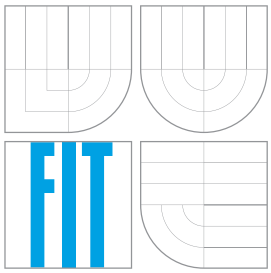
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRO PRÁCI S OBJEKTOVĚ ORIENTOVANÝMI PETRIHO SÍTĚMI

THE TOOL FOR OBJECT ORIENTED PETRI NETS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN JOSEFÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK KOČÍ, Ph.D.

BRNO 2016

Zadání bakalářské práce/18426/2015/xjosef00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Josefík Martin**
Obor: Informační technologie
Téma: **Nástroj pro práci s Objektově orientovanými Petriho sítěmi**
The Tool for Object Oriented Petri Nets
Kategorie: Softwarové inženýrství

Pokyny:

1. Prostudujte formalismus Objektově orientovaných Petriho sítí (OOPN), jazyk PNtalk a jazyk pro popis Petriho sítí PNML.
2. Seznamte se s aktuální implementací simulačního serveru OOPN a komunikačním protokolem.
3. Navrhněte nástroj umožňující vytváření, editaci a ukládání modelů OOPN ve formátu PNML a jazyka PNtalk a export modelů do formátu EPS (Encapsulated PostScript).
4. Implementujte nástroj. Rozšiřte nástroj o možnost komunikace se simulačním serverem - ukládání a načítání modelů. V případě potřeby upravte komunikační protokol.
5. Diskutujte dosažené výsledky a navrhněte možná rozšíření nástroje. Výsledky také prezentujte formou posteru.

Literatura:

- Janoušek, V.: Modelování objektů Petriho sítěmi, Brno, CZ, UIVT FEI VUT, 1998
- Petri Net Markup Language, http://www.pnml.org

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

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

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

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

Vedoucí: **Kočí Radek, Ing., Ph.D.**, UITS FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

L.S.

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

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Tato práce se zabývá tvorbou nástroje pro práci s Objektově orientovanými Petriho sítěmi za použití dvou rozdílných jazyků PNML a PNtalk. PNML slouží pro popis Petriho sítě, je založený na XML a jeho výhodou je, že dokáže každý prvek Petriho sítě popsat pomocí svých elementů. PNtalk je konkrétní implementací OOPN a je založený na jazyce Smalltalk. Nástroj pro práci s OOPN bude komunikovat s externí aplikací systémem PNtalk z důvodu výměny modelů. Na závěr budou navržena rozšíření nástroje.

Abstract

This thesis deals with creation of the tool for Object Oriented Petri Nets which uses two different languages PNML and PNtalk. PNML is an XML-based interchange format for Petri nets. Its advantage is that it is possible to describe each element of Petri net by element of PNML language. PNtalk is based on the formalism of Object Oriented Petri Nets and also on Smalltalk language. The tool for OOPN will communicate with the external application PNtalk system in order to exchange models between these two applications. There will be discussed possible extensions for the tool at the end of the thesis.

Klíčová slova

Objektově orientované Petriho sítě, simulace, PNtalk, PNML, simulační server

Keywords

Object oriented Petri nets, simulation, PNtalk, PNML, simulation server

Citace

JOSEFÍK, Martin. *Nástroj pro práci s Objektově orientovanými Petriho sítěmi*. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Kočí Radek.

Nástroj pro práci s Objektově orientovanými Petriho sítěmi

Prohlášení

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

.....
Martin Josefík
12. května 2016

Poděkování

Zde bych rád poděkoval panu Ing. Radku Kočímu, Ph.D za odborné rady, čas a vedení této práce.

Obsah

Kapitola 1

Úvod

Petriho sítě poskytují prostředky pro modelování nedeterminismu a paralelismu. Jsou matematickým formalismem a umožňují návrh, verifikaci a analýzu modelů dynamických diskrétních systémů. Petriho sítě však nenabízí mechanismy strukturování, jako jsou funkce a objekty, proto se jejich výhody daly využít pouze pro modelování jednoduchých a velmi abstraktních systémů. Pro modelování složitých systémů se musely používat běžné programovací jazyky, protože navržené modely v Petriho sítích byly nepřehledné a zbytečně složité. Běžné programovací jazyky zase nenabízí matematický základ, který by umožnil formální analýzu modelů. Vznikla proto snaha o zavedení strukturovacích mechanismů a objektově orientovaného paradigmatu do Petriho sítí.

Objektově orientovaná Petriho síť rozšiřuje Petriho síť o koncept objektové orientace [3]. Zůstává matematickým formalismem a neztrácí výhody Petriho sítí. Struktura sítí je stále tvořena místy, přechody a hranami. Místa sítí mohou obsahovat značky, které reprezentují objekty. Systém je možné modelovat množinou objektů, které mohou během existence systému dynamicky vznikat a zanikat. Objekty mezi sebou komunikují prostřednictvím zasílání zpráv. Metody a strukturu objektů uchovávají třídy.

Cílem této práce je vytvořit nástroj umožňující vytváření a editaci Objektově orientovaných Petriho sítí za použití jazyků *PNML* a *PNtalk*.

Jazyk PNML je založený na XML a slouží pro popis Petriho sítí, jednotlivé prvky Petriho sítě reprezentuje pomocí XML elementů. Díky standardizaci *ISO/IEC 15909* je možné přenášet modely Petriho sítí mezi různými nástroji. V současné době je jazyk PNML standardizován pouze pro typy Petriho sítí: *High-Level Petri Nets*, *Symmetric Nets* a *Place/Transition Nets*. Standard PNML pro ukládání a přenos OOPN neexistuje. V kapitole 3 bude představena modifikovaná verze jazyka PNML vycházející ze standardu pro *Place/Transition Nets* umožňující ukládání OOPN.

PNtalk je graficko-textový jazyk založený na OOPN vyvinutý pro prototypování a modelování distribuovaných systémů. Pro popisy sítí využívá jazyk Smalltalk, sítě jsou specifikovány graficky i textově.

Nástroj pro práci s OOPN bude provádět konverzi mezi jazyky PNML a PNtalk z důvodu výměny modelů OOPN mezi nástrojem a externí aplikací, simulačním serverem systémem PNtalk. Server poskytuje akce jako například simulace modelů, krokování simulací, registrace událostí, získávání výsledků a statistik simulací a další. Jazyk PNtalk bude detailněji popsán v kapitole 4, server včetně komunikačního protokolu v 4.2.

Kapitola 2

Objektově orientované Petriho síťě

OOPN jsou formalismem, který nabízí výhody Petriho sítí a objektové orientace. Petriho sítě umožňují popsat modely a části systému formálními prostředky a objektová orientace umožňuje vhodné strukturování. V této kapitole budou popsány jednotlivé prvky OOPN.

2.1 Objektová orientace

OOPN zahrnuje tři nejdůležitější koncepty objektově orientovaného paradigmatu: dědičnost, zapouzdření a polymorfismus. Každá z těchto vlastností bude detailněji rozepsána a bude také uvedeno, jak se projevuje přímo v OOPN.

2.1.1 Dědičnost

Dědičnost umožňuje sdílet chování a strukturu mezi objekty. Platí, že obecnější objekt poskytuje konkrétnějšímu objektu své vlastnosti. Konkrétnější objekt neboli potomek pak může přidávat další vlastnosti a zděděné vlastnosti dále upřesňovat a nahrazovat. Díky dědičnosti lze definovat nové struktury a objekty na základě starých.

V OOPN se uplatňuje pouze jednoduchá dědičnost, tzn. každá třída může mít maximálně jednoho předka. Třídy se definují inkrementální modifikací existujících tříd [3].

2.1.2 Zapouzdření

Klíčovou vlastností objektové orientace je zapouzdření. To znamená, že k obsahu objektu nemá přístup nikdo jiný než vlastník, vnitřní struktura objektu je skryta. Navenek se zobrazuje jen rozhraní objektu - zprávy, které lze objektu zaslat. Reakcemi na zprávy jsou metody definující chování objektu.

Zapouzdření je v OOPN realizováno množinou funkcí a množinou míst, která jsou těmito funkcemi sdílena. Funkce se nazývají metody a sdílená místa atributy. Tato struktura tvoří třídu [3].

2.1.3 Polymorfismus

Obecně umožňuje tvorbu generického softwaru. Rozhodnutí, která metoda bude volána, probíhá dynamicky až za běhu programu (tzv. pozdní vazba). Poskytuje objektům volání metody se stejným jménem, ale rozdílnou implementací. Je možné poslat zprávu se stejným jménem objektům různých tříd, přičemž v těchto třídách mohou být definovány rozdílné metody pro zpracování této zprávy.

2.2 Struktura OOPN

Objektově orientovaná Petriho síť je množina tříd hierarchicky organizovaná podle vztahu dědičnosti. Jedna z tříd je prohlášena za počáteční třídu [3].

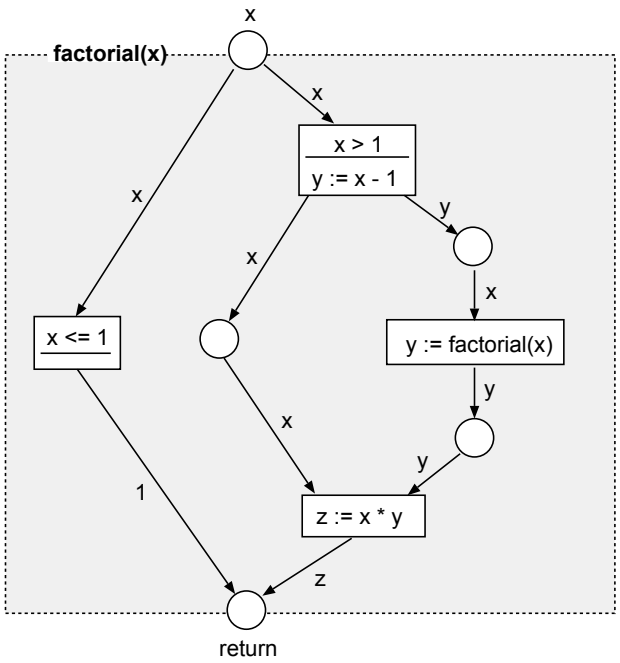
2.2.1 Třídy

Třída je tvořena:

- sítí objektu, která definuje reprezentaci instancí třídy (atributy)
- množinou sítí metod definující reakce na zprávy, které jsou zasílané z akcí přechodů
- množinou synchronních portů definujících reakce na synchronní zprávy, které jsou zasílané ze stráží přechodů [3].

2.2.2 Metody

Metody jsou reprezentovány funkcemi. Uvažujme funkci *max (a, b)*, která vrací maximum ze zadaných parametrů *a* a *b*. Má-li být tato funkce implementována Petriho sítí, musí mít síť odpovídající počet vstupních (parametrových) portů (míst) pro předání parametrů *a*, *b* a jeden výstupní port pro předání výsledku (*return*). Funkce lze volat uvnitř akce přechodu. Sítě metod mohou v rámci třídy sdílet místa sítě objektu [3]. Na obrázku 2.1 je příklad metody provádějící výpočet faktoriálu.



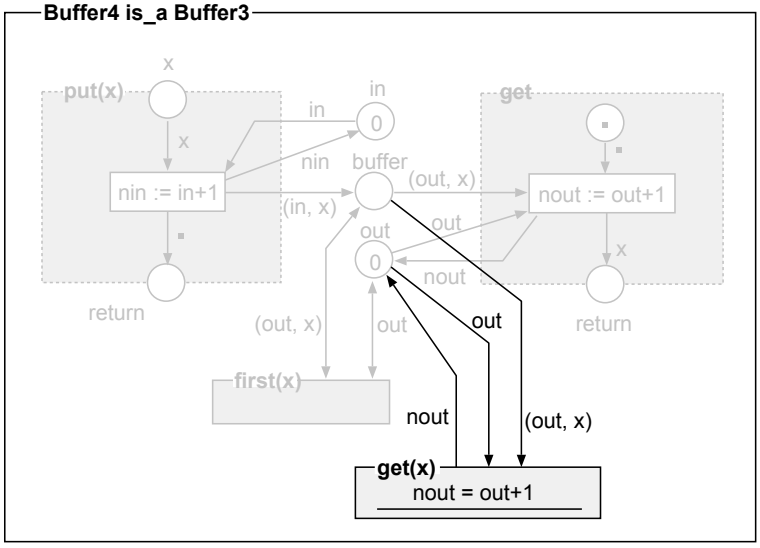
Obrázek 2.1: Funkce *factorial(x)* tvořena Petriho sítí, převzato z [3].

2.2.3 Synchronní porty

Synchronní porty jsou součástí specifikace tříd. Synchronní porty slouží k synchronní interakci mezi objekty. Synchronní porty mají charakter přechodů i metod [5]. Slouží k testování a případné změně stavu volaného objektu. Nejsou to sítě, takže neobsahují místa a přechody. Podobně jako přechod, může být synchronní port propojen hranami s místy sítě objektu a může obsahovat stráž, nemůže mít definovanou akci. K synchronnímu portu je podobně jako k síti metody, připojen vzor zprávy, na kterou reaguje, a může být volán zasláním zprávy ze stráže libovolného přechodu [3].

Přechod může být proveden právě tehdy, pokud mohou být provedeny všechny volané synchronní porty. Synchronní port může být volán s předem vyhodnocenými parametry, ale i s volnými proměnnými [5].

Speciálním případem synchronního portu je predikát, který neovlivňuje stav objektu (je propojen s místy sítě objektu pouze testovacími hranami) [3].



Obrázek 2.2: Ukázka třídy *Buffer4* se synchronním portem *get(x)*, převzato z [3].

2.2.4 Negativní predikáty

Negativní predikát je zvláštním případem synchronního portu. Jeho sémantika je ale narozdíl od synchronního portu převrácená - přechod může být proveden, pokud nemůže být proveden negativní predikát [5].

Kapitola 3

Jazyk PNML

Jedná se o jazyk pro popis Petriho sítě. PNML je založen na XML, což přináší řadu výhod, například přenositelnost, rozšiřitelnost do budoucna pro nové druhy Petriho sítí apod. Současně také existuje spousta programových nástrojů a API pracujících a validujících formát XML. Výhodou může být i výměna modelů Petriho sítí mezi různými nástroji. Za účelem podpory různých typů Petriho sítí se PNML zaměřuje na univerzálnost a flexibilitu.

3.1 Vlastnosti PNML

Flexibilita znamená, že jazyk PNML by měl být schopný reprezentovat libovolnou Petriho síť se všemi specifickými rozšířeními a vlastnostmi. PNML by neměl omezovat vlastnosti nějakého druhu Petriho sítě, nebo dokonce vyžadovat upuštění od specifických informací Petriho sítě při konverzi do PNML. Za účelem dosažení této flexibility, je Petriho síť považována za orientovaný graf se značkami, kde všechny specifické informace mohou být uloženy ve značkách. Značka může být spojena s uzlem, hranou nebo se sítí samotnou [2].

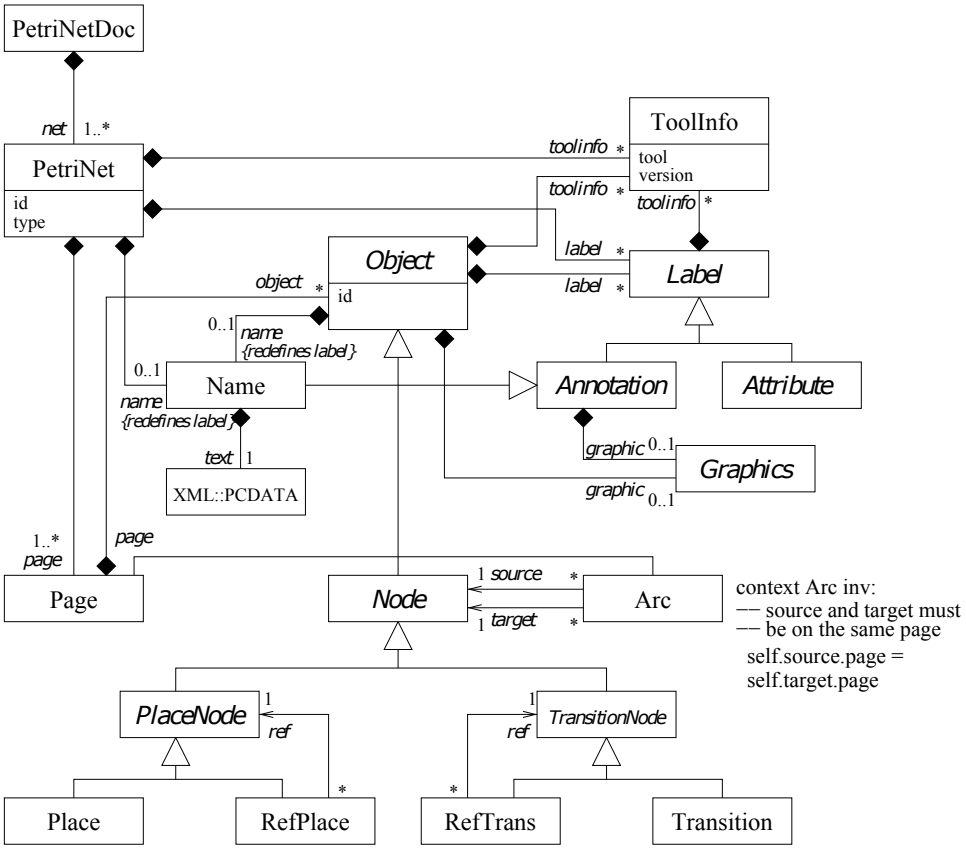
Pomocí jednotlivých XML elementů a atributů, které reprezentují prvky Petriho sítě, dovoluje jazyk PNML Petriho síť uložit v čitelném a snadno zpracovatelném formátu.

PNML podporuje definici různých typů Petriho sítí. Podobně jako v jazyce XML existují definiční soubory (s koncovkou *xsd*), u jazyka PNML jsou to soubory s koncovkou *PNTD* (*Petri net type definition*), někdy nazývány také jako meta modely. Vymezují platné značky pro patřičný typ Petriho sítě [1].

3.2 Struktura PNML

Pro poskytnutí a definování XML syntaxe PNML používá UML meta modely [4]. Základem PNML jsou části *Meta model*, *Feature Definition Interface* (dále FDI) a *Type Definition Interface* (dále TDI). Základní struktura PNML souboru a všechny koncepty, které jsou sdíleny všemi druhy Petriho sítí, jsou definovány meta modelem. FDI umožňuje definici nových vlastností Petriho sítí a TDI dovoluje definovat nové typy Petriho sítí [1]. Konkrétní XML syntax je pak definována mapováním konceptů těchto UML modelů na XML elementy. PNML je standardizován podle ISO/IEC 15909 jako syntax pro tři druhy Petriho sítí: *Place/Transition-Nets*, *High-level Petri Nets* a *Symetric Nets* [4], jejichž definiční meta modely jsou přístupné na referenčních stránkách PNML[1].

[1] http://www.pnml.org/grammar.php



Obrázek 3.1: PNML Core Model, převzato z [4].

3.2.1 Meta model

V této kapitole bude popsán základní meta model nazývaný jako *PNML Core Model*, ze kterého vychází a dědí další meta modely specifikující konkrétní typy Petriho sítí. Na nejvyšší úrovní meta modelu se nachází *Petri net file* (nebo také *PetriNetDoc*), který splňuje požadavky meta modelu. Tento dokument může obsahovat libovolný počet Petriho sítí.

Petriho sítě a objekty

Každá Petriho síť se skládá z objektů, které reprezentují grafovou strukturu Petriho sítě. Každý objekt má unikatní identifikátor, který může být využit pro odkazování se na tento objekt [1]. Navíc, každý objekt může obsahovat grafickou informaci, která definuje jeho pozici, velikost, barvu, tvar a další informace. Záleží také na typu objektu, jakou grafickou informaci ponese [2]. Objektem se zde rozumí prvky Petriho sítě - místo, přechod a hrana.

Stránky a odkazující uzly

PNML zahrnuje ještě objekt - *Page* (stránka), který může obsahovat další objekty, včetně dalších objektů stránka. Lze tak tvořit hierarchii pod-stránek. PNML vyžaduje, aby hrana propojovala uzly (místa a přechody) pouze na jedné stránce. Důvod pro toto omezení je takový, že hrana propojující uzly na různých stránkách nemůže být graficky zobrazena na jedné stránce [4].

V nástroji toto omezení nebude dodrženo, pokud jednotlivé stránky uvnitř PNML souboru definujícího třídu OOPN reprezentují metody. V tomto případě mohou hrany propojovat sdílená místa v síti objektu s přechody uvnitř metod a naopak. Aby šlo propojit uzly na různých stránkách, zástupce jednoho z těchto dvou uzlů bude zobrazen na té samé stránce jako druhý uzel. Tento zástupce se nazývá *reference node* - tzn. *reference place* a *reference transition*. Nejsou však dovoleny cyklické reference [4] [2].

Značení

Každý objekt může mít své značení *label* za účelem přiřazení dalších informací k objektu. Typicky značení reprezentuje jméno uzlu; počáteční značení místa; podmínku, časování či stráž přechodu nebo anotaci hrany. I Petriho síť samotná a její stránky mohou mít definovány nějaké značení (nazývány *global labels*), například deklarace funkcí [1].

PNML Core Model neklade na značky žádné omezení, tudíž *PNML Core Model* umožňuje reprezentovat libovolnou Petriho síť [2].

Jsou rozlišovány dva druhy značení - anotace a atributy. Anotace zahrnuje informace, které jsou typicky zobrazeny jako text u korespondujícího objektu. Uveďme například počáteční značení, stráž přechodu nebo časování přechodu. Narozdíl od anotace, atribut není zobrazen jako text u korespondujícího objektu, nýbrž nese informaci o grafické podobě objektu. Příkladem mohou být tvar a barva objektu, font písma, tloušťka hrany [4]. Třídy pro značení, anotaci a atribut jsou v meta modelu pouze abstraktní[2].

Grafická informace

Každý objekt a každá anotace obsahuje grafickou informaci. Pro prvky typu uzel (místo a přechod) se jedná o absolutní pozici. Hrana může být tvořena buď úsečkou mezi dvěma uzly, nebo lomenou čárou skládající se z více bodů, v tom případě grafická informace obsahuje seznam těchto bodů. Anotace objektu se zobrazuje vedle patřičného objektu - grafickou informací je pak relativní pozice ke korespondujícímu objektu. Dalším případem může být uchování informace o tvaru objektu, velikosti a barvě [1].

3.3 Technologie PNML

Existuje několik XML technologií, které mohly být využity pro implementaci PNML. Pro definici PNML dokumentu byla zvolena technologie *RELAX NG* (*REgular LAnguage for XML Next Generation*), jejíž definiční schéma tvoří vzor PNML dokumentu. Tento koncept byl nutný pro oddělení PNML meta modelu od definice typu Petriho sítě (PNTD). Existuje spousta nástrojů pro podporu XML schémat, takže PNML může být převeden na XML schéma (podobný postup byl využit při implementaci nástroje viz kapitola 5.3).

[2]Konkrétní třídy jsou definovány příslušným typem Petriho sítě.

Příslušnou RELAX NG gramatiku včetně příkladů Petriho sítí v jazyce PNML lze nalézt na referenčních stránkách PNML[3].

3.4 Mapování PNML meta modelu na PNML elementy

Každá konkrétní třída PNML meta modelu je přeložena na odpovídající XML element. Mapování překladu je zobrazeno v tabulce 3.1. Tyto XML elementy tvoří klíčová slova jazyka PNML [1].

Nenachází se zde žádné elementy pro značení, protože meta model pro ně nedefinuje konkrétní třídy. Konkrétní třídy pro značení jsou definovány konkrétními typy Petriho sítí.

Class	**XML element**	**XML Attributes**
PetriNetDoc	<*pnml*>	xmlns: anyURI
PetriNet	<*net*>	id: ID
		type: anyURL
Place	<*place*>	id: ID
Transition	<*transition*>	id: ID
Arc	<*arc*>	id: ID
		source: IDRef (Node)
		target: IDRef (Node)
Page	<*page*>	id: ID
RefPlace	<*referencePlace*>	id: ID
		ref: IDRef (Place or RefPlace)
RefTrans	<*referenceTransition*>	id: ID
		ref: IDRef (Transition or RefTrans)
ToolInfo	<*toolspecific*>	tool: string
		version: string
Graphics	<*graphics*>	

Tabulka 3.1: Překlad PNML meta modelu na PMNL elementy, převzato z [2].

Parent element class	**Sub-elements of <*graphics*>**
Node, Page	<*position*> (required)
PetriNet	<*dimension*>
	<*fill*>
	<*line*>
Arc	<*position*> (zero or more)
	<*line*>
Annotation	<*offset*> (required)
	<*fill*>
	<*line*>
	<*font*>

Tabulka 3.2: Elementy v <*graphics*> elementu v závislosti na rodičovském elementu, převzato z [1].

[3] http://www.pnml.org/version-2009/version-2009.php

XML element	Attribute	Domain
<*position*>	x	decimal
	y	decimal
<*offset*>	x	id: ID
	y	decimal
<*dimension*>	x	nonNegativeDecimal
	y	nonNegativeDecimal
<*fill*>	color	CSS2-color
	image	anyURI
	gradient-color	CSS2-color
	gradient-rotation	vertical, horizontal, diagonal
<*line*>	shape	line, curve
	color	CSS2-color
	width	nonNegativeDecimal
	style	solid, dash, dot
<*font*>	family	CSS2-font-family
	style	CSS2-font-style
	weight	CSS2-font-weight
	size	CSS2-font-size
	decoration	underline, overline, line-through
	align	left, center, right
	rotation	decimal

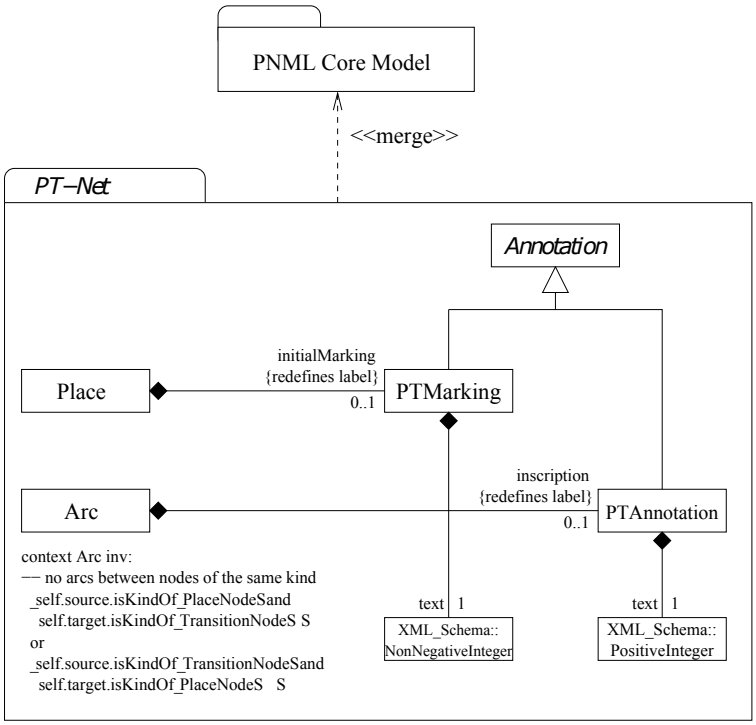
Tabulka 3.3: Grafické PNML elementy, převzato z [4].

PNML elementy a značení obsahují grafickou informaci. Struktura <*graphics*> elementu závisí na elementu, v němž se nachází. Konkrétní mapování zobrazuje tabulka 3.2. V každém uzlu je vyžadován element <*position*> značící absolutní pozici. Naopak element *offset* definuje relativní pozici (ke korespondujícímu uzlu) a je vyžadován u anotací [1]. Ostatní elementy jsou volitelné. Pro hranu může být přítomno 0 a více elementů *position*, protože hranu může tvořit lomená čára. Každý element *position* udává jeden z prostředních bodů. Každá absolutní i relativní pozice představuje bod v kartézských souřadnicích (x, y) [1].

Tabulka 3.3 zobrazuje atributy, které se mohou vyskytovat uvnitř jednotlivých grafických elementů. Sloupec *Domain* obsahuje datový typ XML schématu, Kaskádových stylů (CSS2) nebo vyjmenování přípustných hodnot pro daný atribut [2]. Element <*dimension*> definuje výšku a šířku uzlu. V závislosti na poměru výšky a šířky bude místo zobrazeno jako elipsa nebo kruh, přechod zase jako čtverec nebo obdélník. Elementy <*fill*> a <*line*> definují vnitřek a obrys korespondujícího elementu. Pro text značení slouží element <*font*> [1].

3.5 Formát PNML pro OOPN

Nyní bude popsán přesný formát PNML pro ukládání a přenos Objektově orientovaných Petriho sítí. Formát vychází z definice PNML pro *Place/Transition nets*.



Obrázek 3.2: The package *PT-Net*, převzato z [4].

3.5.1 PNML pro *P/T* Petriho sítě

PNML pro *P/T* Petriho sítě rozšiřuje tzv. *PNML Core Model* uvedený na obrázku 3.1. Každé místo obsahuje počáteční značení (přirozené číslo) v podobě elementu *PTMarking*. Stejně tak, každá hrana musí mít své značení (tentokrát nenulové přirozené číslo) v podobě elementu *PTAnnotation*. Narozdíl od *PNML Core Model*u, *P/T* sítě nedovolují spojit hranou místo s místem a přechod s přechodem. Na obrázku 3.2 jsou znázorněny ještě třídy *Place* a *Arc*, které jsou importovány z *PNML Core Model*u za účelem definování konkrétního značení pro tyto třídy pro tento konkrétní typ Petriho sítě [4].

3.5.2 Modifikace PNML pro uložení OOPN

PNML pro *P/T* sítě stále nedokáže pokrýt veškeré vlastnosti Objektově orientované Petriho sítě. Proto bylo nutné tento formát ještě rozšířit o několik parametrů. V této sekci budou tato rozšíření popsána pouze textově. Výstupem těchto modifikací je XML definiční schéma jazyka PNML pro popis Objektově orientovaných Petriho sítí. Na základě získaného schématu bude provedena implementace ukládání a načítání OOPN v jazyce PNML, podrobně vysvětleno v kapitole 5.3.

Synchronní porty

OOPN definuje narozdíl od *P/T* sítí synchronní porty (vysvětleno v 2.2.3). Synchronní porty mají charakter metod a přechodů, proto je lze i podobným způsobem ukládat. K definici PNML stačilo přidat novou třídu *SyncPort*, jejímž předkem je třída *Node* (také předek objektu *Transition*).

Synchronní port má definovanou stráž, takže potřebuje tuto informaci uchovávat. Za tímto účelem stačilo přidat objekt *PTMarking* do nově vytvořené třídy *SyncPort*.

Negativní predikáty

Jelikož je negativní predikát pouze speciálním případem synchronního portu, není potřeba definovat samostatnou třídu. V jazyce PNtalk se odlišuje od synchronního portu pouze klíčovým slovem. Synchronní port je značen klíčovým slovem *sync*, negativní predikát klíčovým slovem *inhibitor*. Pro reprezentaci negativního predikátu v PNML stačí třída *SyncPort*.

Přechody - stráž a akce

PNML pro popis *P/T* sítí neuvažuje žádné dodatečné ukládání informací (např. značení) o přechodu. OOPN definuje pro přechod stráž a akci. Podobně jako u synchronních portů, stačilo využít existující třídy *PTMarking* pro uchování těchto dvou parametrů.

Testovací hrany

Dalším rozšířením OOPN je přítomnost testovacích hran (více v 4.1.2). V tomto případě lze pro jednoduchost přidat příznak pro rozlišení, zda se jedná o testovací hranu.

Příznaky pro rozlišení vstupních a výstupních míst

Jako v předchozím případě nás zajímá pouze binární informace. Metody jsou mimo jiné tvořeny i vstupními (parametrovými) místy a výstupním (*return*) místem. Nástroj neumožňuje odstranění vstupních a výstupních míst metody, protože by mohlo dojít k porušení konzistence mezi vzorem zprávy a parametrovými místo metody. Součástí definice metody je vzor zprávy (viz 4.1.3), v němž jsou jména vstupních míst uvedena jako formální parametry. Proto také dochází k rozlišení typu míst.

Import míst

Síť metody může být v rámci definice třídy propojena se sítí objektu hranami mezi místy sítě objektu a přechody sítě metody. Sdílená místa jsou v PNtalku specifikována jen v síti objektu [3].

Metoda může přistupovat k místům objektu a měnit je. Nástroj taková místa musí ukládat dvakrát (pro třídu a pro metodu), z důvodu zachování rozdílných pozic prvků uvnitř třídy a metody. Uvnitř sítě metody je uložen stejný objekt místa jako v síti třídy. Metoda navíc ještě uchovává seznam referencí importovaných míst, pro potřeby rozlišení, že se jedná o importované místo.

Kapitola 4

Jazyk a systém PNtalk

V této kapitole bude popsán jazyk, který je konkrétní implementací Objektově orientovaných Petriho sítí. PNtalk je inspirován jazykem Smalltalk. Tato kapitola čerpá z [3]. Programování v PNtalku spočívá ve vytváření tříd neprimitivních objektů a jejich zařazování do hierarchie dědičnosti. Třídy jsou definovány množinami Petriho sítí, které se skládají z prvků místo, přechod, synchronní port, negativní predikát a hrana. Tyto prvky mohou mít přiřazeny popisy v jazyce PNtalk.

4.1 Jazyk PNtalk

4.1.1 Zaslání zprávy

Prvky Petriho sítě mohou mít přiřazeny výrazy pro zaslání zprávy nějakému objektu. Tyto výrazy mohou být součástí stráže i akce přechodu. Syntax zaslání zprávy je následující

⟨ *adresát* ⟩ ⟨ *zpráva* ⟩.

Zpráva se skládá ze selektoru a argumentů. Podle tvaru selektoru rozlišujeme zprávy:

- Unární - Zpráva nemá žádné argumenty.
- Binární - Selektor zprávy je jeden ze symbolů:

  *+ - / * = < > ~= <= >= & | // \\ , == ~==*

  Následuje jeden argument. Pro zaslání zprávy *1 + 2* představuje *1* adresáta zprávy *+ 2*, *+* selektor zprávy a *2* argument.
- Zprávy s klíčovými slovy - zpráva obsahuje jeden nebo více klíčů, což jsou identifikátory, zakončené dvojtečkou, s jejich vlastními argumenty.

Jsou-li adresát zprávy i argumenty termy, jde o jednoduché zaslání zprávy. PNtalk připouští i složené zaslání zprávy - jako příjemnce nebo argument zprávy je uvedeno opět zaslání zprávy. Pořadí vyhodnocování zpráv lze ovlivňovat použitím závorek.

Zasílání zpráv se vyskytuje ve výrazech, které specifikují stráže a akce přechodů. Stráž přechodu je tvořena sekvencí výrazů, které jsou odděleny tečkou. Každý dílčí výraz je zaslání zprávy. Akce přechodu může obsahovat výraz přiřazení ve formátu

⟨ *proměnná* ⟩ := ⟨ *zaslání zprávy* ⟩.

4.1.2 Místa, přechody a hrany

Sítě se v PNtalku specifikují graficky. Místo je reprezentováno kružnicí nebo elipsou. Každé místo má jméno a může mít specifikováno počáteční značení. Přechod je reprezentován čtyřúhelníkem. I přechod má jméno, může mít specifikovanou stráž a akci. Místa a přechody propojují hrany. Ke každé hraně je připojen hranový výraz reprezentující multimnožinu mající tvar n_1 ‘ c_1, n_2 ‘ c_2, …, n_m ‘ c_m,

kde n_i je term a c_i je term nebo seznam.

Rozlišujeme tři druhy hran:

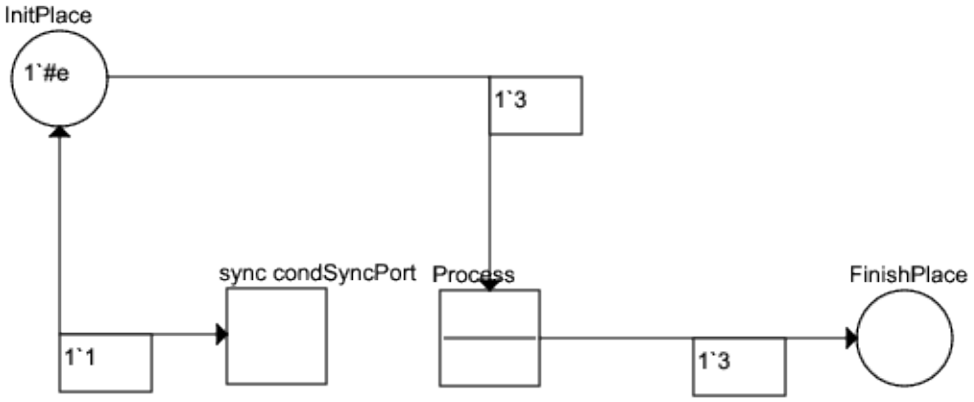
- Vstupní - orientovaná z místa do přechodu, reprezentuje vstupní podmínku přechodu.
- Výstupní - orientovaná z přechodu do místa, reprezentuje výstupní podmínku přechodu.
- Testovací - obousměrná hrana, jejich sémantika je v případě atomického provedení přechodu ekvivalentní dvojici opačně orientovaných hran, ohodnocených stejným hranovým výrazem jako testovací hrana. V případě neatomického provedení přechodu, tj. když je přechodem invokována síť metody, vstupní hrana přechodu odebere ze vstupního místa potřebné značky a výstupní hrany se uplatní až po ukončení invokace (v tomto případě není ekvivalentní dvojici opačně orientovaných hran). Testovací hrana stále ponechává testované značky v přislušném místě.

Na obrázku 4.1 je znázorněna elementární třída *PTClass*, jejíž definice v jazyce PNtalk odpovídá:

```
class PTClass is_a PN
object

    place InitPlace(1‘#e)
    place FinishPlace()
    trans Process
        precond InitPlace(1‘3)
        postcond FinishPlace(1‘3)
    sync condSyncPort
        cond InitPlace(1‘1)
```

Třída obsahuje místa *InitPlace* s počátečním značením *1‘#e* a *FinishPlace* bez počátečního značení. Přechod s názvem *Process* nemá definovanou stráž ani akci. Hrana, která propojuje místo *InitPlace* s přechodem, má značení *1‘3*. Z pohledu přechodu se jedná o vstupní hranu, proto je uvozena klíčovým slovem *precond*. Hrana propojující přechod s místem *FinishPlace* je uvozena klíčovým slovem *postcond*, z pohledu přechodu se jedná o výstupní hranu. Ve třídě je ještě definován synchronní port, který je propojen s místem *InitPlace* testovací hranou. Testovací hrany jsou značeny slovem *cond*. Syntax jazyka PNtalk, se kterou nástroj operuje, je uvedena v příloze B.



Obrázek 4.1: Definice třídy PTClass, vyexportováno z implementovaného nástroje.

4.1.3 Metody

Metody reprezentují reakci objektu na příchozí zprávu. V definici třídy se může nacházet libovolný počet sítí metod. Sítě metody neobsahují synchronní porty a negativní predikáty.

Síť metody je tvořena z míst, přechodů a hran, stejně jako síť objektu, ale ke každé síti metody je přiřazen vzor zprávy, jejíž přijetí objektem vyvolá dynamické vytvoření instance sítě metody. Vzor zprávy je složen ze selektoru zprávy a formálních parametrů. Podmnožina (může být i prázdná) míst sítě objektu jsou parametrová místa, která slouží k předání parametrů při vyvolání metody. Jejich jména musí odpovídat jménům formálních parametrů ve vzoru zprávy. Každá síť metody obsahuje jedno výstupní místo (pojmenované *return*), které slouží k předání výsledku vyhodnocení zprávy volajícímu objektu.

4.1.4 Třídní hierarchie

Jména tříd jsou globálně dostupná. Model v PNtalku se skládá z množiny tříd. Jedna z nich je deklarována jako počáteční třída. Jak už bylo zmíněno v kapitole 2.2, třída je složena ze sítě objektu, množiny sítí metod a synchronních portů. V PNtalku přibývají ještě konstruktory. Každá třída OOPN má právě jednoho předchůdce v hierarchii dědičnosti (jednoduchá dědičnost). Kořenová třída je třída *PN*, ze které dědí všechny třídy definované Petriho sítěmi. Třída *PN* je prázdná.

Každá třída dědí od své nadtřídy (přímé i nepřímé) strukturu sítě objektu a všechny metody, konstruktory a synchronní porty. Všechny tyto zděděné vlastnosti od třídy *PN* a jejích potomků může třída předefinovat. Metody, konstruktory a synchronní porty mohou být předefinovány uvedením nových definic pro stejný selektor zprávy, jako u nadtřídy. Síť objektu může být předefinována po částech redefinicí jednotlivých míst a přechodů, tj. uvedením míst a přechodů se stejným jménem jako v nadtřídě. V případě redefinice přechodu jsou současně redefinovány i okolní hrany (hrany jsou chápány jako součást přechodů). V případě redefinice místa zůstávají okolní hrany beze změny.

4.2 Simulační server PNtalk 2.0

Simulační server PNtalk 2.0 je externí aplikace, rozšiřuje původní implementaci nástroje *PNtalk system*. Aplikace je simulačním frameworkem, který je založený na formalismu OOPN[1]. Umí provádět simulace nad vytvořenými modely, spouštět jednotlivé simulační kroky a sbírat výsledky simulací a statistiky. Z tohoto pohledu je výhodné propojit server s aplikací pro vytváření a editaci Objektově orientovaných Petriho sítí.

4.2.1 Platforma

Jak bylo uvedeno v kapitole 4, jazyk PNtalk je založený na objektově orientovaném jazyce Smalltalk. Rovněž i PNtalk 2.0 je napsán v jazyce Smalltalk. Pro jeho běh je zapotřebí mít nainstalovaný virtuální stroj *Pharo*. Virtuální stroj lze získat na oficiálních stránkách Pharo[2].

4.2.2 Komunikační protokol

Server operuje pouze s jazykem PNtalk (viz 4), nejsou pro něj podstatné veškeré informace, které zahrnuje např. jazyk PNML. Před odesláním Objektově orientované Petriho sítě z nástroje na server se provede získání příkazů jazyka PNtalk. Pro komunikaci mezi těmito dvěma aplikacemi lze využít sokety. Nástroj pro editaci OOPN tvoří klienta.

4.2.3 Popis komunikačního protokolu

Server má definovány tyto příkazy:

> *existModel*, *addClass*, *delClass*, *getClass*, *newSimulation*, *getState*, *stepSim*, *simulateForSteps*, *registerEvent*, *simulateToEvent*, *destroySim*, *stats*, *listClasses*

Každý z nich má ještě specifikovány upřesňující parametry. Pro potřeby komunikace s nástrojem stačí podporovat tři příkazy *addClass*, *getClass* a *listClasses*.

Notace zápisu

(nazev_pozadavku, param1, param2, ...) => (status, res1, res2, ...)

listClasses

Slouží pro poskytnutí seznamu tříd, které lze ze serveru získat.

> Formát zápisu:
> *(listClasses) => (seznam trid dostupnych ke stazeni ze serveru)*

Příklad:

Struktura požadavku	Odpověď ze serveru
listClasses	*OK Ackermann R1 TestAckermann*

[1] http://perchta.fit.vutbr.cz/pntalk2k/
[2] https://pharo.org

getClass

Slouží pro získání třídy Objektově orientované Petriho sítě z repozitáře serveru. Zdrojový kód třídy bude získán v jazyce PNtalk.

Formát zápisu:
(getClass, 'nazev tridy') => (zdrojovy kod tridy v jazyce PNtalk)

Příklad:

Struktura požadavku
getClass
R1

Odpověď ze serveru
OK
class R1 is_a PN
object

```
place p1(1‘#e)
place p2()
place p3()
place p4()
trans t1
    precond p1(1‘#e)
    postcond p2(1‘10)

trans t2
    precond p2(1‘v)
    postcond p3(2‘v)

trans t3
    precond p3(1‘v)
    postcond p4(1‘v)
```

addClass

Slouží pro přidání třídy Objektově orientované Petriho sítě do repozitáře na serveru. Kód třídy musí být zapsán v jazyce PNtalk.

Formát zápisu:
(addClass, 'zdrojovy kod tridy v jazyce PNtalk') => ()

Příklad:

Struktura požadavku
addClass
'class TestAckermann is_a PN
object

```
place p1(1‘#e)
place p2()
place p3()
place x(1‘1, 1‘2)
place y(2‘2)
trans t1
    precond p1(1‘#e)
    action {
        ack := Ackermann new .
}
    postcond p2(1‘ack)

trans t2
    precond p2(1‘ack), x(1‘x), y(1‘y)
    action {
        z := ack x: x y: y .
}
    postcond p3(1‘z)'
```

Odpověď ze serveru
OK

Pozn.: Zdrojové kódy jsou převzaty z repozitáře tříd na serveru. Pro přehlednost jsou naformátovány. Při komunikaci se serverem nejsou ve zdrojovém kódu jazyka PNtalk přítomny znaky konce řádku. Znaky konce řádku pouze oddělují jednotlivé složky požadavku. Vzhledem k jednoduchosti komunikačního protokolu je velmi snadné rozšířit nástroj o podporu dalších akcí komunikace se simulačním serverem PNtalk 2.0.

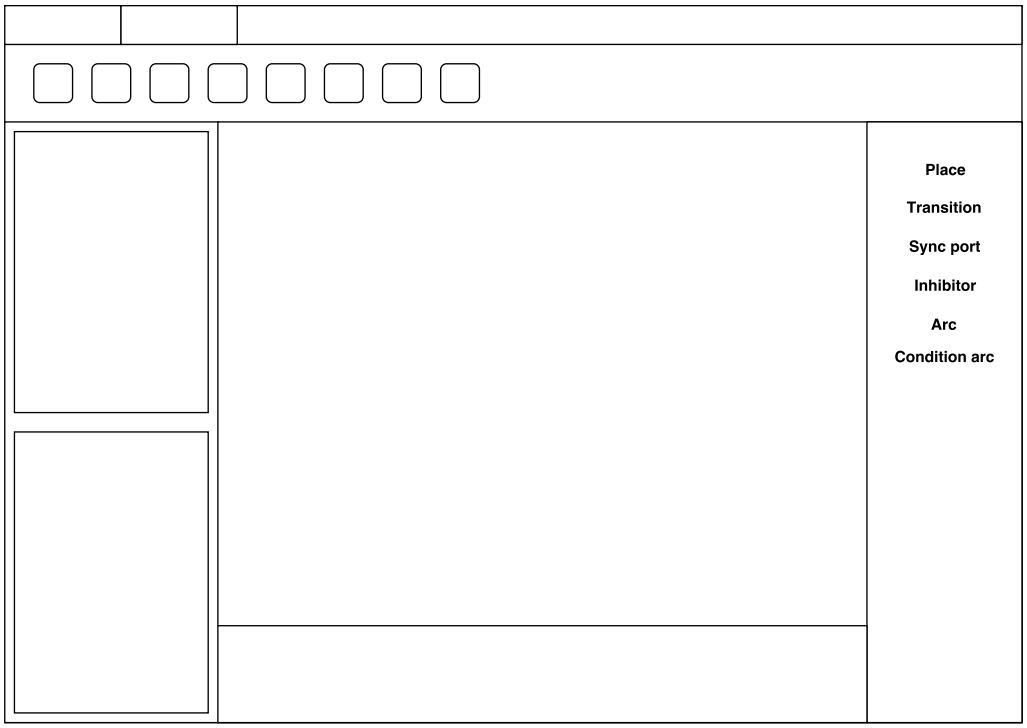
Kapitola 5

Návrh a implementace

V této kapitole bude popsán návrh nástroje společně s architekturou, která bude doplněna o diagram tříd. Poté bude rozebrána implementace nástroje včetně hlavních problémů a jejich řešení.

5.1 Grafické uživatelské rozhraní

Nástroj byl po vzhledové stránce a uspořádání grafických prvků inspirován existujícími editory a IDE, Eclipse a Netbeans. V horní části se nachází lišta s tlačítky pro nejčastěji používané akce jako tvorba nového souboru, otevření OOPN třídy, uložení, nápověda apod.



Obrázek 5.1: Návrh grafického uživatelského rozhraní.

V levé části zobrazují dva panely repozitář otevřených OOPN tříd. Oba panely tvoří stromovou strukturu, kde listy stromu představují metody tříd. První panel se jeví jako seznam otevřených tříd, pokud třída obsahuje minimálně jednu metodu, lze uzel rozbalit a přistoupit k metodám. Druhý panel zobrazuje otevřené OOPN třídy ve stromu dědičnostní hierarchie. Kořenem stromu je třída, tvořící vrchol dědičnostní hierarchie, typicky třída *PN*. Uzly kořene tvoří třídy dědící z této třídy. Třída se v tomto stromu zobrazí pouze tehdy, pokud je její nadtřída už otevřena, pokud není, nemůže být ve stromu zařazena. Strom také zobrazuje pouze přímé nadtřídy. Po každém otevření nové třídy se stromy sestaví znovu včetně znovuuspořádání tříd v dědičnostní hierarchii. Uzly jsou propojeny odkazy s objekty Objektově orientovaných Petriho sítí, takže změna stejné třídy v jednom stromě se projeví i v druhém stromě.

Centrální plochu pokrývá pracovní panel pro editaci sítě třídy či metody. Samozřejmě musí být třída nebo metoda otevřena, než do ní bude možné prvky přidávat. Otevření třídy a metody proběhne po vzoru z jiných IDE dvojitým poklepáním na odpovídající uzel v jednom ze stromů v levé části nástroje. Při otevírání jednotlivých tříd a metod se na pracovní ploše začnou přidávat odpovídající panely. Tyto panely lze po skončení práce a uložení opět zavírat. Zavírat lze i celé otevřené třídy uvnitř stromových zobrazení. Stačí na třídu kliknout pravým tlačítkem myši a vybrat možnost *Close class*, tato možnost byla opět inspirována existujícími IDE.

V metodě lze používat místa definovaná uvnitř třídy - vysvětleno v 3.5.2. Při otevřeném panelu metody lze místa třídy do metody importovat. Po kliknutí pravého tlačítka myši na pracovní ploše se zobrazí seznam míst nacházejících se uvnitř třídy, výběrem některého z nich se místo zobrazí i uvnitř metody.

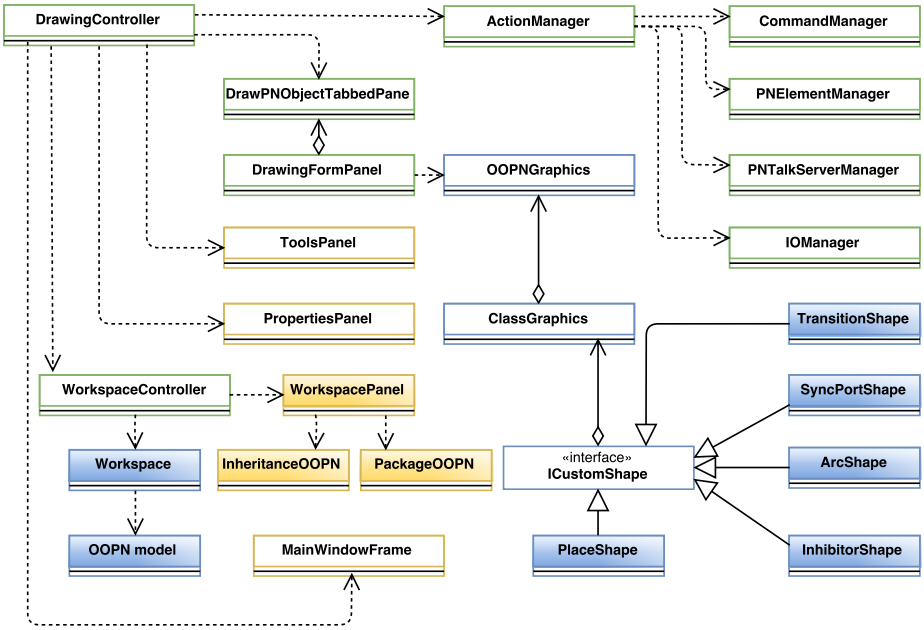
V pravé části se nachází panel s prvky Petriho sítě, tzn. na výběr se nabízí místo, přechod, synchronní port, negativní predikát (neboli *inhibitor*), vstupní/výstupní hrana a testovací hrana. Při kliku na jeden z elementů se tento prvek vybere a při kliku na pracovní plochu je možné jej vložit do sítě třídy či metody.

Spodní panel slouží pro editaci vložených prvků na pracovní ploše. Při kliku na libovolný grafický objekt se zobrazí korespondující panel s odpovídajícími parametry prvku. Zobrazené parametry lze poté libovolně upravit.

5.2 Architektura

Celý nástroj bude vytvořen pomocí architektonického návrhového vzoru *MVC* (*Model View Controller*). MVC rozděluje aplikaci na datový model, vzhled či grafické uživatelské rozhraní a řídicí logiku. MVC definuje tyto tři komponenty jako nezávislé, v nástroji tomu tak ale bohužel úplně neplatí z důvodu přesného určení jednotlivých prvků OOPN. Řídicí část (*controller*), která se stará o budování objektů Petriho sítí je tak silně závislá na vygenerovaných třídách z XML definičního schématu (konkrétní postup generování je uveden v sekci 5.3). MVC vzor byl zvolen z důvodu oddělení jednotlivých logických celků a kvůli jednodušší implementaci obsluhy akcí pro jednotlivé panely, které spolu komunikují.

Na obrázku 5.2 je zobrazen zjednodušený diagram tříd nástroje. Přítomny jsou zde pouze nejdůležitější třídy aplikace, modrá barva značí část modelu, žlutá vzhled a zelená řídicí třídy. Některé z panelů se chovají jako *controller* (například *DrawingFormPanel*), protože přijímají vstup v podobě akcí od uživatele. Pro zobrazení jednotlivých prvků Petriho sítě slouží třídy implementující rozhraní *ICustomShape*. Tyto nově definované grafické tvary patří do modelu. Do části modelu patří také všechny třídy vygenerované z definičního schématu jazyka PNML, v diagramu označeno souhrně pod třídou *OOPN model*.



Obrázek 5.2: Zjednodušený diagram tříd nástroje zobrazující nejdůležitější třídy. Barvy reprezentují jednotlivé části vzoru MVC: modrá - *model*, žlutá - *view*, zelená - *controller*.

Ke každé síti OOPN třídy existuje korespondující grafický objekt obsahující grafické objekty jednotlivých prvků Petriho sítě, v diagramu nazvaný jako *ClassGraphics*. Tyto grafické objekty se sdružují do třídy *OOPNGraphics*. Třída *OOPNGraphics* je svázána s panelem centrální plochy *DrawingFormPanel*, na kterém probíhá tvorba či editace sítě třídy nebo metody. Jednotlivé prvky Petriho sítě určené pro vkládání na editační plochu jsou dostupné z *ToolsPanel*u. *PropertiesPanel* umožňuje úpravu prvků Petriho sítě, měnit jejich jména a značení. *Workspace* označuje repozitář OOPN tříd a operace nad němi, zahrnující např. vkládání metod do tříd. *WorkspacePanel* zobrazuje dva stromy otevřených OOPN tříd a metod. *Workspace a WorkspacePanel* jsou řízeny *controller*em *WorkspaceController*. Do části *controller* patří ještě třída *ActionManager* provádějící většinu operací nad uživatelskými akcemi. Mezi ně patří například tvorba nových tříd a metod, ukládání sítí, realizace komunikace mezi nástrojem a PNtalk serverem a také vkládání nových prvků Petriho sítě v podobě grafických objektů. Ústředním *controller*em je *DrawingController*, který řídí celou aplikaci. Přijímá vstupy od uživatele a podle typu požadavku volá ostatní *controller*y.

5.3 Implementace nástroje

Nástroj byl implementován v programovacím jazyce Java. V této kapitole budou popsány klíčové postupy a algoritmy.

5.3.1 Zpracování PNML v Java

Jak bylo uvedeno v kapitole 3, jazyk PNML je založený na značkovacím jazyce XML. Existuje několik přístupů jak zpracovávat XML dokument v jazyce Java. Nejznámějšími jsou tzv. parsery *SAX* a *DOM*. Sax umožňuje sériový přístup k XML, dokument se rozdělí na

jednotlivé části podle obsahu elementů. Je výhodnější pro sekvenční čtení XML dokumentu. Narozdíl od DOM vyžaduje méně paměti. DOM se naopak hodí pro náhodný přístup k XML obsahu. Jedná se o API pro přístup a modifikaci obsahu XML dokumentu, ke kterému je přistupováno jako ke stromu. Vyžaduje načtení celého souboru do paměti.

5.3.2 JAXB

Existuje ještě další API, které se nazývá *JAXB*[1]. Umožňuje Java vývojářům mapovat Java objekty do XML struktury a naopak. Tyto dvě akce se nazývají *marshalling* (česky také serializace) a *unmarshalling* (deserializace). Java objekty mohou být mapovány na XML pomocí určitých *anotací* a pravidel. Stejně tak je možné zkonvertovat XML schéma na příslušné Java třídy. Anotace označují jednotlivé elementy, atributy a typy z XML formátu. Specifikováno je i pořadí jednotlivých elementů. Atributy elementů mohou nebo nemusí být povinné. Všechny tyto informace a omezení jsou ošetřeny pomocí anotací. JAXB lze také využít pro validování XML dokumentu nebo Java objektu proti XML schématu a zajištění, že XML struktury a Java objekty dodržují pravidla definovaná schématem.

Při generování Java tříd z XML schématu velmi záleží na struktuře schématu. Podle této struktury lze vygenerovat i třídy, které budou tvořit dědičnostní vztah. Ve schématu jsou rovněž definovány komplexní typy pomocí primitivních typů. Komplexní typy mohou být využity pro tvorbu dalších, ještě komplexnějších typů. Každý typ je převeden na odpovídající datový typ ve vygenerované Java třídě.

5.3.3 Definiční schéma jazyka PNML

Pro ukládání a načítání Objektově orientované Petriho sítě v jazyce PNML aplikace využívá služeb API JAXB. Na referenčních stránkách jazyka PNML jsou dostupné metamodely pro jednotlivé typy Petriho sítí[2] ve formátu *Ecore*. Tento formát lze vytvořit či editovat pomocí *Eclipse Modeling Framework*u[3]. Pro vygenerování korespondujících Java tříd pomocí JAXB bylo ještě nutné zkonvertovat metamodel z formátu Ecore na odpovídající XSD schéma a provést rozšiřující modifikace (vysvětleno v sekci 3.5.2). Z tohoto schématu už bylo snadné vygenerovat třídy pro ukládání jednotlivých prvků Petriho sítě.

5.3.4 Omezení respektovaná nástrojem

Jazyk PNML byl navržen tak, aby se dal snadno rozšířit pro uložení Petriho sítě libovolného typu. Pro uložení OOPN nástroj nepoužívá všechny vlastnosti a prostředky, které jazyk PNML nabízí. Některá omezení by se dala odstranit implementací rozšíření uvedených v sekci 6.2.

Mapování metod a tříd na PNML elementy

Nyní bude vysvětleno, jakým způsobem se jednotlivé prvky Petriho sítě (místo, přechod, hrana, metoda či třída) mapují na PNML elementy.

Struktura PNML souboru je tvořena objektem *PetriNetDoc*. Tento objekt může obsahovat libovolné množství objektů *PetriNet*, které nástroj zpracovává jako OOPN třídy. Nástroj dodržuje konvenci: jeden soubor, jedna třída.

[1] Java Architecture for XML Binding
[2] http://www.pnml.org/grammar.php
[3] http://eclipsesource.com/blogs/tutorials/emf-tutorial/

Objekt *PetriNet* se skládá z libovolného počtu objektů třídy *Page*. Pro OOPN je definováno, že síť třídy je mimo jiné tvořena také místy, přechody a hranami. Proto na této úrovni nelze pokládat objekty *Page* za metody (nešlo by tak ukládat místa, přechody, hrany, synchronní porty a negativní predikáty). Z tohoto důvodu nástroj považuje objekt *Page* stále za síť třídy (pouze jeden výskyt *Page* uvnitř *PetriNet*). Objekt *Page* dále umožňuje uložení libovolného prvku Petriho sítě, včetně sám sebe. Na této úrovni už nástroj na objekty *Page* pohlíží jako na metody, přesněji sítě metod. Do nadřazeného objektu *Page* pak ukládá i ostatní prvky Petriho sítě tvořící síť objektu.

Konzistence uloženého souboru

Při otevírání a ukládání OOPN tříd musí soubor obsahující třídu splňovat určitá omezení, které *Marshaller* a *Unmarshaller* (více v sekci 5.3.2) kontrolují. Velmi důrazně se doporučuje soubory neměnit ručně bez použití nástroje. I zdánlivě malá úprava může způsobit chybu při otevírání souboru v aplikaci.

Názvy tříd

Dalším z omezení je formát názvu třídy, který nesmí obsahovat bílé znaky. Porušením tohoto pravidla nebude možné korektně rozlišit třídu v rámci dědičnostní hierarchie. Každá třída dědí od nějaké nadtřídy. Tento vztah je zapsán formulí:

classChild is_a classParent

Nástroj rozlišuje, která z tříd je předek a která potomek. Mezery v názvu by způsobily kolize. Třídy se v nástroji zobrazují dvěma způsoby. První z nich je seznam otevřených tříd, druhý zobrazuje třídy právě podle dědičnostní hierarchie.

Každá třída musí mít také unikátní název. Nástroj nedovolí vytvořit či otevřít třídu se shodným jménem, jako má již existující otevřená třída.

Zděděné vlastnosti

Pokud třída dědí z nějaké nadtřídy místa, přechody, metody či jiné prvky, nejsou tyto prvky žádným způsobem zobrazeny v aktuální třídě. Toto omezení je způsobeno tím, že každá třída OOPN je definována v jednom odděleném souboru ve formátu PNML. Třída tedy nemá žádný přímý přístup ke zděděným vlastnostem. V nástroji v rámci otevřených tříd by musela být pomocná logika, která by přiřazovala zděděné vlastnosti konkrétním třídám, kde by se tyto vlastnosti daly nadále upravovat. Pokud by ale uživatel některou z nadtříd nástroji nedodal a podtřídy by se nějakým způsobem odkazovaly na prvky této nadtřídy, způsobilo by to konflikt a nenalezení požadovaných zdrojů.

Kontrola značení prvků Petriho sítě

Aplikace neprovádí žádnou kontrolu (syntaktickou ani sémantickou) značení míst, přechodů, synchronních portů a hran. Uživatel může prvkům Petriho sítě přiřadit libovolný řetězec. Nevýhodou může být převod těchto značení do jazyka PNtalk a následný přenos na simulační server. Pokud některé značení nebylo korektně uvedeno a server zdrojový kód nepřijme, uživatel nezjistí, kde přesně se chyba vyskytuje.

5.3.5 Grafické objekty

Grafické objekty na pracovní ploše zobrazují jednotlivé prvky Petriho sítě. Bylo použito několik algoritmů pro práci s grafickými objekty. Každý otevřený panel Petriho sítě ukrývá buď OOPN třídu nebo metodu zároveň s její třídou. Pro každý z těchto dvou objektů existuje vlastní *Graphics* objekt. V Graphics objektu se nachází všechny grafické objekty popisující Petriho síť. Při libovolném otevřeném panelu a pohybu myši se z Graphics objektu vybere aktivní grafický objekt pro další interakce, například posunutí objektu na pracovní ploše.

Propojení grafických objektů

Grafické objekty jsou různorodé, přechod je reprezentován čtyřúhelníkem, místo kruhem nebo elipsou. Propojení těchto objektů pomocí hrany vyžaduje rozdílné algoritmy pro výpočet průsečíku hrany a okraje místa či přechodu. Hrana je reprezentována úsečkou.

Průsečík úsečky s obvodem čtyřúhelníka

Nalezení průsečíku hrany a okraje čtyřúhelníka lze zjednodušit na výpočet průsečíků dvou úseček. Každou hranu čtyřúhelníka lze považovat za úsečku. Úsečka hrany propojující čtyřúhelník (přechod) má koncový bod uprostřed čtyřúhelníka reprezentujícího přechod, tzn. v každém případě je nalezen průsečík s některým z okrajů čtyřúhelníka.

Průsečík úsečky s obvodem elipsy

Kruh je z pohledu geometrie speciálním případem elipsy, proto lze použít stejný algoritmus pro výpočet průsečíku.

Parametrické vyjádření přímky pro dva body se souřadnicemi (x_1, y_1) a (x_2, y_2) a parametrem t, který nabývá hodnot v rozmezí $< 0, 1>$:

$$x(t) = x_1 + (x_2 - x_1)t \tag{5.1}$$

$$y(t) = y_1 + (y_2 - y_1)t \tag{5.2}$$

Pro zjednodušení výpočtu průsečíku dojde k přepočítání polohy elipsy tak, aby její střed byl v bodě $S = [0, 0]$. Po získání souřadnic průsečíku se délka posunutí přičte k souřadnicím. Rovnice elipsy se středem v bodě $S = [0, 0]$:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} \tag{5.3}$$

kde:

- a je délka hlavní poloosy
- b je délka vedlejší poloosy
- $[x, y]$ jsou souřadnice libovolného bodu elipsy

Po dosazení rovnic 5.1 a 5.2 do 5.3 získáme kvadratickou rovnici, z níž pokaždé získáme pouze jeden průsečík (dáno koncovým bodem hrany, který je vždy ve středu elipsy).

Odstranění grafických objektů

Grafické objekty lze z pracovní plochy libovolně odstraňovat, přičemž se zároveň odstraní i odpovídající prvky Petriho sítě z definice třídy nebo metody. Smazáním místa nebo přechodu se odstraní i hrany vedoucí z a do tohoto uzlu.

Změna zdroje nebo cíle hrany

Uživatel může měnit propojení grafických objektů. Změnu zdrojového nebo cílového uzlu, který hrana propojuje s jiným uzlem, lze provést kliknutím myši na okraj hrany a táhnutím k jinému uzlu. Nový uzel lze propojit, jen pokud operace odpovídá pravidlům Petriho sítí. Nelze propojit místo z místem, přechod s přechodem, přechod se synchronním portem apod.

5.3.6 Převod z PNML do PNtalk

Simulační server *PNtalk 2.0* nepřijímá všechny informace uchovávané jazykem PNML. Před odesláním OOPN třídy na server musí proběhnout konverze, která sestaví definici třídy v jazyce PNtalk. Algoritmus konverze prochází všechny objekty kořenového objektu *PetriNetDoc* a sestavuje zdrojový kód jazyka PNtalk dodržující syntaxi z přílohy B. Vynechává nepotřebné informace, zejména grafickou reprezentaci objektů, kterou simulační server nepotřebuje.

5.3.7 Převod z PNtalk do PNML

Konverze z jazyka PNtalk do PNML probíhá pomocí algoritmu rekurzivního sestupu. Tato technika byla zvolena jako přímočařejší postup než například parsování vstupu v jazyce PNtalk pomocí regulárních výrazů. Převod z PNtalku probíhá jen při přenosu OOPN ze simulačního serveru. Také je předpokládána stoprocentně správná syntax vstupu v jazyce PNtalk. Nástroj nekontroluje syntax jazyka, rekurzivní sestup byl použit pro detekci posloupnosti *tokenů*. Pro podporu syntaktické kontroly by byl vyžadován ještě lexikální analyzátor, který by rozlišoval typy tokenů. Syntax jazyka PNtalk je uvedena v příloze B.

Definice třídy v PNtalku neobsahuje informace ohledně grafické reprezentace objektů, nástroj přizpůsobuje velikost a pozici jednotlivých objektů. Objekty míst, přechodů, synchronních portů a negativních predikátů (jedním slovem uzly) jsou velké podle délky zdrojového kódu značení v jazyce PNtalk, v případě prázdnosti mají nastavenu výchozí velikost. Značení hran se zobrazují uvnitř čtyřúhelníka uprostřed hrany, který také mění svou velikost v závislosti na délce zdrojového kódu značení. Pozice míst se generují do řady s nastavenou výchozí mezerou. Ostatní uzly se generují do druhé řady. Hrany se vytvoří, až jsou všechny uzly vygenerovány.

Při běhu rekurzivního sestupu jsou očekávány určité tokeny podle definice syntaxe PNtalku. Při kontrole každého koncového pravidla (obsahující lexikální symbol) dochází ke zpracování symbolů, vytváření objektů Petriho sítě a formování prvků do PNML struktury.

Zpracování míst

Ve zdrojovém kódu jazyka PNtalk je místo zapsáno spolu se svým počátečním značením ve formátu:

placeName(initialMarking)

kde:

- *placeName* je jméno místa
- *initialMarking* reprezentuje počáteční značení místa

Po získání jména a značení se místo uloží do objektu korespondující třídy nebo metody.

Zpracování přechodů

Přechody jsou syntakticky podobné synchronním portům a negativním predikátům (ty mají navíc definovaný vzor zprávy, který se zpracovává v odděleném pravidle), proto i jejich zpracování je velmi podobné. Stráž a akci nástroj zpracovává ve formátu:

guard { guardExpr }

action { actionExpr }

kde:

- *guardExpr* a *actionExpr* vyjadřují libovolné značení skládající se z libovolných řetězců. Stráž a akce je ukončena tokenem *}*.

Součástí definice přechodu, synchronního portu a negativního predikátu jsou přítomny i hrany. Podle klíčových slov *cond*, *precond* a *postcond* se rozliší typ a orientace hrany.

Zpracování hran

Pro každé ze tří výše uvedených klíčových slov může být definováno více hran. Definice hrany v PNtalku je velmi podobné definici místa:

placeName(arcMarking)

kde:

- *placeName* je jméno místa, které je s přechodem, synchronním portem nebo negativním predikátem spojeno
- *arcMarking* reprezentuje značení hrany

Po rozparsování tokenu určujícího hranu se před uložením hrany ještě v definici třídy nebo metody provede nalezení zdrojového a cílového uzlu, které hrana propojuje.

5.3.8 Historie akcí

Jednou z nejdůležitějších vlastností z pohledu ovládání aplikace je podpora operace *undo*, tedy možnost vrátit provedené kroky zpět. Nástroj si uchovává historii některých provedených akcí, vrátit jdou však jen v některých případech.

Mezi ukládané akce patří pouze kroky provedené při vytváření či editaci sítě, tedy vložení prvku Petriho sítě, posun prvku, úprava jmen a značení, smazání prvku a importování místa ze sítě třídy do sítě metody. Tyto akce se vždy vážou ke konkrétnímu panelu otevřené sítě.

Za každým otevřeným panelem se schovává zásobník akcí. Pokud uživatel otevře nebo se přesune na jiný panel s jinou sítí, je aktivní nový zásobník akcí. Při provedení operace undo se zvrátí akce pro správný panel. Proto se neukládají akce jako vytvoření metody, stažení sítě ze serveru a ostatní akce, protože nesouvisí s otevřenou Petriho sítí. Stejně tak uživatel, pokud panel uzavře, nemůže operaci undo pro uzavřenou Petriho síť provést. Tento způsob omezení je zároveň ochranou před operací undo, kdy uživatel nevidí, jaké akce se vracejí zpět.

5.3.9 Kontext

Aplikace si udržuje globální kontext pro lepší přehlednost a intuici.

Kontext otevřených souborů

Nástroj si uchovává absolutní cestu souboru každé otevřené třídy. Díky této možnosti lze realizovat bezproblémové ukládání Objektově orientovaných Petriho sítí bez nutnosti pokaždé obtěžovat uživatele, aby zvolil kam danou síť uložit.

Kontext upravených souborů

Podobným způsobem je realizováno i hlídání upravených a zároveň neuložených souborů. Pokud uživatel v síti metody nebo v síti třídy provedl nějaké změny a neuložil je, při zavírání bude upozorněn a vybídnut k uložení provedených změn.

5.3.10 Externí knihovny a další zdroje

Při implementaci nástroje byly využity externí zdroje, zejména pro rozšiřující vlastnosti nástroje.

Export do formátu EPS

Pro export do formátu EPS byla využita knihovna *jlibeps*[4] ve verzi 0.1. Knihovna je šířena pod licencí *GNU GPL*. Umožňuje tvorbu EPS obrázků z Java objektů třídy *Graphics2D*. Knihovna je postavena na této třídě a jejich metodách, rozšiřuje ji o vlastní implementace metod *draw*[5] a tím tak sestaví korespondující PostScript objekt.

Definiční schéma PNML pro *P/T* sítě

Za účelem sestavení definičního XSD schématu pro generování Java tříd byl využit Meta model pro *P/T* sítě ve formátu ecore[6]. Tento soubor byl transformován pomocí *Eclipse Modeling Framework*u na XSD schéma, na kterém byly provedeny rozšiřující modifikace popsané v 3.5.2.

Ikony

Za účelem poskytnutí dostatečně intuitivního grafického uživatelského rozhraní byla použita sada ikon převzata z portálu *Icon Archive*[7].

[4] http://jlibeps.sourceforge.net/
[5] *drawLine*, *drawString*, *drawRect*, *drawOval*, *drawPolygon*, a další
[6] http://www.pnml.org/metamodels/placeTransition.ecore
[7] http://www.iconarchive.com/

Kapitola 6

Závěr

6.1 Výsledky

Nástroj umožňuje reprezentovat modely Objektově orientovaných Petriho sítí pomocí dvou jazyků PNML a PNtalk. Současně také poskytuje komunikační rozhraní pro výměnu sítí se simulátorem modelů systémem PNtalk. Uchování a přenos sítí pomocí jazyka PNML bylo docíleno modifikací formátu PNML pro P/T sítě a následnou implementací těchto úprav. Simulační server PNtalk operuje nad modely pouze ve zdrojovém kódu jazyka PNtalk. Spousta informací o síti, která jsou ukládána pomocí jazyka PNML např. informace o pozici prvku, velikosti, atp., je pro něj nepodstatná. Proto nástroj pro práci s OOPN musí provádět konverzi mezi jazyky PNML a PNtalk z důvody výměny modelů mezi těmito dvěma aplikacemi. Opačná operace, převod sítě z jazyka PNtalk do PNML, zase provádí vygenerování chybějících parametrů.

Doplňující možností ukládání sítí je ještě konverze modelů do EPS formátu realizována pomocí externí knihovny *jlibeps*.

Jelikož je každá třída OOPN reprezentována jedním souborem ve formátu PNML, nedokáže nástroj poskytnout zděděné vlastnosti z nadtříd potomkům. V každém souboru je uložena pouze definice aktuální třídy, nikoliv zděděné metody a atributy z nadtříd. Proto také nelze jakýmkoliv způsobem redefinovat zděděné prvky, například způsobem importování přechodu s názvem *InitProcess* z nadtřídy pro specifikování jiných hran, které by propojovaly přechod s novými místy.

Dědičnost a redefinice zděděných vlastností se projeví až při prohlížení či simulaci modelů v systému PNtalk. Simulační server poskytuje jména tříd globálně dostupná a například metodu, která má stejný název jako má metoda v nadtřídě, interpretuje jako redefinovanou.

6.2 Návrh rozšíření

Zde budou diskutována případná rozšíření pro poskytnutí pokročilých vlastností nástroje.

Mechanismus dědičnosti

V současné implementaci nástroje neexistuje jiná podpora dědičnosti než zobrazení tříd v hierarchii dědičnosti. Každá třída OOPN je tvořena jedním souborem ve formátu PNML a neobsahuje žádné informace o nadtřídě kromě jejího názvu. Z tohoto důvodu nelze přímo redefinovat zděděné metody a atributy a doplňovat rozšiřující vlastnosti.

Rozšíření pro podporu dědičnosti by mohlo umožnit sloučení všech otevřených tříd v nástroji a rozlišení, která třída dědí jaké vlastnosti. Následně by mohlo tyto vlastnosti promítnout do podtříd a umožnit jejich úpravu a redefinici. Současně by bylo vhodné sjednotit úložiště tříd například do databáze a tím se tak vyhnout potížím s chybějící třídou (viz 5.3.4) apod.

Propracovanější komunikace se simulačním serverem

Jak bylo zmíněno v kapitole 4.2, komunikační protokol je velmi jednoduchý. Bylo by tak možné rozšířit množinu akcí komunikace se serverem o další operace, například spuštění simulace vytvořeného modelu nebo sběr statistik.

Syntaktická kontrola popisů míst, přechodů a hran

Místa, přechody a hrany mají přiřazeny popisy v jazyce PNtalk. Další možným rozšířením nástroje by bylo provádět kontrolu těchto popisů při vytváření modelů. Když je v některém popisu chyba a uživatel třídu odesílá simulačnímu serveru, dostane jen chybovou zprávu *Compile error*. Uživatel ale už nezjistí, na kterém místě se chyba nachází a musí tak složitě debugovat celý model. Toto rozšíření by velmi urychlilo práci s nástrojem a pomohlo při analýze těchto chyb.

Propracovanější operace *undo* a podpora *redo*

Nynější implementace nástroje uchovává jen některé provedené akce pro podporu operace undo. Bylo by vhodné algoritmus historie akcí rozšířit o uchovávání všech akcí včetně vytváření metod, stahování tříd z PNtalk serveru a dalších. Současně se k historii akcí vztahuje ještě operace redo, kterou nástroj nyní nedisponuje.

Znovupoužitelný formát PNML

V podkapitole 3.5 byl popsán formát jazyka PNML použitý pro ukládání tříd Objektově orientovaných Petriho sítí. Tento mírně upravený formát vychází z formátu PNML pro P/T Petriho sítě, protože oficiální a standardizovaný formát pro OOPN neexistuje. Velmi vhodným rozšířením by bylo vytvořit a standardizovat jazyk PNML pro podporu OOPN, aby bylo možné vytvořené modely používat napříč různými aplikacemi pro tvorbu modelů Objektově orientovaných Petriho sítí.

Prvky pro zpřehlednění používání nástroje

Vhodným rozšířením by také mohlo být přidání dalších ovládacích prvků. V dnešní době jsou velmi oblíbené klávesové zkratky, které by učinily používání nástroje ještě více intuitivním.

Literatura

[1] Billington, J.; Christensen, S.; Van Hee, K.: The Petri Net Markup Language: Concepts, Technology, and Tools [online]. `http://www.pnml.org/papers/PNML_CTT.pdf`, Mar. 2003, 24th International Conference on Application and Theory of Petri Nets 2003, LNCS volume 2679, pages 483-505.

[2] Hillah, L. M.; Kindler, E.; Kordon, F.; aj.: A primer on the Petri Net Markup Language and ISO/IEC 15909-2 [online]. `http://www.pnml.org/papers/pnnl76.pdf`, October 2009, petri Net Newsletter 76:9–28.

[3] Janoušek, V.: *Modelování objektů Petriho sítěmi*. Dizertační práce, Fakulta informačních technologií Vysokého učení technického v Brně, 1998.

[4] Kindler, E.: PNML: Concept, Status and Future Directions [online]. `http://www.pnml.org/papers/PNML-EKA06.V3.pdf`, May 2006, entwurf Komplexer Automatisierungssysteme (EKA), volume 9, pages 35-55.

[5] Kočí, R.; Janoušek, V.; Zbořil, F.: Object Oriented Petri Nets - Modelling Techniques Case Study. *International Journal of Simulation Systems, Science & Technology*, ročník 10, č. 3, 2010: s. 32–44, ISSN 1473-8031.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=9194

Přílohy

Seznam příloh

Příloha A

Obsah CD

Kořenový adresář CD obsahuje:

- `lib/` - adresář obsahující knihovny
- `nbproject/` - adresář obsahující skripty k přeložení aplikace
- `resource/` - adresář s externími zdroji, ikonami a XSD schématem PNML pro OOPN
- `src/` - adresář se zdrojovými kódy
- `xml-resources/` - adresář s *JAXB binding*
- Soubory potřebné k přeložení aplikace
- `LICENCE`
- `README`
- `Poster.pdf` - plakát
- `PNtalk.zip` - archiv se simulačním serverem systémem PNtalk

Příloha B

Syntax jazyka PNtalk

Syntax byla převzata z [3]. Syntax není kompletní, pro potřeby nástroje byla použita jen tato podmnožina pravidel.

```
class: "class" classhead [objectnet] [methodnet|constructor|sync|inhib]*
classhead: id "is a" id

objectnet: "object" net
methodnet: "method" message net
constructor: "constructor" message net
sync: "sync" message [cond] [precond] [guard] [postcond]
inhib: "inhibitor" message [cond] [precond] [guard] [postcond]
message: id | binsel id | [keysel id]+
net: [place|transition]*

place: "place" expr

transition: "trans" id [cond] [precond] [guard] [action] [postcond]
cond: "cond" expr
precond: "precond" expr
postcond: "postcond" expr
guard: "guard" "{" expr "}"
action: "action" "{" expr "}"

id: letter[letter|dig]*

binsel: selchar[selchar]
selchar: "+" | "-" | "*" | "/" | "~" | "|" | "," | "<" | ">" | selchar2
selchar2: "=" | "&" | "n" | "@" | "%" | "?" | "!"

keysel: id ":"

expr: posloupnostLibovolnychRetezcu
```