Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

BAKALÁŘSKÁ PRÁCE

Diagramy tříd a jazyk OCL

2015

Hojný Pavel

Vedoucí práce: Arnošt Večerka, RNDr.

Studijní obor: Aplikovaná informatika, kombinovaná forma

Bibliografické údaje

Autor:	Hojný Pavel
Název práce:	Diagramy tříd a jazyk OCL
Typ práce:	bakalářská práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2015
Studijní obor:	Aplikovaná informatika, kombinovaná forma
Vedoucí práce:	Arnošt Večerka, RNDr.
Počet stran:	32
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Hojný Pavel
Title:	Class diagrams and OCL
Thesis type:	bachelor thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2015
Study field:	Applied Computer Science, combined form
Supervisor:	Arnošt Večerka, RNDr.
Page count:	32
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Cíl práce je setavit aplikaci, která umožní kreslení diagramů tříd s podporou jazyka OCL. Konstrukce diagramu tříd by měla mít standardní formu dle UML. Nad diagramem tříd bude možné definovat invarianty tříd, počáteční hodnoty atributů, "pre" a "post" podmínky operací, těla dotazovacích operací, odvozené atributy.

Synopsis

The purpose of this thesis is to set up an application that allows drawing class diagrams and support OCL language. Designer of class diagrams should be according to the standard UML form. For classes can be defined invariants classes, initial attribute values, "pre" and "post" conditions for operarions, defining operation bodies, derived attributes.

Klíčová slova: Třídní diagramy; OCL; Object Constraint Language

Keywords: Class diagrams; OCL; Object Constraint Language

Děkuji RNDr. Arnošt Večerka za vedení mé bakalářské práce, doporučení literatury a cenné rady při konzultacíh.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam vět

Seznam zdrojových kódů

1 Úvod

Tématem této práce je sestavení grafického editoru třídních diagramů s možností zadávání omezení pomocí jazyka OCL. V rámci OCL konstrukcí je kontrolována syntaktická správnost zápisu a kontextová návaznost na objekty vytvořené v rámci modelovaného diagramu.

2 Třídní diagram

Třídní diagram je jeden z mnoha členů diagramů UML. Patří do rodiny digramů popisujících strukturu systému. Je také jedním z hlavních stavebních kamenů objektově orientovaného modelování. Může být použit k obecnému popisu aplikace, ale také k detailnímu modelování, které je následně převáděno do samotného kódu programu.

Třídní diagramy jsou i cílem reverzního inženýrství kdy je zdrojový kód převáděn do modelu. Pomocí třídních diagramů modelujeme prostředky a vztahy mezi nimi. Zobrazuje třídy, jejich atributy a operace (metody), které jde nad danou třídou provádět a vztahy mezi nimi.

Klíčové využití třídního diagramu:

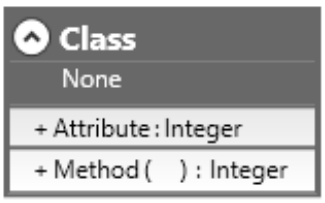
- Definování systémových zdrojů,
- Definování vztahů mezi jednotlivými entitami,
- Generování zdrojového kódu,
- Reverzní inženýrství

2.1 Třída (Class)

Třída tvoří základ třídního diagramu. Pomocí tříd definujeme prostředky, jejich možnosti a způsob práce s nimi. Každá třída musí mít v rámci modelu unikátní jméno. Toto jméno ji pak reprezentuje a můžeme se na něj v rámci diagramu odkazovat.

V diagramech zobrazujeme třídu jako obdélník rozdělený na 3 kontejnery. Každý jeden blok slouží k zápisu určité informace. První částí je jméno, v další části jsou zahrnuty atributy a v poslední třetí jsou zahrnuty operace (metody).

UML dále umožňuje tyto základní 3 kontejnery rozšířit o další uživatelsky definované kontejnery. V mé práci však nejsou uživatelské kontejnery zahrnuty a diagramy pracují pouze se standardní trojicí.



Obrázek 1: Grafické znázornění třídy

2.1.1 Atribut (Attribute)

Atributy tříd jsou zařazeny ve druhém kontejneru grafického znázornění třídy. Určují informace, které budou objekty třídy obsahovat. Mohou obsahovat i reference na jiné třídy nebo dokonce jejich kolekce. Zadání atributu dle UML se podléhá definici.

[**visibility**] [/] **name** [**: type**] [**multiplicity**] [= **default**] [**property-string**]

- name - název atributu
- visibility - viditelnost atributu vůči modelu
- type - datový typ

V předpisové této šabloně nejsou všechny prvky povinné, proto je v rámci přiloženého projektu využita pouze trojce nejdůležitějších údajů.

2.1.2 Operace (Operation)

Operace jsou zařazeny do třetího grafického bloku zobrazení třídy. Operace nám udávají chování objektu dané třídy a možnosti jakými lze s objektem manipulovat či manipulovat s jeho atributy.

[**visibility**] **name** ([**parameter-list**]) **':'** [**return-result**] [**properties**]

- name - název atributu
- visibility - viditelnost atributu vůči modelu
- parameter-list - seznam parametrů
- return-result - návratová hodnota

Zde je obdobně jako u atributů využita pouze nejdůležitější část předepsaných údajů.

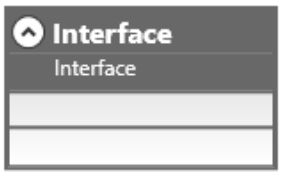
2.1.3 Viditelnost (Visibility)

V předchozích dvou bodech jsme se setkali s pojmem viditelnost. Tento údaj nám udává. Uplatňuje se jak u atributů, metod tak i samotných tříd. Viditelnost udává, komu bude daná vlastnost objektu přístupná a je pomocí ní možné rozdělit přístup k vlastnostem objektu.

- Private (-) - udává viditelnost pouze danému objektu, tyto atributy/metody slouží k realizaci vnitřních režií
- Package ( ) - viditelnost v rámci balíčku
- Public (+) - veřejná, přístupná v ramci balíčku
- Protected (#) - Přístupná v rámci děděných tříd

2.2 Rozhraní (Interface)

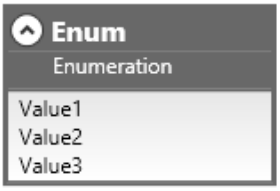
Třída rozhraní obsahující předpis chování, který musí implementující třída podporovat. Operace uvnitř rozhraní nejsou metody, obsahují pouze předpis chování. Atributy uvedené uvnitř rozhraní musí být abstraktní, to znamená, že atributy nemusí být deklarovány prováděcí třídou, pokud třída implementující rozhraní může poskytnout přístup ke stejným informacím prostřednictvím operace.



Obrázek 2: Grafické znázornění rozhraní

2.3 Výčtový typ (Enumeration)

Výčtový typ je uživatelsky definovaný datový typ. Jeho hodnoty jsou definovány jako literály. V grafickém zobrazení se jedná o obdelník rozdělený na dvě části. V první části je uveden název výčtového typu a klíčové slovo **enumeration** a v druhé části se nachází seznam hodnot, které výčtovému typu náleží.



Obrázek 3: Grafické znázornění výčtového typu

2.4 Komentář (Note)

Ke zlepšení přehlednosti a čitelnosti diagramů je zaveden komentář. Ten umožňuje doplnit diagramy o volný text a napojit jej k jakémukoli prvku modelu.

Obrázek 4: Grafické znázornění komentáře

2.5 Vztahy

Vztahy mezi objekty jsou nedílnou součástí diagramu. Každý z nich vyjadřuje učitý způsob propojení mezi dvěma entitami.

2.5.1 Asociace (Association)

Je nejjednodušší forma vztahu, která podporuje komunikaci mezi objekty. Asociace indikuje vztah mezi objekty dvou a více tříd. Často popisuje schopnost jedné instance poslat zprávu druhé instanci. Znázorňuje se plnou čarou.

Obrázek 5: Asociační vazba

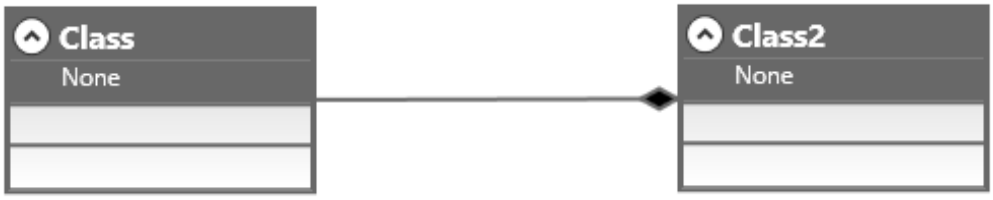
2.5.2 Agregace (Aggregation)

Agregace je specifičtější vztah než výše popsaná asociace a volnější vztah než níže popisovaná kompozice. Oproti kompozici je agregace volnější v tom, že zánik majitele nemusí zapříčinit zánik odkazovaného objektu. Znázorňuje se plnou čarou s bílým (nevyplněným) kosočtvercem.

Obrázek 6: Agregační vazba

2.5.3 Kompozice (Composition)

Kompozice je nejsilnějším z trojice vztahů asociace – agregace – kompozice. Odkazovaný objekt je zásadní součástí majitele. Existence odkazovaného objektu bez majitele nemá smysl. Zánikem majitele tak většinou zaniká i odkazovaný objekt. Znázorňuje se plnou čarou s černým (vyplněným) kosočtvercem.



Obrázek 7: Kompoziční vazba

2.5.4 Multiplicita/násobnost (Multiplicity)

Multiplicita je vyjádřením násobnosti u vztahů asociace, agregace a kompozice (u kompozice je násobnost možná pouze z jedné strany).

- 1 (libovolné číslo) - označuje konkrétní hodnotu
- * - libovolný počet, do výčtu padá i nula, někde označováno jako N
- 1..* - tento zápis označuje interval

Obrázek 8: Multiplicita/násobnost

2.5.5 Realizace (Realization)

Vztah realizace je mezi rozhraním a třídou, která tento interface implementuje. Třída implementující rozhraní je k samotnému rozhraní připojena vazbou podobnou dědičnosti. Znázorňuje se přerušovanou čarou s bílou (nevyplněnou) trojúhelníkovou hlavou.

Obrázek 9: Vztah realizace

2.5.6 Dědičnost (Generalization)

Hierarchický vztah tříd, v němž třída - potomek dědí atributy a operace třídy - předka. Kromě zděděných vlastností může mít potomek ještě své specifické vlastnosti. Zděděné vlastnosti z předka mohou být v potomkovi modifikovány. Znázorňuje se přerušovanou čarou s bílou (nevyplněnou) trojúhelníkovou hlavou. Znázorňuje se plnou čarou s bílou (nevyplněnou) trojúhelníkovou hlavou.

Obrázek 10: Vztah dědišnosti

2.5.7 Závislost (Dependency)

Jedná se o nejslabší vztah indikující závislost jedné třídy na jiné. Závislost je realizována například použitím druhé třídy jako parametru (nebo lokální proměnné) v některé z metod první třídy. Vztah je znázorněn přerušovanou čarou a jednoduchou šipkou.

Obrázek 11: Vztah závislosti

3 OCL

Pomocí UML diagramů nelze popsat veškeré podrobnosti modelu. Proto byl do standardu UML přiřazen jazyk OCL (Object Constraint Language). Jak už z názvu jazyka vyplývá tak slouží k popisům integritních omezení modelu. Pochází z metodiky Syntropy (IBM) a jde o čistě funkcionální jazyk určený k vytváření invariant. OCL je silně typovaný jazyk – každý výraz má definovaný typ a OCL má své vlastní předdefinované doplňkové typy. OCL není programovacíím jazykem. Je určen především jako jazyk pro vytváření invarinatů tříd tzn. jedná se o specifikační jazyk. Zápisy OCL konstrukcí je nutné chápat jako předpis pro programy, které budou integritní omezení uvedené v OCL kontrukcích zajišťovat.

3.1 Předdefinované typy OCL

V rámci své práce jsem omezil výčet standardních datových typů. Toto omezení vyplynulo z potřeby podpory OCL konstrukcí nad diagramem. Složitější datové typy (např. oclAny) vyžadovali zvláštní kontrolní pravidla. Proto jsou v OCL výrazech povoleny následující datové typy:

- Integer – celočíselný datový typ
- Real – reálná čísla
- String – textový řetězec
- Boolean – pravdivostní hodnota True/False
- Date - reprezentace datumové hodnot

3.2 Syntaxe OCL

V následující části si ukážeme několik nejzákladnějších příkazů jazyka OCL, jejich význam a ukázkové použití. Pomocí těxhto kontrukcí je možné dodat tvořenému modelu dodatečné informace, ktreré jsou požadovány.

3.2.1 Context

Context nám umožňuje výběr modelové třídy, pro kterou chceme specifikovat omezení. Na kontext třídy můžeme odkazovat pomocí jména tak pomocí klíčového slova **self**.

```
1  context Trida
2  inv: self.atribut < 18
3
4  context Trida
5  inv: atribut < 18
```

Zdrojový kód 1: OCL - použití contextu

3.2.2 Init

Zadávání výchozích hodnot nebylo UML umožněno. Proto je součástí jazyka OCL i výraz **init**, pomocí něhož je možné nastavovat výchozí hodnoty atributů a vyvářet objekty asociovaných tříd.

```
1  context Osoba::vek
2  init: 18
```

Zdrojový kód 2: OCL - použití init

3.2.3 Derive

Derive slouží k odvozování atributů. Je to jiný způsob jakým zjišťovat hodnotu atributu nebo objektu asociované třídy.

```
1  context Osoba::Cele_jmeno : String
2  derive: jmeno.concat(' ').concat(prijmeni)
```

Zdrojový kód 3: OCL - použití derive

3.2.4 Inv

Invarianty jsou omezení a pravidla, která by měla být splněna během životního cyklu objektu. Invarianty lze použít na jednotlivé atributy třídy.

```
1  context Student
2  inv: self.pohlavi = Pohlavi::zena
```

Zdrojový kód 4: OCL - použití inv

3.2.5 Pre, post, @pre

Tato dvojce určuje podmínky, které musí být splněny v určitém okamžiku životního cyklu objektu. Jedná se o omezující podmínky pro operace. Podmínky **pre** jsou vyhodnocovány jako pravdivostní výrazy a vyhodnocují se v momentě, kdy dojde ke spuštění operace. Jejím účelem je kontrola zda je možné operaci provést. Pokud bude podmínka vyhodnocena jako nepravdivá, nebude operace provedena. U **post** je situace obdobná pouze se podmínka vyhodnocuje až momentě doběhnutí operace. Hodnota podmínky určuje, zda operace doběhla v pořádku. Speciálním operátorem je **@pre**. Tento operátor lze použít pouze ve výstupních podmínkách tzn. V tělech výrazů post.

```
1  context Ucet::vybratHotovost(castka: Integer)
2  pre: castka > 0 and castka <= povolenyVyber
3  post: zustatek >= 0
4
5  context Zakaznik::maNarozeniny()
6  post: vek = vek@pre + 1
```

Zdrojový kód 5: OCL - použití pre, post, @pre

3.2.6 Body

Pomocí **body** definujeme výsledek dotazovacích operací. Výsledek dotazovací operace může být plně definován pouze jedním výrazem. Dotazovací operace nemají žádné vedlejší efekty, výsledkem je pouze hodnota nebo sada hodnot.

```
1  context Najemnik::jeDluznik(): Boolean
2  body: zustatekUhrad < 0
```

Zdrojový kód 6: OCL - použití body

3.2.7 Def

Slouží k definici nových atributů či dotazovacích metod. Metoda či atribut vytvořená pomocí def je globální pro celý kontext.

```
1  context Tenista
2  def: soucasneSkore: Integer =
3    vyhry->size() - prohry->size()
```

Zdrojový kód 7: OCL - použití def

3.2.8 Let

Funguje obdobně jako **def**. V případě let však dochází k definici lokálních atributů nebo metod.

```
1  context Firma
2  inv: let autaVArealu:Integer = prijezdy->size() - odjezdy->size()
3    in autaVArealu >= maximalniPovolenyPocet
```

Zdrojový kód 8: OCL - použití let

4 Programové zpracování

K vytvoření aplikace je požit programovací jazyk C#, .NET framework 4.5 a ke grafickému zpracování aplikace použita technologie WPF. Aplikace byla vytvořena pro operační systémy Windows verze 7 a výše. Samotný projekt je vytvořen ve vývojovém prostředí Visual Studio 2013. K návrhu grafických prvků aplikace byl použit nástroj Blend for Visual Studio 2013, který je součástí standardní instalace Visual Studio 2013 Professional. Jako základ grafického editoru byl použit příklad z reference. Tento příklad byl upraven a rozšířen pro potřeby mé práce. V této kapitole popíší nejvýznamnější třídy a komponenty programové části.

4.1 Třída DiagramObject

Základní objekt diagramu. Tato třída slouží jako šablona pro všechny objekty třídního diagramu (třída, výčotvý typ..). Podle přiřazeného typu objektu je instanci třída přiřazena grafická šablona a upraveny některé vlastnosti. Nejdůležitější atributy objektu:

- Type - určuje typ vloženého objetku
- AttributeList – seznam atributů

- MethodList – seznam metod
- Note – textová položka, má dvě využít buď jako poznámka nebo nositel OCL kódu

DiagramObject je potomkem .NET třídy `Thumb`, ke které je připjeno několik dalších rozhraní rozšiřujících základní funkčnost jako umožnění funkcionality drag&drop či kontrola propojitelnosti s různými druhy objektů pomocí `IConnectable` rozhraní.

4.2 Třída LineConnector

Třída reprezentuje vztahy mezi objekty. Je potomkem třídy `DiagramObjectConnector`, která obsahuje dva závislotní atributy nesoucí informaci o počátečním a koncovém bodu spojení. Díky těmto dvěma atributů dochází k automatické aktualizaci pozice bodů kreslené čáry vztahu. Samotná LineConnector třída obsahuje vniřní instanci .NET třídy `Canvas`, která slouží jako plátno pro vykreslování všech náležitostí konektoru. Z důvodu nutnosti vykreslovat složitější typy čar skládájící se z více segmentů doplněných o dodatčné informace jako popisky či geometrické obrazce pro konec čáry, jsou na této třídě přetíženy metody na získávání grafických částí konektoru.

Samotný konektor není je plátno s členitou čárou, ale nese na sobě také další důležité informace.

- ObjectFrom/ObjectTo - objekty, které konektor spojuje
- cardinalFrom/cardinalTo - textové řetězce s informacemi o popiscích
- Type - typ konektoru
- Container - samotné plátno nesoucí všechny grafické části konektoru
- Head - polygon, jeho tvar se liší dle typu konektoru
- LinePoints - seznam bodu čáry vyjma počátečního a koncového bodu
- adorners - důležitý seznam drag&drop prvků pro manipulaci s body čáry v LinePoints

4.3 Třída DiagramDesigner

Hlavní nosná třída diagramu. Slouží jako poskytovatel základních služeb pro manipulaci s objekty diagramu. Významnou službou je `SelectionService` tato služba zaopatřuje označování objektů v plátně diagramu. Další důležitou službou, kterou DiagramDesigner poskytuje je `ToolService`. Tato služba zaobaluje základní logiku na vytváření nových objektů pro třídní diagram.

4.4 Třída ProgramData

Statická třída ProgramData obsahuje veškeré objekty diagramu. Účelem této třídy je snadný přístup ke všem částem diagramu. Obsahuje seznam objektů `ObjectList` (seznam objektů `DiagramObject` ukládaných jako `UIElement`), seznam konektorů `ConnectorList` a další dva seznamy vlastností. Tyto seznamy slouží jako zdroj informací pro ukládání a načítání souborů. Navíc do této třídy přistupuje i samotná kontrola výrazů jazyka OCL a to za účelem ověření správnosti kontextu.

4.5 Třída Tokenizer

První třída pro zpracování OCL konstrukcí. Tato třída zpracuje vstupní text a vrací seznam tokenů. Každý token nese informaci:

- column - číslo znaku
- line - číslo řádku
- value - hodnota přečteného tokenu

Takto zpracovaný vstupní text je následně předán dál ke zpracování.

4.6 Parser

Samotný parser jako celek je soustava na sebe navazujících tříd. Každá třída spadající do zpracování OCL představuje jeden neterminál z pravidel gramatiky uvedných v této kapitole. Postupným průchodem předaného seznamu tokenů je kontrolována syntaxe zapsané OCL konstrukce. Parser zpracovává i kontrolu kontextu. Syntaktická kontrola podporuje podmnožinu standardních vestavěných funkcí OCL.

4.6.1 Vestavěné funkce OCL

V zápisech OCL výrazů je možné použít standardní vestavěné funkce. Seznamy funkcí, které lze použít ve výrazech jsou uloženy ve statické třídě `Global`. Ta obsahuje všechna podporovaná klíčová slova. Vzhledem k rozsahu bakalářské práce jsou tyto funkce vyhodnocovány vždy stejným způsobem. Dochází ke kontrole povolené operace, ale parametry či tělo funkce se s ohledem na zadanou operaci nekontrolují. Např. operace `isEmpty` nemá parametry a operace `includesAll` vyžaduje jako parametr kolekci. U obou dojde k následujícímu vyhodnocení:

1. najdi jméno povolené operace
2. přečti parametry
3. vyhodnoť parametr

Jelikož implementovaná gramatika povoluje zápis funkcí i bez parametrů je možné zapsat `includesAll` bez jakéhokoliv parametru a výraz bude vyhodnocen jako syntakticky správný.

4.7 ObjectParametersInherit

Důležitá třída pro získávání informací třídního diagramu. Primárně slouží jak zdroj dotazovacích metod k získávání existence a dostupných kontextů pro OCL konstrukce. Seznam nejpoužívanějších metod:

- `public static List<string> GetClassAttributes(string name)` - vrací seznam atributů zadané třídy
- `public static List<string> GetClassMethods(string name)` - vrací seznam metod zadané třídy
- `public static string GetClassByRole(string context, string role)` - vrací název třídy podle zadané třídy a jmené role
- `public static List<string> GetClassNamedRoles(string name)` - vrací seznam rolí třídy

4.8 Gramatická pravidla OCL

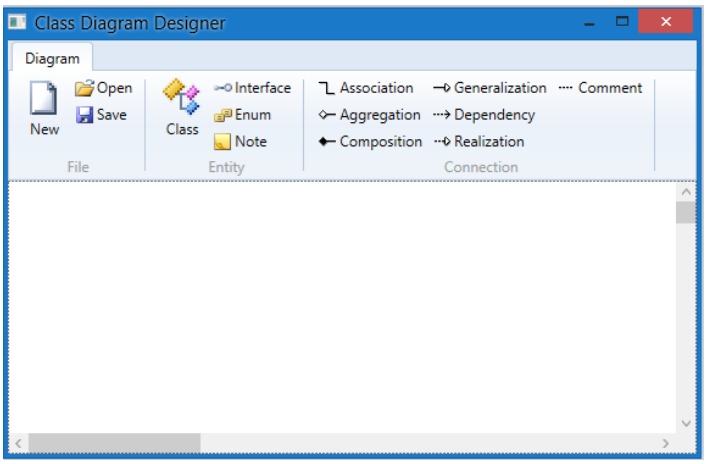
Gramatická pravidla uvedená v tomto textu jsou použita pro vyhodnocování OCL kontrukcí v programové části mé práce. Pravidla vznikla jako kombinace EBNF pravidel z oficiální gramatiky OCL a gramatických pravidel uvedených v referencích. Vysvětlivky k operátorům pravidel:

- ***** - určuje výskyt výrazu 0..n
- **+** - určuje výskyt výrazu 1..n
- **?** - označuje volitelný výraz v rozmezí 0..1

Pravidla jsou přiložena k práci jako samostaný textový soubor viz. přílohy.

5 Uživatelská příručka

Výsledný program umožňuje vytvářet třídní diagramy a nad třídami diagramu definovat OCL konstrukce. V této kapitole si popíšeme jakým způsobem s programem pracovat.



Obrázek 12: Aplikace - hlavní okno

5.1 HW/SW požadavky

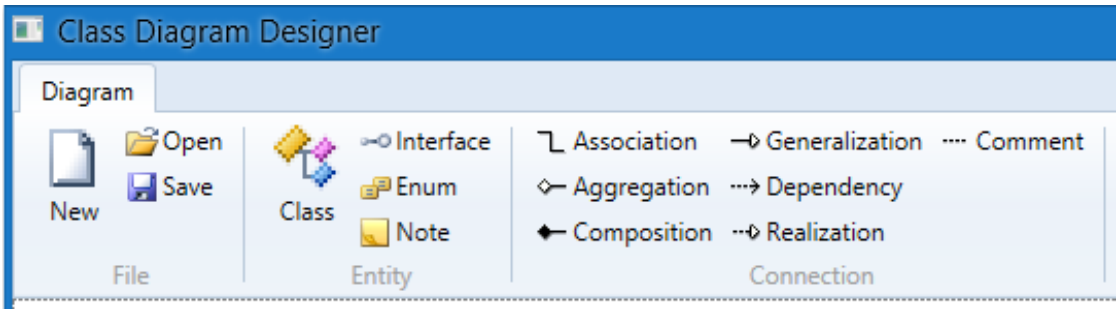
Aplikace byla vyvinuta pro operační systém Windows. Podporovány jsou všechny verze, které umožňují instalaci :NET Framework 4.5. Aplikace byla vyvinuta a testována na Window 8 32bit a Windows 8.1 64bit. Doporučená konfigurace:

- **Procesor**: Intel® Core®2 Duo
- **Operační systém**: Windows 8 32bit
- **paměť RAM**: 2GB
- **.NET framework**: verze 4.5
- **Minimální rozlišení**: 1024x768

Instalační balík je vytvořen standardní cestou pomocí nástroje Visual Studio 2013.

5.2 Hlavní menu

Po spuštění aplikace se zobrazí úvodní okno aplikace. Je zde základní nabídka povolených operací v hlavním Ribbon menu aplikace. To je rozděleno do tří skupin. První skupina `File` slouží pro ukládání a načítání uložených diagramů. `Entity` obsahuje nástroje po vytváření objektů diagramu. Třetí blok `Connection` obsahuje nástroje pro vytváření vztahů mezi objekty diagramu.

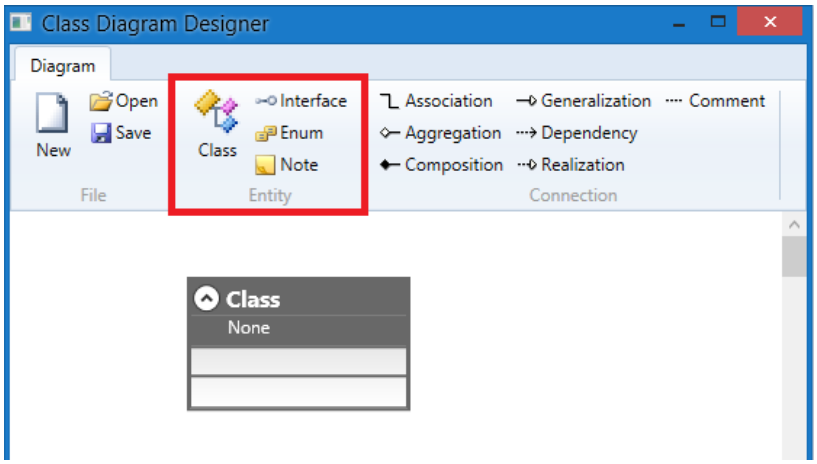


Obrázek 13: Aplikace - Hlavní menu

5.3 Vytváření objektů

Pro přídání nového objektu vybereme typ objektu z hlavního menu ve skupině `Entity`. Vytvoření probíhá ve dvou krocích:

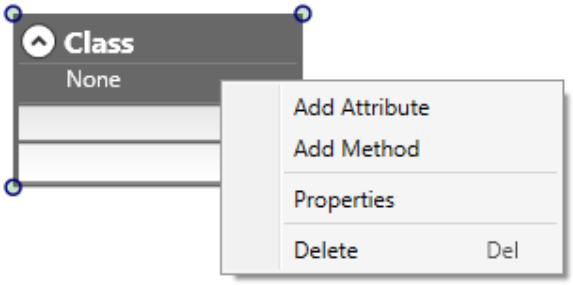
1. Kliktnutím na ikonu objektu zvolíme požadovaný objekt
2. Kliknutím do okna diagramu vytvoříme objekt na požadovaném místě



Obrázek 14: Aplikace - Vytváření objektu

5.3.1 Úpravy vlastností

U každému objektu je možno měnit jeho vlastnosti. Přístup k vlastnostem je umožněn přes kontextovou nabídku vyvolanou kliknutím pravým tlačítkem na vytvořený objekt.



Obrázek 15: Aplikace - Kontextová nabídka objektu

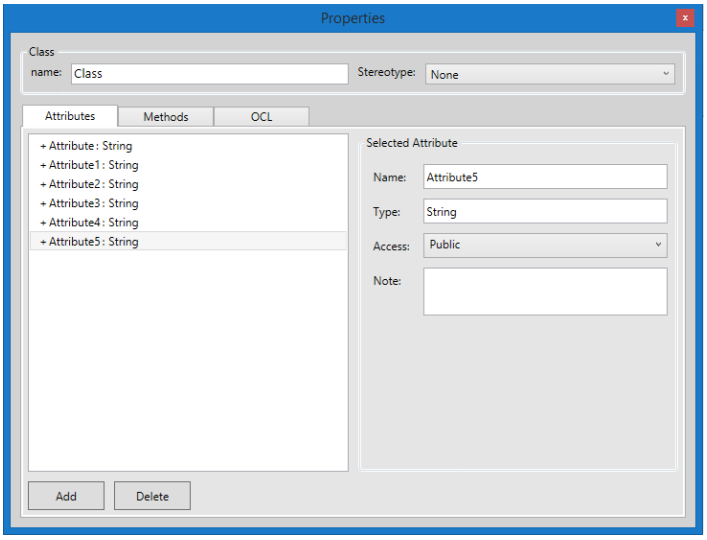
- **Add Attribute** - vytvoří nový atribut
- **Add Method** - vytvoří novou metodu
- **Properties** - otevře okno pro správu vlastností objektu
- **Delete** - odstraní vybraný objekt a všechy napojení

5.3.2 Dialog úprav

Pro komplexnější úpravy je určeno dialogové okno `Properties`. Okno je dostupné přes položku kontextové nabídky - `Properties`. Okno je rozděleno na tři záložky:

- **Attributes** - záložka pro správu atributů
- **Methods** - záložka pro správu metod
- **OCL** - záložka OLC editoru

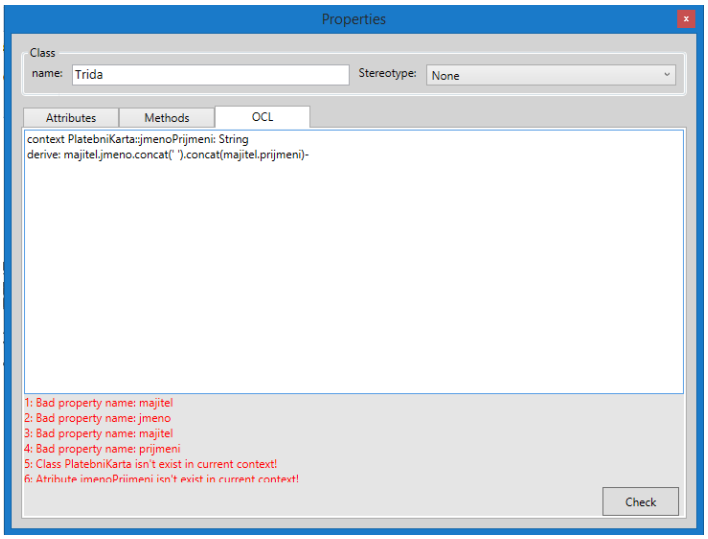
Obsluha atributů a metod je obdobná. V levé části záložky se nachází seznam vytvořených položek, v práve části se nachazí kolonky pro úpravy hodnot jednotlivých položek. Pomocí tlačítka `ADD` vytvoříte nový záznam. Tlačítkem `DELETE` dojde k odstranění vybraného záznamu.



Obrázek 16: Aplikace - Dialogové okno vlastností objektu

5.3.3 Editor OCL

Editor OCL konstrukcí je součástí dialogového okna `Properties`. Je rozdělen na dvě části. Horní část je určena pro zápis OCL konstrukcí spodní část zobrazuje případné chyby. Po zapsání OCL konstrukce nedochází k automatické kontrole syntaxe ani kontextu. Pro kontolu vepsaného výrazu je nutné stisknout tlačítko `Check`. Po stisku dojde ke kontrole. Pokud je výraz správně zapsán je dolní část editoru prázdná, jinak v ní lze najít seznam chyb, které zapsaný výraz obsahuje.



Obrázek 17: Aplikace - OCL editor

Při definování kontextu a přístupu k atributům přes role je potřeba uvést vždy celou cestu vedoucí k objektu z hlavního kontextu. Jinak bude výraz vyhodnocen jako kontextově nesprávný. viz příklad `fakulta.xml` v příloze

```
1 -- správně
2 context Fakulta
3 inv: obory->collect(obory.studenti)->collectNested(obory.studenti.
      rodice)->asSet()
4
5 --chybně
6 context Fakulta
7 inv: obory->collect(studenti)->collectNested(rodice)->asSet()
```

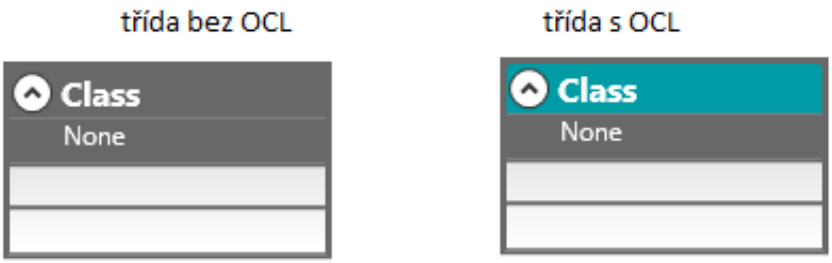
Zdrojový kód 9: Aplikace - správné uvádění kontextu

Dalším drobným omezením je zápis operací uvedených u atributů typu třída. Dochází ke kontrole pouze povoleného datového typu, ale dotaz na operaci navázanou k třídě, která není s diagramem spojena kontrola kontextu nedosáhne. viz příklad `snatek.xml`

```
1 -- chyba - nedoje k dohledání operece jePred()
2 context Manzel
3 inv: Snatek::datum.jePred(Udaje::dnes)
```

Zdrojový kód 10: Aplikace - omezení operací

Objekty nesoucí OCL kontrukce jsou graficky odlišeny. Po zadání OCL dojde ke změně barvy hlavičky grafického znázornění.



Obrázek 18: Aplikace - Grafické rozlišení objektů s OCL

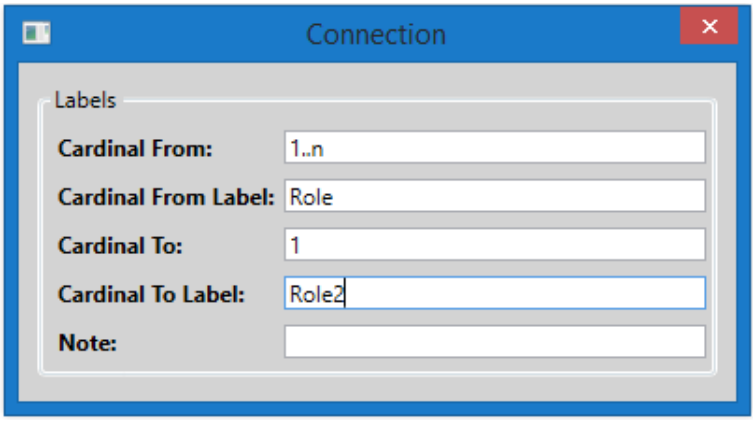
5.4 Vytváření propojení (vztahů)

Pro vytvoření nového propojení vybereme typ z hlavního menu ve skupině `Connection` . Propojení lze tvořit vždy mezi dvěma objekty, nebo jeden objekt "sám do sebe". Vytvoření probíhá ve čtyřech krocích:

1. Kliknutím na ikonu objektu zvolíme požadovaný typ propojení
2. Kliknutím do okna diagramu vytvoříme objekt ze kterého propojení povede
3. Držíme stále stisknuté levé tlačítko myši a táhneme na cílový objekt
4. Jakmile jsme dorazili myší na zvolený objekt tlačítko uvolníme

5.4.1 Úpravy vlastnosti propojení

Vlastnosti se mění stejným způsobem jako u objektů pomocí kontextové nabídky. Pouze dialogové okno pro úpravy je upraveno pro změny týkající se propojení.



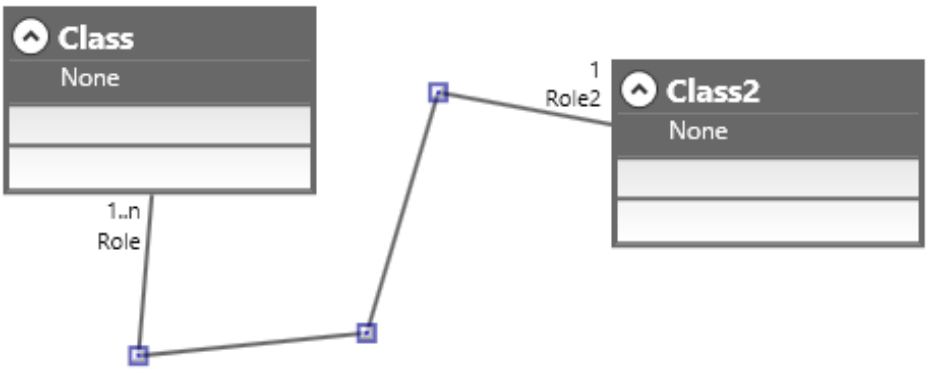
Obrázek 19: Aplikace - Dialogové okno vlastností propojení

5.4.2 Manipulace s čárou propojení

V diagramu je možné upravovat vzhled čáry. Čáru propoje lze rozdělovat na segmenty a následně s nimi manipulovat pro zpřehlednění diagramu. Dvojklikem na čáru propoje dojde k vytvoření bodu zlomu. Po označení čáry se zobrazí všechny body se kerými lze čárou manipulovat. Body určené k manupulaci jsou označeny jako modrý čtvereček viz. obrázek. Odstranění zlomového bodu se provede opět dvojklikem, ale tentokrát na zlomový bod.

5.5 Ukládání/načítání diagramů

Diagramy se ukládájí do souborů ve formátu XML. Samotné operace načítání je možné spusitit z první skupiny ovládacích prvků v hlavním menu. Pro výběr souboru jsou použity standardní systémové dialogy.



Obrázek 20: Aplikace - Manipulace s čárou

Závěr

Cílem mé bakalářské práce bylo vytvoření editoru třídních diagramů s podporou jazyka OCL. K vytvoření bylo použito vývojové prostředí Visual Studio 2013. Výslednou aplikace lze použít jako pomocný nástroj při návrhu jednodušších třídních diagramů. Díky zabudované podpoře OCL je možné výsledné diagramy doplnit o integritní omezení což v editorech podporujících pouze UML standardní diagramy nelze.

Conclusions

The purpose of my thesis was to create graphical editor application for drawing class diagrams with support for OCL language. Application was created using Visual Studio 2013. Final application can be used as a helper tool for designing simplier class diagrams. With built-in OCL support class diagrams can be improved by constraints which classic editors that supports only standard UML diagrams can't do.

A Obsah přiloženého CD/DVD

bin/
: Samostatně spustitelná zkompilovaná verze programu.

install/
: Instalátor SETUP.EXE programu.

doc/
: Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

src/
: Kompletní zdrojové texty programu ClassDiagramEditor se všemi potřebnými zdrojovými texty, knihovnami a dalšími soubory potřebnými pro bezproblémové vytvoření spustitelných verzí programu. Soubor DIAGRAM.ZIP obsahující zkomprimovaný adresář se všemi zdrojovými kódy.

readme.txt
: Instrukce pro instalaci a spuštění programu Program, včetně všech požadavků pro jeho bezproblémový provoz. Přiložena kompletní adresářová struktura přiloženého disku.

Navíc CD/DVD obsahuje:

example/
: Ukázkové a testovací diagramy použité pro potřeby testování aplikace.

gramatika/
: Textový soubor GRAMATIKA.TXT - gramatika jazyka OCL použitá v programové části.

  Textový soubor STANDARDNI_FUNKCE.TXT - výčet podporovaných standardních funkcí jazyka OCL.

  Textový soubor STANDARDNI_FUNKCE_KOLEKCE.TXT - výčet standardních funkcí pro kolekce jazyka OCL.

Bibliografie

[1] WARMER Jos, KLEPPE Anneke: Object Constraint Language, The: Getting Your Models Ready for MDA. Addison Wesley, 2003. ISBN: 0-321-17936-6

[2] PENDER Tom: UML Bible. John Wiley & Sons, 2003. ISBN: 0764526049

[3] VUYKA Denis: WPF DIAGRAMMING. DRAWING A CONNECTION LINE BETWEEN TWO ELEMENTS WITH MOUSE. Dostupné z: https://denisvuyka.wordpress.com/2007/10/21/wpf-diagramming-drawing-a-connection-line-between-two-elements-with-mouse/

[4] VUYKA Denis: WPF DIAGRAMMING. SIMPLE DIAGRAMMING CTP PROJECT FOR PROCESS DESIGNING. Dostupné z: https://denisvuyka.wordpress.com/2007/11/13/wpf-diagramming-simple-diagramming-ctp-project-for-process-designing/

[5] OCL Grammar Dostupné z:http://www.yancy.org/education/phd/links/ocl-grammar-01b.pdf

[6] OMG.org: Documents Associated With Object Constraint Language (OCL), Version 2.4 Dostupné z:http://www.omg.org/spec/OCL/2.4/