VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

NÁSTROJ PRO PODPORU VÝVOJE SOFTWAROVÝCH SYSTÉMŮ

THE TOOL FOR SOFTWARE SYSTEM DESIGN

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE DANIEL HRUBÝ
AUTHOR

VEDOUCÍ PRÁCE Ing. RADEK KOČÍ, Ph.D.
SUPERVISOR

BRNO 2016

Abstrakt

Nástroj pro podporu vývoje systémů je aplikace sloužící pro usnadnění průběhu vývoje softwarových systémů. Pomocí diagramu užití pomáhá k lepšímu porozumění požadavků od zákazníka, následně s využitím diagramu tříd lze přepsat návrh do objektového konceptu a díky Objektově orientovaným Petriho sítím lze popsat chování systémů. V práci nejdříve popíší programy věnující se stejné problematice, poté se vysvětlí jednotlivé diagramy a Objektově orientované Petriho sítě. Následně bude představena samotná aplikace, její testování a možná vylepšení.

Abstract

The tool for software systems design is an aplication for simplification of system development. Use case diagram helps to make comunication between customer and developer better, then with the use of class diagram rewrite draft to object concept and thanks to object oriented Petri nets describe aplications behavior. First we will look on programs with similiar specialization, then describe each diagram and Object oriented Petri net. After that we will introduce application itself, testing and possible extensions.

Klíčová slova

UML diagramy, Diagram případů užití, Diagram tříd, Objektově orientované Petriho sítě, Propojení UML diagramů, Nástroj pro vývoj systémů

Keywords

UML diagrams, Use case diagram, Class diagram, Object oriented Petri nets, Connection of UML diagrams, Tool for software development

Citace

Daniel Hrubý: Nástroj pro podporu vývoje softwarových systémů, bakalářská práce, Brno, FIT VUT v Brně, 2016

Nástroj pro podporu vývoje softwarových systémů

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.

……
Daniel Hrubý
17. května 2016

Poděkování

Rád bych poděkoval mému vedoucímu práce panu Ing. Radku Kočímu, Ph.D. za odbornou pomoc, rady a profesionální přístup při tvorbě práce.

Obsah

1 Úvod

Při vývoji softwarových systémů se vyskytuje mnoho problémů při komunikaci mezi jednotlivými lidmi pracujícími na aplikaci. Základem je správné zjištění požadavků od zákazníka, který ve většině případů není obeznámen s vývojem a má pouze svojí myšlenku, kterou chce realizovat. Proto může dojít k nedorozumění mezi ním a programátorem. Pro lepší komunikaci mezi nimi slouží diagram užití, který nabízí abstrakci problému a zákazník může vidět, jaké funkce bude jeho aplikace poskytovat. Poté je potřeba z diagramu užití vytvořit základní strukturu tříd, které se posléze bude programátor držet. Následně se popíše chování jednotlivých tříd pomocí Objektově orientovaných Petriho sítí, které vychází z formalismu pro popis modelů v softwarovém inženýrství.[1]

Nástroj umožňuje tvorbu diagramu užití pro zpracování požadavků od zákazníka. Díky propojení jednotlivých diagramů vytvoří i základní strukturu tříd. To urychlí vývoj a usnadní vytvoření diagramu tříd. Následně lze každou vytvořenou třídu popsat pomocí Objektově orientovaných Petriho síti. Pro lepší práci s diagramy jsou všechny propojeny, takže změna v jednom se propíše i do ostatních. Program se snaží o intuitivní grafické rozhraní a lehké ovládání. Umožňuje uložení práce do různých formátů.

Ve druhé kapitole si popíšeme aplikace, které se zabývají podobnou problematikou. Probereme si nástroje dostupné pomocí webové služby i desktopové aplikace. Řekneme si, co všechno zvládají, jaké mají silné stránky a naopak i jejich slabé stránky, co programu chybí pro ulehčení vývoje aplikací. Třetí kapitola obsahuje seznámení s diagramy použitých v této práci. Nejdříve se seznámíme s UML modelovacím jazykem používaným při vývoji, poté s diagramem užití a diagramem tříd. Budou popsány jejich prvky, práce v diagramech a jejich využití při vývoji systémů. Čtvrtá kapitola obsahuje Objektově orientované Petriho sítě. Nejprve se seznámíme se základními Petriho sítěmi, posléze vysvětlíme Objektově orientované Petriho sítě, které se používají v našem editoru. Řekneme si, k čemu se používají, jaké mají základní prvky a jejich využití. V páté kapitole si popíšeme samotný program. Nejdříve se podíváme na grafické rozhraní aplikace, poté jeho základní ovládací prvky a objasníme si chování jednotlivých záložek. Bude vysvětlena implementace programu, import, export a použité knihovny. Předposlední kapitola obsahuje testování programu. Zahrnuje zhodnocení od testovaných uživatelů, jejich nápady na vylepšení a zhodnocení testování. V závěru se zhodnotí použitelnost aplikace v praxi a možné vylepšení.

[1] http://www.fit.vutbr.cz/research/view_pub.php?id=10637

2 Problematika nástrojů pro podporu vývoje softwarových systémů

V této části se zaměříme na nástroje pro podporu vývoje softwarových systémů. Protože neexistuje téměř žádný online nástroj pro vývoj Petriho sítí, zaměříme se spíše na diagramy užití a tříd. Kapitola bude rozdělena na dvě části – webové nástroje a desktop programy. V každé části bude vybrán jeden placený a jeden neplacený program.

2.1 Webové aplikace

V dnešní době je trend vytvářet nástroje online, protože uživatel nemusí nic stahovat ani instalovat, ale stačí mu pouze připojení k internetu a webový prohlížeč. Na internetu je několik nástrojů pro tvorbu UML diagramů – draw.io, gliffy.com, genmymodel.com atd. Jeden z nejznámějších neplacených online nástrojů nejen pro tvorbu UML diagramů je draw.io, které si popíšeme. Mezi nejlepší placené online aplikace patří Creately, které získalo i několik ocenění[2].

2.1.1 Draw.io

Draw.io[3] je neplacená internetová aplikace, která nabízí nejen tvorbu UML diagramů, ale i tvorbu grafů, mockupů a mnohé dalšího. Podporuje většinu UML diagramů. Obsahuje všechny prvky, které jsou potřebné pro tvorbu plnohodnotného diagramu užití a diagramu tříd. Pomocí nástrojů lze všechny prvky upravovat, měnit barvy, přidávat popisky, aby byl diagram co nejvíce přehledný a použitelný.

Pro urychlení vývoje diagramů existuje i celá sada klávesových zkratek. Pokud program nepodporuje námi potřebný tvar, lze ho pomocí knihoven do aplikace nahrát a dále s ním pracovat.

2.1.1.1 Výhody

Aplikace je hojně používaná od svého začátku (rok 2005). Za tu dobu prošla mnoha změnami, aby co nejlépe zvládala požadavky uživatelů. Nabízí velké množství nástrojů při vývoji, je přehledná, rychlá. Vytvořené diagramy lze uložit do počítače na pevný disk nebo na podporované internetové úložiště – Google Drive, Dropbox, OneDrive. Aplikace nabízí pro export hned několik formátů – png, svg, html atd. Rozdělanou práci lze uložit i ve formátu XML a následně zase nahrát a pokračovat dále v práci. Program obsahuje i další hojně používané nástroje pro lepší práci a přehled – lupa, undo, redo, výběr spojovacích hran, výběr všech objektů a mnoho dalšího.

Pro každý vytvořený prvek existuje celá řada nástrojů pro jeho úpravu – vložení textu, vybarvení, průhlednost, stíny atd. Všechny prvky lze jednoduše propojovat pomocí spojovacích bodů, které se snaží usnadnit provázání a všechny čáry se snaží dělat pomocí pravých úhlů.

[2] http://creately.com/presscoverage
[3] Dostupné na https://www.draw.io/

2.1.1.2 Nevýhody

Program nepodporuje vývoj Petriho sítí. I když pomocí nástrojů lze vytvořit Petriho síť v aplikaci, její tvorba je zdlouhavá a nelze vytvořit všechny potřebné prvky.

Přestože nabízí několik diagramů, žádné nejsou mezi sebou provázané, tudíž každý se musí vytvářet zvlášť. To zdržuje od dalšího vývoje.

Program nekontroluje diagramy na jejich správnost, takže při vývoji velkých vývojových diagramů může vývojář udělat chybu a aplikace ho na ni neupozorní a nemusí si toho následně všimnout. Při manipulaci se spojovacími hranami občas dochází k nevyžádanému chování, protože se snaží najít co nejkratší cestu propojení. U větších diagramů to nemusí být vždy to nejlepší řešení, co se týče následné přehlednosti.

Obrázek 2.1- Diagram užití vytvořený pomocí Draw.io

2.1.2 Creately

Creately[4] je obří webová aplikace vytvořená firmou Cinergix roku 2008. Nabízí možnost vývoje ve více než 40-ti diagramů. Zaměřuje se spíše na firmy, které tento nástroj hojně využívají. Mezi nejznámější uživatele patří Amazon, eBay, PayPal, Adobe a mnoho dalších[5]. Aplikace vyhrála i několik ocenění a bylo o ní vydáno mnoho článků.

Zvládá všechny náležitosti při vytváření diagramu užití a diagramu tříd. Nabízí velké množství nadstandardních tvarů. Každý můžeme upravovat podle vlastních potřeb – měnit barvy, velikost písma, vkládat další tvary atd.

[4] Dostupné na http://creately.com/
[5] http://creately.com/Customer-Love

Existuje i jako desktopová aplikace pro Windows, Mac OS X i Linux. Lze synchronizovat desktopovou aplikaci s internetovou a nabízí přes 1000 předem vytvořených šablon.

2.1.2.1 Výhody

Program jde nejlépe využít při práci v týmu. V placené týmové verzi nabízí sdílení projektů, real-time spolupráci na diagramech, import i export v mnoha formátech, přidávat komentáře k diagramům, poznámky k vývoji a mnoho dalších nástrojů pro usnadnění komunikace a práce i v mnohačlenném týmu.

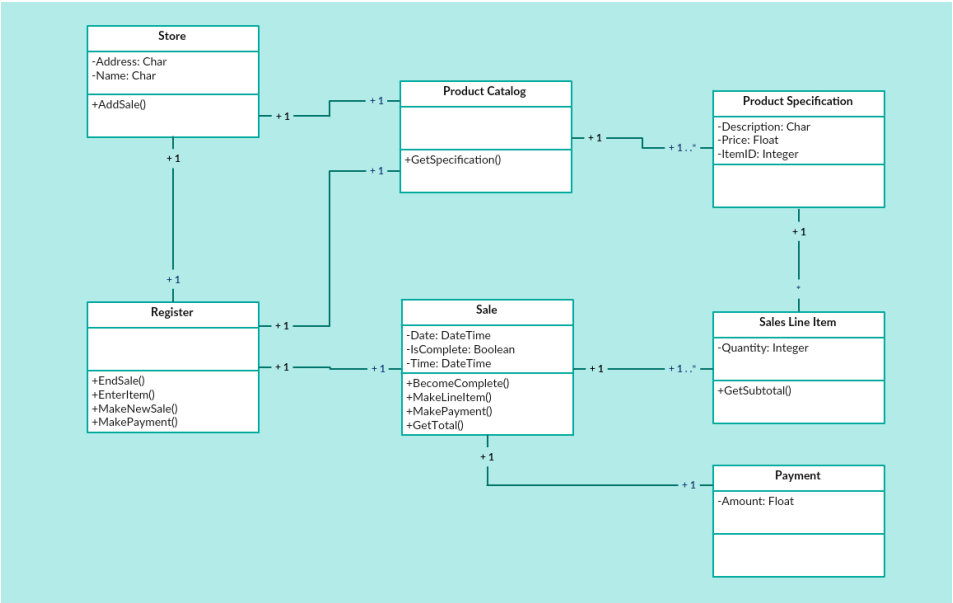
Aplikace se snadno ovládá, všechny důležité prvky jsou dobře uspořádané a jsou vývojáři co nejvíce při ruce. Podporuje všechny základní operace pro práci s diagramy – kopírovat, vložit, undo, redo, lupa atd.

Lze vytvářet projekty z mnoha diagramů, ke kterým si můžete přidávat tagy. V diagramech se jednoduše orientuje pomocí vyhledávání a filtrů. Program se snaží při vytváření diagramů udělat co nejvíce práce za vývojáře pomocí napovídaní, které je velice intuitivní.

2.1.2.2 Nevýhody

Je určen spíše pro firmy. V neplacené verzi lze vytvořit jen jeden projekt a 5 veřejných diagramů (nelze mít privátní), nelze importovat a exportovat vytvořené diagramy, což je jedna z nejdůležitějších vlastností.

I přes velikost aplikace, diagramy nejsou mezi sebou nijak propojeny, takže každý se musí vytvářet zvlášť. Chybí podpora pro vývoj Petriho sítí. Opět lze vytvořit Petriho síť pomocí tvarů a nástrojů, ale vývoj je pomalý a nepřináší všechny potřebné nástroje.



Obrázek 2.2 - Diagram tříd vytvořený pomocí Creately [6]

2.2 Desktopové aplikace

Kvůli popularitě webových nástrojů a konkurenci v podobně placených aplikací neexistuje moc kvalitních neplacených desktopových aplikací, které by splňovaly požadavky pro velké projekty. Mezi známé a uživateli chválené patří ArgoUML, který si popíšeme. Pro velké projekty se vyplatí zakoupení komerčních programů pro tvorbu vývojových diagramů. My si popíšeme známý nástroj Visio od společnosti Microsoft. Další známé programy jsou SmartDraw, Creately desktop, ConceptDraw a další.

2.2.1 ArgoUML

ArgoUML[6] je open-source desktopová aplikace pro modelování UML diagramů. Je naprogramovaný v Javě. Jeho verze 0.26 byla stažena více než 80000 krát. Poskytuje možnost tvorby 7 diagramů – diagram užití, diagram tříd, sekvenční diagram, diagram spolupráce, stavový diagram, diagram aktivit a diagram nasazení. Program poskytuje všechny potřebné prvky pro tvorbu diagramu užití a diagramu tříd.

2.2.1.1 Výhody

Program nabízí hodně možností a úprav pro vytvořené objekty. Při označení objektu program vypíše přidružené objekty rozdělené podle vztahu, jaký mezi sebou mají. Tato vlastnost se hodí při vývoji větších diagramů, kde může docházet ke ztrátě přehlednosti.

Aplikace umožňuje import i export ve formátech – zargo, xmi, xml, uml a zip. Projekty lze ukládat a následně načítat. Diagramy lze uložit i ve formě obrázků – png, gif, svg, ps a eps.

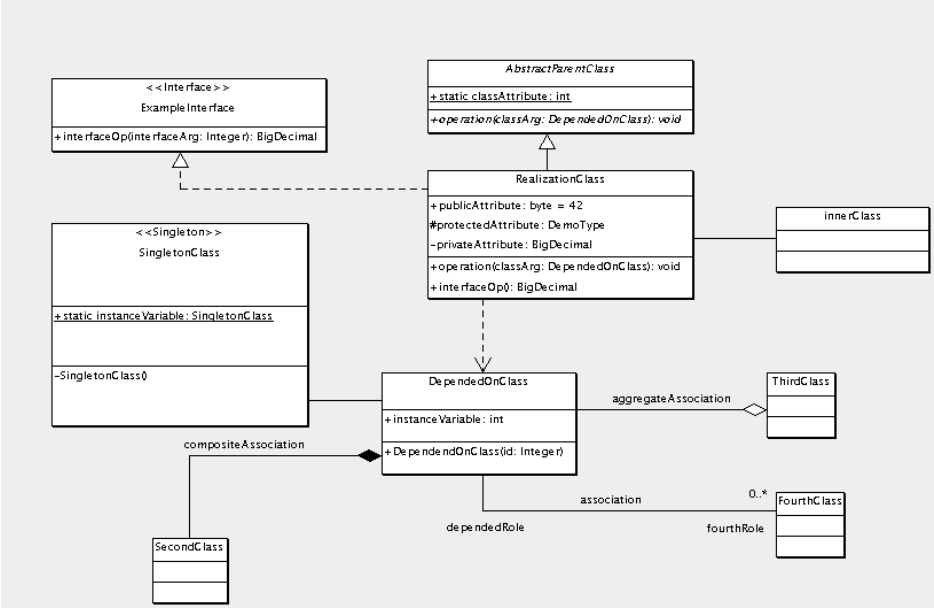
Při tvorbě diagramu tříd nabízí program možnost pro automatické generování tříd do různých programovacích jazyků – C++, C#, Java, PHP4, PHP5 a SQL. Vygenerované třídy obsahují atributy a metody vytvořené v diagramu tříd. Lze automaticky měnit notaci v třídách na UML, Java nebo C++ notaci.

2.2.1.2 Nevýhody

Program nepodporuje provázání jednotlivých diagramů. Neumí Petriho sítě, nelze je vytvořit ani pomocí nástrojů programu.

Při tvorbě velkého projektu (30 a více diagramů) se program trochu seká a je pomalejší. Při vývoji nenapovídá uživateli, neobsahuje žádnou kontrolu, takže práce na větších projektech není ideální. Aplikace se snaží nabídnout co nejvíce možností úprav, ale toto množství působí neintuitivně a uživatel může být ze začátků zmatený.

[6] http://argouml.tigris.org/



Obrázek 2.3 - Diagram tříd vytvořený pomocí ArgoUML [7]

2.2.2 Visio

Microsoft Visio[7] je program pro kreslení diagramů. Jedná se o jeden z nástrojů Microsoft Office balíku. Poprvé byl představen roku 1992 firmou Visio Corporation, kterou v roce 2000 koupil Microsoft. Aplikace se vydává ve dvou verzích – pro domácnost a pro firmy, kde každá verze je rozdělena do dalších kategorií. Velká výhoda je, že studenti mohou získat Visio zadarmo.

Program nabízí nejen tvorbu UML diagramů, ale i tvorbu map, plánů, obvodů, systémů a mnohé další. Z UML diagramů podporuje diagram užití, diagram tříd, sekvenční diagram, stavový diagram, diagram aktivit a diagram činností.

2.2.2.1 Výhody

Jedná se o velkou, komplexní aplikaci, která zvládá tvoření i velkých diagramů. Ve všem se snaží uživateli vyjít vstříc a co nejvíce mu usnadnit práci. Zarovnává objekty vodorovně i vertikálně. Spojovací čáry lze libovolně upravovat, přidávat popisky atd. Program obsahuje všechny základní prostředky pro tvorbu diagramů i mnohé nadstandardní nástroje, které proces vývoje usnadní a urychlí. Lze vytvářet několik oken, mezi kterými jde jednoduše přepínat.

Práci lze ukládat na internetové úložiště OneDrive nebo přímo do počítače. Diagramy se mohou ukládat v mnoha formátech – Visio formát, png, jpeg, emg, svg, pdf atd. Pokud zvolíme internetové úložiště, můžeme jednoduše pozvat osoby, aby se na práci podívaly.

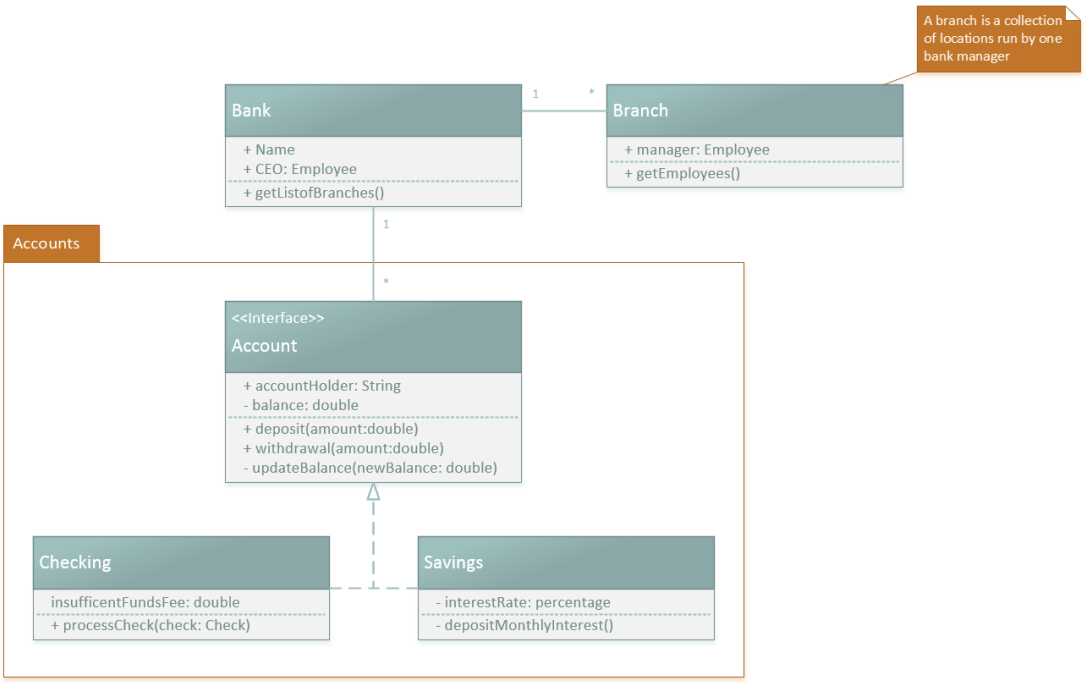
Visio má moderní vzhled, stejně jako celá sada Office. Jedná se o velice intuitivní prostředí, které pomocí mnoha možností lze upravit přesně podle potřeb každého uživatele – přidávání/odebírání karet, možnosti rychlého přístupu, velké množství doplňků atd.

Velkou výhodou jsou tzv. vzorníky, pomocí kterých lze přidávat do programu vlastní tvary. Díky této možnosti lze vytvářet i Petriho sítě.

[7] https://products.office.com/cs-cz/visio/flowchart-software

2.2.2.2 Nevýhody

Protože se jedná o dlouhodobě vyvíjenou aplikaci od velké společnosti, program nemá žádné velké citelné nedostatky. Jedna z mála nevýhod je, že diagramy mezi sebou nejsou provázané, takže například nelze z diagramu užití vygenerovat diagram tříd a je nutné ho vytvořit celý sám.



Obrázek 2.4 - Class diagram vytvořený pomocí Microsoft Visio [8]

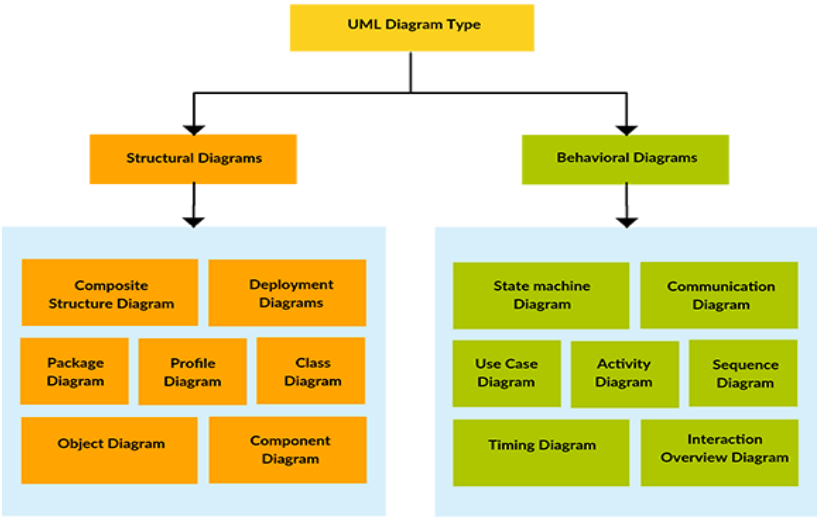
3 UML

UML (Unified Modeling Language) je modelovací jazyk používaný při vývoji systémů. Zavedl se z důvodu, že v dnešní době již nestačí jeden člověk na naprogramování celého systému. Jsou velké a komplexní, jednomu programátorovi by trvalo dlouhou dobu vytvořit takový systém. Na projektech začalo pracovat více lidí – manažeři, analytici, programátoři atd. Při práci v týmu je důležitá komunikace, tudíž se hledal prostředek, kterému porozumí každý. V průběhu 90. let se firmě Ration Software podařilo vytvořit první standard UML. Velké firmy si tento jazyk hned oblíbily, protože jim usnadňoval práci a zároveň mohl sloužit jako dokumentace pro velké projekty.

3.1 Rozdělení UML

V dnešní době již existuje 14 UML diagramů, kde každý má různé pohledy na modelovaný systém a dá se použít v určité fázi programování systémů. Tyto diagramy jsou rozděleny do základních dvou skupin [8]:

- Diagramy struktury (Structural Diagrams)
- Diagramy chování (Behavioral Diagrams)



Obrázek 3.1 - souhrn diagramů [9]

3.2 Diagram případů užití

Diagram případů užití (Use Case Diagram) se používá hlavně pro komunikaci se zákazníkem, kde je potřeba abstrakce systému, aby zákazník dokázal popsat, co od systému požaduje bez znalosti programování. V diagramu je zapsáno, co systém musí umět a vztah k požadavkům, ale není řečeno, jak to bude implementováno. Je to většinou jeden z prvních diagramů, které se vytváří při návrhu aplikace.

[8] http://www.itnetwork.cz/navrhove-vzory/uml/uml-uvod-historie-vyznam-a-diagramy/

3.2.1 Případ užití (Use Case)

Případ užití určuje jednu z věcí, které systém bude umět – např. přidání komentáře, smazání uživatele atd. Chování je popsané jako tzv. blackbox (černá skříňka), to znamená, že všechny věci, které systém vykoná při akci, jsou skryté (popíší se v jiném diagramu).

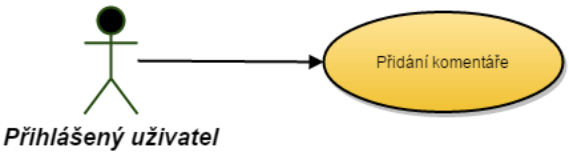
Značí se jako elipsa, která je asociována s okolními objekty. Uvnitř je název funkce, kterou vykonává (viz obrázek 3.2).

Obrázek 3.2 - Případ užití

3.2.2 Aktér (Actor)

Aktér představuje konkrétního uživatele systému. Danému aktéru jsou přiřazeny případy užití, aby bylo vidět, co může vykonávat za akce. V systému může být více aktérů, každý s jinými případy užití. Uživatelé mohou od sebe i dědit, kdy k vlastním funkcím se přiřadí i funkce zděděné.

Zakreslí se jako jednoduše zobrazená postavička člověka, pod kterou je popsána funkce aktéra v systému (viz obrázek 3.3).



Obrázek 3.3 - Aktér

3.2.3 Zobecnění, specializace

Vztah používaný k rozšíření možností aktéra o možnosti rodičovského aktéra. Zděděný uživatel následně může využívat všechny funkce předka a k tomu i svoje (specializace). Naopak předek nemůže použít případy užití svého potomka (zobecnění). Používá se v diagramech, ve kterých se vyskytuje více aktérů, kteří mezi sebou tvoří tzv. strom dědičnosti.

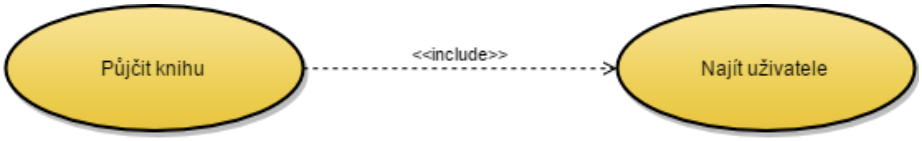
Značí se pomocí šipky, která vede od potomka k rodiči (viz obrázek 3.4).

Obrázek 3.4 - Vztah mezi aktéry

3.2.4 Vztah include

Vztah, který značí, že při použití jednoho případu užití je potřeba použít i další. Například v systému knihovny u případu půjčit knihu, musíme použít další případ najít uživatele, abychom mohli zapsat, kdo si knihu vypůjčil.

Kreslí se jako čárkovaná šipka od hlavního případu k rozšiřujícímu případu. Pro lepší čitelnost se většinou přidává i popis relace pomocí „<<include>>“ (viz obrázek 3.5).

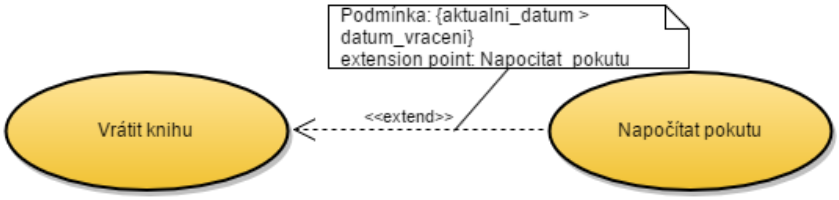


Obrázek 3.5 - Include

3.2.5 Vztah extend

Vztah se používá nepřímo, je volán (inicializován), pokud je splněna podmínka u předchozího případu užití. Například pokud zákazník vrací knihu po datu, kdy jí měl vrátit - je splněna podmínka *aktuální datum > datum vrácení*, takže se zavolá extendující případ užití *napočítat pokutu*.

Značí se pomocí čárkované šipky od extedujícího případu použití k hlavnímu a označen slovem „<<extend>>“. K šipce se váže pomocí čáry obdélník, ve kterém je popsána podmínka použití rozšiřujícího případu (viz obrázek 3.6).



Obrázek 3.6 – Extend

3.3 Diagram tříd

Diagram tříd (Class Diagram), oproti diagramu užití, již není používán pro komunikaci s uživatelem, ale mezi samotnými členy vývojového týmu. Pro jeho pochopení je zapotřebí znát základní principy objektově orientovaného programování, kterých je využíváno při návrhu. Existují dva typy diagramu tříd – analytický a návrhový. Analytický slouží k vytvoření tříd a jejich vztahů. Obsahuje pouze tzv. byznys třídy. V modelu jsou vyznačeny názvy tříd, vztahy mezi nimi, názvy metod a atributů bez datových typů a návratových hodnot. Díky tomu je platformově nezávislý. U návrhového modelu jsou již zapsány i datové typy a návratové hodnoty, tudíž je závislý na platformě.

Diagram tříd slouží jako návod pro programátora, který již nemusí přemýšlet, jaké třídy, metody a atributy bude potřebovat, vše potřebné dokáže z diagramu vyčíst, čímž se urychlí proces programování systému.

3.3.1 Popis třídy

Třída (Class), je jeden ze základních kamenů objektově orientovaného programování. Představuje objekt z problémové domény, díky čemuž se při programování dosahuje určité abstrakce a pro programátora je lehčí převést reálný problém do programovacího jazyka. Každá třída má své atributy – vlastnosti objektu a metody, které specifikují chování objektu. Pomocí třídy pak lze v programu vytvářet objekty (instance), které mezi sebou komunikují pomocí systému zasílání zpráv a mění atributy, aby odpovídaly algoritmu řešení problému.

V diagramu tříd je popsána jako obdélník, který je rozdělen na 3 části. V první horní části se uprostřed nachází název třídy (většinou tučně), který je jedinečný v celém jmenném rozsahu nebo balíku (záleží na zvoleném programovacím jazyce).

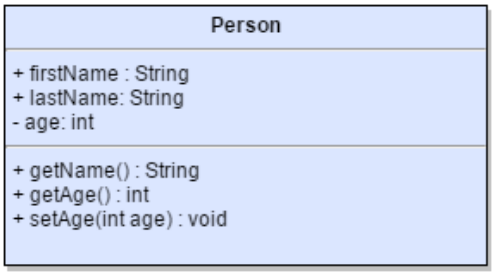
V prostřední části se nachází atributy třídy, které, dle zvolené konvence, mohou začínat malým nebo velkým písmenem a před sebou mají tzv. modifikátor přístupu – určuje, v jakém rozsahu

bude možné k atributu přistupovat. A za jménem atributu je zobrazen jeho datový typ, který je oddělen dvojtečkou.

Modifikátory přístupu

- „+“ – public – přístup je veřejný, lze odkudkoliv přistupovat
- „-“ – private – privátní přístup, pouze v dané třídě
- „#“ – protected – přístup pouze z třídy nebo zděděné třídy
- „~“ – package – přístup v daném balíčku

V poslední části jsou zobrazeny metody, které se zapisují podobně jako atributy – modifikátor přístupu název : datový typ. S tím rozdílem, že datový typ značí, jaký typ metoda vrátí, nebo void, pokud nic nevrací.



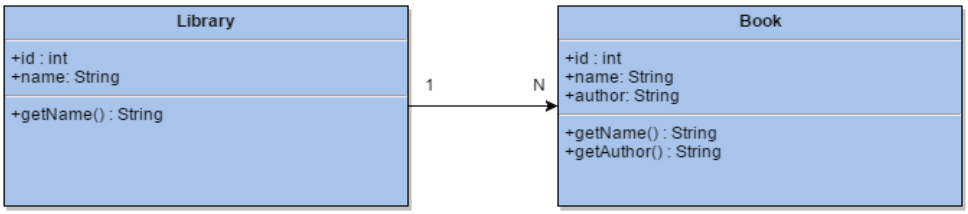
Obrázek 3.7 - Třída

3.3.2 Vztahy mezi třídami

Určují, jak se dvě nebo více tříd k sobě chovají, jaký mají k sobě vztah. V diagramu tříd existuje celkem pět vztahů: asociace (association), dědění (inheritance), realizace (realization), závislost (dependency) a kompozice (composition).

3.3.2.1 Asociace (Association)

Asociace specifikuje vztah mezi dvěma a více třídami. Může mít více stupňů – binární (mezi dvěma třídami), ternární (mezi třemi třídami), … Dále vyjadřuje kardinalitu, která udává počet objektů jedné proti počtu objektů spojené třídy - př. knihovna – kniha, *jedna* knihovna může obsahovat *n* knih. Nejčastější typy kardinalit jsou: 0..1, 1, N (někdy se používá „*“).

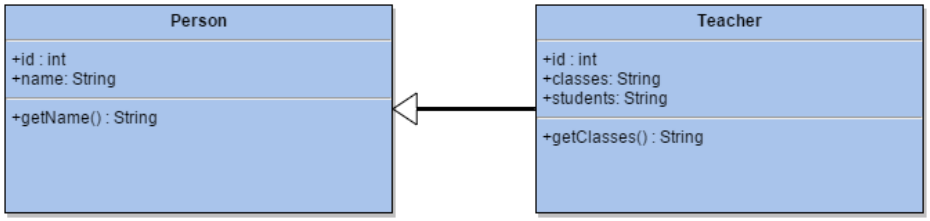


Obrázek 3.8 - Asociace tříd

3.3.2.2 Dědění (Inheritance)

Dědičnost se používá, pokud dvě nebo více tříd mají stejné prvky. Tyto prvky se vloží do rodičovské třídy. Potomci následně mohou používat atributy i metody z rodičovské třídy nebo je mohou přepsat. Díky dědičnosti se stejný kód nemusí psát vícekrát, napíše se jen jednou a pak je použit ve zděděných třídách. Říká se, že potomek je specializovaná třída, protože obsahuje to, co rodič a k tomu navíc své vlastnosti a funkce. Naopak rodiči se říká obecná třída, lze ho použít pro více než jednoho potomka. Některé programovací jazyky podporují vícenásobnou dědičnost. U jazyků, které vícenásobnou dědičnost nepodporují, se místo ní používá rozhraní (interface), které se může dědit libovolně.

Dědičnost je v diagramu značena čarou, která má na konci (u rodiče) prázdnou šipku.

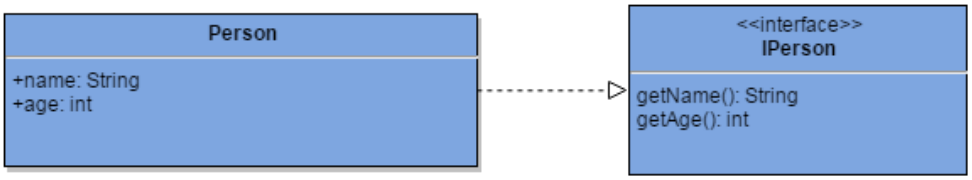


Obrázek 3.9 - Dědičnost tříd

3.3.2.3 Realizace (Realization)

Značí funkcionalitu, která není implementována ve své třídě, ale použije se z jiné. V objektově orientovaném programování značí realizace dědění rozhraní (interface) – zajišťuje správnou komunikaci mezi třídami. Určuje, jaké vlastnosti a metody bude třída mít, ale neříká, jak budou implementovány.

Realizace je značená jako čárkovaná čára, která má na konci nevyplněnou šipku. Používá se ve směru od třídy k interface.



Obrázek 3.10 - Třída implementující interface

3.3.2.4 Závislost (Dependency)

Vztah značí, že třída nějakým způsobem využívá druhou třídu. Realizuje se například použitím druhé třídy jako parametru metody v první třídě.

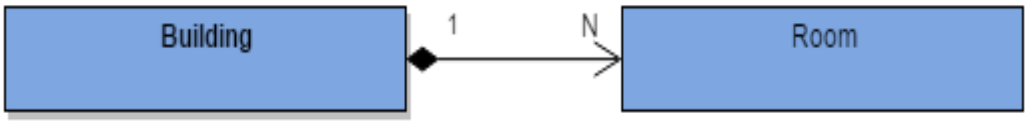
Značí se čárkovanou čarou na konci se šipkou. Směrem od třídy, která obsahuje odkaz k referované třídě.

Obrázek 3.11 - Závislost tříd

3.3.2.5 Kompozice (Composition)

Představuje „silný“ vztah mezi třídami. Je podobná jako asociace, ale s tím rozdílem, že pokud se instance třídy smaže, smaže se i přidružená třída – nemůže existovat bez druhé třídy. Vztah obsahuje kardinality (mohutnost) - 0..1, 1, N.

Značí se pomocí šipky od hlavní třídy k odkazované, kde na začátku je vyznačen plný kosočtverec. Na začátku i konci šipky jsou vyznačeny kardinality vztahu.

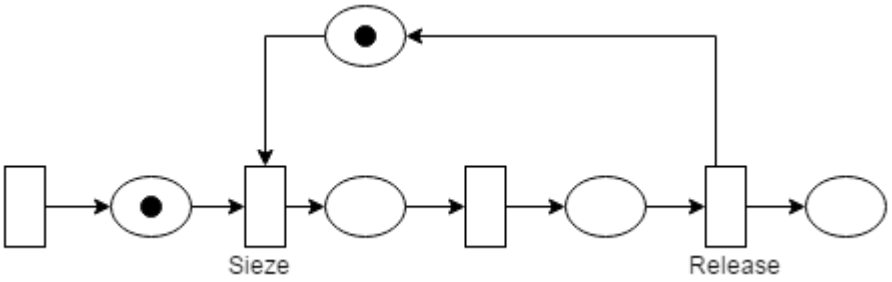


Obrázek 3.12 - Kompozice tříd

4 Petriho sítě

Jejich základy položil roku 1962 německý matematik Carl Adam Petri. Od té doby se postupně rozvíjely, aby pokryly potřeby modelování diskrétních systémů. V dnešní době existuje několik typů od C/E (Condition / Event) Petriho sítě až po vysokoúrovňové sítě typu Pr/T (Predicate – transition) Petriho sítě.

Základními částmi jsou *místa, přechody a hrany.* Hrany vždy spojují místo s přechodem nebo naopak. Nikdy však nemohou spojovat dva přechody nebo místa. Místa, ze kterých vychází hrana, jsou označovány za vstupní místa přechodu a do kterých hrana přichází, se značí jako výstupní místa. Jsou značeny kolečkem, které může obsahovat tečky tzv. tokeny. Přechod tokenů z jednoho místa do druhého se nazývá *značení*. Přechody se značí jako obdélník, mohou obsahovat prioritu, některé přechody budou upřednostňovány před ostatními – čím vyšší číslo, tím větší priorita. Dále existují pravděpodobnostní přechody. U přechodu je napsána pravděpodobnost daného přechodu. Prioritní a pravděpodobnostní přechody se nemohou nikdy kombinovat. Aby model odpovídal reálnému systému, přidává se ještě modelový čas a čas přechodů. Může být konstantní (každých 5 sekund) nebo náhodně generovaný.



Obrázek 4.1 - Obslužní linka pomocí Petriho sítí

Formální zápis Petriho sítí:

Pětice (S, T, F, M_0, W), kde:

S – konečná množina míst

T – konečná množina přechodů

F – konečná množina hran, kde žádná hrana nemůže spojovat dva stejné prvky – $F \subseteq (S \times T) \cup (T \times S)$

M_0 - $S \rightarrow N$ je počáteční označkování, kde v každém místě $s \in S$ je $n \in N$ teček

W : $F \rightarrow N^+$ je množina vážených hran, které přiřazuje každé hraně $f \in F$ nějaké číslo $n \in N^+$ označující, kolik teček je odebráno z místa tímto přechodem

4.1 Objektově orientované Petriho sítě

V systému popsaném Objektově orientovanými Petriho sítěmi (OOPN – Object Oriented Petri Nets) dynamicky vznikají a zanikají objekty, které mezi sebou komunikují pomocí zpráv. Systém je popsán množinou tříd těchto objektů, které jsou hierarchicky poskládány podle dědičnosti. Před spuštěním modelu je jedna třída označena jako prvotní a je spouštěna na začátku simulace. Třída se skládá z následujících částí:

- síť objektu, která reprezentuje instanci třídy (atributy), jejich aktivitu
- množinou sítí metod, které říkají, jak se mají chovat na různé zprávy

- množinou synchronních portů, definující odpověď na zprávy zaslané ze stráží přechodů

4.2 Síť objektu

Při vytvoření objektu pomocí třídy se aktivuje jeho objektová síť – Petriho síť znázorňující jeho logiku. V síti je každá metoda reprezentována pomocí místa, které obsahuje svojí vlastní síť, síť metody. Tokeny v síti již nejsou pouze značky, ale mohou značit reference na objekty – primitivní (řetězec, číslo, symbol…) i neprimitivní (instance tříd definovaných Petriho sítěmi).

Přechody v objektu jsou proveditelné, pokud je splněna stráž přechodu. Ta se skládá ze sekvencí výrazů, kde každý je zaslání zprávy. Aby stráž měla hodnotu TRUE, je potřeba, aby každý výraz byl vyhodnocen kladně. Výrazy se zapisují za sebou, oddělené tečkou. Př. $o\ state{:}\ x.x < 30$ – stráž se pokusí najít v místě objekt x, jehož synchronní port *state:x* vrátí objekt, který bude mít menší hodnotu než 30.

Pokud je stráž splněna, zavolá se akce, která specifikuje zaslání zprávy. Formální vyjádření akce: $y := x_0 * msg(x_1, \dots x_n)$, kde x_i je literál, *msg* selektor zprávy. Adresát zprávy je označen jako x_0 a parametry jsou předány pomocí x_1 až x_n. Přechod objektu závisí na adresovaném objektu.

Sítě metod obsahují tzv. parametrová místa, která slouží pro uložení parametrů zprávy při invokaci. Dále výstupní místo return, které vrací návratovou hodnotu metody. Značky v sítích představují reference na objekty. Známe 3 druhy objektů:

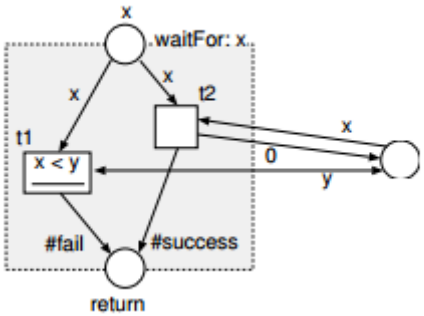
- primitivní – čísla, znaky, konstanty, seznamy,…
- neprimitivní – objekty definované Petriho sítěmi
- třídy – nemění se po dobu modelu

4.3 Sítě metod

Každá metoda objektu je reprezentována svojí vlastní Petriho sítí. Vzniká při zavolání metody, po jejím ukončení opět zanikne. Metodu lze volat n-násobně, tudíž může vzniknout více instancí metody.

Každá metoda (i metoda bez parametrů) obsahuje parametrové místo, kam se vkládají parametry při zavolání metody. K metodě lze přistoupit libovolně ze všech přechodů a metody mají přístup k místům objektu.

V každé metodě musí existovat místo *return*, které slouží k navrácení z volané metody. Konec metody se zjistí tak, že místo return obsahuje token, který se vrátí jako návratová hodnota. Pokud by obsahovalo více tokenů, náhodně se vybere jeden.



Obrázek 4.2 - Metoda zapsána pomocí Petriho sítí [1]

4.4 Synchronní porty

Synchronní porty mají charakter přechodů, slouží ke komunikaci s objekty. K tomu slouží i tzv. predikáty, ale ty jsou omezené, protože nemohou měnit stav objektu. V tom se synchronní porty liší od predikátů – mohou měnit stav objektu.

Mohou obsahovat stráž, nikoliv akci, protože musí být proveditelný atomicky, což u akce nemusí nastat vždy. Je propojen pomocí hran s dalšími místy sítě. Lze ho zavolat posláním zprávy ze stráže jakéhokoliv přechodu. Port může být ve stavu proveditelný nebo neproveditelný pro navázání proměnných. Zjišťování navázání se děje stejně jako u proveditelnosti přechodů.

.

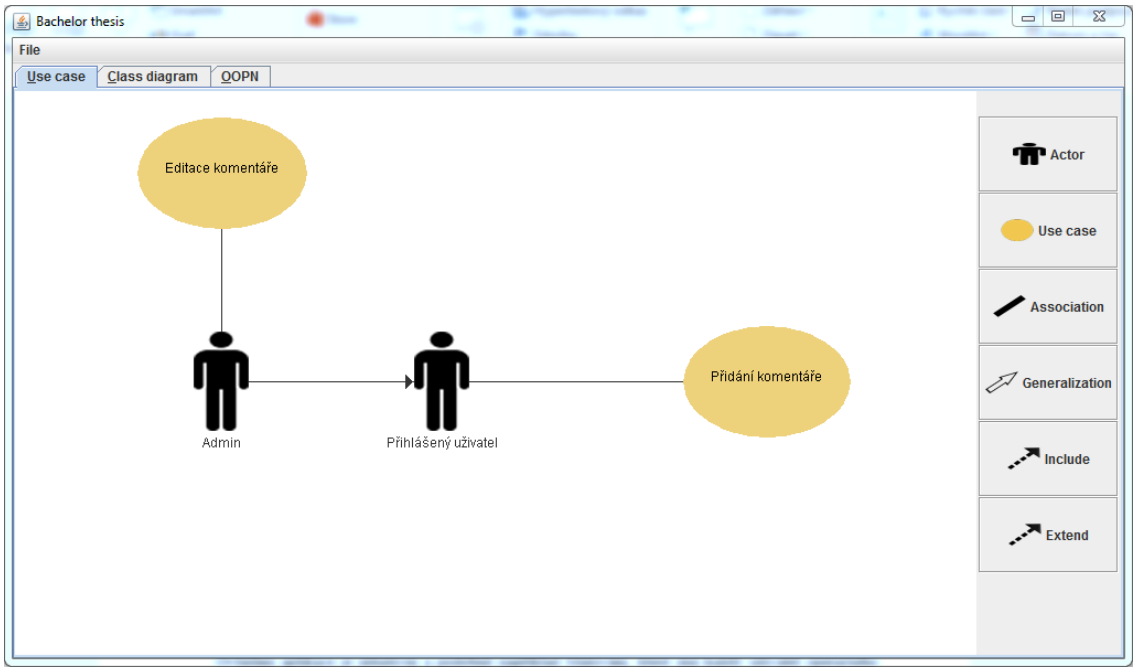
5 Popis editoru

Aplikace umožňuje popis vývoje systému dvěma diagramy – diagram užití, diagram tříd a Objektově orientovanými Petriho sítěmi. Hlavní rozdíl oproti programům popsaných v předešlé kapitole je, že naše aplikace umožňuje popsání aplikace pomocí Objektově orientovaných Petriho sítí, to žádná výše zmíněná aplikace neumožňovala. Dále podporuje provázání jednotlivých diagramů, které usnadní a zrychlí vývoj díky tomu, že změna v jednom diagramu se propíše i do ostatních.

5.1 Vzhled aplikace

Aplikace má jednoduchý vzhled, ve kterém se vyzná každý uživatel už při prvním spuštění. Snaží se o intuitivní ovládání, které je zažité z podobných programů pro kreslení vývojových diagramů.

V horní liště je nabídka akcí, které může uživatel dělat s projekty – Save project, Save graphic, Load, Help a Exit. Pod horní lištou se nachází tři taby, kde každý přestavuje jeden z navrhovaných diagramů nebo Petriho síť. Pro rychlejší orientaci se uživatel může přepínat mezi taby pomocí klávesových zkratek (levý alt + začáteční písmeno tabu). Každý tab obsahuje kreslící plochu, kde se vytváří samotné diagramy. Na pravé stráně straně je vždy panel s tlačítky pro kreslení prvků. V panelu se nachází vždy tlačítka aktuální pro námi zvolený diagram. Při vybrání prvku se tlačítko zvýrazní, aby uživatel věděl, s jakým prvkem zrovna pracuje. Při práci s Objektově orientovanými Petriho sítěmi se v tabu navíc zobrazí select box, který vypíše všechny metody obsažené ve vybrané třídě.



Obrázek 5.1 - Graficke uživatelské rozhraní aplikace

5.2 Základní ovládací prvky

Ovládání aplikace je intuitivní a podobné například Malování, které zná každý uživatel operačního systému Windows. Pro kreslení objektů je potřeba nejdříve vybrat s jakým objektem chceme pracovat vybráním tlačítka v pravém panelu. Po označení kreslíme objekt jednoduchým stisknutím levého tlačítka myši na kreslící ploše. Při propojování prvků je potřeba mít označený vztah, kterým chceme objekty spojit. Aby se začala spojovací čára kreslit, musíme kliknout na prvek. Čára se kreslí průběžně při tažení myší. Aby čára byla platná (uložila se), je potřeba dojet myší až na druhý objekt, jinak by čára nespojovala dva objekty a neuložila se.

Lze vybírat z tlačítka pro malování pomocí šipek na klávesnici, následně označit vybrané tlačítko pomocí mezerníku. Takže uživatel nemusí přejíždět myší přes celou kreslící plochu, ale může využít obě ruce, což mu zrychlí vývoj diagramů.

Všechny prvky lze libovolně posouvat po kreslící ploše. Stačí kliknout levým tlačítkem myši na požadovaný prvek a držet tlačítko po celou dobu přesunu. Při přesouvání se překreslují i všechny spojovací čáry.

Pro smazání objektu je zapotřebí na něj kliknout, kolem objektu se objeví barevný obdélník, který značí, že je vybraný a pomocí klávesy Delete můžeme prvek vymazat. Při vymazání prvku se vymažou i všechna jeho spojení s okolními objekty. Tuto operaci lze provádět i na spojovacích čárách.

Všechny změny vlastností objektů se dělají pomocí formulářů, které se zobrazí při dvojím kliknutí myši na požadovaný objekt.

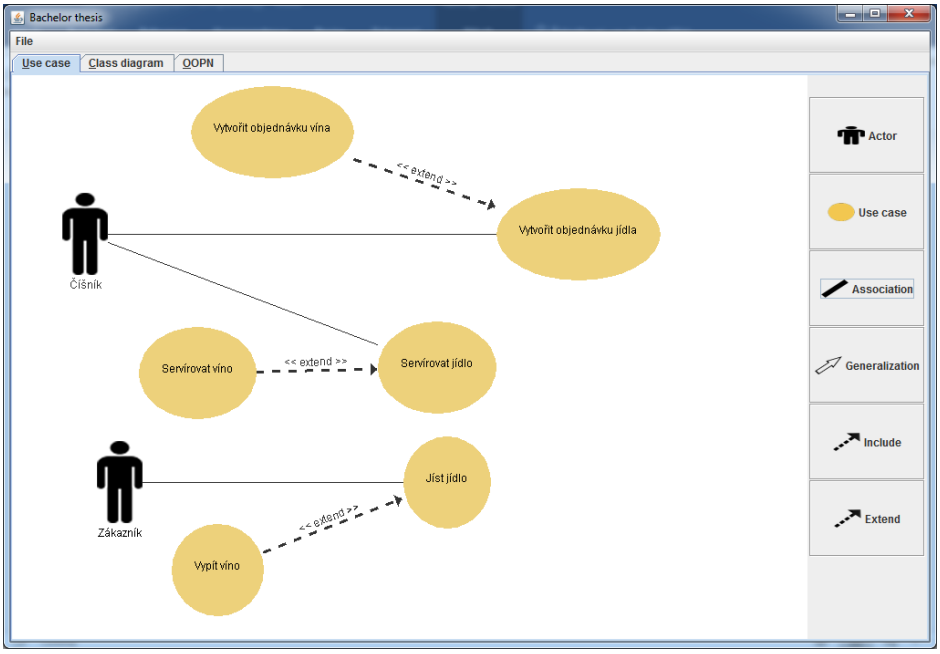
5.3 Tab diagramu užití

Jedná se o první ze tří tabů. Lze k němu přistupovat pomocí klávesové zkratky levý alt + u. Při vytváření diagramu užití se pracuje hlavně s dvěma prvky – aktér a případ užití. Tyto objekty se dají propojovat pomocí asociace, generalizace, include a extend vztahů. Mezi aktéry lze použít pouze vztah generalizace, při pokusu o jiný se spojení neuloží. U případů užití zase naopak nelze vytvořit vztah generalizace.

Při vytvoření aktéra se automaticky zobrazí pod siluetou osoby popisek User (u případu užití Use case). Pro změnu názvu je nutné dvakrát kliknout na aktéra, zobrazí se formulář pro změnu jména. Lze ho nechat i bez popisku. Popis případu užití se může měnit stejným způsobem, ale nelze ho nechat bez popisku. Z logického hlediska by pak ztratil význam. Šířka případu užití se vždy přizpůsobují svému obsahu.

Prvky použité v tabu:

- Actor
- Use case
- Association
- Generalization
- Include
- Extend



Obrázek 5.2 - Grafické rozhraní záložky Use Case

5.4 Tab diagramu tříd

Druhý ze tří tabů, slouží pro něj zkratka levý alt + c. U diagramu tříd jsou dva hlavní objekty – třída a rozhraní. Vytvořená třída má automatický název Class. Rozhraní má v popisku navíc <<*Interface*>> popisek, který značí, že se jedná o rozhraní, jako základní popisek má Interface. Oba objekty jsou rozděleny na 3 částí. První část je hlavička, která obsahuje název třídy (rozhraní). Ve druhé části jsou atributy, které se skládají z identifikátoru přístupu, názvu atributu a jeho datový typ. Poslední část obsahuje metody, které mají podobnou strukturu jako atributy, pouze místo datového typu mají návratový typ.

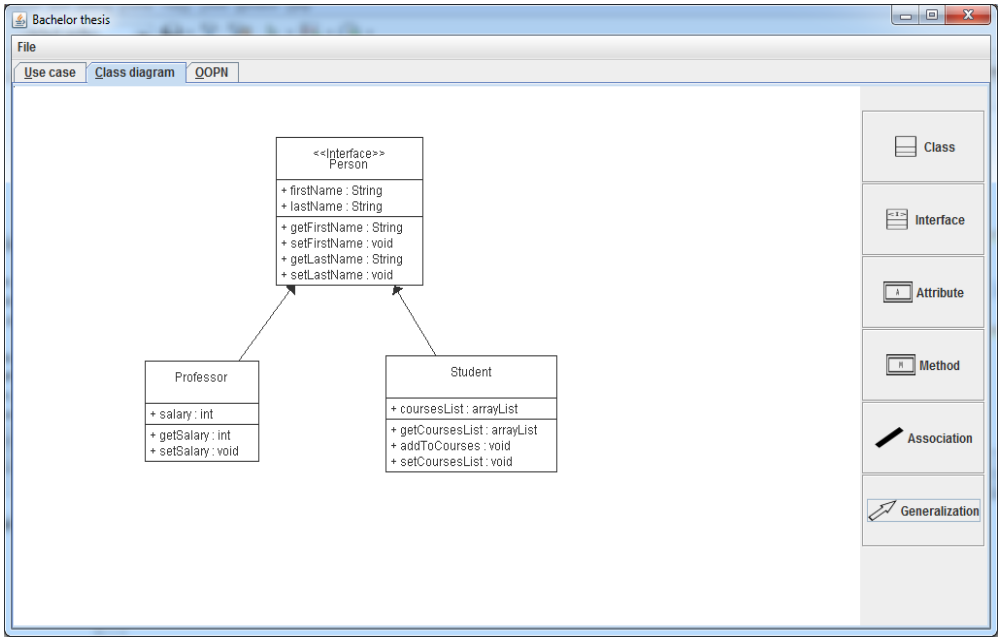
Atributy a metody lze přidávat vybráním potřebného tlačítka v pravém panelu a levým kliknutím myši na objekt. Jeho šířka se přizpůsobí obsahu objektu. Zobrazí se formulář pro přidání. Ten obsahuje select box s možnými identifikátory přístupu i s popisem (vloží se pouze identifikátor) a dvě textová pole pro název a datový/návratový typ. Aby bylo možné atribut uložit, je zapotřebí vyplnit obě textová pole. U metody lze nechat návratový typ prázdný a automaticky se nastaví jako void. Pro změnu názvu objektu musíme dvakrát kliknout na jeho hlavičku a zobrazí se předvyplněný formulář, kde můžeme upravit jméno. Jméno nesmí zůstat prázdné. Pokud chceme změnit (odstranit) atribut nebo metodu, musíme dvakrát kliknout na název a zobrazí se předvyplněný formulář se třemi tlačítky – Edit, Cancel a Delete. Zase platí, že musí být vyplněné textové pole (u metody jen název). Celý objekt lze smazat označením (pravým kliknutím myši) a stisknutím tlačítka Delete. Opět se smažou i všechny vztahy s ostatními objekty.

Třídy lze propojovat pomocí vztahu asociace a generalizace. Třída s rozhraním se může propojit pouze přes genetalizaci.

Prvky použité v tabu:

- Class
- Interface
- Attribute
- Method

- Association
- Generalization



Obrázek 5.3 - Grafické rozhraní záložky Class diagram

5.5 Tab Objektově orientované Petriho sítě

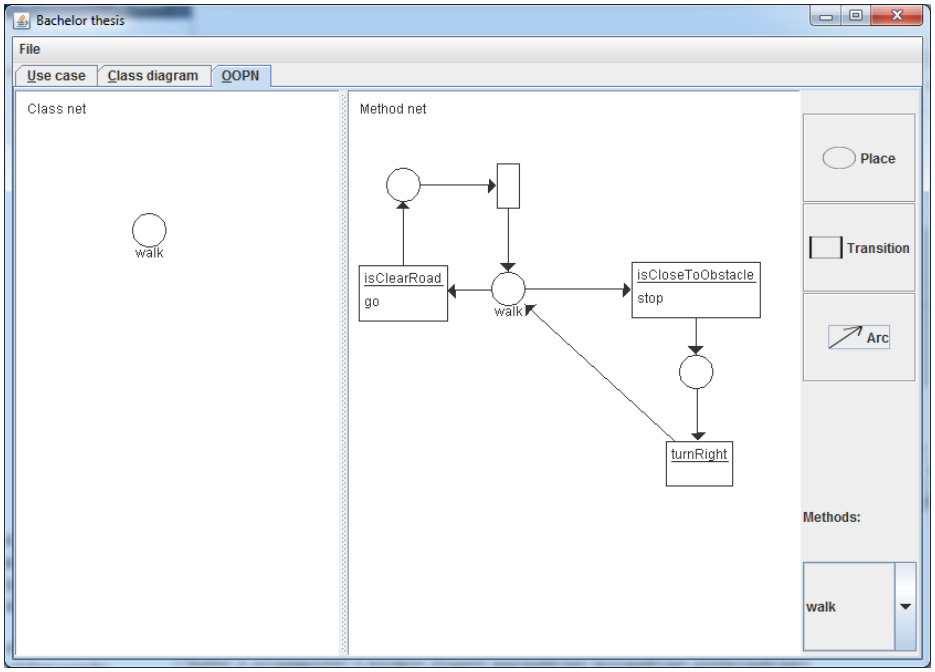
OOPN je poslední z tabů. Jeho klávesová zkratka je levý alt + o. Aby bylo možné přejít na tab, je zapotřebí mít označenou třídu, pro kterou chceme kreslit Petriho síť. V jiném případě se ukáže hláška: No class has been selected. Kreslící část je rozdělena na dvě plochy (viz obrázek 5.2). Levá část slouží pro tvorbu sítě třídy a pravá část pro síť metody. Metody se přepínají pomocí select boxu v pravém panelu. Po vybrání se síť metody zobrazí v pravé části. Lze propojovat prvky sítě třídy s prvky sítě metody. Pokud máme vybranou metodu, nelze kreslit síť třídy, nelze ani přesouvat prvky z jedné části do druhé. Pro kreslení sítě třídy nesmí být v select boxu vybrána žádná metoda.

Pro změnu názvu místa stačí na místo dvakrát kliknout levým tlačítkem myši a zobrazí se formulář pro změnu názvu s jedním textovým polem Name. Pole může zůstat i prázdné, místo nebude mít žádný název. Stráže a přechody můžeme přidávat dvojím kliknutím na přechod pomocí formuláře přechodu. Formulář obsahuje dvě textová pole, jedno pro zadání stráže a jedno pro akci. Obě pole mohou zůstat prázdné – přechod bude bez stráže a akce.

Místa a přechody se mohou spojit přes orientované hrany. Nelze spojit dva objekty stejného typu – místo:místo nebo přechod:přechod, v takovém případě se hrana neuloží. Pro přidání popisku ke spojovací hraně slouží formulář, který se zobrazí pomocí dvojkliku na hranu. Formulář obsahuje pouze jedno pole pro vložení popisku, pole může zůstat prázdné.

Prvky použité v tabu:

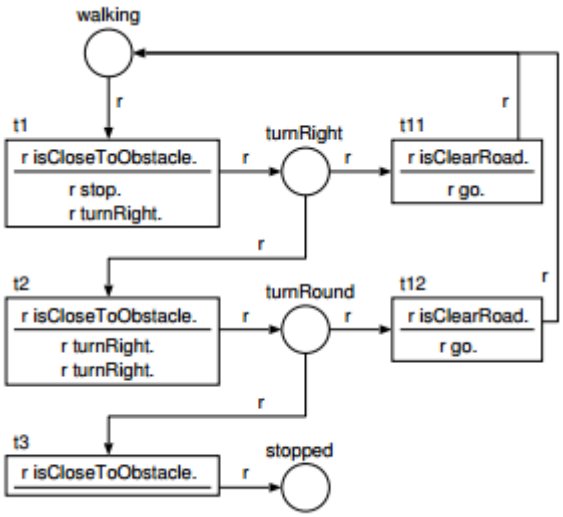
- Place
- Transition
- Arc



Obrázek 5.4 - Grafické rozhraní záložky OOPN

5.6 Propojení diagramů

Modelování, implementace a testování patří mezi základní kameny procesu vývoje systému. Modelování slouží pro definování architektury a popisu chování systému. Modely pak jsou následně transformovány do kódu programovacího jazyka a jejich testování je závislé na kódu, takže následný vývoj a ladění s použitím modelů není možné. Avšak díky vysokoúrovňových jazyků jako Objektově orientované Petriho sítě lze použít tyto modely i v cílovém prostředí. Díky provázání modelů a Objektově orientovaných Petriho síťí není potřeba generovat žádný kód a lze je použít v dalším vývoji a testování. Na obrázku 5.2 můžeme vidět Petriho síť, která slouží popsání chování robota [1].



Obrázek 5.5 - Chování robota popsané Objektově orientovanými Petriho sítěmi [1]

Vztah mezi diagramem užití a diagramem tříd je, že každému objektu v diagramu užití odpovídá jedna třída. Při změně objektu se změna propíše i do třídy. Při změně názvu se změní i název třídy. Při smazání objektu se smaže i odpovídající třída. Před smazáním se editor zeptá, jestli jste si jistý vymazáním, poněvadž akce ovlivní i ostatní diagramy.

V diagramu tříd má každá třída svoji Petriho síť. Každá metoda odpovídá jednomu místu v síti. Každý atribut odpovídá také jednomu místu, ale již neobsahuje svojí síť. Při vytvoření metody nebo atributu se automaticky vytvoří odpovídající místo v Petriho síti. Jejich změna názvu se propíše i do sítě a změna názvu v síti, se propíše do diagramu tříd. Při vymazání třídy se smaže i celá Petriho síť. Před vymazáním se program opět zeptá, zdali jste si akcí jistý.

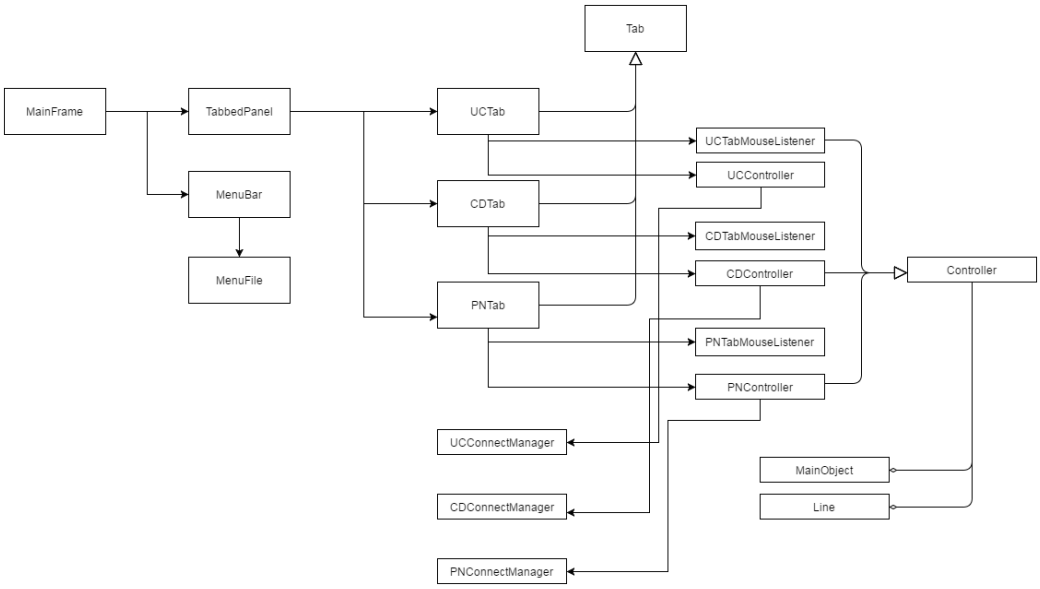
5.7 Implementace programu

Při spuštění se zavolá třída MainFrame, která nastaví základní vlastnosti oknu a vytvoří další objekty – MenuBar a TabbedPanel. Následně objekty přidá. Základní struktura programu je znázorněna na obrázku 5.6.

MenuBar slouží pro zobrazení horní lišty, která obsahuje třídu MenuFile. Ta obsahuje menu programu, které se skládá z JMenuItem prvků. Menu obsahuje položky – Save project, Save graphic, Load, Help aEexit. Každý prvek obsahuje vlastní listener, který vykoná příslušnou akci. Při kliknutí na Save nebo Load se zavolá metoda instance třídy XMLParser.

V konstruktoru TabbedPanelu se vytvoří instance třech základních tabů – Use case, Class diagram a OOPN, nastaví se klávesové zkratky a přidají se. Každý tab má svojí vlastní třídu, ale všechny dědí základní vlastnosti od předka Tab. Jednotlivé taby si vytvoří svůj vlastní toolbar, který obsahuje tlačítka, u OOPN tabu ještě select box pro výběr metody. Tlačítka jsou tvořeny pomocí třídy Button, která obsahuje ikonu tlačítka a popisek. Třída dostane v konstruktoru název souboru ikony, který je uložen ve složce resources. Každou ikonu zmenší na požadované rozměry a přidá k tlačítku. Všechny objekty daného tabu jsou uloženy v tzv. Controlleru, kdy každý tab má svůj. Controller obsahuje Arraylist objektů a spojovacích čar. Stará se o přidávání a mazání objektů, pracuje s pohybujícími a vybranými objekty. Pracuje také se spojovacími čárami. Pomocí ConnectManageru se stará o správné propojení prvků. Vždy před přidáním hrany do pole se zavolá metoda canBeConnected, které se předají spojované objekty a metoda určí, zdali se objekty mohou propojit. Controller obsahuje také formuláře, které slouží pro změnu vlastností objektů. Tab pro Objektově orientované Petriho sítě má trochu odlišnou strukturu, obsahuje totiž dva panely rozděleny pomocí JSplitPane. Uživatel díky tomu může měnit velikost jednotlivých tabů. Levý tab slouží pro tvoření sítě třídy a pravý pro síť vybrané metody.

Každý tab má i svůj MouseListener, který obsahuje metody pro práci se vstupy od myši. Listener zpracuje vstup a zavolá příslušnou metodu z Controlleru. Tab obsahuje také metodu paintComponent, která se stará o překreslení komponenty vždy při zavolání metody repaint. Metoda projde všechny objekty a nad každým zavolá metodu draw. Každý objekt se stará sám o svoje vykreslení. Všechny objekty dědí od předka MainObject, který má základní informace, které jsou společné pro všechny objekty – souřadnice, popis, výšku, šířku atd. Každý objekt má svoji vlastní třídu, ve které jsou specializované informace pro konkrétní objekty a přepsaná metoda draw pro vykreslení. O vykreslování spojovacích čar se stará třída Line, která obsahuje počáteční a koncové souřadnice, typ hrany a počáteční a koncový objekt.



Obrázek 5.6 - Diagram tříd aplikace

5.8 Import a export

V aplikaci je možný export grafiky do obrázkových formátů JPEG a PNG. Pro uložení obrázků je zapotřebí v menu kliknout na tlačítko Save graphic. Ukáže se okno, kde je potřeba vybrat, kam se soubor uloží a jak se bude jmenovat. Vždy se uloží grafika jen z právě aktuálního tabu. Při ukládání se zavolá metoda printAll nad JPanelem, která vyvolá metodu pro překreslení a to se uloží do BufferedImage. Ten převedeme pomocí ImageIO do potřebného formátu a soubor uložíme. Pro možnost následného načtení projektu je zapotřebí uložit projekt v XML formátu. K tomu slouží třída XMLParser, která obsahuje dvě základní metody load a save. Při kliknutí na tlačítko load program vyzve uživatele k vybrání souboru. Pokud soubor bude jiného typu než XML nebo nebude obsahovat validní vstup, v aplikaci se objeví odpovídající hláška. Při ukládání se objeví okno pro výběr místa uložení a zadání názvu souboru. Soubor se vždy uloží jako XML soubor.

5.8.1 XML struktura souboru

Pro možnost následného přiřazení asociovaných objektů, obsahuje každý objekt své jedinečné ID, které je generováno pomocí třídy UUID. Následně každý objekt, pokud má asociovaný objekt, obsahuje ID svého přiřazeného prvku. Díky tomu se po načtení projektu z XML souboru mohou objekty znovu propojit a ke spojovacím hranám lze přiřadit počáteční a koncové objekty.

Exportovaný soubor má základní element root. Ten má dva elementy – UseCase a ClassDiagram. Use case obsahuje uzly Actors, UseCases a Lines. V Actors a UseCases jsou uloženy všechny objekty a jejich vlastnosti – id, id asociovaného objektu, souřadnice, výška, šířka atd. V Lines jsou uloženy všechny propojovací hrany. Každá hrana má uložené svoje počáteční a koncové souřadnice, identifikátor počátečního a koncového objektu a svůj typ.

Uložení diagramu tříd je o něco těžší, protože každá třída navíc obsahuje svojí Petriho síť. Uzel ClasDiagram obsahuje dva hlavní elementy - Classes a Interfaces. Classes obsahuje jednotlivé třídy tvořené uzly Class. Nejdříve jsou uloženy všechny atributy a metody, které si ukládají své id, id asociovaného objektu, identifikátor přístupu, název a datový typ/návratovou hodnotu. Poté se do uzlu

PetriNet uloží Petriho síť. Uloží se všechna místa sítě. Pokud se jedná o metodu, uzel obsahuje další element PetriNet, ve kterém je uložena síť metody. Každé místo má uložené své základní informace. Následně se uloží přechody, které si navíc uloží i stráž a akci. V interfaces jsou uložená jednotlivá rozhraní. Každé rozhraní má informace o svých atributech a metodách.

Na ukázce exportovaného souboru (kód 5.1) můžeme vidět, jak vypadá soubor exportovaný z editoru a pod ním kreslící plochu tabu UseCase po importu souboru (obrázek 5.7). Jedná se o jednoduchý diagram užití, který obsahuje pouze dva objekty. V diagramu tříd není vytvořen žádný objekt, z toho vyplývá, že neexistuje ani žádná Objektově orientovaná Petriho síť, takže zbylé taby jsou prázdné.

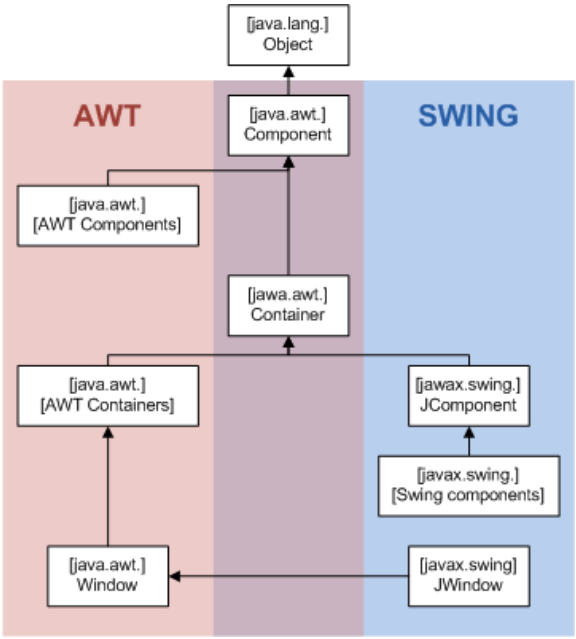

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<root>
  <UseCaseDiagram>
    <Actors>
      <Actor>
        <ID>b7bb54fb-b1ac-484a-ab08-2c2ab46e13ea</ID>
        <AssocObjectId/>
        <X>298</X>
        <Y>189</Y>
        <Width>50</Width>
        <Height>110</Height>
        <Label>Admin</Label>
        <Delete>False</Delete>
        <IsGeneratedAssocObject>False</IsGeneratedAssocObject>
      </Actor>
    </Actors>
    <UseCases>
      <UseCase>
        <ID>da14a3f9-abf2-4261-961d-2ee8f0ad5679</ID>
        <AssocObjectId/>
        <X>503</X>
        <Y>184</Y>
        <Width>128</Width>
        <Height>100</Height>
        <Label>Přidání článku</Label>
        <Delete>False</Delete>
        <IsGeneratedAssocObject>False</IsGeneratedAssocObject>
      </UseCase>
    </UseCases>
    <Lines>
      <Line>
        <ID>dd91ffae-de90-4d28-9e65-b59346daf02c</ID>
        <StartObjectID>b7bb54fb-b1ac-484a-ab08-2c2ab46e13ea</StartObjectID>
        <EndObjectID>da14a3f9-abf2-4261-961d-2ee8f0ad5679</EndObjectID>
        <Type>Association</Type>
        <StartX>348</StartX>
        <StartY>234</StartY>
        <EndX>502</EndX>
        <EndY>234</EndY>
      </Line>
    </Lines>
  </UseCaseDiagram>
  <ClassDiagram>
    <Classes/>
    <Interfaces/>
    <Lines/>
  </ClassDiagram>
</root>
```


Kód 5.1 – XML kód exportovaného projektu

Obrázek 5.7 - Obsah editoru po načtení projektu

5.9 Použité knihovny

Pro vytvoření grafického rozhraní byly použity dvě základní knihovny v Javě pro tvorbu grafiky – AWT a Swing. AWT (Abstract Window Toolkit) je první standartní knihovna používaná pro tvorbu grafického rozhraní v Javě. Je označována jako heavyweight knihovna, protože vykreslování grafických prvků deleguje na operační systém. Takže prvky na rozdílných systémech se mohou trochu lišit. V tom se liší knihovna Swing, která se značí jako lightweight knihovna, protože o veškeré vykreslování se stará sama. Díky tomu výsledný program vypadá na všech systémech stejně, obsahuje více funkcí (nemusí se starat, jestli systém funkcionalitu podporuje). Kvůli tomu však spotřebuje více prostředků při vykreslování. Swingové komponenty dědí od komponent knihovny AWT (viz. obrázek 5.8).



Obrázek 5.8 - Hirearchie grafických knihoven [10]

Pro základní grafické prvky jsou v aplikaci použity komponenty z knihovny Swing. Například JPanel, JDialog, JButton atd. Z knihovny AWT byly použity pouze pomocné třídy jako Point a Dimension,.

6 Testování

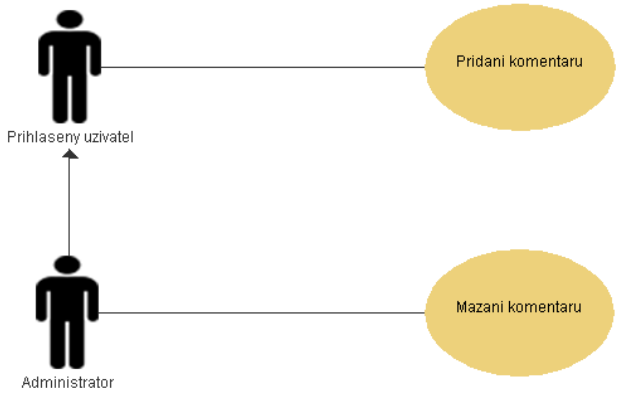
Pro zjištění správné činnosti aplikace a intuitivního pozadí necháme vyzkoušet program další uživatele, kteří budou mít za úkol vytvořit jednoduchý model systému a na základě jejich činnosti a zpětné vazby zjistíme silná a slabá místa editoru.

6.1 Průběh testování

Protože program se zabývá problematikou vývoje systémů, kterým každý nerozumí a testování na takových uživatelích by nemělo velký význam, bude program testován na studentech VUT, kteří jsou s touto problematikou obeznámeni.

Testování probíhalo na dvou skupinkách uživatelů. Nejprve na uživatelích, kteří znají základní principy vývoje systému a jsou obeznámeni s diagramem užití a tříd, ale nikoliv s Petriho sítěmi. Druhá skupinka byli uživatelé, kteří se aktivně zajímají danou problematikou, znají Objektově orientované Petriho sítě a jsou schopni v nich popsat jednoduché třídy.

Všichni testovaní uživatelé viděli aplikaci poprvé. Mají však zkušenosti ve vytváření diagramů v aplikacích podobného typu. První skupina uživatelů bude mít za úkol vytvořit jednoduchý systém o dvou aktérech – administrátor a přihlášený uživatel. Uživatel bude moci přidávat komentáře, administrátor může pracovat se stejnými případy užití jako uživatel a navíc bude moci mazat komentáře (viz obrázek 6.1). Vyzkouší si použití základní asociace objektů i generalizaci pro vytvoření diagramu užití. Tento model bude muset popsat diagramem tříd, vytvořit příslušné třídy, jejich metody a atributy. Druhá skupina měla stejný úkol a navíc musela chování tříd popsat pomocí Objektově orientovaných Petriho sítí.



Obrázek 6.1 - Diagram užití vytvořený uživateli

6.2 Uživatelé se základními znalostmi

Po prvním seznámení s programem se uživatelé rychle zorientovali v nabízených vlastnostech programu a začali vytvářet diagram užití. Všichni uživatelé zvládli vytvořit první diagram během několika minut. To ukazuje intuitivní chování aplikace, její jednoduchost a dobré uživatelské rozhraní.

Při změně záložky na diagram tříd byli uživatelé příjemně překvapeni předem vytvořenými třídami pomocí provázání diagramů. Jediný problém v této části byl při mazání třídy. Uživatel by rád viděl nějakou nápovědu, kde by bylo základní chování aplikace vysvětleno. Na základě poznatku byla nápověda implementována.

Všichni uživatelé zvládli vytvořit oba diagramy bez větších potíží. Líbila se jednoduchost aplikace, která nezatěžuje uživatele zbytečnými prvky, které při vytváření jednoduchého systému nepotřebuje.

6.3 Pokročilí uživatelé

Stejně jako u předchozí skupinky se všichni uživatelé rychle seznámili s aplikací a bez potíží začali pracovat na vývojových diagramech.

Pro zkušené uživatele nebylo vytvoření diagramu užití o dvou aktérech a případů užití žádný problém. První část zvládli téměř ihned. Při vytváření diagramu tříd také nenastaly žádné potíže, uživatelé byli zvyklí z podobných programů mazat objekty po označení pomocí tlačítka delete, takže nenastal tento problém. Jediný problém nastal při pokusu o přepnutí na záložku Petriho sítí. Po přečtení hlášky pochopili funkčnost programu. Při modelování Objektově orientovaných Petriho sítí problém nenastal.

Uživatelé opět zvládli namodelovat zadaný systém bez větších problémů. U Petriho sítí se jim líbila možnost modelovat síť metody a přitom být schopni propojovat jednotlivá místa i se sítí třídy. Bylo navrženo pár typů, jak program vylepšit. Uživatelům by se líbila při kliknutí pravým tlačítkem na objekt kontextová nabídka, která by umožňovala jednotlivé akce s objekty. Dále možnost volnějšího kreslení spojovacích hran. Návrhy na vylepšení mohou sloužit jako budoucí vylepšení programu pro snadnější práci.

6.4 Shrnutí testování

Všichni testovaní uživatelé zvládli zadaný úkol, chválili si propojení jednotlivých diagramů, které jim usnadnilo vývoj, jednoduché ovládání a uživatelské rozhraní aplikace. V tabulce 6.1 můžeme vidět výsledné hodnocení programu uživateli, kteří hodnotili jednotlivé vlastnosti na stupnici od 1 do 10, kde 10 je nejlepší hodnocení a 1 nejhorší. Pomocí zpětné vazby bylo upozorněno na některé nedostatky aplikace, které byly následně opraveny. Byla přidána nápověda programu, kde se popisují základní ovládací prvky, chování aplikace a její využití.

Pokročilým uživatelům se líbila možnost popsání tříd pomocí Objektově orientovaných Petriho sítí.

Vlastnost	Průměrné hodnocení
Vzhled programu	6
Intuitivnost programu	7
Provázání diagramů	8
Tvorba diagramu užití	8
Tvorba diagramu tříd	6
Tvorba Objektově orientovaných Petriho sítí	7

Tabulka 6.1 - Zhodnocení jednotlivých vlastností editoru

7 Závěr

V závěru si popíšeme možná vylepšení aplikace, díky kterým by se stala více uživatelsky přívětivá a mohla by podporovat i další funkce, které nebyly přímým úkolem této práce. Následně si aplikaci zhodnotíme a řekneme si, jestli je použitelná v praxi.

7.1 Vylepšení aplikace

Undo a redo

Při práci s nástroji tohoto typu se stává, že uživatel udělá chybu při práci a chtěl by daný krok vrátit. K tomu by sloužily operace undo a redo (zpět a vpřed). Pomocí jednoho tlačítka následné může vrátit poslední provedenou akci. Na těchto operacích sice aplikace přímo nezávisí, ale uživatelé jsou na ně zvyklé například i z webového prohlížeče a díky nim by byl program uživatelsky více přívětiví.

Simulování Petriho sítí

Po návrhu Petriho sítě by bylo dobré, aby si uživatel mohl danou síť simulovat, aby viděl, jestli je síť správně navržená a jestli se navrhovaný program chová správně. Neexistuje mnoho aplikací, které by umožňovali navrhování Objektově orientovaných Petriho sítí a zároveň možnost jejich simulace. Díky možnosti navržení diagramu užití, diagramu tříd, Petriho sítí a zároveň její simulace by program získal na cennosti a v daný problematice by byl velice přínosný. Další možností by mohl být export Petriho sítí ve speciálním formátu, aby mohla být možnost napojit aplikaci na další program, který by simulaci podporoval (například PNtalk [9]).

Vytvoření zdrojových kódů

Větší aplikace podporují možnost vytvoření zdrojových kódů pomocí diagramu tříd. Vylepšení usnadní vytvoření základní hierarchie aplikace, takže programátor se nebude muset zabývat prací, která se dá dobře zautomatizovat. Program by mohl nabídnout možnost vytvoření zdrojových kódů pomocí diagramu tříd. V nabídce by mohlo být více programovacích jazyků pro univerzálnější použití.

Přidání základních nástrojů pro práci s grafikou

Při práci jsou důležité nejen prvky pro tvorbu diagramů, ale i podpůrné nástroje, které nám pomáhají zjednodušit a zpřehlednit výslednou práci. Jedná se například o nástroj lupa, kreslení vlastních tvarů, možnost vybarvení objektů atd. Díky tomu by program nabídl širší škálu možností práce s diagramy.

[9] http://perchta.fit.vutbr.cz:8000/projekty/12

7.2 Závěrečné zhodnocení

Výsledná aplikace splňuje všechny požadavky pro tvorbu diagramu užití, diagramu tříd a Objektově orientovaných Petriho sítí. Oproti ostatním nástrojům, které se používají k výrobě diagramů, navíc nabízí jejich provázání, které pomáhá k urychlení vývoje vytvořením potřebných objektů v diagramu tříd a Petriho sítí a provázání daných objektů díky vztahům mezi jednotlivými diagramy. Aplikace se snaží o intuitivní ovládání a jednoduchý vzhled, díky čemu není kladena velká náročnost na její ovládání.

Při testování bylo zjištěno, že aplikaci zvládají ovládat i méně zdatní uživatelé, které nemají tolik zkušeností s vývojem softwarových systémů. Aplikace může být využita při vývoji menších a středních aplikací. Lze jí použit i při tvorbě větších projektů, ale v tomto případě by bylo vhodné implementovat některá navrhovaná vylepšení, které by ještě více zjednodušili práci.

Literatura

[1] Formal Models in Software Development and Deployment: A Case Stady. Radek Kočí, Brno University of Technology, Czech Republic. Vladimir Janoušek, Brno University of Technology, Czech Republic, ISBN 987-1-61208-304-9

[2] Janoušek, V. : Modelování objektů Petriho sítěmi. Disertační práce 1998

[3] Martin Hanák: Generování kódu z Objektově orientovaných Petriho sítí, diplomová práce, Brno, FIT VUT v Brně, 2015

[4] UML Tutorial. Tutorials point. [online]. [cit. 19.2.2016]. Dostupné z: http://www.tutorialspoint.com/uml/

[5] Vaughn Vernon: Software crafts. *Vaughn Vernon.* [online]. [cit. 19.2.2016]. Dostupné z: *https://vaughnvernon.co/?page_id=31*

[6] CREATELY. *Creately Class diagram example* [obrázek]. [cit. 22.4.2016]. Dostupný na WWW: http://creately.com/diagram/example/iliw20n31/Point+of+Sales+System+%28POS%29+-+Class+Diagram

[7] ARGOUML. *ArgoUML Class diagram* [online]. [cit. 1.5.2016]. Dostupný na WWW: http://argouml-stats.stage.tigris.org/documentation/manual-0.24/images/reference/class_diagram.gif

[8] Office Blogs, Professional, flexible & beatiful UML content [obrázek]. [cit 1.5.2016]. Dostupný na WWW: https://blogs.office.com/wp-content/uploads/migrated-images/40/5557.UMLClass.PNG

[9] CREATELY. The complete Guide to UML Diagram Types with Examples [obrázek]. [cit. 1.5.2016]. Dostupný na WWW: http://static3.creately.com/blog/wp-content/uploads/2012/02/UML-Diagram-types1.png

[10] Wikipedia. Swing (Java) [obrázek]. [cit. 7.5.2016]. Dostupný na WWW: https://cs.wikipedia.org/wiki/Swing_(Java)#/media/File:AWTSwingClassHierarchy.png

Příloha A

Obsah CD

- Application – obsahuje spustitelnou aplikaci
- Examples – obsahuje příklady jednotlivých diagramů, které lze importovat do aplikace
- Source – zdrojové kódy programu
- Bachelors thesis – teoretická část bakalářské práce
- README – textový soubor popisující základní strukturu CD