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

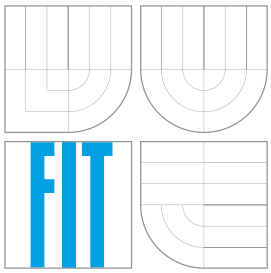
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

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
25. března 2014

Poděkování

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

© Karel Hala, 2013.
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

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.

Nástroj tohoto docílí za použití několika částí. Případu užití [7] pro jednotlivé znázornění zákazníkových potřeb, diagramu tříd pro vlastní vizualizaci a pochopení vyvýjeného softwaru a nakonec objektově orientované petriho sítě, sloužící pro porozumění celého systému do hloubky.

1.1 Seznámení s nástrojem

Jak bylo již řečeno pro porozumění požadavků zákazníka nám poslouží diagram případu užití, odsud se předgenerují jednotlivé třídy a ty následně bude možné upravit a popsat v části objektově orientovaných petriho sítí.

Kapitola 2

Zhondocení konkurence

Z hlediska snažšího porozumění byl brán zřetel na konkurentní nástroje. Převážně byl brán zřetel na jejich jednoduchost a co je dělá tak hojně využívané a snadno použitelné. 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** [3] aplikace primárně určena pro Unix based 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 její hlavní výhody 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 mnohov lastností. Při ohlédnutí na to, že je volně ke stažení a volně k používání, aplikace používá licenci GNU general public license [2]. Což umožňuje komunitu být zapojenou při vývoji, odchytávat chyby a sami je opravovat. Pokud vývojář nicméně chce být zahrnut do vývoje, musí bý členem KDE komunity. Umbrello poskytuje mnoho UML grafů a práce s touto aplikací je poměrně jednoduchá, ale její jednoduchý design má mnoho špatných návrhů. Pro menší aplikace a pro začínající firmy je naprosto dokonalá, ale absence pokročilého editoru a ne příliš jednoduché práce s editorem je opravdu žalostná.

2.1.2 Výhody a nevýhody

Umbrello aplikace poskytuje několikero UML grafů. **Class Diagram**, pro popis tříd a provázanosti mezi nimi. **Sequence Diagram**, pro popis chodu aplikace, komunikace mezi vlákny a komunikace mezi aplikacemi. **Use Case Diagram** pro popis akcí v systému, aplikaci. **State Diagram** pro popis stavu systému a průběhu jednotlivých operací, převážně zachování v určitých momentech aplikace na dané podmínky. **Activity Diagram** popisuje akce a průběh celého programu graficky. **Component Diagram** pro popis spojení a komunikace mezi jednotlivými komponentami. **Deployment Diagram** popisuje fyzické uložiště

a mašiny, na kterých program bude pracovat. A v neposlední řadě **Entity Relationship Diagram** pro popis vztahu dat mezi sebou.

Jak je vidno Umbrello poskytuje velké množství editorů 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 genraci a refactoring zdrojového kódu pro některé jazyky. Stačí pouze importovat jednu již vytvořenou třídu a Umbrello vás provede jednoduchým nastavením, výběrem které třídy namodelované v aplikaci se mají vygenerovat a nabídne také možnost upravit mírně zvolený kód. Celkově funkce pro generování zdrojového kódu a jeho úprava je velice potěšující a s ohlédnutím na to, že Umbrello je volně šířitelné, je tato vlastnost velice dobrá.

Umbrello dále poskytuje možnost gerevání částečné dokumentace, i když nejsou poskytnu komentáře. 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`. Výběr je tedy více než bohatý.

Dále Umbrello poskytuje možnost exportování objektů jako PNG obrázky. Tato funkčnost je velice příjemná při psaní dokumentů stačí část aplikace jednoduše okopírovat a vložit jako obrázek. Exportovat jako obrázek, je také možno celý diagram. A také tisknout daný diagram.

hlavní nevýhoda aplikace je pravděpodobně ne příliš dobře udělaný editor. Editor je sice jednoduchý 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 zdlouhavé a špatné.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.

Co je ale největší slabinou Umbrello aplikace, je nevalidace daného diagramu. Je čistě na uživateli hlídat si veškeré chyby a validitu modelované aplikace.

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 se naopak hodí. Z aplikace Umbrello bylo především použita jednoduchost a intuitivní ovládání. Rozdělení aplikace na jednotlivé části a poté jednoduchý grafický design. Není za potřebí ve výsledné aplikaci mnoho náročných grafických elementů. Tyto elemnty by byly rušivé, tudíž na základě Umbrello aplikace byl zvolen jednoduchý design.

Na základě chyb a nedostatků, převážně tomu, že Umbrello neposkutuje jakoukoliv kontrolu daného modelu, bylo v aplikaci Nástroj pro podporu vývoje softwarových systémů implementována možnost kontroly editovaného grafu. A to přesněji kontrola spojů, zda je daný spoj mezi dvěma objekty na pracovním plátně validní.

V Umbrello nelze dané spoje nikterat ohýbat, což dělá výdledný diagram velice nepřehledný. 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.

V Umbrello není nikde vidět, zda daný objekt má hrany, které vedou do a nebo z obejktu. Toto bylo ve výsledné aplikaci přidáno, což následně změnilo grafické cítění modelu

a jednoduchou identifikaci objektů.

Ve výsledné aplikaci, bylo také vyhnuto vlastnosti automatického vyskakování okna při přidání nového objektu, toto bylo nahrazeno pozdější úpravou daného objektu. Bylo vzáno v potaz, že uživatel bude chtít v jeden moment vytvořit více objektů a až poté e pojmenovat a pospojovat.

2.2 Enterprise Architect

Aplikace [1] je tvořena firmou sparx system `http://www.sparxsystems.com`, tato firma se soustřeďuje na tvorbu nástrojů pro tvorbu UML grafů. Bohužel firmou nabízené produkty jsou placené, ale za dané služby člověk obdrží opravdu robustní systém pro tvorbu UML grafů a možnost refaktoringu pro několik jazyků.

2.2.1 Popis aplikace

Enterprise Architect (dále jen EA) je velice roustní 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. A skupinu **Behavior Diagrams** pro popis chování aplikace a komunikace mezi jednotlivými částmi aplikace. Aplikace poskytuje velkou škálu grafů a diagramů od klasických diagramů pro popis chování dat mezi sebou, přes diagramy pro návrh Win32 UI až po návrh testování výsledné aplikace.

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 intuitivní a jednoduché pro používání. Většina pokročilých funkcí je bohužel schována a je náročné ji najít. Přidávání atributů pro jednotlivé třídy je opět zdlouhavé a neintuitivní.

Výhodou této aplikace jistě je možnost vytváření diagramů pro jednotlivé jazyky, celkem jedenáct jazyků je podporovaných. `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. Pro import aplikace je možné použít také binární soubor výsledného programu. Generování dokumentace je jednoduché a přehledné.

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.

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 ve výsledné aplikaci je vyřešen elegantněji a intuitivněji.

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. Tudíž bylo ve výsledné aplikaci zvoleno umístnění otevřených oken se soubory v horní části aplikace, lépe přístupné uživateli.

Taktéž diagramy a grafy, se kterými se pracuje v danou chvíli mají umístnění v EA své místo, ale pro vyvýjený projekt naprosto nepoužitelné. V EA jsou diagramy umístněny ve stromové struktuře, z pochopitelných důvodů rozsáhlosti aplikace a možnostem grafů. Pro pozdější vývoj bylo zvoleno umístnění otevřených diagramů pod záložkami souborrů. Takto má uživatel přepínání mezi částmi aplikace velice pohodlně umístněné a jednoduché pro přepínání.

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 ve výsledné aplikaci vzhledem na to, že aplikace se soustředí čistě jenom na vytváření a editaci diagramů a UML grafů.

Během testování nebylo v žádné aplikaci možno udělat Objektově orientované Petriho sítě, nebo jakýkoliv jiný rozšířený pohled na implementaci jednotlivých tříd a komunikaci mezi nimi. Dále v testovaných aplikacích nebylo možné předgenerovat si třídy na základě Případu užití, nebo naopak editovat tyto třídy popřípadě smazat.

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. Editace hran byla ve většině editorů nepříliš kvalitní. Změna typu hrany a změna objektů pro danou hranu je zapotřebí pro jednodušší práci s editorem.

Dále chyběla jakákoliv provázanost mezi jednotlivými částmi aplikace, vytvoření několika případů užití nevedlo k předvytvoření tříd a naopak. Pokud ovšem chyběl některý objekt v připadu užití a nebo třída v diagramu tříd nebylo toto opět viditelné. Tyto vlastnosti byli implementovány a znázorněny, takže uživatel je schopný upravit jednotlivé části a velice jednoduše spozorovat chybějící, nebo neprovázané objekty.

Kapitola 3

Grafy pro popis částí

Pro grafické znázornění jednotlivých částí byly vybrány z UML grafů Diagram případů užití 3.1 pro popis jednotlivých akcí v systému, Diagram tříd 3.2 pro grafické znázornění tříd a operace mezi nimi. A jako poslední část, která nespadá pod UML bylo vybráno objektově orientovaných petriho sítí 3.3 pro popis chování jednotlivých tříd.

3.1 Případ užití

Případy 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 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 veci 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. [7]

3.1.1 Diagram 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 „includeq“. Spoj je veden z nadřazeného případu užití k nižšímu případu. [7]

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 *hlavnho* případu užití. [7]

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. [7]

Pro lepší znázornění a oddělení případů užití je vhodné použít organizaci, ta se znázorňuje rámečkem okolo několika případů užití.

3.1.2 Pužité 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 Diagram případu užití 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. Tyto kontroly je ovšem možné vypnout.

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 nějakými jinými objekty tyto spoje jsou znázorněny po vybrání tohoto objektu. Zvýrazněné spoje jsou pouze ty spoje, které vedou z vybraného objektu.

3.2 Diagram tříd

Rozdíl mezi třídou a objektem je ten, že třída je předpis pro objekt. Objekt je datová reprezentace předepsaných dat a práce s těmito datami. V UML jsou třídy popsány poomcí diagramu tříd a tyto diagramy se skládají z attributů, předpis pro formát dat. Dále se skládají z metod, předpis pro práci s daty a relací mezi třídami. [6]

3.2.1 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, první písmeno v třídě by mělo být velké. Poté třída obsahuje attributy, jednotlivá slova neoddělujeme podtržítky, ale velkým písmenem. A poslední část tříd jsou operace, metody. Pro operace platí stejné pravidla jak pro attributy. Omezení v názvosloví je čistě doporučení a ne nutnost.[6]

Určitou podmnožinou třídy jsou abstraktní třídy, tyto třídy nemohou být incializovány, 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. [6]

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. [6]

Zápis attributů a operací tříd je stanove, 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 attribut, nebo operaci. [6]

Jednotlivé hrany pro spojení se dělí na **Generalization**, **Association**, **Aggregation**, **Composition** a **Implementation**.

Generalization hrana

Hrana **Generalization** 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**. [6]

V UML je tato hrana popsána šipkou ukončenou bílím trojúhelníkem na konci, je vedena směrem od generické třídy k více specifické. [6]

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**. [6]

Association hrana

Vztah mezi třídami určující, že objekty jedné třídy jsou používány v objektech druhé třídy. [6]

Association je kresleno 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** spoj, což znamená že třída, ze které šipka vede je použita třídou do které šipka ukazuje, ale ne naopak. [6]

Dále jsou tyto hrany opatřeny násobností, neboli označením kolik objektů je použito kolika objekty.

Aggregation hrana

Použití spoje Aggregation značí, že určitý objekt potřebuje, ke svému chodu další objekt. Pokud ovšem přestane existovat původní objekt, další objekt lze nadále v systému použít. To znamená, že objekt třídy A potřebuje ke svému chodu objekt třídy B. V momentě odstranění objektu třídy A ze systému, ale objekt třídy B nezaniká. Tomuto se říká slabá agregace. [6]

Tato hrana je kreslena souvislou hranou opatřenou bílým diamantem na konci. [6]

Composition hrana

Tato hrana je z části stejná, jako hrana Aggregation, 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. [6]

Tato hrana je kreslena souvislou hranou opatřenou černým diamantem na konci. [6]

Implementation 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. [6]

Hrana je kreslena přerušovanou šipkou ve směru od implementovaně třídy k interfacu.

3.2.2 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.

Třídy byli rozděleny na tři druhy. Třída **Actor**, která odpovídá objektu aktér v diagramu případů užití. Třída **Activity**, tato třída odpovídá případu užití. A třída, bez jakékoliv role znázorňující ostatní podpůrné třídy. Název druhu třídy se nachází nad danou třídou a u základní třídy, tedy třídy bez jakékoliv role, je tento název skryt.

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 je třída ohraničena čárkovaným orámováním.

Zvýrazněná třída je oddělena od ostatních odlišnou barvou orámování a hrany, které vedou z 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. Tyto kontroly je možné vypnout.

3.3 Objektově orientované Petriho sítě

3.3.1 Objektově orientované programování

3.3.2 Petriho sítě

3.3.3 Objektově orientované Petriho sítě

[8] [5] [4]

3.3.4 Přidané prvky

Kapitola 4

Popis aplikace

Jak již bylo řečeno v předešlé části, 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.

4.1 Editace a chování prvků

4.2 Editace a chování spojů

Kapitola 5

Závěr

Závěrečná kapitola obsahuje zhodnocení dosažených výsledků se zvlášť vyznačeným vlastním přínosem studenta. Povinně se zde objeví i zhodnocení z pohledu dalšího vývoje projektu, student uvede náměty vycházející ze zkušeností s řešeným projektem a uvede rovněž návaznosti na právě dokončené projekty.

Literatura

[1] Enterprise architect. http://www.sparxsystems.com/products/ea/, accessed: 22.03.2013.

[2] Gnu General public license v 2.0. http://www.gnu.org/licenses/old-licenses/gpl-2.0.html, accessed: 23.03.2013.

[3] Umbrello Project. http://umbrello.kde.org/, accessed: 22.03.2013.

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

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

[6] Kettenis, J.: *Getting Started With UML Class Modeling*. Oracle, May 2007, accessed: 22.03.2013.

[7] Kettenis, J.: *Getting Started With Use Case Modeling*. Oracle, May 2007, accessed: 22.03.2013.

[8] Radek Kočí, Vladimír Janoušek, and František Zbořil, jr. : Object Oriented Petri Nets – Modelling Techniques Case Study. In *International Journal of Simulation Systems, Science and Technology*, *3*, ročník 10, May 2009.

Příloha A

Obsah CD

Příloha B

Manual