených vnitřních oken. V menu "Window" lze přepnout na způsob rozvrstvení oken připomínající dlaždice (položka menu "Tile") nebo na rozvrstvení oken kaskádně pod sebou (položka menu "Cascade"). Mezi okny lze přepínat kliknutím na jméno daného okna v menu "Window".

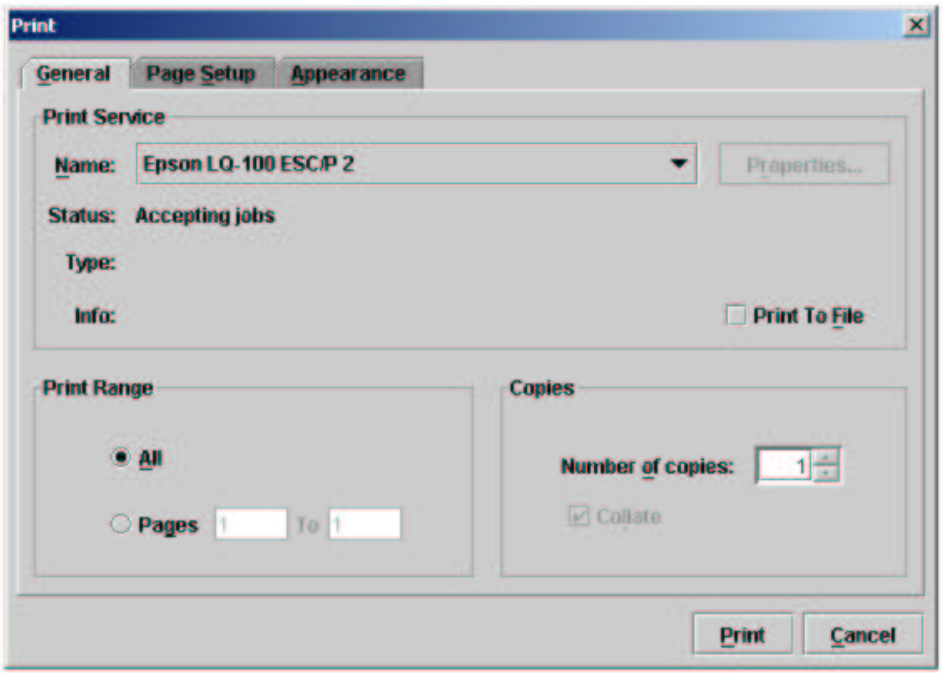
5.4.3 Načtení a uložení sítě, tisk (menu "File")

Jako formát pro ukládání sítě je použit formát PNML, popsaný v 4.2. Uložení sítě na disk se spouští položkou "Save" v menu "File", klávesovou zkratkou <CTRL+S> nebo na nástrojové liště ikonou "Uložit". Spuštěním této funkce se zobrazí standardní dialog pro uložení souboru. Pokud byla daná síť již jednou uložena, dialog pro ukládání souboru se už nezobrazuje a síť se uloží do souboru vytvořeném při prvním uložení. Pokud uživatel chce změnit jméno souboru až po prvním uložení, musí v menu "File" kliknout na položku "Save As" (v tomto případě se dialog pro uložení souboru zobrazí vždy).

Načtení souboru se aktivuje v menu "File" položkou "Open", klávesou zkratkou <CTRL+O> nebo v liště nástrojů ikonou "Otevřít". Zobrazí se standardní dialog pro otevření souboru, kde uživatel vybere požadovaný soubor. Funkce pro zpracování načítaného souboru je ošetřena tak, aby načetl i soubor v PNML formátu, který nebyl vytvořený v PN Editoru.

Kromě uložení sítě v PNML formátu je editor vybavený funkcí pro export nakreslené sítě do obrázku typu JPEG. Tato funkce se aktivuje v menu "File" položkou "Export to JPEG", kdy se zobrazí standardní dialog pro uložení souboru. Do tohoto souboru se poté převede nakreslená síť (rozměr obrázku je stejný jako rozměr pracovní plochy). Pokud si uživatel přeje exportovat pouze část nakreslené sítě, stačí tuto část označit a použít stejný postup (rozměr obrázku bude odpovídat velikosti označené sítě). Kvalita zakódování obrázku je nastavena na 100% (bezeztrátová komprese).

Dalším výstupem editoru je tisk na tiskárně. Aktivuje se v menu "File" položkou "Print" nebo klávesovou zkratkou <CTRL+P>. Zobrazí se dialog na *obrázku 5.9*, kde v záložce "General" lze zvolit typ tiskárny (Print Service), rozsah tisku (Print Range) a počet kopií (Copies). V záložce "Page setup" se nastavuje typ tištěného média (Media), orientace (Orientation) a zarovnání okrajů (Margins). Na poslední záložce "Apperance" si uživatel může vybrat, zda chce barevný či černobílý tisk (Color Apperance), kvalitu tisku (Quality), způsob potisku papíru (Sides) a atributy tiskové úlohy (Job Attributes).

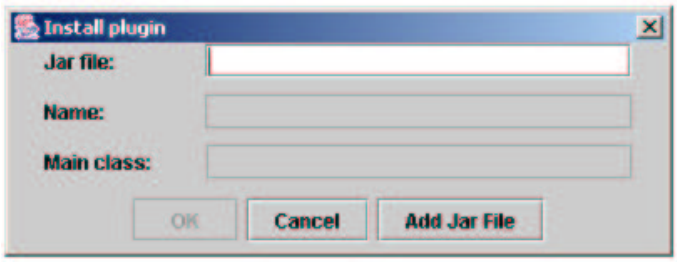


Obrázek 5.9: Dialog pro výběr tiskárny a nastavení vzhledu stránky

5.5 Systém pluginů (menu "Plugin")

Plugin (zásuvný modul) je program, který lze do editoru nainstalovat a který slouží k rozšíření jeho možností. Příkladem takového programu je plugin "Graf of reachable markings", který je s editorem dodáván. Tento plugin slouží k zobrazení grafu dosažitelných stavů Petriho sítě, který byl spočítán v Petri Net Matlab Toolboxu. Protože prostředí Matlab nemá vhodné visualizační prostředky, plugin využívá možnosti jazyka Java k zobrazení grafu dosažitelných stavů.

Instalace pluginu probíhá pomocí dialogu, který se zobrazí po kliknutí na položku "Install Plugin" v menu "Plugin". Zobrazí se dialog na *obrázku 5.10*, kde se tlačítkem "Add Jar File" zobrazí dialog pro vyhledání pluginu. Pokud je vyhledaný plugin v pořádku, automaticky se vyplní pole s názvem pluginu a třídou určenou pro spuštění pluginu. Potvrzením tlačítkem "OK" se plugin nainstaluje do menu "Plugins" a od této chvíle lze plugin spustit kliknutím na jeho jméno. Plugin lze odinstalovat kliknutím na položku "Uninstall Plugin" v menu "Plugin", kdy se zobrazí dialog s instalovanými pluginy. Výběrem požadovaného pluginu a stiskem tlačítka "OK" se plugin odinstaluje.



Obrázek 5.10: Dialog pro instalaci nového pluginu

5.5.1 Vytvoření nového pluginu

Tato část je určena pro vývojáře v jazyku Java, kteří chtějí vytvořit vlastní plugin, proto běžný uživatel editoru tuto část může přeskočit. Zdrojové soubory editoru a dokumentaci lze stáhnout na adrese `http://dce.felk.cvut.cz/cak/Research/PN/index_PN.htm`. Postup vytvoření nového pluginu se dělí na dvě části - vývoj vlastního pluginu a poté úprava pluginu do tvaru vhodného pro distribuci:

1. Vytvoření programu tvořící plugin s implementovaným rozhraním `cz.cvut.k335.pneditor.plugins.PluginInterface`
2. Zabalení pluginu do jar archívu tvořící plugin

Rozhraní `cz.cvut.k335.pneditor.plugin.PluginInterface` (*výpis 11*) musí v jedné ze svých tříd implementovat každý plugin. Toto rozhraní předepisuje pluginu implementovat tyto metody:

1. `public String getPluginName();`
   Tato metoda vrací jméno pluginu, který se zobrazí v dialogovém okně při instalaci pluginu a poté v menu "Plugins".
2. `public void initialize(Net net);`
   Tato metoda je volána před spuštěním pluginu pro jeho počáteční inicializaci. Parametr `net` je třída popisující aktuální Petriho síť nakreslenou na pracovní ploše.
3. `public void runPlugin(Net net);`
   Touto metodou se plugin spouští. Parametr `net` je třída popisující aktuální Petriho síť nakreslenou na pracovní ploše.
4. `public boolean callDrawMethodOnRepaint();`
   Tato metoda vrací hodnotu `true`, pokud si autor pluginu přeje, aby byl při každém překreslení sítě nakreslené na pracovní ploše také překreslen obsah pluginu. PN Editor potom při každém průchodu hlavní překreslovací smyčkou ve třídě `WorkSpace` zavolá metodu pluginu `draw(Graphics2D g2)`.
5. `public void draw(Graphics2D g2);`
   Pokud metoda `public boolean callDrawMethodOnRepaint()` vrací hodnotu `true`, je tato metoda zavolána pokaždé, kdy dojde k překreslení sítě na pracovní ploše editoru. Parametr `g2` je grafický kontext, do kterého byla síť na pracovní ploše vykreslena a umožňuje tak pluginu vykreslit na pracovní plochu vlastní objekty.

```
package cz.cvut.k335.pneditor.plugin;

import java.awt.*;
import cz.cvut.k335.pneditor.Net;

public interface PluginInterface {
    public String getPluginName();
    public void initialize(Net net);
    public void runPlugin(Net net);
    public boolean callDrawMethodOnRepaint();
    public void draw(Graphics2D g2);
}
```

Výpis 11: Rozhraní `cz.cvut.k335.pneditor.plugin.PluginInterface`

Pokud je plugin hotový, musí se vytvořit jar archív, který se používá při instalaci pluginu. V tomto archívu by měly být všechny soubory potřebné pro běh pluginu (přeložené třídy, obrázky, konfigurační soubory apod.). Vytvoření archívu `plugin.jar` z adresáře `plugin` se zapíše pomocí příkazu:

```
jar cmf Plugin_Manifest.mf plugin.jar plugin
```

Ještě předtím je však třeba vytvořit manifestační soubor `Plugin_Manifest.mf`, který se k archívu připojuje a který popisuje obsah archívu, konkrétně umístění třídy implementující rozhraní `PluginInterface`. Obsah tohoto souboru ukazuje výpis 9.

```
Manifest-Version: 1.0
Created-By: 1.4.1_01 (Sun Microsystems Inc.)
Main-Class: plugin.Main
```

Výpis 12: Obsah souboru `Plugin_Manifest.mf` popisující umístění třídy implementující rozhraní `PluginInterface` (třída `Main` v balíku `plugin`)

5.6 Technická data

Hardwarové nároky

Minimální konfigurace:
Intel Pentium II 300MHz, 64 MB RAM

Optimální konfigurace:
Intel Pentium III 933MHz, 256 MB RAM

Testované operační systémy

Microsoft Windows 98/2000/XP
Red Hat Linux 7.2

5.7 Seznam klávesových zkratek

Kláv. zkratka	Jméno	Popis
CTRL + 1	Select object	Přepne na nástroj pro výběr nakreslených objektů
CTRL + 2	Discrete transition	Přepne na nástroj pro vložení nového diskrétního přechodu
CTRL + 3	Discrete place	Přepne na nástroj pro vložení nového diskrétního místa
CTRL + 4	Arc	Přepne na nástroj pro kreslení hran
CTRL + 5	Continous transition	Přepne na nástroj pro vložení nového spojitého přechodu
CTRL + 6	Continous place	Přepne na nástroj pro vložení nového spojitého místa
CTRL + 7	Token	Přepne na nástroj pro vložení tokenů do diskrétního místa
CTRL + 8	Text	Přepne na nástroj pro vložení nového textu
CTRL + A	AntiAliasing	Vypne/zapne vyhlazování hran
CTRL + C	Copy	Vloží označenou síť do schránky
CTRL + E	Set Look&Feel	Zobrazí dialog pro výběr vzhledu editoru
CTRL + F1	Contents	Vývolá nápovědu
CTRL + F4	Close	Zavře aktuální okno
CTRL + G	Grid On	Vypne/zapne mřížku pro uchycení objetků
CTRL + I	Set grid size	Zobrazí dialog pro nastavení velikosti mřížky
CTRL + N	New	Otevře nové okno
CTRL + O	Open	Otevře soubor
CTRL + P	Print	Zobrazí dialog pro tisk
CTRL + R	Rotate	Otočí prvky sítě o 90 stupňů
CTRL + S	Save	Uloží aktuální síť
CTRL + V	Paste	Vloží síť ze schránky
CTRL + W	Properties	Zobrazí dialog pro editaci vlastnosti prvku
CTRL + X	Exit	Ukončí editor
CTRL + Y	Redo	Posune stav sítě o jeden krok vpřed
CTRL + Z	Undo	Vratí stav sítě o jeden krok zpět
Delete	Delete	Vymaže označené prvky

Kapitola 6

Závěr

Tento dokument popisuje navržené řešení editoru pro kreslení Petriho sítí. Podařilo se vytvořit plně funkční editor pro pohodlné kreslení a editaci různých typů Petriho sítí. Umožňuje načíst a uložit síť popsanou ve formátu PNML, přičemž pro potřeby editoru byl tento formát rozšířen o některé další části. Kromě běžných funkcí jako je kopírování, vkládání, operací zpět a znovu a mnoha dalších, editor také obsahuje systém pluginů, kterými ho lze jednoduše rozšířit o nové funkce. Systém pluginů byl vyzkoušen na pluginu zobrazující graf dosažitelných stavů, který byl vypočítán v MTPN.

Jednou z překážek při programování editoru se ukázala nižší rychlost Javy ve srovnání s jazykem C++. I když byly provedeny různé optimalizace kódu, rychlost aplikace nedosahuje výkonu se srovnatelnou aplikací napsanou v jazyku C++ a tato skutečnost se negativně odráží ve vyšších hardwarových nárocích na počítač uživatele. Tento problém je společný pro všechny desktopové aplikace napsané v jazyce Java a lze ho řešit dvěma způsoby:

- kompilací do nativního kódu daného procesoru a operačního systému, čímž dojde k podstatnému nárůstu výkonu. Zároveň však dojde ke ztrátě platformové nezávislosti a navíc takový kompilátor je k dispozici jen pro některé platformy.
- nahrazením grafických komponent z knihovny Swing komponentami z knihovny SWT. SWT je perspektivně se rozvíjející grafická knihovna vyvinutá v rámci projektu Eclipse [10] společností IBM. Její předností je rychlost srovnatelná s aplikacemi napsanými v C++, avšak omezení spočívá v nepříliš dobré dokumentaci a faktu, že není součástí java platformy.

Pro využití editoru v MTPN je třeba dokončit rozhraní toolboxu, které přečte soubor ve formátu PNML a převede ho na matice používané pro další výpočty. Jedna z cest, kudy by se také mohl ubírat další vývoj editoru, je v úpravě generovaného formátu PNML. Jak je popsáno v části 4.2.1, musel se doplnit formát PNML tak, aby do něj mohli být uloženy všechny vlastnosti Petriho sítě nakreslené v editoru. Dle autorů PNML bude příští verze obsahovat mnohé rozšíření jako podpora různých typů sítí a vlastností jejich prvků. Bude proto možné přepsat některé části generovaného PNML tak, aby vyhovovaly novému standardu a odstranily se některé části specifické pouze pro PN Editor.

Literatura

[1] Ricardo, Padilha. *JARP.*
http://jarp.sourceforge.net.

[2] PROMATIS AG. *INCOME Process Designer.*
http://www.promatis.com/products/income_processdesigner/index.htm

[3] Petri Net Kernel Team. *Petri Net Kernel.*
http://www.informatik.hu-berlin.de/top/pnk/

[4] Mortensen, Kjeld. Rölke, Heiko. *Petri Nets Tools Database.*
http://www.daimi.au.dk/PetriNets/tools/quick.html

[5] David, René. Alla, Hassane. *Petri Nets and Grafcet.*
Prentice Hall, 1992.

[6] Jungel, Matthias. Kindler, Ekkart. Weber, Michael. *The Petri Net Markup Language.*
http://www.informatik.hu-berlin.de/top/pnml/download/JKW_PNML.ps.

[7] World Wide Web Consortium. *Extensible Markup Language (XML).*
http://www.w3c.org/XML.

[8] IntelliJ. *IntelliJ IDEA.*
http://www.intellij.com/idea.

[9] Sun Microsystem. *Java Development Kit 1.4.1 (JDK 1.4.1).*
http://java.sun.com/j2se/1.4.1

[10] Eclipse consortium. *Eclipse project.*
http://www.eclipse.org/eclipse.

[11] Svádová, Martina. Hanzálek, Zdeněk. *Matlab Toolbox for Petri Net.*
http://dce.felk.cvut.cz/cak/Research/PN/index_PN.htm.

[12] Sun Microsystem. *JavaHelp.*
http://java.sun.com/products/javahelp.

[13] Sun Microsystem. *Java Virtual Machine.*
http://java.sun.com/getjava.

[14] Russel, Jordan. *Inno Setup.*
http://www.innosetup.com.

[15] Eckel, Bruce. *Thinking in Java 2nd Edition.*
Prentice Hall, 2000.

Kapitola 7

Příloha

7.1 Obsah přiloženého CD

```
Bin
    Other_OS
        j2re-1_4_1_01-linux-i586-rpm.bin    - JVM pro počítače s OS Linux
        j2re-1_4_1_01-solaris-sparc.sh      - JVM pro počítače s OS Solaris
        pneditor.tgz    - instalace PN Editoru bez JVM
    Windows
        setup.exe       - instalační soubor PN Editoru pro OS Windows
Docs
    Javadoc             - dokumentace PN Editoru vygenerovaná z programu JavaDoc
    Pnml
        jkw_pnml.ps     - popis formátu PNML
    Prezentace          - krátká prezentace diplomové práce
    diplomka.ps         - tento dokument
Source
    Instalator          -
        build_pneditor.ss   - skript pro sestavení instalátoru v programu Inno Setup
    Java2exe            - zdr. soubory pro vytvoření souboru "PNEditor.exe"
    M - file
        Test            - soubory pro ověření funkce "graph2rem.m"
        graph2rem.m     - funkce pro export matic grafu dosažitelných stavů
    PN Editor - IntelliJ    - zdrojové soubory PN Editoru pro prostředí IntelliJ IDEA
```

7.2 M-file - uložení matic grafu dosažitelnosti

```
function graph2rem(A, RM, filename)
% Save graph of reachable markings to a file named filename
% A -   Adjacency matrix (A(i,j) means oriented arc from vertex i to vertex j)
%      value of A(i,j) means index of fired transition
% RM -  Matrix of reachable states (each column represents marking of one state)

fid = fopen(filename,'w+');
fwrite(fid,size(A,1));%size ax
fwrite(fid,size(A,2));%size ay
fwrite(fid,size(RM,1));
fwrite(fid,size(RM,2));
fwrite(fid,A);
fwrite(fid,RM);
fclose(fid);
```

7.3 Skript pro sestavení instalátoru v programu Inno Setup

```
; Script generated by the My Inno Setup Extensions Script Wizard.
; SEE THE DOCUMENTATION FOR DETAILS ON CREATING INNO SETUP SCRIPT FILES!
[Setup]
AppName=PN Editor
AppVerName=PN Editor 1.0
AppPublisher=LuboÜ Svoboda, K335, Czech Technical Univesity in Prague, Czech Republic
DefaultDirName={pf}\PN Editor
DefaultGroupName=PN Editor
[Tasks]
Name: "desktopicon"; Description: "Create a &desktop icon"; GroupDescription: "Additional icons:"
[Files]
Source: "...\IMAGES\*.*"; DestDir: "{app}\images"; CopyMode: alwaysoverwrite
Source: "...\*.*"; DestDir: "{app}"; CopyMode: alwaysoverwrite
Source: "...\PLUGINS\*.*"; DestDir: "{app}\plugins"; CopyMode: alwaysoverwrite
Source: "...\HELP\*.*"; DestDir: "{app}\help"; CopyMode: alwaysoverwrite
Source: "...\HELP\html\first_step\*.*"; DestDir: "{app}\help\html\first_step"; CopyMode: alwaysoverwrite
Source: "...\JRE\*.*"; DestDir: "{app}\jre"; CopyMode: alwaysoverwrite
Source: "...\JRE\bin\*.*"; DestDir: "{app}\jre\bin"; CopyMode: alwaysoverwrite
Source: "...\JRE\lib\*.*"; DestDir: "{app}\jre\lib"; CopyMode: alwaysoverwrite
Source: "...\JRE\bin\client\*.*"; DestDir: "{app}\jre\bin\client"; CopyMode: alwaysoverwrite
Source: "...\JRE\bin\log\*.*"; DestDir: "{app}\jre\bin\log"; CopyMode: alwaysoverwrite
Source: "...\JRE\lib\cmm\*.*"; DestDir: "{app}\jre\lib\cmm"; CopyMode: alwaysoverwrite
Source: "...\JRE\lib\ext\*.*"; DestDir: "{app}\jre\lib\ext"; CopyMode: alwaysoverwrite
Source: "...\JRE\lib\fonts\*.*"; DestDir: "{app}\jre\lib\fonts"; CopyMode: alwaysoverwrite
Source: "...\JRE\lib\i386\*.*"; DestDir: "{app}\jre\lib\i386"; CopyMode: alwaysoverwrite
Source: "...\JRE\lib\im\*.*"; DestDir: "{app}\jre\lib\im"; CopyMode: alwaysoverwrite
Source: "...\JRE\lib\images\cursors\*.*"; DestDir: "{app}\jre\lib\images\cursors"; CopyMode: alwaysoverwrite
Source: "...\JRE\lib\security\*.*"; DestDir: "{app}\jre\lib\security"; CopyMode: alwaysoverwrite
Source: "...\JRE\lib\zi\*.*"; DestDir: "{app}\jre\lib\zi"; CopyMode: alwaysoverwrite
Source: "...\JRE\lib\zi\africa\*.*"; DestDir: "{app}\jre\lib\zi\africa"; CopyMode: alwaysoverwrite
Source: "...\JRE\lib\zi\america\*.*"; DestDir: "{app}\jre\lib\zi\america"; CopyMode: alwaysoverwrite
Source: "...\JRE\lib\zi\antarctica\*.*"; DestDir: "{app}\jre\lib\zi\antartica"; CopyMode: alwaysoverwrite
Source: "...\JRE\lib\zi\asia\*.*"; DestDir: "{app}\jre\lib\zi\asia"; CopyMode: alwaysoverwrite
Source: "...\JRE\lib\zi\atlantic\*.*"; DestDir: "{app}\jre\lib\zi\atlantic"; CopyMode: alwaysoverwrite
Source: "...\JRE\lib\zi\australia\*.*"; DestDir: "{app}\jre\lib\zi\australia"; CopyMode: alwaysoverwrite
Source: "...\JRE\lib\zi\etc\*.*"; DestDir: "{app}\jre\lib\zi\etc"; CopyMode: alwaysoverwrite
Source: "...\JRE\lib\zi\europe\*.*"; DestDir: "{app}\jre\lib\zi\europe"; CopyMode: alwaysoverwrite
Source: "...\JRE\lib\zi\indian\*.*"; DestDir: "{app}\jre\lib\zi\indian"; CopyMode: alwaysoverwrite
Source: "...\JRE\lib\zi\pacific\*.*"; DestDir: "{app}\jre\lib\zi\pacific"; CopyMode: alwaysoverwrite
[Icons]
Name: "{group}\PN Editor"; Filename: "{app}\PNEditor.exe"; WorkingDir: "{app}"
Name: "{group}\Uninstall PN Editor"; Filename: "{uninstallexe}"
Name: "{userdesktop}\PN Editor"; Filename: "{app}\PNEditor.exe"; WorkingDir: "{app}"; Tasks: desktopicon
[Run]
Filename: "{app}\PNEditor.exe"; Description: "Launch PN Editor"; WorkingDir: "{app}";
Flags: shellexec postinstall skipifsilent
```