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 SOFTWAROVÝCH SYSTÉMŮ

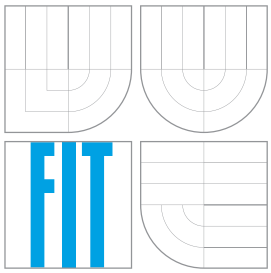
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

KAREL HALA

BRNO 2013

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 SOFTWAROVÝCH SYSTÉMŮ

TOOL FOR SOFTWARE SYSTEMS DESIGN

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — KAREL HALA
AUTHOR

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

BRNO 2013

Abstrakt

Nástroj pro podporu vývoje softwarových systému je aplikace sloužící pro vizuální znázornění průběhu vývoje aplikace. Slouží k porozumění zákazníkovým požadavkům, připravení návrhu a rozvržení jednotlivých tříd. Aplikace má za cíl ulehčit rozvržení práce před její implementací a pomoci uživateli odhalit některé chyby, které by mohly nastat během implementace.

Abstract

Tool for software systems design is application for visualization of development application. It's main gola is to achieve connection between developer and customer. It should be used to understand customer's needs, prepare workflow of project and understanding each class. This application should make easier to understand some flaws, that might occure when creating software.

Klíčová slova

Případ užití, Diagram tříd, Nástroj, Softwarový vývoj

Keywords

Use Case, Class diagram, Tool, Software development

Citace

Karel Hala: Nástroj pro podporu vývoje softwarových systémů, bakalářská práce, Brno, FIT VUT v Brně, 2013

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
5. května 2014

Poděkování

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

Obsah

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 vyvýjený software pomocí různých grafů. Rozebereme jejich přednosti a jejich slabiny. Poté si vybemere 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 tojici diagramů **Diagram případů užití, Diagram tříd, Objektově orientované Petriho sítě**. Provázání mezi těmito diagramy poté vysvětlíme v kapitole 3.5.

1.1 Seznámení s nástrojem

Kapitola 2

Zhondocení konkurence

Z hlediska snažšího porozumění byl brán zřetel na konkurentní 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** [1] aplikace primárně určena pro Unix systémy, nicméně je možné jej bez problémů nainstalovat také na Windows za pomoci KDE installeru a vybrání požadovaného balíčku. Ovládání tohoto installeru, je velice jednoduché a intuitivní.

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

2.1.1 Popis aplikace

Aplikace je velice robustní a poskytuje mnoho vlastností. Při ohlédnutí na to, že je volně ke stažení a volně k používání [2] aplikace nabízí velké množství vlastností a možností. Umbrello poskytuje mnoho UML grafů a práce s touto aplikací je poměrně jednoduchá, ale její jednoduchý design má pár neduhů. 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.

2.1.2 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.

[1] Domovská stránka aplikace: http://umbrello.kde.org/

[2] 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.

- **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 uložiště a mašiny, 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 uživatele provede jednoduchým nastavením, výběrem které třídy namodelované v aplikaci se mají vygenerovat a nabídne také možnost upravit mírně zvolený kód. Vzhledem na to, že aplikace je plně zdarma je tato vlastnost velice přínosná.

Umbrello dále poskytuje možnost částečné generace i v případě, že když nejsou poskytnu komentáře na základě kterých by tato dokumentace mohla být vytvořena. Při generování tříd je možnost, 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 na používání a intuitivní, nicméně dlouhodobější práce je zdlouhavá a příliš repetitvní. Nutnost vytvoření objektu na pracovním 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.

2.1.3 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 za potřebí ve výsledné aplikaci mnoho náročných grafických elementů, tyto elementy by byly rušivé. Grafické rozložení a dosažitelnost většiny editačních prvků je v aplikaci Umbrello je velice jednoduché a intuitvní z tohoto důvodu bylo ve výsledné aplikaci přihlíženo na tyto vlastnosti.

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

V Umbrello nelze dané spoje ohýbat, lze dané spoje ohýbat, ale tato funkcionalita je velice špatně navrhnutá. Dále není možno v Umbrello nikterak editovat dané spoje, tomuto bylo vyhnuto a přidána editace spojů s možností jednak změny typu, dále také změny objektu na který daná hrana ukazuje a také možnost jednoduchého odebrání hrany.

K nepřehlednosti aplikaci Umbrello přispívá také skutečnost, že není vidět hrany vycházející z a nebo do daného objektu. K zpřehlednění výsledného diagramu bylo proto navrhnuto zvýraznění těchto hran. Což vedlo jednak k spřehlednění a také k lepšímu cítění a použitelnosti.

Aplikace Umbrello při přidání nového objektu donutí uživatele zadat jméno, což opět vede ke zdlouhavé práci a repetetivním 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.

2.1.4 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.

2.2 Enterprise Architect

S ohlédnutím na Aplikaci Umbrello a některé její nedostatky, bylo nutné pro pozdější vývoj aplikace prozkoumat další systém. Tato aplikace se jmenuje Enterprise Architect a je tvořena firmou **Sparx Systems** [3], tato firma se soustřeďuje na tvorbu nástrojů pro tvorbu UML grafů. Bohužel 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 (dále jen EA) je velice robustní systém, již nemá takové nedostatky jako aplikace Umbrello a nabízí několikero možností pro refaktoring projektu. Aplikace rozděluje UML grafy do dvou skupin, skupina

- **Structural Diagrams** pro diagramy na popis vyvýjené 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]

[3] http://www.sparxsystems.com

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 bohužel schována a je náročné ji najít, jako například přidání a editace atributů třídy.

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, toto lze také provést importem binárního souboru, například `.jar` souboru a aplikace se může postarat o vygenerování jak dokumentace, tak výsledného diagramu tříd.

Pro jednodušší a rychlejší možnost spojení jednotlivých objektů na pracovní ploše, lze použít rychlého nástroje vedle objektu. Na stejném místě lze nalést také funkce pro ryhclou 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. Při kliknutí mimo jakýkoliv objekt spoj zůstane u původního objektu. [1]

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 za potřebí možnosti jednoduchého zalomení hran a možnosti jejich jednoduché editace. Taktéž přesun po pracovním plátně je v aplikace 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 poněkuk 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 na to, 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 svižnejší práci s výslednou aplikací.

Grafický prvek celé aplikace bude za potřebí jednoduchý a snadný k používání, ale důležitým prvkem je dobré zviditelnění některých objektů. Jako například znázornění tříd, které nemají odpovídající prvky v případu 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 Diagram případů užití 3.2 pro popis jednotlivých akcí v systému, Diagram tříd 3.3 pro grafické znázornění tříd a operace mezi nimi. A jako poslední část, která nespadá 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 objěktově orientované programování. Toto je speciální styl programování používaný převážně při vývojí větších a náročnějčích aplikací.

3.1.1 části objěktově orientovaného programování

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

3.1.2 Objekt

Základ takovéhoto programování je samozřejmě objekt, je to specifický pohled na fyzikální a konceptální jednotky reálného světa.

- 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. Zaslaná 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. [3]

3.1.3 Instanciace, třídy

Ve většině programovacích jazyků se vytváření objektů provádí specílní 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 spustění programu a zanikají až při jeho ukončení. Dynamicky instanciované proměnné vznikají a zanikají v průběhu chodu aplikace na požadavky uživatele. [3]

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 implementování částečně, nebo úplně. A v neposlední řadě lze rozlišit dědění násobné a jednoduché podle počtu dědění. [3]

3.2 Diagram případů užití

Diagram 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.

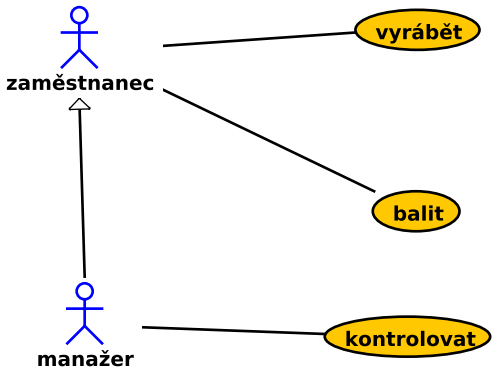
Základními prvky Diagramu případů užití jsou případy užití a aktéři. V požadavcích na systém rozpoznáváme takzvané „Stakeholdery“ jsou to osoby, nebo věci, které mají zájem na úspěšné provedení systému. Případy užití popisují jak se má systém zachovat na reakce stakeholderů, akterů, a dodat určité výsledky. 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 „Podpůrný aktér“. Aktér není přímo chápán jako jedinec, nýbrž jako role v systému. [6]

3.2.1 Rozbor Diagramu připadů 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 možný 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 vyjmuto 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 funckinalitu 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.

3.2.2 Použité vlastnosti

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

Pokud případu užití, nebo aktér nemá odpovídající třídu v Diagramu tříd je toto znázorněno pod případem užití, nebo po akterém a to formou textu „No Class“. Pokud daný spoj mezi objekty nemá odpovídající spoj mezi třídamy je toto znázorněno odlišnou barvou od normálních spojů. Dále pokud objekt není nijak spojen s ostatními objekty je znázorněn přerušovaným okrajem.

Vybraný objekt je znázorněn odlišnou barvou a jeho okolí je ohraničeno rámečkem. Pokud je ojekt 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 zpřehlednění 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 kominakci 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 obdélínků obsahující další informace o dané třídě. V horní části je název třídy. Poté třída obsahuje instační proměnné, attributy. 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 attributy a operace, ale nenachází se v takových třídách implementace těchto operací. Tato implementace se nachází až ve třídách, které implementují takový interface.

Zápis attributů 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 attributů a operací pomocí identifikátoru. **Private** označované „-“, **protected** jako „#“ a **public** znázorněné pomocí „+“. Tyto identifikátory se píší před jméno attributu, nebo operace. [5]

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

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 bílím trojúhelníkem na konci, 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™ několikanásobná dědičnost není povolena, ale je možné použít **interface**. [5]

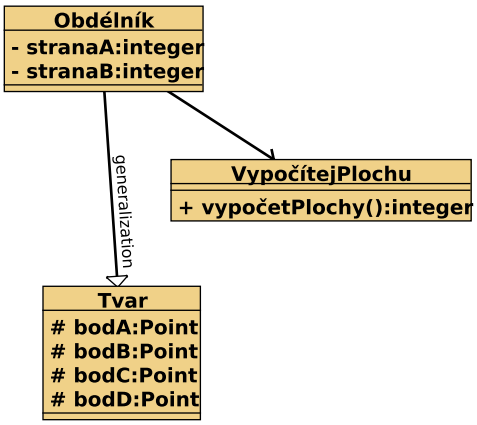
Asociační hrana

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

Asociace je kreslena v UML pomocí hrany s oběma konci opatřenými šipkou. Takovýto vztah se jmenuje **bidirectional** a znamená, že obě třídy se používají navzájem, je možné také použit **unidirectional** (častěji používaný) spoj, což znamená že třída, ze které šipka vede je použita třídou do které šipka ukazuje, ale ne naopak. Dále je možné tyto hrany opatřit násobností, neboli označením kolik objektů je použito kolika objekty. [5]

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



Obrázek 3.2: Příklad Diagramu tříd.

agregace. Tato hrana je kreslena souvislou hranou opatřenou bílým diamantem na konci. [5]

Kompoziční hrana

Tato hrana je z části stejná, jako hrana Agregační, 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 odtraněn objekt třídy B. Tato hrana je kreslena souvislou hranou opatřenou černým diamantem na konci. [5]

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 attributy třídy B a přetíží operace třídy B, které implementuje. Hrana je kreslena přerušovanou šipkou ve směru od implementovaně třídy k interfacu. [5]

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 bjektu 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 případe interface a None třídy jsou tyto názvy skryty.

Podobně jako tomu je v případe Diagramu případů užití i zde je názorně vidět, zda některá hrana vede z a nebo do třídy a to tak, že třída bez jakýhkoliv 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 připadů 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 paralerních 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**, $P_N \bigcap T_N = \emptyset$.
- $PI_N : P_N \to \mathbb{N}$ **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. $RECOND_t^N : PN \to \mathbb{N}$ jsou **vstupní podmínky** přechodu.
  2. $POSTCOND_t^N : P_N \to \mathbb{N}$ 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 jednotlivímy 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 splněný zanesených pravidel se okamžitě provedou. Nebo mohou být časované, což znamená že po určitém čase se daný přechod provede. Toto je opět velice jednoduché rošíření základních Petriho sítí.

úspěšné provádění přehodů probíhá v pořadí odebrání značek ze vstupního místa, přídání značek na výstupní místo.

Dále můžeme sítě rozšířit o jednoduché oprioritizování spojení mezi místy a přechody, nebo vložením procentuální pravděpodobností provedení daného spoje. [3]

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 organitována podle dědičnosti. Každá z těchto tříd obshauje tyto prvky:

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

Sítě metod mohou sdílet attributy objektu. Mají navíc parametrová místa, do kterých se při invokaci takové sítě vloží parametry ze zprávy, dále mají výstupní místo **return**, coč 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 třídy.

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

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

$$y := x_0.msg(x_1, \ldots, x_n) \tag{3.1}$$

kde y je proměnná, x_i je term a msg je jméno zprávy. Objekt se jménem x_0 je adresátem zaslané 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. [3]

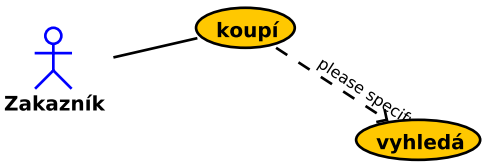
3.4.3 Grafické znázornění

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

Obrázek 3.3: Příklad Objektově orientované petriho sítě pro metodu.

3.5 Spojení diagramů

Během vytvření 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.



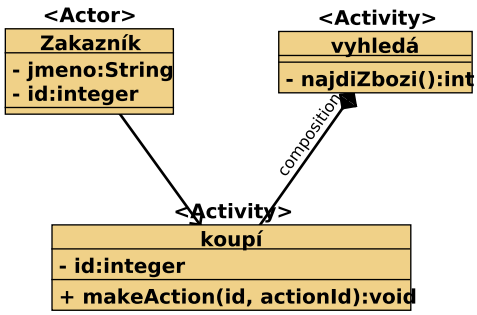
Obrázek 3.4: Nástin problému případů užití.

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 orientovém analytikovi. Ten by se měl postarat o další převod a upřesnění těchto analytických tříd.

Analytické třídy by měli obsaovat pouze ty nejdůležitější atributy, operace jsou opět klíčové služby, které musí třída poskytovat. Často se může stát že operace je poté rozbita do několika menších metod. [4]

3.5.1 Použití ve výsledné aplikaci

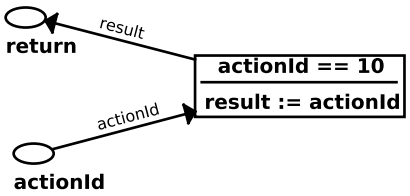
Provázání mezi UML částmi prbíhá na velice nizké úrovni a to taj, že při vytvoření objektu v diagramu případů užití vznikne třída se stejným názvem v diagramu tříd. Na orientovaném analytikovi potom je potřeba zadat jednotlivé atributy a operace. Pokud vznikne případ, že se analytivká třída rozpadne 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 sít a podobně tomu tak je v případě třídy. Objektově orientovaná Petriho sít pro metodu má jedno specifické místo **return**, které nelze odstranit ani změnit její jméno. [3]

V každé Objektově orientované Petriho síti se nachází attributy. V případě Objektově orientované Petriho síti pro třídu jsou to třídní proměnné dané třídy a všech rodičů, které mají viditelnost **protected**, nebo **public**. V případě Objektově orientovaných Petriho sítí pro třídy dále tato sít obsahuje attributy, které jsou vstupními proměnnými dané metody. [7]



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 zpřehlednění 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 kouknurentních 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.

4.0.1 Popis aplikace

Na základě poznatků z ostatních aplikací bylo rozhodnuto o jednoduchosti a intuitivním ovládání aplikace. Tohoto bylo docíleno sjednocením základního designu aplikace, sjednocení klávesových zkratek a porozumění a prozkoumání moderních aplikací.

Hlavní okno se skládá z několika panelů. Horní část obsahuje otevřené projekty, jejich částí a ovládací prvky. Celý hlavní obsah aplikace se nacháazí ve zbylích dvou třetinách okna. V levé části se dále nacházejí tlačítka pro možnost přepínání jednotlivých vkládaných obejktů. Dolní část obsahuje informační a editační možnosti zvoleného prvku a pro hlavní část, je vyhraněn 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. Bylo zvoleno jednoduché a intuitivní chování aplikace, po kliknutí se vytvoří objekt, po tažení se přesune objekt a pro spojení dvou objektů hranou je zapotřebí kliknout na první objekt a poté na druhý, přičemž spoj je viditelný od kurzoru po první objekt. Pro zahnutí spoje je možno kliknout pravým tlačítkem myši, což vyvolá vytvoření ohybu spoje.

Kapitola 5

Závěr

Literatura

[1] *Enterprise Architect User Guide*. Accessed: 12.04.2014.

[2] *Umbrello UML Modeller Handbook*. Accessed: 12.04.2014.

[3] janoušek, V.: *Modelování objektů petriho sítěmi*. Dizertační práce, 2008.

[4] Jim Arlow, I. N.: *UML2 a unifikovaný proces vývoje aplikací*. Computer Press, a.s., 2011, iSBN-978-80-251-1503-9.

[5] Kettenis, J.: *Getting Started With UML Class Modeling*. Oracle, May 2007, accessed: 22.03.2014.

[6] Kettenis, J.: *Getting Started With Use Case Modeling*. Oracle, May 2007, accessed: 22.03.2014.

[7] 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

Příloha B

Manual