FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Export OntoUML modelů do přirozeného jazyka
Student:	Alina Zarubaeva
Vedoucí:	doc. Ing. Robert Pergl, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Jazyk OntoUML (https://ontouml.org) umožňuje přesné zachycení pojmů a vztahů reálného světa (ontologie). Jeho grafická notace s matematickým pozadím však není úplně vhodná pro netechnické uživatele. Řada profesí (zejména např. právníci) preferují textovou formu specifikací. Cílem práce je implementovat export OntoUML modelů do přirozeného jazyka.

1. Proveďte rešerši problematiky transformace grafických konceptuálních modelů do přirozeného jazyka.
2. Navrhněte algoritmus pro transformaci z OntoUML modelů do vhodné podmnožiny přirozeného jazyka. Berte v potaz možnosti různých přístupů procházení grafu modelu.
3. Navržený algoritmus implementujte v podobě transformace z XMI formátu generovaným nástrojem OpenPonk do Markdown syntaxe.
4. Vytvořené řešení otestujte a demonstrujte na případové studii.
5. Výsledky zdokumentujte a okomentujte.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 18. listopadu 2023 v Praze.

Bakalářská práce

EXPORT ONTOUML MODELŮ DO PŘIROZENÉHO JAZYKA

Alina Zarubaeva

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: doc. Ing. Robert Pergl, Ph.D.
6. května 2024

Odkaz na tuto práci: Zarubaeva Alina. *Export OntoUML modelů do přirozeného jazyka.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2024.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěla bych poděkovat především vedoucímu mé bakalářské práce doc. Ing. Robertu Perglovi Ph.D., který mi pomohl zvolit téma, vždycky poradil a ukázal cestu správným směrem k úspěšnému dosažení všech cílů. Ráda bych také poděkovala svému manželovi a matce za jejich podporu během celého studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 6. května 2024

Abstrakt

Hlavním cílem práce je navrhnout a implementovat aplikaci pro převod schematický reprezentovaných OntoUML modelů vytvořených v aplikaci OpenPonk ve formátu XMI na jejích textovou reprezentaci v přirozeném jazyce. Aplikace musí umožnit uživateli nahrání XMI souboru s modelem OntoUML, zvolení entity, která v generovaném textu bude vystupovat jako hlavní, tzn. popis bude začínat právě zvolenou entitou a nahrání výsledného textu do souboru PDF. Uživatel musí mít možnost přidání více sekcí s různými kombinacemi režimů a počátečních entit. Součástí práce je taky analýza možných způsobů reprezentace modelu na text a návrh algoritmů pro export modelu do textu. Výsledkem práce je aplikace umožňující nahrání XMI souboru s modelem OntoUML, následně převod a uložení textového souboru s popisem schématu v přirozeném jazyce.

Klíčová slova OntoUML, konceptuální modelovani, XMI, export, PDF

Abstract

The main aim of this work is to suggest and implement the application for transformation of schematic OntoUML models created in OpenPonk application in XMI format to a text representation using natural language. The application must allow the user to select the file with the OntoUML model, choose the entity which will be the main in the generated text (it means that the description will be started exactly with the selected entity) and save the result text to PDF file. User should be able to add more sections with starting entities and modes for one generated file. The work also includes an analysis of possible ways of representing the model to text and the design of algorithms for exporting the model to text. The result of the work is the application allows users to choose the XMI file with the OntoUML model, transform and save the text file with the schema's description in natural language.

Keywords OntoUML, conceptual modeling, XMI, export, PDF

Seznam zkratek

OntoUML	Ontological Unified Modeling Language
WYSIWYG	What You See Is What You Get
UML	Unified Modeling Language
XML	Extensible Markup Language
XMI	XML Metadata Interchange
OMG	Object Management Group
SBVR	Semantics of Business Vocabulary and Business Rules
PDF	Portable Document Format
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphic
HTML	HyperText Markup Language
QML	Qt Modeling Language
GUI	Graphical User Interface
FIT	Fakulta informačních technologií

Úvod

0.1 Motivace a přínosy práce

Konceptuální modelování je důležitou disciplínou, která umožňuje lidem si zjednodušit pochopení procesu porozumění v profesionální oblasti. Schematická reprezentace, která je řízena svými určitými předpisy, hraje roli jazyka, kterým mluví každý, kdo ví, jak fungují pravidla daného způsobu reprezentace modelu. Z tohoto vyplývá následující problém: osoba, pro kterou je daný typ modelování neznámý, nemusí umět správně přečíst navržené schéma a pochopit, co se přesně říká takovým způsobem vytvořený popis.

V současné době velké množství lidí pracuje v oblasti informačních technologií. Mezi nimi patří nejen zaměstnanci se vzděláním technického zaměření, například, vývojáři nebo testeři, ale i ti, kteří pro svou činnost nepotřebují, ale stejně se setkávají s pojmy a výsledky práce kolegů přímo pracujících s informačními technologiemi. Mezi takovýmito zaměstnanci patří, například, manažeři nebo vedoucí projektů. Také obecně platí, že technicky vzdělaní specialisté mohou vytvořit něco, co není pochopitelné pro ostatní pracovníky. OntoUML diagramy jsou přesným příkladem podobného výsledku práce. Jelikož je to velmi dobrý a mocný nástroj, který dovoluje stručně a podrobně popsat svoje myšlenky, je důležité rozumět tomu, co znamená každý stereotyp a označení ve výsledném schématu, aby se přesnost neztratila.

Pro vyřešení tohoto problému je potřeba, aby se člověk, který nikdy a nikde dřív nepotkal modely OntoUML, naučil tomu, jak má správně přečíst a pochopit informace reprezentované modelem. Z tohoto vyplývá další problém, spojený s tím, že, například, pro zaměstnance to může znamenat to, že získá informace nebo dovednosti, které využije jen zřídka, ale ztratí s tím hodně času. Z pohledu byznysu a ekonomiky podniku to znamená úbytek peněz. Druhou cestou je to, že nějaký zkušený technicky vzdělaný člověk vysvětlí všechny podrobnosti tomu kolegovi, který to potřebuje pro svou činnost vědět. Tady zase vidíme problém toho, že z pohledu podniku je to minimálně dvakrát zvětšený objem ztracených peněz a času pro jeden drobný problém.

Existuje však i jiné řešení spojené s automatizací procesu popisu modelu OntoUML včetně všech nezbytných podrobností. Takovým řešením je vývoj softwaru, který automaticky převede model z jazyka OntoUML do podoby v přirozeném jazyce v textové formě. Výhodami této varianty řešení jsou znovupoužitelnost, jednoduchost aplikace pro uživatele s minimálními znalostmi konceptu OntoUML a také možnost zkombinovat výslednou aplikaci s již existujícími nástroji spojenými s práci s OntoUML (například automatická kontrola správnosti modelu).

Nedílnou součástí této bakalářské práce je analýza a zvolení nástrojů a technik pro implementaci požadovaného softwaru. Pro vytvoření samotného programu je v backendu použit programovací jazyk C++ a pro implementaci uživatelského rozhraní je zvolen Qt framework. Výsledná aplikace dovoluje uživateli zvolit soubor s modelem OntoUML ve formátu XMI, vybrat potřebná nastavení pro transformaci modelu a následně uložit vygenerovaný text v PDF formátu.

0.2 Cíl práce

Cílem této bakalářské práce je analýza problematiky převodu OntoUML modelů do podoby formální dokumentace v přirozeném jazyce. Součástí tohoto bodu je prozkoumání možných vazeb a vztahů v daném typu modelů. Navazujícím cílem je také hledání a vymyšlení vhodných pro použití algoritmů sloužících k prohledávání modelů OntoUML, reprezentovaných ve formátu XMI a zvolení několika z popsaných možností pro implementaci výsledného programu. V praktické částí bakalářské práce cílem je zaměření se na navržení a implementaci aplikace pro převod OntoUML modelů do přirozeného jazyka. Součástí tohoto kroku je vytvoření grafického uživatelského rozhraní a implementace backendové logiky programu. Posledním a neméně důležitým cílem této závěrečné práce je otestování výsledné aplikace, popis provedených testů a shrnutí dosažených výsledků.

0.3 Struktura práce

Struktura práce odpovídá úkolům, které je nutné vykonat pro splnění cílů práce. V kapitole Teoretický základ a definice pojmů se nachází shrnutí a základní popis pojmů a termínů, které jsou používaný ve zbytku práce. Následující kapitola Analýza a návrh se zabývá zkoumáním problému a také zvolením technologií a metod, které mohou být použity pro úspěšnou realizaci cílů, které jsou stanoveny na začátku. Navíc používá koncepty jazyka UML pro podrobný popis budoucí aplikace. Kapitola Implementace obsahuje popis implementovaného programu, použitých metod a technologií a vysvětlení, proč a k jakým účelům jsou tyto technologie využity. Obsahem kapitoly Testování je popis použitých pro testování materiálů, jednotkových testů, použitých pro to zdrojů, které jsou nezbytné k ověření funkčnosti aplikace, a celkový souhrn výsledků provedených evaluací s příklady. Další kapitola Případová studie obsahuje ukázky práce programu s použitím velkého netriviálního OntoUML modelu. V poslední kapitole Závěr jsou popsany možnosti pro rozvoj a vylepšení aplikace v budoucnosti a také shrnutí celé provedené práce včetně povídání o výsledcích a přínosech pro autora této bakalářské práce.

Kapitola 1

Teoretický základ a definice pojmů

1.1 OntoUML

OntoUML neboli Ontological Unified Modeling Language(Ontologický Unifikovaný Modelovací Jazyk) je definován jako rozšíření UML(Ontologický Unifikovaný Modelovací Jazyk) a je založen na UML profilech. Je to konceptuální modelovací jazyk zaměřený na budování ontologicky dobře odůvodněných modelů.[1] Jelikož v současné době OntoUML není tak populární a často používaný jako UML, zatím toto rozšíření nemá tak hodně rešerši, dokumentace ani natolik velkou komunitu profesionálů, kteří by se aktivně zabývali tvorbou nových doplňkových nebo podpůrných aplikací.

Jako základní konstrikční prvky každého OntoUML modelu jsou použité stereotypy.

1.2 Stereotypy v OntoUML modelech

Každý ze stereotypů OntoUML modelu patří buď do skupiny třídních nebo do skupiny vztahových stereotypů. Třídní stereotypy lze popsat jako prvky, které přesně definují nějaký jev, předmět nebo například vlastnost. V závislosti na přesném stereotypu je dál možné určit, zda je prvek modelu rigidní, antirigidní, sortál nebo nonsortál. Stereotypy vazební jsou určeny především pro charakteristiku vztahů mezi různými třídam. Typ vazby vyjadřuje specifický druh vztahu. Například, typ Characterization se může vztahovat k nositeli a jeho rysu, rys je vždy existenčně závislý na svém nositeli, to znamená, že nemůže existovat bez toho, k čemu se podle vazby charakterizace vztahuje.

1.3 Format souboru XMI

XMI je standardem, který umožňuje popis objektů ve formátu XML, který je univerzálním způsobem reprezentace dat v celosvětové síti. Tento standard je aktivně používán pro objektově orientované programování, protože umožňuje standardní reprezentaci objektů a slouží k mapování objektů na formát XML, který není objektově orientovaný.[2]

Soubor typu XMI má jasně stanovenou strukturu a lze snadno pochopit, jak je možné přečíst a interpretovat data, reprezentovaná v souboru daného typu. Struktura XMI souboru pro OntoUML modely, vzniklé v aplikaci OpenPonk, je popsána v podkapitole 2.1.2.

1.4 Modelovací platforma OpenPonk

OpenPonk je metamodelovací platformou a pracovním nástrojem pro metamodeling. Daný nástroj je implementován v jazyku Pharo a je používán pro účely modelování, simulace, generování zdrojového kódu atd.[3] Tato bakalářská práce je zaměřená výhradně na práci s OntoUML modely, proto pro vytváření schémat těchto modelů je použit nástroj, který je v samotné aplikaci OpenPonk definován pod jménem *OntoUml UFO-A Editor*.

1.5 HTML

HTML je zkratka pro HyperText Markup Language, což je hypertextový značkovací jazyk, který se používá k vytváření textové šablony pro webové stránky nebo jiné dokumenty. HTML umožňuje formátování textu, přidávání grafů, zvuků, videí a také ukládání obsahu v textovém nebo ASCII souboru.[4] V této bakalářské práci daný formát je použit pro vytvoření šablony pro výsledný vygenerovaný PDF soubor. Díky použití tohoto jazyka je možné ve výsledku mít dobře strukturovaný dokument, který jistě má správné odsazení a zarovnání, a v závěru vygenerovaný dokument je dobře čitelný a lze se v něm snadno zorientovat.

1.6 Git

Git je distribuovaný systém správy verzí dostupný na všech běžných vývojových platformách prostřednictvím bezplatné softwarové licence.[5] Je určen hlavně pro ukládání všech změn, které jsou provedeny na projektu a snadnou orientaci mezi nimi. Nástroje, které zajišťují správu verzi projektu a umožňuji hromadný přístup několika programátorům současně a souběžný vývoj, jsou v současné době nezbytné pro vytvoření dobrého softwaru.

Git umožňuje vytvářet větve, průběžně ukládat provedené změny, provádět merge(sloučení změn ze dvou větví) a mnoho dalších operací, užitečných při práci na projektu.

1.7 Techniky transformace modelů

V současnosti je problém transformace modelů do textu již prozkoumán a existují určité metody exportu schémat do přirozeného jazyka. Dle studie o převodu UML modelů do textu, odpovídajícího standardu OMG SBVR(sémantika slovníku a pravidel podniku), lze techniky pro tvorbu takovýchto transformací rozdělit na dvě skupiny: automatické a poloautomatické.[6] Obě skupiny mají své výhody a nevýhody.

Do první skupiny patří aplikace, které pro svou činnost nevyžadují interakci s uživatelem, během transformace neumožňují provádět změny nebo nastavovat parametry, a proto obvykle nejsou přizpůsobené potřebám uživatele. Pro export tyto aplikace používají pravidla nastavená pomocí skriptu. Texty vygenerované s použitím těchto aplikací často poskytují méně detailů o vstupních modelech.

Skupina aplikací, které používají poloautomatickou transformaci, se od automatických liší tím, že pro provedení operací vyžadují interakci s uživatelem, který může nastavit potřebná pravidla a filtry. Pravidla pro export jsou nastavena přímo v kódu, proto změny pravidel vyžadují úpravu kódu. Popis vygenerovaný aplikacemi s poloautomatickou technikou práce obsahuje více podrobností.

V této bakalářské práci je použita technika poloautomatické generace textu. Volba vychází z toho, že OntoUML je rozsáhlejší než UML, proto je nutné mít možnost vygenerovaní popisu, který neosahuje zbytečné informace, ale zároveň je podrobný a odpovídá požadavkům uživatele.

1.8 Požadavky na jazyky pro převod modelů do textové podoby

Jazyky, které jsou určeny pro převod modelů do jejich textové podoby, byly již dříve navrženy, například, pro export schémat OMG UML. Dle konferenčního referátu [7] lze vyjádřit několik hlavních požadavků pro podobné jazyky. Mezi nimi patří:

- **Kontrola struktury:** jazyk by měl podporovat kontrolu struktury generovaného textu
- **Mechanizmy kontroly:** jazyk by měl umožňovat používání základních mechanismů kontroly toku dat(cykly, podmínky)
- **Kombinace kódu a textu:** jazyk by měl podporovat použití kombinace přirozeného textu a kódu, které lze v průběhu nahradit jiným textem
- **Systémové služby:** jazyk by měl umožňovat manipulaci s řetězci a komunikaci se systémovými službami, jako je například zjištění aktuálního času
- **Snadné pochopení:** jazyk by měl být snadno pochopitelný, tzn. výsledek by měl být jasný pro uživatele
- **Výraznost:** jazyk by měl být dostatečně expresivní, a je nutně, aby se dodržovala rovnováha mezi výrazností a snadností

1.9 Existující nástroje pro transformaci modelů do textu

V současné době neexistuje veřejně dostupný nástroj pro převod modelů OntoUML do přirozeného textu. Však jako příklad podobného softwaru lze uvést nástroje pro export modelů UML. Příkladem je popis systému pro transformaci jednoduchých UML modelů uvedený v jedné z pozičních zprav.[8] V článku je popsán software, jehož hlavním účelem je pomáhat učitelům a studentům při výuce UML.

Další příklad podobné práce je uveden v článku o nástrojích určených pro transformaci UML modelů.[9] Daná práce obsahuje porovnání existujících instrumentů včetně možností provedení transformace modelů na text.

Kapitola 2
Analýza a návrh

2.1 Softwarová analýza

V následující kapitole je popsána softwarová analýza budoucí aplikace pro transformaci modelů OntoUML do přirozeného jazyka, která vychází z požadavků vedoucího práce.

2.1.1 Požadavky na aplikaci

V této podsekci jsou popsany funkční a nefunkční požadavky na výslednou aplikaci. První skupina zahrnuje funkce a operace, které aplikace uživatelům umožní. Druhá popisuje podmínky, za nichž musí aplikace správně fungovat(například kompatibilita s jinými programy a podporovaná rozšíření souborů). Jelikož aplikace nepřistupuje k síti a nevyžaduje žádné osobní údaje od uživatele, není třeba řešit problémy, které by tyto funkce mohly způsobit.

2.1.1.1 Analýza funkčních požadavků

Na obrázku 2.1 jsou představeny všechny funkční požadavky na program, které vznikly po analýze a shrnutí požadavků od zadavatele zadání. Níž se nachází podrobný popis popis každého z nich.
F1: Nahrávání XMI souboru, vytvořeného v aplikaci OpenPonk, s popisem modelu OntoUML

Aplikace dokáže zobrazit složky a soubory v souborovém systému a umožní zvolení XMI souboru pro následnou transformaci. Po úspěšném načtení se dole objeví možnost zvolení režimu transformace a počáteční entity.

Aplikace musí omezit výběr pouze na soubory ve formátu XMI.

Následně aplikace musí zkontrolovat, zda nahraný soubor má v sobě validní OntoUML model, reprezentovaný pomocí souboru ve formátu XMI. Pokud je soubor platný, umožní pokračovat v nastavení režimů a filtrů pro transformaci, a také zobrazení náhledu a uložení výsledku konverze. V případě, že zvolený soubor neodpovídá pravidlům, okno volby souboru začne svítit červeně a tlačítka pro nastavení transformace budou nedostupná.

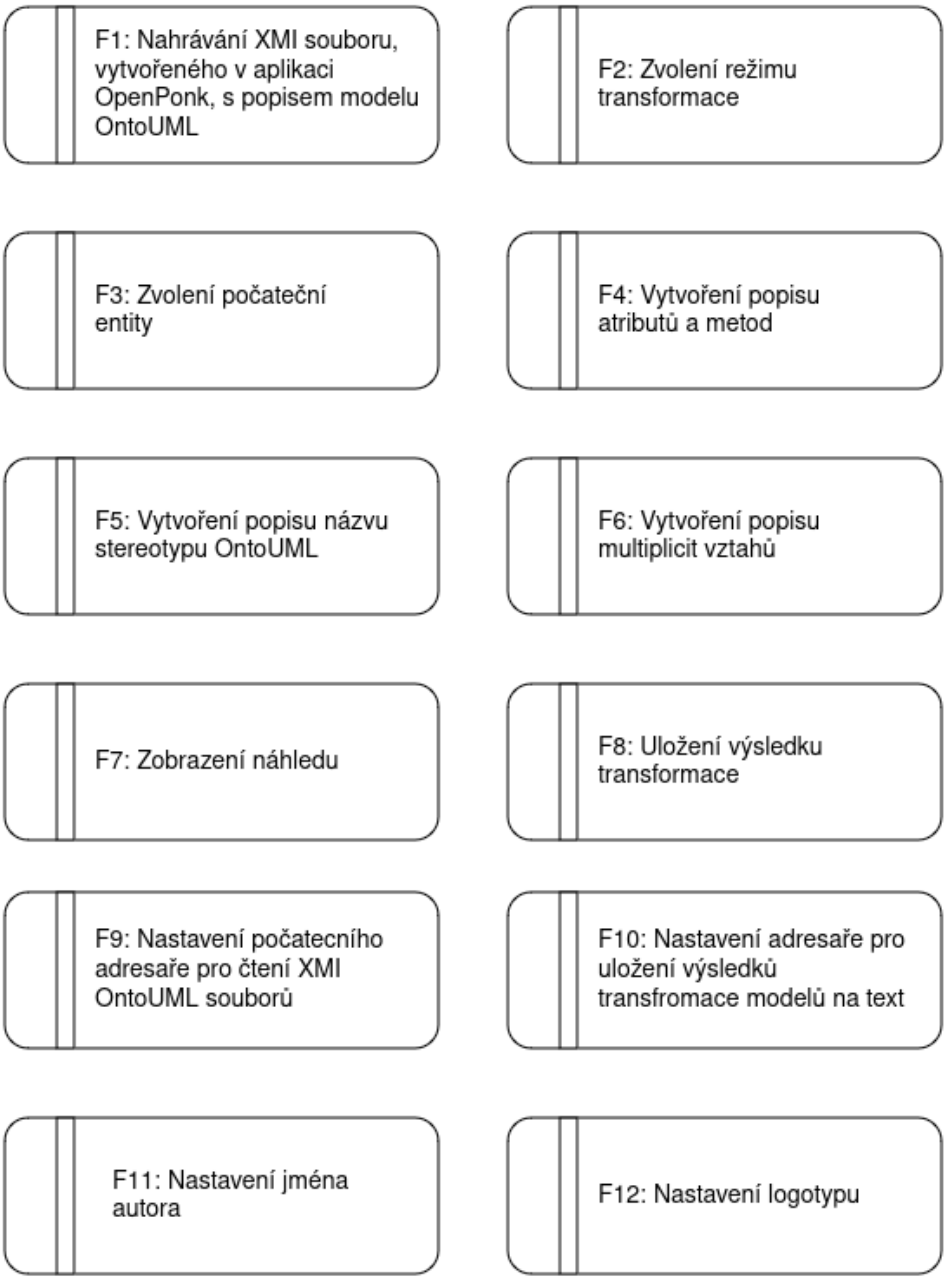
Tato funkcionalita je součástí záložky Transformer.

F2: Zvolení režimu transformace

Uživatel si bude moci zvolit jeden z předem definovaných režimů pro transformaci.

Mezi možnými režimy patří možnost popisu stromu dědičnosti(*Inheritance show*) a popis existujících stromů vztahů(*Deep associations*).

Ve výchozím nastavení je vybrán režim popisu dědičnosti a je automaticky použit, pokud uživatel režim ručně nenastavil.



Obrázek 2.1 Funkční požadavky

Tato funkcionalita je součástí záložky Transformer.

F3: Zvolení počáteční entity

Před spuštěním transformace si uživatel bude muset zvolit počáteční entitu, od které bude popis zahájen.

Entity jsou řazeny v abecedním pořadí podle jejich názvu.

Po nahrání XMI souboru je jako výchozí zvolena entita, která je první v abecedním pořadí.

Tato funkcionalita je součástí záložku Transformer.

F4: Vytvoření popisu atributů
Uživatel si může zvolit, zda chce ve výsledném textu zahrnout popis atributů, nebo nikoliv.
Tato funkcionalita je součástí záložky Transformer.

F5: Vytvoření popisu názvu stereotypu OntoUML
Uživatel si bude moci zvolit, zda chce do výsledného textu zahrnout popis názvů třídních stereotypů.
Tato funkcionalita je součástí záložky Transformer.

F6: Vytvoření popisu multiplicit vztahů
Uživatel bude schopen nastavit možnost popisu toho, kolik má mít vazba entit na každé stráně a také bude schopen dostat informaci o tom, jestli je nějaká vazba povinná nebo volitelná.
Tato funkcionalita je součástí záložky Transformer.

F7: Zobrazení náhledu
Po nastavení všech potřebných parametrů pro transformaci si uživatel může zobrazit náhled výsledného textového souboru, který představuje výsledek transformace.
Tato funkcionalita je součástí záložky Transformer.

F8: Uložení výsledku transformace
Po zvolení všech potřebných nastavení může uživatel uložit výsledek transformace do souborového systému počítače ve formátu PDF. Nastavení umístění pro uložení výsledného souboru je popsáno ve funkčním požadavku F10.
Tato funkcionalita je součástí záložky Transformer.

F9: Nastavení počátečního adresáře pro čtení XMI OntoUML souborů
Po spuštění aplikace a během další práce si uživatel může zvolit adresář, který bude výchozím bodem pro procházení souborového systému při výběru XMI souboru pro transformaci. Ve výchozím nastavení je to adresář Home.
Tato funkcionalita je součástí záložky Settings.

F10: Nastavení adresáře pro uložení výsledků transformace modelů na text
Po spuštění aplikace a během další práce si uživatel může zvolit adresář, do kterého se bude ukládat výsledek transformace. Ve výchozím nastavení se ukládá do adresáře Home. Další možnosti zahrnují:

- Výběr jiného adresáře ze souborového systému jako výchozího
- Volbu, aby se aplikace vždy zeptala uživatele, do jakého adresáře chce uložit vygenerovaný soubor

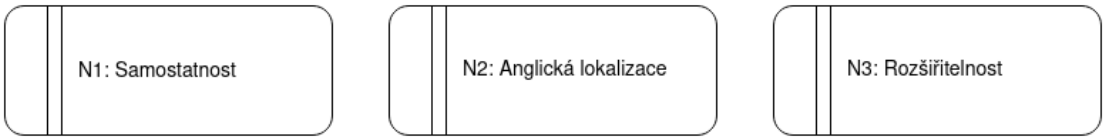
Tato funkcionalita je součástí záložky Settings.

F11: Nastavení jména autora
Po spuštění aplikace a během další práce si uživatel může zadat jméno autora, které bude ve výsledném souboru uvedeno pod názvem transformovaného OntoUML modelu.
Tato funkcionalita je součástí záložky Settings.

F12: Nastavení logotypu
Po spuštění aplikace a během další práce si uživatel může zvolit soubor s obrázkem(PNG/JPEG), který bude použit jako logotyp v levém horním rohu první stránky výsledného transformovaného textu.
Tato funkcionalita je součástí záložky Settings.

2.1.1.2 Analýza nefunkčních požadavků

Seznam nefunkčních požadavku je zobrazen na obrázku 2.2. Tato podkapitola obsahuje podrobný popis každého požadavku ze seznamu.



Obrázek 2.2 Nefunkční požadavky

N1: Samostatnost
Aplikace je samostatná, nevyžaduje integraci s jinými programy a lze ji spustit v samostatném okně bez nutnosti spouštět jakýkoliv jiný software.

N2: Anglická lokalizace
Texty a fráze, které jsou součásti grafického uživatelského rozhraní, včetně všech nápisů na tlačítčích a názvu okna samotného, jsou uvedeny v anglickém jazyce.

N3: Rozšiřitelnost
Do aplikace lze přidávat nové režimy, nastavení a další komponenty, potřebné pro požadovanou transformaci modelu OntoUML do přirozeného jazyka.

2.1.2 Analýza vstupních dat ve formátu XMI

Tato podkapitola se primárně zabývá analýzou dat, která má přijímat výsledný software jako vstup. Data, která bude software přijímat jako vstup, musí mít formát souboru XMI a musí být vygenerovana pomocí aplikace OpenPonk pro správný, ideálně verifikovaný OntoUML model, vytvořený pomocí této aplikace. Jelikož takový soubor vždy obsahuje přesnou strukturu, což je důležité pro analýzu a organizaci dat do vhodných tříd a struktur, v dalších podkapitolách se popisuje struktura informací v souboru XMI.

2.1.2.1 Identifikace tříd a vazeb

Informace o použitých stereotypech a jejich identifikačních údajích je uvedená vždy na konci souboru a má následující strukturu(hranaté závorky označují volitelnou část):

1. **OntoUML**:*Typ-třídy-nebo-vazby*
2. **xmi:id**="*Unikátní-id-xmi-elementu*"
3. **base_Element**="*Unikátní-id-třídy-nebo-vazby*"
4. [**essential**="*pravdivostní-hodnota*"]
5. [**inseparable**="*pravdivostní-hodnota*"]
6. [**immutablePart**="*pravdivostní-hodnota*"]
7. [**immutableWhole**="*pravdivostní-hodnota*"]

Prvky struktury, která je popsaná výše, jsou mezi sebou oddělený mezerami, a celý vzniklý řádek je celkově obalen špičatými závorkami.

Jako typ třídy nebo vazby v položce **OntoUML** je vždy uveden platný název jednoho z OntoUML stereotypů. Do seznamu stereotypů tříd patří *Kind, SubKind, Phase, Role, Collective, Quantity, Relator, Category, PhaseMixin, RoleMixin, Mixin, Mode, Quality*. Vazební stereotypu mohou mít následující názvy: *Formal, Material, Mediation, Characterization, Derivation, Structuration, ComponentOf, Containment, MemerOf, SubquantityOf, SubCollectionOf*.

Prvek **xmi:id** reprezentuje identifikátor elementu, který patří do struktury daného XMI souboru. Daný typ identifikace je užitečný pro aplikaci OpenPonk samotnou, ale není používán jako součást popisu vztahů tříd a jiných vlastností OntoUML modelu, proto toto id nehraje významnou roli při zpracování dat ve výsledné aplikaci a není nutné ho ukládat.

Položka **base_Element** naopak obsahuje identifikační řetězec pro danou entitu nebo vazbu. Ten se používá zejména při popisu různých vlastnosti tříd a vztahů mezi nimi. Prvek **base_Element** je vždy unikátní pro každou položku a lze pomocí něj jednoznačně určit, na kterou třídu je nutně se odkázat.

Hodnota prvků z bodů 4 až 7 označena jako *"pravdivostní-hodnota"* reprezentuje pravdivostní hodnotu, která vyjadřuje, jestli je daný parametr pro tento element nastaven. V závislosti na nastavení vazby tento parametr má hodnotu buď „true"(parametr je nastaven) nebo není uveden(když parametr není nastavený). Tyto volitelné položky(4-7) se používají pouze u vztahů typu „celek-část". Do této skupiny jsou zařazeny vazební stereotypy *ComponentOf*, *SubCollectionOf*, *MemberOf*, *Containment* a *SubQuantityOf*. Najednou mohou být použity jen parametry ve dvojicích: *essential* lze použit s *inseparable* a nebo s *immutableWhole*, *inseparable* je možné použit spolu s *essential* a *immutablePart*. Důvod je ten, že *essential*, respektive *inseparable*, charakterizují rigidní část, resp. celek, zatímco *immutablePart* a *immutableWhole* jsou vlastnostmi antirigidních části a celků.

2.1.2.2 Prvky bez typu

V některých OntoUML modelech se kromě jasně určených prvků vyskytují také elementy, které nemají žádný konkrétní typ. Pro účely transformace lze s takovými prvky zacházet jako s třídami, protože mohou obsahovat atributy a mít různé vazby s jinými třídami. Jediný rozdíl spočívá v tom, že nemají specifický typ.

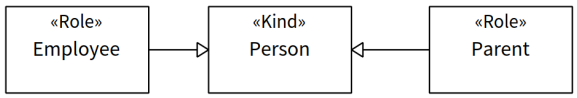
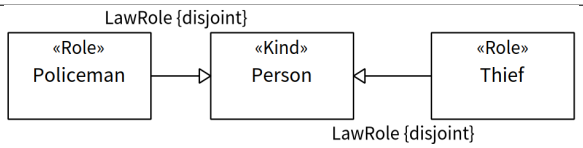
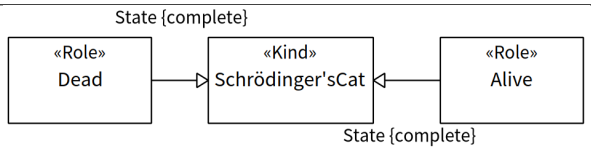
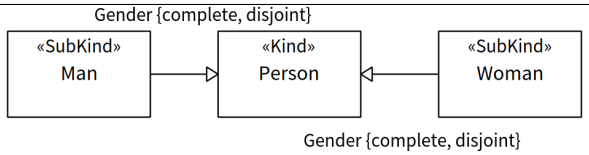
Vazby v těchto modelech také nemusí mít stereotypy, protože mohou být využity k vyjádření a popisu zvláštních vztahů. Příkladem takové zvláštní vazby je vztah mezi běžnou entitou a entitou, která slouží jako kontejner pro data(například atributy). Takové vazby také mají multiplicitu na obou stranách a mohou mít své specifikované jméno.

2.1.2.3 Vlastnosti vztahů dědičnosti

Obecně pro každý OntoUML model platí, že libovolná vazba vyjadřující vztah dědičnosti mezi třídami, může být popsána pomocí sjednocení dědicích entit do skupin. To umožňuje upřesnit typ a význam těchto vazeb. Tento nástroj pro sdružení tříd, které mají společný nadtyp, se nazývá *generalization set* nebo *množina nadtypu*.

Tento instrument lze rozšířit použitím speciálních meta-atributů, které jsou vyjádřeny klíčovými slovy *„disjoint"* a *„covering"*(nebo *„complete"*). Prvním pojmem se označuje jev, když žádná instance nemůže mít víc než jeden podtyp. Druhý pojem vyjadřuje to, že uvedená množina podtypů popisuje celou množinu možnosti pro nadtyp, tzn. instance nadtypu je vždycky zároveň instanci jednoho s podtypu ze skupiny „complete". V tabulce 2.1 jsou uvedeny příklady pro lepší pochopení vlivu pojmů „disjoint", „complete".

V datech reprezentovaných pomocí souboru XMI je množina nadtypu uvedená jako packageElement s xmi:type="uml:GeneralizationSet". Příklad struktury pro tento element je uveden v kódu 2.1. Pole *generalization* obsahuje identifikátory entit(identifikátory jsou v daném příkladě

Flagy	Příklad	Význam
žádný		Člověk může být zároveň zaměstnancem a rodičem. Nemusí být žádným z nich. Například je mladým podnikatelem.
disjoint		Policista zadržel zloděje, proto nemůže to být stejný člověk. Ale nemusí to být ani jeden z nich. Například, je svědkem.
covering		Schrödingerova kočka má jen dvě možnosti. Je živá nebo mrtvá. Však pro nás obě možnosti mohou platit zároveň.
disjoint, covering		Od narození je člověk za mužem nebo ženou. Identifikovat ho jako oba typy zároveň není možné (až na zvláštní výjimky).

Tabulka 2.1 Příklady použití disjoint a complete flagů s vysvětleními

zkrácené pro jednoduchost čtení), které do této množiny nadtypu patří, *isCovering* a *isDisjoint* jsou nastavené na „true“, resp. „false“(případně nemusí být uvedeny pro „false“), pokud množina má, resp. nemá, vlastnosti „covering“, „disjoint“.

Pak třídy, které jsou součásti nějaké množiny nadtypu, mají ve svém popisu pole general, označující identifikátor svého nadtypu, a také pole generalizationSet, které označuje množinu nadtypu, do které jsou zařazeny.

Pokud třída nepatří do žádné množiny nadtypu, ale je podtypem, pak obsahuje jen pole general. Pokud není ani podtypem, pak neobsahuje žádnou z těchto možnosti.

```
<packagedElement xmi:type="uml:GeneralizationSet" xmi:id="gs"
    generalization="g1␣g2" isCovering="true"
        isDisjoint="true" name="Gender"/>
```

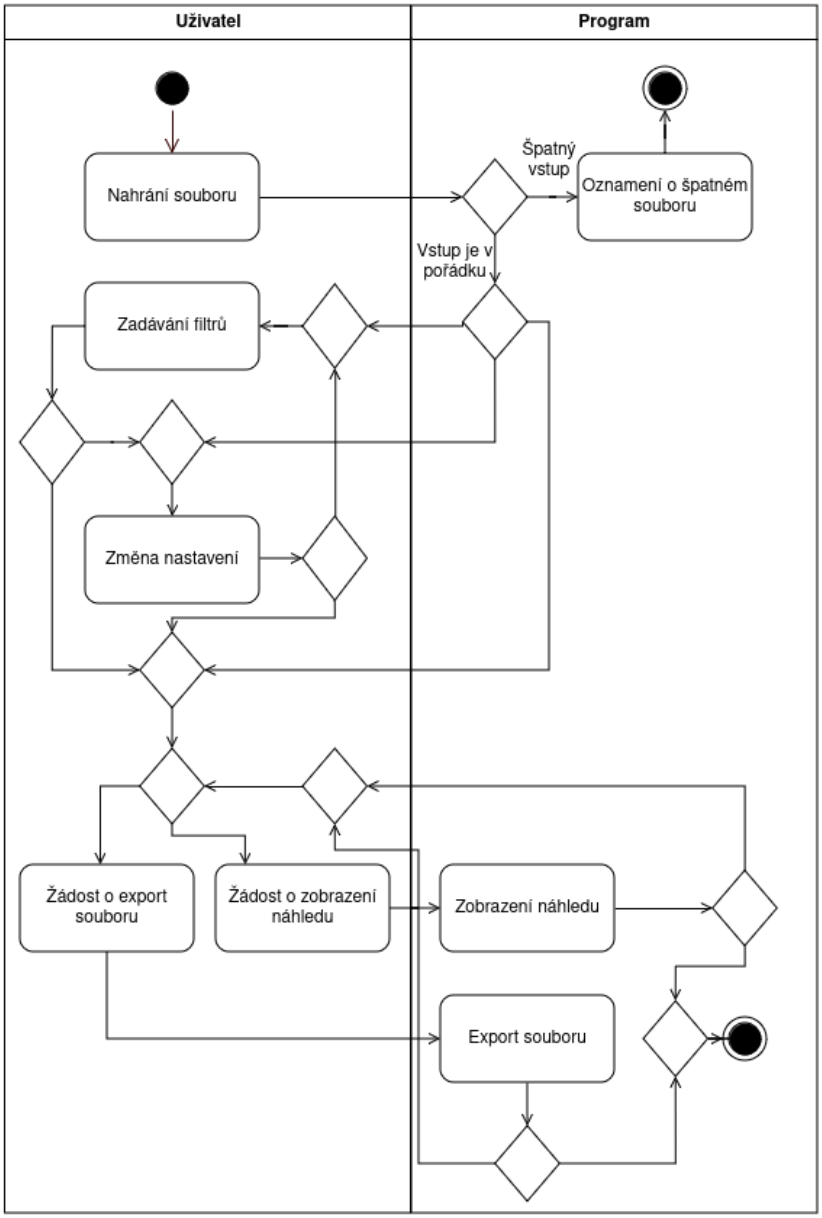
Výpis kódu 2.1 Generalization set in XMI OntoUML model

2.1.3 Struktura a procesy systému

Na základě provedené analýzy popsané v předchozích podkapitolách je možné navrhnout budoucí systém. Pro návrh a popis aplikace je použit diagram tříd UML, uvedený v příloze A. UML je modelovací jazyk, který je v současné době de facto standardem, který poskytuje způsoby

vytváření vizuální reprezentace dat.[10] Existují dva hlavní typy těchto diagramů, a to jsou diagramy struktur a diagramy chování. Diagram tříd je patří do první skupiny a popisuje třídy a vztahy mezi nimi.

Dalším diagramem, který je pro danou práci navržen je diagram aktivit. Ten spadá do druhé skupiny, což znamená, že jde o diagram chování. Hlavním důvodem použití tohoto druhu diagramů je globální uspořádání atomických částí chování, které se nazývají aktivity.[11] Diagram popisující základní chování systému při interakci s uživatelem je uveden na obrázku 2.3.



Obrázek 2.3 Základní diagram aktivit

2.2 Výběr technologií

Tato podkapitola obsahuje popis technologií, metod a nástrojů, použitých pro implementaci výsledné aplikace.

2.2.1 Zvolení programovacího jazyka

Pro implementaci aplikace je nutné zvolit vhodný programovací jazyk. Jelikož aplikace musí pro pohodlí uživatele obsahovat nejen backednovou, ale i frontednovou část, musí zvolený jazyk dovolovat jednoduše vytvářet grafický uživatelský interface. Pro tvorbu backednové části této aplikace jsem zvolila programovací jazyk C++ s použitím Qt frameworku. Uživatelské rozhraní je implementováno pomocí jazyka QML.

Volba daného programovacího jazyka a frameworku není náhodná. Požadavky, které jsou uvedeny v 1.8 lze při použití C++, Qt a šablony HTML snadno splnit. Dalším důvodem volby Qt frameworku pro implementaci programu je to, že Qt je rozsáhlý, má velkou komunitu a podrobnou dokumentaci a navíc umožňuje multiplatformní vývoj. Další výhodou jsou moje skoro dvouleté zkušenosti s tvorbou aplikaci pomocí tohoto frameworku.

Pro napsání jednotkových testů, které jsou nezbytné pro snížení počtu chyb ve výsledném softwaru, je použit framework Qt Test, který umožňuje testování aplikací, vytvořených na základě Qt frameworku. Tento způsob implementace unit testu je také další výhodou, proto volba tohoto frameworku je dobrá.

2.2.2 Výběr vývojového prostředí

Celý program, to znamená jeho backendová i frontendová část, jsou vytvořeny ve vývojovém prostředí Qt Creator. Je to mocný nástroj, který je navržen specialně pro programování aplikaci využívajících Qt framework. Qt Creator podporuje práci s distribuovaným verzovacím systémem Git(při propojení s Gitem ukazuje číslo a autora commitu, umožňuje porovnávání kódu v uživatelském rozhraní a další činnosti). Více informací o daném vývojovém prostředí lze nalézt v [12].

Navíc pro tvorbu grafického uživatelského rozhraní v Qt frameworku existuje několik nástrojů, které podporují různé způsoby vytváření interface. Prvním způsobem je ruční psaní kódu do souborů s rozšířením QML. Při používání tohoto typu programátor často používá konstrukty jazyka JavaScript, protože Qt obsahuje část syntaxe, shodné se syntaxi Javascriptu. Tento typ tvorby interface je užitečný pro případy tvorby složitějšího grafického rozhraní s použitím většího množství logiky nebo komplikovanějšího kódu pro vytváření GUI.

Další možností tvorby uživatelského rozhraní je použití Qt Designer tool, který funguje podle principu WYSIWYG(česky *co vidíte, to dostanete*), to znamená umožňuje tvorbu GUI pomocí přetahování grafických prvků v uživatelském rozhraní vývojového prostředí. Tento způsob je dobrý pro vytváření jednodušších ze strany uživatelského rozhraní aplikace, protože programátor okamžitě vidí, jak bude aplikace vypadat, a snadno nadefinuje, co bude obsahovat. Tento typ tvorby grafického uživatelského intefacu je použit v aplikaci pro převod OntoUML modelu, protože tato aplikace obsahuje nejkomplikovanější logiku uvnitř backendové C++ části, a na stráně GUI jsou většinou potřeba jen jednoduchá tlačítka nebo složitější kombinovaná pole, které ale jsou naplněné již v backendové části.

2.2.3 Volba nástroje pro automatizaci překladu programu

Programy vytvořené v jazyce C++ a používající Qt Framework definovat postup buildu dvěma hlavními způsoby.

Jako první lze zvolit klasický CMake, který je známý pro všechny programátory v jazyce C a C++. Hlavní výhodou CMake je to, že díky tomu, že tento software používá vysoký počet specialistů, proto lze najít hodně materiálů a návodů jak na internetu, tak i, například, v tištěné literatuře. Navíc tento instrument je vyučován i na ČVUT FIT, například, v předmětu Programování a algoritmizace a je základem pro každého studenta této fakulty. Další výhodou, je to, že CMake je velmi mocným nástrojem, který dovoluje nadefinovat, popsat a vytvořit skoro všechno, pokud je napsán znalcem. Naopak nevýhodou je, že CMake je docela komplikovaný na prozkoumání a není snadno se naučit psát dobré a hlavně multifunkční kód pomocí CMake. Navíc, co se tyče kompatibility s Qt Creatorem, CMake pro toto vývojové prostředí není primárním způsobem generování make souborů, proto vyžaduje hodně manuální práce, jako je ruční přidávání souborů CPP a H do CMake.

Druhou cestou pro automatizaci generování souborů make je použití speciální analogie CMake, která se nazývá QMake. QMake je hlavním nástrojem, který se používá ve vývojovém prostředí Qt Creator. Hlavní jeho výhodou, je to, že díky tomu, že je určen pro práci s Qt frameworkem, nevyžaduje tolik manuálního přidávání souboru a dalšího psaní kódu, protože umí tyto definice vytvářet automaticky. Mezi dalšími výhodami patří, například, to, že tento nástroj je velmi snadný pro pochopení a učení se, proto dovoluje rychle s nim začít pracovat dokonce i tomu, kdo dříve neměl s tím žádnou zkušenost. Co se tyče negativních vlastností, se kterými se programátor setká při zvolení tohoto způsobu provedení buildu projektu, lze říct, že kvůli snadnosti tohoto způsobu už není tak mocný a rozsáhlý jako CMake, tedy umožňuje nadefinovat a vytvořit mnohem méně věcí. Navíc QMake je méně běžný, a bez ohledu na to, že má dobrou, podrobně popsanou dokumentaci, nemá tak velkou komunitu uživatelů, proto má i méně informace, například, na internetu. Další nevýhodou je, že je obtížně intergrovatelný pro knihovny třetích stran, protože velmi málo takových knihoven používá také QMake, a CMake není použitelný v jednom projektu spolu s nástrojem QMake.

Během procesu finálního rozhodování mezi těmito dvěma nástroji byly zvážené všechny pozitivní a negativní vlastnosti obou způsobu generování souborů make. Nakonec bylo rozhodnuto, že QMake líp odpovídá požadavkům na tuto aplikaci, protože nemusí obsahovat složitou logiku a je lepší, pokud více práce ohledně nadefinování způsobu vytváření buildu programu bude ležet na tomto nástroji, než na programátorovi samotném. Výsledný software požaduje mít jednoduchou logiku tvorby buildu a jednoduché uživatelské rozhraní. Nejnáročnější částí vzniklé aplikace je backend, proto není potřeba zbytečně komplikovat práci dalších lidí, kteří se budou o program starat a rozvíjet ho.

2.2.4 Metoda generování PDF souboru

Aplikace musí generovat soubor PDF, který obsahuje textovou analogii OntoUML modelu ze vstupu programu. Pro vytvoření tohoto souboru jsem rozhodla použit HTML model jako základ budoucího textu. Šablona s HTML syntaxi obsahuje pravidla pro vzhled a obsah výsledného textu. Vzor se řídí obvyklými pravidly hypertextového značkovacího jazyka, to znamená, že používá stejné tagy jako každý soubor s HTML obsahem. Příklady tagů použitých k vytvoření HTML šablon pro generování PDF souborů v této práci jsou uvedeny v následujícím seznamu:

- **br** – odřádkování
- **h*X*** – velký tučný text, použitý pro nadpisy, kde X je číslice od 1 do 6
- **i** – text psaný kurzovou
- **li** – položka seznamu (typ označení zaleží na tom, čím jsou položky s tímto tagem obaleny)
- **ul** – odrážkový seznam (obalení pro tagy typu *li*)

Podrobnější popis použité HTML šablony je uveden v podkapitole Implementace HTML šablony pro vytvoření textu.

Kapitola 3
Implementace

Náplní této části bakalářské závěrečné práce je hlavně popis zvolených technologií pro tvorbu softwaru, architektury programu, použitých návrhových vzoru. Nedílnou součástí je také deskripce zvolených instrumentů, které slouží k naprogramování aplikace pro transformaci modelů OntoUML na jejich textovou reprezentaci.

3.1 Tvorba uživatelského rozhraní

Tato podkapitola obsahuje popis uživatelského rozhraní, zejména tvorby designu a vysvětlení postupu vytváření tohoto GUI.

3.1.1 Návržení vzhledu modelu grafického uživatelského rozhraní

Před začátkem implementace frontendové části budoucí aplikace v kódu je vždy důležité se zamyslet nad tím, jak zákazník, nebo v případě této závěrečné práce vedoucí, by chtěl, aby výsledná aplikace vypadala. Hlavním principem, který by měl být použit při tvorbě grafického uživatelského rozhraní, je nejprve se zaměřit na to, co potřebuje uživatel, nikoliv na technologie, které chce použit programátor. Vývojář by měl odpovědět na otázky, které se týkají zkušenosti uživatele, jeho potřeb a běžných úkolů, které by měla aplikace umožnit provést.[13]

To se týká hlavně nastavení, jako je například výběr souboru s modelem OntoUML v souborovém systému pro následnou generaci jeho textové podoby. Aby se minimalizoval počet chyb ze strany uživatele, lze použít následující řešení: uživatel bude volit soubor přímo v novém okně, které zobrazí pouze soubory vhodné pro použití jako vstupní data. K tomu je nutné mít tlačítko, které vyvolá toto okno. Pokud by byl nahraný soubor strukturálně neplatný, je nutné to uživateli oznámit pomocí obarvení rámečku s cestou k souboru červenou barvou, která signalizuje chybu.

Navíc při znalosti toho, že uživatel si bude chtít textový výsledek transformace prohlédnout a dál ho uložit, je potřeba do aplikace přidat příslušná tlačítka.

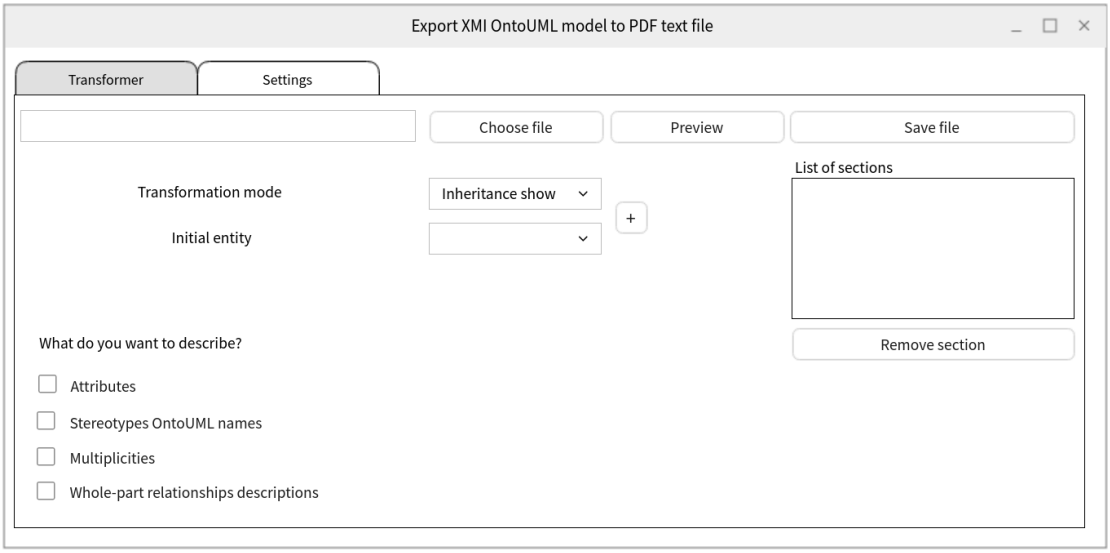
Na základě tohoto principu a existujících funkčních a nefunkčních požadavků na aplikaci, následně byla provedená analýza požadovaných UI prvků výsledného programu.

Zaprvé je důležité vzít v úvahu to, že žádný software není bez chyb, ale také to, že se i uživatel může nějakým svým neočekávaným chováním dostat do chybného stavu.

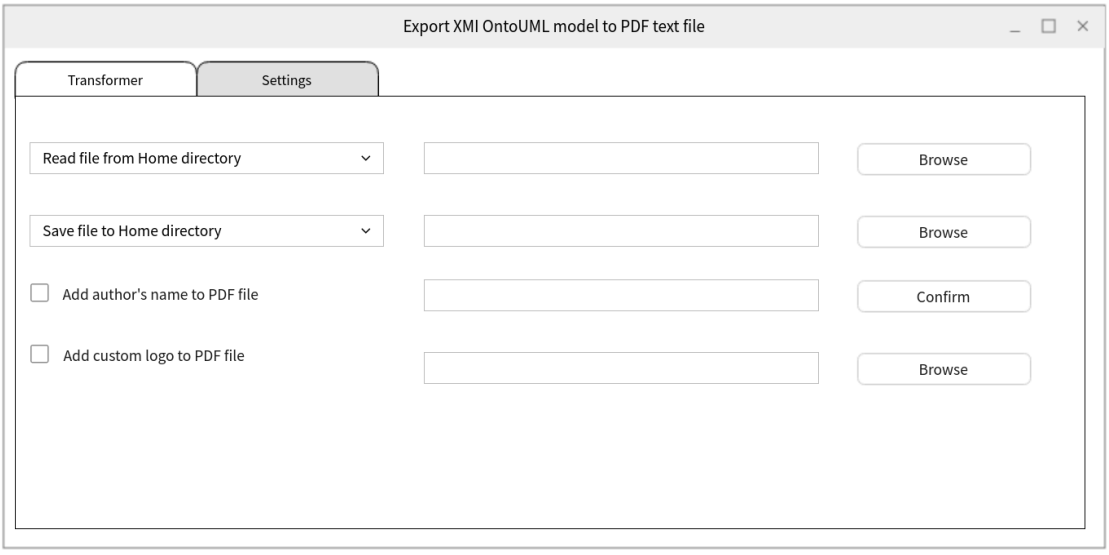
Pro zvolení nastavení z takových skupin, kde lze použit jen jednu konfiguraci z celé skupiny najednou, jako řešení bylo zvoleno přidání do aplikace elementů, které se nazývají ComboBoxy(nebo kombinovaná pole). Tento ovládací prvek grafického uživatelského rozhraní dovoluje vybírat jeden z připojeného seznamu, a proto přesně odpovídá požadavkům uživatele.

Dál, po zkoumání možností tvorby GUI pro nastavení, která mohou být zvolená vícekrát, se ukázalo, že dobrou volbou je přidání do aplikace speciálního pole, do kterého lze přidat neomezený počet nastavení určitého typu. Tímto způsobem si uživatel může prohlédnout všechna dříve provedená nastavení, a také bude mít možnost smazat položky z tohoto seznamu, proto je nutné přidat tlačítko pro mazání.

Po provedení této analýzy již lze namodelovat to, jak by výsledná aplikace měla přibližně vypadat. Pro tvorby takového mock-up modelu jsem zvolila použití online nástroje, který nabízí základní prvky, modelující elementy grafického uživatelského rozhraní, MockFlow. Pomocí tohoto nástroje lze vytvořit model GUI a následně ho použít v procesu tvorby frontendové části aplikace pomocí kódu. Výsledný mock-up model je znázorněn na obrázcích 3.1 pro obrazovku záložky *Transformer* a 3.2 pro obrazovku záložky *Settings*.

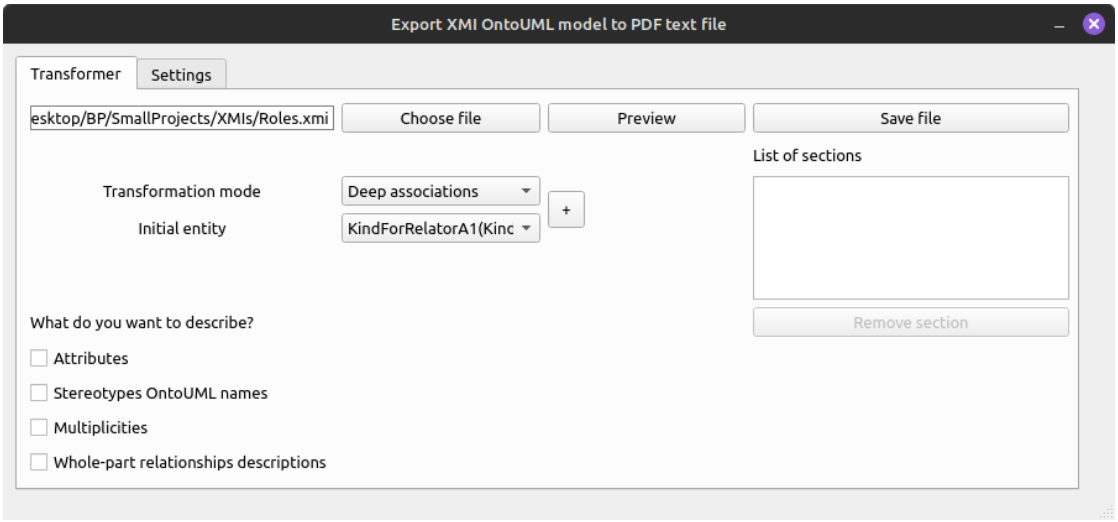


Obrázek 3.1 Mock-up model GUI aplikace: obrazovka *Transformer*

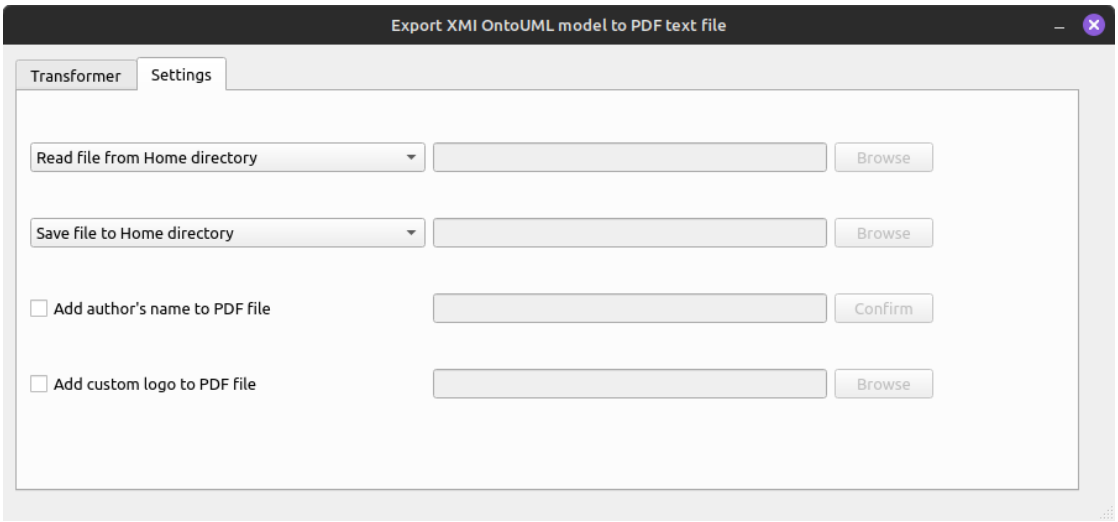


Obrázek 3.2 Mock-up model GUI aplikace: obrazovka *Settings*

Výsledný frontend aplikace pro transformaci OntoUML modelu na text, vytvořený v aplikaci pro vývoj uživatelského rozhraní Qt Designer vypadá jako na obrázcích 3.3 a 3.4.



Obrázek 3.3 Výsledná obrazovka *Transformer* v aplikaci



Obrázek 3.4 Výsledná obrazovka *Settings* v aplikaci

3.2 Implementace backendové logiky pro převod modelu na text

Dalším krokem při implementaci aplikace pro převod OntoUML modelu do textové podoby ve formátu PDF je návrh způsobu konverze modelových struktur ve formátu XMI do přirozeného textu. Je důležité zdůraznit, že každý model má vlastní strukturu a množina typů, které může model obsahovat, je vždy konečná. Díky tomu lze celý problém rozdělit na menší podproblémy.

3.2.1 Obecná pravidla a filtry

Na začátku je možné nadefinovat to, jak bude fungovat převod jednoduchých tříd, protože pro popis budou v každém režimu použity třídy a jejich jména. Transformace základních informací o entitě bude ve výsledku vždy obsahovat jméno této entity napsané kurzívou.

Pak, pokud má entita, která musí být převedená na text, přesně určený stereotyp, a je zaškrtávací pole, které označuje zapnuti filtru *Stereotypes OntoUML names*, zaškrtnuto, vedle jména entity v kulatých závorkách bude uvedeno jméno samotného OntoUML stereotypu. Lze říct, ze filtr *Stereotypes OntoUML names* je obecným filtrem, který lze použít v obou režimech transformace.

Dalším možným obecným filtrem je „Attributes“. Pokud je tato možnost v aplikaci zaškrtnuta, jsou ve výsledném textu popsány všechny atributy, které každá s popisovaných entit má. Popis této části do textu přidává větu ve formátu: „List of the attributes for the *jméno-třídy* class: *výčet-atributů*“. *Jméno-třídy* označuje místo, kde je nasáno konkrétní jméno popisované třídy, *výčet-atributů* zastupuje názvy atributu, oddělené čárkou a mezerou. Celá věta má na konci tečku.

Dalším obecným pravidlem pro všechny módy je to, že každá sekce, která se v popisu vyskytuje, začíná názvem režimu práce a pokračuje slovem „for“. Končí jménem třídy, která je pro danou sekci počáteční. Tato věta je napsána tučně a označuje název sekce.

3.2.2 Transformace pro různé režimy

Následně je důležité nadefinovat to, jak bude transformovaný text vypadat, pokud budou nastaveny různé režimy práce aplikace.

3.2.2.1 Transformace pro režim Inheritance show

První možný režim, který definuje způsob transformace je mód „Inheritance show“. Je určen především k popisu vztahů dědičnosti v daném OntoUML modelu. Při zvolení tohoto režimu lze použít pouze dva rozšiřující filtry: „Stereotypes OntoUML names“, „Attributes“. Ostatní možnosti jsou určené pro popis podrobnosti o asociativních vztazích, a vztahy dědičnosti do této skupiny nepatří.

Popis v tomto režimu vždy začíná entitou, zvolenou jako počáteční v poli „Initial entity“. Popis vždy začne základní informací o entitě(jménem, které ja napsáno podle pravidel, nadefinovaných v předchozí podsekci).

Pokud má daná entita podtřídy, popis pokračuje frází „*is a supertype for*“. Dál bude následovat výčtem, obsahujícím názvy podtříd, které budou oddělený čárkou a mezerou. Za posledním prvkem výčtu se umístí tečka.

Nějaké z podtříd mohou být sjednoceny pomocí tzv. *generalization set*, který popisuje nějakou skupinu tříd a její vlastnosti, důležité pro deskripci dědičnosti. V tomto případě dál popis bude obsahovat slovo „*Classes*“/„*Class*“(v závislosti na tom, jestli se v daném generalization set nachází víc než 1 třída) s výčtem tříd, tvořících danou množinu. Věta pokračuje frázi „*is/are in generalization set*“, a pokud množina má nějaký název, končí tímto názvem s tečkou, jinak jen tečkou samotnou. Dál bude uvedena informace o tom, jestli je tento generalization set „disjoint“/„covering“(tyto vlastnosti jsou popsané v podsekci Vlastnosti tříd) a vypadá to ve větě jako „This set is disjoint and covering“. Opakem pojmu *disjoint* je pojem *overlapping*, opakem pojmu *complete* je *incomplete*.

Pokud popisovaná entita již nemá žádné podtřídy, pak popis obsahuje větu začínající jménem entity a končící frází „*has no related subtypes*“. Podrobnější informace o tom, jak se prochází model při spouštění transformace je uvedená v podsekci Algoritmus procházení modelu v režimu „Inheritance show“.

Pokud je zapnut filtr „Attributes“, a popisovaná entita má atributy, popis pokračuje deskripcí těchto atributů podle pravidel uvedených výše.

3.2.2.2 Transformace pro režim Deep associations

Druhou možností při volbě režimu je „Deep associations“. Tento režim je určen především k detailnímu popisu vztahů mezi entitami v modelu OntoUML. Při volbě tohoto režimu se předpokládá, že s nim mohou být použity filtry, které rozšiřují výsledný popis. Do této skupiny patří možnosti „Stereotypes OntoUML names“, „Attributes“, „Multiplicities“, „Whole-part relationships descriptions“. První a druhý filtr s tohoto seznamu jsou obecné a jsou popsány v podsekci Obecná pravidla a filtry.

Dalším filtrem je „Multiplicities“. Tato možnost je slouží k popisu multiplicit, které ukazují, kolik prvků typy druhé entity může mít první entita entita, a naopak. První a druhá entita jsou třídy, které jsou spojeny asociativní vazbou. Ve výsledném textu je tato část napsána ve dvou větách, kdy druhá následuje po první, a obě mají tvar „Type *jméno-první-třídy* has *popis-multiplicity jméno-druhé-třídy*“, kde *jméno-první-třídy* a *jméno-druhé-třídy* jsou jména tříd, vytvořená podle obecných pravidel, a *popis-multiplicity* je jedná z následujících frázi:

- exactly 1 type
- from one to Y types
- no or has only one
- any number of types
- maximum Y types
- X or more types
- from X to Y types

X, resp. Y, v tomto případě znamenají minimální, resp. maximální, hranici multiplicity na jedné stráně asociace.

Posledním používaným filtrem, který je důležité popsat, je „Whole-part relationships descriptions“. Tato možnost slouží k podrobnějšímu popisu vztahů celek-část, které jsou v grafické reprezentaci modelu OntoUML znázorněny jako kosočtverec. V případě vyplněného kosočtvercového symbolu vypadá popis takto: „*Jméno-části* is part of *jméno-celku* but *jméno-části* can exist without *jméno-celku*“. Naopak, pokud je prázdný, výsledný text obsahuje frázi ve tvaru „*Jméno-části* is integral part of *jméno-celku*“.

Hlavní částí OntoUML modelu, která je popisována při spouštění transformace v režimu „Deep associations“, je deskripce asociativních vztahů uvedených v modelu. Tyto vztahy se liší stereotypy, proto je třeba provádět konverzi podle toho, jaký má daný vztah stereotyp. V tabulce 3.1 jsou uvedeny názvy stereotypu, jejich textové analogie(tvar ve výsledném textu) a příklady, jak by mohl vypadat popis toho či jiného vztahu.

Podrobnější informace o tom, jak se prochází model při spuštění transformace jsou uvedeny v podsekci Algoritmus procházení modelu v režimu „Deep associations“.

3.2.3 Algoritmy procházení OntoUML modelu pro různé režimy

Tato podkapitola obsahuje podrobnosti týkající se algoritmů převodu OntoUML modelů na text s využitím různých režimů. Obecně platí, že každý OntoUML model lze reprezentovat jako graf, a tvar tohoto grafu závisí na požadovaném výsledku.

Stereotyp	Textový popis v transformaci	Příklad
ComponentOf	is component of	*CarEngine* is component of *Car*.
Characterization	characterizes	*Weight* characterizes *Product*.
Mediation	mediates	*Enrollment* mediates *Person*. *Enrollment* mediates *Institution*.
DomainFormal	can be compaired with	*Person* can be compared with *Person*. Relationship has name „older-than“.
Material	is united with	*Husband* is united with *Wife*. Relationship has name „married to“.
MemberOf	is member of	*Musician* is member of *Orchestra*.
SubCollectionOf	is subcollection of	*TeachingStaff* is subcollection of *SubjectDept*.
SubQuantityOf	is subquantity of	*Water* is subquantity of *Tea*.
Jiný/žádný	is in relation with	*Profile* is in relation with *ProfileData*.

Tabulka 3.1 Popis způsobu převodu vazebních stereotypů na text s příklady

3.2.3.1 Algoritmus procházení modelu v režimu Inheritance show

Při procházení OntoUML modelu v tomto režimu program používá to, že každý vrchol ví o tom, jaké podtřídy má. Jelikož model lze v tomto případě reprezentovat jako orientovaný strom, je možné snadno vypsat celý podstrom vybrané třídy. Přesná pravidla toho, jak se vytváří popis stromu dědičnosti, jsou uvedena v podkapitole Transformace pro režim Inheritance show.

3.2.3.2 Algoritmus procházení modelu v režimu Deep associations

Při procházení OntoUML modelu v tomto režimu je použita modifikovaná verze algoritmu prohledávání do hloubky. Graf pro spouštění DFS je vytvořen z tříd, které jsou reprezentovány vrcholy, a asociativních vazeb, které jsou reprezentovány hranami grafu.

Modifikace základního algoritmu DFS, který je detailně popsán v knize [14], spočívá v tom, že běh končí, až jsou navštíveny všechny dosažitelné hrany. Během hledání cesty v průběhu algoritmu program vytváří výsledný text a řídí se pravidly z podkapitoly 3.2.2.2.

3.2.4 Popis tříd

V dané podkapitole jsou popsané jednotlivé třídy, které jsou použitý v programu. Třídy, které dědí od jiných tříd a v sobě nemají žádnou složitou logiku jsou popsané v ramci popisu svých rodičů. Celý diagram tříd je uveden v příloze A. Dokumentace kódu vytvořená pomocí Doxygen[15] obsahuje také malé diagrámy, které popisují vztahy mezi třídami.

3.2.4.1 Třída MainWindow

Třída MainWindow je hlavní třídou, která propojuje grafické uživatelské rozhraní s celou logikou na straně backendu aplikace. Když uživatel zmáčkne tlačítka v GUI, jsou odesílány signály „clicked“, které jsou propojeny se sloty této třídy. Celý seznam akcí prováděných v GUI a odpovídajících slotů je uveden v tabulce 3.2.

3.2.4.2 Třída DataContainer

Hlavním úkolem třídy DataContainer je uložení dat a manipulace s nimi. Dále tato třída ukládá popis whole-part vztahů, název modelu a definici všech názvů stereotypů OntoUML.

Akce	Slot uvnitř třídy MainWindow
Spuštění volby souboru se vstupními daty	OnBrowseButtonClicked
Spuštění generování textu a ukládání výsledku transformace	OnSaveButtonClicked
Spuštění generování textu a náhledu výsledku transformace	OnPreviewButtonClicked
Spuštění volby adresáře pro soubor se vstupními daty	OnSelectInputDir
Spuštění volby adresáře pro uložení souboru s výsledkem transformace	OnSelectOutputDir

Tabulka 3.2 Seznam slotů, které propojují GUI a backend

Třída DataContainer umožňuje načítání dat ze vstupního souboru, vyhledávání tříd a vztahů a také spravuje stavy tříd pro vyhledávací algoritmy.

3.2.4.3 Třída FileReader

Třída FileReader zajišťuje načtení cesty k souboru a poté načtení souboru do instance třídy QDomDocument, která reprezentuje XMI dokument. Slouží především k ověření správnosti cesty, formátu a dostupnosti souboru pro čtení.

3.2.4.4 Třída GeneralizationSet

Třída GeneralizationSet popisuje množinu nadtypu. V sobě obsahuje vlastnosti, které mohou být definované pro generalization set a identifikační údaje pro třídy, které jsou jeho součástí. Mezi vlastnostmi patří „disjoint“, „complete“, které jsou popsané v podsekci 2.1.2.3.

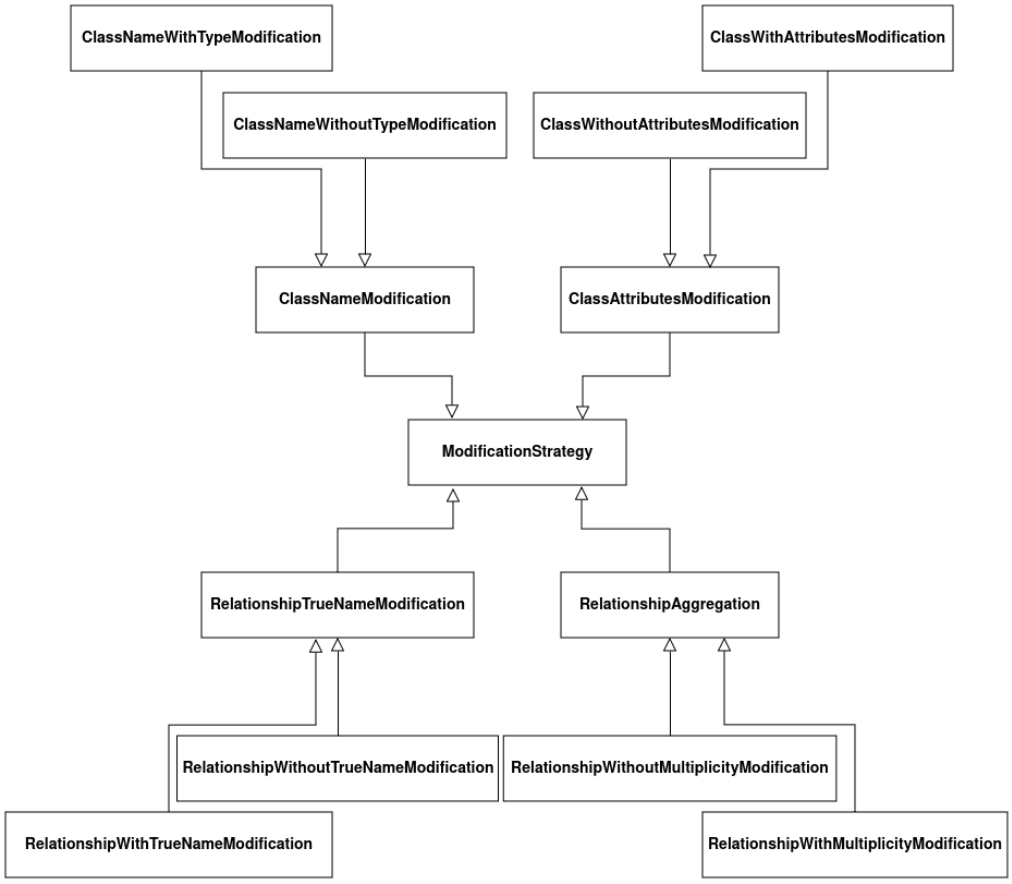
3.2.4.5 Třída ModificationStrategy

Třída ModificationStrategy je abstraktní třída určená k definici dodatečných filtrů používaných při transformaci. Obsahuje abstraktní metodu *apply*, určenou pro aplikaci filtru(přidání informace do výsledného textu v závislosti na nastaveních). Celý strom dědičnosti je zobrazen na obrázku 3.5.

Názvy koncových tříd pro nastavení vždy obsahují slovo „With“/„Without“(příslušná modifikace je vždy použita v závislosti na tom, na jakou hodnotu je nastaven daný filtr uživatelem). V tabulce 3.3 jsou popsany třídy pro filtry a jména nastavení, která jsou uvedena v grafickém uživatelském rozhraní. Hvězdička(*) v názvu třídy slouží jako zástupný symbol pro „With“, „Without“.

Jméno třídy pro filtr	Filtr v uživatelském rozhraní
ClassName*Modification	Stereotypes OntoUML names
Class*AttributesModification	Attributes
Relationship*MultiplicityModification	Multiplicities
Relationship*Aggregation	Whole-part relationships descriptions

Tabulka 3.3 Třídy pro filtry



Obrázek 3.5 Strom dědičnosti třídy ModificationStrategy

3.2.4.6 Třída PdfExporter

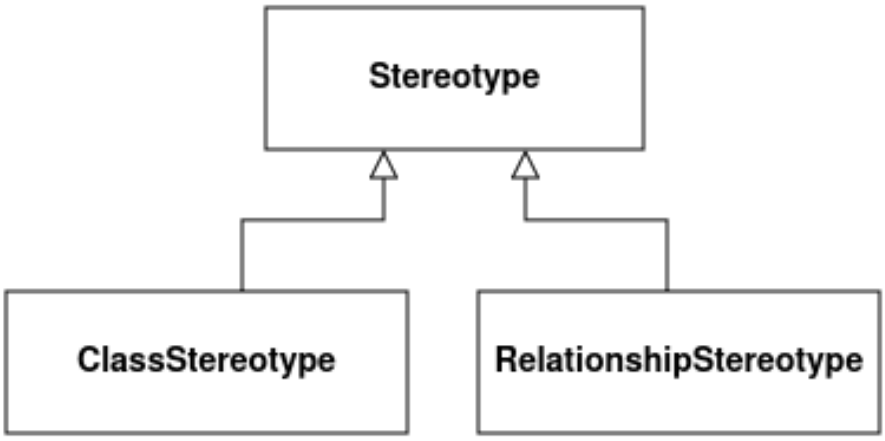
Hlavním účelem třídy PdfExporter je vytváření výsledného PDF souboru s textem popisujícím nahraný OntoUML model. Každá instance této třídy je po vytvoření okamžitě smazána. V konstruktoru je volána metoda „run“, která spouští proces tvorby a uložení PDF souboru.

3.2.4.7 Třída Stereotype

Třída Stereotype je slouží k ukládání informací o třídních a vazebních stereotypech. Obrázek 3.6 ukazuje strom dědičnosti pro danou třídu. Třída ClassStereotype reprezentuje instance třídních stereotypů, třída RelationshipStereotype slouží k ukládání instancí vazebních stereotypů. Tyto třídy se liší vnitřní strukturou, protože mají různé vlastnosti. Například vazby mají vlastnost multiplicity, kterou třídy nemají.

3.2.4.8 Třída SettingsModel

Třída SettingsModel ukládá a spravuje nastavení dodatečných informací z obrazovky Settings.



Obrázek 3.6 Strom dědičnosti třídy Stereotype

3.2.4.9 Třída TextGenerator

Třída TextGenerator obsahuje metody pro transformaci OntoUML XMI modelu na jeho textovou podobu. Ukládá nastavení filtrů, spouští algoritmy pro různé režimy, upravuje stavy tříd po procházení algoritmy a modifikuje text podle nastavení provedeného uživatelem.

3.2.5 Implementace nastavení v kódu

V této kapitole jsou popsány způsoby aplikování nastavení, které jsou provedeny uživatelem, pro vytvoření výsledné transformace. Všechna nastavení jsou rozděleny do 5 skupin:

- **Nastavení sekci**
  Daný typ nastavení je určen pro správu sekci a je v uživatelském rozhraní zobrazen jako pole pro seznam sekci a tlačítko mazání pod tímto polem
- **Nastavení počáteční entity**
  Tetno typ nastavení je určen pouze pro nastavení počáteční entity pro následnou transformaci, nachází se na obrazovce Transformer a má tvar kombinovaného pole v uživatelském rozhraní
- **Nastavení režimu**
  Tetno typ nastavení je určen použe pro nastavení režimu transformace, nachází se na obrazovce Transformer a má tvar kombinovaného pole v uživatelském rozhraní
- **Nastavení dodatečných filtrů**
  Tato nastavení jsou umístěná na obrazovce Transformer, používají zaškrtávací pole v uživatelském rozhraní
- **Nastavení dodatečných informací pro výsledný vygenerovaný PDF soubor**
  Jsou to všechna nastavení na obrazovce Settings

Všechna nastavení jsou v backendové části zpracovaná až po spuštění transformace. Transformace se spustí, když uživatel zmáčkne tlačítko Preview nebo Save file. Nastavení se nezpracovávají okamžitě, když je uživatel změní, protože uživatel může změnit svou volbu vícekrát, a počet těchto změn není žádným zbůsobem omezen.

3.2.5.1 Nastavení sekci

Aplikace dovoluje uživateli zvolit víc počátečních entit a režimy, ve kterých budou tyto entity popsány. V backendu programu jsou tato nastavení prováděná v třídě MainWindow uvnitř metody parseSections.

3.2.5.2 Nastavení počáteční entity

Nastavení počáteční entity v backendové části aplikace je povaděno v metodě parseSections třídy MainWindow. Jelikož každý genrerovany PDF soubor může výsledně mít několik sekcí v závislosti na volbě uživatele, každá sekce je označená v uživatelském rozhraní jako „*Název režimu* for *jméno počátečního stereotypu*“. Díky tomuto tvaru je v metodě parseSections možné definovat název počáteční entity a následně určit i samotný objekt, který za danou entitu odpovídá.

Po určení entity je všechna potřebná informace dál předávána třídě TextGenerator.

3.2.5.3 Nastavení režimu

Nastavení režimu v backendové části aplikace je realizováno třídou TextGenerator v metodě selectModeAndGenerateText. Jak vyplývá z názvu metody, je určená hlavně pro zvolení režimu transformace a tvorbu výsledného textu.

V závislosti na zvoleném režimu pro danou počáteční entitu(která je předána jako argument této metody) je uvnitř této metody zavolána metoda describeRelationshipsForClass pro režim Deep associations nebo describeGeneralizationTreeForClass pro režim Inheritance show.

3.2.5.4 Nastavení dodatečných filtrů

Do dodatečných nastavení nebo filtrů patří nastavení z obrazovky Transformer, které lze měnit pomocí zaškrtávacích poli(nastavení zobrazení atributů, deskripce multiplicit, popis whole-part vztahů, zobrazení jmen třídních stereotypů).

Nastavení dodatečných filtrů je realizováno ve třídě MainWindow v metodě configureApp. V závislosti na tom, na jakou hodnotu je nastaveno zaškrtávací pole, jsou nastaveny příslušné třídy pro filtry. Všechna tato nastavení patří do třídy TextGenerator a jsou implementovaná pomocí abstraktní třídy ModificationStrategy.

3.2.5.5 Nastavení dodatečných informací pro výsledný vygenerovaný PDF soubor

Dodatečná nastavení(jméno autora transformovaného textu, logotyp, volba souborů pro čtení a ukládání dat) se nachází na obrazovce Settings a v backendové části jsou uložená jako instance třídy SettingsModel.

Implementace nastavení dodatečných informací je založena na systému SIGNAL-SLOT Qt frameworku. Ve chvíli, kdy je hodnota nějakého z nastavení z dané skupiny změněna a následně potvrzena(pokud se potvrzení očekává), díky signálu o změně hodnoty se vždy zavola příslušný setter pro aktuální nastavení. Při vytváření výsledného textu, ať už jde o soubor PDF, nebo náhled, se k nastaveným hodnotám přistupuje pomocí getterů.

3.2.6 Implementace HTML šablony pro vytvoření textu

Obsahem dané podkapitoly je popis implementované HTML šablony pro generování náhledu a souboru formátu PDF.

Celý mechanismus vytváření struktury výsledného textu je založen na vlastnosti toho, že pomoci metody replace třídy QString lze s řetězcem provest operaci výměny jedné sekvence symbolů

za jinou. Například řetězec QString(”Hello, my name is Honza”) se po volání QString(”Hello, my name is Honza”).replace(”Honza”, „Jan”) stane QString(”Hello, my name is Jan”).

V případě generování textu pro OntoUML model tato vlastnost je užitečná, protože dovoluje přídat do původní šablony cokoliv, a díky tomu lze výsledný text rozšířit nebo přidat doposud neznámou informaci.

HTML šablona pro generování textu se v projektu nachází v souboru report_template.h a uvnitř něj je nadefinováno několik jmenných prostorů. Celkem jsou v tomto souboru definovány tři jmenné prostory(víz následující seznam):

- **ReportPlaceholder**
- **ReportTemplates**
- **ReportHTMLConstants**

Jmenný prostor **ReportHTMLConstants** je určen pouze k definici řetězců reprezentujících HTML tagy. Tagy použité v této závěrečné práci jsou popsány v podsekci 2.2.4. Tagy jsou vždy obaleny ostrými závorkami.

ReportTemplates je hlavním jmenným prostorem, pomoci kterého je vytvářen výsledný text. Zde jsou nadefinovány konstantní řetězce, které obsahují informaci o zástupných symbolech pro budoucí text, tagy pro formátování textu a také samotné textové šablony, které budou zobrazeny ve výsledném textu beze změn.

Poslední nepopsaný jmenný prostor je **ReportPlaceholder**. V dané části jsou nadefinovány konstantní řetězce, které slouží jako zástupné symboly a jsou nahrazovány během generování výsledku transformace. Všechny zástupné symboly začínají znakem „#“, aby bylo možné rozlišit obyčejný text a zástupný symbol a omylem nenahradit již správný text. Každý zástupný symbol navíc nesmí být prefixem jiného. Toto pravidlo musí platit, protože jinak z pohledu třídy QString bude možné nahradit text špatně. Pro příklad se lze podívat na kód 3.1.

```
QString text("Hello, #name, my name is #nameMain");
text.replace("#name", "Katerina");
```

Výpis kódu 3.1 Příklad řetezce a operace replace

Po provedení poslední operace je v proměnné *text* uložená hodnota „Hello, Katerina, my name is KaterinaMain“, tzn. i když programátor chtěl nahradit pouze tag *#name* bude nahrazen i tag *#nameMain*, protože *#name* je prefixem tagu *#nameMain*.

Kapitola 4
Testování

V dané kapitole jsou popsány postupy, použité pro otestování funkčnosti výsledné aplikace. Podprojekt s testy je umístěn ve vlastní složce a je oddělen od hlavní části projektu se samotnou aplikací, proto je spustitelný samostatně bez nutnosti spouštění celého programu.

4.1 Jednotkové testy

Daná podkapitola poskytuje přehled a stručný popis způsobu implementace jednotkových testu pro celý projekt.

4.1.1 Stručný popis struktury jednotkových testů

Jednotkové testy nebo unit testy, které jsou nezbytné pro kvalitní software k omezení počtu chyb během vývoje nových funkcí nebo refaktoringu kódu. Jsou implementovány pomocí frameworku Qt Test[16][17], který poskytuje řadu maker(anglicky *macros*) pro zjednodušení psaní jednotkových testů. Pro testovaní aplikace pro transformace OntoUML modelů do přirozeného textu byla použitá makra, která jsou popsána v tabulce 4.1.

Název makra	Popis
QCOMPARE*(aktuální, očekávaná)*	Makro, které porovnává *aktuální* a *očekávanou* hodnotu. Pokud se shodují, pak běh testu pokračuje. Jinak se v logu testování objeví chyba, a běh aktuálního testu bude přerušen.
QVERIFY*(podmínka)*	Makro, které ověřuje, zda je *podmínka* splněná(je nastavená na *true*). Pokud ano, pak běh testu pokračuje. Jinak se v logu testování objeví chyba, a běh aktuálního testu bude přerušen.
QVERIFY2*(podmínka, zpráva)*	Makro, které se chová stejně jako **QVERIFY**. Jediný rozdíl je v tom, že v případě chyby do logu vypíše nejen hlásku o chybě, ale i *zprávu*, která je předaná jako argument.

Tabulka 4.1 Použitá v jednotkových testech makra

Každý jednotkový test začíná vytvořením instance testované třídy, pokračuje voláním testované metody a používá makra pro ověření stavu instance před a po jejím spuštění. Každá testovaná projektová třída má vlastní testovací třídu, každá metoda je testovaná ve své vlastní metodě testovací třídy.

4.1.2 Přehled jednotkových testu

Bylo vytvořeno celkem 65 jednotkových testů pro ověření funkčnosti programu. Informace o jednotlivých testovacích třídách jsou uvedeny v tabulce 4.2. Název testovací třídy vždy začíná slovem „Test“, pokračuje jménem třídy, kterou testuje. Například testovací třída TestClassStereotypes je určena pro testování třídy ClassStereotypes.

Testovací třída	Testů	Chybných	Ignorováno
TestFileReader	4	0	0
TestDataContainer	3	0	0
TestClassStereotype	6	0	0
TestRelationshipAggregation	5	0	0
TestRelationshipMultiplicityModification	4	0	0
TestRelationshipTrueNameModification	4	0	0
TestRelationshipTypeModification	9	0	0
TestClassAttributesModification	5	0	0
TestPdfExporter	3	0	0
TestRelationshipStereotype	13	0	0
TestSettingsModel	6	0	0
TestTextGenerator	3	0	0

Tabulka 4.2 Třídy s jednotovými testy ve výsledne aplikaci

Díky tomu, že testovací podprojekt není částí aplikace samotné, lze všechny testy spustit bez nutnosti spouštění programu pro transformaci. Ověření funkčnosti po spuštění je popsáno v další podkapitole 4.2.

4.2 Manuální testování

V dané podkapitole jsou popsány testovací scénáře pro manuální testování funkčnosti aplikace. Celkem je uvedeno devět základních scénářů, které v budoucnu lze rozšířit nebo změnit podle přidaných nebo změněných funkcí.

- **TC1 - Ověření funkčnosti módu „Deep associations“**
  - **Počáteční stav**: je zvolen soubor XMI s modelem OntoUML, který obsahuje alespoň dvě třídy, a alespoň dvě třídy jsou mezi sebou propojené nějakou vazbou, která není dědičností
  - **Kroky**
    1. Zvolit pro popis mód „Deep associations“
       - **Akceptační kritéria**: název zvoleného režimu v okně odpovídá volbě „Deep associations“
    2. Vybrat entitu, která je propojená s jinou entitou pomocí vazby
       - **Akceptační kritéria**: název zvolené entity v okně odpovídá skutečné volbě
    3. Zmačknout tlačítko „Preview“

    - **Akceptační kritéria**: vygenerovaný náhled obsahuje popis existujících vazeb nebo jedné vazby. Popsany jsou všechny vazby, které souvisí se zvolenou počáteční entitou(včetně nepřímých vazeb)

- **TC2 - Ověření funkčnosti módu „Inheritance show“**
  - **Počáteční stav:** je zvolen soubor XMI s modelem OntoUML, ve který obsahuje alespoň dvě třídy, a alespoň dvě třídy jsou propojeny vazbou, popisující dědičnost, což znamená, že jedna třída je podtřídou druhé
  - **Kroky**
    1. Zvolit pro popis mód „Inheritance show“
       - **Akceptační kritéria**: název zvoleného režimu v okně odpovídá volbě „Inheritance show“
    2. Vybrat entitu, která je propojená s jinou entitou pomocí vazby dědičnosti
       - **Akceptační kritéria**: název zvolené entity v okně odpovídá skutečné volbě
    3. Zmačknout tlačítko „Preview“
       - **Akceptační kritéria**: vygenerovaný náhled obsahuje popis existujících dědičností tak, že začíná entitou, zvolenou jako počáteční a pokračuje „směrem dolu“, tzn. rekurzivně popisuje strom podtříd. Popsany jsou všechny podtřídy počáteční třídy

- **TC3 - vypsání atributů při popisu vazeb**
  - **Počáteční stav**: je zvolen soubor XMI s modelem OntoUML, ve kterém alespoň jedna ze tříd obsahuje v těle minimálně jeden atribut
  - **Kroky**
    1. Zvolit pro popis mód „Deep associations“
       - **Akceptační kritéria**: v okně je zobrazen název zvoleného režimu „Deep associations“
    2. Zvolit entitu, která ve svém těle má alespoň jeden atribut
       - **Akceptační kritéria**: název zvolené entity v okně odpovídá skutečné volbě
    3. Zaškrtnout čtverec „Attributes“, který se nachází v záložce pro nastavení transformace
       - Je zaškrtnut čtverec „Attributes“
    4. Zmačknout tlačítko „Preview“
       - **Akceptační kritéria**: vygenerovaný náhled dokumentu obsahuje větu ve formátu: „List of the attributes for the *Název entity* class: *názvy atributů*“. *Názvy atributů* musí zahrnovat všechny atributy entity

- **TC4 - přidání názvů OntoUML třídních stereotypů do výsledku transformace**
  - **Počáteční stav**: je zvolen soubor XMI s modelem OntoUML, který obsahuje alespoň jednu entitu, používající jeden z definovaných OntoUML třídních stereotypů (Kind, SubKind, Role, Phase, Collective, Quantity, Relator, Category, PhaseMixin, RoleMixin, Mixin, Mode, Quality)
  - **Kroky**
    1. Zvolit pro popis jeden z módů, nabízených aplikaci
       - **Akceptační kritéria**: název zvoleného režimu v okně odpovídá skutečné volbě
    2. Vybrat entitu, která používá jeden z třídních stereotypů OntoUML
       - **Akceptační kritéria**: název zvolené entity v okně odpovídá skutečné volbě
    3. Zaškrtnout čtverec „Stereotypes OntoUML names“, který se nachází v záložce pro nastavení transformace

4. Zmačknout tlačítko „Preview“
   * **Akceptační kritéria**: V popisu každé asociace nebo dědičnosti se vedle jména popisované entity nachází název její OntoUML stereotypu v kulatých závorkách

- **TC5 - přidání popisu multiplicit při popisu vztahů**
  - **Počáteční stav:** zvolen soubor XMI s modelem OntoUML, ve kterém jsou popsány alespoň 2 třídy a alespoň 2 třídy jsou mezi sebou propojené asociativní vazbou (Mediation, ComponentOf atd.) a na koncích jsou popsané multiplicity
  - **Kroky**
    1. Zvolit pro popis mód „Inheritance show“
       * **Akceptační kritéria**: název zvoleného režimu v okně odpovídá volbě „Inheritance show“
    2. Vybrat entitu, která je propojená s jinou entitou pomocí asociativní vazby
       * **Akceptační kritéria**: název zvolené entity v okně odpovídá skutečné volbě
    3. Zmačknout tlačítko „Preview“
       * **Akceptační kritéria**: vygenerovaný náhled dokumentu obsahuje ve svém textu popis vazeb a jejich multiplicit. Popis se bude lišit v závislosti na multiplicitách vazeb na jejich koncích, takže musí odpovídat modelu

- **TC6 - přidání většího počtu entit pro popis vztahů a/nebo dědičnosti**
  - **Počáteční stav:** zvolen soubor XMI s modelem OntoUML, ve kterém jsou popsány entity X a Y, které nejsou na sobě závisle (entita X nemá žádný(ani nepřímý) vztah (dědičnost ani asociaci) s entitou Y)
  - **Kroky**
    1. Zvolit pro popis mód „Deep associations“
       * **Akceptační kritéria**: název zvoleného režimu v okně odpovídá volbě „Deep associations“
    2. Vybrat entitu X
       * **Akceptační kritéria**: název zvolené entity v okně odpovídá volbě X
    3. Zmačknout tlačítko „+“
       * **Akceptační kritéria**: pravý seznam pro sekce obsahuje na konci řadek „*Název režimu* for *název entity*“
    4. Zvolit pro popis mód „Deep associations“
       * **Akceptační kritéria**: název zvoleného režimu v okně odpovídá volbě „Deep associations“
    5. Vybrat entitu Y
       * **Akceptační kritéria**: název zvolené entity v okně odpovídá volbě Y
    6. Zmačknout tlačítko „+“
       * **Akceptační kritéria**: pravý seznam pro sekce obsahuje na konci řadek „*Název režimu* for *název entity*“
    7. Zmačknout tlačítko „Preview“
       * **Akceptační kritéria**: vygenerovaný náhled dokumentu obsahuje ve svém textu dvě podsekce. První popisuje všechny asociativní vazby pro entitu X. Druha popisuje všechny asociativní vazby pro entitu Y. Texty v podsekcích neobsahují stejné věty(jelikož X a Y *jsou na sebe nezávislé*)

- **TC7 - Přidání vlastního logotypu do levého horního rohu na začátek vygenerovaného textu**

- **Počáteční stav: v aplikaci je zvolen soubor XMI s validním popisem OntoUML modelu**
- **Kroky**
  1. Klikem zvolit v horním menu aplikace záložku „Settings“
     * **Akceptační kritéria**: Zobrazené okno odpovídá vzhledu záložce „Settings“
  2. Zaškrtnout pole „Add custom logo to PDF file“
     * **Akceptační kritéria**: Tlačitko pro zvolení souboru loga je aktivní
  3. Po zmačknutí tlačítka „Browse“(na stejném řádku s „Add custom logo to PDF file“) v okně zvolit obrázek ve formátu JPEG nebo PNG a potvrdit volbu zmačknutím klavesy Enter
  4. Klikem v horním menu zvolit záložku „Transformer“
     * Zobrazené okno odpovídá vzhledu záložky „Transformer“
  5. Vybrat jeden z módu nabizených aplikaci
     * **Akceptační kritéria**: název zvoleného režimu v okně odpovídá skutečné volbě
  6. Zvolit počáteční entity pro transformaci modelu
     * **Akceptační kritéria**: název zvolené entity v okně odpovídá skutečné volbě
  7. Zmačknout tlačítko „Preview“
     * **Akceptační kritéria**: V levém horním rohu se nachází logo (obrazek), které bylo zvoleno uživatelém při nastavení v záložce „Settings“

- **TC8 - Přidání vlastního jména autoru do střední části pod záhlavím na začátku vygenerovaného textu**
  - **Počáteční stav: v aplikaci je zvolen soubor XMI s validním popisem OntoUML modelu, který má alespoň jednu entitu**
  - **Kroky**
    1. Klikem zvolit v horním menu aplikace záložku „Settings“
       * **Akceptační kritéria**: Zobrazené okno odpovídá vzhledu záložky „Settings“
    2. Zaškrtnout pole „Add author’s name to PDF file“
       * **Akceptační kritéria**: Pole pro zvolení jména a tlačítko pro potvrzení jsou aktivní
    3. S použitím klavesnice napsat preferováné jméno do pole na stejném řádku s „Add author’s name to PDF file“
    4. Potvrdit volby jména zmačknutím tlačítka „Confirm“
    5. Klikem v horním menu zvolit záložku „Transformer“
       * **Akceptační kritéria**: Zobrazené okno odpovídá vzhledu záložce „Transformer“
    6. Vybrat jeden z módu nabizených aplikaci
       * **Akceptační kritéria**: název zvoleného režimu v okně odpovídá skutečné volbě
    7. Zvolit počáteční entity pro transformaci modelu
       * **Akceptační kritéria**: název zvolené entity v okně odpovídá skutečné volbě
    8. Zmačknout tlačítko „Preview“
       * **Akceptační kritéria**: Uprostřed dokumentu na začátku pod záhlavím reportu a datem vytvoření se nachází jméno autora ve formátu „Author: *jméno*“. Zobrazené jméno je shodné s vybraným a potvrzeným jménem v záložce „Settings“

- **TC9 - Zvolení adresáře, ze kterého bude začínat volba souboru při zvolení modelu OntoUML pro trnsformaci**
  - **Počáteční stav: aplikace je otevřená**

- **Kroky**
  1. Klikem zvolit v horním menu aplikace záložku „Settings“
     * **Akceptační kritéria**: Zobrazené okno odpovídá vzhledu záložky „Settings“
  2. Na prvním řádku kliknout na pole pro vyběr módu čtení vstupního souboru
     * **Akceptační kritéria**: jsou zobrazeny dvě možnosti
  3. Zvolit možnost „Select folder to read file from“
     * **Akceptační kritéria**: název zvoleného moódu v okně odpovídá volbě „Select folder to read file from“
  4. Vedle této volby zmačknout tlačítko „Browse“
     * **Akceptační kritéria**: je otevřeno okno s přehledem souborobého systému
  5. V okně najít potřebný adresář a potvrdit volbu zmačknutím tlačítka „Open“(nebo dvakrát zmačknout tlačítko Enter na klavesnici)
     * **Akceptační kritéria**: cesta odpovídá cestě do zvoleného souboru
  6. Klikem v horním menu zvolit záložku „Transformer“
     * **Akceptační kritéria**: Zobrazené okno odpovídá vzhledu záložky „Transformer“
  7. Zmačknout tlačítko „Choose file“
     * **Akceptační kritéria**: V okně pro výběr XMI OntoUML modelu je zobrazen souborový systém a toto hledání začíná ve zvoleném souboru(v okně je zobrazen jeho obsah). V případě neplatného vstupu je zobrazen souborový systém, který začíná ve složce Home(výchozí chování)

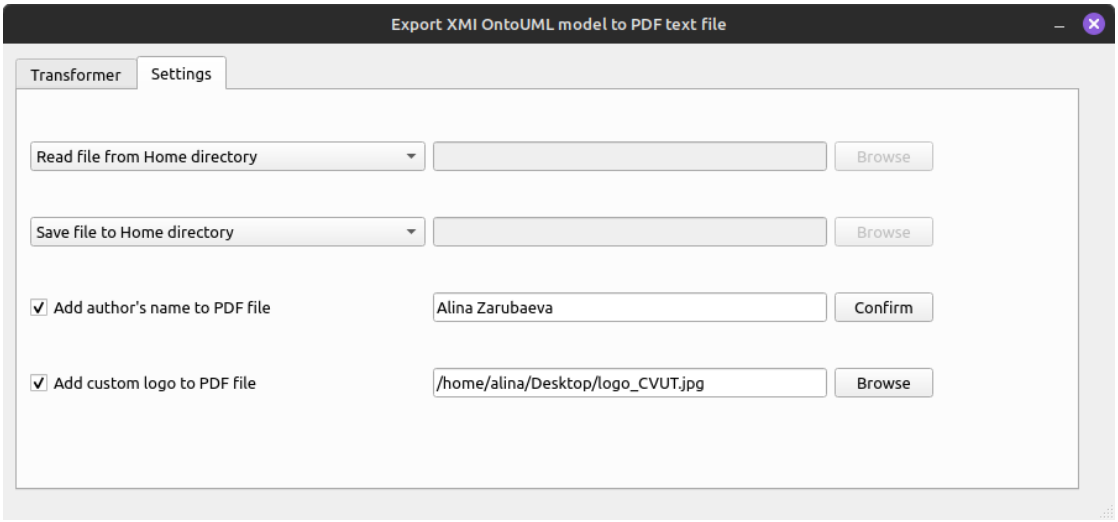
4.3 Výsledky testování

Jednotkové testy byly vytvářeny při přidávání nových metod do programu a pravidelně spouštěny během celého vývoje. Občas se vyskytly drobné chyby jako například chybějící odsazení nebo překlepy, které byly ihned po zjištění opraveny. Po vytvoření šablony s větším počtem sekcí se ukázalo, že se text pro každou novou sekci posunuje doprava. Tento nedostatek byl včas zaznamenán a opraven.

Kapitola 5
Případová studie

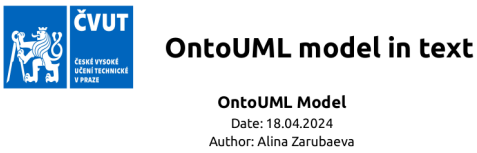
V dané podkapitole jsou uvedeny příklady zpracování modelu OntoUML programem pro export takového typu modelu do přirozeného jazyka. Pro demonstraci je použit model uvedený v příloze B. Tento model byl vytvořen v rámci projektu OpeNest[18] a jeho datového inkubátoru společnosti REMMARK, a.s.[19] Podrobné informace o modelech vytvořených v rámci projektu lze nalézt v závěrečných pracích [20], [21], [22], [23] a [24]. Kompletní výsledek transformace je uveden v příloze C. Více informací o účelech případové studie naleznete článku [25].

Nastavení na obrazovce Settings jsou zobrazena na obrázku 5.1.



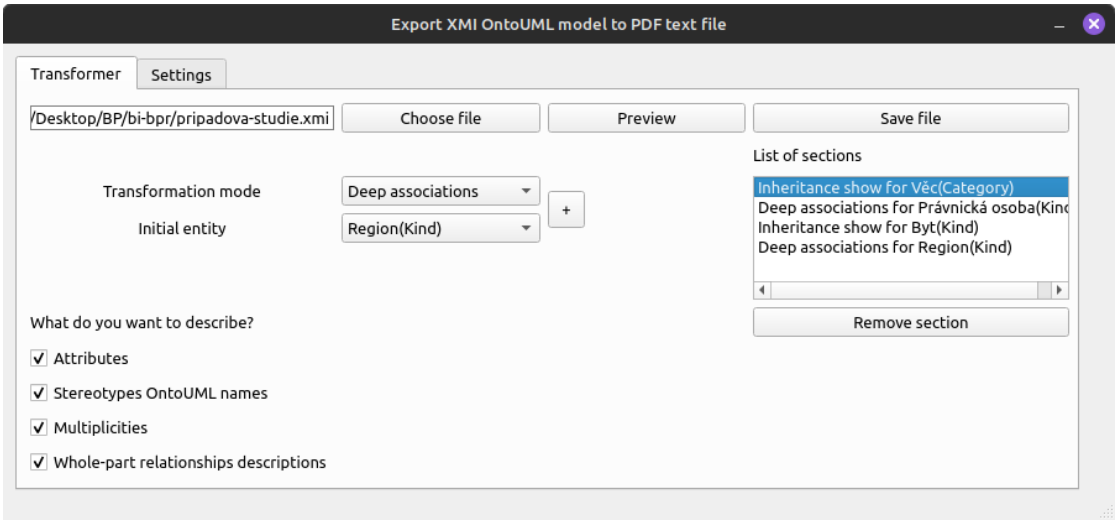
Obrázek 5.1 Nastavení pro obrázovku Settings

Hlavička výsledného souboru při těchto nastaveních vypadá jako na obrázku 5.2.



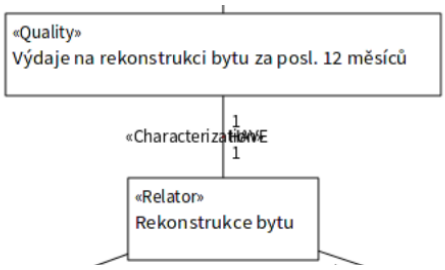
Obrázek 5.2 Hlavička souboru s logotypem a jménem autora

Nastavení na obrazovce Transformer jsou zobrazená na obrázku 5.3.



Obrázek 5.3 Nastavení pro obrázovku Transformer

Na modelu B lze ukázat, jak funguje popis vazeb s určeným stereotypem. Příkladem je část celého modelu zobrazená na obrázku 5.4. Zde je použit stereotyp *Characterization*. Výsledek exportu této části je uveden na obrázku 5.5.



Obrázek 5.4 Část modelu se stereotypem Characterization

Výdaje na rekonstrukci bytu za posl. 12 měsíců(Quality) characterizes *Rekonstrukce bytu*(Relator). Relationship has name 'HAVE'.

- Type *Rekonstrukce bytu* has exactly 1 type *Výdaje na rekonstrukci bytu za posl. 12 měsíců*.
- Type *Výdaje na rekonstrukci bytu za posl. 12 měsíců* has exactly 1 type *Rekonstrukce bytu*.

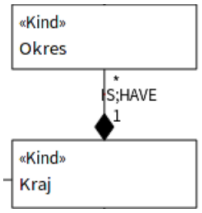
Obrázek 5.5 Příklad výsledku exportu pro stereotyp Characterization

Příkladem, kdy vazba postrádá stereotyp a není vztahem typu „celek-část“, je screenshot části modelu na obrázku 5.6, a výsledek exportu je zobrazen na obrázku 5.7.

Vstupní a výstupní data zobrazená na obrázcích 5.6 a 5.7, popisují export pro vztah typu „celek-část“. Dva poslední řádky(s tečkami na začátku) ilustrují popis multiplicity vztahu.

Popis atributu pro entitu z obrázku 5.8 je zobrazen formou screenshotu na obrázku 5.9.

Dalším příkladem je popis v režimu „Inheritance show“. Je spuštěn s počáteční entitou *Věc*, a strom dědičnosti znázorněný na obrázku 5.10 v textu vypadá jako na obrázku 5.11.



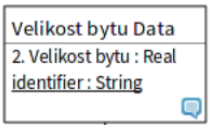
Obrázek 5.6 Část modelu pro ukázku exportu pro vztah typu „celek-část“

Okres(Kind) is in part-whole relation with *Kraj*(Kind). Relationship has name 'IS;HAVE'.

- Type *Kraj* has 1 or more types *Okres*.
- Type *Okres* has exactly 1 type *Kraj*.

Okres is integral part of *Kraj*.

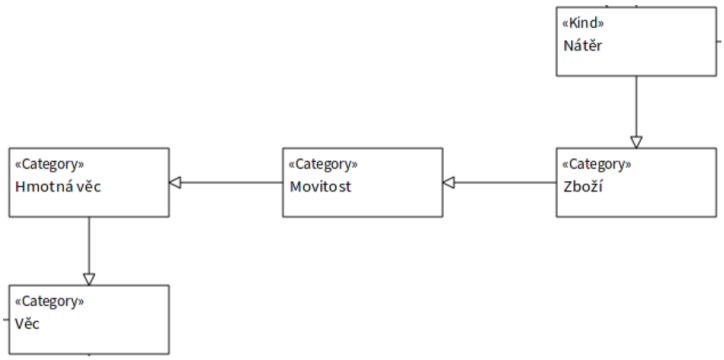
Obrázek 5.7 Příklad výsledku exportu pro vztah typu „celek-část“



Obrázek 5.8 Část modelu pro ukázku exportu atributů

Velikost bytu(Quality) is in relation with *Velikost bytu Data*. List of the attributes for the *Velikost bytu Data* class: identifier, 2. Velikost bytu.

Obrázek 5.9 Příklad výsledku exportu pro atributy



Obrázek 5.10 Část modelu pro ukázku exportu v režimu „Deep associations“

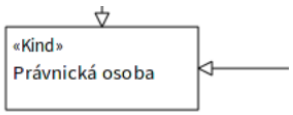
Při spuštění transformace na entitě *Právnická osoba* 5.12 v režimu „Deep associations“ program detekuje absenci asociací pro danou entitu a uživateli to sdělí v textové zprávě, jak je to znázorněno na obrázku 5.13.

Opačným příkladem je entita *Byt* 5.14, která není se žádnou entitou ve vztahu dědičností, proto výsledný text bude vypadat jako text z obrázku 5.15.

Inheritance show for Věc(Category)

Věc is a supertype for *Hmotná věc*.
Hmotná věc is a supertype for *Movitost*.
Movitost is a supertype for *Zboží*.
Zboží is a supertype for *Nátěr*.
Nátěr has no related subtypes.

Obrázek 5.11 Příklad výsledku exportu pro režim „Deep associations“

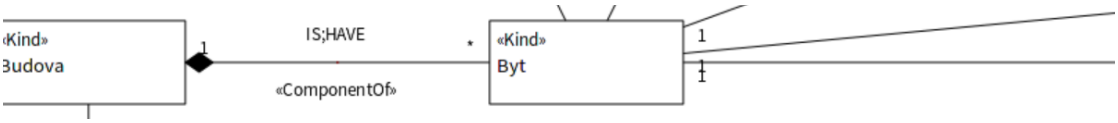


Obrázek 5.12 Část modelu pro ukázku exportu s entitou bez vazeb v režimu „Deep associations“

Deep associations for Právnická osoba(Kind)

No direct associations found for this case.

Obrázek 5.13 Příklad výsledku exportu pro entitu bez vazeb a režim „Deep associations“



Obrázek 5.14 Část modelu pro ukázku exportu s entitou bez dědičností v režimu „Inheritance show“

Inheritance show for Byt(Kind)

Byt has no related subtypes.

Obrázek 5.15 Příklad výsledku exportu pro entitu bez dědičností a režim „Inheritance show“

Kapitola 6
Závěr

Výsledkem této bakalářské práce je aplikace pro export OntoUML modelů do přirozeného jazyka.

Na začátku bylo rozhodnuto nadefinovat teoretický základ, na kterém bude vybudovaná celá budoucí aplikace. Kvůli tomu, že OntoUML není tak populární a používaný jako UML, v současné době neexistují nástroje pro export modelů OntoUML do přirozeného textu. Však lze říct, že existující teoretické základy, které jsou dostupné na internetu a popsané v knihách, jsou vhodné pro tvorbu jakýchkoliv aplikací, které na vstupu dostávají jakýkoliv model a na výstupu mají uživateli předat již jeho textovou podobu.

Po rešerši existujících návrhů pro tvorbu požadované aplikace následovala fáze analýzy požadavků na software a definice formátu, obsahu a vlastnosti vstupních dat. V daném kroku bylo rozhodnuto, jaké funkcionality musí mít výsledná aplikace a co musí splňovat, aby odpovídala očekáváním zadavatele práce. V průběhu této části práce na závěrečné práci bylo rozhodnuto, jaké nástroje(jazyk, framework) budou pro implementaci použity.

Následujícím po analýze krokem byla samotná implementace aplikace, během které bylo rozhodnuto, jak přesně bude aplikace vypadat, pomocí čeho budou požadované funkcionality implementované, jaké algoritmy mohou být použity pro prohledávání modelů. V této část již byly známé všechny nezbytné pro implementace třídy a také bylo určeno, jakou roli bude hrát každá z nich. Tento krok obsahoval také popis vnitřní struktury aplikace(stručná deskripce implementace backendu).

Po dokončení předchozích kroků vývojový proces pokračoval fází testování. Jednotkové testy byly implementovány již ve fázi implementace, proto hlavním cílem bylo ověřit správnost programu v závislosti na výsledcích unit testů. Během procesu ověření se vyskytlo několik drobných chyb, které byly opraveny hned po nalezení. Manuální testovací scénáře byly vytvořeny pro ověření funkčnosti backendu programu ve spolupráci s grafickým uživatelským rozhraním. V tomto kroku nebyly nalezeny žádné chyby a interface se ukázal jako uživatelsky přívětivý a snadný pro použití.

Nakonec byly navrženy možnosti pro další vývoj programu v budoucnu. I když v dané fázi aplikaci lze prohlásit za funkční a odpovídající požadavkům zadavatele, do budoucna lze uvážit několik způsobů její vylepšení pro to, aby měla větší počet funkcí a byla pro uživatele snadnější na ovládání.

Největším přínosem pro mě, jako autora dané závěrečné práce a aplikace, byla možnost si vyzkoušet projít procesem tvorby softwaru od začátku, kdy jsou známé jen nějaké požadavky zadavatele bez formálního popisu, až na po prohlášení aplikace za funkční a splňující očekávaní.

6.1 Budoucnost aplikace a další vývoj

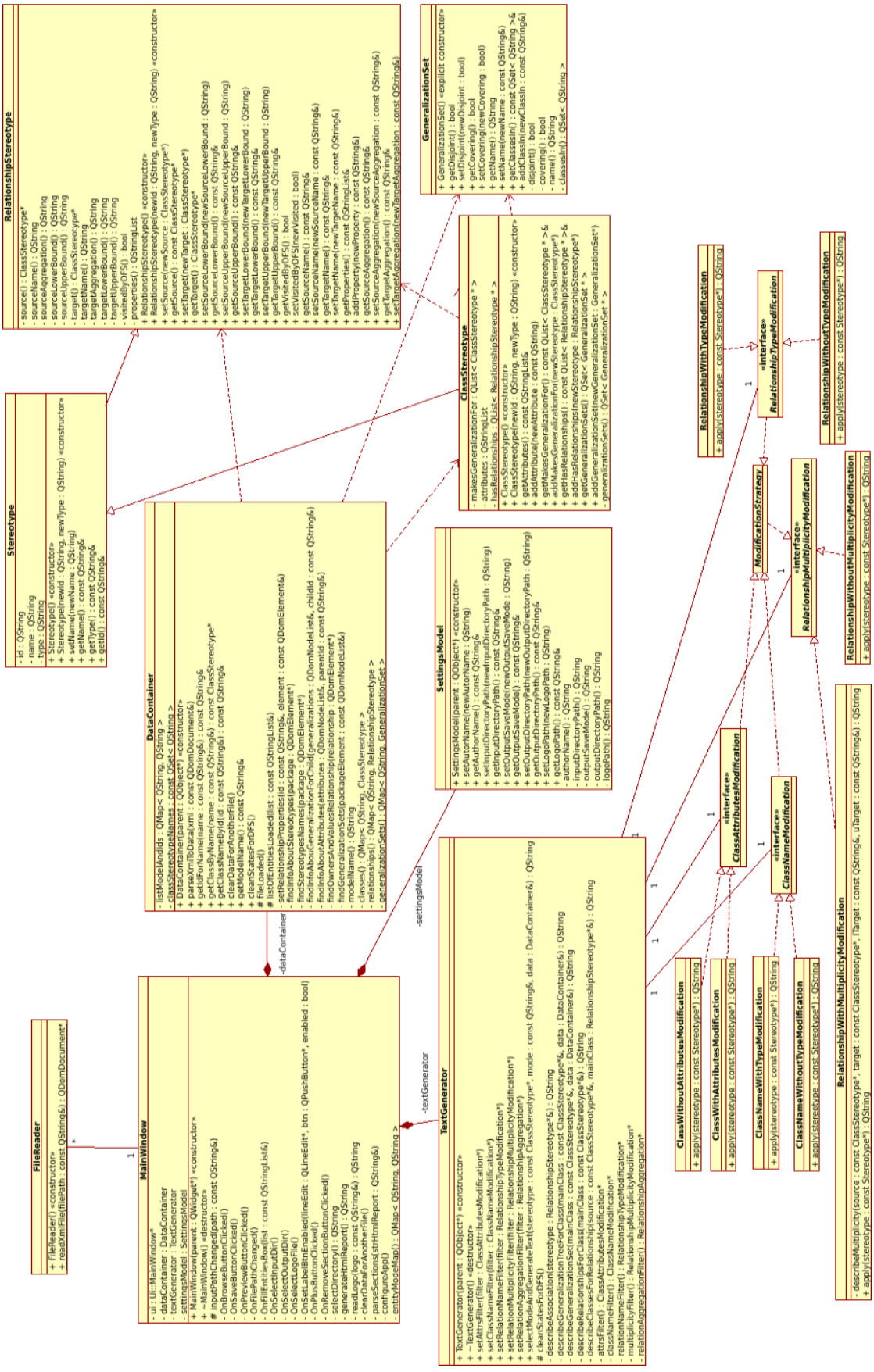
Naimplementovaná aplikace splňuje všechny požadavky a odpovídá očekáváním. Avšak v budoucnu lze již existující program rozšířit pro to, aby byla snadněji ovladatelná a dovolovala uživateli vytvářet více přizpůsobenou textovou podobu OntoUML modelu. Seznam možných rozšíření:

- Přidání obsahu s odkazy na text
- Přidání možnosti vytváření a ukládání šablon s parametry(režim s předem vyplněnými parametry, který eliminuje nutnost ručního výběru)
- Přidání grafického rozhraní pro volbu počátečních entit přímo v nakresleném modelu
- Přidání možnosti volby entit, které musí být popisované(nový režim, který nebude procházet model jako graf, ale bude obsahovat jen stručný popis zvolených entit)
- Přidání dalších jazyků do uživatelského rozhraní a výsledného textu(lze jednoduše udělat pomocí nástroje Qt Linguist[26])

Tyto navrhované funkcionality jsou pouze malou části toho, jak lze v budoucnu existující program rozvíjet.

Příloha A

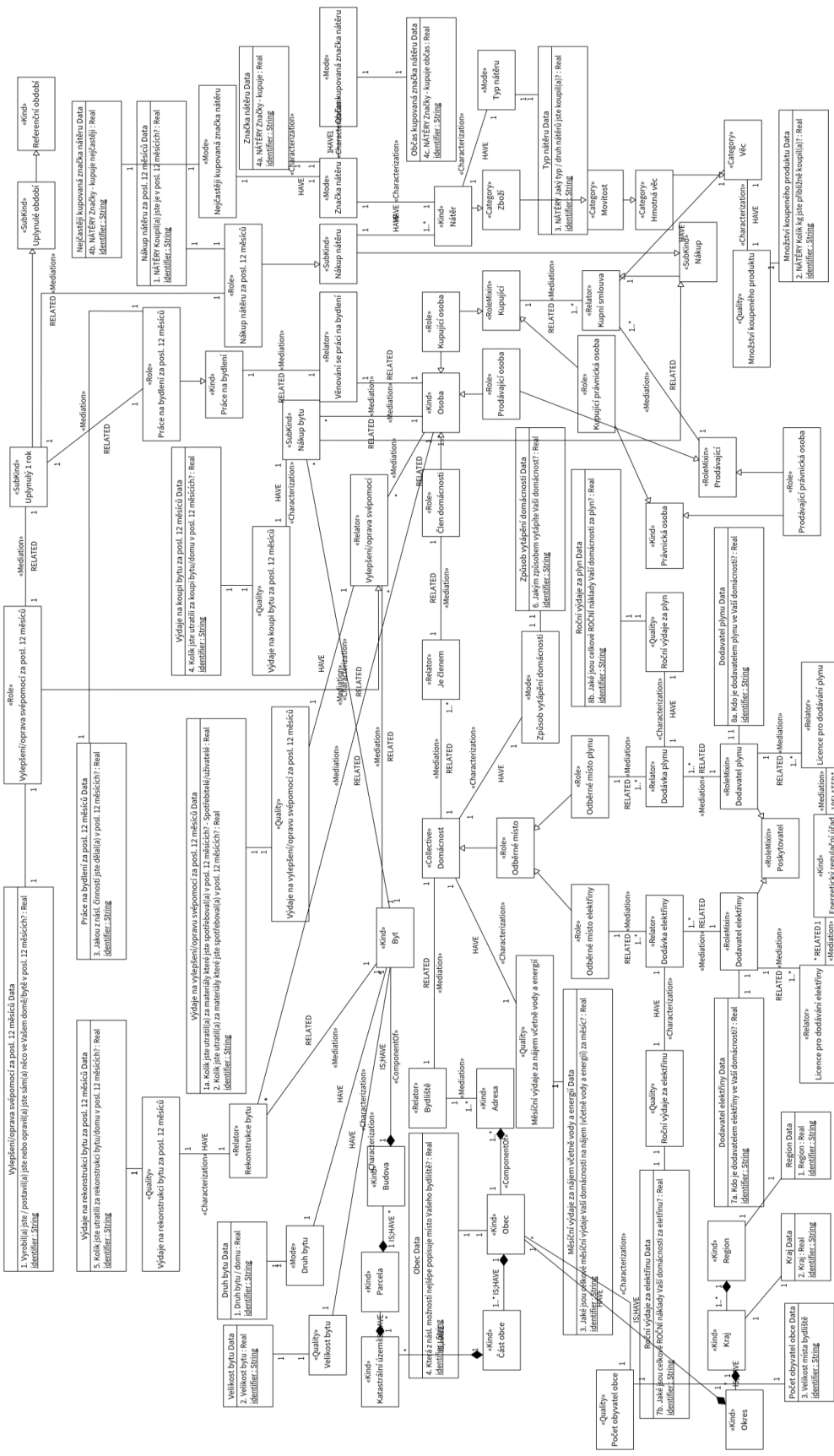
Diagram tříd pro aplikaci pro export modelů OntoUML do textu



Obrázek A.1 Diagram tříd pro aplikaci pro export modelů OntoUML do textu

Příloha B

Model pro případovou studií



Obrázek B.1 Model „Bydlení“ pro případovou studii

Příloha C

Výsledek exportu modelu pro případovou studií

OntoUML model in text

OntoUML Model
Date: 18.04.2024
Author: Alina Zarubaeva

Inheritance show for Věc(Category)

Věc is a supertype for *Hmotná věc*.
Hmotná věc is a supertype for *Movitost*.
Movitost is a supertype for *Zboží*.
Zboží is a supertype for *Nátěr*.
Nátěr has no related subtypes.

Deep associations for Právnická osoba(Kind)

No direct associations found for this case.

Inheritance show for Byt(Kind)

Byt has no related subtypes.

Deep associations for Region(Kind)

Kraj(Kind) is in part-whole relation with *Region*(Kind).

- Type *Region* has 1 or more types *Kraj*.
- Type *Kraj* has exactly 1 type *Region*.

Kraj is integral part of *Region*.
Okres(Kind) is in part-whole relation with *Kraj*(Kind). Relationship has name 'IS;HAVE'.

- Type *Kraj* has 1 or more types *Okres*.
- Type *Okres* has exactly 1 type *Kraj*.

Okres is integral part of *Kraj*.
Obec(Kind) is in part-whole relation with *Okres*(Kind). Relationship has name 'IS;HAVE'.

- Type *Okres* has 1 or more types *Obec*.
- Type *Obec* has exactly 1 type *Okres*.

Obec is integral part of *Okres*.

1

Obrázek C.1 Model „Bydlení“ pro případovou studií(stránka 1)

Část obce(Kind) is in part-whole relation with *Obec*(Kind). Relationship has name 'IS;HAVE'.

- Type *Obec* has 1 or more types *Část obce*.
- Type *Část obce* has exactly 1 type *Obec*.

Část obce is integral part of *Obec*.
Katastrální území(Kind) is in part-whole relation with *Část obce*(Kind). Relationship has name 'IS;HAVE'.

- Type *Část obce* has 1 or more types *Katastrální území*.
- Type *Katastrální území* has exactly 1 type *Část obce*.

Katastrální území is integral part of *Část obce*.
Parcela(Kind) is in part-whole relation with *Katastrální území*(Kind). Relationship has name 'IS;HAVE'.

- Type *Katastrální území* has 1 or more types *Parcela*.
- Type *Parcela* has exactly 1 type *Katastrální území*.

Parcela is integral part of *Katastrální území*.
Budova(Kind) is in part-whole relation with *Parcela*(Kind). Relationship has name 'IS;HAVE'.

- Type *Parcela* has 1 or more types *Budova*.
- Type *Budova* has exactly 1 type *Parcela*.

Budova is integral part of *Parcela*.
Budova(Kind) is in part-whole relation with *Byt*(Kind). Relationship has name 'IS;HAVE'.

- Type *Byt* has exactly 1 type *Budova*.
- Type *Budova* has 1 or more types *Byt*.

Byt is integral part of *Budova*.
Nákup bytu(SubKind) mediates *Byt*(Kind). Relationship has name 'RELATED'.

- Type *Byt* has 1 or more types *Nákup bytu*.
- Type *Nákup bytu* has exactly 1 type *Byt*.

Osoba(Kind) mediates *Nákup bytu*(SubKind). Relationship has name 'RELATED'.

- Type *Nákup bytu* has exactly 1 type *Osoba*.
- Type *Osoba* has 1 or more types *Nákup bytu*.

Osoba(Kind) mediates *Rekonstrukce bytu*(Relator). Relationship has name 'RELATED'.

- Type *Rekonstrukce bytu* has 1 or more types *Osoba*.

2

Obrázek C.2 Model „Bydlení“ pro případovou studií(stránka 2)

- Type *Osoba* has 1 or more types *Rekonstrukce bytu*.

Rekonstrukce bytu(Relator) mediates *Byt*(Kind). Relationship has name 'RELATED'.

- Type *Byt* has 1 or more types *Rekonstrukce bytu*.
- Type *Rekonstrukce bytu* has exactly 1 type *Byt*.

Vylepšení/oprava svépomocí(Relator) mediates *Byt*(Kind). Relationship has name 'RELATED'.

- Type *Byt* has 1 or more types *Vylepšení/oprava svépomocí*.
- Type *Vylepšení/oprava svépomocí* has exactly 1 type *Byt*.

Osoba(Kind) mediates *Vylepšení/oprava svépomocí*(Relator). Relationship has name 'RELATED'.

- Type *Vylepšení/oprava svépomocí* has exactly 1 type *Osoba*.
- Type *Osoba* has 1 or more types *Vylepšení/oprava svépomocí*.

Věnování se práci na bydlení(Relator) mediates *Osoba*(Kind). Relationship has name 'RELATED'.

- Type *Osoba* has exactly 1 type *Věnování se práci na bydlení*.
- Type *Věnování se práci na bydlení* has exactly 1 type *Osoba*.

Věnování se práci na bydlení(Relator) mediates *Práce na bydlení*(Kind). Relationship has name 'RELATED'.

- Type *Práce na bydlení* has exactly 1 type *Věnování se práci na bydlení*.
- Type *Věnování se práci na bydlení* has exactly 1 type *Práce na bydlení*.

Výdaje na vylepšení/opravu svépomocí za posl. 12 měsíců(Quality) characterizes *Vylepšení/oprava svépomocí*(Relator). Relationship has name 'HAVE'.

- Type *Vylepšení/oprava svépomocí* has exactly 1 type *Výdaje na vylepšení/opravu svépomocí za posl. 12 měsíců*.
- Type *Výdaje na vylepšení/opravu svépomocí za posl. 12 měsíců* has exactly 1 type *Vylepšení/oprava svépomocí*.

Výdaje na vylepšení/opravu svépomocí za posl. 12 měsíců(Quality) is in relation with *Výdaje na vylepšení/opravu svépomocí za posl. 12 měsíců Data*. List of the attributes for the *Výdaje na vylepšení/opravu svépomocí za posl. 12 měsíců Data* class: identifier, 1a. Kolik jste utratil(a) za materiály které jste spotřeboval(a) v posl. 12 měsících? - Spotřebitelé/uživatelé, 2. Kolik jste utratil(a) za materiály které jste spotřeboval(a) v

Obrázek C.3 Model „Bydlení“ pro případovou studií(stránka 3)

posl. 12 měsících?.

- Type *Výdaje na vylepšení/opravu svépomocí za posl. 12 měsíců Data* has exactly 1 type *Výdaje na vylepšení/opravu svépomocí za posl. 12 měsíců*.
- Type *Výdaje na vylepšení/opravu svépomocí za posl. 12 měsíců* has exactly 1 type *Výdaje na vylepšení/opravu svépomocí za posl. 12 měsíců Data*.

Velikost bytu(Quality) characterizes *Byt*(Kind). Relationship has name 'HAVE'.

- Type *Byt* has exactly 1 type *Velikost bytu*.
- Type *Velikost bytu* has exactly 1 type *Byt*.

Velikost bytu(Quality) is in relation with *Velikost bytu Data*. List of the attributes for the *Velikost bytu Data* class: identifier, 2. Velikost bytu.

- Type *Velikost bytu Data* has exactly 1 type *Velikost bytu*.
- Type *Velikost bytu* has exactly 1 type *Velikost bytu Data*.

Druh bytu(Mode) characterizes *Byt*(Kind). Relationship has name 'HAVE'.

- Type *Byt* has exactly 1 type *Druh bytu*.
- Type *Druh bytu* has exactly 1 type *Byt*.

Druh bytu(Mode) is in relation with *Druh bytu Data*. List of the attributes for the *Druh bytu Data* class: identifier, 1. Druh bytu / domu.

- Type *Druh bytu Data* has exactly 1 type *Druh bytu*.
- Type *Druh bytu* has exactly 1 type *Druh bytu Data*.

Výdaje na rekonstrukci bytu za posl. 12 měsíců(Quality) characterizes *Rekonstrukce bytu*(Relator). Relationship has name 'HAVE'.

- Type *Rekonstrukce bytu* has exactly 1 type *Výdaje na rekonstrukci bytu za posl. 12 měsíců*.
- Type *Výdaje na rekonstrukci bytu za posl. 12 měsíců* has exactly 1 type *Rekonstrukce bytu*.

Výdaje na rekonstrukci bytu za posl. 12 měsíců(Quality) is in relation with *Výdaje na rekonstrukci bytu za posl. 12 měsíců Data*. List of the attributes for the *Výdaje na rekonstrukci bytu za posl. 12 měsíců Data* class: identifier, 5. Kolik jste utratili za rekonstrukci bytu/domu v posl. 12 měsících?.

- Type *Výdaje na rekonstrukci bytu za posl. 12 měsíců Data* has exactly 1 type *Výdaje na rekonstrukci bytu za posl. 12 měsíců*.

4

Obrázek C.4 Model „Bydlení“ pro případovou studií(stránka 4)

- Type *Výdaje na rekonstrukci bytu za posl. 12 měsíců* has exactly 1 type *Výdaje na rekonstrukci bytu za posl. 12 měsíců Data*.

Výdaje na koupi bytu za posl. 12 měsíců(Quality) characterizes *Nákup bytu*(SubKind). Relationship has name 'HAVE'.

- Type *Nákup bytu* has exactly 1 type *Výdaje na koupi bytu za posl. 12 měsíců*.
- Type *Výdaje na koupi bytu za posl. 12 měsíců* has exactly 1 type *Nákup bytu*.

Výdaje na koupi bytu za posl. 12 měsíců(Quality) is in relation with *Výdaje na koupi bytu za posl. 12 měsíců Data*. List of the attributes for the *Výdaje na koupi bytu za posl. 12 měsíců Data* class: identifier, 4. Kolik jste utratili za koupi bytu/domu v posl. 12 měsících?.

- Type *Výdaje na koupi bytu za posl. 12 měsíců Data* has exactly 1 type *Výdaje na koupi bytu za posl. 12 měsíců*.
- Type *Výdaje na koupi bytu za posl. 12 měsíců* has exactly 1 type *Výdaje na koupi bytu za posl. 12 měsíců Data*.

Počet obyvatel obce(Quality) characterizes *Obec*(Kind). Relationship has name 'HAVE'.

- Type *Obec* has exactly 1 type *Počet obyvatel obce*.
- Type *Počet obyvatel obce* has exactly 1 type *Obec*.

Počet obyvatel obce(Quality) is in relation with *Počet obyvatel obce Data*. List of the attributes for the *Počet obyvatel obce Data* class: identifier, 3. Velikost místa bydliště.

- Type *Počet obyvatel obce Data* has exactly 1 type *Počet obyvatel obce*.
- Type *Počet obyvatel obce* has exactly 1 type *Počet obyvatel obce Data*.

Obec(Kind) is in relation with *Obec Data*. List of the attributes for the *Obec Data* class: identifier, 4. Která z násl. možností nejlépe popisuje místo Vašeho bydliště?.

- Type *Obec Data* has exactly 1 type *Obec*.
- Type *Obec* has exactly 1 type *Obec Data*.

Obec(Kind) is in part-whole relation with *Adresa*(Kind).

- Type *Adresa* has exactly 1 type *Obec*.
- Type *Obec* has 1 or more types *Adresa*.

Obec is integral part of *Adresa*.
Adresa(Kind) mediates *Bydliště*(Relator).

Obrázek C.5 Model „Bydlení“ pro případovou studii(stránka 5)

- Type *Bydliště* has 1 or more types *Adresa*.
- Type *Adresa* has exactly 1 type *Bydliště*.

Bydliště(Relator) mediates *Domácnost*(Collective). Relationship has name 'RELATED'.

- Type *Domácnost* has exactly 1 type *Bydliště*.
- Type *Bydliště* has exactly 1 type *Domácnost*.

Je členem(Relator) mediates *Domácnost*(Collective). Relationship has name 'RELATED'.

- Type *Domácnost* has 1 or more types *Je členem*.
- Type *Je členem* has exactly 1 type *Domácnost*.

Je členem(Relator) mediates *Člen domácnosti*(Role). Relationship has name 'RELATED'.

- Type *Člen domácnosti* has exactly 1 type *Je členem*.
- Type *Je členem* has exactly 1 type *Člen domácnosti*.

Měsíční výdaje za nájem včetně vody a energií(Quality) characterizes *Domácnost*(Collective). Relationship has name 'HAVE'.

- Type *Domácnost* has exactly 1 type *Měsíční výdaje za nájem včetně vody a energií*.
- Type *Měsíční výdaje za nájem včetně vody a energií* has exactly 1 type *Domácnost*.

Měsíční výdaje za nájem včetně vody a energií Data is in relation with *Měsíční výdaje za nájem včetně vody a energií*(Quality). List of the attributes for the *Měsíční výdaje za nájem včetně vody a energií Data* class: identifier, 3. Jaké jsou celkové měsíční výdaje Vaší domácnosti na nájem (včetně vody a energií) za měsíc?.

- Type *Měsíční výdaje za nájem včetně vody a energií* has exactly 1 type *Měsíční výdaje za nájem včetně vody a energií Data*.
- Type *Měsíční výdaje za nájem včetně vody a energií Data* has exactly 1 type *Měsíční výdaje za nájem včetně vody a energií*.

Způsob vytápění domácnosti(Mode) characterizes *Domácnost*(Collective). Relationship has name 'HAVE'.

- Type *Domácnost* has exactly 1 type *Způsob vytápění domácnosti*.
- Type *Způsob vytápění domácnosti* has exactly 1 type *Domácnost*.

6

Obrázek C.6 Model „Bydlení“ pro případovou studii(stránka 6)

Způsob vytápění domácnosti(Mode) is in relation with *Způsob vytápění domácnosti Data*. List of the attributes for the *Způsob vytápění domácnosti Data* class: identifier, 6. Jakým způsobem vytápíte Vaši domácnost?.

- Type *Způsob vytápění domácnosti Data* has exactly 1 type *Způsob vytápění domácnosti*.
- Type *Způsob vytápění domácnosti* has exactly 1 type *Způsob vytápění domácnosti Data*.

Kraj Data is in relation with *Kraj*(Kind). List of the attributes for the *Kraj Data* class: identifier, 2. Kraj.

- Type *Kraj* has exactly 1 type *Kraj Data*.
- Type *Kraj Data* has exactly 1 type *Kraj*.

Region(Kind) is in relation with *Region Data*. List of the attributes for the *Region Data* class: identifier, 1. Region.

- Type *Region Data* has exactly 1 type *Region*.
- Type *Region* has exactly 1 type *Region Data*.

Obrázek C.7 Model „Bydlení“ pro případovou studií(stránka 7)

Bibliografie

1. GUIZZARDI, Giancarlo. *Ontological Foundations for Structural Conceptual Models*. Centre for Telematics a Information Technology, Telematica Instituut, 2005. PhD thesis. ISBN 90-75176-81-3.
2. GROSE, T.J.; DONEY, G.C.; BRODSKY, S.A. *Mastering XMI: Java Programming with XMI, XML and UML*. Wiley, 2002. OMG. ISBN 9780471265566. Dostupné také z: `https://books.google.cz/books?id=RHlGkb-ZS-sC`.
3. UHNÁK, Peter; PERGL, Robert. The OpenPonk modeling platform. In: *Proceedings of the 11th edition of the International Workshop on Smalltalk Technologies*. 2016, s. 1–11.
4. CASTRO, E. *HTML for the World Wide Web*. Peachpit Press, 2003. HTML for the World Wide Web with XHTML & CSS. ISBN 9780321130075. Dostupné také z: `https://books.google.cz/books?id=Dq9L3KkDOFIC`.
5. SPINELLIS, Diomidis. Git. *IEEE Software*. 2012, roč. 29, č. 3, s. 100–101. Dostupné z DOI: `10.1109/MS.2012.61`.
6. SKERSYS, Tomas; DANENAS, Paulius; BUTLERIS, Rimantas. Extracting SBVR business vocabularies and business rules from UML use case diagrams. *Journal of Systems and Software*. 2018, roč. 141, s. 111–130.
7. OLDEVIK, Jon; NEPLE, Tor; GRØNMO, Roy; AAGEDAL, Jan; BERRE, Arne-J. Toward Standardised Model to Text Transformations. 2005, s. 239–253. ISBN 978-3-540-32093-7.
8. BROSCH, Petra; RANDAK, Andrea. Position paper: m2n—A tool for translating models to natural language descriptions. *Electronic Communications of the EASST*. 2010, roč. 34.
9. GRØNMO, Roy; OLDEVIK, Jon. An empirical study of the UML model transformation tool (UMT). *Proc. First Interoperability of Enterprise Software and Applications, Geneva, Switzerland*. 2005, s. 16.
10. EVANS, A.S. Reasoning with UML class diagrams. 1998, s. 102–113. Dostupné z DOI: `10.1109/WIFT.1998.766304`.
11. ESHUIS, Rik. Symbolic model checking of UML activity diagrams. *ACM Transactions on Software Engineering and Methodology (TOSEM)*. 2006, roč. 15, č. 1, s. 1–38.
12. RISCHPATER, Ray. *Application development with qt creator*. Packt Publishing Birmingham, 2013.
13. MARTINEZ, Wendy L. Graphical user interfaces. *WIREs Computational Statistics*. 2011, roč. 3, č. 2, s. 119–133. Dostupné z DOI: `https://doi.org/10.1002/wics.150`.
14. SEDGEWICK, Robert; WAYNE, Kevin. *Algorithms*. Addison-wesley professional, 2011.

15. HEESCH, Dimitri van. *Doxygen* [online]. [cit. 2024-05-04]. Dostupné z: `https://www.doxygen.nl/`.

16. OLAN, Michael. Unit testing: test early, test often. *Journal of Computing Sciences in Colleges*. 2003, roč. 19, č. 2, s. 319–328.

17. THELIN, Johan. *Foundations of Qt development*. Springer, 2007.

18. REMMARK, a.s. *OpeNest* [online]. [cit. 2024-05-04]. Dostupné z: `https://www.openest.cz/`.

19. REMMARK, a.s. *Remmark* [online]. [cit. 2024-05-04]. Dostupné z: `https://www.remmark.cz/`.

20. ŠIMON, Matouš. *Ontologické modelování dat pro projekt Datového inkubátoru*. 2023. B.S. thesis. České vysoké učení technické v Praze. Vypočetní a informační centrum.

21. JAN, Pecka. *Využití ontologické analỳzy pro zajištění sémantické interoperability heterogenních dat*. 2023. B.S. thesis. České vysoké učení technické v Praze. Vypočetní a informační centrum.

22. VÁCLAV, Šír. *Využití ontologické analỳzy pro zajištění sémantické interoperability heterogenních dat*. 2022. B.S. thesis. České vysoké učení technické v Praze. Vypočetní a informační centrum.

23. JANA, Martínková. *Využití ontologické analỳzy pro zajištění sémantické interoperability marketingovỳch dat*. 2021. B.S. thesis. České vysoké učení technické v Praze. Vypočetní a informační centrum.

24. HAMPEIJS, Ondřej. *Využití ontologické analỳzy pro zajištění sémantické interoperability heterogenních dat*. 2023. B.S. thesis. České vysoké učení technické v Praze. Vypočetní a informační centrum.

25. FLYVBJERG, Bent. Case study. *The Sage handbook of qualitative research*. 2011, roč. 4, s. 301–316.

26. ATLURI, Rajeev; SRIDHAR, K.V. Design of multilingual fire panel user interface using embedded Qt. In: *2015 International Conference on Communications and Signal Processing (ICCSP)*. 2015, s. 1644–1648. Dostupné z DOI: `10.1109/ICCSP.2015.7322797`.

Obsah příloh

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `impl` ..... zdrojové kódy implementace(samotný program a jednotkové testy)
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
- `doc` ..... adresář s Doxygen dokumentací k projektu