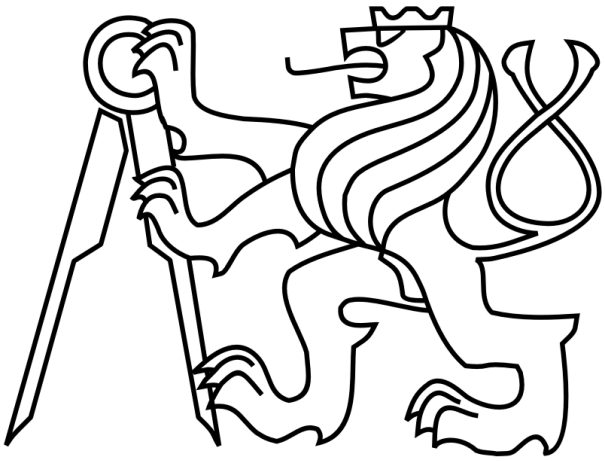
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA ELEKTROTECHNICKÁ



Katedra řídicí techniky

Bakalářská práce

Název tématu:	Simulátor Petriho sítí
Vedoucí bakalářské práce:	Ing. Libor Waszniowski
Autor:	Jan Ramba

Praha 2006

stránka se zadáním

Prohlášení

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

V Praze dne ……. …….
podpis

Abstrakt

Tato práce se zabývá vytvořením simulátoru Petriho sítí. Je součástí projektu, jehož cílem je simulátor systému diskrétních událostí, který bude schopen prostřednictvím OPC rozhraní komunikovat s řídicím systémem. Simulátor tak umožní pohodlně odlaďovat řídicí systémy na modelu řízeného systému.

Vytvořená aplikace umožňuje v grafickém uživatelském rozhraní řídit vývoj Interpretované Petriho sítě. Předpokládá se, že je síť vytvořena v některém z existujících komplexních grafických editorů Petriho sítí a uložena ve standardizovaném výměnném formátu pro Petriho sítě PNML. Stav Petriho sítě je také možné uložit do PNML souboru. Ovládání simulace probíhá jak pomocí ručního určení přechodů k aktivaci, tak pomocí automatického (náhodného) výběru přechodů. Náhodný výběr přechodů umožňuje spustit sekvenci zadaného počtu aktivací a jejich periodu. Aplikace je vytvořena v programovacím jazyku C#.

Abstract

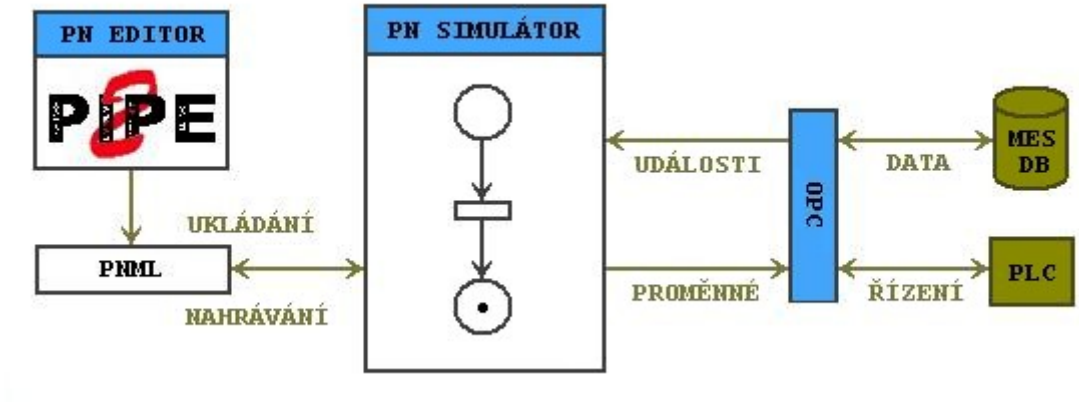
This project aims to create the Petri Nets simulator. It is a part of a bigger project whose goal is to develop discrete event systems simulator capable of communication with control system using OPC interface. Simulator will enable to comfortably debug control system at the model of controlled system.

Created application makes it possible to manage the development of the Interpreted Petri Net. It is assumed that the Petri Net is created in some of existing complex Petri Net graphical editor and saved in PNML, the standardized exchange format for Petri Nets. It is possible to save the state of Petri Net as a PNML file. A control of a simulation is accomplished either via manual selection of the transition to fire or via automatic (random) selection of the transition to fire. The random selection of the transition enables to start the sequence of demanded number and the period of firings. Application is implemented in C# programming language.

Obsah

1. Úvod

Tato bakalářská práce se zabývá možností simulovat chování systému diskrétních událostí pomocí Petriho sítí. V případě, kdy chceme komunikovat s distribuovaným systémem a chybí nám řídící hardware nebo jiná řídící aplikace nebo chceme testovat řídící program, můžeme právě simulátorem Petriho sítí nahradit tento systém. Chování modelovaného systému diskrétních událostí je pak dáno strukturou a vývojem Petriho sítě, která se dá jednoduše měnit.



Obr. 1 – Schéma systému se Simulátorem Petriho sítí

Petriho sítě jsou vhodným nástrojem pro modelování systémů diskrétních událostí, protože jsou schopné jednoduše a názorně modelovat všechny typické situace, se kterými se setkáváme u těchto systémů. Interpretovaná Petriho síť je druh Petriho sítě, který umožňuje synchronizaci sítě s vnějšími událostmi a export vlastních proměnných. Komunikace s distribuovanými systémy (například s MES systémem, Manufacturing Execution System, nebo PLC, Programable Logical Controller) by měla probíhat přes OPC (OLE for Process Control) rozhraní. Standart OPC je přímo určen pro komunikaci mezi řídícími zařízeními.

V rámci této bakalářské práce byl vytvořen pouze simulátor Petriho sítí. Komunikace přes OPC rozhraní bude součástí jiné práce.

Vytvořená aplikace SiPeSi (Simulátor Petriho Sítí) umožňuje v grafickém uživatelském rozhraní řídit vývoj Interpretované Petriho sítě. Předpokládá se, že je síť vytvořena v některém z existujících komplexních grafických editorů Petriho sítí, preferován je editor PIPE 2. Pro ukládání Petriho sítí existuje standardizovaný výměnný formát PNML (Petri Net Markup Language), který podporuje řada editorů. SiPeSi z tohoto formátu načte síť a do tohoto formátu také může ukládat aktuální stav sítě. Ovládání simulace probíhá jak pomocí ručního určení přechodů k aktivaci, tak pomocí automatického (náhodného) výběru přechodů. Náhodný výběr přechodů umožňuje spustit sekvenci zadaného počtu aktivací a jejich periodu. Editace parametrů Petriho sítě, které souvisí se simulací, je možná v přehledném okně. Všechny události se zaznamenávají do víceřádkového textového pole a pomocí něho také probíhá komunikace programu s uživatelem. Ovládání programu je možné pomocí menu a panelu nástrojů. Všechny funkce programu jsou popsány v přehledné nápovědě. Aplikace je vytvořena v programovacím jazyku C#, který dovoluje rychlé a relativně jednoduché vytváření aplikací.

Následující druhá kapitola shrnuje teoretické poznatky používané v práci. Obsahuje část o Petriho sítích, část o PNML a část o C#. Ve třetí kapitole je podrobněji rozebírán způsob řešení a popis implementaci programu. Čtvrtá část obsahuje závěr, kde jsou zhodnoceny dosažené výsledky. V poslední páté části jsou vyjmenovány použité zdroje informací.

2. Teorie

V následujících kapitolách jsou definovány pojmy, o které se bude opírat popis jednotlivých částí.

2.1 Petriho sítě

Tato kapitola obsahuje základní znalosti problematiky kolem *Petriho sítí,* text je převzat z přednášky Doc. Dr. Ing. Zdeňka Hanzálka z předmětu X35LOR – Logického řízení.

Petriho sítě jsou grafický a matematický nástroj vhodný pro modelování a analýzu systémů diskrétních událostí.

Díky své schopnosti jednoduše modelovat systémy diskrétních událostí obsahující paralelismus, sdílení zdrojů, vazbu na reálný čas a vazbu na okolní prostředí se Petriho sítě staly oblíbeným nástrojem teoretiků i praktiků. Teoretické výsledky z této oblasti jsou zajímavé, jelikož umožňují formálně prokázat vlastnosti modelovaných systémů.

Petriho sítě lze zdárně aplikovat v různých oblastech informatiky a automatizace. Zvláště jejich schopnost modulárně reprezentovat paralelně běžící procesy a komunikace je výrazně odlišuje od stavového diagramu.

2.1.1. Základní názvosloví

Neoznačená Petriho síť je orientovaný ohodnocený bipartitní graf. Jako každý graf se i Petriho síť skládá z uzlů, jež jsou propojeny hranami. Přívlastek *orientovaný* značí skutečnost, že hrany grafu jsou orientované. Přívlastek *ohodnocený* značí skutečnost, že hranám mohou být přiřazeny váhy. Přívlastek *bipartitní* značí skutečnost, že množina uzlů grafu se skládá ze dvou disjunktních podmnožin – množiny *míst* a množiny *přechodů*, přičemž místa a přechody se v průběhu cesty střídají.

Označená Petriho síť vznikne umístěním značek do míst neoznačené Petriho sítě.

Místo – může obsahovat nezáporný celý počet *značek.*

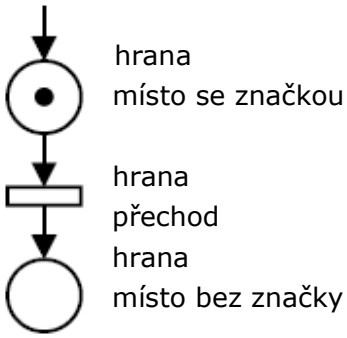
Přechod – v okamžiku jeho aktivace (přeskoku) jsou odebrány značky ze vstupních míst a přidány značky do výstupních míst přechodu.

Orientované hrany - propojují místa a přechody.

Počáteční značení - umístění značek v místech před prvním přeskokem, popisuje počáteční stav systému.

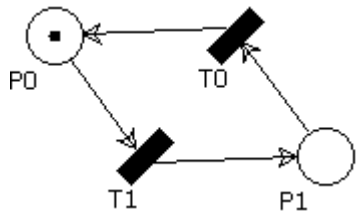
Vývoj systému - je reprezentován přesunem značek v síti na základě aktivace (přeskoku) přechodů

Stav sytému – je reprezentován značením.



Obr. 2 – Grafická reprezentace Označené Petriho sítě.

Graf reprezentující všechna dosažitelná značení v Petriho síti je ekvivalentní stavovému diagramu. Stavový diagram lze také nakreslit jako Petriho síť s totožnou topologií. Tato Petriho síť v sobě bude mít během vývoje systému právě jednu značku (ta se bude nacházet v místě, jež odpovídá aktivnímu stavu), takže nebude schopna reprezentovat paralelismus a pozbude tak veškeré elegance.



Obr.3 - Příklad jednoduché Petriho sítě.

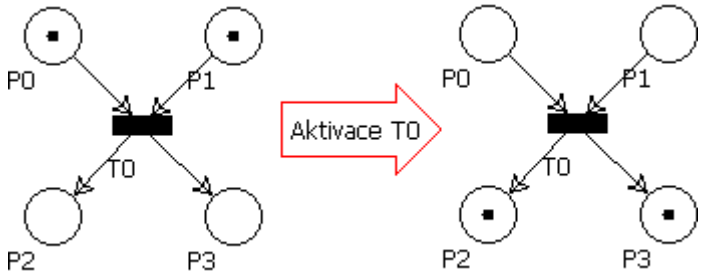
2.1.2. Pravidla pro změnu stavu systému

Jelikož každý přechod může mít jedno nebo více vstupních a výstupních míst, je potřeba definovat, za jakých podmínek dochází k přesunu značek v síti, neboli za jakých podmínek dochází k aktivaci přechodů.

Přechod je uvolněn, jestliže každé jeho vstupní místo obsahuje počet značek ≥ váze hrany spojující vstupní místo a přechod.

Uvolnění přechodu je nezbytnou (nikoli postačující podmínkou) pro jeho *aktivaci (přeskočení)*. Pro aktivaci přechodu musí být splněny další podmínky v závislosti na typu Petriho sítě (například výskyt události asociované s přechodem u *synchronizovaných Petriho sítí* nebo blíže neurčený okamžik u *autonomních Petriho sítí*) a "uvolňující" značky nesmí být spotřebovány jiným přechodem.

Při aktivaci jsou odebrány značky ze vstupních míst a nové značky jsou vloženy do výstupních míst. Počet odebraných/vložených značek odpovídá váze vstupní/výstupní hrany. Aktivace je nedělitelná operace.



Obr. 4 – Aktivace přechodu.

Z výše uvedených pravidel pro aktivaci přechodu lze odvodit některé typické konstrukce, s kterými se setkáme při modelování systémů pomocí Petriho sítí:

paralelismus (přechod s více výstupními hranami) – konstrukce, při níž z jedné sekvence vznikne více souběžně vykonávaných a na sobě nezávislých sekvencí,

synchronizace (přechod s více vstupními hranami) – konstrukce, při níž několik sekvencí na sebe navzájem čeká a poté pokračuje jedinou sekvencí,

výběr (místo s více výstupními hranami) – konstrukce, jež umožňuje, aby se značka pohybovala různými cestami,

spojení (místo s více vstupními hranami) – konstrukce, jež spojuje několik možných cest vývoje systému do jediné.

2.1.3 Formální definice autonomních Petriho sítí

Tato kapitola ukazuje, že Petriho sítě nejsou pouze grafický, ale i matematický nástroj.

Neoznačená Petriho síť R představuje uspořádanou čtveřici $R = (P,T,Pre,Post)$, kde:

$P = \{P_i,...P_m\}$ je konečná množina míst a $|P|=m$.

$T = \{T_i,...T_n\}$ je konečná množina přechodů a $|T|=n$.

Pre je matice $|P|\ x\ |T|$ obsahující celá nezáporná čísla reprezentující váhy hran jdoucích z míst do přechodů (precondition).

$Post$ je matice $|P|\ x\ |T|$ obsahující celá nezáporná čísla reprezentující váhy hran jdoucích z přechodů do míst (postcondition).

Incidenční matice C je matice $|P|\ x\ |T|$, pro kterou platí vztah $C = Post - Pre$.

Značená Petriho síť je dvojice (R,M), kde:

R je neoznačená Petriho síť.

M, značení Petriho sítě, je zobrazení z P do celých nezáporných čísel takové, že $M(P_i)$ je počet značek v místě P_i. Jelikož $|P| = m$, tak $M(P)$ je sloupcový vektor s m prvky. M_0 je počáteční značení Petriho sítě.

Aktivace přechodu. Přechod T_i je uvolněn pro značení M, jestliže $\forall\ P_j \in P;\ M(P_j) \geq Pre(P_j,T_i)$. Neboli počet značek ve vstupním místě musí být větší nebo roven váze hrany jdoucí z tohoto místa do přechodu.

Jelikož chování autonomních Petriho sítí není vztaženo k okolnímu prostředí, tedy ani k času, nemůžeme určit, kdy je přechod aktivován (přeskočen). Toto je důležité zjištění

zejména pro studium vlastností autonomních Petriho sítí, kdy bereme v úvahy všechny možné okamžiky, kdy je přechod aktivován. Přitom pro autonomní Petriho síť platí, že uvolnění přechodu je jedinou podmínkou pro to, aby mohl být aktivován.

Aktivací přechodu Ti se změní původní značení sítě M na nové značení M' tak, že

$\forall P_j \in P;\ M'(P_j) = M(P_j) - Pre(P_j,T_i) + Post(P_j,T_i)$.

Aktivační sekvence S je sekvence přechodů, jež byly uvolněny a přeskočeny v průběhu vývoje systému.

Charakteristický vektor je řádkový vektor s udávající kolikrát se daný přechod objeví v průběhu aktivační sekvence S.

Stavová rovnice Petriho sítě. Vývoj Petriho sítě ze značení M_0 do M je popsán rovnicí:

$M = M_0 + C \,.\, s'$.

2.1.4. Časované Petriho sítě

Časované Petriho sítě patří do skupiny neautonomních Petriho sítí. Jako takové mají vazbu na své okolí, v tomto případě na čas. Všechny sítě uvedené v této kapitole jsou časované Petriho sítě s maximální rychlostí aktivace.

P-časovaná Petriho síť je uspořádaná trojice (R, M_0, D) kde:

R je neznačená Petriho síť,

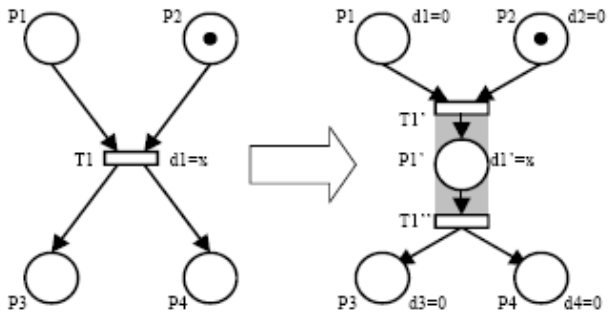
M_0 je počáteční značení,

D je zobrazení z P do množiny nezáporných přirozených čísel, neboli $D(P_i)=d_i$ je čas přiřazený místu P_i.

Jestliže je značka umístěna do místa P_i , potom musí zůstat v tomto místě nejméně po dobu d_i. Říkáme, že značka po dobu d_i není k dispozici. Po uplynutí této doby je značka k dispozici a může uvolnit další přechod.

T-časovaná Petriho síť je neautonomní Petriho síť, kde je každému přechodu T_j přiřazena doba $d_j \geq 0$, po kterou je značka rezervována přechodem T_j.

Jak ilustruje obrázek, tak T-časovaná Petriho síť může být jednoduše transformována na P-časovanou Petriho síť. Obdobně lze transformovat P-časovanou Petriho síť na T-časovanou Petriho síť.



Obr. 5 – Ilustrace transformace T-časované Petriho sítě na P-časovanou Petriho síť.

2.1.5. Interpretované Petriho sítě

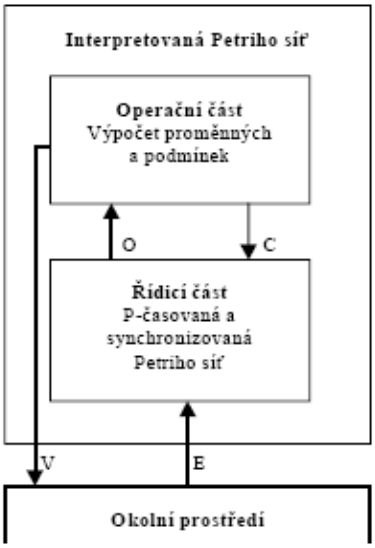
Interpretovaná Petriho síť má tři následující charakteristiky:

- je synchronizovaná (aktivace přechodu je vázána na výskyt události)
- je P-časovaná
- zahrnuje operační část, jejíž stav je dán množinou proměnných $V=\{V_1, V_2, \ldots\}$. Tento stav je modifikován množinou operací $O=\{O_1, O_2, \ldots\}$, jež jsou asociovány k místům. Stav určuje hodnotu podmínek $C=\{C_1, C_2, \ldots\}$, které jsou asociovány k přechodům.

Z výše uvedené definice je zřejmé, že aktivace přechodu T_j je podmíněna třemi faktory:

- zda je přechod T_j uvolněn (včetně toho, zda jsou uvolňující značky k dispozici vzhledem k časům d_i vstupních míst přechodu T_j)
- zda je podmínka C_j pravdivá
- zda v tomto okamžiku nastala událost E_j

Jestliže je značka vložena do místa P_i , potom provedeme operaci O_i a značka není k dispozici po dobu d_i.



Obr. 6 – Princip Interpretované Petriho sítě.

2.2 Petri Net Markup Language

Tato kapitola obsahuje informace o Petri Net Markup Language, vyvinuté v roce 2000 na Humboldt Universität zu Berlin, a o dalším vývoji tohoto formátu. Tyto informace jsou přeložené z textů na domovské stránky projektu http://www2.informatik.hu-berlin.de/top/pnml/about.html .

Petri Net Markup Language (*PNML*) je návrh výměnného formátu pro Petriho sítě na bázi XML. Původně bylo *PNML* zamýšleno jen jako souborový formát pro javovskou verzi programu Petri Net Kernel. Ale ukázalo se, že několik dalších skupin také vyvíjí výměnný formát na bázi XML. Proto bylo *PNML* bráno jen jako příspěvek do diskuze a standartizační snahy toho formátu.

Specifickou vlastností *PNML* je jeho otevřenost: rozlišuje mezi obecnými vlastnostmi Petriho sítí a specifickými vlastnostmi jednotlivých typů Petriho sítí. Specifické vlastnosti jsou definovány v odděleném Petri Net Type Definition (*PNTD*) pro každý typ Petriho sítě. Několik specifických vlastností je také použito ve více jak jednom typu Petriho sítě. Proto tu jsou Conventions Documenty obsahující specifické vlastnosti Petriho sítě. Tak konkrétní *PNTD* jen přidá své typové specifické vlastnosti k *PNML* referencí k Convetion Documentu. Standardizační snahy se ubíraly hlavně tímto směrem.

Na uvedených webových stránkách projektu je možné nalézt příklady *PNML*, několik *PNTD* pro různé typy Petriho sítí a Conversions dokumenty. V původní verzi *PNML* pouze demonstrovalo, že formát na bázi XML může být definován v obecně použitelné podobě. Předmětem diskuze pak bylo, které vlastnosti jsou tak obecné, že musí být zahrnuty v *PNML,* a které jsou specifické v závislosti na jednotlivých typech sítí. Příklady PNML souborů jsou k dispozici na přiloženém CD.

V současné době je tento projekt stále aktivní a probíhá standardizace *PNML* jako součásti normy ISO/IEC 15909-2.

2.2.1 PNML

Petri Net Markup Language (*PNML*) definuje souhrnnou strukturu souboru s Petriho sítí. Existuje mnoho různých typů Petriho sítí. Proto je definován obecný návrh, jak přizpůsobit specifické vlastnosti typů sítě pomocí *PNTD. PNML* je rozšířeno pro implementaci specifického *PNTD* určitými *labely*. *Label* přiřazuje další význam síti nebo objektu sítě. Typicky *label* reprezentuje jméno uzlu, počáteční značení místa, podmínku přechodu nebo váhu hrany. Přípustné *labely* a jejich kombinace jsou definovány v *PNTD.*

2.2.2 PNTD a Conventions Document

Petri Net Type Definition (*PNTD*) určuje konkrétní přípustný formát souboru pro síť určitého typu. Existuje mnoho různých *PNTD.* Na druhou stranu ale mají některé *PNTD* mnoho vlastností společných. Definice těchto vlastností jsou pobírány v *Conventions Documentu.*

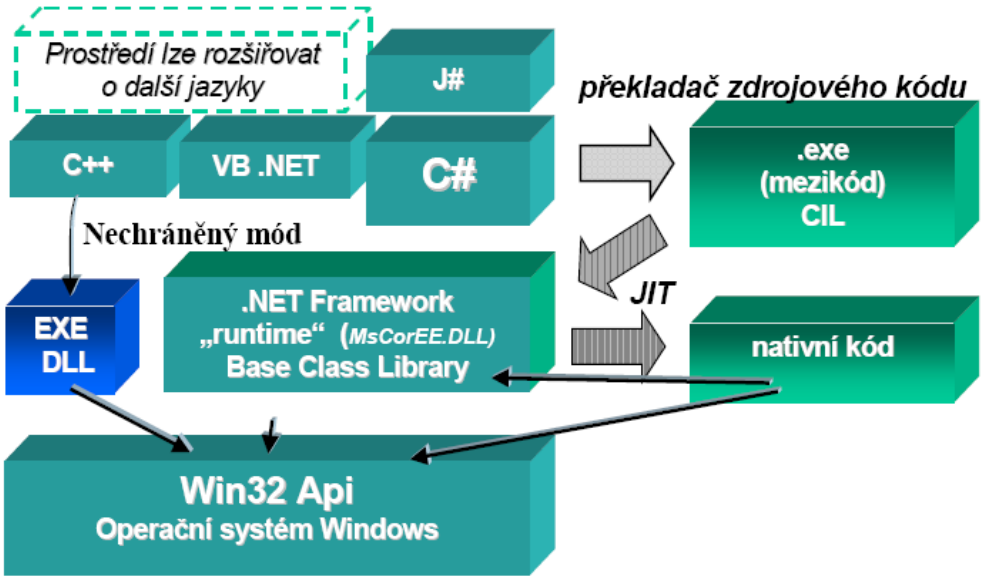
Conventions document je zamýšlen tak, že má sbírat definice všech různých relevantních *labelů*, které jsou používány v Petriho sítích, aby se zajistila kompatibilita mezi různými typy Petriho sítí. *Conventions document* se skládá z posloupnosti více či méně závislých definic *labelů.*

2.3 C#

Tato kapitola obsahuje krátký souhrn vlastností a původu jazyka C#, obsah je z části převzat z přednášek Ing. Richarda Šusty, PhD. http://dce.felk.cvut.cz/pjr/prednasky/plan.htm a od Daniela Večeři http://nb.vse.cz/~zelenyj/it380/eseje/xmard16/c_sharp.htm.

C# je jednoduchý, moderní, objektově orientovaný programovací jazyk odvozený z jazyka C++. C# je dodáván jako část Microsoft Visual Studia .NET. Spojuje jednoduchost VB a přitom výkonnost C++.

.NET Framework je nové prostředí pro aplikace, považované za objektové API, dovolující lepší vzájemné sdílení kódu a služeb. To umožňuje zavést multi-jazykové vývojové prostředí pro MS Windows. Všechny jazyky jsou v chráněném módu překládány do CIL (Common Intermedialte Language), který je matematicky ověřitelný. Do nativního kódu je pak přeloženo CIL při načítání programu. Součástí .NET Frameworku jsou také API ASP .NET (Active Server Pages) pro tvorbu aktivních serverových stránek a ADO .NET (ActiveX Data Object) pro snadnější přístup k databázím.



Obr. 7 – Struktura Visual Studia .NET.

Microsoft přišel s návrhem tohoto nového programovacího jazyka po definitivním rozchodu s Javou firmy Sun Microsystems. Autoři C# měli výhodu v tom, že za pět let existence Javy mohli nasbírat praktické zkušenosti s programovacím jazykem tohoto typu. C# se jeví jako jazyk podstatně flexibilnější a pragmatičtější. C# sice není přenositelný na jiné platformy než MS Windows (existuje projekt MONO, který dovoluje spouštění .NET programů na jiných platformách, ale není příliš rozšířen), ale za to plně využívá Windows API. Díky tomu jsou pak programy v C# efektivnější.

Kombinace jazyka C#, .NET Framework a Visual Studia .NET umožňje rychlý vývoj aplikací.

3. Vypracování

3.1 Rozbor zadání

V této kapitole uvádím, jak jsem se zadaný úkol rozhodl řešit.

Jedním z hlavních úkolů práce je vytvoření simulátoru Petriho sítí, který by přes rozhraní komunikoval s MES systémem a simuloval tak výrobní proces nebo jiný systém diskrétních událostí. Mým cílem je vytvořit jednoduchou a otevřenou aplikaci, kterou bude možno dále rozvíjet. Protože mi stačí vytvořit pouze simulátor Petriho sítí, nebudu tuto aplikaci dále komplikovat a ponechám vytvoření sítě na existujících komplexnějších editorech Petriho sítí. Vytvořenou síť pak uživatel do mého programu nahraje a bude moci ovládat její simulaci. Aby byla zajištěna co nejširší kompatibilita editorů, je požadovaný formát uložení *PNML,* který podporuje řada editorů. Formát *PNML* je založený na XML. Petriho sítě lze sice reprezentovat matematicky, ale simulace v textovém režimu by nebyla přehledná a dobře ovladatelná. Proto bude zobrazování vývoje Petriho sítě a ovládání simulace probíhat v grafickém uživatelském rozhraní.

Kolega, který měl v rámci své bakalářské práce vytvořit MES systém a jeho rozhraní, bohužel přesunul svou práci na příští rok. Proto komunikaci s rozhraním neimplementuji a vytvořím jen simulátor Petriho sítí s tím, že se počítá s možností rozšíření o tuto a další funkce do budoucna.

3.1.1 Reprezentace sítě

Do programu musí být implementována vnitřní reprezentace Petriho sítě a možnost s ní pracovat. *Neoznačená Petriho síť* je reprezentována *konečnou množinou míst*, *konečnou množinou přechodů* a hran mezi nimi, které jsou potřeba pouze pro vykreslování a ze které vyplývají *matice přechodu mezi místy a přechody* a *matice přechodu mezi přechody a místy. Značená Petriho síť* je reprezentována *Neznačenou Petriho sítí* a množinou reprezentující počáteční počet značek v místech. *Interpretovaná Petriho síť* je reprezentována *množinou proměnných, množinou operací* asociovaných k aktivaci přechodu a *množinou podmínek*, asociovaných k přechodům. Předpokládám, že všechny časy, přiřazené místům jsou nulové a synchronizace probíhá pomocí změny hodnoty proměnné událostmi na rozhraní. Operace nejsou přiřazeny k místům, ale k aktivaci přechodu. To je mnohem jednodušší, než provádět operaci pro každou značku v místě. V budoucnu bude, až bude implementována komunikace s OPC rozhraním, budou hodnoty proměnných získávány přes toho rozhraní.

Tyto množiny se načtou ze specifikovaného souboru v *PNML* formátu. Pokud uživatel načte síť z originálního *PNML* souboru, nebude obsahovat všechny elementy, používané v této aplikaci. Operace pak mají výchozí hodnotu, tedy žádnou operaci při aktivaci přechodu, podmínky jsou ve výchozím stavu pravdivé a síť nemá žádné proměnné. Všechny tyto atributy sítě je možno dodefinovat v průběhu programu. K uložení aktuální stavu *Značené Petriho sítě,*

tedy množiny míst, přechodů, hran a současného značení se dá využít základní definice *PNML.* Díky otevřenosti definice *PNML j*e ale možno uložit do speciálních *labelů* i tu část, která dodefinovává Interpretovanou část Petriho sítě*,* tedy množinu proměnných, operací a podmínek. Tyto *labely* jsou pak označeny určením pouze pro tuto aplikaci a nezpůsobují chyby v jiných aplikacích. Vše je možno uložit zpět do souboru, ze kterého byla síť načtena, nebo určit nový soubor k uložení.

Proměnné budou mít identifikátor, typ hodnoty a hodnotu. Typ proměnné bude buď celé číslo, desetinné číslo, boolovská proměnná nebo řetězec.

Operace asociované k aktivaci přechodu a *Podmínky uvolnění přechodu* je možné zapisovat pomocí prefixové syntaxe definované v *Nápovědě* a v tomto textu v kapitole Vypracování. Výrazy jsou vyhodnocovány rekurzivně, takže je možné použít jednu funkci jako argumentu druhé. *Operace asociované k aktivaci přechodu* buďto nesmí nebo musí obsahovat buď symbol pro žádnou operaci nebo přiřazení hodnoty alespoň jedné z proměnných. Může tedy obsahovat více přiřazení najednou. Naopak u *Podmínky uvolnění přechodu* je třeba vyhodnotit pravdivost výrazu, proto musí obsahovat funkci nebo proměnnou, u které se má smysl ptát na její pravdivost.

3.1.2 GUI

Aplikaci budu vytvářet v programovacím jazyku C#, protože je to jeden z moderních objektově orientovaných jazyků, ve kterém se snadno a rychle vytvářejí okenní aplikace díky vývojovému prostředí Visual Studio .NET.

Petriho síť je také grafický nástroj, proto je třeba vytvořit grafické uživatelské rozhraní (GUI – Graphical User Interface). Petriho síť je na základě vnitřní reprezentace grafických vlastností elementů sítě vykreslována do *Grafického panelu.* Po každé změně stavu nebo pohledu je překreslen obsah panelu. Pokud nahraná grafická reprezentace sítě přesáhne rozměry okna, je možno ovládat zobrazení jen části sítě. Události, upozornění a chyby jsou vypisovány do *Logu událostí*, s jehož pomocí probíhá komunikace s uživatelem ze strany programu*.* Při každé aktivaci přechodu nebo jiné akci se do *Logu událostí* vypíše aktivovaný přechod a dále informace o stavu sítě. Pro zpřehlednění výpisu je také možno si výpis některých událostí vypnout. Vypsané události je možné uložit do textového souboru a zachovat tak historii akcí. Naopak uživatel komunikuje s programem pomocí *Panelu nástrojů* s vybranými nejpoužívanějšími funkcemi a *Hlavního menu*, odkud budou přístupné všechny funkce programu.

Ovládání simulace probíhá pomocí *Panelu nástrojů* a *Grafického panelu*. Vybraný přechod je možné aktivovat pomocí kliku na tento přechod na *Grafickém panelu* nebo jeho vybráním z nabídky uvolněných přechodů na *Panelu nástrojů*. Další možností je z *Panelu nástrojů* aktivovat jeden náhodně vybraný přechod z množiny uvolněných přechodů, nebo posloupnost zadaného počtu aktivací náhodně vybraných přechodů se zadanou periodou aktivace. Zadání probíhá v jednoduchém dialogovém okně. Tuto posloupnost aktivací je možno přerušit. Pokud nejsou k dispozici žádné uvolněné přechody, posloupnost aktivací se sama ukončí.

Přidávání a ubírání míst, přechodů a hran není možné, protože to by vyžadovalo modifikaci grafické reprezentace sítě. To chci ponechat na editoru, ve kterém byla síť vytvořena. Naopak atributy, které nevyžadují modifikaci grafické reprezentace sítě a přímo souvisí se simulací, je možno editovat v dialogu *Editace sítě.* V tomto dialogu jsou panely pro jednotlivé editovatelné atributy *Značené Petriho sítě*: počet značek v místě, váha přechodu při náhodném výběru, dále pak atributy *Interpretované Petriho sítě*: počet, typ a hodnota proměnných, *Operace asociované k aktivaci přechodů* a *Podmínky uvolnění přechodů*. Uživatel si vybere položku k editaci, vybere požadovanou akci a spustí tak dialog ve stejném okně. Tento dialog mu umožní editovat požadovaný atribut a zkontroluje zadané hodnoty. Všechny změněné hodnoty se aplikují až po stisku potvrzovacího tlačítka, při stisku rušícího tlačítka se všechny provedené změny zahodí.

Před ukončením aplikace se zobrazí dialog s možností zrušení ukončování aplikace, aby se tak předešlo nepříjemným omylům.

Aplikace bude dále obsahovat přehlednou *Nápovědu*, kterou bude možno spustit z *Hlavního menu, Panelu nástrojů* a okna *Editace sítě.* Zde bude podrobně vysvětleno ovládání aplikace, průběh simulace a funkce jednotlivých příkazů.

Informace o názvu a kontaktu na autora se uživatel dozví z jednoduchého okna *O aplikaci.*

3.2 Popis implementace

V této kapitole detailněji popíšu, jak jsem řešil jednotlivé části aplikace, a uvedu odkazy na některé části kódu.

3.2.1 Reprezentace sítě

V aplikaci potřebuji uchovávat síť a její stav. Všechny atributy sítě je možné uložit a znovu nahrát a zachovat tak zcela přesně nejen reprezentaci sítě, ale i její současný stav.

Značená síť

Síť je reprezentována jednotlivými elementy sítě jako jsou množina míst, množina přechodů a množina hran. Abych je mohl zobrazit, musí mít každý element přidruženy informace o jeho vzhledu a vlastnostech. Tyto informace jsou ale pro každý typ elementu jiné, proto je každý typ elementu reprezentován zvláštní třídou.

Třída `Place` představuje reprezentaci elementu *místo*. Obsahuje všechny informace o místě, jako jsou identifikátor místa, unikátní řetězec znaků jednoznačně určující místo, jeho zobrazované jméno, počet značek, což je celé nezáporné číslo, které místo obsahuje, počáteční počet značek, které obsahovalo místo v okamžiku načtení sítě ze souboru, pozici místa v grafu

(tj. horizontální a vertikální pozici levého horního rohu v pixelech). Dále obsahuje get a v některých případech i set metody, jak tyto informace získat. Pro uchovávání a práci s těmito informacemi nepoužívám konstrukci jazyka C# property, protože s touto vymožeností jsem tak docela nesrostl a raději používám klasické gettery a settery, jak jsem zvyklý z Javy. Set metoda existuje pouze u atributu počtu značek, protože ostatní atributy nemá smysl měnit v tomto programu. Všechny atributy se inicializují při volání konstruktoru, jehož argumenty tyto atributy jsou.

Obr. 8 – Grafická reprezentace místa se 2 značkami.

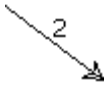
Předpokládá se, že místo je graficky reprezentováno černou kružnicí s pevně určeným průměrem a se středem určeným souřadnicemi levého horního vrcholu čtverce, který by opsal tuto kružnici. Tyto souřadnice jsou uloženy jako atributy místa. Značky jsou graficky reprezentovány plnými černými kruhy, pokud je jejich počet menší než 5, pokud je jich více, je jejich počet znázorněn číslem. Místa jsou rozlišena identifikátory vypsanými pod místem. Pokud se liší jméno místa od identifikátoru, vypíše se pod místo ve formátu [identifikátor]:[jméno].

Třída `Trans` představuje reprezentaci elementu *přechod*. Obsahuje všechny informace asociované k přechodu, metody, pro práci s informacemi, a pomocné metody pro vykreslování a jiné výpočty. Atributy přechodu jsou identifikátor přechodu, který přechod jednoznačně identifikuje, jeho zobrazované jméno, horizontální a vertikální pozice pravého horního vrcholu obdélníku přechodu, úhel otočení přechodu vzhledem k vertikální ose, dále celočíselnou nezápornou prioritu, která reprezentuje váhu jednotlivých přechodů při náhodném výběru přechodu k aktivaci, podmínku uvolnění přechodu a operaci asociovanou k aktivaci přechodu. Podmínky a operace jsou uchovávány jako textové řetězce a vyhodnocovány jsou až za běhu aplikace. Třída má přetížený konstruktor: jeden obsahuje všechny výše uvedené atributy pro případ, že je síť nahrávána ze souboru, který byl uložen mou aplikací, a druhý neobsahuje podmínky a operace pro případ, že se načítá soubor, neupravený mou aplikací. V průběhu programu lze pomocí setterů upravovat pouze prioritu přechodu, operace a podmínky.

Obr. 9 – Grafická reprezentace přechodu

Předpokládá se, že je přechod graficky reprezentován plným obdélníkem s pevně danou velikostí, určuje se pouze pozice levého horního vrcholu obdélníku a úhel otočení vzhledem k vertikální ose. Přechody jsou rozlišeny identifikátory vypsanými pod přechodem. Pokud se liší jméno místa od identifikátoru, vypíše se pod přechod ve formátu [identifikátor]:[jméno]. Barva přechodu se mění v závislosti na tom, zda je přechod uvolněn, pak je přechod znázorněn červenou barvou, nebo není, kdy je přechod černý. Přechod má metodu na výpočet vrcholů obdélníku z závislosti na otočení a pozici pravého horního rohu obdélníku. Dále pak metodu na ověření, zda je zadaný bod uvnitř obdélníku. Ta se využívá u možnosti aktivovat vybraný přechod kliknutím na něj v panelu na vykreslování sítě. Takto se ověřuje, jestli je bod, na který bylo kliknuto, součástí konkrétního přechodu.

Třída `Arc` představuje reprezentaci *hrany* mezi přechody a místy a místy a přechody. Z hlediska této třídy v tom není žádný rozdíl. Obsahuje všechny informace asociované k hraně, metody, které umožňují s nimi pracovat a pomocné metody pro vykreslování. Atributy hrany jsou identifikátor, který jednoznačně určuje jednotlivé hrany, identifikátor zdroje, odkud hrana vede, a identifikátor cíle, kam hrana ukazuje, a seznamy horizontálních a vertikálních souřadnic jednotlivých krajních bodů úseků hrany. Atributy této třídy lze nastavovat pouze v konstruktoru a ve třídě jsou pouze gettery těchto atributů.



Obr. 10 – Grafická reprezentace hrany s váhou 2.

Předpokládá se, že hrana bude složená z konečného počtu úseček, u kterých jsou známy krajní body a šipky u posledního bodu hrany. Krajní body hrotu šipky jsou vypočítány pomocnou metodou na základě souřadnic krajních bodů poslední úsečky hrany. Pokud je váha hrany větší než jedna, je u hrany číslo symbolizující váhu hrany.

Dále uchovávám jako grafické doplňky labely, které sice nemají přímo se sítí nic společného, ale dovolují popsat síť vlastními poznámkami a zpřehlednit tak celou grafickou reprezentaci sítě. Ty jsou reprezentovány třídou `Labels`, která má atributy text poznámky, horizontální a vertikální pozici levého horního rohu obdélníku, který ohraničuje poznámku, jeho šířku a výšku a hodnotu, signalizující, jestli má label také viditelné ohraničení. Tyto atributy lze nastavovat pouze v konstruktoru. Dále obsahuje metody, které příslušné atributy dovolují zjišťovat.

Interpretovaná Petriho síť

Třída `Var` představuje reprezentaci *proměnné*. Proměnná umožňuje uchovávat hodnoty a používat je v podmínkách aktivace přechodů. Atributy proměnné jsou její identifikátor, který ji jednoznačně definuje, její typ a její hodnota. Proměnná má v této chvíli pět typů, a to: celé číslo (int), boolovskou hodnotu (bool), desetinné číslo (float), řetězec znaků (string) a chybu. Konstruktor je přetížen, aby mohl přijmout všechny tyto typy hodnot. Zvláštní konstruktor při rekurzivním vyhodnocování výrazu v podmínce nebo operaci umožňuje předávat popis vzniklé chyby vyšším úrovním. V tomto případě se proměnná označí jako typ chyba a v jejím identifikátoru je její popis. Tato třída obsahuje gettery na všechny atributy a settery hodnoty proměnné pro každý typ zvlášť. Hodnota proměnné je uchovávána jako object, proto je nutné zjistit si vždy typ proměnné a hodnotu proměnné přetypovat na zjištěný typ.

Podmínky jsou, jak už bylo řečeno, uchovávány jako atribut přechodu. Jsou to textové řetězce, které se vyhodnocují při uvolnění přechodu na základě dostatečného počtu značek v místech, odkud vedou hrany do přechodu. Pokud je vyhodnocená podmínka pravdivá, pak teprve je uvolněn přechod doopravdy. Podmínky mohou obsahovat buďto pouze konstanty T, znamenající vždy pravdu, nebo F, znamenající vždy nepravdu, nebo boolovské proměnné nebo funkce, vracející boolovskou hodnotu, aby mělo smysl vyhodnocovat pravdivost podmínky. Funkce, které aplikace podporuje jsou vypsány níže v tabulce funkcí.

Operace jsou také uchovávány jako atribut přechodu. Jsou to textové řetězce, které se vyhodnocují při aktivaci přechodu. Operace musí přiřazovat některé proměnné hodnotu nebo obsahovat konstantu N, která značí žádnou akci. Přiřazovaná hodnota musí typově souhlasit s proměnnou, do které se hodnota přiřazuje. Operace může obsahovat i více přiřazení různým proměnným najednou.

Přehled funkcí a zápis proměnných, podmínek a operací

Přehled funkcí uvádí následující tabulka. Funkce jsou prefixové, takže nejdřív se uvede název funkce, poté do závorek její argumenty. Argumenty funkcí mohou být pouze takové hodnotové typy, které funkce očekává. Existuje ale výjimka v případě, kdy je požadováno jako návratový typ desetinné číslo (float). Pak je možné použít i celočíselný návratový typ (int) a celé číslo se na desetinné číslo snadno překonvertuje. Výrazy jsou vyhodnocovány rekurzivně, takže funkce je možno libovolně kombinovat tak, že uvedeme funkci jako argument jiné funkce. Pokud nebude řetězec nalezený ve výrazu, ani název funkce, ani proměnné, učiní se pokus převést nalezený řetězec na očekávaný hodnotový typ. V případě, že se ani toto nepovede, je generována výjimka a nadřazené výrazu je vrácena chybová proměnná, která nese i informaci o povaze chyby. Za názvem funkce nesmí následovat mezera a vyhodnocování výrazů je citlivé na velká a malá písmena.

[návrat. hodnota] **funkce**([argumenty])	**Význam zápisu**
T	Pravdivá podmínka
F	Nepravdivá podmínka
N	Žádná operace
bool **NOT**(bool V:x)	!x
bool **AND**(bool V:x, bool V:y)	x && y
bool **OR**(bool V:x, bool V:y)	x || y
bool **EQI**(int V:x, int V:y)	x == y (pro celá čísla)
bool **EQF**(float V:x, float V:y)	(pro desetinná čísla)
bool **EQS**(string x, string V:y)	(pro řetězce znaků)
bool **BTI**(int V:x, int V:y)	x > y (pro celá čísla)
bool **BTF**(float V:x, float V:y)	(pro desetinná čísla)
bool **STI**(int V:x, int V:y)	x < y (pro celá čísla)
bool **STF**(float V:x, float V:y)	(pro desetinná čísla)
int **PLUS**(int V:x, int V:y)	x + y (pro celá čísla)
float **PLUS**(float V:x, float V:y)	(pro desetinná čísla)
int **MINUS**(int V:x, int V:y)	x – y (pro celá čísla)
float **MINUS**(float V:x, float V:y)	(pro desetinná čísla)
int **MTP**(int V:x, int V:y)	x * y (pro celá čísla)
float **MTP**(float V:x, float V:y)	(pro desetinná čísla)
float **DIV**(float V:x, float V:y)	x / y
float **RAND**()	Náhodné číslo z intervalu <0,1>

Tab. 1 – Přehled funkcí (x a y jsou proměnné, návratové hodnoty se do výrazů neuvádějí, zde jsou pouze pro orientaci).

Proměnné jsou do výrazu kvůli snadnější identifikaci zadávány jako:
`V:[identifikátor proměnné].`

Podmínky se zapisují buďto jako konstanty T nebo F, nebo jako funkce, proměnné nebo hodnoty typu bool. Například (pro proměnné prom1 typu bool a prom2 typu int):

```
'T' ,
'AND(NOT(V:prom1), EQI(V:prom2, 5))' ,
'V:prom1'.
```

Operace se zapisují buďto jako konstanta N nebo jako přiřazení hodnoty odpovídajícího typu proměnné. Každé přiřazení musí být ukončeno znakem ';'. V jedné operaci se může vyskytnout více přiřazení najednou. Například (pro proměnné prom1 typu bool, prom2 typu int a prom3 typu float):

```
'V:prom1=OR(NOT(V:prom1), BTF(prom2, prom3));' ,
'V:prom2=5; V:prom3=DIV(prom2,2);' ,
'N' .
```

Celá síť

Všechny výše uvedené třídy používá třída `Net`. Tato třída reprezentuje celou síť. S novou instancí třídy Net se síť neinicializuje. Musí se zavolat metoda načítající síť ze souboru, protože z volání konstruktoru by nebylo možné zjistit, jestli bylo načtení sítě úspěšné.

Třída Net obsahuje množiny (jednorozměrná pole) míst, přechodů, hran, a labelů, které se během aplikace nemění. Dají se měnit pouze některé atributy prvků množin, jak bylo popsáno výše. Dále obsahuje aparát, pro reprezentaci a změny stavu sítě. Stav sítě je určen počtem značek v místech a množinou proměnných, na jejichž základě jsou uvolňovány přechody. Pro snadnější určování stavu a práci s ním obsahuje třída jako globální matice (dvourozměrná pole) *Pre*, obsahující celá nezáporná čísla reprezentující váhy hran jdoucích z míst do přechodů, *Post*, obsahující celá nezáporná čísla reprezentující váhy hran jdoucích z přechodů do míst, a *C*, incidenční matici.

Metody pro zjišťování stavu sítě umožní získat seznam uvolněných přechodů nebo tento seznam vypsat. Kontrola uvolnění přechodu se provádí pomocí porovnání počtu značek se sloupcem matice *Pre*. Pokud všechny členy ve sloupci jsou menší nebo rovné počtu značek v příslušném místě, je přechod uvolněn na základě počtu značek. Pokud je toto splněno, provede se kontrola podmínky uvolnění přechodu, jak bylo popsáno výše. Teprve jestli je pravdivá i podmínka, je přechod skutečně uvolněn a přidán do seznamu uvolněných přechodů.

Třída dále obsahuje metody pro změnu stavu sítě. To se provádí pomocí stavové rovnice:

$$M = M_0 + C \cdot s' .$$

Poté se vykoná operace asociovaná k aktivaci přechodu. Další možností je vrátit počet značek v místech do původního stavu v okamžiku načtení. Hodnoty proměnných zůstanou nezměněny.

O dalších metodách této třídy se zmíním v následujících kapitolách.

Načítání a ukládání sítě

Načtení sítě a tím pádem její inicializace probíhá, jak už bylo řečeno, pomocí metody třídy Net, která dostane za parametr cestu k souboru. Předpokládá se, že je soubor uložen v souborovém formátu *PNML.* Bohužel jsem při porovnávání se soubory, uloženými v PNML v různých editorech zjistil, že se souborový formát používaný editorem PIPE 2 (http://pipe2.sourceforge.net), ze kterého jsem vycházel, mírně liší od definice PNML. Odlišnost není velká a PIPE 2 se mi jeví jako velmi dobrý editor, takže jsem tomu úplně přizpůsobil své zobrazování sítě. K upravení načítání a ukládání podle klasického PNML by bylo nutné některé elementy přidat a některé odebrat.

Načítání je nezávislé na okolnosti, jestli je síť uložena v originálním formátu editoru PIPE 2, nebo jestli je uložené mou aplikací. Pokud je soubor uložen v originálním formátu jsou načteny pouze elementy sítě nesouvisející s Interpretovanou Petriho sítí. Atributy, definující Interpretovanou Petriho síť, mají výchozí hodnotu, tedy neexistuje žádná proměnná, podmínky jsou vždy pravdivé a operace nic nevykonávají. Pokud je soubor uložen v mé aplikaci, jsou v ní obsaženy všechny používané atributy sítě.

Ukládání probíhá s ohledem na formát souborů PIPE 2 a na alespoň trochu snadnější budoucí převoditelnost do klasického PNML. Všechny vlastnosti, které formát souborů PIPE 2 obsahuje, jsou uloženy stejným způsobem. Nadstavbové vlastnosti, definující Interpretovanou Petriho síť, jsou ukládány mou aplikací, a proto jsem se snažil je uložit podle návodu, který dává klasické PNML. Využil jsem takzvaného *toolspecific* tagu, který umožňuje využít PNML vytvořené někým jiným, a specifikovat, jaké aplikaci a verzi aplikace je informace, obsažená v tagu, určena. Neovlivňuje tak načítání souboru jinými aplikacemi. Tímto způsobem jsou tedy uloženy operace a podmínky v rámci přechodu. Proměnné jsou uloženy na úrovni míst pod *toolspecific* tagem, přechodů atd., protože jejich počet se nevztahuje k žádnému elementu sítě.

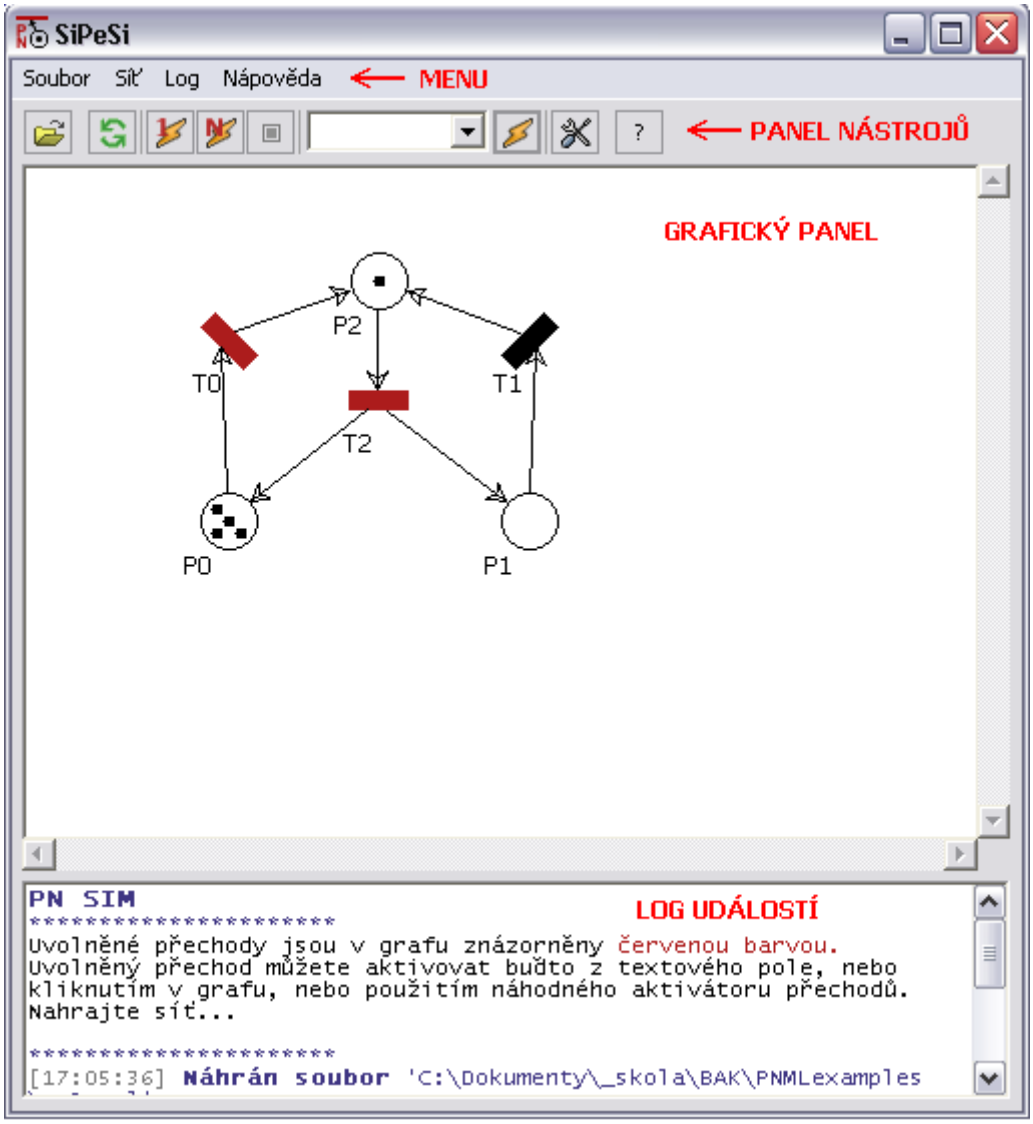
Příklady souboru jednoduché sítě, uložené pomocí editoru PIPE 2, stejné sítě uložené mou aplikací a stejné sítě uložené podle správného formátu PNML, jsou pro porovnání na přiloženém CD.

K PNML souboru musí být přidružen i PNTD soubor, definující význam a strukturu PNML souboru. Protože ale formát PIPE 2, a tím pádem i mé aplikace, nedodržuje klasickou definici PNML, nebudu tento definiční soubor vytvářet. Struktura je zřejmá z odstavců výše a hlavně ze samotného příkladu na CD.

3.2.2 Hlavní okno

Z hlavního okna jsou přístupné všechny funkce programu. Hlavní okno se skládá z hlavního menu s výsuvnými nabídkami, panelu nástrojů, panelu na vykreslování s posuvnými lištami na okraji a textové pole, umožňující formátovaný výpis událostí. Hlavní okno je instance třídy `Wind` , která se spouští vždy při startu aplikace. Hlavní okno se může nacházet ve dvou módech: když není nahraná žádná síť a když síť nahraná je. Pokud není ještě žádná síť k

dispozici, je přístupných pouze několik funkcí a to: nahrání sítě, ukončení aplikace, uložení logu událostí, zobrazení okna nápovědy a okna o aplikaci. Nahrání sítě zpřístupní všechny funkce.



Obr. 11 – Hlavní okno aplikace.

Celou aplikaci je možno spouštět rovnou se zadanou sítí k nahrání. Cesta k souboru se sítí se zadá jako argument při spouštění na příkazové řádce. Pokud není zadán žádný argument, nebo se ze zadané cesty nedá síť načíst, je spuštěno hlavní okno v omezeném módu s výzvou k nahrání sítě. Vstupním bodem do aplikace je třída `RunGUI`, ale ta pouze na základě přítomnosti argumentu při spouštění inicializuje třídu Wind se souborem nebo bez něj. Při přítomnosti souboru k nahrání se inicializuje třída Net, které se jako parametr předá ukazatel na Log událostí, aby třída Net mohla vypisovat chyby a informace.

Metody třídy Wind obsluhují většinou tlačítka a události popsané níže nebo jsou pomocné pro tyto metody.

Ovládání

Ovládání aplikace je možné třemi způsoby. Pomocí hlavního menu, panelu nástrojů a klávesových zkratek. Z hlavního menu jsou přístupné všechny funkce programu, na panelu nástrojů jsou pouze vybrané, nejpoužívanější, funkce, a většina funkcí má svou klávesovou zkratku.

Přehled tlačítek na panelu nástrojů:

 **Otevřít soubor** (Ctrl+L) - otevře dialog s výběrem souboru.
 **Obnovit počáteční stav** (Ctrl+R) - obnoví stav sítě po nahrání souboru.
 **Aktivovat náhodný přechod** (Ctrl+1) - aktivuje náhodně vybraný přechod.
 **Aktivovat více náhodných přechodů** (Ctrl+N) - umožní zadat počet a periodu přechodů k aktivaci.
 **Zastavit náhodnou aktivaci přechodů** (Ctrl+X) - zastaví aktivaci více přechodů.
 **Aktivovat vybraný přechod** (Ctrl+A) - aktivuje přechod vybraný v textovém poli nabídky.
 **Editace sítě** (Ctrl+E) - otevře dialog pro editaci sítě.
? **Nápověda** (F1) - otevře okno nápovědy.

Přehled funkcí hlavního menu (> značí přechod z vyšší úrovně menu do nižší):

Soubor >
> **Uložit** (Ctrl+S) - uloží současný stav sítě do zdrojového souboru.
> **Uložit jako...** - uloží současný stav sítě do vybraného souboru.
 > **Nahrát...** (Ctrl+L) - otevře dialog s výběrem souboru.
> **Konec** (Ctrl+Q) - ukončí program.
Síť >
 > **Obnovit počáteční stav** (Ctrl+R) - obnoví stav sítě po nahrání souoru.
 > **Aktivovat vybraný přechod** (Ctrl+A) - aktivuje přechod vybraný v textovém poli nabídky.
 > **Aktivovat 1 přechod** (Ctrl+1) - aktivuje náhodně vybraný přechod.
 > **Aktivovat N přechodů...** (Ctrl+N) - umožní zadat počet a periodu přechodů k aktivaci.
 > **Zastavit aktivaci** (Ctrl+X) - zastaví aktivaci více přechodů, při malé periodě doporučuji používat klávesovou zkratku.
 > **Editace sítě** (Ctrl+E) - otevře dialog pro editaci sítě.
Log >
> **Výpis proměnných** - zapne/vypne výpis hodnoty proměnných do logu.
> **Výpis uvolněných přechodů** - zapne/vypne výpis množiny uvolněných přechodů do logu.
> **Výpis nepravdivých podmínek** - zapne/vypne výpis množiny přechodů, které nebyly uvolněny kvůli nepravdivé podmínce uvolnění přechodu.
> **Uložit log do souboru...** - uloží obsah výpisového okna do vybraného souboru.
Nápověda >
? > **Obsah nápovědy** (F1) - ukáže okno s nápovědou.
> **O aplikaci** - ukáže okno s informacemi o aplikaci.

Klávesové zkratky k funkcím jsou uvedeny výše v závorkách u názvu funkcí.

Grafický panel

Zobrazování sítě probíhá na základě reprezentace sítě ve třídě Net. Při načtení sítě je volána metoda, která vytvoří novou bitmapu. Rozměry bitmapy jsou určeny nejvyššími nalezenými souřadnicemi v atributech jednotlivých elementů sítě. Do bitmapy se vykreslí všechny elementy sítě a bitmapa je nastavena do grafického panelu. Pokud rozměry bitmapy přesáhnou rozměry panelu jsou uvolněny posuvné lišty, které umožňují posunovat viditelný výřez bitmapy. Posuvné lišty jsou v případě, že je bitmapa dostatečně malá, zakázány. Při jakékoli změně zobrazení je volána událost Paint, která je zodpovědná za překreslování okna. V události Paint se vždy překresluje viditelná část bitmapy do Grafického panelu, jinak by nebyla po změně viditelná. Veškeré vykreslování probíhá pomocí metod třídy Net, která má přístupné všechny elementy sítě.

Když se změní stav sítě, není tvořena vždy nová bitmapa. Změna stavu sítě je pro grafickou reprezentaci pouze změna počtu značek v místě, protože jen ty jsou viditelné. Proto je vymazán pouze vnitřek míst (nakreslením plného kruhu v barvě pozadí) a graficky znázorněno nové označení. Při změně stavu sítě jsou kontrolovány uvolněné přechody, a ty jsou pak znázorněny červenou barvou. Přechody, které nejsou uvolněny, se znázorní černou.

Log událostí

Log událostí je textové víceřádkové pole s formátovaným výpisem. Pokud je výpis delší, než je výška přiděleného prostoru, přibude v textovém poli posuvná lišta, která posouvá zobrazovaný text. Log událostí umožňuje komunikaci programu s uživatelem. Vypisují se do něj všechny události, upozornění a chyby, vzniklé v průběhu programu. Výpis některých událostí je možno vypnout pomocí zaškrtávací položky v hlavním menu. Celý log je možné uložit do textového souboru.

Prostor okna, mezi Logem událostí a Grafickým panelem, je předělen úzkým pásem. Ten lze podle potřeby posunovat nahoru a dolů na úkor jednoho z nich.

3.2.3 Simulace

Nejdůležitější funkcí celé aplikace je simulace vývoje Petriho sítě. Třída Net obsahuje aparát na zjištění a změnu stavu Petriho sítě. Tento aparát využívá třída Wind. Ta definuje způsob aktivace jednotlivých přechodů. Aktivace uvolněných přechodů je možná buď přímým určením konkrétního přechodu nebo funkcemi aktivujícími náhodně vybraný přechod.

Přímé určení přechodu k aktivaci se může provést výběrem pomocí identifikátoru přechodu. Identifikátor přechodu se přímo vepíše do textového pole na Panelu nástrojů nebo se vybere z nabídky identifikátorů uvolněných přechodů ve výsuvné nabídce tohoto textového pole. Volba se potvrdí stiskem Enter nebo tlačítka Aktivace vybraného přechodu. Druhou možností, jak přímo vybrat přechod k aktivaci, je kliknout na něj v Grafické reprezentaci sítě. Třída Wind pak předá třídě Net souřadnice, kam bylo kliknuto a třída Net je porovná se souřadnicemi všech přechodů a zjistí tak, kterému přechodu souřadnice patří.

Náhodný výběr přechodu ponechává na programu, který z uvolněných přechodů aktivuje. Výběr probíhá pomocí generátoru pseudonáhodných čísel. Vygeneruje se náhodné číslo v rozsahu, který odpovídá délce seznamu uvolněných přechodů. V seznamu uvolněných přechodů se vyskytuje identifikátor jednoho přechodu tolikrát, jaká je jeho váha v atributech přechodu (priorita přechodu). Proto je větší pravděpodobnost, že bude vybrán přechod s větší váhou. Váha přechodu se může pohybovat v intervalu <0, 100> (nulová váha přechodu znamená, že přechod nemůže být při náhodné aktivaci vůbec vybrán).

Náhodnou aktivaci je možné spustit dvěma způsoby. Prvním způsobem je aktivovat pouze jeden přechod. Protože ale aktivace nezávisí na uživateli, je možné najednou aktivovat přechodů i více. To je druhý způsob, jak spustit náhodně vybraný přechod. Provádí se pomocí funkce Aktivovat N přechodů (N je myšleno jako přirozené číslo). Tato funkce spustí jednoduchý dialog, který umožní uživateli zadat nezáporný počet aktivací a nezápornou periodu aktivací. Dialog je instance třídy `WindRAT` (Random Activation of Transitions, nic společného s krysou to nemá). V případě, že jsou tyto hodnoty v požadovaném tvaru, spustí se časovač se zadanými vlastnostmi, který zajistí opakovanou aktivaci v daném intervalu. Pokud není zpracování handleru události "vypršení periody aktivace" ukončeno před následující událostí tohoto typu, jsou zpracování těchto handlerů řazena za sebe a vykonávají se postupně od nejstaršího. Tuto aktivaci je možno přerušit pomocí funkce Zastavit aktivaci, která časovač zruší.

Po aktivaci přechodu je změněna grafická reprezentace sítě a do logu událostí je vypsán čas aktivace a identifikátor přechodu, který byl aktivován. Dále jsou v závislosti požadovaných výpisů vypsány uvolněné přechody, hodnoty proměnných a případně identifikátor a podmínka přechodu, který není uvolněn na základě nesplněné podmínky (vypisuje se, protože to z grafické reprezentaci sítě není rozpoznatelné).

3.2.4 Okno Editace sítě

Funkce editace sítě spustí dialogové okno, které umožňuje editovat některé atributy sítě. Není možné upravovat atributy Neznačené Petriho sítě, protože to by změnilo grafickou reprezentaci sítě, kterou chci ponechat nezměněnou (samozřejmě až na značení a zvýraznění uvolněných přechodů). Okno je rozděleno na panely podle atributů, se kterými se v nich pracuje. Panel vždy obsahuje seznam všech atributů jednoho typu. Po označení vybrané položky seznamu je možné spustit jejich editaci v dialogu, který se objeví v dolní části panelu. Dialogové okno obsahuje ve spodní části stavový panel, který slouží pro komunikaci s uživatelem. Dále obsahuje tlačítko, které spustí okno nápovědy. Dialog je ukončen potvrzením nebo zahozením změn.

Okno Editace sítě je instance třídy `WindEd`. Konstruktoru se předává jako parametr ukazatel na třídu Net a tím pádem všechny potřebné informace o síti, které se dají získat pomocí getterů třídy Net.

V panelu *Značky* je možné měnit počet značek v jednotlivých místech.

V panelu *Priority* je možné měnit váhu jednotlivých přechodů při náhodné aktivaci nebo nastavit váhu na výchozí hodnotu 1. Panel je nazván Priority, protože Váhy by bylo možné zaměnit za váhy hran.

V panelu *Proměnné* je možné přidávat, editovat a mazat. Při vytvoření nové proměnné je uživatel vyzván k zadání identifikátoru, typu a hodnoty proměnné, při editaci je možné měnit pouze hodnotu proměnné. Případné chyby jsou uživateli hlášeny ve stavovém panelu.

V panelu *Podmínky* je možné měnit podmínky asociované k přechodu nebo je nastavovat na výchozí hodnotu 'T'. Při změně podmínky je provedena kontrola syntaxe zápisu a případně je na stavový panel okna vypsána zjištěná chyba.

V panelu *Operace* je možné měnit operace asociované k aktivaci přechodu nebo je nastavovat na výchozí hodnotu 'N'. Při změně operace je provedena kontrola syntaxe zápisu a případně je na stavový panel okna vypsána zjištěná chyba.

3.2.5 Okno Nápovědy a okno O aplikaci

K této aplikaci je přidružena nápověda, která se spouští ve zvláštním okně. Je členěna do kapitol, které jsou vypsány v podobě stromu v levé části okna. Výběr tématu způsobí nahrání odpovídající kapitoly do víceřádkového textového okna v pravé části okna.

Okno Nápovědy je instance třídy `WindHelp`. Při uzavření tohoto okna se okno neukončí úplně, ale je pouze skryto. Aby se nespouštělo více instancí okna Nápovědy najednou, při každém spouštění, kontroluje se, zda už okno neexistuje, a v tomto případě je pouze znovu ukázáno. Okno nejde maximalizovat, ani jinak změnit jeho velikost, protože poté by se zničilo formátování textu, které je uzpůsobené pro pevnou velikost. Texty témat jsou uloženy v RTF souborech (Rich Text Format) mimo aplikaci.

Informace o názvu aplikace a kontaktu na autora se uživatel dozví z jednoduchého okna *O aplikaci.* Toto okno je instancí třídy `WindAbout`.

3.3 Návrhy rozšíření funkčnosti

Nejdůležitější rozšíření je implementace komunikace s rozhraním. Takto by bylo možné synchronizovat Interpretovanou Petriho síť s událostmi na rozhraní. Dělo by se tak pomocí změny proměnné, která by měla vliv na uvolnění přechodu. Do mé aplikace by se začlenila další třída, která by obstarávala komunikaci s OPC rozhraním. Tu by pak používala třída Net ke zjišťování hodnoty proměnných.

Dalším důležitým rozšířením je implementace P-časované sítě. K místům by byl dodefinován další atribut, který by určoval, po jakou dobu nejsou značky v místě k dispozici. Takto by teprve byla splněna celá definice Interpretované Petriho sítě. Časování by vedlo k nutnosti použití vláken.

Vlákna by se dala využít i při překreslování Grafického panelu, protože grafické operace s rozsáhlejší bitmapou jsou časově náročné a v některých případech ovlivňují výkon funkcí. To by vedlo i k zefektivnění práce handleru časovače náhodné aktivace přechodů, který by pouze změnil stav sítě a vykreslení by nechal na jiném vlákně.

K funkcím aplikace by se mohla přidat možnost definovat, v jakém stavu by se měla zastavit Náhodná aktivace přechodů (podobné breakpointům). Tak by se dala přeskočit nezajímavá část simulace. Obdobného efektu by se dalo dosáhnout dynamicky upravovanou periodou aktivací, například pomocí posuvného ukazatele na panelu nástrojů.

K úpravám by mělo dojít v zadávání podmínek a operací. To by se mohlo v některých případech z poněkud nepřehledné prefixové podoby předělat na přirozenější infixovou. Rozšířen by také mohl být rejstřík funkcí.

Ke zjednodušení práce s aplikací by mohlo vést přidání možnosti práce s více sítěmi najednou. Každá síť by měla k dispozici vlastní Grafický panel a Log událostí. To také by ale vedlo ke komplikacím v práci s rozhraním.

Spíše kosmetickou změnou je vykreslování pomocí double-bufferingu, které by odstranilo blikání při vykreslování obrázku.

Dalším z jednodušších možných rozšíření je schopnost uložení bitmapy do souboru.

4. Závěr

V této kapitole je popsáno, co bylo vytvořeno v bakalářské práci, naznačen postup práce, a zhodnoceny výsledky práce.

V rámci této bakalářské práce byla vytvořena aplikace umožňující simulování Petriho sítě. Aplikace se zaměřuje na simulaci Interpretované Petriho sítě, která dovoluje synchronizovat běh simulace s vnějšími událostmi a exportovat vlastní proměnné do vnějšího prostředí. Při výchozím nastavení sítě, kdy nejsou nastaveny žádné atributy, definující Interpretovanou Petriho síť, je také možné simulovat autonomní část Interpretované Petriho sítě. Do aplikace je možné načítat soubory v mírně modifikovaném formátu PNML, ve kterém ukládá soubory preferovaný open source grafický editor Petriho sítí PIPE 2 (Platform Independet Petri net Editor, http://pipe2.sourceforge.net). Aktuální stav sítě, včetně všech změn provedených v její reprezentaci, je také možné uložit pomocí tohoto formátu. Pro zápis podmínek a operací Interpretované Petriho sítě byla použita univerzální prefixová syntaxe, dovolující i zápis komplikovaných podmínek a více současných operací. Tato syntaxe je jednoduše rozšiřitelná o další funkce. Simulace Petriho sítě se ovládá jak ručním určením přechodu k aktivaci, tak pomocí náhodného výběru přechodu, což umožňuje spustit běh simulace nezávisle na uživateli (ten zadá pouze počet aktivací a jejich periodu).

Aplikace obsahuje grafické uživatelské rozhraní s panelem pro vykreslování Petriho sítě, víceřádkovým textovým polem pro výpis událostí a komunikaci s uživatelem a nástroji na ovládání aplikace. Dále obsahuje okno pro editaci parametrů sítě souvisejících se simulací (základní struktura se edituje v editoru Petriho sítí), okno s přehlednou textovou nápovědou, rozdělenou do kapitol, a okno, zobrazující informace o aplikaci.

K plnému využití mě aplikace je potřeba implementovat ještě komunikaci s rozhraním, která bude předmětem jiné práce. Má práce je připravená na snadné rozšíření o další funkce popsané v této práci.

5. Použité zdroje informací

[1] *Doc. Dr. Ing. Zdeňka Hanzálek*: Přednáška z předmětu X35LOR - Petriho sítě a Grafcet

[2] Informace o souborovém formátu PNML:
http://www2.informatik.hu-berlin.de/top/pnml/about.html

[3] *Ing. Richard Šusta, PhD.*: Přednášky z přemětu X35PJR o programovacím jazyku C#

[4] Reference o programovacím jazyku C#:
http://msdn.microsoft.com/vcsharp/programming/default.aspx

[5] Příklady kódů v programovacím jazyku C#: http://www.c-sharpcorner.com

[6] Příklady kódů v programovacím jazyku C#: http://www.codeproject.com

[7] *Daniel Večeřa*: Informace o programovacím jazyku C#:
http://nb.vse.cz/~zelenyj/it380/eseje/xmard16/c_sharp.htm

[8] Otevřená encyklopedie Wikipedia: http://en.wikipedia.org

[9] Informace o grafickém editoru PIPE 2: http://pipe2.sourceforge.net/index.html

[10] Informace o grafickém editoru WoPeD: http://woped.ba-karlsruhe.de/woped/PetriNets

Přílohy

Adresářová struktura CD

```
\
+ bin        – Přeložená spustitelná aplikace.
|  + help    – Textové soubory s nápovědou.
+ src        – Zdrojové kódy aplikace.
+ doc        – Dokumentace k aplikací.
+ examples   – Příklady PNML souborů a porovnání originálního PNML s formátem PIPE 2.
+ proj       – Projekt v Microsoft Visual Studiu 2003.
```