

# 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 PODPORU VÝVOJE SOFTWAREVÝCH SYSTÉMŮ

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE

AUTHOR

KAREL HALA

BRNO 2014



**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 PODPORU VÝVOJE SOFTWAREVÝCH SYSTÉMŮ**

TOOL FOR SOFTWARE SYSTEMS DESIGN

**BAKALÁŘSKÁ PRÁCE**

BACHELOR'S THESIS

**AUTOR PRÁCE**

AUTHOR

**KAREL HALA**

**VEDOUCÍ PRÁCE**

SUPERVISOR

**Ing. RADEK KOČÍ, Ph.D.**

BRNO 2014

## Abstrakt

Nástroj pro podporu vývoje softwarových systémů je aplikace sloužící pro vizuální znázornění průběhu vývoje aplikace. Slouží k porozumění požadavkům od zákazníka, tyto požadavky převést do objektového návrhu s možností popsat třídní metody a třídy samotné pomocí Objektově orientovanými Petriho sítěmi. V této práci se postupně popíší konkurenční nástroje, vyberou se jednotlivé diagramy a popíše se způsob implementace těchto diagramů. V závěru budou poté představeny možné rozšíření.

## Abstract

Tool for software systems design is application for visualization of development application. It's main goal is to achieve connection between developer and customer. It should be used to understand customer's needs, prepare workflow of project and understanding of each class by describing these classes and their methods with Objected oriented Petri nets. In this work we will first focus on competition, we pick some diagrams and describe how they were implemented. At the end we will discuss possible extension for this tool.

## Klíčová slova

UML diagramy, Propojení UML diagramů, Příklad užití, Diagram tříd, Nástroj, Softwarový vývoj, Objektově orientované Petriho sítě

## Keywords

UML diagrams, Connection of UML diagrams, Use Case, Class diagram, Tool, Software development, Objected oriented Petri nets

## Citace

Karel Hala: Nástroj pro podporu vývoje softwarových systémů, bakalářská práce, Brno, FIT VUT v Brně, 2014

# Nástroj pro podporu vývoje softwarových systémů

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing., Radka Kočího Ph.D.

.....

Karel Hala  
6. května 2014

## Poděkování

Předem bych rád poděkoval panu doktoru Kočímu, za odbornou pomoc a vysvětlení propojení mezi jednotlivými částmi aplikace.

© Karel Hala, 2014.

*Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.*

# Obsah

<b>1</b>	<b>Úvod</b>	<b>3</b>
<b>2</b>	<b>Konkurenční nástroje</b>	<b>4</b>
2.1	Umbrello	4
2.1.1	Motivace	4
2.1.2	Popis aplikace	4
2.1.3	Výhody a nevýhody	5
2.1.4	Zhodnocení a převzaté vlastnosti	5
2.2	Enterprise Architect	6
2.2.1	Popis aplikace	6
2.2.2	Výhody a nevýhody	6
2.2.3	Zhodnocení a převzaté vlastnosti	7
2.3	Zhodnocení konkurenčních nástrojů	7
<b>3</b>	<b>Grafy pro popis částí</b>	<b>9</b>
3.1	Objektově orientované programování	9
3.1.1	části objektově orientovaného programování	9
3.1.2	Objekt	9
3.1.3	Instanciace, třídy	10
3.1.4	Dědičnost	10
3.2	Diagram případů užití	10
3.2.1	Rozbor Diagramu případů užití	10
3.2.2	Použité vlastnosti	11
3.3	Diagram tříd	11
3.3.1	Prvky Diagramu tříd	11
3.3.2	Použité a upravené části z Diagramu tříd	13
3.4	Objektově orientované Petriho sítě	14
3.4.1	Petriho sítě	14
3.4.2	Objektově orientované Petriho sítě	15
3.4.3	Grafické znázornění	15
3.5	Spojení diagramů	16
3.5.1	Použití ve výsledné aplikaci	16
3.6	Shrnutí	17
<b>4</b>	<b>Samostatná aplikace</b>	<b>19</b>
4.1	Popis aplikace	19
4.2	Ovládací prvky	19
4.3	Jednotlivé editory	20

4.3.1	Editor případů užití . . . . .	21
4.3.2	Editor Diagramu tříd . . . . .	21
4.3.3	Editor objektově orientovaných Petriho sítí . . . . .	22
4.3.4	Provázání jednotlivých částí . . . . .	23
<b>5</b>	<b>Rozšířené možnosti nástroje</b>	<b>24</b>
5.1	Export diagramů . . . . .	24
5.2	Grafické prvky . . . . .	24
5.2.1	Detekce objektu pod kurzorem . . . . .	25
5.2.2	Výpočet vzdálenosti hrany od bodu . . . . .	25
<b>6</b>	<b>Použité knihovny a ikony</b>	<b>27</b>
6.1	XML export . . . . .	27
6.2	PostScript export . . . . .	27
6.3	Použité ikony . . . . .	28
<b>7</b>	<b>Testy a jejich výsledky</b>	<b>29</b>
7.1	Nezkušený uživatel . . . . .	29
7.1.1	Průběh představení . . . . .	29
7.2	Zkušenější uživatel . . . . .	29
7.2.1	Průběh a výsledky uživatelské práce . . . . .	29
7.3	Programátor . . . . .	30
7.3.1	Průběh testování . . . . .	30
7.3.2	Vyhodnocení rogramátorovy práce . . . . .	30
7.4	Výsledky testování . . . . .	30
<b>8</b>	<b>Závěr</b>	<b>31</b>
8.1	Budoucnost aplikace . . . . .	31
8.1.1	Navrhovaná rozšíření . . . . .	31
8.1.2	Shrnutí rozšíření . . . . .	32
8.2	Závěrečné zhodnocení aplikace . . . . .	32
<b>A</b>	<b>Obsah CD</b>	<b>34</b>

# Kapitola 1

## Úvod

Při implementaci jakéholiv nástroje se často naráží na bariéru mezi zákazníkem a programátorem. Zákazník často netuší co chce a programátor často zákazníka špatně pochopí. Tento nástroj bohužel tyto problémy nedokáže odstranit, nicméně měl by napomoci odstranit některé bariéry a pomoci snažší vizualizace problémů.

V kapitole 2 se nejdříve podíváme na konkurentní nástroje, které dokáží vizualizovat vyvíjený software pomocí různých grafů. Rozebereme jejich přednosti a jejich slabiny. Poté si vybereme některé jejich vlastnosti a vysvětlíme případné řešení jejich nedostatků.

Navážeme kapitolou 3 kde vysvětlíme druhy grafů, které se hodí pro implementaci námi zvolených částí. Zaměříme se na trojici diagramů **Diagram případů užití**, **Diagram tříd** a **Objektově orientované Petriho sítě**. Provázání mezi těmito diagramy poté vysvětlíme v kapitole 3.5.

V následující kapitole představíme nástroj, který byl předmětem této práce. Předvedeme jeho silné stránky, jakým způsobem byly vytvořeny určité části aplikace a také mírně nastíníme ovládací prvky tohoto nástroje 4. Na toto volně navážeme představním rozšířených možností tohoto nástroje v kapitole 5.

Po vysvětlení nástroje představíme jeho použitelnost v praxi, představením několika možným budoucím uživatelům a jejich reakcí na tento nástroj. Popíšeme jejich chování a jejich pocity z aplikace 7.

Závěrem práce poté bude představení možných dalších rozšíření tohoto nástroje. Také rozebereme momentální stav této aplikace, kde se zaměříme na přínos nástroje a popřípadě popíšeme možné chyby a nedostatky aplikace 8.

## Kapitola 2

# Konkurenční nástroje

Z hlediska snažšího porozumění byl brán zřetel na konkurenční nástroje. Převážně byla brána v potaz jejich jednoduchost a co je dělá tak hojně využívanými a snadno použitelnými. Z toho důvodu byly prozkoumány dva nástroje, **Umbrello** a komerční **Enterprise Architect**. Detailnější popis jednotlivých aplikací, jejich výhod, nedostatků a převzatých vlastností je popsán v sekci 2.1 a 2.2.

### 2.1 Umbrello

Aplikace celým názvem **Umbrello UML Modeller**<sup>1</sup> aplikace primárně určena pro Unix systémy, nicméně je možné jej bez problémů nainstalovat také na Windows za pomoci KDE installeru a vybrání požadovaného balíčku.<sup>2</sup>

Mezi hlavní výhody Umbrello aplikace patří refaktoring pro některé jazyky. Nicméně graficky zaostává za konkurencí ostatních nástrojů pro tvorbu UML diagramů.

#### 2.1.1 Motivace

Aplikace Umbrello je hojně využívána a chválena mezi uživateli, z toho důvodu byl brán zřetel na tuto aplikaci. Také jednoduché používání této aplikace bylo velice přínosné pro pozdější pochopení potřebných vlastností aplikace zaměřené na dané téma.

#### 2.1.2 Popis aplikace

Aplikace je velice robustní a poskytuje mnoho vlastností. Při ohlédnutí na to, že je volně ke stažení a volně k používání<sup>3</sup> aplikace nabízí velké množství vlastností a možností. Umbrello poskytuje mnoho UML grafů a základní práce s touto aplikací je poměrně jednoduchá, ale její jednoduchý design má pár slabin. Pro menší aplikace a pro začínající firmy je naprosto dokonalá, ale absence pokročilého editoru a ne příliš jednoduché práce s editorem jako například přidání nové metody, zalomení čar a také vytvoření jiného souboru.

<sup>1</sup>Domovská stránka aplikace: <http://umbrello.kde.org/>

<sup>2</sup>Ovládání tohoto installeru, je velice jednoduché a intuitivní. Lze jej nalést na <http://windows.kde.org/download.php>.

<sup>3</sup>Aplikace používá licenci GNU general public license <http://www.gnu.org/licenses/old-licenses/gpl-2.0.html> což umožňuje zapojení komunity do vývoje tohoto nástroje.



### 2.1.3 Výhody a nevýhody

Umbrello aplikace poskytuje několik UML grafů mezi které patří:

- **ClassDiagram** pro popis tříd a provázanosti mezi nimi.
- **Sequence Diagram**, pro popis chodu aplikace, komunikace mezi vlákny a komunikace mezi aplikacemi.
- **Use Case Diagram** pro popis akcí v systému, aplikaci.
- **State Diagram** pro popis stavu systému a průběhu jednotlivých operací, převážně zachování v určitých momentech aplikace na dané podmínky.
- **Activity Diagram** popisuje akce a průběh celého programu graficky.
- **Component Diagram** pro popis spojení a komunikace mezi jednotlivými komponentami.
- **Deployment Diagram** popisuje fyzické uložení a zařízení, na kterých program bude pracovat.
- **Entity Relationship Diagram** pro popis vztahu dat mezi sebou.

Umbrello poskytuje velké množství diagramů a možností, jak znázornit chod aplikace. Aplikace je také velice intuitivní, ať již díky klávesovým zkratkám, tak také díky jednoduchému grafickému designu. Umbrello, také podporuje generaci a refactoring zdrojového kódu pro některé jazyky. Stačí pouze importovat jednu již vytvořenou třídu a Umbrello uživatelé provede jednoduchým nastavením, výběrem které třídy namodelované v aplikaci se mají vygenerovat a nabídne také možnost mírně upravit zvolený kód. Vzhledem na to, že aplikace je plně zdarma je tato vlastnost velice přínosná.

Umbrello dále poskytuje možnost částečné generace dokumentace i v případě, že nejsou poskytnuty komentáře na základě kterých by tato dokumentace mohla být vytvořena. Při generování tříd je možné, také importovat zdrojový kód a nechat umbrello vše vygenerovat samo. Tato vlastnost je, ale podporována pouze pro jazyky **ActionScript**, **Ada**, **C++**, **C#**, **D IDL**, **Java**, **Javascript**, **MySQL**.

Dále Umbrello poskytuje možnost exportování objektů jako PNG obrázky. Při psaní dokumentů je možné vytvořený diagram zkopírovat a vložit přímo do dokumentu. Pro export do formátu PNG je možné vybrat část diagramu, nebo celou síť. s touto možností také přichází možnost tisku celého diagramu. [2]

Umbrello je sice jednoduché a intuitivní na používání, nicméně dlouhodobější práce je zdlouhavá a příliš repetitivní. Nutnost vytvoření objektu na editačním plátně, poté výběru nástroje pro přesun a editaci daného objektu je zdlouhavá a špatná. Jednodušší pro práci by jistě bylo vytvoření objektu a po kliknutí na již vytvořený objekt jej pouze přesunout. Vytvoření nového atributu třídy je opět velice náročné a neintuitivní. Taktéž nemožnost zalomení spojů mezi objekty je velice limitující.

### 2.1.4 Zhodnocení a převzaté vlastnosti

Aplikace Umbrello je špičkou ve své třídě. Je zdarma a poskytuje mnoho nástrojů při návrhu aplikace. Mezi jeho přednosti jistě patří možnost refactoringu a také rozsáhlé možnosti ve variabilitě grafických návrhů.

Aplikace poskytla mnoho informací o vývoji editační aplikace, jakým chybám se vyvarovat a co je naopak nezbytné k vytvoření kvalitní aplikace zaměřené na tvorbu diagramů. Není zapotřebí ve výsledné aplikaci mnoho náročných grafických elementů, tyto elementy by byly rušivé. Grafické roložení a dosažitelnost většiny editačních prvků je v aplikaci Umbrello velice jednoduché a intuitivní z tohoto důvodu bylo ve výsledné aplikaci přihlíženo na tyto vlastnosti.

Ve výsledné aplikaci bylo vyvarováno některým nedostatkům aplikace Umbrello. Jako například zdlouhavé a repetitivní vytváření tříd, jejich přesun po pracovní ploše, nebo spojení těchto tříd.

V Umbrello lze spoje mezi objekty ohýbat velice složitě a z uživatelského hlediska velice náročně. Celková práce se spoji mezi objekty je v Umbrello řešena poměrně složitě, což často vede k vytváření nepřehledných a špatně pochopitelných diagramů.

Aplikace Umbrello při přidání nového objektu donutí uživatele zadat jméno, což opět vede ke zdlouhavé práci a repetitivním akcím, které uživatele zbytečně zdržují od práce. Toto bylo vyhodnoceno jako nepřínosné a nutnost zadat jméno objektu při jeho vytvoření bylo nahrazeno pozdější úpravou. Kdy si uživatel vytvoří objekt a až poté určí jeho jméno.

## 2.2 Enterprise Architect

S ohlédnutím na Aplikaci Umbrello a některé její nedostatky, bylo nutné pro pozdější vývoj aplikace prozkoumat další aplikaci. Tato aplikace se jmenuje **Enterprise Architect** a je tvořena firmou **Sparx Systems**<sup>4</sup>, tato firma se soustřeďuje na tvorbu nástrojů pro tvorbu UML grafů. Firmou nabízené produkty jsou placené, ale uživatel obdrží opravdu robustní systémy pro tvorbu UML grafů a možnost refaktoringu pro několik jazyků.

### 2.2.1 Popis aplikace

Enterprise Architect (EA) je velice robustní, již nemá takové nedostatky jako aplikace Umbrello a nabízí několikero možností pro refaktoring projektu. Aplikace rozděluje UML grafy do dvou skupin

- **Structural Diagrams** pro diagramy na popis vyvíjené aplikace a popis její struktury.
- **Behavior Diagrams** pro popis chování aplikace a komunikace mezi jednotlivými částmi aplikace.

Dále aplikace poskytuje velkou škálu grafů a diagramů od klasických diagramů pro popis chování dat mezi sebou (ERD), přes diagramy pro návrh Win32 UI až po návrh testování výsledné aplikace. [1]

### 2.2.2 Výhody a nevýhody

V aplikaci je možné vytvořit velké množství grafů a diagramů. Grafické rozhraní již není tak jednoduché, jako tomu je u aplikace Umbrello, ale je stále velice intuitivní a jednoduché pro používání. Většina pokročilých funkcí je však schována a je náročné ji najít, jako například přidání a editace atributů tříd.

---

<sup>4</sup><http://www.sparxsystems.com>

Výhodou této aplikace je jistě možnost vytváření diagramů pro jednotlivé jazyky, a poté tyto diagramy exportovat do zdrojového textu. Na výběr je celkem jedenáct jazyků ActionScript, C, C#, C++, Delphi, Java, PHP, Python, VBNet, Visual Basic, WorkFlow Script.

Největší výhodou je jistě možnost importovat celý obsah složky se zdrojovými kódy a tento projekt poté zobrazit v diagramu tříd. Lze také importovat binární soubor, například .jar souboru a aplikace Enterprise Architect může vygenerovat dokumentaci a diagram tříd.

Pro jednodušší a rychlejší možnost spojení jednotlivých objektů na pracovní ploše, lze použít rychlého nástroje vedle objektu. Na stejném místě lze nalézt také funkce pro rychlou editaci a změnu stylu objektu.

Přetažení spoje mezi dvěma objekty lze pomocí uchopení za jeden konec a tažení k dalšímu objektu. Bohužel původní spoj se neztratí do doby, než uživatel pustí myš.<sup>[1]</sup>

### 2.2.3 Zhodnocení a převzaté vlastnosti

Aplikace posloužila pro další zhodnocení požadavků a vyvarování se chyb. Především náročnost a komplexnost aplikace se nehodí pro výslednou aplikaci.

Ve výsledné aplikaci bude zapotřebí možnosti jednoduchého zalomení hran a možnosti jejich jednoduché editace. Taktéž přesun po editačním plátně je v aplikaci EA řešen poměrně chaoticky a nahodile.

Umístnění otevřených souborů je v aplikaci EA poněkud netradiční v dolní části aplikace. Uživatel často nevidí otevřený projekt a často se může stát, že zavře stávající jen kvůli tomu aby našel již otevřený soubor. Při porovnání s aplikací Umbrello, která toto řeší více standartním způsobem, umístnění záložek v horní části okna, je tento způsob poněkud nepraktický.

V Enterprise Architect jsou druhy diagramů a grafů umístněny ve stromové struktuře, z pochopitelných důvodů rozsáhlosti aplikace. Opět nepoužitelné v menších aplikacích a zbytečně náročné na hledání a přidání nového diagramu, nebo grafu.

## 2.3 Zhodnocení konkurentních nástrojů

Při testování nástrojů určených pro tvorbu UML grafů bylo zjištěno mnoho detailů a důležitých vlastností, které bude výsledná aplikace potřebovat. Některé výhody těchto aplikací, jako například generování kódu, nebude potřeba implementovat vzhledem k tomu, že aplikace se soustředí čistě jenom na vytváření a editaci diagramů a UML grafů.

V testovaných aplikacích nebyla nikde možnost částečné předgenerace dalších částí návrhu pomocí předešlé. Například provázanost mezi diagramy případů užití a diagramem tříd nebyla možná. V tomto ohledu bylo rozhodnuto, že je potřeba tyto části propojit a více zautomatizovat pro rychlejší a svižnější práci s výslednou aplikací.

Grafický prvek celé aplikace bude zapotřebí jednoduchý a snadný k používání, ale důležitým prvkem je dobré zviditelnění některých objektů. Jako například znázornění tříd, které nemají odpovídající prvky v případě užití. Nebo znázornit prvky, které nemají žádné spojení s ostratními objekty. Takovéto znázornění nebylo nalezeno v žádné z testovaných aplikací, ale je důležitým prvkem v rozpoznávání editovaného diagramu.

V testovaných aplikacích chyběla možnost, nebo byla nedostatečná, pro zalomení hran. Tato vlastnost velice napomůže pro zpřehlednění výsledného diagramu, podobně tomu bylo

s editací hran. Změna typu hrany a změna objektů pro danou hranu je zapotřebí pro jednodušší práci s editorem a především pro rychlejší práci.

## Kapitola 3

# Grafy pro popis částí

Pro grafické znázornění jednotlivých částí byly vybrány z UML grafů **diagramy případů užití** 3.2 pro popis jednotlivých akcí v systému a ?? 3.3 pro grafické znázornění tříd a operace mezi nimi. Jako poslední část, která nespádá pod UML, bylo vybráno Objektově orientovaných Petriho sítí 3.4 pro popis chování jednotlivých tříd.

### 3.1 Objektově orientované programování

Předtím, než rozebereme grafy, které slouží pro popis vývoje a vytvoření návrhu je potřeba rozebrat objektově orientované programování. Toto je speciální styl programování používaný převážně při vývoji větších a náročnějších aplikací.

#### 3.1.1 části objektově orientovaného programování

Objektově orientované programování se skládá z několika částí a několika principů, jak programovat.

#### 3.1.2 Objekt

Základ takového programování je samozřejmě objekt, je to specifický pohled na fyzikální a konceptální jednotky reálného světa. Jednotlivé části objektu jsou:

- Stav nese informaci o tom v jakém stavu se objekt v daném čase nachází.
- Chování popisuje reakce na vnější a vnitřní události.
- Identita každý objekt je jednoznačně identifikovatelný v rámci systému – má jméno, kterým se liší od každého jiného.

Jako jeden z hlavních rysů objektově orientovaného programování je zapouzdření, skrytí veškerých dat a práci s daty před ostatními objekty. Ostatní objekty jsou schopné komunikovat mezi sebou pomocí zasílání zpráv. Tyto zprávy obsahují jméno operace, parametry – data potřebná pro zpracování a jméno objektu obsahujícího tuto operaci. Zasláná zpráva může sloužit jak ke zpracování a navrácení dat, tak k vnitřní změně objektu, který tuto zprávu přijal.

Veřejné rozhraní objektu je určeno množinou operací, což je výčet schopností objektu. Popis provádění operace je schován uvnitř objektu a je popsán metodami. Uvnitř objektu se také nachází atributy, cože je popis uchování dat. [4]

### 3.1.3 Instanciace, třídy

Ve většině programovacích jazyků se vytváření objektů provádí speciální konstrukcí, takzvanou třídou. Třída je předpis pro objekt.

Třída definuje viditelnost instančních proměnných (atributů) a metod. Dále definuje metody a jejich provádění.

Objekty mohou být instanciovány staticky a dynamicky. Staticky instanciované třídy se vytvoří při spuštění programu a zanikají až při jeho ukončení. Dynamicky instanciované proměnné vznikají a zanikají v průběhu chodu aplikace na požadavky uživatele. [4]

### 3.1.4 Dědičnost

Další ze silných vlastností objektově orientovaného programování je dědění. Slouží ke sdílení jisté charakteristiky jednoho objektu s druhým.

Každý jazyk má tuto vlastnost implementovanou různě. Dynamické a statické dědění může být zaměřeno na třídy nebo objekty. Může být děděna reprezentace, nebo chování. Dědění může být implementováno částečně, nebo úplně. A v neposlední řadě lze rozlišit dědění násobné a jednoduché podle počtu dědění. [4]

## 3.2 Diagram případů užití

Diagramy případů užití jsou často modelovány a používány ke komunikaci mezi zákazníkem, který nemá příliš mnoho zkušeností s vývojem systému, ale rozumí danému problému a svému oboru a programátorem, člověkem který naopak má zkušenosti s vývojem aplikace, ale nechápe do hloubky zákazníkův problém a jeho obor.

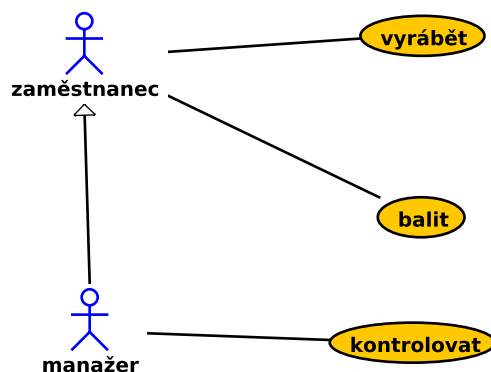
Základními prvky diagramu případů užití jsou případy užití a aktéři. V požadavcích na systém rozpoznáváme takzvané „Stakeholdery“ jsou to osoby, nebo věci, které mají zájem na úspěšné provedení systému. Případy užití popisují jak se má systém zachovat na reakce stakeholderů, akterů, a dodat určité výsledky. Aktéři na druhou stranu, jsou přímí uživatelé nebo věci s určitým chováním. Hlavní aktér je často stakeholderem daného systému a hlavní spouštěč tohoto systému. Občas je zapotřebí aktér, který není přímo v systému, tomuto se říká „Podpurný aktér“. Aktér není přímo chápán jako jedinec, nýbrž jako role v systému. [7]

### 3.2.1 Rozbor Diagramu případů užití

Tyto diagramy se používají pro grafický popis případů užití a zahrnuje několik pravidel. Aktéři jsou znázorňováni ve tvaru postav a případy užití jako elipsy obsahující název prováděného úkonu. Spoje mezi případy užití a aktéry jsou znázorněny úsečkou vedoucí od aktéra k případu užití.

Mezi případy užití mohou být vedeny dále spoje pro jejich rozšíření a upravení. Jsou to **include**, **extends** a **generalizace**. Poslední jmenovaný spoj je možné také vést mezi aktéry.

**Include** znamená to, že pro správný chod daného nadřazeného případu užití je zapotřebí použití nižšího případu užití. Pro vykonání vyššího případu užití je zapotřebí nejdříve vykonat nižší případ užití. Pokud při vykonávání několika případů užití je vykonávána stejná funkcionalita, je vhodné tuto funkcionalitu oddělit z daného případu užití a spojit se všemi případy užití spojem **include**. Spoj **include** je znázorněn přerušovanou šipkou doprovázené textem „include“. Spoj je veden z nadřazeného případu užití k nižšímu případu.



Obrázek 3.1: Příklad Diagramu případů užití.

**Extends** slouží pro rozšíření určitých případů užití. Toto znamená, že je rozšířením hlavního případu užití a toto rozšíření bylo vyjmuta z původního případu. V překladu tento spoj znamená, že jeden případ užití je rozšířen dalším případem užití. Spoj je znázorňován přerušovanou šipkou vedenou do rozšiřovaného (hlavního) případu užití.

**Generalizaci** lze použít jak v případech užití, tak mezi aktéry. Slouží ke znázornění rodičů a potomků. Při použití tohoto spoje znamená, že potomek přebírá veškerou funkcionalitu svého rodiče a může tuto funkcionalitu poupravit, nebo přidat vlastní. Tento spoj je znázorněn spojem ukončeným prázdnou šipkou a je veden od potomka k rodiči. [7]

### 3.2.2 Použité vlastnosti

Ve výsledné aplikaci byl použit Diagram případů užití pro jednoduché znázornění a zanesení zákaznických požadavků. Tento diagram byl rozšířen o několik užitečných vlastností.

Pokud třída není nijak spojena žádným spojem s dalšími třídami, je tato třída bez jakýchkoliv spojů znázorněna přerušovanou čarou. Toto vede ke snadší identifikaci těchto tříd.

Vybraný objekt je znázorněn odlišnou barvou a jeho okolí je ohraničeno rámečkem. Pokud je objekt spojen s některým jiným objektem tyto spoje jsou znázorněny po vybrání tohoto objektu. Zvýrazněné spoje jsou pouze ty spoje, které vedou z vybraného objektu. Toto vede ke zprehlednění celého diagramu případů užití.

## 3.3 Diagram tříd

Po znázornění aplikace pomocí Diagramu případů užití se nabízí další nástroj z UML rodiny a tím je **Diagram tříd**. Tento diagram slouží pro grafické znázornění objektů a koinakci mezi nimi, tyto objekty a zprávy jsou popsány v kapitole 3.1.

### 3.3.1 Prvky Diagramu tříd

Hlavní složkou diagramu jsou třídy, ty jsou znázorněny pomocí obdélníků obsahující další informace o dané třídě. V horní části je název třídy. Poté třída obsahuje instancí proměnné,

atributy. A poslední část tříd jsou operace, metody.

Určitou podmnožinou třídy jsou abstraktní třídy, tyto třídy nemohou být inicializovány, pouze slouží k předpisu pro potomky. To znamená, že třída A je abstraktní a je předkem třídy B, v systému se nenachází objekt třídy A, ale pouze objekt třídy B.

V diagramu tříd lze také namodelovat interface. Toto je speciální třída obsahující předpis pro atributy a operace, ale nenachází se v takových třídách implementace těchto operací. Tato implementace se nachází až ve třídách, které implementují takový interface.

Zápis atributů a operací tříd je stanoven tak, že za jménem následuje dvojtečka a její typ. V případě operací za jménem následuje závorka obsahující seznam argumentů a až poté dvojtečka a návratový typ. Pokud operace nic nevrací je možné indikovat toto pomocí nastavení návratové hodnoty na „void“ Dále je možné omezit vyditelnost atributů a operací pomocí identifikátoru. **Private** označované „-“, **protected** jako „#“ a **public** znázorněné pomocí „+“. Tyto identifikátory se píší před jméno atributu, nebo operace.

Jednotlivé hrany pro spojení se dělí na **Generalizaci**, **Asociaci**, **Agregaci**, **Compozici** a **Hrana Implementace**. [6]

### Generalizační hrana

Hrana **Generalizace** popisuje spojení mezi třídou, která je více generická a třídou, která je specifická. Generická třída se nazývá **superclass**, nebo také **rodič** a více specifická třída se nazývá **subclass**, známá též jako **potomek**.

V UML je tato hrana popsána šipkou ukončenou bílým trojúhelníkem na konci. Tato hrana je vedena směrem od generické třídy k více specifické.

Je také možné několikanásobná dědičnost, ale v případě implementace takovéto dědičnosti se musí vzít v potaz to, zda daný jazyk několikanásobnou dědičnost podporuje. Protože při takovéto dědičnosti často může nastat problém. Především díky tomu, že pokud třída A dědí od tříd B a C a třída B i třída C obsahují stejnou metodu, programátor musí řešit, která metoda se má v třídě A použít. Například v jazyce Java<sup>TM</sup> několikanásobná dědičnost není povolena, ale je možné použít **interface**. [6]

### Asociační hrana

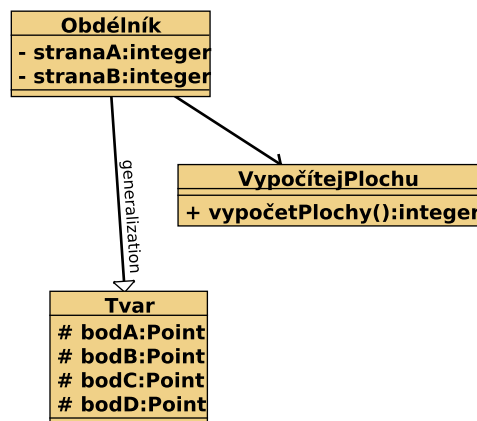
Vztah mezi třídami určující, že objekty jedné třídy jsou používány v objektech druhé třídy.

Asociace je kreslena v UML pomocí hrany s jednou šipkou na konci. A jmenuje se **jednosměrná**, tento styl hrany je používanější. Takto použitá asociace znamená že třída, ze které šipka vede, je použita ve třídě do které tato šipka směřuje. Tuto hrana lze kreslit také se dvěma šipkami na konci, jmenuje se **obousměrná** a znamená, že obě třídy se používají navzájem. [6]

### Agregační hrana

Použití spoje Agregace značí, že určitý objekt potřebuje, ke svému chodu další objekt. Pokud ovšem přestane existovat původní objekt, další objekt lze nadále v systému použít. To znamená, že objekt třídy A potřebuje ke svému chodu objekt třídy B. V momentě odstranění objektu třídy A ze systému, ale objekt třídy B nezaniká. Tomuto se říká slabá agregace. Tato hrana je kreslena souvislou hranou opatřenou bílým diamantem na konci. [6]





Obrázek 3.2: Příklad Diagramu tříd.

### Kompoziční hrana

Tato hrana je z části stejná, jako hrana Agregáční, s tím rozdílem že reprezentuje silnou agregaci. To znamená že pokud objekt třídy A potřebuje ke svému chodu objekt třídy B v momentě odstranění objektu třídy A je ze systému odstraněn objekt třídy B. Tato hrana je kreslena souvislou hranou opatřenou černým diamantem na konci. [6]

### Implementační hrana

Tato hrana je specifická v tom, že se kreslí pouze mezi třídou a interfacem. Slouží ke znázornění, že třída A implementuje interface B, převezme všechny atributy třídy B a přetíží operace třídy B, které implementuje. Hrana je kreslena přerušovanou šipkou ve směru od implementované třídy k interfacu. [6]

### 3.3.2 Použité a upravené části z Diagramu tříd

Ve výsledné aplikaci bylo použito pro popis vztahů mezi jednotlivými třídami diagramu tříd. Tento diagram byl z části upraven pro jednodušší práci a snadnější používání uživatelem.

Kvůli provázanosti s Diagramem případů užití musely být třídy rozděleny do čtyř druhů:

- **Actor** odpovídá bjecktu aktér v Diagramu případů užití.
- **Activity** tato třída je spojena s případem užití.
- **None** pro popis třídy, která se nepromítne v Diagramu případů užití.
- **Interface** speciální druh třídy.

Typ třídy se nachází nad danou třídou, v případě interface a None třídy jsou tyto názvy skryty.

Podobně jako tomu je v případě Diagramu případů užití i zde je názorně vidět, zda některá hrana vede z a nebo do třídy a to tak, že třída bez jakýchkoliv hran je ohraničena čárkovaným orámováním.

Zvýrazněná třída je oddělena od ostatních odlišnou barvou orámování a hrany, které vedou z, nebo do dané třídy jsou znázorněny jinou barvou, než ostatní hrany.

Třídy, které nemají odpovídající předpis v části Diagramu případů užití, jsou znázorněny tak, že jejich obsah je psán nahnutým písmem a pod třídou, se nachází text „No UseCase“. Pokud hrana spojující třídy nemá odpovídající spoj v Diagramu případů užití, je odlišena rozdílnou barvou.

## 3.4 Objektově orientované Petriho sítě

Pro popis chování objektů a komunikace mezi nimi ve výsledné aplikaci bylo zvoleno použití objektově orientovaných Petriho sítí.

### 3.4.1 Petriho sítě

Objektově orientované Petriho sítě jsou rozšířením Petriho sítí. Základní Petriho sítě jsou vhodné pro modelování a teoretické zkoumání paralelních systémů.

Chování dynamického systému je v podstatě sledování stavové proměnné a zaznamenávání její hodnoty v čase.

Petriho síť lze matematicky zapsat jako  $N = (P_N, T_N, PI_N, TI_N)$  kde:

- $P_N$  je konečná množina **míst**.
- $T_N$  je konečná množina **přechodů**,  $P_N \cap T_N = \emptyset$ .
- $PI_N : P_N \rightarrow \mathbb{N}$  **inicializační funkce**.
- $TI_N$  je **popis přechodů**. Je to funkce definovaná na  $T_N$ , taková, že  $\forall t \in T_N: TI_N = (PRECOND_t^N, POSTCOND_t^N)$ , kde
  1.  $RECOND_t^N : P_N \rightarrow \mathbb{N}$  jsou **vstupní podmínky** přechodu.
  2.  $POSTCOND_t^N : P_N \rightarrow \mathbb{N}$  jsou **výstupní podmínky** přechodu  $t$ .

Petriho sítě se skládají z míst a přechodů, tyto prvky jsou spojeny orientovanými hranami. Nemohou být spojeny dvě místa mezi sebou a stejně, tak nemohou být spojeny dva přechody přímým spojem.

Procesy jsou v Petriho sítích znázorněny tečkou uvnitř míst, v průběhu provádění simulace se tento bod přesouvá mezi jednotlivými místy. Tyto přesuny jsou provedeny na základě splnění pravidel zanesená v přechodech. Tyto pravidla jsou atomická, což znamená že se buď toto pravidlo provede, nebo ne. Jednoduché rozšíření je přidání kapacity pro jednotlivá místa.

Přechody mohou být okamžité, při splnění zanesených pravidel se okamžitě provedou. Nebo mohou být časované, což znamená že po určitém čase se daný přechod provede. Toto je opět velice jednoduché rozšíření základních Petriho sítí.

Úspěšné provádění přechodů probíhá v pořadí odebrání značek ze vstupního místa, přidání značek na výstupní místo.

Dále můžeme síť rozšířit o jednoduché oprioritizování spojení mezi místy a přechody, nebo vložením procentuální pravděpodobnosti provedení daného spoje. [4]

## Rozšíření Petriho sítí

Základní Petriho sítě představují příliš nízkoúrovňový model pro modelování reálných systémů. Z toho důvodu bylo zavedeno několik rozšíření. Pro pozdější použití nás nejvíce zajímá rozšíření petriho sítí na funkcionální. Což je rozdělení přechodu na stráž a akci a do místa akce možnost volat jednotlivé části sítí.

Toto lze poté rozšířit o objektové rozdělení, s tím že v místě akce lze volat vytvoření nového objektu jiné sítě a poté zaslání zprávy tomuto objektu.

### 3.4.2 Objektově orientované Petriho sítě

Model systému, ve kterém mohou dynamicky vznikat a zanikat objekty, komunikující předáváním zpráv lze vhodně znázornit pomocí Objektově orientovaných Petriho sítí, jako model je množina tříd, hierarchicky organizována podle dědičnosti. Každá z těchto tříd obsahuje tyto prvky:

- síť objektu, definuje reprezentaci instance třídy (atributy)
- množinu sítí metod, tyto sítě modelují reakce na zaslání zprávy
- množinu synchronních portů, definující reakce na synchronní zprávy

Sítě metod mohou sdílet atributy objektu. Mají navíc parametrová místa, do kterých se při invokaci takové sítě vloží parametry ze zprávy, dále mají výstupní místo **return**, což je místo do kterého se uloží výstupní hodnota.

Značky v sítích nereprezentují samotné objekty, ale pouze reference na tyto objekty. V objektově orientovaných Petriho sítích jsou tři druhy objektů. Primitivní třídy to jsou čísla znaky, řetězce, symboly a seznamy. Další objekty jsou třídy, tyto objekty se v průběhu nemění a rozumějí na volání **new**, což vede k vytvoření nové instance. Ostatní objekty jsou instancemi tříd.

Proveditelnost přechodu v objektově orientovaných Petriho sítích je rozdělena na stráž přechodu a akci. Stráž přechodu obsahuje sekvenci výrazů a každý z nich je zaslání zprávy. Pokud každá další zpráva vrací **true**, stráž je vyhodnocena jako **true**.

Akce přechodu specifikuje zaslání zprávy. Budeme uvažovat tvar zaslání ve tvaru (3.1):

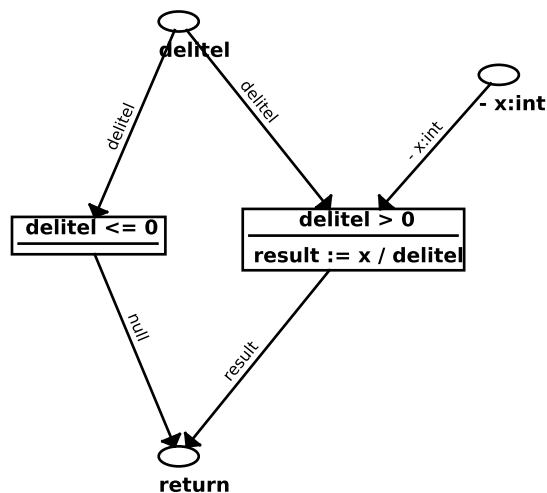
$$y := x_0.msg(x_1, \dots, x_n) \quad (3.1)$$

kde  $y$  je proměnná,  $x_i$  je term a  $msg$  je jméno zprávy. Objekt se jménem  $x_0$  je adresátem zaslání zprávy a  $x_i$  jsou argumenty této zprávy. V těchto akcích je zapotřebí definovat také speciální zprávu *new*, která vede k vytvoření nové instance třídy  $x_0$ .

Výsledná objektově orientovaná petriho síť má všechny atributy objektově orientovaného jazyka. Umožňuje zapouzdření, polymorfismus a dědičnost. [4]

### 3.4.3 Grafické znázornění

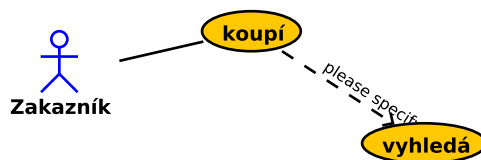
Jednotlivé místa se znázorňují v Objektově orientovaných Petriho sítích jako kolečka obsahující procesy, reference na objekty. Přechody jsou znázorněny obdélníkem obsahujícím stráž a akci. Akce se nachází pod čarou a nad čarou se nachází stráž.



Obrázek 3.3: Příklad popisu metody pro dělení pomocí Objektově orientovaných Petriho sítí.

### 3.5 Spojení diagramů

Během vytváření případů užití je vhodné vytvořit také diagram tříd. Tato akce spočívá v hledání návrhových tříd rozhraní a komponent, jež vzájemně komunikují a projevují chování specifikované pomocí případu užití.



Obrázek 3.4: Nástin problému případů užití.

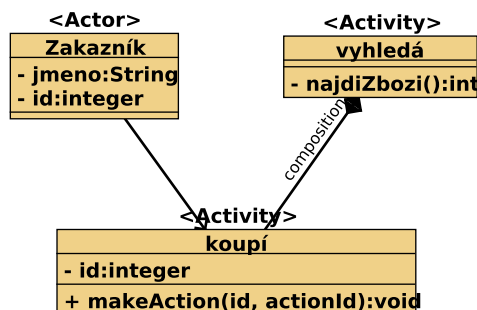
Během tohoto procesu vznikají analytické třídy, tyto třídy by měly mapovat pojmy reálného světa. Nicméně odhad a automatické generování těchto tříd je velice náročné a proto často tato práce zůstává na objektovém analytikovi. Ten by se měl postarat o další převod a upřesnění těchto analytických tříd.

Analytické třídy by měli obsahovat pouze ty nejdůležitější atributy a operace, které musí třída poskytovat. Často se může stát že operace je poté rozbita do několika menších metod. [5]

#### 3.5.1 Použití ve výsledné aplikaci

Provázání mezi UML částmi probíhá na velice nízké úrovni a to tak, že při vytvoření objektu v diagramu případů užití vznikne třída se stejným názvem v diagramu tříd. Na objektovém analytikovi potom je potřeba zadat jednotlivé atributy a operace. Pokud vznikne případ,

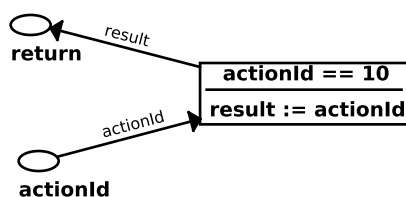
že se analytická třída rozpadne na více tříd, je toto možné realizovat dvojím způsobem. A to tak, že se odstraní analytická třída a vzniknou dvě nové třídy, které se nepromítnou do diagramu případů užití. A nebo zachováním analytické třídy a navázáním nové třídy, která se nepromítne do případů užití.



Obrázek 3.5: Nástin problému převodu případu užití do diagramu tříd.

Při vytváření tříd a operací (metod) v diagramu tříd také vznikají nové objektově orientované třídy. Každá metoda má jednu objektově orientovanou Petriho síť a podobně tomu tak je v případě třídy. Objektově orientovaná Petriho síť pro metodu má jedno specifické místo **return**, které nelze odstranit ani změnit její jméno. [4]

Objektově orientované Petriho síť, mají několik speciálních míst. V případě Petriho síť pro třídu jsou to třídní proměnné této třídy a zděděné třídní proměnné od svých předků. V případě Petriho síť je množina těchto zvláštních míst rozšířena o argumenty metody, které jsou vstupními místy pro tuto Petriho síť. [8]



Obrázek 3.6: Metoda popsána Objektově orientovanou Petriho sítí.

## 3.6 Shrnutí

V této kapitole byly přiblíženy nástroje sloužící ke znázornění postupného vývoje aplikace. Ke zjednodušení vývoje a převážně k zřehlednění komunikace uvnitř aplikace.

Motivací pro výběr těchto nástrojů byla skutečnost, že se v moderním programování používá převážně objektově orientovaných postupů. Dále také možnost jednoduché provázanosti mezi těmito nástroji, což vede ke zjednodušení a urychlení vývoje. S ohledem na moderní způsob agilního vývoje jsou tyto nástroje velice vhodné. A to převážně z hlediska jednoduché integrace zákazníka do vývoje a promítnutí změn od zákazníka okamžitě na celý systém.

## Kapitola 4

# Samostatná aplikace

Po vyzkoušení konkurenčních nástrojů a vybrání vhodných nástrojů z nich bylo zjištěno že žádný nástroj nedosahuje absolutních požadavků a žádný z nich neimplementuje Objektově orientované Petriho sítě. Z toho důvodu bylo rozhodnuto implementovat nástroj, který je hlavním předmětem této práce.

Nejdříve si objasníme design aplikace, rozmístění jednotlivých prvků aplikace a způsob ovládání 4.1. Poté probereme některé speciální ovládací prvky v kapitole 4.2. A závěrem této kapitoly popíšeme jednotlivé editory, jejich rozšíření a používání 4.3.

### 4.1 Popis aplikace

Na základě poznatků z ostatních aplikací bylo rozhodnuto o jednoduchosti a intuitivním ovládání aplikace. Tohoto bylo docíleno sjednocením základního designu aplikace a použití klávesových zkratk, které jsou známé z moderních aplikací.

Hlavní okno se skládá z několika panelů. V horní části se nachází menu obsahující jednoduché operace nad projekty. Dále se zde nachází ovládací prvky, sloužící pro práci se soubory a exporty daného souboru. Celý hlavní obsah aplikace se nachází ve zbylých dvou třetinách okna. V levé části se nacházejí tlačítka pro možnost přepínání jednotlivých vkládaných objektů a přepínání mezi druhy spojů. Dolní část obsahuje informační a editační možnosti zvoleného prvku a pro hlavní část je vyčleň zbytek pracovní plochy, takzvané editační plátno.

Na toto plátno je možné přidávat jednotlivé prvky, spojovat je, přesouvat je a mazat je. Po vybrání některého tlačítka v levém menu je možné začít přidávat prvky. Po zakliknutí vybraného prvku a kliknutí na editační plátno se vytvoří objekt, po tažení se vybraný objekt přesune a pro spojení dvou objektů hranou je zapotřebí vybrat typ hrany z levého menu a poté kliknout na první objekt a poté na druhý. Pokud je potřeba spoj zahrnout pro zřehlednění grafu, je toto možné vyvolat kliknutím na pravé tlačítko myši během tažení spoje.

### 4.2 Ovládací prvky

Pro celou aplikaci je společných několik speciálních ovládacích prvků.

K vytvoření nového projektu lze použít menu, horní toolbar, nebo klávesové zkratky. Také je možné vytvořit nový projekt kliknutím na tlačítko „plus“ vedle záložek s projekty. Pokud je potřeba zavřít nějaký projekt je možné jej zavřít také tlačítkem nacházející se v

záložce daného tabu pro projekt. Pro vytvoření nového projektu lze také použít klávesové zkratky **CTRL + T** a pro zavření zvoleného projektu je možné použít klávesové zkratky **CTRL + W**.

Při výběru přidání objektu, nebo spoje tlačítko zůstane zakliknuté, to je kvůli jednoduššímu přidání více stejných objektů na pracovní plochu.

Smazání objektu probíhá jeho výběrem a zmáčknutím tlačítka **delete** na klávesnici, tímto způsobem lze také mazat hrany mezi objekty.

Pro rozšíření editačního plátna lze přesunout požadovaný objekt mimo viditelnou oblast a editační plátno se roztáhne automaticky. Plátno je fixnuté do levého horního rohu, to znamená že nelze objekty přesunout za levý, nebo horní okraj editačního plátna pouze za pravý a dolní okraj.

Pokud je vybrána hrana a drženo levé tlačítko myši, tažením se hrana „odepne“ od druhého objektu a je možné ji odstranit kliknutím mimo jakýkoliv objekt, nebo ji přidat k jinému objektu. Při manipulaci s hranou je potřeba vybrat požadovaný typ hrany, to znamená vybrat typ hrany z levého menu a poté začít editovat hranu. Pokud vyberete odlišný typ hrany, než se chystáte editovat, tato hrana změní svůj typ na právě vybraný. Lze také vybrat hranu a až poté vybrat její typ, toto je ale mírně náročné.

Jak již bylo řečeno v 4.1 při vytváření hrany je možné tuto hranu zalomit a dosáhnout tak větší přehlednosti aplikace. Bohužel nelze tyto zalomení editovat, ale rozšíření aplikace o tuto funkcionalitu není náročné, jenom toto nebylo považováno za nutnost.

Občas jsou některé hrany ukončeny šipkou, nebo rozšiřujícím obrazcem. Toto je popsáno v kapitole 5.2. Některé hrany diagramů mohou být ukončeny šipkami na obou stranách, pro přehlednost aplikace byla ale tato funkcionality zakázána a všechny hrany jsou tím pádem **Jednosměrné** a to ve směru z prvního objektu k druhému.

Aplikace bohužel neobsahuje možnost návratu akce a opětování akce, jak je tomu u většiny aplikací. Toto je pravděpodobně největší slabinou tohoto nástroje. Absence této funkce není příliš omezující, znamená to pouze pro uživatele opakování stejných akcí. Bohužel při vývoji na tuto funkci nebyl brán zřetel a vytvoření této funkce by znamenalo nedokončení některých velice důležitých částí. Tento nedostatek je brán na vědomí a v rámci pokračování vývoje aplikace by toto mělo být jedno z prvních důležitých rozšíření.

Nelze nastavit odlišný vzhled celé aplikace, jako například barvy jednotlivých objektů, jejich styl a chování. Při vytváření nástroje nicméně s touto možností bylo počítáno, ale opět jako s návratem akce bylo prioritizováno dokončení důležitějších částí. Naštěstí je aplikace na toto plně připravena a pokud bude pokračováno v dalším vývoji přidání takového nastavení nebude vyžadovat přílišné úsilí.

Velice nepraktické je bohužel také to, že uživatel není nijak informován o stavu uložení právě editovaného souboru. To znamená že při zavření souboru není vyzván k jeho uložení a také že není hned na první pohled viditelné, zda právě editovaný soubor je uložený, nebo ne.

### 4.3 Jednotlivé editory

Jednotlivé editory jak již bylo řečeno v přechozích kapitolách jsou **diagram případů užití 3.2**, **diagram tříd 3.3** a **objektově orientované Petriho sítě 3.4** tyto editory byly nadále mírně pozměněny, převážně z grafického hlediska pro zlepšení přehlednosti těchto diagramů a pro jednodušší práci mezi nimi.

Základním odlišením ve všech editorech je zvýraznění objektu co nemá žádnou spojující hranu, jsou zvýrazněny pomocí čárkování. Vybraný objekt je znázorněn ohraničením a



odlišnou barvou. Pokud objekt nemá odpovídající „obraz“, je zvýrazněn kurzívou a textem pod objektem, toto platí převážně pro propojení diagramu tříd a diagramu případů užití. Pokud spoj mezi dvěma objekty nemá tento obraz je zvýrazněn odlišnou barvou.

#### 4.3.1 Editor případů užití

Tento editor je prvním editorem a slouží pro popis chodu aplikace, v tomto editoru se nachází pouze mírné úpravy.

V případě vybraní **Aktéra** je znemožněno uživateli spojit stejné objekty jinak jak Děděním. Dále může uživatel spojit **Aktéra** a **Případ užití** pouze pomocí asociace. Toto lze provést pouze ve směru od Aktéra, tudíž pokud toto uživatel provede naopak je hrana otočena automaticky.

Pro změnu jména objektu je potřeba na něj poklepat dvakrát, zde je také možné změnit jeho propojení s třídou. Je možné propojení změnit, nebo reaktivovat spoj mezi třídou a objektem, pokud v Diagramu tříd není třída odpovídající editovanému objektu. S touto reaktivací se také reaktivují spoje mezi třídami a nebo mezi objekty v Diagramu případů užití.

Pokud chce uživatel hromadně reaktivovat objekty případů užití a opět aktivovat a vytvořit třídy v Diagramu tříd lze toto provést kliknutím na tlačítko „Reactivate all inactive“. Pokud na druhou stranu, uživatel chce hromadně smazat všechny neaktivní objekty, může kliknout na „Delete all inactive“.

#### 4.3.2 Editor Diagramu tříd

V tomto editoru je možné již popsané chování aplikace rozdělit pomocí objektově orientovaného programování popsaného v sekci 3.1. Je možné vytvořit jednotlivé třídy a ty poté editovat.

Pokud je vytvořen a vybrán **Interface** není možné jej spojit s ostatními třídami jinak, než spojem **Implementace**, nebo mezi jednotlivými Interfacy pomocí spoje **Generalizace**.

Pro editaci jména třídy je nutné dvakrát na ni poklepat v okně, které vyskočí po dvojitém kliknutí na třídu je také možné nastavit její „roli“ v systému.

- **Actor** po vybrání tohoto druhu třídy se v Diagramu případů užití vytvoří Aktér.
- **Activity** pokud je vybrán tento typ třídy je v Diagramu případů užití vytvořen případ užití.
- **None** tento druh třídy slouží pro vytváření pomocných tříd.
- **Interface** pouze pro vytvoření metod a třídních proměnných, nelze pro ně vytvořit Petriho síť.

Výběr role je možný pouze u tříd, které nemají doposud žádný spoj. Třídy **None** a **Interface** se nepromítnou do diagramu případů užití, slouží pouze ke snadnější implementaci některých funkcí.

Pokud je potřeba znovu aktivovat a vytvořit obraz v Diagramu případů užití je potřeba třídu nejdříve spojit s další třídou a poté po dvojkliku na tuto třídu bude vyvoláno okno s editací a tato třída bude mít možnost opět aktivace objektu a nebo spojení s jiným objektem.

Pro přidání metod a třídních proměnných se nachází v pravé dolní části editor tříd. Nejdříve je zapotřebí vybrat třídu pro kterou požadujeme přidání některého prvku a poté

zadat viditelnost, jméno a typ. Příslušný řádek slouží pro třídní proměnnou, nebo metodu. Pro zjednodušení je možné nastavit typ proměnné a operace, jako typ jakékoliv třídy, která se momentálně nachází na editačním plátně.

Po úspěšném vložení metod a popřípadě třídních proměnných je možné přidat argumenty metodám, to se provádí v levé spodní části. Po výběru třídy, pro kterou je potřeba editovat argumenty metody, dvojklikem vybereme metodu, kterou je potřeba editovat a poté zadat argumenty oddělené čárkami. Pokud chceme odstranit některou třídní proměnnou, nebo metodu, je toto možné provést tlačítkem vedle položky.

### 4.3.3 Editor objektově orientovaných Petriho sítí

Pro vstup do tohoto editoru je zapotřebí vybrat třídu pro kterou budou editovány Petriho sítě. Automaticky je vybrána Petriho síť pro třídu, přepnutí mezi Petriho sítěmi pro třídu a metody probíhá v levém dolním rohu.

V pravém dolním rohu, pokud není vybrán žádný objekt na pracovním plátně, je možné přidat třídní proměnné, nebo argumenty metody jako **Místa**. Takto přidané místa nelze editovat. Vkládat konstanty, operovat s těmito místy, nebo je mazat lze, ale měnit jejich jméno není povoleno. Podobně tomu je u návratových míst v Petriho sítích pro metody kde je jedno speciální místo **return** toto místo je podobně needitovatelné a navíc jej není možné odstranit.

Přidané místa je možné dále editovat pomocí vkládání konstant, jednoduše vybrat místo a v pravém dolním rohu aplikace lze vidět aktuální konstanta (pokud není žádná je input prázdný), která lze změnit.

Po přidání **Přechodu** (často nazývaném transition) je možné tomuto Přechodu editovat dvě složky. Složku **Stráž** (nazýván též Guard) a složku **Akce** (Activity). Výběrem přechodu se zobrazí v pravé části možnost tyto složky editovat. Po kliknutí na příslušné tlačítko budou vyvolány okna pro editaci jednotlivých složek.

- **Složka Stráže** Pro základní vytváření pravidel je v horní části okna pro editaci stráže volné textové pole, kam je možné zadat jakýkoliv text. Pokud je potřeba zadat více kontrolovaný vstup je možné si ve spodní části vybrat proměnné, nebo zadat konstanty a vybrat jejich podmínku. Poté lze nastavit horní text přímo na tuto podmínku, nebo spojit operátorem s přechozí podmínkou. V textové části lze poté výslednou podmínku editovat a dále upravovat.
- **Složka Akce** Pro vytvoření nové proměnné je možné využít horního pole, je možné také vybrat proměnnou, která je již spojena s tímto přechodem. Další editace akce probíhá přes textové pole, toto pole ale není nijak kontrolované a upravované. Rozdělení takového textu a kontrola je příliš náročná, navíc nebyla předmětem této práce. Ale přidání veškerých kontrol na podobné texty v Objektově orientovaných Petriho sítích je jistě dobrý plán na rozšíření tohoto nástroje.

V objektově orientovaných Petriho sítích je pouze jeden spoj, pro tento spoj platí omezení v podobě možnosti spojit pouze Místo s Přechodem, nebo naopak. Není však možné spojit dvě místa, nebo přechody přímo mezi sebou.

Další rozšíření spojů oproti ostatním editorům je to že lze přidávat proměnné, které jsou přes tento spoj „přenášeny“. Po dvojkliku na hranu si lze vybrat množinu proměnných které jsou automaticky hlídány z míst a přechodů, nebo zadat samostatný text. Tento samostatný text je zde kvůli náročnějším programům a náročnějším operacím, jako je například

pole, nebo seznam. Tento text však není nijak kontrolovaný, převážně kvůli náročnosti na rozdělení do tokenů a provedení analýzy těchto tokenů. Pokud je potřeba dát najevo že hrana neobsahuje žádné data, lze ji nechat prázdnou, popřípadě uvést **null**, nebo **nill** do textového pole spoje.

#### 4.3.4 Provázání jednotlivých částí

Aplikace poskytuje jednoduché provázání a spojení částí, které spolu souvisí. Jako je tomu například u Diagramu případů užití a Diagramu tříd. V případě editace a vytvoření v jedné části vede k vyvolání akce v druhé. Toto chování je částečně zautomatizováno a tím ulehčena práce, nicméně uživateli je ponechána volná ruka v těchto ohledech a pokud potřebuje změnit předgenerované části nejsou mu kladeny žádné překážky.

Aplikace pouze napomáhá vývoji a napomáhá postupnému vytváření určitých částí, nemá za úkol být naprosto zautomatizována a vytvářet vše bez zásahů od uživatele.

## Kapitola 5

# Rozšířené možnosti nástroje

Během vývoje aplikace bylo počítáno s některými rozšířeními a potřebnými úpravami editoru. Převážně z toho důvodu, aby aplikace byla snáze použitelná a přinášela nové prvky.

Řekneme si něco o způsobech exportu aplikace a proč byl vybrán tento druh exportů [5.1](#). Dále si popíšeme v čem je tento nástroj vylepšený oproti konkurenci po grafické stránce některých ovládacích prvků v kapitole [5.2](#).

### 5.1 Export diagramů

Jako jeden z požadavků bylo požadováno exportu do nějakého formátu. Tohoto bylo využito v případě ukládání stavu aplikace, kdy otevřený projekt je možné uložit do formátu **XML**. Tento formát slouží pro ukládání a načítání projektů. Formát XML byl zvolen částečně kvůli jeho možnému upravení, kdy je možné editovat obsah objektů bez nutnosti zaplého nástroje. Toto se ale doporučuje pouze zkušeným uživatelům, protože je možné že se díky tomuto jakýmkoliv způsobem poruší editovaný soubor, který již nebude možné jakkoliv opravit. Pro export do XML byla využita speciální knihovna blíže popsána v kapitole [6.1](#).

Velice užitečným rozšířením je dále možnost vykreslit námi editované diagramy a sítě do formátu **PostScript** (formát EPS). Tímto lze získat obrázky vhodné například pro vědecké práce a pro vložení do dokumentů. V tomto textu byly použity převážně obrázky pocházející z výsledné aplikace. Editovaný projekt se uloží do samostatné složky pod jeho jménem a všechny editory jsou uloženy v této složce. Tato funkcionality byla přidána pomocí knihovny, kterou blíže popíšeme v kapitole [6.2](#).

Při implementaci bylo uvažováno o dalších možnostech exportu, jako například převedení objektově orientovaných Petriho sítí do **PN talku**. Vzhledem k jednoduchosti ukládaného souboru je možné toto rozšíření velice jednoduše přidat. Po přidání tohoto rozšíření by aplikace získala značně na funkčnosti a použitelnosti. Proto byl brán ohled na podobná rozšíření již během implementace základní aplikace. O dalších navrhovaných rozšířeních je možné se dozvědět v kapitole [8.1](#), kde popíšeme možná rozšíření.

### 5.2 Grafické prvky

Pro práci s grafickými prvky na editačním plátně bylo použito různých technik. Například pro detekci objektu pod ukazatelem, nebo detekci hrany pod kurzorem a podobné.

### 5.2.1 Detekce objektu pod kurzorem

Zjištění zda se nachází nějaký objekt pod kurzorem bylo velice jednoduché. Každý objekt má výpočet pro toto zjištění. Tento způsob zjištění objektu byl zvolen převážně kvůli jednoduchosti přidání nového tvaru.

$$\begin{aligned}isX &= |mouseX - objectX| \\ isY &= |mouseY - objectY|\end{aligned}\tag{5.1}$$

Kde (5.1):

- **objectX** a **objectY** jsou souřadnice X a Y pro bod objektu.
- **mouseX** a **mouseY** určují bod kurzoru.
- **isX** a **isY** je vypočítaná vzdálenost na osách X a Y.

Výpočet je prováděn tak, že se vypočítá vzdálenost na osách mezi body a tato vzdálenost se poté prověří zda leží v daném objektu. Porovná se šířka a výška oproti těmto vzdálenostem. Pokud bod kurzoru leží v těchto hranicích, vykoná se příslušná operace, například zvýraznění objektu.

Tento způsob je velice náročný na paměť, převážně kvůli neustálému volání při jakémkoliv pohybu myši. Pro celkovou aplikaci, ale toto omezení nehraje žádnou roli. Pokud by bylo potřeba změnit zjištění takového objektu pomocí sofistikovanější funcce, změna na takovouto funkci není nikterak náročná.

### 5.2.2 Výpočet vzdálenosti hrany od bodu

Na druhou stranu zjištění hrany pod kurzorem již je poněkud náročnější. Převážně z toho důvodu, že hranu máme definovanou jako dvojici bodů. Výpočet pro takové zjištění je poněkud složitější a možností použití různých vzorců pro toto zjištění mnoho.

V tomto nástroji jsou použity dvě metody zjištění vzdálenosti bodu od přímky. Jedna metoda je použita pro výpočet vzdálenosti kraje objektu k bodu a na základě toho je hrana zkrácena (použito převážně pro vykreslení šipky na konci hrany). Druhá metoda slouží pro výpočet bodu kurzoru a úsečky mezi dvěma body.

#### Výpočet kraje objektu k bodu

Před samotným výpočtem průsečíku okraje objektu a přímky z něj vycházející je potřeba zjistit v jakém sektoru se nachází druhý konec úsečky (střed druhého objektu). Pro zjištění sektoru jsou použity úhlopříčky objektu a to ve stylu, že do vzorců pro úhlopříčku dosadíme bod X, nebo bod Y a výsledný bod prověříme zda leží pod a nebo nad touto úhlopříčkou. Pro výpočet bodu lze použít jednu z rovnic (5.2).

Pro samotný výpočet bodu je použito shodnosti trojúhelníků, s tím e máme zadaný střed objektu ke kterému se snažíme zjistit protínaný okraj a bod odkud vychází tato příčka. Proto je po vybrání sektoru zjištěn poměr stran pomocí podělení šířky, nebo výšky (záleží na sektoru) a vzdálenosti středu objektu od bodu na ose X, nebo Y (opět záleží na sektoru). Při zjištěném poměru stran se dopočítá druhá strana trojúhelníku a vynásobí se s tímto poměrem.

Výpočet zda se bod nachází pod, nad a nebo na úhlopříčce.

$$resultY = \frac{YboduA - YboduB}{(XboduA - XboduB)(givenX - XboduA)} + YboduA \quad (5.2a)$$

$$resultX = \frac{givenY - YboduA}{\frac{YboduA - YboduB}{XboduA - XboduB}} + XboduA \quad (5.2b)$$

Kde pro (5.2) znamená:

- **resultX** a **resultY** výsledná souřadnice bodu X, nebo Y.
- **XboduA** a **YboduA** souřadnice prvního bodu úhlopříčky.
- **XboduB** a **YboduB** souřadnice druhého bodu úhlopříčky.
- **givenX** a **givenY** souřadnice X, nebo Y z daného bodu, zde je tento bod souřadnice středu druhého objektu.

### Výpočet bodu kurzoru a úsečky

Výpočet vzdálenosti kurzoru od úsečky je prováděn odlišným vzorcem. Při tomto výpočtu se nejdříve určí zda se bod nachází v rozmezí dané přímky. To znamená, zda se bod nachází pod a nebo nad úsečkou. Pro určení bodu vůči přímce byla použita rovnice (5.3).

$$h = \frac{(Px - Ax)(Bx - Ax) + (Py - Ay)(By - Ay)}{delkaU secky} \quad (5.3)$$

V momentě kdy výsledné **h** z rovnice (5.3) je v rozmezí  $0 < h < 1$  bod se nachází pod a nebo nad touto úsečkou. Lze tedy vypočítat vzdálenost kurzoru od úsečky.

$$\frac{(Bx - Ax)(Py - Ay) - (Ax - Px)(By - Ay)}{delkaU secky} \quad (5.4)$$

Pro rovnice (5.3) a (5.4) platí:

- **Px** a **Py** jsou souřadnice kurzoru.
- **Ax** a **Ay** reprezentují souřadnice bodu **A** úsečky **AB**.
- **Bx** a **By** slouží jako souřadnice bodu **B** na úsečce **AB**.

Vzorce pro výpočet polohy bodu vůči úsečce a pro finální výpočet vzdálenosti tohoto bodu od úsečky byly převzaty z [3].

## Kapitola 6

# Použité knihovny a ikony

Pro zjednodušení práce a možnosti zaměření se na důležitější části nástroje bylo použito několik knihoven pro urychlení vývoje. Tyto knihovny jsou volně ke stažení s velice přehlednou dokumentací. Jejich použití je velice jednoduché a rychlé. Použitím těchto knihoven vedlo k urychlení vývoje důležitých komponent, nicméně tyto knihovny nejsou nutností pro výslednou aplikaci. Pokud by se ovšem odtrhly, bylo by nutné tyto knihovny nahradit vlastní implementací těchto funkcí.

### 6.1 XML export

V případě exportu do XML bylo zvoleno ukládání veškerých grafů do speciální struktury, kterou lze velice jednoduše předat knihovně s názvem **XStream**<sup>1</sup>. Tato knihovna se postará o bezproblémovou serializaci objektu do XML.

Výhodou této knihovny je jednoduchost použití jak při serializaci, tak deserializaci. V případě načtení souboru je práce s touto knihovnou ještě snazší. Další výhodou této knihovny je možnost nastavit aliasy pro XML tagy, takže je možné upravit XML soubor, který bude používán v tomto nástroji pro další potřeby. Pokud například bude potřeba soubory exportované tímto nástrojem upravit v další aplikaci, která dokáže číst a vytvářet XML soubory, je možné upravit vstup a výstup této knihovny tak, že nebude potřeba upravovat načítání a práci s těmito soubory v jiné aplikaci.

Použitím této knihovny bylo docíleno bezchybného, rychlého a jednoduchého vytvoření souboru pro ukládání editovaného projektu.

### 6.2 PostScript export

Export do souboru PostScript nebyl nutností pro tuto aplikaci, nicméně bylo zvoleno, že export do tohoto formátu bude velice užitečný a nástroj tím získá velmi silnou funkcionalitu.

Pro export do formátu EPS bylo vyzkoušeno několik nástrojů, žádný z nich ovšem nebyl tak jednoduchý na použití a nepodporoval tolik funkcí, jako tomu bylo u Nástroje **EPS Graphics**<sup>2</sup>.

Tento nástroj vyčnívá nad konkurencí převážně v množství implementovaných funkcí. Byl vybrán převážně kvůli tomu, že veškerá grafika je tisknuta pomocí **Graphics2D** a tato

---

<sup>1</sup>Domovskou adresu pro tuto adresu lze nalézt na adrese <http://xstream.codehaus.org/>.

<sup>2</sup>Dokumentaci a knihovnu samotnou lze najít na adrese <http://www.abeel.be/epsgraphics/>.

knihovna podporuje většinu funkcí, které jsou používány právě v Graphics2D. <sup>3</sup>

V celém tomto dokumentu byly použity obrázky, které byly exportovány pomocí této knihovny, umožňující velice rychlé a užitečné rozšíření tohoto nástroje.

## 6.3 Použité ikony

V aplikaci bylo použito několika ikon pro zpřehlednění a k odlehčení monotónnosti designu. Přidáním těchto ikon bylo umožněno i méně zručným uživatelům jednoduše používat a osvojit si práci s tímto nástrojem.

Ikony byly vybrány z galerie Open Icon Library<sup>4</sup>, tyto ikony jsou volně ke stažení a pokud tento nástroj nebude využíván pro účely zisku neměl by být žádný problém s licencemi.

---

<sup>3</sup>Pro bližší obeznámení implementovaných funkcí je možné nahlédnout do rozsáhlé API dokumentace na stránce <http://epsgraphics.sourceforge.net/api/1.2/doc/>.

<sup>4</sup>Adresa této galerie je <http://openiconlibrary.sourceforge.net/>.



## Kapitola 7

# Testy a jejich výsledky

Záměrem této kapitoly je přednést testování a výsledky používání této aplikace. Budou představeny návrhy od možných zákazníků a možných budoucích uživatelů tohoto nástroje.

### 7.1 Nezkušený uživatel

Nástroj byl představen nejdříve uživateli, který nemá žádné zkušenosti s vytvářením programů, ani nikdy nebyl zapojen do vývoje programu. Tento test má dokázat zda je nástroj použitelný a nemá některou chybu znemožňující práci s tímto nástrojem.

#### 7.1.1 Průběh představení

Uživateli bylo dáno za úkol vymyslet program, který by mu pomáhal při práci. Tento program byl poté představen a popsán programátorovi. Při tomto testu nebyla nalezena žádná chyba a neostatek, uživatel byl spokojený s rychlostí s jakou se výsledný program rýsoval. Převážně kvůli tomu, že si mohl okamžitě vybavit věci, které již byly nachystány a které bude potřeba vymyslet.

Nástroj byl tedy velice přínosný v tomto případě a bylo možné postoupit k předání nástroje více zkušenému uživateli.

### 7.2 Zkušenější uživatel

Tento uživatel již použil diagram případů užití a rozumí návrhu aplikace. Bylo po něm požadováno, aby předvedl zda by zvládl v této aplikaci představit svoji vizi programátorovi a zda aplikaci neschází některý nutný prvek potřebný pro jednoduché ovládání.

#### 7.2.1 Průběh a výsledky uživatelské práce

Při sledování práce uživatele bylo zjištěno, že jeho učební křivka je velice strmá. To znamenalo, že návrh aplikace byl velice dobrý a uživatel si rychle zvykl na práci a po chvilce zkoušení aplikace, byl schopný vytvořit vlastní diagram případů užití.

Během tvoření tohoto diagramu bylo ovšem zjištěno, že aplikaci schází možnost okamžitého zrušení přidávání nových objektů na editační plátno a zrušení spojení mezi objekty. Tento návrh byl tedy poznamenán a později implementován.

Uživatel neměl žádnou další výtku k nástroji, kromě požadavku na vytvoření nápovědy a jednoduchého popsání jednotlivých částí nástroje.

## 7.3 Programátor

V poslední části byl nástroj představen programátorovi, kterému bylo navrženo aby vytvořil jednoduchý program. Vytvořil Diagram případů užití, poté přidal pár rozšiřujících metod a tyto metody popsal pomocí Objektově orientovaných Petriho sítí. Bylo sledováno převážně zda nechybí některá důležitá funkce, zda není příliš omezený a zda není potřeba příliš představovat tento nástroj.

### 7.3.1 Průběh testování

Tento uživatel velice rychle pochopil základní ovládání nástroje, velice ocenil různé možnosti otevírání a zavírání jednotlivých projektů. Ocenil provázání mezi diagram případů užití a diagramem tříd. Po vysvětlení jak spojit různé objekty napříč diagramy byl velice překvapen a s touto funkcí byl spokojený.

Po vysvětlení jak fungují objektově orientované Petriho sítě, byl překvapený jak jednoduše a rychle lze vytvářet jednotlivé části aplikace. A jak rychle lze popsat jednotlivé metody pomocí Petriho sítí.

### 7.3.2 Vyhodnocení rogramátorovy práce

Uživatel byl spokojený s nástrojem a ocenil byl další rozšíření v podobě exportu do zdrojových kódů a nebo simulace Petriho sítí.

## 7.4 Výsledky testování

Po otestování třemi různými uživateli bylo objeveno několik chyb, v podobě například okamžitého zrušení přidávaných objektů a spojování těchto objektů.

# Kapitola 8

## Závěr

V této kapitole se budeme postupně věnovat navrhovaným možným rozšířením, kterými by bylo vhodné tento nástroj vylepšit a rozšířit. A poté shrneme stav této aplikace, její přínos a aktuální stav. Také popíšeme některé známé chyby, nebo nedostatky.

### 8.1 Budoucnost aplikace

Tato kapitola představuje navrhované rozšíření pro tento nástroj. Nástroj je bez těchto navrhovaných rozšíření absolutně funkční, nicméně přidáním těchto funkcí a možností by z tohoto nástroje udělali pravděpodobně velice robustní nástroj.

#### 8.1.1 Navrhovaná rozšíření

Rozšíření zde navrhnuta není nutné dodržet při další implementaci a přidání dalších funkcí a vylepšení do tohoto nástroje. Nicméně jsou logicky vybrány a z pohledu momentálního stavu aplikace by znamenaly velmi mnoho. Do aplikace je samozřejmě možné přidat mnoho dalších funkcí,

#### Historie akcí

Jak již bylo zmíněno v předešlých kapitolách jistě by se vlice hodilo ukládat postupný chod akcí, přes které by bylo možné postupně kolovat, jak do zadu, tak dopředu. Což by vedlo ke snazší editaci a pohodlnější práci s aplikací. Absence této funkce momentálně neznámá nepoužitelnost aplikace, pouze ji dělá mírně těžkopádnou. Toto rozšíření je momentálně považováno za nejdůležitější a při dalším rozšíření nástroje by bylo vhodné tuto úpravu implementovat jako první.

#### Grafický design a editace designu

Grafický design a momentální vzhled nástroje není špatný, nebo nikterak nepoužitelný, je pouze jednoduchý a velice monotónní. Z tohoto důvodu by bylo velice vhodné přidat možnost změny vzhledu aplikace, ať už nastavení uživatelského rozhraní, tak celkový vzhled objektů na editačním plátně. Aplikace by díky tomuto rozšíření jistě byla více přívětivější k uživateli.

## Možnost exportu Petriho sítě

Pro lepší komunikaci mezi tímto nástrojem a nástroji, které se soustřeďují například na interpretaci Petriho sítí by bylo velice užitečné, kdyby nástroj mohl Objektově orientované Petriho sítě exportovat do specifického formátu, jako například PN <sup>1</sup>.

## Simulace Petriho sítí

Pokud by se využilo předešlého rozšíření 8.1.1 a toto rozšíření by se propojilo s dalším nástrojem, který by se staral o simulaci Petriho sítí vytvořených v tomto nástroji znamenalo by to značné urychlení vývoje aplikací. Kdy by se od zákazníka převzali vstupy a výstupy pro testovací případy a ty by se porovnály s výstupem z tohoto simulátoru, na základě kterých by se dala Petriho síť upravit.

## Vytvoření zdrojových kódů z diagramů a Petriho sítí

Po vzoru konkurentních nástrojů Umbrello 2.1 a Enterprise Architektu 2.2 by bylo velice užitečné mít možnost vytvořit zdrojové kódy pro zvolený jazyk z vytvořených Diagramů. Po zapojení Objektově orientovaných sítí do tohoto exportu by znamenalo vytvoření plně funkční aplikace. Díky tomuto zařízení by tento nástroj byl velice silnou konkurencí pro komerční i nekomerční nástroje. Oproti konkurenci má tento nástroj již velikou výhodu v podobě Objektově orientovaných Petriho sítí a jednoduchých editorů.

### 8.1.2 Shrnutí rozšíření

Bylo navrženo a objeveno mnoho možností, jak tento nástroj rozšířit a udělat tak z něj velice silný nástroj na poli UML editorů. V případě postupné implementace navrhovaných rozšíření by po dosažení **Vytvoření zdrojových kódů z diagramů a Petriho sítí 8.1.1** by z tohoto jednoduchého editoru vytvořilo velice silnou konkurenci pro nástroje jako Umbrello, Enterprise Architect, Dia a mnohých dalších nástrojů.

## 8.2 Závěrečné zhodnocení aplikace

Bez ohledu na rozšíření probrané v předešlé kapitole 8.1 je aplikace velice dobře použitelná a vhodná pro vytvoření návrhu vývoje.

---

<sup>1</sup>Bližší informace o tomto jazyku a o projektu PNTalk lze nalézt na stránkách <http://perchta.fit.vutbr.cz:8000/projekty/12>.

# Literatura

- [1] *Enterprise Architect User Guide*. Accessed: 12.04.2014.
- [2] *Umbrello UML Modeller Handbook*. Accessed: 12.04.2014.
- [3] Andrew S. Glassner, James Arvo: Graphics Gems II. In *The Graphics Gems Series A Collection of Practical Techniques for the Computer Graphics Programmer*, Academic Press, Inc., 1991, iSBN 0-12-059756-X.
- [4] Janoušek, V.: *Modelování objektů petriho sítěmi*. Dizertační práce, 2008.
- [5] Jim Arlow, I. N.: *UML2 a unifikovaný proces vývoje aplikací*. Computer Press, a.s., 2011, iSBN-978-80-251-1503-9.
- [6] Kettenis, J.: *Getting Started With UML Class Modeling*. Oracle, May 2007, accessed: 22.03.2014.
- [7] Kettenis, J.: *Getting Started With Use Case Modeling*. Oracle, May 2007, accessed: 22.03.2014.
- [8] Radek Kočí, Vladimír Janoušek, and František Zbořil, jr. : Object Oriented Petri Nets – Modelling Techniques Case Study. In *International Journal of Simulation Systems, Science and Technology*, 3, ročník 10, May 2009.

**Příloha A**

**Obsah CD**