PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

BAKALÁŘSKÁ PRÁCE

Diagramy chovaní jazyka UML

2014 Pavel Zehnula

Anotace

Práce popisuje tvorbu jednoduchého editoru UML diagramů. V editoru je možné vytvořit diagram tříd, stavový diagram, sekvenční diagram a diagram spolupráce. Editor umožňuje ukládat diagramy do XML a z diagramu tříd je schopen pomocí jednoduchého algoritmu vytvořit základ diagramu spolupráce, nebo základ sekvenčního diagramu.

Chtěl bych poděkovat svému vedoucímu, panu RNDr. Arnoštu Večerkovi, za vedení a konstruktivní připomínky k mé práci.

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

První část práce tvoří popis digramů grafického jazyka UML, který slouží pro vizializaci, specifikaci, navrhování a dokumentaci programových systémů. Tento jazyk může usnadnit návrh a vývoj informačního systému. V další části práce jsou předsatveny již existující editory pro práci s UML.

Ve druhé části je popsán jednoduchý editor pro tvorbu a editaci vybraných diagramů jazyka UML. Základním diagramem je diagram tříd. Po vytvoření nebo načtení toho diagramu bude uživatel moci informace, jako například informace o třídách, operacích, atributech, použít k tvorbě digramu sekvenčního nebo diagramu spolupráce.

2. Jazyk UML

S rozšiřujícím se používáním objektově orientovaného programování bylo nutné tyto vztahy mezi jednotlivými objekty nějakým způsobem zachytit a znázornit. Vzniklo mnoho metodik a způsobů. V průběhu 90. let se se podařilo sjednotit tyto způsoby a vznikl standard UML. Také vzniklo mezinárodní konsorcium OMG (Object Management Group), které na specifikace UML dohlíží. Tato práce se zabývá částí toho standardu a tou jsou diagramy. Konkrétně se bude jednat o diagram tříd, stavový digram, diagram spolupráce a sekvenční diagram.

2.1. Diagram tříd

Diagram tříd patří do skupiny strukturálních diagramů. Představuje náhled na modelovaný systém a jeho hlavním úkolem je zobrazit strukturu navrhovaného systému. Znázorňuje typy objektů nacházející se v navrhovaném systému a jejich vzájemné vztahy. Tvorba tohoto diagramu patří k jedné z prvních a základních fází vývoje softwaru. Právě díky tomu, že diagram zachycuje celou strukturu systému, je velmi užitečný i pro zpětnou úpravu systému.

Při tvorbě diagramu je nutné určit jeho hlavní účel. Existují 3 úrovně diagramu tříd, a to konceptuální model, designový model a implementační model.

Konceptuální model je používán k prvotnímu návrhu softwaru. Obsahuje pouze třídy charakterizující hlavní a nejproblematičtější část aplikace, tzv. byznys třídy (busines classes). Při tvorbě uzlů se většinou vypisují jen nazvy tříd, popřípadě duležité atributy či metody těchto tříd. Hlavní část tohoto typu diagramu tvoří relace, které charakterizují vztahy mezi těmito stěžejními částmi aplikace.

Designový model (model návrhu) navazuje, rozšiřujea zpřesňuje předchozí konceptuální model. Přidává do modelu informace o datových typech atributů a metod a jejich viditelnosti. Model je doplněn o třídy strající se o uživatelské rozhraní (presentation classes) a systémové události (control classes).

Implementační model již obsahuje veškeré implementační charakteristiky daného softwaru. V řadě programů pro tvorbu těchto diagramů je možné nechat si vygenerovat kód konkrétního jazyka.

2.1.1. Asociace

Jak již bylo řečeno, diagram je tvořen uzly a vazbami mezi těmito uzly. Asociace patří mezi základní vazby diagramu tříd. Jedná se vztah mezi dvěma uzly, které mohou existovat nezávisle na sobě. Asociace udává, že uzly jsou schopny spolu komunikovat (uchovávájí odkazy na sebe). Můžeme použít obousměrnou asociaci, kde o sobě ví oba uzly, nebo použít jednosměrnou, kde jeden z uzlů nemá o druhém tušení. Asociace se v diagramu znázorňuje plnou čarou, případně jednosměrnou se šipkou v příslušném směru.

2.1.2. Generalizace

Generalizace, neboli dědičnost, je jednou ze základních vlastností a výhod objektového orientovaného programování. Jedná se o vztah potomek-předek. V předku definujeme obecné chování a vlastnosti pro všechny potomky, kteří již nemusí toto chování definovat. Tato vlastnost značným způsobem šetří čas a udržuje program přehledným. V diagramu je dědičnost znázorněna plnou šipkou od potomka k předkovi.

2.1.3. Agregace

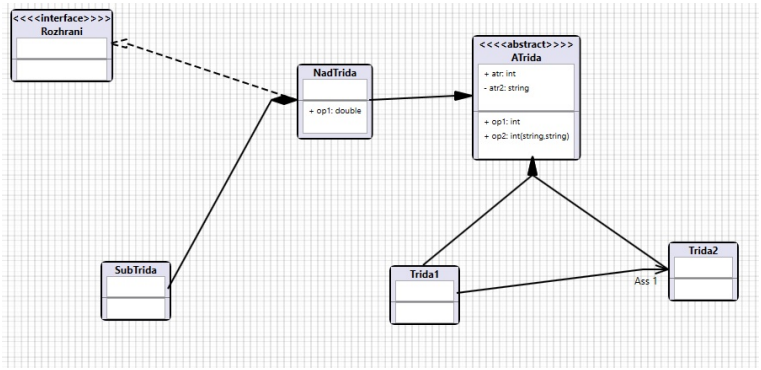
Agregace složí k znázonění vazby část-celek. Ve většině případů je celek určitým druhem kolekce osahující objekty určitého typu. V diagramu je agregace znázorněna plnou čarou s prázdným kosočtvercem na straně uzlu, který znázorňuje celek.

2.1.4. Kompozice

Kompozice je silnější vztah než v případě agregace. Stejně jako v případě agregace se jedná o vztah část-celek. Narozdíl od agregace, ale části nemohou existovat mimo celek. V diagramu je kompozice znázorněna plnou čarou s plným kosočtvercem na straně uzlu, který znázorňuje celek.

2.1.5. Realizace

Poslední vazbou, kterou popíšeme, je realizace. Tato vazba se používa až v implementačním modelu. Popisuje vztah mezi rozhraním, které je v diagramu znázorněno jako uzel s tak zvaným stereotypem. Stereotyp je uveden ve špičatých závorkách nad názvem uzlu. Vazba nám říká, že uzel implementuje dané rozhraní. V diagramu je realizace znázorněna čárkovanou čarou se šipkou od uzlu k rozhraní.



Obrázek 1. Ukázka diagramu tříd.

2.2. Stavový diagram

Stavový diagram patří do skupiny diagramů chování. Jak již z názvu vyplývá, diagram zachycuje stavy vybraného objektu či celé aplikace. Stavový diagram, jak již název napovídá, je tvořen jednotlivými stavy (uzly). Mezi těmito uzly jsou definována propojení - přechody. Na základě událostí a pomocí těchto přechodů je možno zjistit, ve kterém stavu se sledovaný objekt nachází. Stavový diagram je v podstatě konečný automat. Stejně jako v automatu se i v diagramu nachází startovací uzel a jeden či více uzlů koncových.

2.2.1. Stav

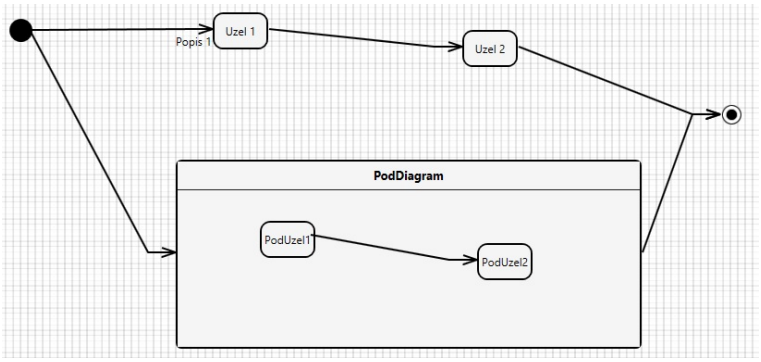
Stav znázorňuje trvání nastavení systému, nebo stav objektu. Stavem může být označena situace, kdy objekt čeká na spuštění události, nebo se objekt nějakým způsoben chová. Ve stavovém diagramu jsou stavy znázorněny obdélníky se zaoblenými rohy a popisem stavu uvnitř. Počáteční stav je znázorněn černým kolečkem. Teto stav se v diagramu může vyskytnout pouze jednou. Koncový stav, znázorněn černým kolečkem s bílými okraji, se v diagramu může vyskytovat vícekrát.

2.2.2. Přechod

Znázorňuje spojení dvou stavů v diagramu. Směřuje od zdrojového stavu k cílovému. Tyto přechody mohou být doplněny o popis přechodu. Syntaxe toho popisu je Událost [podmínka] / Akce.

2.2.3. Složené stavy

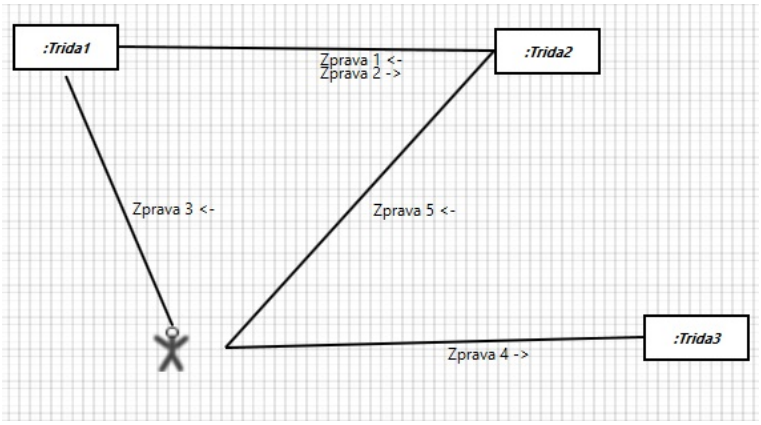
Stavy mohou obsahovat jeden, nebo více vnořených stavových diagramů. Vnořené stavy dědí všechny přechody svých nadstavů. Pokud tedy složený stav obsahuje přechod, znamená to, že všechny vnořené stavy tento přechod obsahují také. Další výhodou je znovupoužitelnost tohoto podstavu. Příkladem může byt třeba zpracování objednávky, které může být společné pro více systémů.



Obrázek 2. Ukázka stavového diagramu.

2.3. Diagram spolupráce

Diagram spolupráce patří do skupiny interakčních diagramů. Spolu se sekvenčním diagramem, o kterém budeme mluvit níže, jsou vzájemně izomorfní, což znamená, že je můžeme vzájemně převádět mezi sebou. Každý z těchto dvou diagramů klade jiný důraz na zobrazované skutečnosti. V diagramu se vyskytují jako uzly jednotlivé objekty systému. U diagramu spolupráce je kladen důraz na to, jaký objekt s jakým objektem komunikuje. V tomto diagramu, narozdíl od sekvečního diagramu, není možné zjistit časové závislosti posílaných zpráv. Jak již bylo řečeno, diagram tvoří uzly znázorňující objekty a vazby, které symbolizují posílané zprávy. Poslaná zpráva se napíše nad vazbu spolu se svými argumenty, stejně jako volání metody v programu. V případě posílání více různých zpráv stejnému objektu se používá pořád jedna a ta samá vazba a tyto zprávy se piší pod sebe s číselným označením.



Obrázek 3. Ukázka diagramu spolupráce.

2.4. Sekvenční diagram

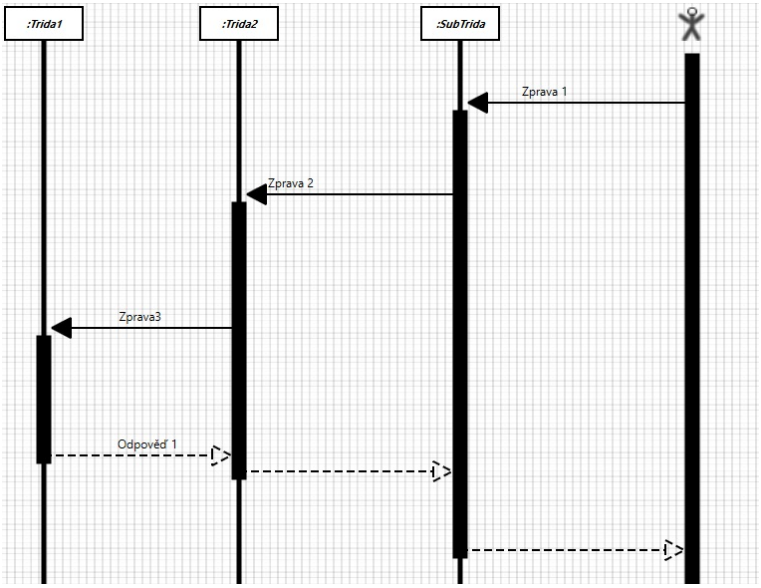
Sekvenční diagram je další digramem ze skupiny interakčních diagramů. Jak již bylo řečeno výše, s diagramem spolupráce jsou vzájemně izomorfní. V tomto typu diagramu opět pracujeme s objekty a jejich vzájemným vyměňováním zpráv. Důležitou součástí diagramu je tzv. čára života jednotlivých objektů, pomocí které můžeme sledovat aktivní dobu života objektu vzhledem k ostatním objektům. Na rozdíl od diagramu spolupráce jsou zde zprávy zobrazovány jednotlivě a jsou definovány pomocí specifikace různé druhy zpráv, jako synchoronní či asynchronní zpráva, nebo odpověď.

2.4.1. Uzel

Uzel, jak již bylo řečeno, je znázornění instance dané třídy. V diagramu je znázorněn jako obdelník, ze kterého vychází dlouhá čára znázorňující život objektu. Speciálním typem uzlu je uživatel, který je znázorněn obrázkem panáčka.

2.4.2. Zprávy

Zprávy můžeme rozdělit na synchronní a asynchrnonní. Synchronní zpráva objekt pošle jinému a čeká na odpověď. Po obdržení odpovědi pokračuje dál. Naproti tomu zprávu asynchronní objekt odešle a pokračuje dál. Odpověď je specifický typ zprávy poslaný objektu jako odpověď na předchozí zprávu. Synchronní zpráva je v diagramu zobrazena jako plná čára s nevyplněným trojúhelníkem na konci. Synchronní zpráva je v diagramu zobrazena jako plná čára s vyplněným trojúhelníkem na konci. Odpověď znázorňujeme čárkovanou čarou s nevyplněným trojúhelníkem na konci.



Obrázek 4. Ukázka diagramu spolupráce.

3. Software pro tvorbu UML

Existuje mnoho softwaru pro tvorbu diagramů UML. Liší se nabízenými funkcemi a nástroji pro práci s UML. Některé nabízejí pouze možnost tvorby diagramů, jiné slouží projektovým managerům pro řízení projektu, další mohou nabízet funkce reverzního inženýrství. Zde bych vyzdvyhnul jeden software, který dle mého názoru, patří k nejlepším. Tento software je rozsáhlý a zastřešuje výše uvedené funkce a mnohé další.

3.1. Aplikace Enterprise Architect

Enteprise Architect je nástroj pro tvorbu modelů založených syntaxi jazyka UML. Program je ověnčen mnoha oceněními z prestižních časopisů a soutěží. Nabízí kvalitní a vysoce výkonné vizuální prostředí pro řízení požadavků, strategické a business modelování (procesy, role, rizika apod.), návrh enterprise architektury a systémovou analýzu.

V dnešní době má Enterprise Architect již více než 250 tisíc instalací po celém světě a je podporován 230 partnery ve 160 zemích. Je široce používán v takových oblastech, jako je například zdravotní péče, letectví, bankovnictví, pojišťovnictví, automobilový průmysl, obrana nebo státní správa.

3.2. Srovnání

Jedinou výhodou, kterou má aplikace nabízí, je její jednoduchost a intuitivní ovládaní. Je to způsobeno malým počtem možností a nástrojů v aplikaci. Moje aplikace je vyhrazena na pouze na návrh aplikace, narozdíl od aplikace Enterprise Architect, která je mnohem komplexnější, a tím i složitější na používaní.

4. Popis algoritmu

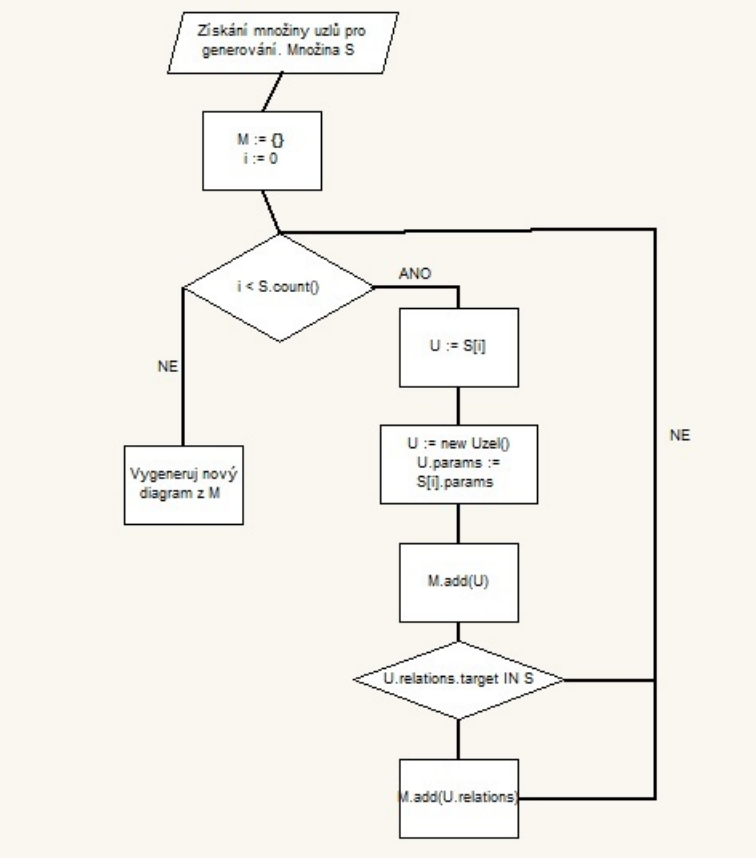
V této sekci popíši jednoduchý algoritmus používaný v mém programu k převodu diagramů.

4.1. Slovní popis

Tento algoritmus slouží k převodu, nebo spíše ke generování sekvenčního digramu, nebo diagramu spolupráce pomocí dříve definovaného diagramu tříd. Mějme vytvořený diagram tříd obsahující uzly rozhraní, třídy, enumerátory a vazby vedené mezi těmito uzly jako jsou asociace, generalizace, agregace, kompozice. Převodní metoda editoru nejprve nabídne uživateli, aby si zvolil, které třídy se budou účastnit generování, protože ne vždy je žádoucí použít všechny uzly z původního diagramu. Následně projde podmnožinu vybraných uzlů a zaměří se na vazby typu asociace. Jiné typy vazeb v generovaném diagramu nepoužijeme, protože se přímo neváží ke vzájemné komunikaci objektů. Program vygeneruje nový uzel pro každou vybranou třídu. Následně projde všechny vazby v předchozím diagramu a pokud existuje asociace mezi dvěma uzly s podmnožiny vybraných uzlů, vytvoří novou vazbu mezi novými uzly reprezentující původní uzly tříd.

Takto program postupuje při generování diagramu spolupráce. Při generování sekvenčního diagramu, ale není možné poznat i časovou závislost vazeb mezi objekty, se kterou tento diagram pracuje. V důsledku toho, nově vygenerovaný diagram obsahuje pouze uzly reprezentující objekty vybraných tříd v původním diagramu. Informace o vazbách jsou k dispozici při tvorbě nových vazeb v tomto diagramu. Uživatel tedy jen určí, kdy ke zprávě došlo.

4.2. Diagram



Obrázek 5. Diagram algoritmu.

5. Tvorba aplikace a použité technologie

5.1. Jazyk C# a platforma .NET

Tato aplikace je napsána v programovacím jazyce C# v prostředí .NET. C# je objektově orientovaný jazyk, vyvinutý společností Microsoft. Tento jazyk jsem zvolil z důvodu jeho pokročilé znalosti a v době začátku psaní aplikace jsem jiný jazyk neovládal. Ze zvolených technologii vyplývá, že tato aplikace je primárně určena pro počítače s operačním systémem Windows.

5.2. Windows Presentation Foundation

Technologie WPF je určena k tvorbě uživatelských rozhraní. Za pomoci této technologie je možné docílit daleko rozmanitějšího prostředí aplikace. Hlavním důvodem volby této technologie byla snaha si ji osvojit, protože se stává čím dál více používanou. Technologie WPF používá značkovací jazyk Extensible Application Markup Language (XAML) vycházející z XML. Největší výhodou těchto technologií je oddělení grafické a logické části aplikace. V jazyce XAML nadefinujete grafickou část aplikace a vytvoříte události, které následně na druhé straně obsloužíte v logické části.

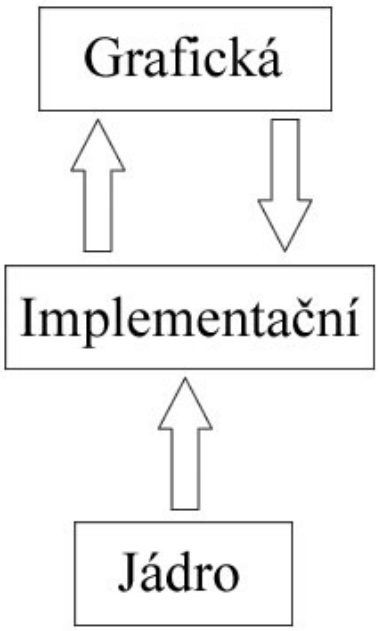
5.3. DataTemplate

Ve WPF má každý objekt své základní zobrazení. Pomocí Template je možné vzhled celého prvku kompletně přepsat. V mé aplikaci využívám DataTemplate, která slouží jako datová šablona pro objekty a lze v ní používat Data Binding.

5.4. Struktůra aplikace

Aplikace je rozdělena do tří vstev. Nejedná se přímo o model MVC, protože aplikace nepracuje a nezískává data z externích zdrojů, jakými jsou například databáze. Nejnižší vrstvu tvoří abstraktní třídy popisující základní chování vykreslovaných objeků v aplikaci. Implementační vrstva je část aplikace, ve které si již implementují třídy a metody pro konkrétní diagramy, které se vyskytují v aplikaci. Grafická vrstva je již pouze zobrazení vytvořených objektů nižších vrstev a jejich vykreslení. Do této vrstvy patří veškerý kód napsaný v jazyce XAML.

- Jádro aplikace
- Implementační vrstva
- Grafická vrstva



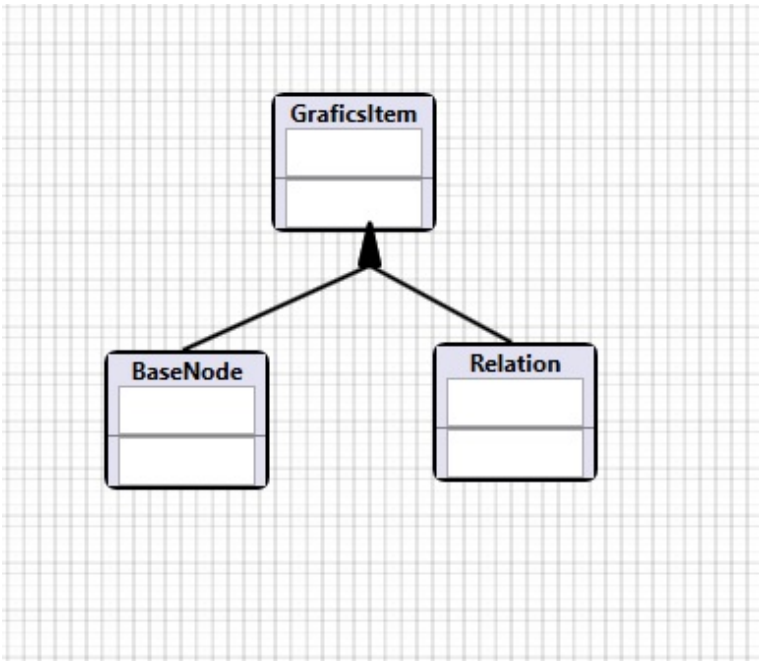
Obrázek 6. Struktura aplikace.

5.5. Jádro aplikace

Základ jádra aplikace tvoří potomci základní abstraktní třídy GraficsItem, která charakterizuje a zapouzdřuje základní vlastnosti a metody jednotlivých grafických objektů v mé aplikaci. Mezi tyto základní vlastnosti patří například:

- jednoznačný identifikátor napříč celým systémem,
- výška a šířka objektu,
- pozice objektu na vykreslovací ploše,
- dopočítaný střed objektu pomocí výšky a šírky,
- množinu bodů sloužících pro určení míst pro natažení vazeb,
- a mnohé jiné...

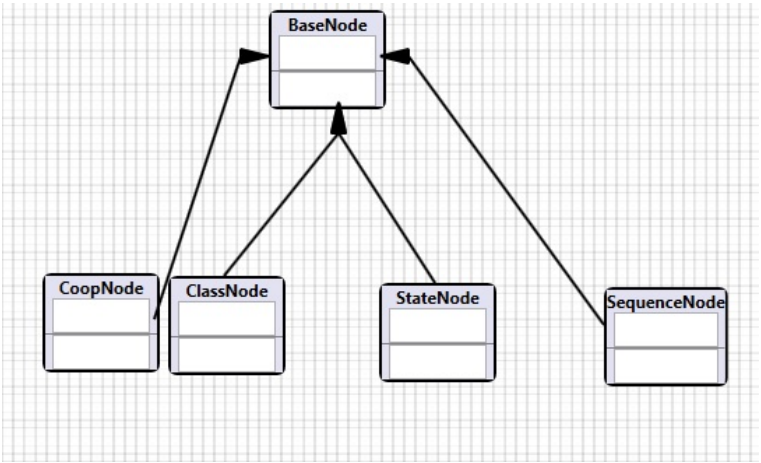
Dalšími abstraktními třídami dědících z třídy GraficsItem jsou BaseNode a Relation. Třída BaseNode zapouzdřuje základní metody pro práci s ulzy v grafickém editoru, jako například pohyb. Tato třída implementuje základní metodu pro uložení objektu do XML, o které bude řeč dále. Třída Relation je společným předkem všech vazeb, vyskytujících se programu. Obsahuje informace o orientaci vazby, cílovém a zdrojovém uzlu, a další.



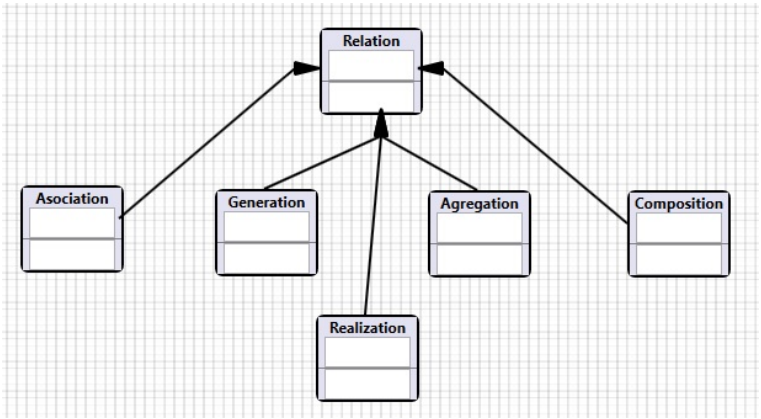
Obrázek 7. Jádro aplikace.

5.6. Logická část aplikace

V této části aplikace dochází ke konkrétní implementaci jednotlivých diagramů. Zakladní třídy jako BaseNode jsou rozširovány o další funkcionalitu. Například u diagramu tříd dochází k přidání metod a atributů pro práci s vlastnostmi a operacemi konkrétního uzlu. V této vrstvě se rozšiřuje i třída Relation. Vznikají nové třídy charakterizující vazby typu asociace, generalizace, realizace, kompozice a agregace. Zvlášt třída Association je hojně využívaná v ostatních typech diagramů. Nedílnou součástí logické části aplikace je statická factory třída Manager poskytující metody pro tvorbu instancí jednotlivých objektů diagramu. Obsahuje informace o právě tvořeném diagramu, seznam objektů a poskytuje rozhraní pro grafickou vrstvu. Manager patří k nejrozsáhlejší a nejkomplexnější třídě celé aplikace.



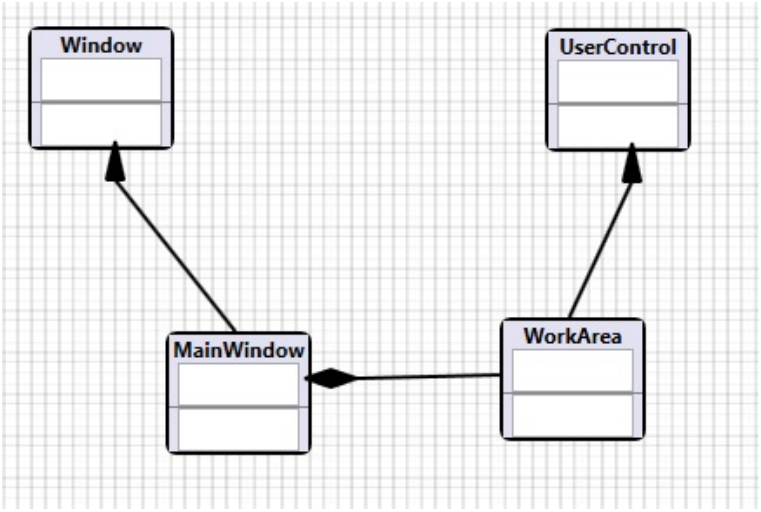
Obrázek 8. Implementace jednotlivých uzlů.



Obrázek 9. Implementace jednotlivých vazeb.

5.7. Grafická část aplikace

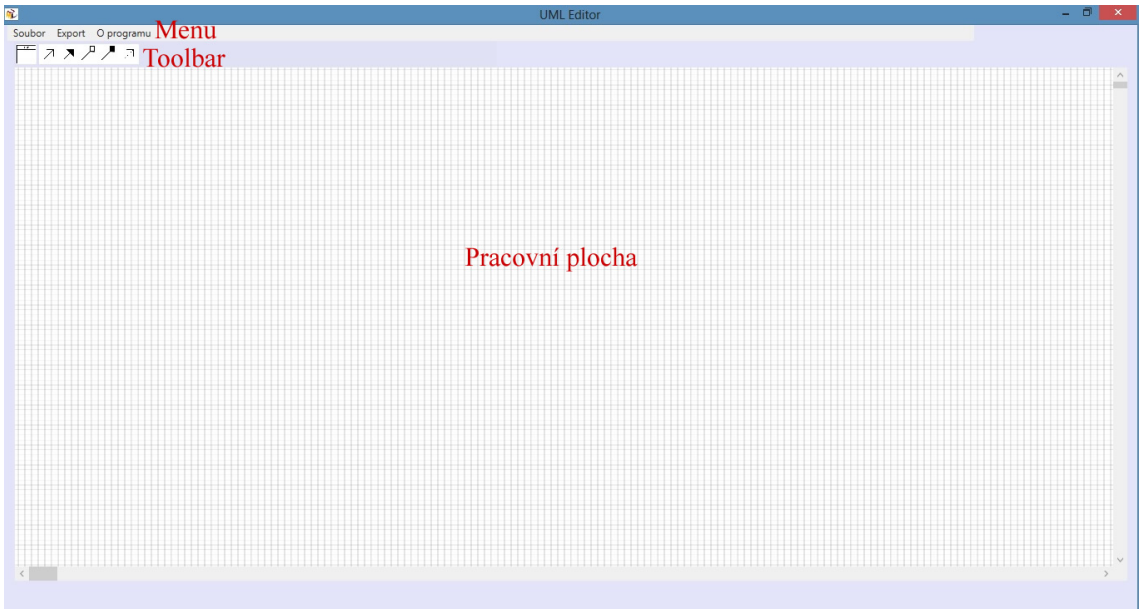
Hlavní třídou grafické části aplikace je třída WorkArea, potomek třídy UserControl. Třída tvoří pracovní plochu v prohlížeči. Na této ploše je možné umisťovat uzly a pomocí myši s nimi pohybovat. Dále je zde možné natahovat konkrétní vazby mezi uzly, opět za použití myši. Dalším prvkem grafické části je třída MainWindow, potomek třídy Window. Jedná se o hlavní okno aplikace a poskytuje funkcionalitu toolbaru a menu aplikace. Na úrovni této vrstvy dochází i k přepnutí jednotlivých template v závislosti na typu vykreslovaného objektu.



Obrázek 10. Implementace grafické vrstvy.

6. Uživatelská dokumentace

V této sekci popíši základy práce s programem.

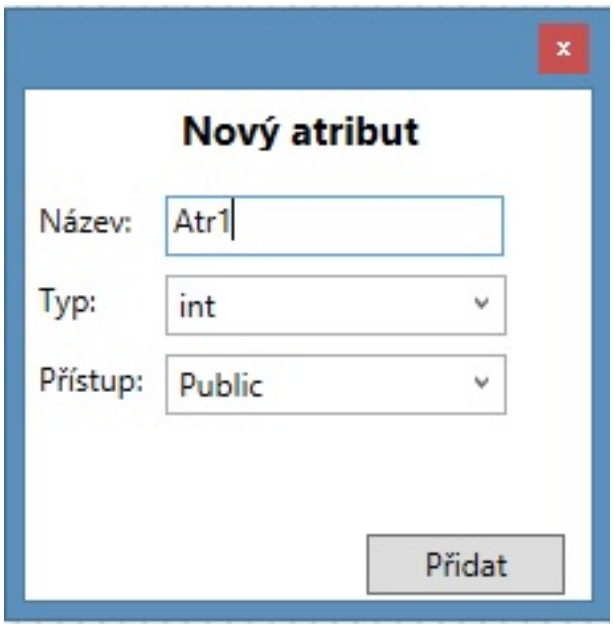


Obrázek 11. Výchozí okno aplikace.

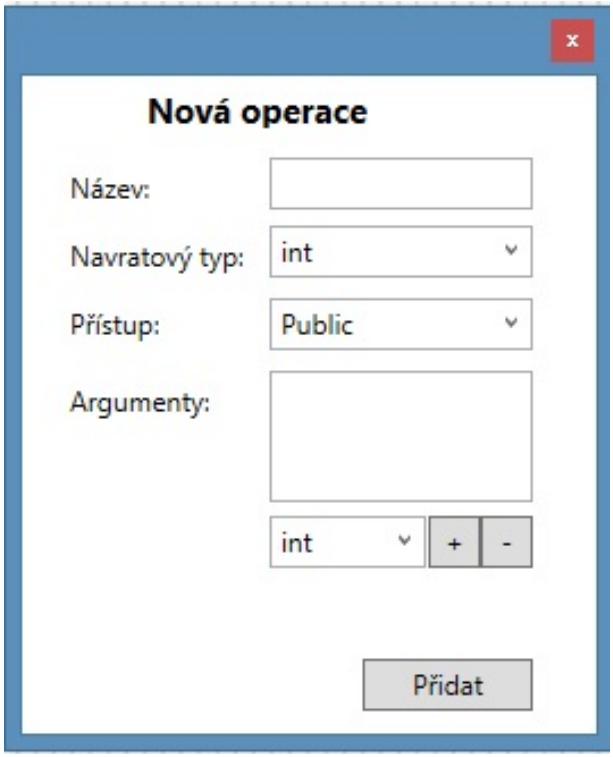
6.1. Pracovní plocha

Pracovní plocha tvoří hlavní část editoru. Pomocí myši do ní můžeme umisťovat uzly přetažením z toolbaru v levém horním rohu. Opět pomocí myši můžeme uzly libovolně pohybovat, a to kliknutím na uzel a následným držením a tahem. Pravým tlačítkem na uzel vyvoláme editační okno, v němž můžeme měnit nastavení uzlu. Po kliknutí na uzel jeho okraj zezelená, což značí právě vybraný uzel. U vybraného uzlu se nám v levém horním rohu objeví dvě ikonky, pomoci kterých můžeme opět vyvolat editační nabídku, nebo uzel smazat. Smazání uzlu je možné i jeho označením a stiskem klávesy DELETE. Smazáním uzlu budou smazány i vazby vztahující se k tomuto objektu.

Vazby je možné tvořit opět přetažením z toolbaru na konkrétní uzel, ve kterém má vazba začínat. Poté kliknutím na další uzel, nebo v případě reflexivní vazby na tentýž uzel, vytvoříme vazbu mezi těmito uzly. Stejně jako u uzlů, je možné vazbu mezi těmito uzly označit. Projeví se zezelenáním vazby. Druhým klikem na označenou vazbu vyvoláme editační okno vazby. V tomto okně je možné měnit orientaci vazby a její popis.



Obrázek 12. Tvorba atributu.

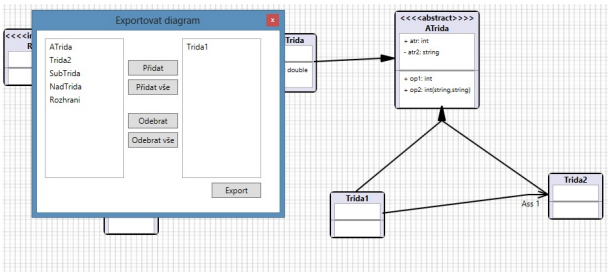


Obrázek 13. Tvorba operace.

6.2. Menu

Pomocí menu můžeme vytvořit nový diagram. Při tvorbě nového diagramu nejprve dojde ke kontrole a případnému uložení diagramu stavávajícího. Diagramy lze ukládat do formátu XML, který je program schopen znovu načíst. XML je zvleno z důvodu jeho lehké čitelnosti a snadné tvorby v programu. Dalším důvodem je jednoduší řešení konfliktů, pokud bychom chtěli digramy uchovávat například pod SVN.

Součástí menu je možnost exportovat diagram tříd na diagram sekvenční, nebo diagram spolupráce. Po kliknutí na zvolený export, vyvoláme exportovací okno. V tomto okně můžeme vybrat třídy, které se mají účasnit.



Obrázek 14. Export diagramu.

6.3. Tvorba diagramu tříd

V této sekci popíši tvorbu diagramu tříd. Obecně tvorba diagramů v aplikaci funguje stejně, jen diagram tříd má navíc i správu atributů a operací.

6.3.1. Třída

Nejprve v menu zvolíme možnost Nový diagram a zvolíme diagram tříd. Před námi je prázdná pracovní plocha. V toolbaru nalezneme vše co pro tvorbu diagramu potřebujeme. Třídu vytvoříme přetažením z toolbaru na pracovní plochu. Vyskočí nám dialogové okno pro zadání názvu a popřípadě stereotypu. S uzlem je možné pomocí myši manipulovat. Kliknutím na název a držením myší lze uzel přesunout.

Po vytvoření třídy vidíme prázdný seznam atributů a operací. Dvoj-klikem na prázdnou položkuv seznamu atributů se zobrazí okno pro přidání atributu ke třídě. Analogicky můžeme postupovat při přidání operace k třídě.

6.3.2. Vazby

Vazby máme opět v toolbaru. V závislosti na typu diagramu jsou k dispozici různé vazby. Vazbu vložím tak, že z toolbaru přetáhnu vybranou vazbu na uzel, ze kterého chci vycházet. Následným klikem na jiný uzel vytvořím příslušnou vazbu. Vazba je vždy v základním nastavení. Toto nastavení mohu změnit dvouklikem na vazbu. Objeví se mi dialogové okno s příslušným nastavením.

7. Závěr

V rámci této práce byl představen velmi jednoduchý editor pro tvorbu vybraných UML diagramů, kterými byli diagram tříd, stavový diagram, diagram spolupráce a sekvenční diagram. Program slouží pouze k nastínění tvorby těchto diagramů a určitě nemůže konkurovat profesionálním programům pro tvorbu návrhu a analýzu softwaru. Mým soukromým cílem nebylo pouze vytvořit editor, ale prohloubit si znalosti UML a také si vyzkoušet tvorbu grafické části aplikace pomocí technologie WPF. Musím uznat, že pokud bych měl začít tvořit tuto aplikaci znovu, použil bych nyní již jiné technologie a koncipoval bych tuto aplikaci jako webovou. Myslím si, že pro práci v týmu je webové prostředí lepší a snadnější pro sdílení mezi jednotlivými členy. Nicméně beru tuto práci pozitivně a myslím, že jsem se při jejím psaní naučil mnoho nového, co se týká programování i návrhu aplikace jako takové.

Reference

[1] *Unified Modeling Language^{TM} (UML®) Resource Page* [online]. 30.10.2014 [cit. 2014-12-10]. Dostupné z: http://www.uml.org/

[2] ARLOW, Jim a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky* .ARLOW, Jim a Ila NEUSTADT. Vyd. 1. Překlad Bogdan Kiszka. Brno: Computer Press, 2007, 567 s. ISBN 978-80-251-1503-9.

8. Obsah přiloženého CD

Soubory jsou na přiloženém disku rozčleněny do těchto adresářů:

- Text - obsahuje elektronickou verzi textu a soubory nutné pro sestavení dokumentu.
- Program - obsahuje zrdojové kódy