Zadání diplomové práce

Název:	Použití technologií RDF pro zefektivnění využití konceptuálních modelů v OntoUML
Student:	Bc. Tereza Machačová
Vedoucí:	Ing. Marek Suchánek
Studijní program:	Informatika
Obor / specializace:	Manažerská informatika
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

OntoUML umožňuje tvorbu strukturálních konceptuálních modelů s využitím konceptů z vyšší ontologie UFO. Tím je dosaženo vyšší expresivity v porovnání s jinými tradičními modelovacími jazyky. V principu jde ale především o diagramy pro dokumentaci dané domény. Cílem této práce je prozkoumat, navrhnout a demonstrovat využití technologií RDF a existující podpory pro zvýšení interoperability OntoUML modelů.

- Seznamte se s technologiemi RDF a existující podporou pro OntoUML/UFO.
- Popište možnosti využití technologií RDF (RDF, RDFS/OWL, SPARQL, SHACL/ShEx, RML apod.) v oblasti konceptuálního modelování.
- Navrhněte, jak technologie využít pro zefektivnění vybraných činnosti (např. validace a verifikace, vyhodnocování kvality, instanciace a prohledávání) v OntoUML modelech převedených do RDF.
- Demonstrujte užití RDF technologií na netriviálních příkladech z reálného světa (podniky, státní instituce, občanskoprávní vztahy atd.).
- Zhodnoťte přínosy použití RDF technologií ve srovnání s tradičními přístupy konceptuálního modelování (např. technologie pro UML).

Elektronicky schválil/a Ing. David Buchtela, Ph.D. dne 23. září 2022 v Praze.

Diplomová práce

Použití technologií RDF pro zefektivnění využití konceptuálních modelů v OntoUML

Bc. Tereza Machačová

Katedra softwarového inženýrství
Vedoucí práce: Ing. Marek Suchánek

1. ledna 2023

Poděkování

Ráda bych poděkovala vedoucímu své práce Ing. Markovi Suchánkovi, za konzultace, poskytnuté materiály, rady, a i všechnu další pomoc. Zejména za ochotný přátelský přístup a velmi rychlé zodpovídání dotazů.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu 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 předpisu, 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 1. ledna 2023 . . . .

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2023 Tereza Machačová. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.

Odkaz na tuto práci

Machačová, Tereza. *Použití technologií RDF pro zefektivnění využití konceptuálních modelů v OntoUML*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Abstrakt

Práce se věnuje modelovacímu jazyku OntoUML a především možnostem jeho využití ve spojení s technologiemi RDF. Jazyk OntoUML se zaměřuje na reprezentaci reality a je vhodný převážně k pochopení a přehledu modelované domény. Kvalitu modelu lze měřit dle různých metrik, které lze automatizovat pomocí dotazovacích a validačních jazyků. Tyto jazyky operují nad daty v jazyce pro popis zdrojů RDF, ve kterém lze s pomocí dalších ontologických jazyků zachytit informace OntoUML modelu. V práci jsou představeny všechny zmíněné technologie a na vytvořených modelech jsou provedeny měření kvality dle různých kritérií. Dotazy, validace i měření kvality jsou přizpůsobeny konceptům OntoUML modelů, které, narozdíl od například ER modelů, obsahují stereotypy s mnoha pravidly. I kritéria měření kvality jsou tak pro ně v některých ohledech odlišná. Součástí práce je i návrh exportu do technologií RDF pro nástroj OpenPonk, vyvíjený na FIT ČVUT v Praze, ve kterém lze vytvářet OntoUML modely. Návrh poukazuje na nedostatky současné podoby gUFO, a popisuje úpravy, jejichž začleněním by se rozšířily možnosti využití RDF technologií pro OntoUML modely.

Klíčová slova OntoUML, UFO, RDF, OWL, gUFO, SPARQL, SHACL, metriky kvality, konceptuální modelování, interoperabilita, OpenPonk

Abstract

This diploma thesis deals with OntoUML modelling language and especially the possibilities of its use via RDF technologies. OntoUML is focused on representation of reality and is suitable for understanding the modelled domain and its overview. Quality of the model can be measured by various metrics that can be automated by query languages and validation languages. These languages operate on data described by resource description language called RDF, that can express OntoUML model information via its ontological languages. All of the technologies mentioned are explained and quality is measured according to various criteria on created models. Querying, validation and quality metrics are adjusted to OntoUML concepts that use stereotypes with many rules, unlike ER models for example. The thesis also includes an export design to RDF technologies designed for OpenPonk tool developed on FIT CTU in Prague, that supports OntoUML modelling. The design points out some missing concepts of the current gUFO, and proposes their modifications which, if incorporated, would extend the possibilities of using RDF technologies for OntoUML models.

Keywords OntoUML, UFO, RDF, OWL, gUFO, SPARQL, SHACL, quality metrics, conceptual modelling, interoperability, OpenPonk

Obsah

Úvod **1**

1 Cíl práce **3**

2 Analýza modelovacích a ontologických jazyků **5**
2.1 Modelovací jazyky 6
2.1.1 BPMN 6
2.1.2 UML 6
2.1.3 OntoUML 7
2.2 RDF 7
2.3 Ontologické jazyky pro RDF 8
2.3.1 RDFS 9
2.3.2 OWL 9
2.3.3 gUFO 9
2.3.3.1 Reprezentace UFO stereotypů v gUFO 10

3 Analýza dotazovacích a validačních jazyků **17**
3.1 Turtle 18
3.2 SPARQL 18
3.2.1 Syntax SPARQL 19
3.2.2 Dotazování 20
3.3 SHACL 23
3.3.1 Typy tvarů 23
3.3.2 Komponenty omezení 25
3.4 ShEx 27
3.5 RML 28

4 Měření kvality OntoUML modelů **33**
4.1 Metriky kvality 34

4.1.1 Vnímaná a měřená kvalita 34
4.1.2 Měření kvality spojených OntoUML modelů 35
4.2 Ukázkový model 36
4.3 Propojení modelů 39
4.4 Validace v SHACL 42
4.5 Dotazování ve SPARQL 45
4.5.1 Měření kvality dle kritérií studie 46
4.5.2 Detekce chyb a anti-patternů 49
4.5.3 Dotazování nad daty 55

5 Nástrojová podpora **57**
5.1 Porovnání nástrojů OpenPonk a Visual Paradigm 58
5.1.1 Využití nástrojů v práci 58
5.1.2 Nastavení vlastností vazeb 59
5.1.3 Pohyb po pracovní ploše 60
5.1.4 Funkce Visual Paradigm 61
5.1.5 Závěr porovnání nástrojů 61
5.2 Návrh exportu do gUFO pro nástroj OpenPonk 62
5.2.1 Směr vazeb 62
5.2.2 Návrh funkcí a stereotypů 63
5.2.3 Shrnutí návrhu exportu 75

6 Zhodnocení **77**
6.1 Možnosti OntoUML v RDF 77
6.2 Měření kvality modelů 78
6.3 Další využití RDF pro modely a instance 79
6.4 Nástrojová podpora 79

Závěr **81**

Literatura **83**

A Seznam použitých zkratek **87**

B Obsah přiloženého CD **89**

Seznam obrázků

4.1 Model Posilovna 37
4.2 Model Přehled invalidity 40
4.3 Model Příčiny invalidity 41
4.4 Chybný model Přehled invalidity 52

5.1 Asociace v OpenPonk 59
5.2 Asociace ve Visual Paradigm 60
5.3 Hierarchie stereotypů 66

Úvod

V současnosti existuje mnoho modelovacích jazyků určených k reprezentaci reálného světa. Tyto jazyky mají obvykle nějaké specifické zaměření, například softwarový vývoj, a nejsou tak vždy vhodné k modelování reality mimo informační technologie. Jazyk OntoUML, založený na ontologii UFO, se zaměřuje přímo na modelování reality. Detailní reprezentace je dosaženo za pomoci stereotypů entit a vazeb, které jim dodávají další vlastnosti. [1]

OntoUML je vhodné převážně k pochopení a přehledu modelované domény. Spojením více modelů lze často doménu popsat detailněji, či obsáhnout v jednom modelu více domén najednou. Při vytváření velkých modelů, které obvykle vznikají spojením menších modelů, ale může také docházet k různým problémům. U velkých modelů se nelze vyhnout nižší čitelnosti a přehlednosti oproti menším modelům. Ke snížené přehlednosti přispěje i nevyhnutelné křížení čar a kontrola kvality modelu se tak stane obtížnější. Některé modelovací nástroje umožňují automatické verifikace detekující chyby v modelu, ale není to samozřejmostí. Kritéria kvality lze do jisté míry automatizovat, ale některá kritéria jsou subjektivní a závislá na znalosti domény [2]. Rychlé a pohodlné spojení několika modelů dohromady navíc neumožňují všechny modelovací nástroje.

S řešením těchto problémů mohou pomoci technologie RDF. RDF je jazyk pro popis zdrojů, zachycující informace v trojicích, takzvaných tripletech [3]. Pro jazyk RDF existují vlastní ontologické jazyky, díky kterým je možné v RDF reprezentovat například i OntoUML modely [4]. Při převedení do textového formátu je mnohem snazší modely propojit a lze provádět automatizované testy kvality pomocí dotazovacích a validačních jazyků určených pro RDF. Pro měření kvality modelu lze využít kritéria stanovena studiemi, či měřit dle vlastních pravidel. Ověřovat lze i přítomnost chyb či takzvaných anti-patternů, což jsou struktury v modelu které nejsou chybné, ale často k chybám vedou.

V práci budou představeny všechny zmíněné technologie a na vytvořených

modelech budou provedeny měření kvality dle různých kritérií. Práce s modely, tedy dotazování, validace i měření kvality, budou přizpůsobeny konceptům OntoUML modelů. OntoUML má mnohem více pravidel než mají UML a ER modely a pravidla jsou složitější. Proto je výhodné využít dotazovací a validační jazyky i na kontrolu samotného modelu a ne pouze dat. Kvůli stereotypům obsahujícím další informace bude i odlišné měření kvality. Pravidla, která platí pro kritéria kvality u jiných modelů, mohou být pro OntoUML modely odlišné.

Součástí práce bude i návrh exportu do technologií RDF pro nástroj OpenPonk, ve kterém lze vytvářet OntoUML modely. OpenPonk je vyvíjený na FIT ČVUT v Praze. V rámci návrhu bude OntoUML porovnáno s jeho reprezentací v technologiích RDF, nazývanou gUFO. Kromě návrhu implementace stereotypů z dokumentace bude návrh obsahovat i funkce kterými OpenPonk disponuje, například možnost vytvářet poznámky. Bude se jednat pouze o teoretický návrh s analýzou potřeb a ukázkami výstupů, bez samotné implementace.

Kapitola 1

Cíl práce

Cílem této práce je zanalyzovat technologie RDF a navrhnout jejich efektivní využití pro modely v jazyce OntoUML. Nejdříve budou představeny použité jazyky, převážně gUFO sloužící pro zápis OntoUML, SPARQL pro dotazování a SHACL pro validaci.

Vybrané modely budou nejprve převedeny do RDF pomocí gUFO. Dotazovat se bude nejen nad daty, ale i nad modelem samotným a to podle různých kritérií měření kvality, vycházejících ze studií či praxe. Měření kvality bude přizpůsobeno konceptům OntoUML, které narozdíl od ER modelů či UML modelů musí dodržovat více pravidel. Dalším rozdílem je vyšší informační hodnota každé entity a vazby díky stereotypům. Modely se budou validovat jazykem SHACL, validace bude využita na kontrolu správnosti dat a detekci chyb a anti-patternů. Modely, se kterými se bude pracovat, budou popisovat existující doménu a jsou inspirovány reálnými datovými sadami.

Součástí práce bude také návrh exportu do gUFO pro nástroj OpenPonk, vyvíjený na FIT ČVUT v Praze. V současné podobě dokumentace gUFO je mnoho důležitých vztahů stále nedefinovaných či definovaných dle staré verze. Návrh popíše zjištěné nedostatky a navrhne úpravy, jejichž začleněním by se rozšířily možnosti využití RDF technologií pro OntoUML modely. Většina změn se bude zabývat aktualizací na OntoUML 2.0 a přidání chybějících stereotypů. Návrh také bude obsahovat reprezentaci pro funkce nástroje OpenPonk, kterými je například možnost vytvářet poznámky v modelu. Bude se jednat o teoretický návrh s popisem řešení problémů a ukázkami výstupů, bez samotné implementace.

Těmto dílčím cílům odpovídá i struktura práce. Všechna zjištění budou shrnuta v kapitole Zhodnocení.

Kapitola **2**

Analýza modelovacích a ontologických jazyků

V této kapitole budou popsány použité modelovací, popisovací a ontologické jazyky. Cílem je jednotlivé jazyky stručně popsat, seznámit se s nimi a porozumět tomu, jak na sebe navazují a využívají vlastností ostatních jazyků. Budou představeny některé známé modelovací jazyky a jejich zaměření a bude vysvětlen odlišný přístup jazyka OntoUML. Následovat budou informace o RDF, jazyce pro popis zdrojů, a ontologických jazycích pro něj určených. Tyto ontologické jazyky umožňují zpřesnit a rozšířit možnosti RDF. Nejdetailněji bude popsán ontologický jazyk gUFO, což je RDF varianta ontologie UFO využívané v OntoUML. Pro jazyk gUFO jsou uvedeny příklady reprezentace OntoUML stereotypů, jsou zde uvedeny i příklady množinových omezení jako je *disjoint* a *complete*.

Ontologické jazyky jsou psány v pořadí dle toho, jak na sebe navazují, další uvedený jazyk tak rozšiřuje předchozí jazyky. Transformace OntoUML do kódové podoby tedy využívá všechny ontologické jazyky z této kapitoly, a při využití gUFO jsou tak pro rozšíření RDF využity i RDFS a OWL. V ukázkách reprezentace UFO stereotypů v gUFO se tedy objevují i pojmy z ostatních ontologických jazyků.

2.1 Modelovací jazyky

V současnosti existuje mnoho jazyků které se zabývají modelováním nějaké domény. Mezi nimi je i mnoho jazyků určených k modelování reprezentace reálného světa. Tyto jazyky mají obvykle nějaký konkrétní účel, omezený na doménu pro kterou jsou využívány. To jim často brání se plně zaměřit na to, aby byly věrné realitě. Zatímco například jazyk UML vznikl se zaměřením na software design, ontologie *UFO*, Unified Foundational Ontology, je zaměřena právě na co největší podobu reálnému světu. Na UFO je založen modelovací jazyk OntoUML, hlavní předmět zájmu této práce. Různé jazyky mají navíc různé přístupy, některé se více zaměřují na vztah celku a části, jiné na hierarchii tříd a dědění. UFO se zabývá širším spektrem a kombinuje vše výše zmíněné. Zároveň řeší některé problémy konceptuálního modelování, které do té doby nebyly nijak uspokojivě vyřešeny. [1]

Stručně budou představeny některé známé modelovací jazyky, UML a BPMN, pro ukázku jejich zaměření a odlišnosti oproti OntoUML.

2.1.1 BPMN

BPMN, Business Process Model and Notation, je jazyk sloužící pro modelování podnikových procesů. Interní podnikové procesy jsou zachyceny v grafické notaci a pomáhají v orientaci návaznosti jednotlivých částí procesů. Tyto procesy představují komunikaci a rozhodování situací v rámci podniku. Obvykle zahrnují i komunikaci se zákazníkem, bývají propojené s ostatními procesy a také s procesy jiných podniků. Některé nástroje na tvorbu BPMN zároveň umožňují orchestraci modelu, která dokáže simulovat fungování procesů a případně odhalit slabé místo. [5]

2.1.2 UML

UML, Unified Modeling Language, je univerzální modelovací jazyk který se používá převážně při vývoji softwaru. Existuje mnoho různých variant UML diagramů sloužících pro různé účely, dva nejdůležitější typy diagramů jsou diagramy struktury a diagramy chování. Mezi diagramy struktury patří například diagram tříd, vizualizující třídy a jejich propojení, nebo diagram balíčků, zobrazující vztahy mezi jednotlivými komponentami, které vytváří systém. Mezi hojně využívané diagramy chování patří diagram případů užití, také nazývaný *use case diagram*. Ten slouží k popsání funkcí systému a jak uživatelé se systémem komunikují, ale již se nezabývá způsobem, jak toho dosáhnout, jak jsou jednotlivé funkce v systému implementovány. Dalším známým diagramem chování je sekvenční diagram, který zobrazuje podnikové procesy a chronologický sled pořadí zpráv a interakcí. UML lze také využít na různé databázové modely. [6]

2.1.3 OntoUML

Jak již bylo zmíněno, modelovacích jazyků je mnoho a mají nějaké určité zaměření. Zaměřením OntoUML je abstrakce reality dle určité konceptualizace, dle určitého úhlu pohledu. Modely jsou využívány k přehlednému zobrazení vztahů v reálném světě, nad kterým se dá poté dále pracovat. Vzhledem k podstatě jazyka OntoUML a jeho způsobu popisování vztahů, modelovaná realita často slouží k pochopení dané problematiky zobrazené v modelu. OntoUML lze využít na identifikaci a vysvětlení faktů v doméně a může pomoci s řešením nejasností. Modely zároveň slouží jako zdroj informací se sjednocenými pojmy, které jsou užitečné při diskutování domény. [1]

OntoUML je jazyk pro konceptuální modelování vytvořený jako rozšíření UML a založený na ontologii UFO, Unified Foundational Ontology, kterou vytvořil Giancarlo Guizzardi. Pro vytvoření OntoUML byla ontologie navržena tak, aby vznikla ontologicky založená verze UML která může být využita jako vhodný konceptuální modelovací jazyk. UML již nabízí mechanismy jako jsou třídy, vztahy, atributy a podtypy, ale pro úplnou a podrobnou reprezentaci byly přidány ještě další vlastnosti, převážně stereotypy, a to jak stereotypy entit, tak stereotypy vazeb. Tyto stereotypy určují další vlastnosti entit. Modelování v OntoUML slouží k reprezentaci reality, ale vždy záleží na úhlu pohledu osoby, která model vytváří, jelikož na různé skutečnosti se dá nahlížet různě. Například banku lze vnímat jako roli budovy, nebo jako druh instituce, a dle tohoto úhlu pohledu se liší, jak bude v modelu reprezentována. [1]

Stereotypy přidávají svým nositelům další vlastnosti, ale zároveň přináší pravidla, která s nimi souvisí, a která musí být při modelování dodržována. Tato pravidla a z nich související omezení mohou vést k chybám v modelu, pokud nejsou splněna. Některé nástroje, například OpenPonk vyvíjený na FIT ČVUT v Praze, dokáží tyto chyby detekovat. V OntoUML se lze setkat i s *anti-patterny*, což jsou struktury v modelu, které nejsou nutně chybné, ale často jejich výskyt vede ke vzniku chyby či nejasnosti. Anti-patterny by tedy měly být kontrolovány a případně opraveny. Jejich detekce je také součástí nástroje OpenPonk. [1, 7]

2.2 RDF

RDF, Resource Description Framework, je jazyk pro popis zdrojů. Zdroje jsou popisovány pomocí *tripletů*, trojic, skládajících se ze subjektu, predikátu a objektu. Subjekt je popisovaný zdroj který má nějakou vlastnost, predikát určuje onu vlastnost a objekt konkrétní hodnotu vlastnosti. Tyto části trojice se také dají nazývat jako podmět, přísudek a předmět. RDF udává, jak informace zapsat, ale nemá vlastní definovanou syntax a lze ho vyjádřit pomocí různých formátů a jazyků. Jedním z nich je XML. Jelikož je jazyk XML strojové zpracovatelný, machine-actionable, stroje mohou XML dokument, a tedy i takto zapsaný jazyk RDF, číst a pracovat s ním. [3]

Díky tomu, že jazyk RDF lze vyjádřit pomocí různých formátů a jazyků, je také velmi flexibilní. Je tak možné v něm vyjádřit téměř cokoliv, tedy i OntoUML modely, přesněji základní informace OntoUML modelů. K přesnější a úplné transformaci se všemi informacemi slouží ontologické jazyky. RDF je zde popsáno pouze stručně, jelikož v pozdější sekci 3.2 jsou vlastnosti i syntax vysvětleny a ukázány na příkladech pro dotazovací jazyk RDF, kterým je SPARQL.

RDF používá k identifikaci zdrojů *URI*. URI vzniká spojením URL a URN. URL určuje adresu zdroje a URN název zdroje, URI tedy obsahuje adresu i název zdroje. Používat místo názvů URI je vhodné z několika důvodů. Pokud by například triplet pojednával o nějaké osobě identifikované jejím jménem, nemuselo by být jasné, o jakou osobu se jedná, jelikož může existovat více osob se stejným křestním jménem i příjmením. Přiřazením URI je daná osoba odlišitelná od ostatních. Pro jednu věc také může existovat několik různých pojmenování. V rámci jednoho modelu příliš nezáleží na tom, které z existujících pojmenování bude zvoleno jako název, ale při propojování několika modelů už by ona stejná věc mohla být nazývána dvěma a více názvy. Data by se pak tedy nepropojila tak jak je žádoucí a vnášelo by to do modelu nekonzistence. Odkazováním na URI přímo popisující danou věc se tyto problémy dají mitigovat, až zcela eliminovat. Může také nastat i opačný problém, a to ten, že několik věcí může mít stejný název. To nastává převážně u predikátů. Predikát „jméno“ určující jméno subjektu může být použit pro jméno osoby, ale zároveň pro název webových stránek, což jsou zcela odlišné věci. Proto je vhodné používat URI nejen pro subjekty a objekty, ale i pro predikáty. V RDF je zároveň možné používat takzvané prázdné uzly, *blank nodes*, které nemají URI, a tedy nenesou informaci. Tyto uzly slouží k propojení s ostatními částmi grafu. V grafu může být takovýchto uzlů více, a proto využívají své vlastní identifikátory, jelikož na rozdíl od URI a hodnot nejsou považovány za skutečnou část RDF grafu. [3]

2.3 Ontologické jazyky pro RDF

Jazyk RDF lze již v základní podobě bez přidaných ontologií využít na reprezentaci modelu. Vznikne ale minimalistický model, obsahující popisy, zdroje a vlastnosti, ale nebudou tam obsaženy všechny informace, například stereotypy nebo multiplicita. Aby se s daty mohlo efektivně pracovat, je vhodné využít ontologie a slovníky, které lépe popíší vztahy mezi daty a souvisejícími pojmy. Kromě přesné definice by u termínu měly být uvedeny také vztahy s ostatními termíny. Z toho důvodu byly vytvořeny různé ontologické jazyky. S využitím vhodných ontologických jazyků lze OntoUML model převést do RDF se všemi informacemi. [1, 8]

2.3.1 RDFS

RDFS, RDF Schema, je sémantickým rozšířením RDF. Na rozdíl od XML může definovat sémantiku specifikací a umožňuje vytvářet vlastní ontologie, třídy a jejich vlastnosti. I přesto existuje spousta vlastností, které jsou v OntoUML využívány, ale v RDFS je nelze vyjádřit. To se týká například množinových omezení *disjoint* a *complete*, značících disjunktní a vyčerpávající množinu. Tedy, zda jsou v množině obsaženy všechny možnosti a zda je možné být instancí dvou a více možností najednou. V RDFS zároveň není možné vyjádřit multiplicitu vazeb. [1, 8]

2.3.2 OWL

Jelikož ani s využitím RDFS není možné plnohodnotně reprezentovat všechny informace a vztahy obsažené v OntoUML modelech, byly vyvíjeny další jazyky pro reprezentaci ontologií. Těmito jazyky byly DAML-ONT a OIL, ze kterých poté spojením a dalšími úpravami vznikl jazyk OWL, Ontology Web Language. OWL je jazyk pro reprezentaci znalostí v ontologii. Původní OWL je z roku 2004, roku 2009 vznikl OWL 2, ale při diskuzi OWL se obvykle předpokládá právě novější OWL 2. [1]

Podobně jako RDF využívá místo názvů URI, OWL využívá *IRI*, které je rozšířením URI o další povolené znaky. Kromě informací obsažených v již zmíněných terminologiích a slovnících může ontologie využívat získané znalosti k předpokladům. To se nazývá asertivní znalost. Když v klasické databázi nějaká informace není, považuje se za nepravdivou. V OWL se považuje za chybějící, a nelze tedy říct, zda je pravdivá či není. Vzhledem k asertivním znalostem je často vhodné a dokonce potřeba popsat nejen pravdivé vztahy, ale i nepravdivé vztahy. Kdyby nepravdivé vztahy nebyly popsány, předpokládalo by se, že tato informace pouze chybí, a je tedy potenciálně pravdivá. Na rozdíl od RDFS, OWL umožňuje vyjádřit disjunktnost tříd a multiplicitu. Multiplicita je, stejně jako v OntoUML, vyjádřena buď přesnou hodnotou, nebo rozsahem. Rozsah se skládá z minimální a maximální možné hodnoty, tedy minimální a maximální kardinality. Stejně jako v OntoUML může jedinec být instancí více tříd najednou. OWL zároveň umožňuje definovat vztahy jako symetrické, asymetrické, reflexivní, ireflexivní či tranzitivní. [9]

2.3.3 gUFO

Pro ontologii UFO, na níž je OntoUML založené, existuje její OWL reprezentace nazvaná gUFO. Vlastnosti a stereotypy využívané v OntoUML modelech jsou definovány i v gUFO, a je tak tedy možné transformovat OntoUML modely do RDF formátu i s informacemi o stereotypech. [4]

V gUFO tedy lze zachytit princip identity, sortalitu a stálost, nazývanou rigidita. Tyto pojmy jsou pro OntoUML stěžejní a musí být tedy definované

v cílovém jazyce, do kterého mají být modely převedeny. gUFO definuje stereotypy z UFO, a to jak stereotypy tříd, tak i stereotypy vazeb. Pro některé stereotypy tříd je dle pravidel UFO nutné využít i konkrétní stereotypy vazeb, využívané převážně u vztahů celek část. Tyto vztahy se řídí dalšími pravidly jako je povinnost či zaměnitelnost části/celku, a rolí jednotlivých částí v rámci celku. Z tohoto důvodu je jen pro vztahy celku a části v UFO několik stereotypů. Dalšími důležitými vztahy jsou Relatory s Mediation vazbami a vlastnosti charakterizující aspekty. Použitím stereotypu je již dána sortalita a rigidita objektu, stejně jako v UFO. Lze také označit třídy jako abstraktní. Tyto funkce a stereotypy jsou definovány i v gUFO, bohužel dva stereotypy v dokumentaci chybí. Vazba Containment, využívaná pro popsání vztahu mezi Quantity a její nádobou, a vazba Derivation, využívaná pro Material vazbu a Relator. [4]

Již existuje práce [10], která se zabývala výzkumem exportu modelu do RDF, se zaměřením na anti-patterny. Součástí byl software Menthor Editor, prostředí pro konceptuální modelování umožňující detekci anti-patternů a validaci, stejně jako OpenPonk. Nástroj také generoval kód pro sémantický web, tedy RDF a OWL. Součástí byl i export OntoUML do gUFO. Dle informace od vývojářů dostupné z [11], Menthor Editor již není nadále udržován, ale byl nahrazen OntoUML pluginem do nástroje Visual Paradigm, dostupným z [12], kde je všechny jeho funkce možné nadále využívat. Nástroj i s pluginem je popsán v sekci 5.1.

2.3.3.1 Reprezentace UFO stereotypů v gUFO

Většina z následujících příkladů byla převzaty z dokumentace gUFO [4]. Ostatní byly exportovány z jednoduchých OntoUML modelů vytvořených pro účely ukázky do gUFO pomocí nástroje Visual Paradigm.

V příkladech jsou prezentovány všechny stereotypy kromě dvou, a to Structuration a Containment. Structuration je pokročilý typ vazby, který se nevyužívá, a ani v práci tedy nebude popsán. Containment je druh vazby využívající se mezi Quantity a její nádobou, bohužel není v gUFO definován. V rámci návrhu exportu do gUFO v nástroji OpenPonk bude navržena transformace i pro něj, jelikož na rozdíl od Structuration je Containment rozšířený a běžně využívaný stereotyp.

Kind `Person`:

```
:Person rdf:type gufo:Kind.
```

`Man` a `Woman` je SubKind od Kind `Person`:

```
:Man rdf:type gufo:SubKind;
    rdfs:subClassOf :Person.
```

```
:Woman rdf:type gufo:SubKind;
    rdfs:subClassOf :Person.
```

Phase `Adult` představující vnitřní stav či vlastnost Kind `Person`:

```
:Adult rdf:type gufo:Phase;
    rdfs:subClassOf :Person.
```

`Student` je existenčně závislá Role od Kind `Person`:

```
:Student rdf:type gufo:Role;
    rdfs:subClassOf :Person.
```

Relator `Marriage` mezi dvěma Kind `Person`, propojeny vazbami Mediation:

```
:Marriage rdf:type owl:Class;
        rdfs:subClassOf gufo:Relator.

:marriageInvolves rdf:type owl:ObjectProperty;
                rdfs:subPropertyOf gufo:mediates;
                rdfs:domain :Marriage;
                rdfs:range :Person.
```

Ukázka na konkrétních instancích `John` a `Mary`:

```
:John rdf:type :Person.
:Mary rdf:type :Person.

:JohnMarysMarriage rdf:type :Marriage;
                :marriageInvolves :John,
                                  :Mary.
```

Material vazba `marriedWith` se pomocí `isDerivedFrom` dá odvodit od Relator `Marriage`, což je reprezentace vazby Derivation:

```
:marriedWith rdf:type owl:ObjectProperty,
             gufo:MaterialRelationshipType;
         rdfs:domain :Person;
         rdfs:range :Person;
         gufo:isDerivedFrom :Marriage.
```

`LivingThing` je Category, pod kterou patří Kind `Person`:

```
:LivingThing rdf:type gufo:Category.
:Person rdf:type gufo:Kind;
    rdfs:subClassOf :LivingThing.
```

`LivingAnimal` je PhaseMixin několika Phase s odlišnými principy identity, `LivingCat` s identitou Kind `Cat` a `LivingDog` s identitou Kind `Dog`:

```
:LivingThing rdf:type gufo:PhaseMixin.
:LivingCat rdf:type gufo:Phase.
:LivingDog rdf:type gufo:Phase.
:Cat rdf:type gufo:Kind.
:Dog rdf:type gufo:Kind.

:LivingCat rdfs:subClassOf :LivingThing.
:LivingDog rdfs:subClassOf :LivingThing.
:LivingCat rdfs:subClassOf :Cat.
:LivingDog rdfs:subClassOf :Dog.
```

`Customer` je RoleMixin několika Role s odlišnými principy identity, a to `CorporateCustomer` s identitou Kind `BusinessOrganisation` a `PersonalCustomer` s identitou Kind `Person`:

```
:Customer rdf:type gufo:RoleMixin.
:PersonalCustomer rdf:type gufo:Role.
:CorporateCustomer rdf:type gufo:Role.
:Person rdf:type gufo:Kind.
:BusinessOrganisation rdf:type gufo:Kind.

:PersonalCustomer rdfs:subClassOf :Customer.
:CorporateCustomer rdfs:subClassOf :Customer.
:PersonalCustomer rdfs:subClassOf :Person.
:CorporateCustomer rdfs:subClassOf :BusinessOrganisation.
```

`IllegalGoods` je Mixin rigidního Kind `Drugs` a anti-rigidní Role `StolenCar`:

```
:IllegalGoods rdf:type gufo:Mixin.
:Drugs rdf:type gufo:Kind.
:StolenCar rdf:type gufo:Role.
:Car rdf:type gufo:Kind.

:Drugs rdfs:subClassOf :IllegalGoods.
:StolenCar rdfs:subClassOf :IllegalGoods, :Car.
```

`Mass` je Quality, vlastnost s měřitelnou hodnotou Kind `Moon`:

```
:Mass rdf:type owl:Class;
      rdfs:subClassOf gufo:Quality.

:Moon rdf:type gufo:Kind.
:MoonsMass rdf:type :Mass;
```

```
           gufo:inheresIn :Moon
           gufo:hasQualityValue "7.34767309E22"^^xsd:double.
```

Součástí příkladu je i přiřazení hodnoty. Kromě přiřazení hodnoty lze také specifikovat jednotky, zároveň je možné uvést několik různých jednotek:

```
:massInKilograms rdf:type owl:DatatypeProperty;
                 rdfs:subPropertyOf gufo:hasQualityValue;
                 rdfs:domain :Mass;
                 rdfs:range xsd:double.

:MoonsMass rdf:type :Mass;
           :massOf :Moon;
           :massInKilograms "7.34767309E22"^^xsd:double;
```

Hodnota nemusí být jen číselná, může se jednat i o výčet. Vzhledem k definici Quality by jednotlivé možnosti měly být dány, a měly by být navzájem porovnatelné a seřaditelné. Takovým případem je například `ShirtSize`:

```
:ShirtSize rdf:type owl:Class;
        rdfs:subClassOf gufo:QualityValue;
        owl:equivalentClass [ rdf:type owl:Class;
                               owl:oneOf ( :S
                                           :M
                                           :L
                                           :XL
                                         )
                             ].
:S rdf:type :ShirtSize.
:M rdf:type :ShirtSize.
:L rdf:type :ShirtSize.
:XL rdf:type :ShirtSize.
```

`Headache` je Mode, vlastnost s neměřitelnou hodnotou Kind `Person`, `headacheOf` je Characterization vazba mezi Mode `Headache` a Kind `Person`:

```
:Headache rdf:type owl:Class;
        rdfs:subClassOf gufo:IntrinsicMode.

:headacheOf rdf:type owl:ObjectProperty;
            rdfs:subPropertyOf gufo:inheresIn;
            rdfs:domain :Headache;
            rdfs:range :Person.
```

Mode nemá narozdíl od Quality dané hodnoty. Některé Mode je ale možné rozšířit o další informace pomocí Quality – například Mode `Headache` by mohl mít Quality `HeadacheIntensity`.

`SchoolClass` je Collective skládající se z alespoň dvou Role `Student`, které mají v rámci celku stejnou funkci, jedná se tedy o vazbu MemberOf, zde na ukázce konkrétní instance:

```
:SchoolClass rdf:type gufo:Kind;
    rdfs:subClassOf gufo:VariableCollection;
:Student rdf:type gufo:Role.

:John rdf:type :Student;
    gufo:isCollectionMemberOf :SchoolClass.

:Person rdf:type gufo:Kind.
:Student rdfs:subClassOf :Person.
```

`JohnsBrain` je `Brain` který je součástí `John`, tedy vazba ComponentOf:

```
:John rdf:type :Person.

:Brain rdf:type owl:Class;
      rdfs:subClassOf gufo:Object.

:JohnsBrain rdf:type :Brain;
      gufo:isComponentOf :John.
```

Quantity `Alcohol` je součástí Quantity `Beer`, tedy vazba SubQuantityOf:

```
:Beer rdf:type owl:Class;
    rdfs:subClassOf gufo:Quantity.
:Alcohol rdf:type owl:Class;
    rdfs:subClassOf gufo:Quantity.
    rdfs:subClassOf [
  rdf:type owl:Restriction;
  owl:onProperty gufo:isSubQuantityOf;
  owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger;
  owl:onClass :Beer
].
:Beer rdfs:subClassOf [
  rdf:type owl:Restriction;
  owl:onProperty [ owl:inverseOf gufo:isSubQuantityOf ];
  owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger;
  owl:onClass :Alcohol
].
```

`LocalPopulation` je součástí Collective `Population`, tedy vazba SubCollectionOf:

```
:Population rdf:type gufo:Kind;
    rdfs:subClassOf gufo:VariableCollection;
    rdfs:subClassOf [
  rdf:type owl:Restriction;
  owl:onProperty :populationHasLocalPopulation;
  owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger;
].
:LocalPopulation rdf:type gufo:Kind;
    rdfs:subClassOf gufo:VariableCollection;
    rdfs:subClassOf [
  rdf:type owl:Restriction;
  owl:onProperty :populationHasLocalPopulation;
  owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger;
].
:populationHasLocalPopulation rdf:type owl:ObjectProperty;
    rdfs:subPropertyOf gufo:isSubCollectionOf;
    rdfs:domain :Population;
    rdfs:range :LocalPopulation.
```

Formal vazba `taller_than` je odvozená vazba od Kind `Person` sloužící k porovnání výšky, v gUFO nazývaná ComparativeRelationshipType:

```
:Person rdf:type owl:Class, gufo:Kind;
:height rdfs:domain :Person;
    rdfs:range xsd:int;
    rdf:type owl:DatatypeProperty;
    rdfs:subPropertyOf gufo:hasQualityValue.
:taller_than rdf:type owl:ObjectProperty;
    rdfs:domain :Person;
    rdfs:range :Person;
    rdf:type gufo:ComparativeRelationshipType.
```

Kind `Person` je buď SubKind `Man` nebo SubKind `Woman`, nemůže nastat jiná možnost, podtřídy jsou *complete*, také nazývané *covering*, vyčerpávající množina:

```
:PersonSex gufo:partitions :Person;

:Man rdf:type gufo:SubKind.
:Woman rdf:type gufo:SubKind.
:Man rdfs:subClassOf :Person.
:Woman rdfs:subClassOf :Person.
```

```
:PersonSex owl:equivalentClass [
   rdf:type owl:Class;
   owl:unionOf (:Man :Woman)
].
```

Kind `Animal` je druhem, `AnimalSpecies`, buďto `Hyena` nebo `Lion`, ale ne obou najednou, podtřídy jsou tedy *disjoint*, disjunktní:

```
:AnimalSpecies gufo:partitions :Animal.

:Hyena rdf:type :AnimalSpecies;
       rdfs:subClassOf :Animal.
:Lion rdf:type :AnimalSpecies;
      rdfs:subClassOf :Animal;
      owl:disjointWith :Hyena.
```

Pokud by podtřídy byly disjunktní i vyčerpávající množina, značilo by se tak pomocí `owl:disjointUnionOf`.

Kind `Product` může být v maximálně jedné objednávce Relator `Order`, ale nemusí být v žádné, tedy multiplicita `0..1`:

```
:Product rdf:type owl:Class, gufo:Kind;
:Order rdf:type owl:Class;
  rdfs:subClassOf gufo:Relator.

:Product rdfs:subClassOf [
  rdf:type owl:Restriction;
  owl:onProperty [ owl:inverseOf gufo:mediates ];
  owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger;
  owl:onClass :Order
].
```

Pokud je minimální kardinalita nula, není nutné ji zapisovat – pokud není řečeno jinak, předpokládá se minimální kardinalita nula. Podobně je to s maximální kardinalitou nekonečno, značené hvězdičkou. Pro stanovení přesné hodnoty slouží

```
owl:qualifiedCardinality
```

Kind `Person` má atribut `height`, s datovým typem integer:

```
:Person rdf:type owl:Class, gufo:Kind;
:height rdfs:domain :Person;
    rdfs:range xsd:int;
    rdf:type owl:DatatypeProperty;
    rdfs:subPropertyOf gufo:hasQualityValue.
```

Kapitola 3

Analýza dotazovacích a validačních jazyků

V této kapitole budou představeny jazyky zaměřené na práci s RDF. Jelikož RDF lze vyjádřit pomocí různých formátů, například jako strojově čitelný formát XML, je možné s ním dále pracovat. Lze využít jazyk Turtle pro kompaktnější zápis, který se může nadále využívat i v dalších jazycích pracujících s RDF. RDF má také svůj dotazovací jazyk podobný jazyku SQL, nazvaný SPARQL. Strukturu a hodnoty grafu lze kontrolovat pomocí jazyků na validaci dat, kterými jsou SHACL a ShEx. V těchto jazycích je možné vytvářet pravidla a omezení, proti kterým se daný RDF graf validuje.

3.1 Turtle

Turtle je jazyk pro RDF umožňující zápis v kompaktní a přehledné formě. Následující informace vychází z dokumentace Turtle [13]. Je podporován také zjednodušený zápis tripletů, a to i pro dotazovací jazyk SPARQL, který je více popsán v následující sekci.

Nejjednodušším zápisem pro triplet je sekvence subjektu, predikátu a objektu, které jsou odděleny mezerami. Celý triplet je zakončen tečkou. Často se stává, že stejný subjekt je popisován opakovaně, tedy několika různými predikáty. V Turtle proto lze využít středníku pro opakování subjektu pro triplety, kde se mění predikát a objekt, ale subjekt zůstává stejný. Středník se udává za tripletem místo tečky. Obdobně nastává situace, kdy je stejný subjekt popisován několikrát, stejným predikátem, a liší se pouze hodnota. Tato situace nastává zejména při zadávání slovních hodnot, jako je například název, s různými jazykovými tagy, tedy například v českém a anglickém jazyce. Pro opakování subjektu i predikátu najednou se využívá interpunkční čárky, uvedené za tripletem místo tečky, následované další hodnotou objektu. Turtle zároveň umožňuje zkrátit poměrně dlouhé IRI pomocí prefixů, což zlepšuje čitelnost a přehlednost kódu. Existují dvě varianty pro zápis prefixu.

První je originální syntax Turtle:

```
@prefix somePrefix: <http://example.org/datatype#>.
```

Druhá varianta je syntax jazyka SPARQL:

```
PREFIX somePrefix: <http://example.org/datatype#>
```

IRI tedy může být zapsána klasicky či pomocí prefixu:

```
<http://example.org/book/book1>
```

```
PREFIX book: <http://example.org/book/>
book:book1
```

3.2 SPARQL

Díky RDF a ontologiím je možné získat přesnou reprezentaci OntoUML modelu ve strojově čitelné a zpracovatelné podobě. Dalším krokem je tyto výstupy využít, a to pomocí jazyka SPARQL, což je dotazovací jazyk pro RDF, podobný jazyku SQL. SPARQL, stejně jako OWL, využívá IRI, rozšíření URI o další povolené znaky. Celá tato sekce, i s názornými příklady, vychází z dokumentace SPARQL [14]. Uvedené příklady jsou v datovém formátu Turtle.

Výsledky dotazu ve formě setu jsou zobrazeny v tabulkové podobě. Stejně jako RDF, SPARQL využívá triplety. Kterýkoliv prvek z trojice subjektu, predikátu a objektu může být proměnná. Každý výsledek je zobrazen jako jeden řádek v tabulce a požadavkům zadaným v dotazu může vyhovovat více

výsledků, nebo i žádný. Při dotazování ve SPARQL se zadáním podmínek udává, jaká část znalostního grafu se má extrahovat, tedy které podmínky musí uzly extrahovaných částí grafu splnit. Pomocí dotazů lze vytvořit tabulku s výsledky, nebo znalostní graf s výsledky. Jednoduchý dotaz má dvě části: `SELECT` klauzuli identifikující proměnné a `WHERE` klauzuli poskytující pravidla. Za výrazem `SELECT` je seznam proměnných, který určuje sloupce výsledné tabulky – každá proměnná vytvoří jeden sloupec. Jednotlivé řádky jsou pak tvořeny dosazenými hodnotami. [15]

Dotaz s podmínkami i výslednou tabulkou jsou ukázány na jednoduchém příkladu:

Vstupní data:

```
<http://example.org/book/book1>
  <http://purl.org/dc/elements/1.1/title> "SPARQL Tutorial".
```

Dotaz:

```
SELECT ?title
WHERE
{
  <http://example.org/book/book1>
      <http://purl.org/dc/elements/1.1/title> ?title.
}
```

Výsledek dotazu:

```
# title
"SPARQL Tutorial"
```

3.2.1 Syntax SPARQL

RDF literály jsou používány pro hodnotu, jako je řetězec, číslo, či datum. Literál může, ale nemusí, být následován datovým typem. Datový typ je značený pomocí IRI nebo prefixu. Příkladem literálu následovaným datovým typem je například

```
"a"^^xsd:string
```

Uvedená trojice písmen `xsd` je zkratka XML Schema Definition, doporučení dané mezinárodním konsorciem W3C, vyvíjejícím webové standardy pro World Wide Web. Tyto standardy popisují a validují strukturu a obsah XML dokumentu a používají se k definování elementů, atributů a datových typů které dokument může obsahovat. Mezi další datové typy podporované jazykem SPARQL patří například `xsd:integer`, `xsd:boolean` nebo `xsd:dateTime`.

Datový typ String musí být uzavřen ve dvojitých nebo jednoduchých uvozovkách. Pokud hodnota obsahuje uvozovky nebo znak nového řádku, celý

literál je uzavřen ve třech dvojitých či jednoduchých uvozovkách. Textový literál může také být následován jazykovým tagem, který specifikuje jazyk, ve kterém je hodnota zadána.

Ukázka literálu, literálu s jazykovým tagem, literálu s datovým typem, a literálu s hodnotou obsahující uvozovky:

```
"chat"
'chat'@fr
"xyz"^^<http://example.org/ns/userDatatype>
"abc"^^appNS:appDataType
'''The librarian said, "You would enjoy 'War and Peace'."'''
```

Čísla mohou být psána přímo, tedy bez uvozovek či explicitně uvedených datových typů. Pokud se jedná o celé číslo, je interpretováno jako literál datového typu `xsd:integer`. Pokud číslo obsahuje symbol tečky (ale neobsahuje exponent), je interpretováno jako desetinné číslo typu `xsd:decimal`. Čísla s exponentem jsou interpretována jako `xsd:double`. Hodnoty datového typu `xsd:boolean` mohou být zapsány i jako true nebo false.

V ukázce jsou na každém řádku uvedeny dva možné zápisy stejného literálu:

```
1, "1"^^xsd:integer
1.3, "1.3"^^xsd:decimal
1.0e6, "1.0e6"^^xsd:double
true, "true"^^xsd:boolean
```

Proměnné užívané v dotazech jsou v jazyce SPARQL globální, tedy pokud se v dotazu využívá název proměnné vícekrát, je identifikován jako stále jedna stejná proměnná. Proměnné mají prefix `?` nebo `$`. Prefix není součástí názvu, takže například `?abc` a `$abc` identifikují stejnou proměnnou.

V dotazu lze využít klíčové slovo `CONSTRUCT`, aby výsledkem nebyla tabulka, ale RDF graf.

3.2.2 Dotazování

Data pro následující dva příklady, obsahující literál s jazykovým tagem a celé číslo:

```
PREFIX dt:  <http://example.org/datatype#>
PREFIX ns:  <http://example.org/ns#>
PREFIX :    <http://example.org/ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

:x ns:p "cat"@en.
:y ns:p "42"^^xsd:integer.
```

Literály mohou být doplněny jazykovým tagem, poskytujícím informaci, že se jedná o informaci v daném jazyce, například angličtině. Jazykové tagy se přidávají pomocí znaku @.

Literál s jazykovým tagem se při porovnání s literálem stejné hodnoty ale bez jazykového tagu považuje za jiný literál:

```
SELECT ?v WHERE { ?v ?p "cat" }
```

Výsledek tohoto dotazu je prázdný, jelikož cat není stejný literál jako cat@en. Aby dotaz fungoval, je potřeba do vyhledávání zahrnout i jazykový tag:

```
SELECT ?v WHERE { ?v ?p "cat"@en }
```

```
# v
<http://example.org/ns#x>
```

Jak již bylo řečeno, je možné psát čísla přímo, bez datových typů. Číslo 42 je vlastně zkrácená podoba:

```
"42"^^<http://www.w3.org/2001/XMLSchema#integer>
```

Tuto jednoduchou formu lze využít i přímo v dotazu:

```
SELECT ?v WHERE { ?v ?p 42 }
```

Výsledek:

```
# v
<http://example.org/ns#y>
```

Součástí RDF grafu mohou být i prázdné uzly, a mohou se tedy vyskytovat i ve SPARQL. Prázdné uzly se zapisují v podobě _:, následované označením. Použití stejného označení indikuje stejný uzel.

Data:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
_:a foaf:name "Alice".
_:b foaf:name "Bob".
```

Dotaz:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?x ?name
WHERE { ?x foaf:name ?name }
```

Výsledek:

```
# x name
_:c "Alice"
_:d "Bob"
```

Označení prázdných uzlů může být odlišné, slouží jen k rozpoznání, zda se jedná o odlišné prázdné uzly.

Další možný výsledek:

```
# x name
_:r "Alice"
_:s "Bob"
```

Oba výsledky nesou stejnou informaci i přesto, že mají různá označení. Mezi označením prázdného uzlu v datovém grafu a ve výsledku nemusí být žádná souvislost. Pokud je potřeba vyhledat shody obsahující konkrétní řetězec, využívá se klíčové slovo `FILTER`. Vyhledávání hledá shodu pouze u literálů bez jazykových tagů. Zároveň je možné využít různé funkce na vyhledání shody dle regulárního výrazu.

V následujícím příkladu je navíc využit *flag*, který upravuje chování pravidla. Flag použitý v tomto příkladu značí, že se nemají rozlišovat velká a malá písmena.

Data:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX :   <http://example.org/book/>
PREFIX ns: <http://example.org/ns#>

:book1 dc:title "SPARQL Tutorial".
:book1 ns:price 42.
:book2 dc:title "The Semantic Web".
:book2 ns:price 23.
```

Dotaz:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { ?x dc:title ?title
        FILTER regex(?title, "web", "i" ) }
```

Výsledek:

```
# title
"The Semantic Web"
```

`FILTER` lze využít i pro čísla na aritmetické výrazy, a tím určit konkrétní rozmezí, které musí hodnota splňovat.

```
PREFIX  dc: <http://purl.org/dc/elements/1.1/>
PREFIX  ns: <http://example.org/ns#>
SELECT ?title ?price
WHERE { ?x ns:price ?price.
        FILTER (?price < 30.5)
        ?x dc:title ?title. }
```

Výsledek:

```
# title price
"The Semantic Web" 23
```

3.3 SHACL

SHACL, Shapes Constraint Language, je jazyk pro specifikaci omezení za účelem validace RDF dat. Validace probíhá proti setu podmínek zachycených v RDF grafu. Skupina podmínek se nazývá *tvar*, shape. Graf se skládá z několika tvarů a nazývá se proto *graf tvarů*, shapes graph. Graf, který je dle těchto podmínek validován, se nazývá *datový graf*, data graph. Jelikož se grafy tvarů používají k ověření skutečnosti, že datové grafy splňují daný set podmínek, mohou být považovány za popis datových grafů. SHACL je rozdělen na SHACL Core a SHACL-SPARQL. SHACL Core se skládá z často používaných funkcí na reprezentaci tvarů, omezení a cílů. Všechny SHACL implementace musí obsahovat alespoň SHACL Core. SHACL-SPARQL obsahuje pokročilé funkce, omezení dle jazyka SPARQL a možnost deklarovat nové komponenty omezení. [16]

3.3.1 Typy tvarů

IRI, literály a prázdné uzly se nazývají *RDF termy*. RDF term který je validován se nazývá *ohniskový uzel*, focus node. Jazyk SHACL Core definuje dva typy tvarů. Tvary o samotném ohniskovém uzlu, nazývané tvary uzlů, a tvary o hodnotě konkrétní vlastnosti či cesty pro uzel, nazývané tvary vlastností. `sh:Shape` je nadtřída obou těchto typů, a má tedy podtřídy `sh:NodeShape` a `sh:PropertyShape`. [16]

Tvar je definován jako IRI nebo prázdný uzel, který musí splnit alespoň jednu z následujících podmínek: [16]

- tvar je instancí tvaru uzlů nebo tvaru vlastností
- tvar je subjektem tripletu, který má jako predikát `sh:targetClass`, `sh:targetNode`, `sh:targetObjectsOf` nebo `sh:targetSubjectsOf`
- tvar je subjektem tripletu, který má jako predikát parametr

- tvar je hodnota parametru očekávajícího tvar, jako je například `sh:node`, nebo člen SHACL listu, který je hodnota parametru očekávajícího tvar, jako je například `sh:or`

Podsekce i s příklady vychází z knihy [17]. Následuje ukázka validace s využitím tvaru uzlů – graf tvarů, který určuje omezení, deklaruje, že uzly musí být IRI. Omezení se aplikuje na všechny uzly, které jsou instancí `:User`:

```
:UserShape a sh:NodeShape;
           sh:targetClass :User;
           sh:nodeKind sh:IRI.
```

Datový RDF graf:

```
:alice a :User.
<http://other.uri.com/bob> a :User.
_:1 a :User.
```

První i druhý uzel omezení splňují, ale třetí uzel ne, jelikož se jedná o prázdný uzel, který používá vlastní identifikátor, a ne IRI, tedy bude vrácena chyba s touto informací.

Ukázka validace s využitím tvaru vlastností – graf tvarů zde deklaruje, že uzly, které jsou instancí `:User`, musí mít hodnotu pro vlastnost `schema:knows` nebo `schema:follows` a tato vlastnost musí být IRI. Zároveň všechny uzly tranzitivně propojené přes `schema:knows` musí mít i vlastnost `schema:email`, jejíž hodnota musí být také IRI:

```
:UserShape a sh:NodeShape;
 sh:targetClass :User;
 sh:property [
   sh:path [sh:alternativePath (schema:knows schema:follows)];
   sh:nodeKind sh:IRI;
   sh:minCount 1
 ];
 sh:property [
   sh:path ([sh:oneOrMorePath schema:knows] schema:email);
   sh:nodeKind sh:IRI
 ].
```

Datový RDF graf:

```
:alice a :User;
       schema:follows <mailto:alice@mail.org>;
       schema:knows :bob, :carol.
:bob   schema:email <mailto:bob@mail.org>;
       schema:knows :carol.
```

```
:carol schema:email <mailto:carol@mail.org>.
:dave a :User;
       schema:knows <mailto:dave@mail.org>;
       schema:knows :carol, :emily.
:emily schema:email "Unknown".
```

Uzel `:alice` splňuje omezení – má vlastnost `schema:email` s IRI a všechny uzly, kterých lze dosáhnout přes `schema:knows`, mají také `schema:email` s IRI. Bude však vrácena chyba pro uzel `:dave`, jelikož jeden z uzlů dosažitelných přes `schema:knows` má `schema:email`, jehož hodnota není IRI, jedná se o uzel `:emily`.

3.3.2 Komponenty omezení

Tvary mohou určit omezení použitím *komponent omezení*. Ty jsou identifikovány pomocí IRI a jejich parametry se skládají z povinných a případných volitelných parametrů. Každá komponenta má alespoň jeden povinný parametr, z nichž každý je vlastností. Komponenta může mít navíc ještě volitelné parametry, každý z nich je opět vlastností. [16]

Podsekce vychází z knihy [17].

Parametry komponent jsou také identifikované pomocí IRI a mají hodnoty. SHACL Core obsahuje již vytvořené komponenty omezení. Většina z nich má pouze jeden parametr a dodržuje konvenci, že název parametru je součástí názvu komponenty omezení, a to podle vzoru, že z názvu parametru `sh:p` vyplývá název komponenty omezení `sh:pConstraintComponent`. Například komponenta `sh:MinCountConstraintComponent` deklaruje jediný parametr `sh:minCount`, reprezentující omezení, že uzel musí pro nějakou vlastnost splnit danou minimální hodnotu. Pokud komponenta používá jediný parametr, může být použit víckrát v rámci stejného tvaru. Každá hodnota určuje individuální omezení a všechna tato omezení se aplikují. [16]

Validátor zkontroluje všechna omezení a vrátí validační chybu pro každé omezení, které nebylo splněno. Výsledkem validátoru je validační zpráva, ve které jsou informace o chybách, pokud nastaly, nebo potvrzení že graf byl validován a žádná chyba nenastala.

Komponenty omezení umožňují vytvořit mnoho různých druhů omezení. Z těch základních lze jmenovat třeba omezení povolených datových typů uzlů nebo požadavek, aby uzly byly IRI. Je možné určit seznam hodnot a dané uzly musí nabývat právě některé z hodnot uvedené v seznamu. Dále existují pravidla i pro jazykové tagy, například aby se u uzlů neopakovaly tagy se stejným jazykem nebo pravidla pro omezení délky stringu. Pro číselné hodnoty lze specifikovat kardinalitu, tedy určení minimální a maximální možné hodnoty, lze deklarovat i konkrétní hodnotu, které musí uzly nabývat. To je možné použít i pro nečíselné hodnoty – lze hledat shodu nějakého řetězce nebo jeho části pomocí regulárních výrazů. Také je možné pravidla doplnit flagy,

které mohou upravit chování daného pravidla. Například je velmi používaný flag, který udává, že shoda nebude rozlišovat velká a malá písmena, jelikož ve výchozím chování jsou velká a malá písmena rozlišována. Mezi pokročilejší pravidla patří logická omezení, tedy spojky `sh:and`, `sh:or`, `sh:not` a `sh:xone`. `sh:xone` vyjadřuje pravidlo „přesně jeden". I když se logická spojka `sh:and` může jevit jako poněkud zbytečná, neboť se dle pravidel jazyka SHACL vždy aplikují všechna pravidla, může zlepšit čitelnost kódu. Tyto logické spojky je možné kombinovat a spojovat do složitějších výrazů a omezení a lze takto simulovat chování příkazu IF-THEN.

V některých případech je žádoucí validovat jen některé uzly datového grafu. K tomu slouží možnost deklarovat cílové uzly. Uzly označené jako cílové, tedy s predikátem `sh:targetNode`, budou validovány a musí tedy vyhovět nějakému tvaru. Zbytek uzlů není validován.

V následujícím příkladu jsou v grafu tvarů tři uzly deklarovány jako cílové, a to uzly `:alice`, `:bob` a `:carol`. Budou tedy validovány pouze tyto tři uzly:

```
:UserShape a sh:NodeShape;
   sh:targetNode :alice, :bob, :carol;
   sh:property [
    sh:path schema:name;
    sh:minCount 1;
    sh:maxCount 1;
    sh:datatype xsd:string;
  ].
```

Datový RDF graf:

```
:alice schema:name "Alice Cooper".
:bob foaf:name "Bob".
:carol schema:name 23.
:dave schema:name 45.
```

Uzel `:alice` podmínkám vyhoví, uzel `:bob` nevyhoví, jelikož nemá `schema:name`, a uzel `:carol` také nevyhoví, protože hodnota není typu string, jak bylo požadováno. Pro uzly :bob a `:carol` tedy bude vrácena chyba. Uzel `:dave` je při kontrole ignorován, neboť nebyl deklarován jako cílový.

3.4 ShEx

Jazyk ShEx, Shape Expressions, je jazyk pro popis struktur RDF grafu. S jazykem SHACL sdílí společný cíl, a to popis a validaci RDF dat. I způsob jakým se tohoto cíle snaží dosáhnout je velmi podobný, a díky tomu je pro většinu běžných případů možné mezi jazyky ShEx a SHACL převádět. [17]

Proto je tato sekce zaměřená převážně na vlastnosti, které mají tyto dva jazyky společné, a na vlastnosti, ve kterých se naopak liší.

Oba jazyky využívají tvary obsahující omezení. Tvary v SHACL jsou podobné tvarovým výrazům v ShEx s tím rozdílem, že odkazy na datové uzly jsou v SHACL vyjádřeny pomocí deklarací cílů a v ShEx pomocí map tvarů, shape maps. ShEx definuje několik druhů map tvarů, a to fixní, sloužící jako vstup pro validační proces, dotazovou, sloužící jako dotazovací mechanismus k vytvoření fixní mapy tvarů, a výsledkovou, zobrazující výsledek validace. Oba jazyky umožňují omezení specifikovaná povolenými datovými typy, povolenými hodnotami, či omezení pro vlastnosti na příchozích i odchozích koncích uzlu. V ShEx lze, stejně jako v jazyce SHACL, určovat kardinality a používat logické operátory, jen místo `xone` je využíván operátor `oneOf`, značený |. [17]

Mezi hlavní rozdíl patří jednoduchá rozšiřitelnost SHACL – uživatelé mohou díky SHACL-SPARQL vytvářet své vlastní komponenty omezení a sémantika takovýchto rozšíření je definována. ShEx není tak rozšiřitelný jako SHACL, jeho konstrukty jsou limitovány tím, co definuje komunitní skupina ShEx, takže v tomto ohledu je podobný SHACL Core. Další rozdíl je v samotné validaci. SHACL má široké možnosti pro rozsah validace, je možné zadat, aby se daný tvar aplikoval jen na určité IRI či jen na instance konkrétní třídy. S využitím funkcí SPARQL lze dosáhnout ještě pokročilejšího výběru validovaných částí. ShEx má limitovanou schopnost rozsahu validace. Součástí SHACL je zároveň slovník, definující jak by měl vypadat validační report a výsledek. ShEx takovýto slovník nemá, kromě informace, zda validovaný graf vyhověl nebo ne. [18]

V jazyce ShEx také není možné vyjádřit některá omezení, například aby kombinace predikátu a jazykového tagu byla unikátní nebo porovnání hodnot vlastností. Tedy nelze zadat omezení, že datum narození musí být menší než datum úmrtí. ShEx také neumožňuje vyjádřit pravidlo, udávající, že zdroj je tranzitivně členem třídy. V SHACL všechna tato pravidla vyjádřit lze, i když pro některé jejich pokročilejší varianty je potřeba využít SHACL-SPARQL. ShEx má navíc některé komponenty omezení, která nejsou součástí SHACL Core, ale lze je vyjádřit přes regulární výrazy nebo je definovat pomocí SHACL-SPARQL. [18]

3.5 RML

RML, RDF Mapping Language, je škálovatelný mapovací jazyk k vyjádření pravidel, která mapují data z heterogenních struktur do datového RDF modelu. RML je rozšířením mapovacího jazyka R2RML, který mapuje data z relačních databází do RDF. V RML je mapování dat založeno na jedné či více map tripletů, triples map, definujících jak budou triplety generovány. Mapa tripletů určuje pravidla pro generování RDF tripletů sdílejících stejný subjekt. Mapa tripletů se skládá z logického zdroje, mapy subjektů a nepovinných map predikátů a objektů. Logický zdroj je tvořen třemi částmi. Odkazem na vstupní zdroj, referenční formulací specifikující jak odkazovat na data, a iterátorem specifikujícím jak přes data iterovat. Mapa subjektů obsahuje vzor URI, který definuje, jak je generován subjekt každého tripletu a případně i jeho typ. Reference na vstupní data se provádí pomocí referenční formulace uvedené v logickém zdroji. Mapa predikátů a objektů generuje triplety – mapa predikátů specifikuje, jak je generován predikát, a mapa objektů specifikuje, jak je generován objekt. RML lze využít na vytvoření RDF ze souboru JSON, XML či TSV. [19]

Převod z JSON formátu do RDF tripletů bude ukázán na příkladu, který vychází z dokumentace RML [19]. V JSONu jsou zadána data dvou osob. Osoba má zadané ID, křestní jméno, příjmení, a barvu vlasů. Příjmení a barva vlasů nejsou povinné, druhá osoba proto nemá zadané příjmení.

```
{
 "persons": [
   {
     "id": "0",
     "firstname": "Anna",
     "lastname": "Smith",
     "hair": "black"
   },
   {
     "id": "1",
     "firstname": "David",
     "hair": "brown"
   }
 ]
}
```

Na převod jsou potřeba dva sety pravidel, pravidla popisující JSON soubor a pravidla definující jak se mají generovat RDF termy a poté i triplety. V pravidlech je potřeba definovat, že IRI reprezentující osobu je generována spojením `http://example.org/person/` a ID osoby. Tato IRI bude použita jako subjekt tripletu. Osoba bude vyjádřena přes vlastnost `schema:Person`, křestní jméno přes `schema:givenName`, příjmení přes `schema:lastName` a barva vlasů

přes `dbo:hairColor`. Pravidla jsou psána v Turtle, jelikož jsou sama v RDF formátu. Pro mapování je potřeba vložit několik prefixů, a to pro ontologii RML, ontologii R2RML, slovník dotazovacího jazyka, slovník RDF konceptů, slovník schema.org a ontologii DBpedia. Slovník schema.org a ontologie DBpedia se využijí na třídy a vlastnosti.

```
@prefix rml: <http://semweb.mmlab.be/ns/rml#>.
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix ql: <http://semweb.mmlab.be/ns/ql#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix : <http://example.org/rules/>.
@prefix schema: <http://schema.org/>.
@prefix dbo: <http://dbpedia.org/ontology/>.
```

Dále se přidají pravidla definující z jakého JSON souboru se budou získávat informace a jak se bude iterovat přes objekty:

```
:TriplesMap a rr:TriplesMap;
  rml:logicalSource [
    rml:source "persons.json";
    rml:referenceFormulation ql:JSONPath;
    rml:iterator "$.persons[*]"
  ].
```

`:TriplesMap a rr:TriplesMap;` definuje mapu tripletů sjednocující všechna pravidla pro osoby v souboru.

Prázdný uzel `:TriplesMap rml:logicalSource [ ... ]` je logický zdroj a obsahuje všechna pravidla pro JSON soubor.

`rml:source "persons.json"` udává, že se přistupuje k JSON souboru nazvaném persons.json.

`rml:referenceFormulation ql:JSONPath` udává, že k přístupu k datům v JSON souboru je využíván JSONPath.

`rml:iterator "$.persons[*]"` udává, že se iteruje přes všechny objekty vyhovující výrazu `$.persons[*]` daným přes JSONPath.

Následující pravidlo definuje, jak je generována IRI subjektu osoby:

```
:TriplesMap rr:subjectMap [
  rr:template "http://example.org/person/{id}"
].
```

`:TriplesMap rr:subjectMap [ ... ]` je mapa subjektů a obsahuje všechna pravidla o subjektu tripletu.

`rr:template "http://example.org/person/id"` udává, že subjekt je generován spojením `http://example.org/person/` s atributem id objektu osoba.

Dalším krokem je vyjádření každé osoby přes třídu `schema:Person`:

```
:TriplesMap rr:predicateObjectMap [
  rr:predicate rdf:type;
  rr:objectMap [ rr:constant schema:Person ];
].
```

`:TriplesMap rr:predicateObjectMap [ ... ]` je mapa predikátů a obsahuje všechna pravidla o predikátu tripletu.

`rr:predicate rdf:type` udává, že je využíván predikát `rdf:type`.

`rr:objectMap [ ... ]` je mapa objektů a obsahuje všechna pravidla o objektu tripletu. `rr:constant schema:Person` udává, že každá osoba splňuje, že objekt tripletu je `schema:Person`. Použitím všech výše zmíněných pravidel se vygenerují dva triplety, pro každou osobu jeden:

```
@prefix schema: <http://schema.org/>.

<http://example.org/person/0> a schema:Person.
<http://example.org/person/1> a schema:Person.
```

Dále je potřeba vyjádřit hodnoty ve `firstname` přes `schema:givenName`:

```
:TriplesMap rr:predicateObjectMap [
  rr:predicate schema:givenName;
  rr:objectMap [
    rml:reference "firstname"
  ]
].
```

Jelikož objekt není stejný pro všechny osoby, pravidla jsou zde odlišná, namísto `rr:constant` je používáno `rml:reference`. Obdobně se vytvoří pravidla pro příjmení a barvu vlasů:

```
:TriplesMap rr:predicateObjectMap [
  rr:predicate schema:lastName;
  rr:objectMap [
    rml:reference "lastname"
  ]
].
:TriplesMap rr:predicateObjectMap [
  rr:predicate dbo:hairColor;
  rr:objectMap [
    rml:reference "hair"
  ]
].
```

Použitím všech uvedených pravidel se již získá finální výstup se všemi informacemi z JSON souboru převedenými do RDF:

```
@prefix dbo: <http://dbpedia.org/ontology/>.
@prefix schema: <http://schema.org/>.

<http://example.org/character/0> a schema:Person;
  dbo:hairColor "black";
  schema:givenName "Anna";
  schema:lastName "Smith".

<http://example.org/character/1> a schema:Person;
  dbo:hairColor "brown";
  schema:givenName "David".
```

Kapitola 4

Měření kvality OntoUML modelů

Tato kapitola se zabývá měřením kvality OntoUML modelů dle různých kritérií. První sekce se věnuje teorii a studiím zabývajícími se měřením kvality. Dle článku [20], pravidla měření kvality nejsou jasně definovaná, jedná se o stále se vyvíjející koncept, a často tak není možné je plně využít. Některé z kritérií používaných pro měření kvality jsou objektivní, ale jiné mohou být subjektivní. Jejich vnímání je také ovlivněno zkušenostmi a sociokulturními faktory. [2]

Použitá kritéria tak budou vycházet ze studií a z vlastních zkušeností.

V dalších dvou sekcích budou představeny čtyři modely vytvořené za účelem dotazování, kontroly chyb a měření kvality. První model byl vytvořen tak, aby obsahoval velké množství různých stereotypů. Druhý a třetí model jsou již vytvořeny dle existující domény, jsou inspirovány existujícími datovými sadami. Čtvrtý model vznikl spojením druhého a třetího.

Další dvě sekce se již budou věnovat využití jazyků SPARQL a SHACL pro dotazování, kontrolu, a měření kvality. Pomocí jazyka SPARQL se dá nad modelem dotazovat a vyhledávat v něm, stejně tak nad daty. Obzvlášť při využití u větších modelů či spojování modelů je jednodušší a spolehlivější se dotazovat pomocí SPARQL, než informace vyhledávat ručně. Validační jazyk SHACL je určen ke kontrole a detekci chyb a lze ho také využít na detekci anti-patternů.

Pro práci s modely a jejich následný export do gUFO jsou využívány dva nástroje, a to OpenPonk a Visual Paradigm s OntoUML pluginem. OpenPonk byl použit na vytvoření OntoUML modelu, tedy na jeho vymyšlení a přehledné rozvrhnutí jednotlivých částí. Obrázky modelů v této práci jsou z nástroje OpenPonk. Nástroj Visual Paradigm s pluginem pro OntoUML byl také použit na vytvoření OntoUML modelu za účelem exportu modelu do gUFO, v exportu ale byly prováděny značné úpravy. Oba nástroje jsou detailně popsány a porovnány v další kapitole v sekci 5.1.

4.1 Metriky kvality

Cílem modelování je plné a přesné pochopení domény reálného světa, pro které je nutná vysoká kvalita modelu. I přes její důležitost ale není jasně definovaná a jedná se o rychle se vyvíjející koncept. Metrikami pro kvalitu modelu tak mohou být vlastnosti, které se nacházejí v modelech považovaných za vhodné, a měly by tedy být přítomny i v ostatních modelech. Pokud jsou dány nějaké definice, bývají komplikované a přesto nejasné. Existuje mnoho frameworků, které se do této problematiky snaží vnést řád, a zaměřují se jak na kvalitu modelů, tak na kvalitu modelovacího procesu. Zřejmě nejznámějšími a nejúspěšnějšími jsou *LSS* a *BWW*. LSS je framework autorů Lindland, Sindre, a Sølvberg. BWW je framework autorů Wand a Weber založený na ontologii autora Bunge. I když mají oba frameworky pevný teoretický základ, každý se na problematiku dívá z jiného pohledu. LSS se zaměřuje na produkt konceptuálního modelování, tedy model, zatímco BWW se zaměřuje na proces samotného modelování. Pomocí obou byly provedeny významné výzkumy, ale jejich výsledky jsou fragmentované a obtížně se integrují. Frameworky úspěšně popsaly vztahy mezi elementy dle svého zaměření, ale v každém nějaké elementy a vztahy chybí, a proto je pochopení obtížné a nekompletní. [20]

Na základě těchto frameworků tak vznikl *CMQF*, Conceptual Modeling Quality Framework. Ten využívá síly každého ze dvou výše uvedených, kombinuje je, a definuje dimenze vycházející z jejich kvalit tak, aby se propojily. Mnoho studií se zaměřuje na kvalitu konceptuálního modelu, ale jen málo studií se věnuje na modelářově znalosti domény a reprezentace. CMQF díky spojení nehodnotí pouze výsledek modelovacího procesu, ale i modelovací proces samotný. Jedná se zatím o pouze teoretický framework, může ale sloužit jako teoretický základ pro budoucí výzkum kvality konceptuálního modelování. [20]

4.1.1 Vnímaná a měřená kvalita

Zaměřením na kvalitu samotného modelu se zabývá studie [2]. Soustředí se na experiment orientovaný na porovnání vnímané a měřené kvality ER modelů (Entity-Relationship Model). Podsekce vychází z této studie.

Kvalitu konceptuálního modelu lze definovat jak set vnímaných charakteristik. Některé z těchto charakteristik jsou objektivní, ale jiné mohou být subjektivní. Vnímání uživatelů a účastníků (stakeholders) informačního systému je ovlivněno jejich zkušenostmi a sociokulturními faktory. Jelikož neexistuje jediné kritérium kvality sjednocující všechny charakteristiky, nelze jednoduše vytvořit takový pohled na doménu, který bude stejně vhodný a kvalitní pro všechny.

Účastníci při hodnocení kvality modelu spoléhají na své vlastní vnímání reality a zkušenosti. Proto je potřeba mít několik kritérií pro měření kvality, v tomto experimentu byly definovány čtyři:

- Minimalita – měří, zda se v modelu nevyskytují redundantní koncepty.
- Jasnost – měří, jak jednoduše lze schéma číst. Založená na heuristice, že schéma obsahující N hran může mít nejvýše N křížení hran. Ve skutečnosti to ale bude mnohem méně, neboť modeláři usilují o přehledné rozložení modelu.
- Jednoduchost – schéma je jednoduché, pokud je postaveno na jednoduchých konceptech a dá se mu tak rychle porozumět. Měření jednoduchosti je založeno na předpokladu, že komplexita modelu roste s počtem vztahů, včetně dědičnosti a agregace.
- Expresivita – měří bohatost schématu, zda jsou koncepty dostatečně expresivní pro zachycení aspektů reality.

Experimentu se účastnilo celkem 113 osob, 87 odborníků na informační systémy a 26 koncových uživatelů. Skupina odborníků se skládala ze specialistů na různé domény. Mezi všemi účastníky byly osoby se zkušenostmi i bez nich, muži i ženy, a zástupci různých věkových kategorií. Skupina tak byla velmi různorodá.

Každý účastník experimentu obdržel osm ER modelů (stejné pro všechny účastníky) a měl je ohodnotit na základě zmíněných čtyř kritérií na stupnici od jedné do osmi. Tím vznikla vnímaná kvalita, dle této stupnice byly modely ohodnoceny i měřením. Poté se spočítal rozdíl mezi naměřenou a vnímanou kvalitou. Měřilo se tedy osm modelů, dle čtyř kritérií, na stupnici od jedné do osmi. Vzhledem k velikosti stupnice se očekávala odchylka průměrně 3.5 pro jedno hodnocení. Pro všech 32 hodnocení by se tedy jednalo o celkovou odchylku 112. Skutečná odchylka ale byla pouze 40, což nasvědčuje tomu, že účastníci experimentu vnímali kvalitu modelů podobně, jako byla naměřena přes zadefinovaná kritéria pro měření kvality. Pro dva nejextrémnější modely, tedy nejlepší a nejhorší pro každé kritérium, byla odchylka pouze 4 namísto očekávaných 28.

Lze tedy stanovit závěr, že hodnocení kvality dle definovaných kritérií je relevantní a není zde žádný významný rozdíl oproti vnímané kvalitě.

4.1.2 Měření kvality spojených OntoUML modelů

Nejsou tedy žádné standardizované metriky kvality, které by model měl splňovat. Při měření kvality se tak využijí čtyři kritéria zmíněná ve studii výše, a vlastní kritéria. Těmi může být mimo jiné počet chyb a anti-patternů v modelu. Chyby a anti-patterny v modelu sice některé modelovací nástroje dokáží

detekovat, pokud se ale pomocí technologií RDF modely převedou do textového formátu, tato detekce fungovat nebude. Řešením je samozřejmě kontrola modelů před samotným exportem. Pokud jsou ale technologie využívány ve větší míře a s jejich pomocí se spojuje více modelů dohromady, výsledky již mohou být odlišné. Spojením více modelů mohou některé chyby a anti-patterny vzniknout, ale jiné se tím naopak mohou vyřešit.

Příkladem vzniku chyby může být například dědění identity – pokud by v každém z modelů byl poskytovatel identity nazván jinak, vznikly by tak dvě rozdílné entity, od kterých by podtyp dědil, což je v rozporu s pravidly OntoUML. Přitom by každý z modelů zvlášť byl správně. Případ, kdy by se sloučením modelů chyba vyřešila, je například pro chybějící pečetidlo Role. Pokud se jeden model zabývá doménou dané Role pouze okrajově a domodelování potřebných vztahů by model zbytečně rozšířilo a komplikovalo, může být učiněno rozhodnutí, že se dané vztahy nebudou modelovat. Spojením s modelem, který se zabývá danou doménou, a je tam tedy tato Role i se všemi vztahy, bude chyba vyřešena. Tohoto se dá aktivně využívat pro zjednodušení modelů, pokud se již dopředu plánuje spojení několika modelů.

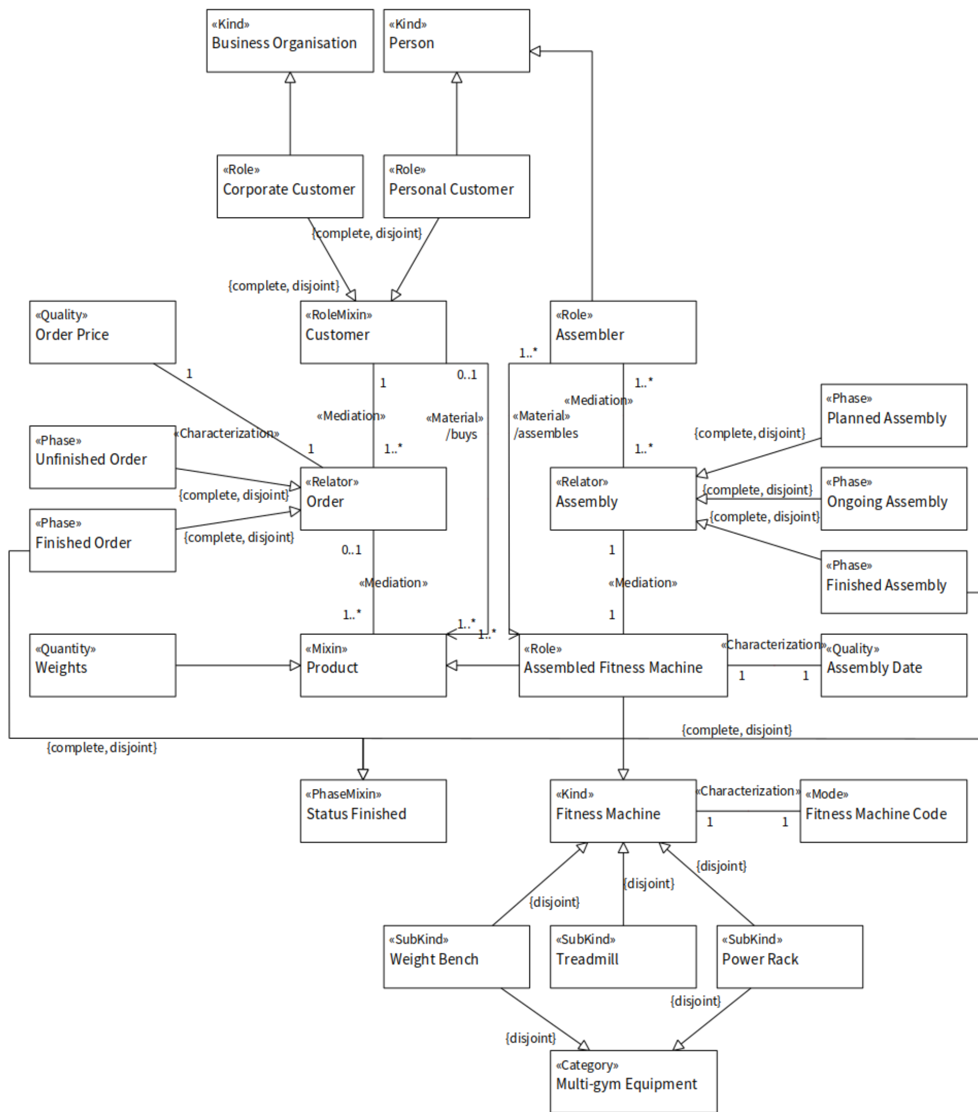
I přesto, že model bude dle různých metrik kvalit správně a nebude obsahovat chyby ani anti-patterny, bohužel nezaručuje, že je model opravdu vhodně namodelován. Mohou být například využity špatné stereotypy, nebo modelovaná doména nebude odpovídat realitě. Na posouzení těchto faktorů je obvykle zapotřebí zkušený modelář a doménový expert, ideálně více osob, kvůli různým pohledům na danou doménu. Stále jsou ale jakékoliv automatizované kontroly kvality velmi nápomocné a vhodné pro minimalizaci chyb a maximalizaci čitelnosti modelu.

4.2 Ukázkový model

Prvním modelem kde budou technologie využity, bude jednoduchý diagram zabývající se výrobou a prodejem produktů určených pro posilovny. Tento model nepopisuje složitou doménu, která by byla potřeba objasnit, ale obsahuje velké množství různých stereotypů a lze tak na něm spoustu věcí ukázat a vyzkoušet. Model se bude dále nazývat Posilovna a je přiložen na obrázku 4.1.

Popis domény modelu: Zákazníkem, který vytváří objednávku, může být osoba nebo společnost. U objednávky je uvedena cena a sleduje se její stav, tedy zda je objednávka již vyřízena nebo ne. Objednat lze závaží a posilovací stroje. Společnost prodává pouze již sestavené posilovací stroje, zákazník tak obdrží již smontovaný stroj připravený k použití. Montáže strojů provádí montéři, průběh montáže každého stroje je také sledován. Samotné stroje mají své kódové označení a mohou se dělit do kategorií.

Model Posilovna reprezentovaný pomocí technologií RDF je poměrně rozsáhlý a plná délka je téměř 200 řádků. Z toho důvodu jsou zde zobrazeny pouze části kódu, a kompletní kód je v příloze.



Obrázek 4.1: OntoUML model popisující výrobu a prodej produktů určených pro posilovny, využívající téměř všechny stereotypy OntoUML.

Jednotlivé části kódu lze rozdělit do několika skupin. Nejdříve je část obsahující deklaraci prefixů:

```
@prefix : <http://example.com#>.
@prefix gufo: <http://purl.org/nemo/gufo#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
```

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
```

Poté je potřeba definovat jednotlivé entity a jejich stereotypy:

```
:Person rdf:type gufo:Kind.
:PersonalCustomer rdf:type gufo:Role.
:BusinessOrganisation rdf:type gufo:Kind.
:CorporateCustomer rdf:type gufo:Role.
:Customer rdf:type gufo:RoleMixin.
:Assembler rdf:type gufo:Role.
:PlannedAssembly rdf:type gufo:Phase.
:Product rdf:type gufo:Mixin.
```

S tím souvisí definování podtypů a nadtypů, tedy všech generalizací obsažených v modelu:

```
:CorporateCustomer rdfs:subClassOf :BusinessOrganisation.
:PersonalCustomer rdfs:subClassOf :Person.
:CorporateCustomer rdfs:subClassOf :Customer.
:PersonalCustomer rdfs:subClassOf :Customer.
:Assembler rdfs:subClassOf :Person.
:PlannedAssembly rdfs:subClassOf :Assembly.
:OngoingAssembly rdfs:subClassOf :Assembly.
:FinishedAssembly rdfs:subClassOf :Assembly.
:AssembledFitnessMachine rdfs:subClassOf :Product.
```

Při využití nadtypů a podtypů je obvykle zapotřebí množiny blíže specifikovat, tedy zda jsou v modelu uvedeny všechny možné existující podtypy a zda je možné být instancí několika entit z množiny, nebo jen jedné. V kódu jsou tak uvedeny i disjoint a complete tam kde je potřeba:

```
:Order owl:equivalentClass [
  rdf:type owl:Class;
  owl:unionOf (:FinishedOrder :UnfinishedOrder)
].
[ rdf:type owl:AllDisjointClasses ]
    owl:members (:Treadmill :PowerRack :WeightBench).
[ rdf:type owl:AllDisjointClasses ]
    owl:members (:WeightBench :PowerRack).
```

Následují informace o popsaných vazbách, v tomto případě vazby Material:

```
:assembles rdf:type owl:ObjectProperty;
    rdfs:domain :Assembler;
    rdfs:range :AssembledFitnessMachine;
    rdf:type gufo:MaterialRelationshipType.
:buys rdf:type owl:ObjectProperty;
```

```
    rdfs:domain :Customer;
    rdfs:range :Product;
    rdf:type gufo:MaterialRelationshipType.
```

V modelu se vyskytují multiplicity, které je také potřeba transformovat:

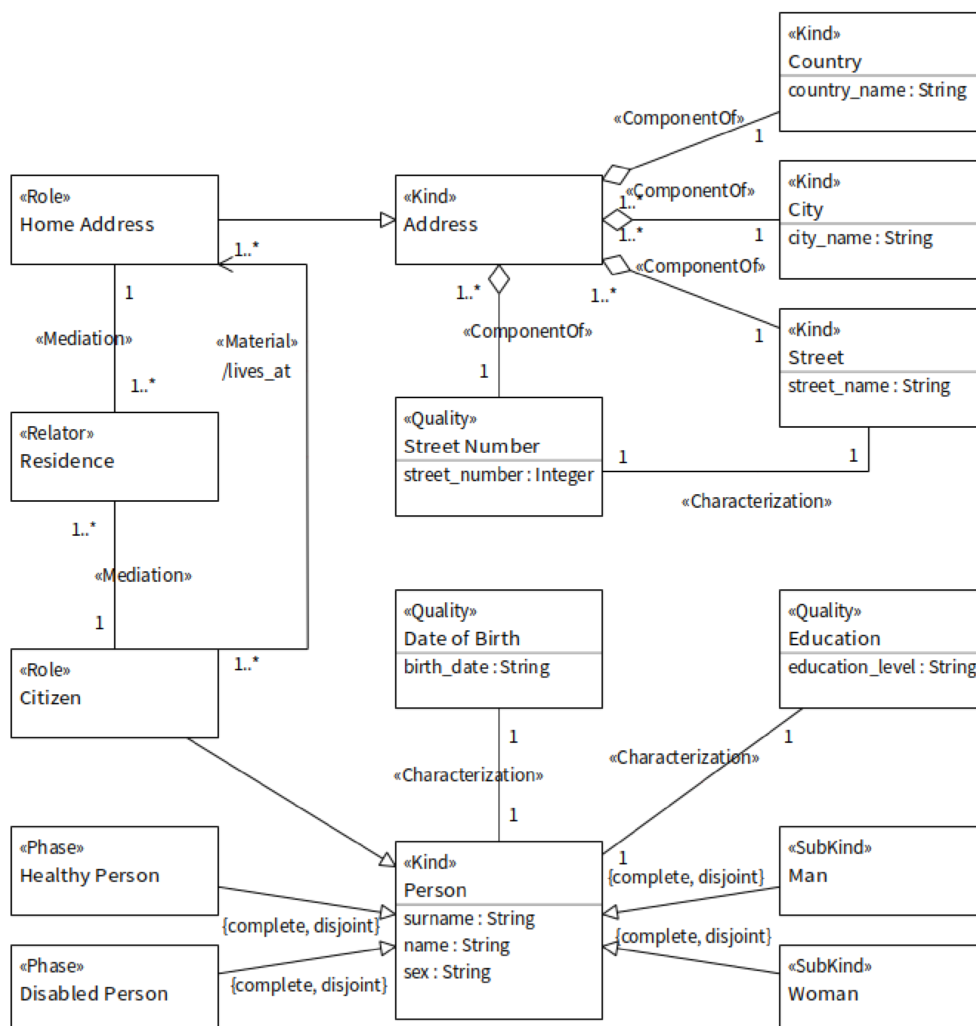
```
:AssemblyDate rdfs:subClassOf [
  rdf:type owl:Restriction;
  owl:onProperty [ owl:inverseOf gufo:inheresIn ];
  owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger;
  owl:onClass :AssembledFitnessMachine
].
:AssembledFitnessMachine rdfs:subClassOf [
  rdf:type owl:Restriction;
  owl:onProperty gufo:inheresIn;
  owl:minQualifiedCardinality "1"^^xsd:nonNegativeInteger;
  owl:onClass :AssemblyDate
```

4.3 Propojení modelů

V této sekci bude vše ukázáno na modelech vycházejících více z praxe. Modely reprezentují dvě podobné datové sady týkající se informací o invalidních osobách. Byly zvoleny tak, aby se jednalo o sady malého rozsahu, aby modely nebyly příliš obsáhlé. Datové sady se týkají stejné domény, ale s odlišným zaměřením, každá datové sada také pochází z jiného zdroje. Doména invalidity byla zvolena z praktických důvodů. Těmi bylo využití něčeho, co se opravdu sleduje a měří, na doméně, která bude snadno představitelná a pochopitelná. Tedy nebude příliš abstraktní a nebude obsahovat odborné pojmy. V kombinaci s cílem najít dvě podobné menší sady z různých zdrojů byly zvoleny právě následující datové sady.

Modely nejsou přesnou reprezentací datových sad, ale byly upraveny pro potřeby práce. V běžně využívaných datových sadách v praxi jsou data anonymizována a agregována, a nebylo by tedy možné všechny používané technologie zcela využít vzhledem k rozsahu práce. V práci tedy nejsou využita skutečná data sad, všechna data instancí v modelu jsou vytvořena pro účely práce a nejsou vytvářena podle skutečných osob.

Prvním modelem je Přehled invalidity, viz obrázek 4.2, zabývající se obecnými informacemi o invalidních osobách. Těmi jsou rok narození, pohlaví, vzdělání a adresa pobytu, skládající se ze státu, města, ulice a čísla orientačního. Model vychází z datové sady Eurostatu nazvané Population by sex, age, disability status and educational attainment level. [21]

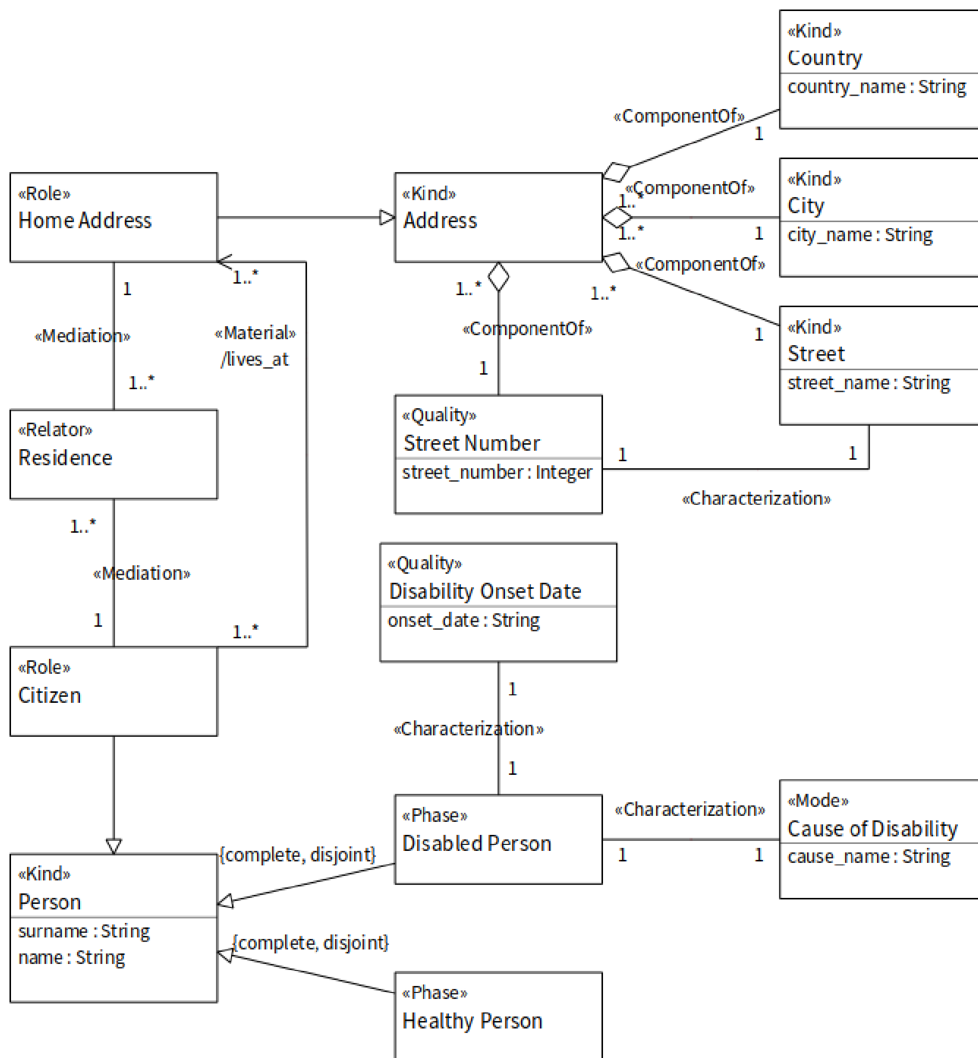


Obrázek 4.2: Model Přehled invalidity obsahující obecné informace o invalidních osobách.

Druhý model se nazývá Příčiny invalidity, viz obrázek 4.3. Stejně jako předchozí model obsahuje informace o invalidních osobách a o jejich adrese pobytu, skládající se ze státu, města, ulice a čísla orientačního. Již se ale nesoustředí na demografické informace, jako jsou rok narození, pohlaví a vzdělání, ale obsahuje informaci o příčinách invalidity daných osob, společně s datem vzniku. Model vychází z datové sady České správy sociálního zabezpečení nazvané Nejčastější příčiny vzniku invalidity. [22]

Propojit více modelů může být různě obtížné, v závislosti na velikosti modelů a použitém modelovacím nástroji. I pokud se ale jedná o malé a srozumitelné modely, pokud jich bude více spojeno dohromady, stanou se ne-

přehlednými a modelář se zřejmě nevyhne křížení čar. Při reprezentaci přes technologie RDF je spojení modelů velmi jednoduché a nenese žádné další nevýhody jako nižší přehlednost, jelikož se s daty pracuje pomocí dotazovacích a validačních jazyků. Modely Přehled invalidity a Příčiny invalidity tak byly propojeny přes technologie RDF do jednoho, dále nazývaným model Invalidity, nad kterými budou dále dělány dotazy a validace. Všechny exporty do gUFO jsou v příloze práce.



Obrázek 4.3: Model Příčiny invalidity zabývající se vznikem invalidity u jednotlivých osob.

4.4 Validace v SHACL

Pro modely vyjádřené pomocí technologií RDF lze využít mnoho jazyků určených právě pro práci s RDF. Některé z nich již byly popsány v předchozí kapitole, jedná se zejména o RML, SHACL a SPARQL. RML lze využít pro převod dat do podoby RDF z jiného formátu. Představuje tak jednoduchý způsob, jak pomoci s převodem modelů do RDF, jelikož model samotný lze exportovat do gUFO přes nástroj, například Visual Paradigm, a data lze převést právě pomocí RML. Vše potřebné je tak připraveno a sjednoceno do podoby RDF a může se využít SHACL a SPARQL.

Jazyk SHACL je vhodný pro validaci dat, tedy jestli jsou v požadovaném formátu, datovém typu, a zda jsou kompletní. Pomocí jazyka SPARQL se lze nad modelem i nad jeho daty dotazovat a vyhledávat v nich. Obzvlášť u větších či spojených modelů je jednodušší a spolehlivější se dotazovat takto, než informace vyhledávat ručně. SPARQL je také vhodný pro detekování chyb a anti-patternů v modelu. I přesto, že se jedná o dotazovací jazyk a SHACL je validační jazyk, je lepší pro detekci chyb a anti-patternů v modelu využít SPARQL, ve kterém jsou dotazy oproti SHACL jednodušší.

Při validování dat lze kontrolovat, že jsou vyplněny všechny povinné údaje, a že jsou vyplněny správně, například, že souhlasí datový typ. Pro model Invalidity byla vytvořena data (nevycházející ze skutečných osob), která jsou instancemi OntoUML modelu v gUFO, a lze se nad nimi dotazovat a vyhledávat v nich. V datech je i instance, která má jeden údaj chybný a dva údaje chybí. Data jsou v příloze.

Pro validace se používají následující prefixy. Ty jsou z důvodu přehlednosti práce uvedeny pouze zde, a ne u každé validace:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix schema: <http://schema.org/>.
@prefix sh: <http://www.w3.org/ns/shacl#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
```

První validací je kontrola, že všechny čísla ulic, `street_number`, jsou celá čísla, tedy datového typu `xsd:integer`:

```
:UserShape a sh:NodeShape;
  sh:targetClass schema:Person;
  sh:property [
    sh:path schema:street_number;
    sh:datatype xsd:integer;
  ].
```

Výsledkem je validační report oznamující chybu u záznamu osoby číslo 3:

```
[
  a sh:ValidationResult;
  sh:resultSeverity sh:Violation;
  sh:sourceConstraintComponent sh:DatatypeConstraintComponent;
  sh:sourceShape _:n1567;
  sh:focusNode <http://example.org/character/3>;
  sh:value "three";
  sh:resultPath schema:street_number;
  sh:resultMessage "Value does not have datatype xsd:integer";
].
```

Pomocí jednoho validačního dotazu lze kontrolovat i několik atributů najednou. Podmínky pro jednotlivé atributy se jednoduše zapíšou za sebe, není potřeba žádná další úprava, jako například přidání klíčových slov či využití závorek.

V následující validaci je kontrolováno, že jméno osoby, `name`, je datového typu `xsd:string`, atribut pohlaví, `sex`, nabývá hodnot `Man` nebo `Woman` a datum narození, `birth_date`, je typu `xsd:date`:

```
:UserShape a sh:NodeShape;
  sh:targetClass schema:Person;
  sh:property [
    sh:path schema:name;
    sh:datatype xsd:string;
  ];
  sh:property [
    sh:path schema:sex;
    sh:or ( [sh:in ("Man" "Woman")] );
  ];
  sh:property [
    sh:path schema:birth_date;
    sh:datatype xsd:date;
  ].
```

Výsledkem této validace není report, což znamená, že validace proběhla v pořádku a nebyla nalezena žádná chyba.

Další možnou validací je kontrola, že pro každou osobu jsou v datech zadány potřebné údaje. Může být vyžadováno, aby byly zadány všechny údaje, či pouze údaje pro atributy, které jsou označeny jako povinné. V následujícím příkladu se kontroluje, zda má každá osoba uvedeno datum vzniku invalidity, `onset_date`, a příčinu invalidity, `cause_name`:

```
:UserShape a sh:NodeShape;
  sh:targetClass schema:Person;
  sh:property [
    sh:path schema:onset_date;
    sh:minCount 1;
  ];
  sh:property [
    sh:path schema:cause_name;
    sh:minCount 1;
  ].
```

Pro jednu osobu, značenou jako číslo 3, chybí oba tyto atributy a je to tedy popsáno ve validačním reportu:

```
[
  a sh:ValidationResult;
  sh:resultSeverity sh:Violation;
  sh:sourceConstraintComponent sh:MinCountConstraintComponent;
  sh:sourceShape _:n6221;
  sh:focusNode <http://example.org/character/3>;
  sh:resultPath schema:onset_date;
  sh:resultMessage "Less than 1 values";
].
[
  a sh:ValidationResult;
  sh:resultSeverity sh:Violation;
  sh:sourceConstraintComponent sh:MinCountConstraintComponent;
  sh:sourceShape _:n6222;
  sh:focusNode <http://example.org/character/3>;
  sh:resultPath schema:cause_name;
  sh:resultMessage "Less than 1 values";
].
```

SHACL není příliš vhodný na kontrolu chyb a anti-patternů v modelu. Kontrola je možná, ale zápis je obtížnější a delší než je v dotazovacím jazyce SPARQL. Pro tento druh kontroly modelů je zapotřebí přidat další prefixy:

```
@prefix gufo: <http://purl.org/nemo/gufo#> .
@prefix : <http://example.com/myshapes#> .
```

Pro ukázku je přiložena část validačního dotazu pro detekci anti-patternu FreeRole. Tento anti-pattern vzniká, pokud existuje Role bez Mediation vazby, a tato Role je podtypem jiné Role která má Mediation vazbu. Tato validace kontroluje pouze přítomnost Mediation vazeb u stereotypů Role, ale již nekontroluje případný nadtyp a jeho možnou Mediation vazbu:

```
:FreeRoleAntiPattern
  a sh:NodeShape;
  sh:targetClass gufo:Role;
  sh:or (
    [
      sh:name "FreeRole Anti-Pattern (via domain)";
      sh:path ( [ sh:inversePath rdfs:domain ]
                rdfs:subPropertyOf );
      sh:value gufo:mediates;
      sh:minCount 1;
    ]
    [
      sh:name "FreeRole Anti-Pattern (via range)";
      sh:path ( [ sh:inversePath rdfs:range ]
                rdfs:subPropertyOf );
      sh:value gufo:mediates;
      sh:minCount 1;
    ]
  ).
```

Vyjádření nadtypu a podtypu je v jazyce SHACL obtížné, a plná validace FreeRole by tak byla ještě mnohem složitější. Většina detekce chyb i anti-patternů OntoUML se ale bez vztahu nadtypu a podtypu neobejde. SHACL tak není vhodným jazykem pro tento druh kontroly. Vyjádření nadtypu a podtypu ve SPARQL je ale naopak velmi jednoduché a vytváření dotazů pro kontrolu je velmi intuitivní.

4.5 Dotazování ve SPARQL

Po převodu modelů do RDF pomocí gUFO, což může být učiněno například přes nástroj Visual Paradigm, se mohou využít jazyky SHACL a SPARQL. SHACL je vhodný pro validaci dat, tedy jestli jsou v požadovaném formátu, datovém typu, a zda jsou kompletní. Není ale příliš vhodný na detekci chyb a anti-patternů v modelu. Pomocí jazyka SPARQL se lze nad modelem i nad jeho daty dotazovat a vyhledávat v nich. Obzvlášť u větších či spojených modelů je dotazování pomocí SPARQL vhodnější, než informace vyhledávat ručně. SPARQL je ale také vhodný pro kontrolu modelu samotného, tedy detekování chyb a anti-patternů. I přesto, že SPARQL je dotazovací jazyk a SHACL validační, je vhodnější využít SPARQL, ve kterém jsou dotazy oproti SHACL jednodušší.

V průběhu vytváření práce a seznamování se s doménou technologií RDF bylo nalezeno mnoho nedostatků, jak v exportu do gUFO v nástroji Visual Paradigm, tak i v samotné dokumentaci gUFO. Nástroj mnoho informací exportuje chybně či je zcela vynechává a v dokumentaci chybí některé důležité

informace. Zcela chybí dva stereotypy a dokumentace není aktuální dle ustálené verze OntoUML 2.0. Aby bylo do budoucna možné všechny koncepty OntoUML lépe využít, byl v práci navrhnut export do gUFO, ve kterém jsou popsány všechny nedostatky a chyby a přidány chybějící stereotypy. V návrhu, který se nachází v následující kapitole v sekci 5.2, jsou popsány všechny nalezené problémy a jsou pro ně navrhnuta řešení. Jelikož v současné podobě gUFO je mnoho důležitých vztahů stále nedefinovaných či definovaných dle staré verze, tak díky aktualizovanému návrhu bude možné technologie RDF lépe využít. Pro dotazování, kontrolu, měření kvality a další práci s modely je použita současná verze gUFO exportu, pouze byly opraveny chyby vzniklé nesprávným exportem ve Visual Paradigm.

Rozdíly v jednotlivých modelovacích jazycích byly popsány již v první kapitole, kde byly popsány i výhody a přínosy OntoUML. Reprezentace OntoUML modelů v RDF má ale více výhod oproti například UML či ER modelům. Je tak z toho důvodu, že OntoUML má mnohem více pravidel než mají UML a ER modely, a pravidla jsou navíc mnohem složitější. Proto je velmi výhodné využít dotazovací a validační jazyky právě na kontrolu. Kvůli dodatečným informacím obsažených ve stereotypech je i měření kvality odlišné. Pravidla, která platí pro kritéria kvality například u ER modelů, mohou být pro OntoUML modely poupravené.

Dotazování a měření bude prováděno na všech představených modelech, včetně spojeného a ukázkového, tedy Přehled invalidity, Příčiny invalidity, Invalidita a Posilovna.

Pro všechny dotazy jsou používány následující prefixy, které z důvodu přehlednosti nebudou v práci vkládány v rámci každého dotazu:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX gufo: <http://purl.org/nemo/gufo#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
```

4.5.1 Měření kvality dle kritérií studie

Kvalitu modelu lze měřit dle již zmíněné studie [2], zabývající se čtyřmi kritérii pro vnímanou a měřenou kvalitu. Přesný postup měření je ale uveden jen pro jedno kritérium ze čtyř. Ostatní tedy budou posuzována dle vlastních pravidel navržených tak, aby se co nejvíce přiblížily zadání a přitom respektovaly podstatu OntoUML.

Minimalita

Minimalita měří, zda se v modelu nevyskytují redundantní koncepty. Ty modelu zbytečně zhoršují čitelnost a přehlednost, a přitom nepřidávají žádné důležité informace. Narozdíl od ER modelů může být omezení redundantních konceptů v OntoUML poněkud obtížnější, neboť kvůli pravidlům může být zapotřebí namodelovat i skutečnosti, které již nejsou součástí domény. To

se často může stát například u Relatoru, kde je potřeba doplnit Mediation vazby a Role a jejich poskytovatele identity. Obdobný problém může nastat pro Phase, kdy by měly být vypsány všechny Phase, kterých jejich nositel může nabýt, jelikož generalizační set pro Phase má být disjoint a complete. V rámci OntoUML tak z pohledu minimality modelu zastávat dva směry: buď bude model obsahovat i nějaké množství redundantních konceptů, nebo budou povoleny určité druhy chyb a anti-patternů, aby se tak omezila velikost modelu.

Redundantní koncepty jsou poměrně subjektivním kritériem, kdy je zřejmě nejvhodnější zkontrolovat každou entity dle její příslušnosti do dané domény. To není obvyklými strojovými kontrolami možné, ale lze najít typy vtahů, které pravděpodobně budou redundantní. V modelu je možné je poznat a měřit například tak, že se najde entita, dále nazývaná izolovaná, která splňuje dvě pravidla. Má pouze jednu hranu a s modelem je spojena přes entitu mající pouze dvě hrany – jedna ji pojí s izolovanou entitou a druhá ji pojí s modelem. Takováto izolovaná entita bude mít poměrně vysokou šanci, že bude redundantní. Například při rozmodelování vztahu pro Relator se taková entita s největší pravděpodobností vyskytne.

Ani jeden ze čtyř modelů neobsahuje izolovanou entitu, a lze je tak považovat za vyhovující kritériu minimality.

Jasnost

Jasnost měří, jak jednoduše lze schéma číst. Je založena na heuristice, že schéma obsahující N vazeb může mít nejvýše N křížení vazeb. Ve skutečnosti to ale je mnohem méně, neboť modeláři usilují o přehledné rozložení modelu.

To je zároveň velmi dobře vidět i na měřených modelech (kromě modelu Invalidity, který vznikl na úrovni technologií RDF a nemá tak grafický model). V modelu Posilovna se vyskytuje jediné křížení vazeb, ale je možné ho poupravit přesunutím entit tak, aby se vazby nekřížily. Model by tak byl o něco větší a mohl by být méně přehledný, jelikož by některé vazby vedly okolo modelu. Ve zbylých dvou modelech se vazby nekříží vůbec.

Křížení vazeb je problémem, který se lépe řeší na grafické úrovni. Způsobem, jak tento problém řešit na úrovni textové, by mohly být *planární grafy*. To jsou grafy, pro které existuje rovinné nakreslení tak, že se žádné dvě hrany nekříží. Tyto grafy musí splňovat určité vzorce, aby byly planárními. Při implementaci těchto postupů do jazyka SPARQL by tak bylo možné detekovat i nevyhnutelné křížení vazeb.

Jednoduchost

Jednoduchost vychází z předpokladu, že schéma je jednoduché, pokud je postaveno na jednoduchých konceptech a dá se mu tak rychle porozumět. Toto měření je založeno na předpokladu, že komplexita modelu roste s počtem vztahů, včetně dědičnosti a agregace. Pro kritérium jednoduchosti může být v OntoUML opět rozdíl, a to proto, že narozdíl od ER modelů mají entity i vazby stereotypy, definující další vlastnosti. Stejná doména reprezentovaná

jako ER model a jako OntoUML model tak může být velmi rozdílná, i když bude snaha vytvořit modely co nejpodobnější. OntoUML model bude mít informace navíc, které ho budou i při větší komplexnosti pravděpodobně činit jednodušší ke čtení a k porozumění než ER model, neboť vztah reprezentovaný vazbou bude jasnější.

Je pravděpodobné, že vyšší počet vztahů nedělá model nutně komplexnějším, ale jedná se spíše o poměr entit a vazeb. Pokud bude vazeb značně více než entit, model se stane nečitelným. Předpokladem je, že model bude příliš komplexní a bude obsahovat příliš mnoho vztahů, pokud poměr vazeb ku entitám bude `3:2` či vyšší. Zároveň je ale jisté, že vazeb bude v modelu minimálně `N-1`, kde `N` je počet entit. Model obsahující takto nízký počet vazeb bude jednoduchý, ale s největší pravděpodobností bude obsahovat mnoho chyb a neobsáhne všechny důležité informace. Poměr vazeb ku entitám jednoduchého modelu je tak určen jako rozmezí od `1:1` do `3:2`.

Všechny čtyři modely splňují pravidlo, že poměr vazeb ku entitám je v rozmezí uvedených poměrů. Pro Posilovnu se jedná o `29:25`, Přehled invalidity `17:15`, Příčiny invalidity `14:13` a pro spojený model Invalidity je poměr `18:17`.

Počty entit i vazeb lze počítat pomocí SPARQL dotazů. Počet entit jednoho stereotypu lze zjistit jednoduchým dotazem:

```
SELECT (count(DISTINCT ?s) AS ?c)
WHERE { ?s rdf:type gufo:Kind. }
```

Obdobně i pro vazby:

```
SELECT (count(DISTINCT ?s) AS ?c)
WHERE { ?s owl:onProperty gufo:mediates. }
```

Pro zjištění počtu všech entit či všech vazeb je zapotřebí do dotazu uvést všechny stereotypy:

```
SELECT (count(DISTINCT ?s) AS ?c)
WHERE { ?s ?p ?o.
      FILTER (?o IN (gufo:Kind, gufo:SubKind, gufo:Role,
              gufo:Phase, gufo:PhaseMixin, gufo:RoleMixin,
              gufo:Mixin, gufo:Relator, gufo:Quality,
              gufo:IntrinsicMode, gufo:Category,
              gufo:VariableCollection, gufo:Quantity)) }
```

Pro sečtení všech entit nebo všech vazeb nelze využít formy sčítající různé objekty začínající jako `rdf:type gufo:` pro entity a `owl:onProperty gufo:` pro vazby. Generalizace je totiž značena jako `rdfs:subClassOf`, ale takto jsou značeny i stereotypy entit Relator, Quality a Mode. Je tak kvůli tomu, že dokumentace gUFO vychází ze starší verze OntoUML. Tato problematika je popsána v následující kapitole, v sekci 5.2.2 popisující nedostatky dokumentace gUFO. Počet vazeb by tak musel obsahovat sečtení `rdfs:subClassOf`

bez těchto tří stereotypů. Pro počet entit by dotaz musel také obsahovat `rdfs:subClassOf`, ale upravený jen pro tyto tři stereotypy.

Expresivita

Expresivita měří bohatost schématu, tedy zda jsou koncepty dostatečně expresivní pro zachycení aspektů reality. Pro toto kritérium je OntoUML oproti ER modelům ve značné výhodě, a to právě díky stereotypům definujícím další vlastnosti entit a vazeb. Již z podstaty OntoUML je velmi pravděpodobné, že model bude dostatečně expresivní pro zachycení aspektů reality.

OntoUML obsahuje 13 stereotypů entit a 11 stereotypů vazeb (nezapočítána Structuration), celkem se tedy jedná o 24 různých stereotypů. Bylo by vhodné, aby jich pro zajištění co nejvyšší expresivity model obsahoval několik. Je ale velmi obtížné, a pro menší modely téměř nemožné, aby se využily všechny. Pokud by se navíc modelář zaměřil na využití co nejvíce různých stereotypů, mohlo by dojít k zanedbání samotného modelování domény a ke zkreslení reality. I menší model by měl ale využívat alespoň třetinu stereotypů, tedy alespoň 8 různých. Tento požadavek je velice nízký, jelikož jen namodelovaný Relator dle pravidel OntoUML se všemi potřebnými částmi se skládá ze šesti různých stereotypů. Horní hranice počtu použitých stereotypů není určena a zároveň neplatí, že model s více stereotypy je expresivnější než model který jich má méně.

Model Posilovna obsahuje 16 různých stereotypů, Přehled invalidity i Příčiny invalidity obsahují každý 10 stereotypů, model Invalidity pak 11. Všechny tedy splňují minimální počet.

Podobně jako u kritéria jednoduchosti, i pro měření expresivity by bylo vhodné nejprve zadefinovat stereotypy pro jednodušší dotazování. Poté by se již jediným dotazem zjistilo, kolik různých stereotypů model obsahuje. Stále je ale možné informaci zjistit za pomoci několika základních dotazů, zjišťujících, zda je v modelu obsažen konkrétní stereotyp.

Ukázka dotazu, který vyjadřuje, zda je v modelu RoleMixin pomocí hodnot ano či ne:

```
ASK { ?s rdf:type gufo:RoleMixin. }
```

Tyto jednoduché dotazy lze agregovat do jednoho složitějšího dotazu využívajícího SELECT tak, aby odpovědí byl počet stereotypů v modelu.

4.5.2 Detekce chyb a anti-patternů

Pomocí jazyka SPARQL lze v modelech vyhledávat chyby a anti-patterny. Tyto možné nedostatky v modelech lze také považovat za měřítko kvality, neboť model by neměl obsahovat chyby, a ideálně by neměl obsahovat ani anti-patterny. Anti-patterny jsou obvykle vyřešeny tím, že je daná část rozmodelována více dopodrobna, což ale vede k větší složitosti a velikosti modelu. Je tak možné, že vyřešením jednoho anti-patternu vznikne nový. Konkrétní

podmínky, tedy jaké nedostatky by se v modelu neměly vyskytovat nebo které jsou naopak povoleny, lze určit individuálně, lze například vybrat jen některé anti-patterny. Navržená míra požadavků na kvalitu je následující: model by neměl obsahovat žádné chyby a neměl by obsahovat žádný z anti-patternů DepPhase, FreeRole, MixRig, MultDep a RelRig.

První dotazy na ukázku budou provedeny na modelu Invalidity. Složitější dotazy budou využity pro upravený model Přehled invalidity. Do toho byly pro lepší demonstraci přidány dvě chyby a jeden anti-pattern.

Nejdříve je možné využít některé jednoduché dotazy, které dokáží odhalit specifický druh chyby v modelu. Obvykle je ale zapotřebí použít více takovýchto dotazů za sebou nebo vytvořit jeden složitější dotaz, aby byly pro danou chybu pokryty všechny možnosti vzniku. To je ukázáno na následujícím příkladu.

Jednou z možných chyb je SubKind který nemá princip identity. Lze tedy vyhledat stereotypy SubKind, které nejsou podtypem a nemohou tak dědit princip identity:

```
SELECT (count(DISTINCT ?s) AS ?c)
WHERE { ?s rdf:type gufo:SubKind.
        FILTER ( !EXISTS { ?s rdfs:subClassOf ?o. } ) }
```

Počet stereotypů SubKind bez vztahu že jsou podtřídou je v modelu:

```
"0"^^xsd:integer
```

Každý SubKind je tedy podtypem, a snižuje se tak pravděpodobnost chyby pro tento stereotyp. Stále je ale možné, že je například podtypem Role či Phase, což je chybně, ale tato kontrola to neodhalí. Pokud by byl SubKind podtypem jiného SubKind, bylo by potřeba kontrolovat i jeho nadtyp, kterým může být opět SubKind. Tento dotaz pouze kontroluje, zda je podtypem jiného stereotypu a již nekontroluje daný nadtyp a zda je to dle pravidel OntoUML povoleno.

Dalším dotazem, který může odhalit chybu pro specifický případ, je porovnání počtu aspektů a počtu Characterization. Každý aspekt musí mít přesně jednu Characterization vazbu. Pokud se počty nerovnají, v modelu je jistě chyba.

Sečtení stereotypů Quality a Mode:

```
SELECT (count(DISTINCT ?s) AS ?c)
WHERE { { ?s rdfs:subClassOf gufo:Quality. }
        UNION
        { ?s rdfs:subClassOf gufo:IntrinsicMode. } }
```

Stereotypů Quality a Mode je v modelu celkem:

```
"5"^^xsd:integer
```

Sečtení vazeb Characterization:

```
SELECT (count(distinct ?s) as ?c)
WHERE { ?s owl:onProperty gufo:inheresIn }
```

Vazeb stereotypu Characterization je v modelu celkem:

```
"5"^^xsd:integer
```

Počty se rovnají, čímž se snižuje riziko chyby pro tyto stereotypy, ale nezaručuje, že jsou dobře. Stále je možné, že jeden z aspektů nemá žádnou Characterization vazbu a jiný má naopak dvě, obě možnosti jsou chybné.

Splnění těchto podmínek lze považovat za nutné podmínky modelu bez chyb, nikoliv za dostačující.

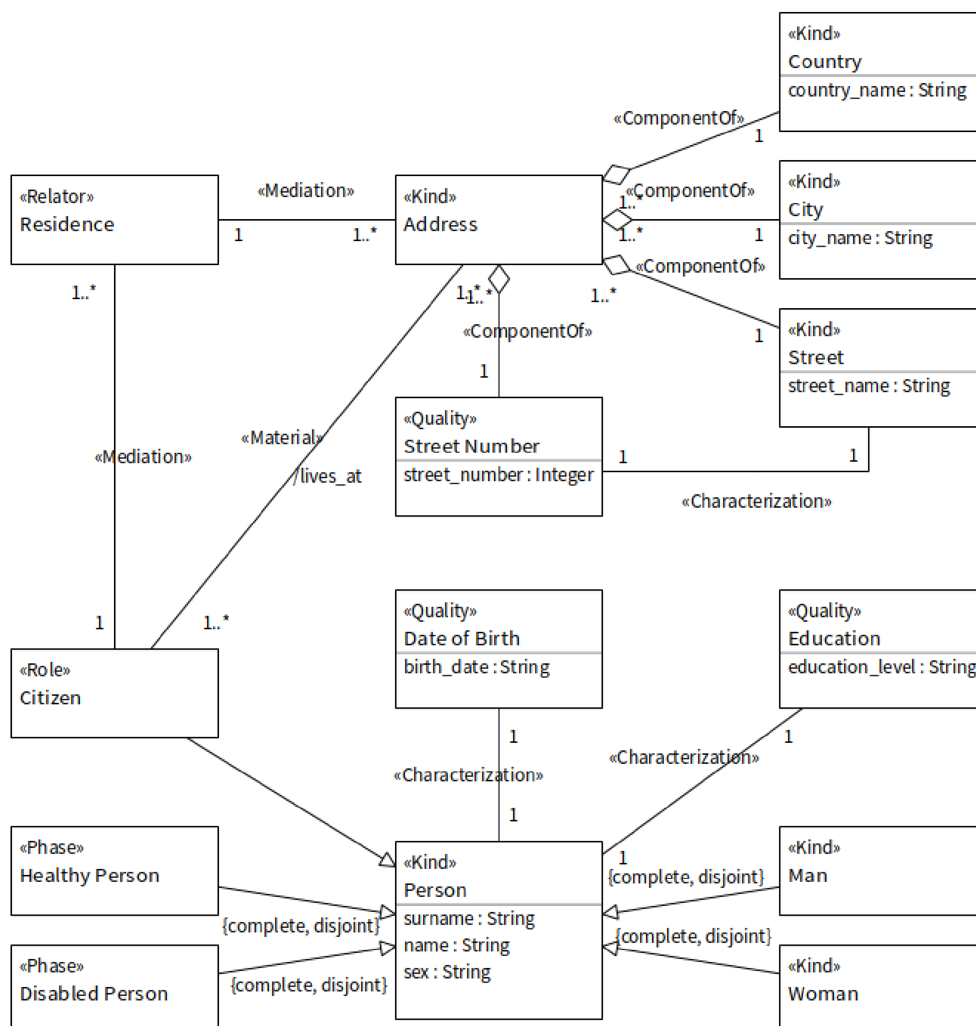
Pro lepší ukázku detekce chyb a anti-patternů byl model Přehled invalidity upraven, aby se na něm dala provádět validace. Byly přidány dvě chyby a jeden anti-pattern. Upravený model je přiložen na obrázku 4.4. RDF zápis modelu pomocí gUFO je v příloze.

`Man` a `Woman`, které byly SubKind, jsou v chybném modelu změněny na Kind a stále ponechány jako podtyp `Person`. Dále byla odebrána Role `Home Address` a Relator `Residence` byl spojen přímo s Kind `Address`.

Změnou entit `Man` a `Woman` ze SubKind na Kind a jejich ponecháním jako podtyp Kind `Person` vznikly dvě entity Kind dědící od jiné entity Kind. To je proti pravidlům OntoUML, jelikož Kind již má svůj princip identity a nemůže ho tedy dědit. Proto, jak je uvedeno v dokumentaci, Kind není povoleným podtypem a nadtypem stereotypu Kind.

Tuto chybu lze najít pomocí následujícího dotazu. Ten nejdříve vyhledá všechny stereotypy Kind, které jsou podtypem. Jelikož Kind nemůže dědit princip identity, je povoleno, aby byl podtypem pouze stereotypů Category a Mixin, ty jsou tedy z dotazu odfiltrovány. Nakonec jsou z výsledků odstraněny prázdné uzly. To je potřeba kvůli tomu, že pro vlastnosti vazeb se využívá prázdný uzel který je podtypem počáteční entity, a nesprávně by se tak detekoval jako chyba.

```
SELECT ?m
WHERE { ?m rdf:type gufo:Kind.
        ?m rdfs:subClassOf ?p.
        FILTER ( !EXISTS { ?p rdfs:type gufo:Category. } )
        FILTER ( !EXISTS { ?p rdfs:type gufo:Mixin. } )
        FILTER ( !isBlank(?p) ) }
```



Obrázek 4.4: Model Přehled invalidity obsahující chyby a anti-pattern

Anti-pattern RelRig značí, že Relator je spojený Mediation vazbou s rigidním stereotypem. Rigidními stereotypy jsou Kind, SubKind, Category, Mode, Quality, Quantity a Relator. Výjimkou rigidního stereotypu se kterým může být Relator spojen, aby se nejednalo o anti-pattern, je další Relator. Anti-pattern tedy vznikne při spojení Relatoru Mediation vazbou s Kind, SubKind, Category, Mode, Quality nebo Quantity. V chybném modelu tak došlo právě ke spojení stereotypů Relator a Kind, jelikož byla odebrána Role `Home Address` a tedy spojením Relatoru `Residence` přímo s Kind `Address`.

Anti-pattern RelRig lze detekovat následujícím dotazem. Nejdříve se najdou Relatory, které mají jako podtyp prázdný uzel. Tam jsou uloženy vlastnosti vazby. Najdou se jen ty vazby, které jsou Mediation. Relator může být ve

vazbě počáteční i cílovou entitou, je tedy potřeba zohlednit obě možnosti. Nakonec se vybere cílová entita, a pokud je stereotypu Kind, SubKind, Quantity, Mode nebo Quality, bude vypsána jako výsledek dotazu.

```
SELECT ?a
WHERE { ?m rdfs:subClassOf gufo:Relator.
        ?m rdfs:subClassOf ?p.

        { ?p owl:onProperty gufo:mediates. }
        UNION
        { ?p owl:onProperty ?p2.
          ?p2 owl:inverseOf gufo:mediates. }

        ?p owl:onClass ?a.

        { ?a rdf:type gufo:Kind. }
        UNION
        { ?a rdf:type gufo:SubKind. }
        UNION
        { ?a rdf:type gufo:Category. }
        UNION
        { ?a rdfs:subClassOf gufo:Quantity. }
        UNION
        { ?a rdfs:subClassOf gufo:Mode. }
        UNION
        { ?a rdfs:subClassOf gufo:Quality. } }
```

Posledním dotazem v této podsekci je ukázka anti-patternu FreeRole. Ten se sice nenachází v žádném z využívaných modelů, ale jedná se o užitečný dotaz, který byl zmíněn a z menší části vytvořen již v jazyce SHACL v předchozí sekci. Dotaz na anti-pattern FreeRole také slouží jako ukázka jednoho ze složitých dotazů SPARQL. Bylo zjištěno, že reprezentace nadtypu a podtypu je v SHACL velmi obtížná, a plná reprezentace validace FreeRole by tak byla dlouhá a komplikovaná. Součástí tohoto dotazu jsou čtyři jednotlivé dotazy na podtyp.

V dotazu se nejdříve vyhledá Role, dále nazvána jako `freeRole`, která je podtypem jiné Role, dále nazvané jako `parentRole`. Pro `parentRole` se zkontroluje přítomnost Mediation vazby pro oba směry. Poté se zkontroluje existence Mediation vazby pro `freeRole`, opět pro oba směry. Poslední část dotazu specifikuje, že Mediation vazba `freeRole` a `parentRole` nesmí být ta stejná. Jelikož je `freeRole` podtypem `parentRole`, dědí její vlastnosti a tedy i Mediation vazbu. Při hledání Mediation vazby pro `freeRole` se tak musí odfiltrovat tato děděná. Aby vznikl anti-pattern FreeRole, `freeRole` nesmí mít žádnou jinou Mediation vazbu:

```
SELECT ?freeRole ?parentRole
WHERE {
    ?freeRole rdf:type gufo:Role.
    ?freeRole rdfs:subClassOf ?parentRole.
    ?parentRole rdf:type gufo:Role.
    ?parentRole rdfs:subClassOf ?parentRoleRestr.
    ?parentRoleRestr rdf:type owl:Restriction.

    { ?parentRoleRestr owl:onProperty gufo:mediates. }
    UNION
    { ?parentRoleRestr owl:onProperty ?p1.
      ?p1 owl:inverseOf gufo:mediates. }

    FILTER NOT EXISTS {
        ?freeRole rdfs:subClassOf ?freeRoleRestr.
        ?freeRoleRestr rdf:type owl:Restriction.

        { ?freeRoleRestr owl:onProperty gufo:mediates. }
        UNION
        { ?freeRoleRestr owl:onProperty ?p2.
           ?p2 owl:inverseOf gufo:mediates. }

        ?parentRole rdfs:subClassOf ?anyParentRestr.
        ?anyParentRestr rdf:type owl:Restriction.
        FILTER (?freeRoleRestr != ?anyParentRestr)
    }
}
```

Následuje seznam dalších vztahů, které by mohly a měly být kontrolovány pomocí SPARQL a dodrženy, aby nebyla porušena pravidla OntoUML:

- Role a Phase dědí princip identity.
- Phase je vždy členem generalizačního setu, který je disjoint a complete.
- Vazby MemberOf mají multiplicitu pro část alespoň 2..*.
- Mode a Quality mají vždy Characterization vazbu.
- Relator má dvě Mediation vazby, nebo jednu s multiplicitou alespoň 2..*.

Také by měly být detekovány a opraveny následující anti-patterny:

- Phase má Mediation vazbu – DepPhase.
- Mixin nemá alespoň jeden rigidní a alespoň jeden anti-rigidní podtyp – MixRig.
- Role má přesně jednu Mediation vazbu, a ne víc – MultDep.

4.5.3 Dotazování nad daty

SPARQL lze využít pro dotazy nad modelem jako tomu bylo dosud, nebo pro dotazování nad daty. Pro model Invalidity byla vytvořena data (nevycházející ze skutečných osob), která jsou instancemi OntoUML modelu v gUFO, a lze se nad nimi dotazovat a vyhledávat v nich. Pro kontrolu dat, zda byly zadány všechny požadované údaje v požadovaném formátu, se využívá validační jazyk SHACL.

Aby bylo možné se dotazovat nad daty, je potřeba přidat další prefix:

```
PREFIX schema: <http://schema.org/>
```

Obvyklým dotazem je například zobrazení data narození všech osob, jejichž důvodem invalidity byla cirhóza jater:

```
SELECT ?o
WHERE { ?s schema:birth_date ?o.
        ?s schema:cause_name "Cirrhosis of the liver". }
```

Tento důvod invalidity splňuje jediná osoba s datem narození:

```
"1983-01-29"^^xsd:date
```

Kromě klasického dotazování lze SPARQL využít na měření kvality a kvantity dat. Pro kvalitu může být například žádoucí mít data z co nejvíce zemí pro dostatečně reprezentativní vzorek. Osoby v datech mají v rámci své adresy uveden i stát, a je tak možné se dotazovat na počet různých států, které jsou v datech:

```
SELECT (count(DISTINCT ?o) AS ?c)
WHERE { ?s schema:country_name ?o }
```

Počet různých států v datech:

```
"4"^^xsd:integer
```

Pro měření kvantity se lze podobným způsobem dotázat, o kolika osobách jsou v datech informace:

```
SELECT (count(DISTINCT ?s) AS ?c)
WHERE { ?s ?p schema:Person }
```

Výsledkem je opět hodnota 4, neboť každá osoba v datech je obyvatelem jiného státu:

```
"4"^^xsd:integer
```

Pomocí jazyka SPARQL byla změřena kvalita modelů, či bylo navrženo, jak jazyk pro měření využít. Na příkladech bylo ukázáno, že ho lze velmi dobře využít i na nalezení chyb a anti-patternů které se v OntoUML modelu mohou vyskytnout. Součástí byly i příklady vyhledávání v datech, které také mohou sloužit jako měření kvality dat.

Kapitola 5

Nástrojová podpora

Tato kapitola se v první části zabývá porovnáním nástrojů OpenPonk a Visual Paradigm a v druhé části návrhem exportu do gUFO pro nástroj OpenPonk. Nejdříve jsou tedy popsány a porovnány nástroje OpenPonk a Visual Paradigm, jelikož oba podporují modelování v OntoUML a verifikaci modelu. Nástroj Visual Paradigm navíc již obsahuje možnost exportu modelu do gUFO. Pro práci bylo této možnosti využito pro usnadnění práce s přepisem modelu do gUFO, z exportu se ale pouze vychází, není využit v přesné vygenerované podobě. Obsahoval mnoho chyb a mnoho informací naopak chybělo, proto byly všechny exportované modely ručně upraveny.

Druhá část kapitoly se již věnuje návrhu exportu do gUFO pro nástroj OpenPonk, vycházející z oficiální dokumentace gUFO [4]. Návrh popisuje chyby přítomné v exportu nástroje Visual Paradigm, doplňuje stereotypy chybějící v oficiální dokumentaci a navrhuje možné úpravy reprezentace některých stereotypů dle verze OntoUML 2.0. V práci je vytvořen pouze teoretický návrh s analýzou potřeb a ukázkami výstupů, bez samotné implementace.

5.1 Porovnání nástrojů OpenPonk a Visual Paradigm

Pro práci s modely a jejich následný export do gUFO jsou využívány dva nástroje, a to OpenPonk a Visual Paradigm s OntoUML pluginem. Oba nástroje jsou používány s výchozím uživatelským nastavením.

OpenPonk je modelovací nástroj vyvinutý na FIT ČVUT v Praze ve vývojovém prostředí Pharo. Je zaměřený pouze na několik jazyků, a to převážně UML (konkrétně diagramy tříd) a OntoUML. [23]

Visual Paradigm je rozšířený modelovací nástroj podporující mnoho jazyků a diagramů, mezi které patří BPMN, databázové modely a mnoho druhů UML diagramů, tedy nejen diagram tříd. Je také možné vytvářet wireframy, součástí jsou i nástroje pro agilní vývoj a vizualizace dat pomocí tabulek a grafů. [24]

Pro tuto práci je využit Visual Paradigm Community Edition, tedy verze zdarma pro nekomerční využití, kterou lze stáhnout z oficiálních stránek Visual Paradigm [25]. Ta je rozšířena o plugin dostupný z GitHubu [12] přidávající modelování v OntoUML. Visual Paradigm s pluginem pro OntoUML bude dále nazýván pouze Visual Paradigm.

5.1.1 Využití nástrojů v práci

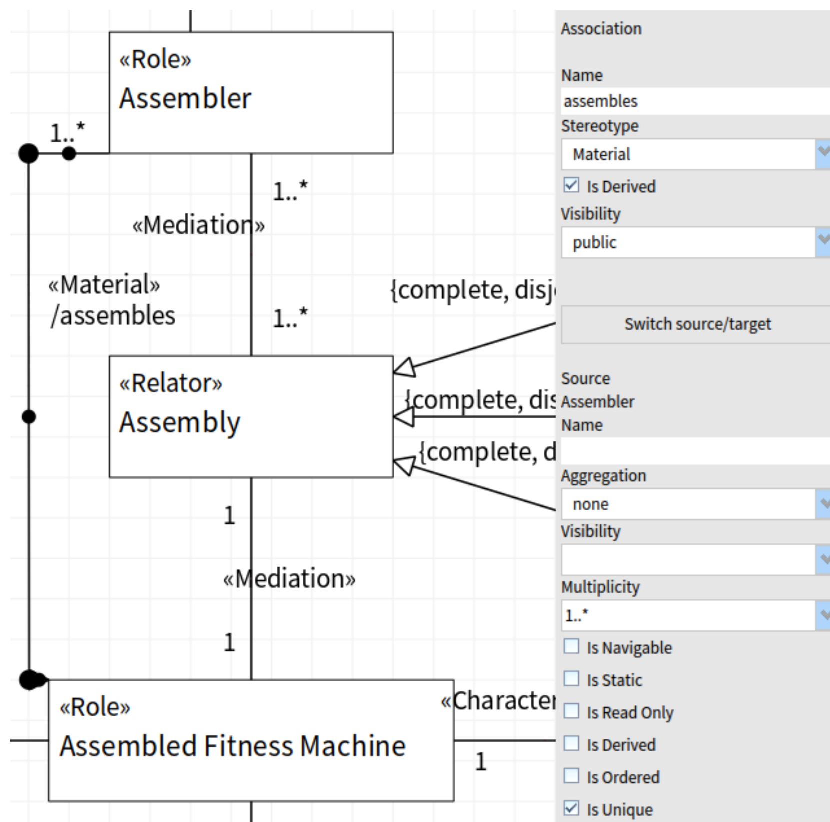
OpenPonk byl použit na vytvoření OntoUML modelu, tedy na jeho vymyšlení a přehledné rozvrhnutí jednotlivých částí. Obrázky modelů v této práci jsou z nástroje OpenPonk. Nástroj Visual Paradigm byl také použit na vytvoření OntoUML modelu, ale čistě za účelem exportu modelu do gUFO. Tím se usnadnila práce, jelikož nebylo potřeba psát celý export ručně. Pro lepší čitelnost a přehlednost byly z výsledného exportu některé informace odstraněny, jako například jazykové tagy. Jiné informace byly přidány ručně, jelikož se je nepodařilo exportovat z modelu. Export také obsahoval chyby, které byly ručně opraveny. Chyb bylo více druhů a detailněji jsou popsány v sekci 5.2.

Důvodů, proč modely byly nejdříve vytvářeny v OpenPonku a poté opět ručně modelovány ve Visual Paradigm, je několik. Prvním z nich jsou zkušenosti s nástrojem OpenPonk, zatímco práce ve Visual Paradigm pro mě, jakožto nového uživatele, byla obtížná a neintuitivní. Dalším důležitým důvodem je skutečnost, že OpenPonk je poměrně jednoduchý nástroj s jednoduchým rozhraním. Nenabízí tedy tak obsáhlé možnosti jako Visual Paradigm, ale je velmi přehledný a práce v něm je rychlá. Je navíc uzpůsoben modelování v OntoUML, a je tedy možné rovnou vytvářet entity s požadovanými stereotypy, což ve Visual Paradigm není možné. Jelikož modelování v OntoUML ve Visual Paradigm je možné pouze díky pluginu, je nutné vytvářet nejdříve UML model, a postupně jednotlivým entitám a vazbám přiřazovat stereotypy.

V nástroji OpenPonk je pohodlnější a přesnější zarovnávání entit díky mřížce a možnostem posunu a přiblížení modelu, které bude popsáno později. Tyto funkce sice nemají vliv na samotný export a informace obsažené v modelu, ale ovlivňuje to přehlednost modelu a orientaci v něm.

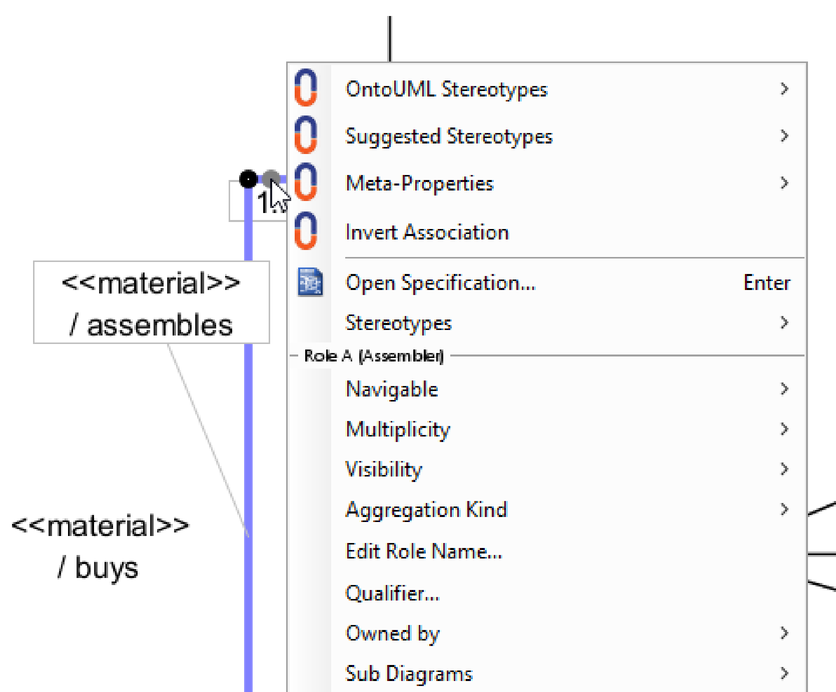
5.1.2 Nastavení vlastností vazeb

Jeden z hlavních rozdílů při práci v nástrojích spočívá v nastavení vlastností vazeb. V OpenPonku se při kliknutí na entitu či vazbu objeví v pravé části obrazovky menu, viz obrázek 5.1, kde lze nastavovat vlastnosti označené entity či vazby. V případě asociace se jedná o název, stereotyp, multiplicity, agregace, směr, ale i nastavení viditelnosti či odvozenost vazby. Menu asociace je otevřené a je možné v něm dělat změny, dokud je daná asociace označena, tedy dokud uživatel neoznačí jinou vazbu nebo entitu, nebo celý model. Po zvolení asociace je tedy možné navolit všechny potřebné vlastnosti najednou, a to pro oba konce vazby.



Obrázek 5.1: Menu v pravé části OpenPonku obsahující všechny vlastnosti nastavitelné pro asociaci. Obrázek je oříznut, menu pokračuje dál stejnými možnostmi pro druhý konec asociace, tedy cíl, target.

Visual Paradigm nabízí stejné možnosti nastavení vlastností, ale s tím rozdílem, že kliknutím na vazbu pravým tlačítkem myši se zobrazí okno s výběrem možností, viz obrázek 5.2. Těmi jsou tedy pro asociaci opět stereotyp, multiplicity, agregace a další. Stejně jako v OpenPonku se po zvolení dané vlastnosti objeví menu s hodnotami, tedy například nastavení rozmezí multiplicity. Po výběru se ale celé okno zavře a je potřeba ho znovu otevřít, pokud chce uživatel u vazby nastavit další vlastnost. Pro nastavení vlastnosti vztahující se ke konkrétnímu konci vazby, například multiplicity či agregace, je nutné kliknout na daný konec asociace. Při kliknutí do středové části asociace se zobrazí okno s méně možnostmi, jelikož zde nebudou zobrazeny vlastnosti nastavitelné právě pro konce asociace.



Obrázek 5.2: Okno ve Visual Paradigm obsahující všechny vlastnosti nastavitelné pro asociaci. Obrázek je oříznut, okno pokračuje možnostmi kopírování, mazání a zobrazení.

5.1.3 Pohyb po pracovní ploše

Dalším rozdílem, ovlivňujícím práci zejména s většími modely, je pohyb po pracovní ploše. V OpenPonku funguje přibližování a oddalování modelu pomocí kolečka myši, a držením pravého tlačítka se lze pohybovat po pracovní ploše. Ve Visual Paradigm slouží kolečko myši k posunu nahoru a dolů, pro posun je zapotřebí ho držet. Pro přibližování a oddalování je potřeba podržet klávesu Ctrl a poté přibližovat či oddalovat plochu kolečkem. Dalším způso-

bem je použití ikony v pravém dolním rohu. Zobrazí se malé okno, kde se pomocí náhledu modelu lze posouvat po pracovní ploše. Přiblížení a oddálení se nastavuje klikáním na ikony lupy, což je velmi nepraktické. Pro oba způsoby jsou ale v nástroji omezení. Například, není obvykle možný takový posun, aby okrajová entita byla ve středu obrazovky, protože plocha ve Visual Paradigm má určitou velikost a ohraničení.

5.1.4 Funkce Visual Paradigm

Visual Paradigm má ale také mnoho funkcí, které v OpenPonku chybí. Mezi nejzásadnější patří dvě, které jsou v dnešní době považovány za samozřejmost, a to možnost označit a přesunout více objektů najednou, a možnost vrátit změny o krok zpět. V OpenPonku tyto dvě funkce nejsou implementovány. Je tedy nutné přesouvat všechny objekty po jednom, což při přeskupování větších částí modelu bývá velmi zdlouhavé. V OpenPonku zároveň neexistuje tlačítko zpět ani příslušná klávesová zkrátka – co se omylem vytvoří se musí smazat, a co se omylem smaže se musí opět vytvořit. To je nepraktické obzvlášť při nechtěném odstranění entity, ke které vede několik již plně nastavených asociací, které jsou tím také smazány.

Další funkcionalitou jsou generalizační sety. Ty jsou v OpenPonku plně implementovány, ale jejich vytváření je poněkud zdlouhavé, jelikož obnáší spoustu potvrzování, a je potřeba vše opakovaně navolit pro každou generalizaci které se týká. Ve Visual Paradigm stačí generalizační set vytvořit jednou a v jeho nastavení pouze přidat všechny generalizace, kterých se týká.

5.1.5 Závěr porovnání nástrojů

Visual Paradigm je známější a rozšířenější než OpenPonk, a jak již bylo zmíněno, nabízí možnosti, které v OpenPonku chybí. Existují i další nástroje, kde lze vytvářet OntoUML modely, například Enterprise Architect [26]. Enterprise Architect obsahuje, stejně jako Visual Paradigm, spoustu jazyků a typů diagramů a nabízí mnoho možností a nastavení. Bohužel s nástrojem Visual Paradigm sdílí i některé stinné stránky, tedy například nastavení vlastností přes okno, které se po nastavení jedné vlastnosti zavře. Kvůli velkému množství možností a oken se programy stávají nepřehlednými, zejména pro nové uživatele. V nástroji OpenPonk je vše velmi minimalistické, většinu obrazovky zaujímá modelovací plocha a lze se tak soustředit na to důležité – samotné modelování. Díky tomu je, alespoň dle mého názoru, modelování v OpenPonku příjemnější, protože je oproti předchozím dvěma nástrojům velmi jednoduché, intuitivní a rychlé. To je vidět zejména na popsaných příkladech, kterými jsou vytváření entit a vazeb se stereotypem, pohyb po pracovní ploše a nastavení vlastností vazby. Proto by bylo vhodné přidat funkci exportu do gUFO do nástroje OpenPonk, i přesto, že již existuje jiný modelovací nástroj, který touto funkcí disponuje.

5.2 Návrh exportu do gUFO pro nástroj OpenPonk

Tato sekce se zabývá návrhem exportu OntoUML modelu z nástroje OpenPonk do gUFO. Navržený export je porovnáván s exportem v nástroji Visual Paradigm a vychází z oficiální dokumentace gUFO [4]. Vztahy které se ve Visual Paradigm neexportují správně jsou zde popsány, společně s návrhem správné varianty. Sekce zároveň obsahuje návrh reprezentace několika funkcí, kterými OpenPonk disponuje navíc, například možnost vkládat poznámky. Součástí je doplnění stereotypů Containment a Derivation které v oficiální dokumentaci chybí, a úprava reprezentace některých stereotypů dle verze OntoUML 2.0. Jedná se pouze o teoretický návrh s analýzou potřeb a ukázkami výstupů, bez samotné implementace.

5.2.1 Směr vazeb

Před samotným návrhem je důležité zmínit další podstatný rozdíl mezi nástroji OpenPonk a Visual Paradigm, a tím je směr vazeb. V některých případech, například pro vztah Relatoru a Role, na směru příliš nezáleží. Pro vztah celku a části je směr v samotném modelu také nepodstatný, protože se s ním dále nepracuje, ale z modelu a jeho uspořádání je jasné, která entita je celkem a která částí. Pokud se ale model exportuje do gUFO, je tato informace zásadní, protože při případném otočení směru se část a celek zamění. Visual Paradigm i OpenPonk rozpoznávají směr vazeb a dokáží ho po namodelování jednoduše změnit. OpenPonk se řídí pravidly dokumentace OntoUML [27], kde jsou uvedeny i směry vazeb. Modelář si tedy může jednoduše dohledat, jakým směrem by měly být vazby v modelu orientovány. Pro všechny vazby kde je tato informace důležitá, například vazby celek a část, nebo aspekt a jeho nositel, je kontrola směru vazeb zahrnuta ve verifikacích. Pokud tedy modelář udělá chybu, která by měla negativní vliv na expert do gUFO, bude na ni upozorněn. Součástí chyb jsou odkazy do dokumentace, i nový uživatel tak najde zdroj ze kterého při modelování vycházet. Ve Visual Paradigm kontrola směru vazby není, modelář tak často zjistí až po exportu do gUFO, že některé vazby nebyly v požadovaném směru, jelikož informace v exportu jsou obrácené. Zároveň není jasné, jaká pravidla by pro směr měla být dodržena, pro většinu vazeb se jeví jako opačná oproti nástroji OpenPonk a tedy i dokumentaci OntoUML kterou využívá. Například, dle dokumentace a OpenPonku je nutné, aby vazba Characterization mířila od nositele k aspektu. Aby exportovaný výsledek ve Visual Paradigm odpovídal realitě, je potřeba vazbu mířit od aspektu k nositeli, tedy opačným směrem.

Ve skutečnosti není příliš důležité, která entita je zdrojem a která cílem, jestli se jako zdroj určí spíše celek, nebo spíše část, jedná se pouze o úhel pohledu. Je ale důležité, aby v rámci nástroje tato pravidla byla jednotná, jasně řečená a ideálně kontrolovaná, aby s modelem bylo možné dále pracovat.

5.2.2 Návrh funkcí a stereotypů

Před samotným exportem do gUFO by v OpenPonku mělo být k dispozici nastavení exportu, podobně jako v nástroji Visual Paradigm. To by obsahovalo možnosti zadání URI či výběr výstupního formátu. Navíc by obsahovalo možnost přidat jazykové tagy. Při exportu přes Visual Paradigm jsou ve výchozím nastavení na všech místech anglické tagy. To lze změnit, ale pouze pro jednotlivé entity/vazby, není možné tagy změnit pro celý model najednou.

Následuje seznam všech stereotypů, vlastností a funkcí OpenPonku a návrh jejich exportu do gUFO, s porovnáním s gUFO dokumentací a exportem ve Visual Paradigm. V OpenPonku jsou při vytváření OntoUML modelu dostupné i nějaké funkce využívající se v UML, které v OntoUML povoleny nejsou, a pro ty tedy nebude navržena reprezentace.

Generalizace

Generalizace je reprezentována stejně jako v dokumentaci, a oproti nástroji Visual Paradigm je zde jedna oprava. Visual Paradigm při využití generalizací často nesprávně k nadtypu přidává informaci o funkčním celku. Chybná reprezentace generovaná nástrojem Visual Paradigm vypadá takto:

```
:Person rdf:type gufo:Kind;
    rdfs:subClassOf gufo:FunctionalComplex.
:Student rdf:type gufo:Role;
    rdfs:subClassOf :Person.
```

Tento problém se týká všech stereotypů, kde se vyskytuje generalizace a daný stereotyp je nadtypem, tedy Kind, Category, Mixin, PhaseMixin a RoleMixin. Navržená opravená varianta je bez funkčního celku:

```
:Person rdf:type gufo:Kind.
:Student rdf:type gufo:Role;
    rdfs:subClassOf :Person.
```

Asociace

Reprezentace asociace bez stereotypu není navržena, jelikož dle pravidel OntoUML musí mít každá vazba přiřazen stereotyp. Pokud by některá vazba stereotyp přiřazen neměla, export modelu do gUFO nebude povolen.

Třída

Reprezentace třídy bez stereotypu není navržena, protože podobně jako u asociace, dle pravidel OntoUML musí mít každá entita přiřazen stereotyp. Pokud by některá entita stereotyp přiřazen neměla, export modelu do gUFO nebude povolen.

Atribut

Atribut není v dokumentaci gUFO definován, neboť se nejedná o vlastnost existující pouze v OntoUML, jako jsou stereotypy. Reprezentace je stejná jako

ve Visual Paradigm, dle dokumentace OWL [9]. Atribut je tak vyjádřen pomocí `rdf:type owl:DatatypeProperty`.

```
:name rdfs:domain :Person;
    rdfs:range xsd:string;
    rdf:type owl:DatatypeProperty;
    rdfs:subPropertyOf gufo:hasQualityValue.
```

Operation

Reprezentace Operation není navržena, protože narozdíl od atributů se v OntoUML Operation nevyužívá.

Enumeration

Reprezentace Enumeration také není navržena, jelikož Enumeration není OntoUML stereotypem. Pro jeho reprezentaci lze využít Mode.

Enumeration Literal

Enumeration Literal se využívá v Enumeration, která není stereotypem OntoUML, a proto ani Enumeration Literal není potřeba v rámci návrhu reprezentovat.

Category

Category je reprezentována částečně odlišně oproti Visual Paradigm, jelikož export do gUFO často obsahuje informaci, že Category je funkčním celkem, což není pravda. Tato informace je tedy odebrána:

```
:LivingThing rdf:type gufo:Category.
```

Collective

V dokumentaci gUFO není Collective příliš popsán a nejsou zde žádné ukázky. V nástroji Visual Paradigm je Collective vnímán podobně jako Relator, Quality a Mode, tedy jako podtyp stereotypu Kind. V aktuálně využívané ustálené verzi OntoUML 2.0 je Collective samostatným stereotypem, a tak je i reprezentován v návrhu:

```
:SchoolClass rdf:type gufo:VariableCollection, gufo:Composite.
:Student rdf:type gufo:Role;
    gufo:isCollectionMemberOf :SchoolClass.
```

Ve Visual Paradigm zároveň není povoleno, aby Phase a Role byla podtypem stereotypu Collective. O chybu se ale nejedná, dle pravidel OntoUML 2.0 je zcela v pořádku aby byly podtypem. Proto i v návrhu je povoleno, aby měl Collective Role či Phase.

Kind

Kind je, stejně jako Category, reprezentován částečně odlišně oproti Visual Paradigm. Export do gUFO často obsahuje informaci, že Category je funkčním celkem, což není pravda, a tato informace je tedy odebrána:

```
:Person rdf:type gufo:Kind.
```

Mixin

Stejně jako všechny stereotypy, které jsou nadtypem pro generalizace, je Mixin reprezentován částečně odlišně oproti Visual Paradigm, kvůli chybné informaci, že Mixin je funkčním celkem, tato informace je tedy odebrána:

```
:IllegalGoods rdf:type gufo:Mixin.

:StolenCar rdf:type gufo:Role.
:Drugs rdf:type gufo:Kind.
:StolenCar rdfs:subClassOf :IllegalGoods.
:Drugs rdfs:subClassOf :IllegalGoods.
```

Mode

Visual Paradigm má oproti dokumentaci gUFO nepřesný export, obsahující navíc spojení `gufo:Kind`:

```
:Headache rdf:type gufo:Kind;
   rdfs:subClassOf gufo:IntrinsicMode.
```

Správná varianta dle dokumentace je tedy:

```
:Headache rdf:type owl:Class;
   rdfs:subClassOf gufo:IntrinsicMode.
```

Navrhovaná podoba reprezentace Relatoru je velmi odlišná od exportu Visual Paradigm, ale i od dokumentace gUFO:

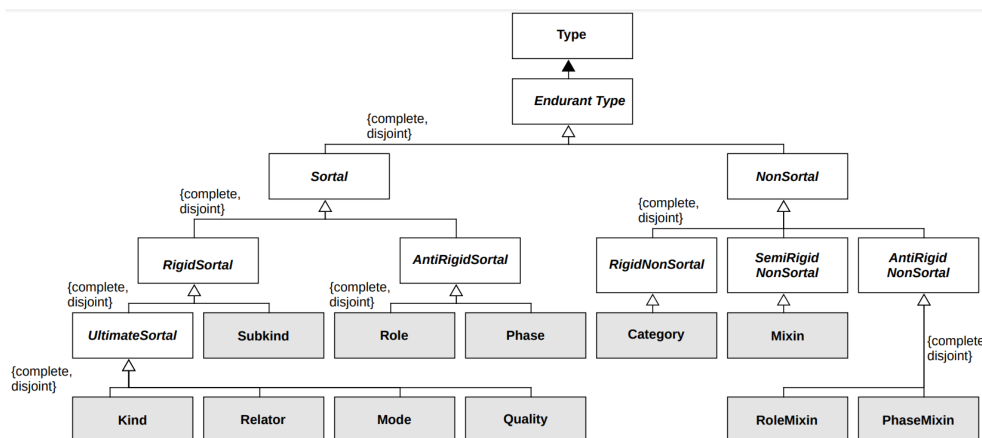
```
:Headache rdf:type gufo:IntrinsicMode.
```

Dokumentace gUFO vychází z mnoha zdrojů, z nichž jedním je práce z roku 2018 [28] zabývající se úpravou stereotypů Relator, Quality a Mode. Ve starší verzi OntoUML nebyla možnost, aby Relatory a aspekty, tedy Quality a Mode, poskytovaly identitu a měly Role či Phase. Aby jim byla tato možnost přidána, staly se podtypy stereotypu Kind, který byl v té době jediným zástupcem takzvaného Ultimate Sortalu, který má tyto vlastnosti. Vznikly tak stereotypy RelatorKind nahrazující Relator, QualityKind nahrazující Quality, a ModeKind nahrazující Kind.

Dnes se již Relator, Quality i Mode považují za samostatné stereotypy, které nejsou podtypy Kind, a proto je navržena tato novější reprezentace. Dělení stereotypů, které se vyučuje i na FIT ČVUT v Praze, je popsáno na obrázku 5.3.

Ve Visual Paradigm není povoleno, aby Phase a Role byla podtypem stereotypu Mode. Dle pravidel OntoUML 2.0 je v pořádku, aby byly podtypem

Mode, a nejedná se tedy o chybu. Proto i v návrhu je povoleno, aby měl Mode Role či Phase.



Obrázek 5.3: Hierarchie stereotypů v OntoUML.

Phase

Reprezentace stereotypu Phase je stejná jako v dokumentaci i Visual Paradigm, ale jsou upravena pravidla. Ve Visual Paradigm není povoleno, aby Phase byla podtypem stereotypů Relator, Mode, Quality, Quantity a Collective. Dle OntoUML 2.0 je v pořádku, aby Phase byla podtypem těchto stereotypů a proto je tak v rámci návrhu učiněno, reprezentace jinak zůstává stejná:

```
:Adult rdf:type gufo:Phase;
    rdfs:subClassOf :Person.
```

PhaseMixin

Jak již bylo řečeno, ve Visual Paradigm není povoleno, aby Phase byla podtypem stereotypů Relator, Mode, Quality, Quantity a Collective. Tato chyba je ale zobrazena až při vytvoření PhaseMixinu, mající pod sebou alespoň jednu z takovýchto Phase. O chybu se ale nejedná, dle pravidel OntoUML 2.0 je zcela v pořádku aby Phase byla podtypem těchto stereotypů, a aby PhaseMixin sdružoval takovéto Phase. PhaseMixin je zároveň dalším stereotypem, který je ve Visual Paradigm často chybně označen za funkční celek, tato informace je tedy odebrána:

```
:Deceased rdf:type gufo:PhaseMixin.

:DeceasedPerson rdf:type gufo:Phase.
:DeceasedAnimal rdf:type gufo:Phase.
:DeceasedPerson rdfs:subClassOf :Deceased.
:DeceasedAnimal rdfs:subClassOf :Deceased.
```

Quality

Stejně jako u stereotypu Mode, i zde je navrhovaná reprezentace odlišná od Visual Paradigm a i dokumentace gUFO. Opět je tak učiněno z důvodu existence Quality jakožto samostatného stereotypu bez závislosti na Kind, jako bylo zapotřebí dříve, dle již zmíněné práce [28]. Obdobně jako pro Mode, ve Visual Paradigm není povoleno, aby Phase a Role byla podtypem stereotypu Quality. Dle pravidel OntoUML 2.0 je v pořádku, aby byly podtypem, a nejedná se tedy o chybu. Proto i v návrhu je povoleno, aby měla Quality Role či Phase.

```
:MoonsMass rdf:type gufo:Quality.
:Moon rdf:type gufo:Kind.
:MoonsMass gufo:inheresIn :Moon.
```

Quantity

Quantity je dalším stereotypem, kde je navrhovaná reprezentace od Visual Paradigm odlišná z důvodu existence Quantity jakožto samostatného stereotypu bez závislosti na Kind, jako bylo zapotřebí dříve, dle [28]. Stejně jako pro ostatní stereotypy s těmito pravidly, ve Visual Paradigm není povoleno, aby Phase a Role byla podtypem stereotypu Quantity. O chybu se opět nejedná, dle pravidel OntoUML 2.0 je zcela v pořádku, aby byly podtypem, a proto i v návrhu je povoleno, aby měla Quantity Role či Phase.

```
:Beer rdf:type gufo:Quantity.
```

Relator

Relator je posledním stereotypem, kde je návrh odlišný z důvodu existence Relatoru jakožto samostatného stereotypu bez závislosti na Kind, jako bylo zapotřebí dříve, dle [28]. Stejně jako u ostatních stereotypů z této skupiny, ve Visual Paradigm není povoleno, aby Phase a Role byla podtypem stereotypu Relator. Jak již bylo řečeno, dle pravidel OntoUML 2.0 je v pořádku, aby byly podtypem a v návrhu je povoleno, aby měl Relator Role či Phase.

```
:Marriage rdf:type gufo:Relator.
```

Role

Reprezentace stereotypu Role je stejný jako v dokumentaci i Visual Paradigm, ale jsou upravena pravidla, stejně jako u stereotypu Phase. Ve Visual Paradigm totiž není povoleno, aby Role byla podtypem stereotypů Relator, Mode, Quality, Quantity a Collective. Dle OntoUML 2.0 je v pořádku, aby Role byla podtypem těchto stereotypů, a proto je tak v rámci návrhu učiněno. Reprezentace jinak zůstává stejná:

```
:Student rdf:type gufo:Role;
    rdfs:subClassOf :Person.
```

RoleMixin

Jak již bylo několikrát popsáno, ve Visual Paradigm není povoleno, aby Role byla podtypem stereotypů Relator, Mode, Quality, Quantity a Collective. Tato chyba je ale zobrazena až při vytvoření RoleMixinu, mající pod sebou alespoň jednu z takovýchto Role. O chybu se ale nejedná, dle pravidel OntoUML 2.0 je zcela v pořádku, aby Role byla podtypem těchto stereotypů, a proto je tomu tak i v návrhu. RoleMixin je zároveň dalším stereotypem, který je ve Visual Paradigm často chybně označen za funkční celek, tato informace je tedy odebrána:

```
:Customer rdf:type gufo:RoleMixin.

:PersonalCustomer rdf:type gufo:Role.
:CorporateCustomer rdf:type gufo:Role.
:PersonalCustomer rdfs:subClassOf :Customer.
:CorporateCustomer rdfs:subClassOf :Customer.
```

SubKind

SubKind je reprezentován beze změny, stejně jako v dokumentaci a Visual Paradigm:

```
:Man rdf:type gufo:SubKind;
    rdfs:subClassOf :Person.
:Woman rdf:type gufo:SubKind;
    rdfs:subClassOf :Person.
```

Characterization

Reprezentace Characterization je dle gUFO dokumentace vyjádřena přes `gufo:inheresIn`. Oproti dokumentaci a Visual Paradigm je změněna reprezentace Quality a Mode. Ve Visual Paradigm jsou také často chybně reprezentovány multiplicity, jejich oprava je více popsána přímo v části zabývající se multiplicitami. Jinak je reprezentace stejná:

```
:MoonsMass rdf:type gufo:Quality.
:Moon rdf:type gufo:Kind.
:MoonsMass gufo:inheresIn :Moon.
```

ComponentOf

Reprezentace ComponentOf je od Visual Paradigm velmi odlišná, jelikož v exportu nástroje informace o vazbě ComponentOf zcela chybí. ComponentOf se často vyskytuje u funkčního celku, jehož zápis je ve Visual Paradigm v rozporu s dokumentací gUFO. Dle dokumentace se funkční celek zapisuje jako `rdf:type`, nikoliv jako `rdfs:subClassOf`. Ukázka chybného exportu z Visual Paradigm:

```
:Address rdf:type gufo:Kind;
    rdfs:subClassOf gufo:FunctionalComplex.
```

Informace o funkčním celku je navíc často umístěna i tam, kde se o funkční celek nejedná. To se týká všech stereotypů, které jsou spojeny generalizací a jsou nadtypem, tedy Kind, Category, Mixin, PhaseMixin a RoleMixin. V návrhu je tato informace odebrána. Pomocí `gufo:FunctionalComplex` by měly být označeny všechny entity ze kterých se funkční celek skládá. Části jsou s celkem spojeny přes `gufo:isComponentOf`. V návrhu je zároveň přidána reprezentace pro kompozici a agregaci:

```
:Address rdf:type gufo:Kind, gufo:FunctionalComplex,
                  gufo:Aggregate.
:Country rdf:type gufo:Kind, gufo:FunctionalComplex;
    gufo:isComponentOf :Address.
```

Containment

Stereotyp Containment nebyl přítomen v dokumentaci gUFO a tedy ani ve Visual Paradigm. Jelikož se ale jedná o běžně využívaný stereotyp OntoUML, je jeho reprezentace doplněna:

```
:Barrel rdf:type gufo:Kind.
:Wine rdf:type gufo:Quantity;
    gufo:Containment :Barrel.
```

Derivation

Ani jeden z nástrojů neumožňuje vytvářet Derivation vazbu. Jedná se totiž o poměrně nestandartní vazbu existující mezi Relatorem a Material vazbou. V OntoUML je to tedy jediná vazba, která neexistuje mezi dvěma entitami, ale mezi entitou a vazbou. Vazba není zmíněna v gUFO dokumentaci. Derivation není nutně potřeba, jelikož je vztah vyjádřen přes vlastnost Material vazby nazvanou isDerivedFrom, ale v návrhu je Derivation zahrnuta pro případnou budoucí implementaci vazby do nástroje:

```
:marriageInvolves rdf:type gufo:Derivation;
    rdfs:domain :Marriage;
    rdfs:range :marriedWith.
```

DomainFormal

Formal vazba je v OpenPonku nazvaná DomainFormal, v gUFO ComparativeRelationshipType. V návrhu je reprezentována stejně jako v dokumentaci a Visual Paradigm:

```
:taller_than rdf:type gufo:ComparativeRelationshipType;
    rdfs:domain :Person;
    rdfs:range :Person.
```

Material

Export ve Visual Paradigm je odlišný od dokumentace, jelikož v exportu nástroje u Material vazby chybí `gufo:isDerivedFrom`. Tato informace je v návrhu doplněna a Material vazba je tedy reprezentována dle dokumentace:

```
:marriedWith rdf:type owl:ObjectProperty,
             gufo:MaterialRelationshipType;
    rdfs:domain :Person;
    rdfs:range :Person;
    gufo:isDerivedFrom :Marriage.
```

Mediation

Reprezentace Mediation je odlišná od dokumentace a tedy i od Visual Paradigm. Jelikož je v OntoUML 2.0 Relator samostatným stereotypem, i Mediation vazba se tak stává samostatným stereotypem, a nebude tak již vyjádřena pomocí `rdfs:subPropertyOf` jak je uvedeno v dokumentaci:

```
:marriageInvolves rdf:type gufo:mediates;
    rdfs:domain :Marriage;
    rdfs:range :Person.
```

MemberOf

Navrhovaná reprezentace je odlišná, jelikož stejně jako u ComponentOf, v exportu Visual Paradigm zcela chybí `gufo:isCollectionMemberOf`. V návrhu je zároveň přidána reprezentace pro kompozici a agregaci:

```
:SchoolClass rdf:type gufo:VariableCollection, gufo:Composite.
:Student rdf:type gufo:Role;
    gufo:isCollectionMemberOf :SchoolClass.
```

SubCollectionOf

Stejně jako ComponentOf a MemberOf, v exportu Visual Paradigm chybí i `gufo:isSubCollectionOf`. V návrhu je také přidána reprezentace pro kompozici a agregaci:

```
:Population rdf:type gufo:VariableCollection, gufo:Composite.
:LocalPopulation rdf:type gufo:VariableCollection;
    gufo:isSubCollectionOf :Population.
```

SubQuantityOf

Reprezentace SubQuantityOf je stejný jako v dokumentaci a Visual Paradigm. V návrhu je pouze přidána reprezentace pro kompozici a agregaci a je opravena reprezentace Quantity:

```
:Beer rdf:type gufo:Quantity, gufo:Composite.
:Alcohol rdf:type gufo:Quantity;
    gufo:SubQuantityOf :Beer.
```

Package Import, Package, Model, Profile

V nástroji OpenPonk je možné v rámci OntoUML modelu vytvořit Package Import, Package, Model a Profile. Ty se ale v samotném OntoUML nevyužívají, jsou v nástroji pouze pozůstatkem UML, ze kterého OntoUML vychází. Pro tyto entity tedy nebude navržen export.

Note

OpenPonk i Visual Paradigm umožňují přidat Note. Jedná se o viditelnou textovou poznámku umístěnou v modelu, se kterou lze pohybovat. Narozdíl od komentáře je vidět neustále. Při exportování do gUFO přes Visual Paradigm se ale nijak nezachytí informace o Note, je zcela vynechána. Součástí návrhu je proto i reprezentace Note:

```
openponk:Note rdf:type owl:Class.

:countryNote rdf:type openponk:Note;
   rdfs:label "Country is modelled in the second model.".
```

Boundary

Boundary slouží k označení zón v modelu, lze tak seskupit entity, které spolu logicky souvisí. Jedná se spíše o záležitost UML, ale lze ji využít pro OntoUML. Proto je navržena reprezentace:

```
openponk:Boundary rdf:type owl:Class.

:PersonBoundary rdf:type openponk:Boundary;
   openponk:contains :Person, :Male, :Female.
```

Komentář

Pro potřeby technologií RDF jsou všechny názvy entit, vazeb a atributů pozměněny, nejčastěji jsou bez mezer. K zachycení skutečného názvu v modelu slouží `rdfs:label`. K jednotlivým entitám a vazbám je ale také možné přidávat komentáře, obsahující další informace. Reprezentace komentáře v návrhu je stejná, jako v nástroji Visual Paradigm:

```
:Vehicle rdf:type gufo:Kind.
    rdfs:comment "Or should it be called a Car?";
```

Agregace

Vztahy celek část nejsou v dokumentaci gUFO příliš popsány a nejsou zde téměř žádné příklady. V nástroji Visual Paradigm je jejich export také velmi neúplný. Zápis funkčního celku je v rozporu s dokumentací gUFO a informace o funkčním celku je často umístěna i na místa, kde se o funkční celek nejedná. Visual Paradigm navíc téměř nedokáže využít příslušné stereotypy pro vazby celek část, tyto informace často zcela chybí. Návrh tak obsahuje mnoho úprav.

Dokumentace gUFO ani Visual Paradigm při generování do gUFO neuvádí žádné informace o tom, zda je celek kompozice či agregace. Proto je v návrhu tato informace přidána přes `gufo:Composite` a `gufo:Aggregate`, připojující se k entitě celku.

Pro vztahy celek část se jedná buď o jednu entitu celek s více entitami část, nebo o jednu entitu celek s jednou entitou část. Pro případ více entit částí se jedná o funkční celek s vazbami ComponentOf. Všechny entity ze kterých se funkční celek skládá by měly být označeny pomocí `gufo:FunctionalComplex`. Části jsou s celkem spojeny přes `gufo:isComponentOf`:

```
:Address rdf:type gufo:Kind, gufo:FunctionalComplex,
    gufo:Aggregate;
:Country rdf:type gufo:Kind, gufo:FunctionalComplex;
    gufo:isComponentOf :Address;
```

Pro vztah jedné entity celku a jedné entity části se může jednat o více stereotypů vazeb, a to MemberOf, SubCollectionOf a SubQuantityOf. Všechny mají velmi podobnou reprezentaci, pouze jsou nahrazeny celek a část, tedy `gufo:VariableCollection` a `gufo:isCollectionMemberOf`, za příslušné varianty:

```
:SchoolClass rdf:type gufo:VariableCollection, gufo:Composite.
:Student rdf:type gufo:Role;
    gufo:isCollectionMemberOf :SchoolClass.
```

Multiplicita

Multiplicita není v dokumentaci gUFO popsána, jelikož se, stejně jako u atributu, jedná o vlastnosti definované v OWL [9]. Reprezentace multiplicity v návrhu se nemění. Pro minimální kardinalitu nula není nutné informaci zapisovat, pokud není řečeno jinak, předpokládá se minimální kardinalita nula. Obdobně s maximální kardinalitou nekonečno, značené hvězdičkou. Pro stanovení přesné hodnoty slouží `owl:qualifiedCardinality`.

Visual Paradigm multiplicity často exportoval nesprávně. U některých typů vazeb, například Characterization, byly multiplicity vždy uvedeny chybně, tedy jinak než byly v modelu. V jednom směru byla vždy uvedena multiplicita `1..*`, i přesto, že v modelu byla definována multiplicita `1..1`. U jiných stereotypů vazeb, obvykle vztah celek část, byla informace o multiplicitách zcela

vynechána. Jelikož se jedná o důležitou informaci, která by neměla být opomíjena, multiplicity by měly být uvedeny u všech vazeb kromě generalizace, dle dokumentace OWL:

```
:Product rdfs:subClassOf [
  rdf:type owl:Restriction;
  owl:onProperty [ owl:inverseOf gufo:mediates ];
  owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger;
  owl:onClass :Order
].
:DateOfBirth rdfs:subClassOf [
  rdf:type owl:Restriction;
  owl:onProperty gufo:inheresIn;
  owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger;
  owl:onClass :Person
].
```

Generalizační sety

Nástroj Visual Paradigm často nevyužívá možnosti vyjádřit disjoint a complete najednou přes `owl:disjointUnionOf` a také nevyužívá sdružení do `gufo:partitions`. V dokumentaci gUFO jsou uvedeny příklady, kdy jsou partitions využity, ale disjoint není vyjádřen vzhledem k partitions. Je vyjádřen vzhledem jedné entity k druhé, nikoliv už ale vzhledem druhé entity k první. Navržená reprezentace je tak odlišná od Visual Paradigm i dokumentace:

```
:PersonSex gufo:partitions :Person.
:PersonSex owl:equivalentClass [
  rdf:type owl:Class;
  owl:disjointUnionOf (:Man :Woman)
].
```

Is Derived

Is Derived je označení odvozené vazby. Využívá se u vazeb Material, odvozených od Relatoru. V exportu ve Visual Paradigm je zápis odlišný od dokumentace, jelikož u Material vazby `gufo:isDerivedFrom` zcela chybí. Proto je v návrhu tato informace doplněna:

```
:marriedWith rdf:type owl:ObjectProperty,
             gufo:MaterialRelationshipType;
    rdfs:domain :Person;
    rdfs:range :Person;
    gufo:isDerivedFrom :Marriage.
```

Is Abstract

Is Abstract je označení abstraktní entity. Využívá se například pro stereotyp Category. V dokumentaci je definován `gufo:AbstractIndividualType`, ale není ukázán na žádném příkladu. Visual Paradigm tuto informaci také neexportuje. Proto je navržena reprezentace:

```
:LivingThing rdf:type gufo:Category,
                      gufo:AbstractIndividualType.
```

Visibility

Visibility je další funkcí, která se využívá v UML, ale nikoliv v OntoUML. I přesto, že OpenPonk tedy umožňuje nastavit visibility jednotlivých atributů, nebude navržena reprezentace.

Další vlastnosti vazeb

Na koncích vazeb mohou být navíc označení Is Navigable, Is Static, Is Read Only, Is Derived, Is Ordered a Is Unique. V dokumentaci gUFO nejsou tyto možnosti vůbec zmíněny a Visual Paradigm žádné z těchto informací neexportuje. Návrh reprezentace je ukázán na příkladu pro `gufo:isNavigable`. Stejným způsobem by byly reprezentovány i ostatní značení, tedy `gufo:isStatic`, `gufo:isReadOnly`, `gufo:isDerived`, `gufo:isOrdered` a `gufo:isUnique`:

```
:livesAt rdf:type owl:ObjectProperty;
    rdfs:domain :Citizen;
    rdfs:range :HomeAddress;
    rdfs:label "lives_at";
    gufo:isNavigable "true"^^xsd:boolean;
    rdf:type gufo:MaterialRelationshipType;
    gufo:isDerivedFrom :Residence.
```

Default Value

Pro atributy je možné nastavit výchozí hodnotu, Default value. To se ale využívá v UML, jakožto implementační záležitost, nikoliv v OntoUML, a proto není navržena reprezentace.

5.2.3 Shrnutí návrhu exportu

V návrhu exportu do gUFO pro nástroj OpenPonk bylo provedeno mnoho změn oproti dokumentaci a již existujícímu exportu do gUFO v nástroji Visual Paradigm. Většina změn se týkala aktualizace na OntoUML 2.0, opravy chyb a přidání chybějících stereotypů a funkcí. Tato podsekce slouží jako přehledný seznam všech navržených změn. Jedná se o návrh bez samotné implementace.

Změny vůči gUFO dokumentaci [4]:

- Collective je samostatný stereotyp a může mít jako podtyp Role a Phase
- Mode je samostatný stereotyp a může mít jako podtyp Role a Phase
- Quality je samostatný stereotyp a může mít jako podtyp Role a Phase
- Quantity je samostatný stereotyp a může mít jako podtyp Role a Phase
- Relator je samostatný stereotyp a může mít jako podtyp Role a Phase
- Phase může mít nadtyp Relator, Quality, Mode, Collective a Quantity
- Role může mít nadtyp Relator, Quality, Mode, Collective a Quantity
- Přidán stereotyp Containment
- Přidán stereotyp Derivation
- Mediation je samostatný stereotyp
- Přidána kompozice a agregace
- Generalizační sety reprezentovány přes partitions
- Přidány vlastnosti pro konce vazeb

Změny vůči Visual Paradigm (zahrnují i všechny změny vůči gUFO dokumentaci):

- Opraven funkční celek
- Využití stereotypů MemberOf, ComponentOf, SubCollectionOf
- Přidáno isDerived pro stereotyp Material
- Opraveny problémy s multiplicitami
- Přidána reprezentace pro Note
- Přidána reprezentace pro Boundary
- Přidána vlastnost Is Abstract

Kapitola 6

Zhodnocení

V této kapitole jsou popsány výsledky a přínosy práce, rozdělené do sekcí.

6.1 Možnosti OntoUML v RDF

Zaměřením modelovacího jazyka OntoUML je abstrakce reality dle určité konceptualizace a modely jsou využívány k přehlednému zobrazení vztahů v reálném světě, nad kterým se dá poté dále pracovat. Jazyk UML, ze kterého OntoUML vychází, již nabízí možnosti vytvoření tříd, vztahů, atributů a podtypů, ale pro úplnou a podrobnou reprezentaci byly přidány ještě další vlastnosti, převážně stereotypy. Ty rozšiřují své nositele o další vlastnosti, ale zároveň přidávají pravidla, která s nimi souvisí a musí být při modelování dodržována. Tato pravidla a z nich související omezení vedou k chybám v modelu, pokud nejsou splněna. V OntoUML se lze setkat také s anti-patterny, což jsou struktury v modelu, které nejsou nutně chybné, ale jejich výskyt obvykle vede ke vzniku chyby či nejasnosti. Je tedy nutné zaručit kvalitu modelů a kontrolovat, zda se v nich nenachází chyby či anti-patterny.

Pro práci s OntoUML modely je možné využít technologie RDF. Samotné RDF, Resource Description Framework, je jazyk pro popis zdrojů a lze ho již v základní podobě využít na reprezentaci modelu. Vznikne ale minimalistický model, kde nebudou obsaženy všechny informace, například stereotypy nebo multiplicita. Aby se s daty mohlo efektivně pracovat, byly využity ontologie a slovníky, kterými jsou RDF Schema a OWL. Pro ontologii UFO, na níž je OntoUML založené, existuje její OWL reprezentace nazvaná gUFO. Vlastnosti a stereotypy využívané v modelech jsou v gUFO definovány, a je tak možné transformovat OntoUML modely do RDF formátu i s informacemi o stereotypech. Součástí práce je ukázka reprezentace všech stereotypů gUFO dle dokumentace [4].

RDF udává, jak informace zapsat, ale nemá vlastní definovanou syntax a lze ho vyjádřit pomocí různých formátů a jazyků, jedním z nich je strojově zpracovatelný jazyk XML. Stroje mohou takto zadané informace jednoduše číst a dál s nimi pracovat, jak je ukázáno v práci. Zápis modelů v jazyce RDF, který je standardem daným mezinárodním konsorciem W3C, zajišťuje lepší interoperabilitu dat než reprezentace pomocí diagramu či vlastní formát konkrétního modelovacího nástroje.

6.2 Měření kvality modelů

Cílem modelování je pochopení domény reálného světa, pro které je nutná vysoká kvalita modelu. I přes její důležitost ale není kvalita modelu jasně definovaná a jedná se o rychle se vyvíjející koncept. Měřením kvality modelu se zabývá studie [2]. Soustředí se na experiment zaměřený na porovnání vnímané a měřené kvality ER modelů. Zvolenými kritérii byla minimalita, jasnost, jednoduchost a expresivita. Pro měření kvality OntoUML modelů v práci se využila tato kritéria a vlastní kritéria, kterými je počet chyb a anti-patternů.

Pro měření kvality v práci byly vytvořeny čtyři modely. První z nich, nazvaný Posilovna, nepopisuje složitou doménu, ale obsahuje velké množství různých stereotypů a lze tak na něm mnoho věcí ukázat a vyzkoušet. Další dva modely, nazvané Přehled invalidity a Příčiny invalidity, reprezentují dvě podobné datové sady týkající se informací o invalidních osobách. Byly zvoleny tak, aby se jednalo o sady malého rozsahu a stejné domény, ale s odlišným zaměřením. Každá datové sada také pochází z jiného zdroje. Doména invalidity byla zvolena z praktického důvodu využití něčeho, co se opravdu sleduje a měří, na doméně, která bude snadno představitelná a pochopitelná. V běžně využívaných datových sadách v praxi jsou data anonymizována a agregována. Nad takovými sadami by bylo dotazování odlišné, a nebylo by tedy možné všechny používané technologie zcela využít vzhledem k rozsahu práce. Modely tedy nejsou přesnou reprezentací datových sad. Pro export do gUFO byl využit existující plugin v rámci nástroje Visual Paradigm.

Na kontroly byl využit jazyk SPARQL, pomocí kterého se lze nad modelem i nad jeho daty dotazovat a vyhledávat v nich. Byla takto změřena kvalita vytvořených čtyř modelů dle zmíněné studie [2]. Přesný postup měření byl ale ve studii uveden jen pro jedno kritérium ze čtyř. Ostatní tedy byla posuzována dle vlastních pravidel navržených tak, aby se co nejvíce přiblížily zadání a přitom respektovaly podstatu OntoUML. Všechny vytvořené modely tato kritéria splnily. Následovalo měření počtu chyb a antipatternů, na které byl opět využit dotazovací jazyk SPARQL. Jelikož ve vytvořených modelech žádné tyto nedostatky nebyly, pro lepší ukázku kontroly byly do jednoho z modelů upraven přidány chyby a anti-pattern. Obě chyby i anti-pattern byly pomocí dotazů úspěšně detekovány.

6.3 Další využití RDF pro modely a instance

Modely Přehled invalidity a Příčiny invalidity byly propojeny přes technologie RDF do jednoho. Pro tento spojený model byla vytvořena data, která jsou instancemi OntoUML modelu v gUFO. Bylo tak možné se nad nimi jazykem SPARQL dotazovat a vyhledávat v nich, a kvalita byla změřena i pro tento model.

Pomocí validačního jazyka SHACL byla data modelu validována, kontrolovaly se správnosti datových typů a zda jsou data kompletní. Zároveň bylo zjištěno, že vyjádření pravidel pro vztah nadtypu a podtypu je v jazyce SHACL obtížné a detekce chyb a anti-patternů by byla velmi složitá. Tento jazyk tak není vhodným pro tento druh kontroly, a proto byl využit SPARQL.

6.4 Nástrojová podpora

V průběhu vytváření práce bylo nalezeno mnoho nedostatků v exportu do gUFO vytvořeným nástrojem Visual Paradigm i v samotné dokumentaci gUFO. Nástroj mnoho informací exportuje chybně, či je úplně vynechává a v dokumentaci chybí některé důležité informace. Zcela chybí dva stereotypy a dokumentace není aktuální dle ustálené verze OntoUML 2.0. Aby bylo do budoucna možné všechny koncepty OntoUML lépe využít, byl v práci navrhnut export do gUFO, ve kterém jsou popsány všechny nedostatky a chyby a přidány chybějící stereotypy. Jedná se o teoretický návrh s analýzou potřeb a ukázkami výstupů, bez samotné implementace.

V návrhu jsou popsány všechny nalezené problémy a jsou pro ně uvedena řešení. Návrh je určen pro nástroj OpenPonk, což je modelovací nástroj vyvíjený na FIT ČVUT v Praze. Kapitola se zabývá i srovnáním nástrojů OpenPonk a Visual Paradigm. Navržený export byl porovnán s exportem v nástroji Visual Paradigm a vychází z oficiální dokumentace gUFO [4]. Vztahy které se ve Visual Paradigm neexportují správně byly popsány, společně s návrhem správné varianty. Návrh zároveň obsahuje reprezentace několika funkcí, kterými OpenPonk disponuje navíc, například možnost vkládat poznámky. Součástí je doplnění stereotypů Containment a Derivation, které v oficiální dokumentaci chybí, a úprava reprezentace některých stereotypů dle verze OntoUML 2.0. Těmi je obvykle nezávislost některého ze stereotypů na jiném a možnost nových nadtypů a podtypů.

Závěr

Cílem práce bylo zanalyzovat a navrhnout využití technologií RDF pro zefektivnění práce s OntoUML modely. Nejdříve byly představeny modelovací jazyky, kde bylo popsáno využití OntoUML oproti jiným modelovacím jazykům. Poté byl analyzován RDF a jazyky určené pro práci s ním. Ty byly popsány i s ukázkami reprezentace.

Poté byly prozkoumány možnosti měření kvality a byla vybrána kritéria dle studie zabývající se měřením kvality ER modelů. Tato kritéria byla upravena pro potřeby a koncepty OntoUML. Jako další kritéria pro měření kvality byl zvolen počet chyb a anti-patternů v modelu.

V rámci práce byly vytvořeny čtyři OntoUML modely, nad kterými byla prováděna validace a dotazování. První model byl vytvořen tak, aby obsahoval mnoho různých stereotypů a sloužil převážně jako ukázkový. Další dva byly namodelovány dle existujících datových sad. Všechny byly poté převedeny do RDF s využitím gUFO. Tento zápis modelů zajišťuje lepší interoperabilitu než reprezentace pomocí diagramu či vlastní formát konkrétního modelovacího nástroje. Poslední model byl vytvořen spojením druhého a třetího modelu pomocí technologií RDF.

Pro spojený model byla vytvořena data, která jsou instancemi OntoUML modelu v gUFO, a lze se nad nimi dotazovat a vyhledávat v nich. Pomocí validačního jazyka SHACL byla tato data validována, zkontrolovaly se správnosti datových typů a zda jsou data kompletní. Dotazovací jazyk SPARQL byl, kromě vyhledávání v datech, využit převážně na měření kvality a na detekování chyb a anti-patternů v modelech.

Pro převod do gUFO byl využit existující plugin v rámci nástroje Visual Paradigm. V průběhu vytváření práce ale bylo v exportu tohoto nástroje nalezeno mnoho nedostatků, některé informace byly exportovány chybně, či úplně vynechány. Také bylo zjištěno, že v dokumentaci gUFO chybí některé důležité informace, zcela chybí dva stereotypy a dokumentace není aktuální dle ustálené verze OntoUML 2.0. Aby bylo do budoucna možné všechny koncepty OntoUML lépe využít, byl v práci navrhnut export do gUFO, bez samotné

implementace, ve kterém jsou popsány všechny nedostatky a chyby a přidány chybějící stereotypy. Návrh je určen pro nástroj OpenPonk, což je modelovací nástroj vyvinutý na FIT ČVUT v Praze.

Všechny cíle práce byly úspěšně splněny, a navíc byly detailně popsány nedostatky současné verze gUFO a navrženy opravy a aktualizace, které by při implementaci dále rozšířily možnosti efektivního využití technologií RDF pro OntoUML modely.

Literatura

[1] Guizzardi, G.: *Ontological Foundations for Structural Conceptual Models*. Dizertační práce, 01 2005, naposledy navštíveno: 2022-10-06. Dostupné z: https://www.researchgate.net/publication/215697579_Ontological_Foundations_for_Structural_Conceptual_Models

[2] Cherfi, S.; Akoka, J.; Wattiau, I.: Perceived vs. Measured Quality of Conceptual Schemas: An Experimental Comparison. 01 2007, s. 185–190, naposledy navštíveno: 2022-11-27. Dostupné z: https://www.researchgate.net/publication/221268516_Perceived_vs_Measured_Quality_of_Conceptual_Schemas_An_Experimental_Comparison

[3] Manola, F.; Miller, E.: RDF Primer. Feb 2004, naposledy navštíveno: 2022-10-11. Dostupné z: https://www.w3.org/TR/rdf-primer/

[4] Almeida, J. P.; Falbo, R.; Guizzardi, G.; aj.: gUFO: A Lightweight Implementation of the Unified Foundational Ontology (UFO). 2019, naposledy navštíveno: 2022-10-18. Dostupné z: https://nemo-ufes.github.io/gufo/

[5] Object Management Group: BPMN specification - business process model and notation. Naposledy navštíveno: 2022-10-05. Dostupné z: https://www.omg.org/spec/BPMN/2.0.2/About-BPMN

[6] Jednoduchý návod k UML diagramům a modelování databází. Sep 2019, naposledy navštíveno: 2022-10-05. Dostupné z: https://www.microsoft.com/cs-cz/microsoft-365/business-insights-ideas/resources/guide-to-uml-diagramming-and-database-modeling

[7] Bělohoubek, M.: *Extending OntoUML Modelling Capabilities on the OpenPonk Platform*. Diplomová práce, Czech Technical Univer-

sity in Prague, 2021, naposledy navštíveno: 2022-10-07. Dostupné z: https://dspace.cvut.cz/bitstream/handle/10467/94590/F8-DP-2021-Belohoubek-Marek-Thesis.pdf?sequence=-1&isAllowed=y

[8] Brickley, D.: RDF Schema 1.1. Feb 2014, naposledy navštíveno: 2022-10-11. Dostupné z: https://www.w3.org/TR/rdf-schema/

[9] Hitzler, P.; Krötzsch, M.; Parsia, B.; aj.: OWL 2 Web Ontology Language Primer (Second Edition). Dec 2012, naposledy navštíveno: 2022-10-13. Dostupné z: https://www.w3.org/TR/owl2-primer/

[10] Guizzardi, G.; Prince Sales, T.: *Anti-patterns in Ontology-driven Conceptual Modeling: The Case of Role Modeling in OntoUML*. 08 2016, ISBN 978-1-61499-675-0, s. 161–187, doi: 10.3233/978-1-61499-676-7-161, naposledy navštíveno: 2022-11-03. Dostupné z: https://www.researchgate.net/publication/305808778_Anti-patterns_in_Ontology-driven_Conceptual_Modeling_The_Case_of_Role_Modeling_in_OntoUML

[11] Guerson, J.; Brasileiro, F.; Sales, T. P.: Menthortools/Menthor-Editor: Menthor editor. Mar 2021, naposledy navštíveno: 2022-11-03. Dostupné z: https://github.com/MenthorTools/menthor-editor

[12] Fonseca, C.; Sales, T. P.: ONTOUML/ONTOUML-VP-Plugin: A plugin for visual paradigm to add support for ONTOUML modeling and Model Intelligence Services. Dec 2021, naposledy navštíveno: 2022-11-16. Dostupné z: https://github.com/OntoUML/ontouml-vp-plugin

[13] Beckett, D.; Berners-Lee, T.; Prud’hommeaux, E.; aj.: RDF 1.1 Turtle. Feb 2014, naposledy navštíveno: 2022-10-27. Dostupné z: https://www.w3.org/TR/turtle/

[14] Prud’hommeaux, E.; Seaborne, A.: SPARQL Query Language for RDF. Jan 2008, naposledy navštíveno: 2022-10-19. Dostupné z: https://www.w3.org/TR/rdf-sparql-query/

[15] Nečaský, M.: Série Znalostní Grafy: Díl 3: SPARQL. Jan 2021, naposledy navštíveno: 2022-10-24. Dostupné z: https://data.gov.cz/%C4%8Dl%C3%A1nky/znalostn%C3%AD-grafy-03-sparql

[16] Knublauch, H.; Kontokostas, D.: Shapes constraint language (SHACL). Jul 2017, naposledy navštíveno: 2022-10-28. Dostupné z: https://www.w3.org/TR/shacl/

[17] Labra Gayo, J. E.; Prud'hommeaux, E.; Boneva, I.; aj.: *Validating RDF Data*, *Synthesis Lectures on the Semantic Web: Theory and Technology*, ročník 7. Morgan & Claypool Publishers LLC, Sep 2017, doi:

10.2200/s00786ed1v01y201707wbe016, naposledy navštíveno: 2022-10-30. Dostupné z: https://doi.org/10.2200/s00786ed1v01y201707wbe016

[18] W3C: SHACL-ShEx-comparison. Mar 2017, naposledy navštíveno: 2022-10-31. Dostupné z: https://www.w3.org/2014/data-shapes/wiki/SHACL-ShEx-Comparison

[19] RML.io: RML introduction. Naposledy navštíveno: 2022-10-31. Dostupné z: https://rml.io/docs/rml/introduction/

[20] Nelson, H.; Poels, G.; Genero, M.; aj.: A conceptual modeling quality framework. *Software Quality Journal*, ročník 20, 03 2012: s. 201–228, doi:10.1007/s11219-011-9136-9, naposledy navštíveno: 2022-11-27. Dostupné z: https://www.researchgate.net/publication/220636080_A_conceptual_modeling_quality_framework

[21] Eurostat: Population by sex, age, disability status and educational attainment level - Products Datasets - Eurostat. May 2015, naposledy navštíveno: 2022-11-18. Dostupné z: https://ec.europa.eu/eurostat/web/products-datasets/-/hlth_dpeh020

[22] Česká správa sociálního zabezpečení: Dokumentace Datové Sady Nejčastější Příčiny Vzniku invalidity. Naposledy navštíveno: 2022-11-18. Dostupné z: https://data.cssz.cz/documentation/nejcastejsi-priciny-vzniku-invalidity

[23] FIT ČVUT v Praze: OpenPonk modeling platform. Naposledy navštíveno: 2022-11-17. Dostupné z: https://openponk.org/

[24] Visual Paradigm: Visual Paradigm. Naposledy navštíveno: 2022-11-17. Dostupné z: https://www.visual-paradigm.com/

[25] Visual Paradigm: Download Visual Paradigm Community Edition. Naposledy navštíveno: 2022-12-21. Dostupné z: https://www.visual-paradigm.com/download/community.jsp

[26] Sparx Systems Ltd und SparxSystems Software GmbH: Enterprise architect Major releases. Naposledy navštíveno: 2022-11-18. Dostupné z: https://www.sparxsystems.eu/newedition

[27] Suchánek, M.: OntoUML Specification. 2018, naposledy navštíveno: 2022-12-03. Dostupné z: https://ontouml.readthedocs.io/en/latest/index.html

[28] Guizzardi, G.; Fonseca, C.; Benevides, A.; aj.: Endurant Types in Ontology-Driven Conceptual Modeling: Towards OntoUML 2.0. 09 2018, ISBN 978-3-030-00846-8, doi:10.1007/978-3-030-00847-5_12, naposledy navštíveno: 2022-12-03. Dostupné z: https://www.researchgate.net/

publication/325995419_Endurant_Types_in_Ontology-Driven_
Conceptual_Modeling_Towards_OntoUML_20

Příloha A

Seznam použitých zkratek

BPMN Business Process Model and Notation

ER Model Entity-Relationship Model

gUFO Gentle Unified Foundational Ontology

IRI Internationalized Resource Identifier

JSON JavaScript Object Notation

OWL Ontology Web Language

RDF Resource Description Framework

RDFS Resource Description Framework Schema

RML RDF Mapping language

SHACL Shapes Constraint Language

ShEx Shape Expressions

SPARQL SPARQL Protocol and RDF Query Language

SQL Structured Query Language

UFO Unified Foundational Ontology

UML Unified Modeling Language

URI Uniform Resource Identifier

XML Extensible Markup Language

Příloha B

Obsah přiloženého CD

- readme.txt .... stručný popis obsahu média
- Pictures .... složka obsahující modely a další obrázky
- Export_and_data .... složka obsahující export do gUFO a data
- machate4-assignment.pdf .... text zadání práce ve formátu pdf
- DP_Machacova_Tereza_2023.pdf .... práce ve formátu pdf
- DP_Machacova_Tereza_2023.tex zdrojová forma práce ve formátu LaTeX