FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Ontologické modelování dat pro projekt Datového inkubátoru
Student:	Šimon Matouš
Vedoucí:	doc. Ing. Robert Pergl, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Práce navazuje na předchozí úspěšné projekty a dále rozvíjí jak obsahovou, tak metodickou stránku projektu Datového inkubátoru, jehož ambicí je propojování heterogenních otevřených datových sad a zajištění jejich sémantické interoperability. Cílem práce je tedy ontologická analýza klíčových domén a jejich propojení s datovými sadami tak, aby byla umožněna jejich sémantická interoperabilita.

1. Seznamte se s projektem Datového inkubátoru, problematikou sémantické interoperability, Unified Foundational Ontology, jazykem OntoUML a nástrojem OpenPonk.
2. Ve spolupráci s vedoucím vyberte několik klíčových domén a souvisejících datových sad.
3. Vytvořte ontologické konceptuální modely těchto domén v jazyce OntoUML s použitím nástroje OpenPonk.
4. Propojte ontologické konceptuální modely s datovými sadami a vytvořte pravidla pro jejich mapování.
5. Implementujte validátor mapovacích pravidel rozšířením existujícího parseru.
6. Zdokumentujte své řešení a přínos pro zlepšení sémantické interoperability dat v Datovém inkubátoru.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 6. února 2023 v Praze.

Bakalářská práce

ONTOLOGICKÉ MODELOVÁNÍ DAT PRO PROJEKT DATOVÉHO INKUBÁTORU

Šimon Matouš

Fakulta informačních technologií
Katedra teoretické informatiky
Vedoucí: doc. Ing. Robert Pergl, Ph.D.
29. června 2023

Odkaz na tuto práci: Matouš Šimon. *Ontologické modelování dat pro projekt Datového inkubátoru.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Rád bych poděkoval doc. Ing. Robertu Perglovi, Ph.D. za cenné rady, věcné připomínky a vstřícnost při konzultacích k mé bakalářské práci. Mé poděkování patří též Bc. Tereze Machačové za užitečné podněty a komentáře.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 29. června 2023 .....

Abstrakt

Tato bakalářská práce se zabývá ontologickou analýzou klíčových domén a jejich datových sad a navazující tvorbou ontologických konceptuálních modelů v jazyce OntoUML pomocí nástroje OpenPonk a dále jejich propojením s datovými sadami pomocí datových pravidel. Další částí práce je vývoj syntaktického analyzátoru pro automatickou kontrolu syntaxe datových pravidel. Syntaktický analyzátor je vyvíjen v programovacím jazyce JavaScript a rozšiřuje již existující parser, který byl vygenerován z gramatiky programem PEG.js. Celý obsah bakalářské práce je součástí projektu Nest Big Data Arena, který je realizován firmou Remmark, a. s.

Přínosem této práce je vytvoření ontologických konceptuálních modelů propojených s příslušnými datovými sadami, které umožňují sémantickou interoperabilitu nejenom mezi danými datovými sadami a jejich klíčovými doménami, ale i s již existujícími datovými sadami. Dále syntaktický analyzátor na kontrolu datových pravidel umožní členům modelovacího týmu zautomatizovat syntaktickou kontrolu mapovacích pravidel a tím omezí prostor pro lidské chyby a zrychlí proces kontroly.

Konceptuální modely byly přidány do Datové platformy, která je vyvíjená firmou Remmark, a. s. v rámci projektu Nest Big Data Arena. Syntaktický analyzátor je aktivně využíván členy modelovacího týmu.

Klíčová slova ontologické modelování, OntoUML, UFO, OpeNest, datový inkubátor, Remmark, syntaktický analyzátor, JavaScript

Abstract

This bachelor's thesis deals with the ontological analysis of key domains and their data sets, subsequent creation of ontological conceptual models using the modelling language OntoUML in the OpenPonk platform and connecting them together with the respective data sets using data rules. Next part of this work is the development of a syntax analyzerfor automatic checking of the syntax of data rules. The syntax analyzer is developed in the JavaScript programming language and extends an already existing parser that was generated from a grammar by the PEGjs program. The entire content of this bachelor's thesis is part of the Nest Big Data Arena project, which is implemented by Remmark, a. s.

The outcome of this work is the creation of ontological conceptual models connected to their relevant datasets, which enable semantic interoperability not only between the given datasets and their key domains, but also with already existing datasets. Furthermore, the syntax analyzer will allow members of the Remmark modeling team to automate the mapping rule checking, thereby reducing the room for human error and accelerating the checking process.

Conceptual models have been added to the Data Platform, which is developed by Remmark, a.s. pp. within the Nest Big Data Arena project. The syntax analyzer is being actively used by members of the modeling team.

Keywords ontological modelling, OntoUML, UFO, OpeNest, data incubator, Remmark, syntax analyzer, JavaScript

Seznam zkratek

UFO	Unified Foundational Ontology
BFO	Basic Formal Ontology
DOLCE	Descriptive Ontology for Linguistic and Cognitive Engineering
SUMO	Suggested Upper Merged Ontology
UML	Unified Modeling Language
OMG	Object Management Group
JSON	JavaScript Object Notation
CCMi	Centre for Conceptual Modelling and Implementation
BORM	Business Objects Relation Modelling
FSM	Finite State Machine
BPMN	Business Process Modeling and Notation
API	Application Programming Interface
SO ORP	Správní obvod obce s rozšířenou působností
NBDA	Nest Big Data Arena
MEA	Measurement
UI	User Interface
EXE	Executable

Kapitola 1
Úvod

V dnešní době všichni rozumí tomu, že shromažďování dat a práce s s nimi je nedílnou součástí téměř každého pracovního nebo vědeckého procesu. Data se ukládají z mnoha důvodů, ať už v běžných databázích webových stránek, nebo třeba v průběhu obsáhlých věděckých studií. Ve velkém množství případů vystačí struktura databáze vytvořená a upravovaná podle aktuální potřeby a bez velkého plánování do budoucnosti. Problém ale nastává, když potřebujeme zachytit nějakou rozsáhlou nebo odbornou oblast či více podobných oblastí do velkého detailu.

Tato bakalářská práce se zaměřuje právě na tyto oblasti. Probíhá pod projektem Nest Big Data Arena firmy Remmark a. s. a je součástí Datové platformy. Datová platforma obsahuje stovky různých datových sad, v nichž každá spadá do nějaké domény. Tyto datové sady jsou důkladně zdokumentovány a propojeny s ontologickými konceptuálními modely. Díky tomu datové sady umožňují jejich sémantickou interoperabilitu.

Sémantická interoperabilita datových sad je dosažena důkladnou ontologickou analýzou příslušných klíčových domén a studiem struktury již existujících datových sad a jejich modelů. Na analýzu navazuje tvorba ontologických modelů a jejich následné propojení s příslušnými datovými sadami pomocí mapovacích pravidel.

Dalším záměrem této bakalářské práce je zjednodušení samotného procesu tvorby ontologických konceptuálních modelů, konkrétně procesu kontroly správnosti mapovacích pravidel, které se používají k propojení datových sad a ontologických konceptuálních modelů. Toto bude dosaženo implementací syntaktického validátoru mapovacích pravidel, který bude určen členům modelovacího týmu.

Kapitola 2
Cíle práce

Prvním cílem této bakalářské práce je ontologická analýza několika klíčových domén a jejich propojení s datovými sadami tak, aby byla zajištěna jejich sémantická interoperabilita.

Navazujícím cílem je tvorba ontologických konceptuálních modelů podle předešlé ontologické analýzy klíčových domén. Tvorba těchto modelů proběhne v jazyce OntoUML s použitím nástroje OpenPonk.

Dalším cílem je propojení těchto ontologických konceptuálních modelů s příslušnými datovými sadami vytvořením mapovacích pravidel. Tyto dokončené konceptuální modely budou přínosem pro projekt Datové platformy firmy Remmark a. s. tím, že zpřístupní nové domény a jejich příslušné datové sady, u kterých je zajištěná sémantická interoperabilita.

Posledním cílem této bakalářské práce je implementace syntaktického validátoru pro kontrolu mapovacích pravidel pomocí rozšíření již existujícího parseru. Tento program je vytvořený pro členy modelovacího týmu firmy Remmark a. s. a umožní automatizaci části procesu kontroly korektnosti konceptuálních ontologických modelů, konkrétně mapovacích pravidel. Kromě snížení času potřebného na kontrolu těchto mapovacích pravidel také zmenší možnost proklouznutí chyby způsobené lidským faktorem do produkce.

Část I

Teoretická část

Kapitola 3
Ontologie

Pojem ontologie má v různých vědeckých disciplínách odlišný význam. Obvykle je ale chápána jako studie toho, co existuje. Popisuje informace o obecných vlastnostech entit a vztazích mezi těmito entitami. [1, 2, vlastní překlad]

3.1 Různé interpretace ontologie

Poprvé byl termín ontologie představen už v 19. století německým filozofem Rudolfem Gockelem v jeho díle Lexicon Philosophicum. Ve filozofii se proces budování ontologie zabývá tvořením systémů kategorií, které odpovídají určitým vizím světa. První známý systém kategorií byl navržen Aristotelem. V jeho systému se pomocí kategorie klasifikuje cokoliv, co může být řečeno o čemkoliv. [3, vlastní překlad]

Ve 3. století př. n. l. řecký filozof Porfyrios z Tyru rozšířil Aristotelův systém tím, že jeho kategorie zorganizoval do stromového diagramu. Tato struktura je známá jako Porfyriův strom(viz obrázek 3.1). [3, vlastní překlad]

Nejčastěji používaná definice ontologie pochází od Thomase R. Grubera: „*Ontologie je formální, explicitní specifikace společné konceptualizace.*“ *Konceptualizací* je myšlen abstraktní model, *explicitní* znamená, že prvky musí být jasně definované a *formálním* je myšleno, že specifikace by měly být zpracovatelné počítačem. [3, 4, vlastní překlad]

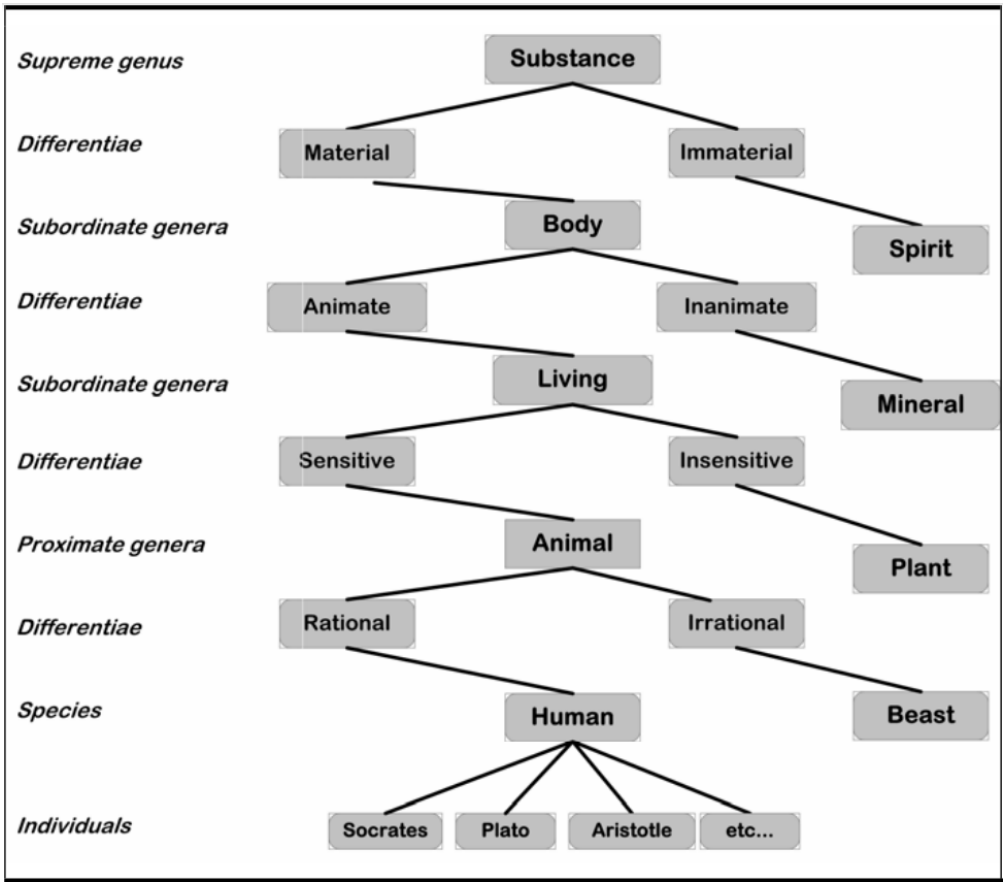
John F. Sowa vytvořil následující definici: „*Ontologie je studie kategorií různých věcí, které existují nebo můžou existovat v nějaké oblasti. Produktem takovéto studie je katalog typů věcí, o kterých předpokládáme, že existují v oblasti O z perspektivy člověka, který používá jazyk J za účelem popsání oblasti O.*“ [5, vlastní překlad] Nicméně neexistuje žádná univerzální definice pro termín ontologie. Jedním z důvodů je široké spektrum možných použití pro tohoto odborného názvu. [3, vlastní překlad]

Nicola Guarino předložil seznam v odborné literatuře nejpoužívanějších definic pro ontologii:

1. „*Ontologie je filozofická disciplína.*“
2. „*Ontologie je neformální konceptualizací systému.*“
3. „*Ontologie je formální sémantický popis.*“
4. „*Ontologie je specifikací konceptualizace.*“
5. „*Ontologie je reprezentace konceptuálního systému pomocí nějaké logické teorie.*“
6. „*Ontologie je slovník používaný nějakou logickou teorií.*“

7. „*Ontologie reprezentuje specifikaci meta-úrovně nějaké logické teorie.*“

[6, 3, vlastní překlad]



Obrázek 3.1 Porfyriův strom, moderní ilustrace [3]

3.2 Ontologie v informatice

„*V informatice se jako ontologie označuje reprezentace a definice kategorií, vlastností a vztahů mezi koncepty (pojmy, daty a entitami).*“ [7]

3.3 Klasifikace typů ontologií

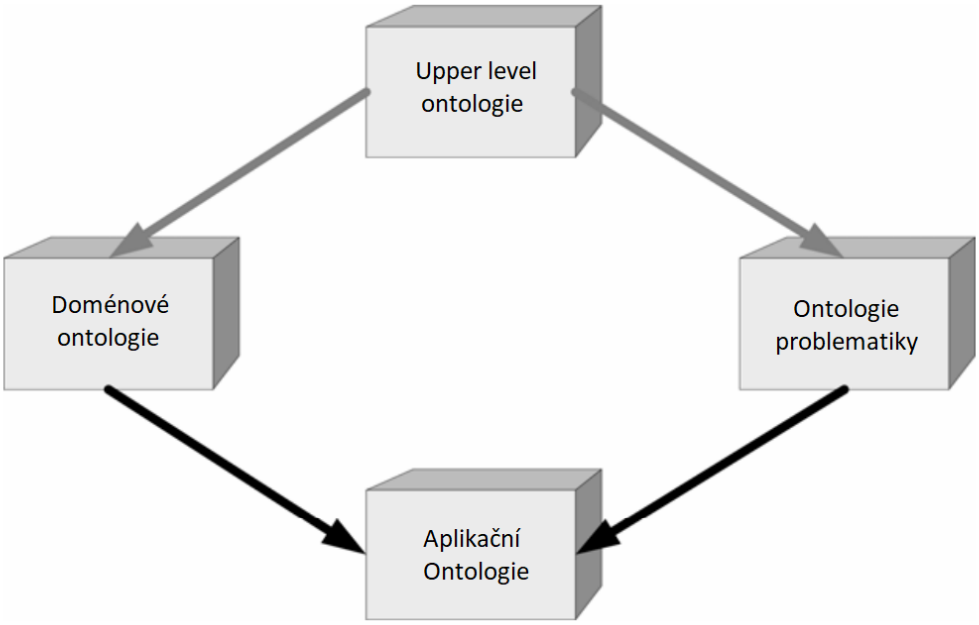
Z důvodů, kvůli kterým neexistuje jednoznačná definice pro ontologii, neexistuje ani jednoznačná klasifikace jejích typů. Můžeme ale říct, že jednou z nejpoužívanějších v informatice je klasifikace typů ontologií podle jejich obecnosti (*classification based on the generality of the ontology*). Toto rozdělení bylo navrženo Nicolem Guarinem a obsahuje následující čtyři (viz obrázek 3.2) kategorie: [6, 3, vlastní překlad]

1. Upper level ontologie

2. Doménové ontologie

3. Ontologie problematiky
4. Aplikační ontologie

[8, 3, vlastní překlad]



Obrázek 3.2 Klasifikace ontologií podle Nicola Guarina. [6, 3, vlastní překlad]

3.3.1 Upper level ontologie

Používají se k usnadnění sémantické integrace doménových ontologií (více viz 3.3.2). Z tohoto důvodu popisují obecné pojmy, které jsou podobné ve všech oblastech, jako například prostor, čas a události. [3, 9, vlastní překlad] Tyto ontologie jsou tím pádem doménově nezávislé a lze je znovu použít ke konstrukci nových ontologií.

3.3.1.1 Příklady upper level ontologií

- Unified Foundational Ontology (UFO)(více viz 3.4)

  Vytvořena Giancarlem Guizzardim a společníky v roce 2005. UFO spojuje již existující upper level ontologie do jedné. Je základem ontologického modelovacího jazyka OntoUML(více viz 5). [10, vlastní překlad]

- Basic Formal Ontology (BFO)

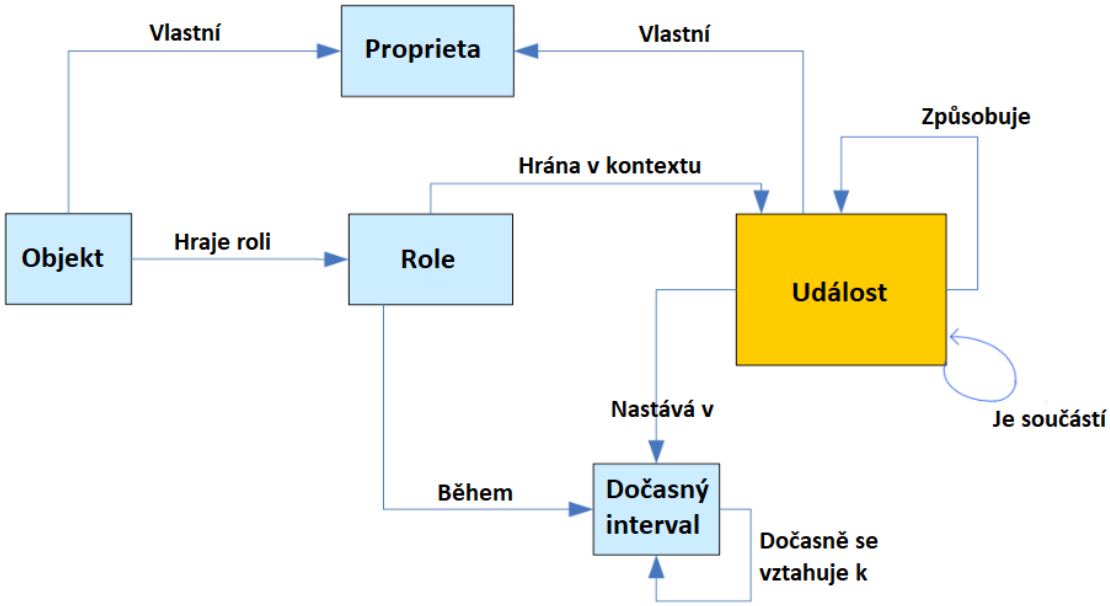
  BFO bylo vytvořeno Barry Smithem a Pierrem Grenonem v roce 2002 za účelem podpory tvorby konzistentních lower level ontologií nejdříve pouze v oblasti biomedicínského výzkumu, ale později i v dalších oblastech (např. obrany a bezpečnosti). [11, vlastní překlad]

- Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE)

  Tato upper level ontologie byla navržena v roce 2002 Nicolem Guarinem a spol. DOLCE je orientována na zachycení ontologických kategorií, které jsou základem přirozeného jazyka a lidského zdravého rozumu. [12, vlastní překlad]

- Suggested Upper Merged Ontology (SUMO)

  SUMO bylo vytvořeno firmou Teknowledge Corporation a první vydání se realizovalo v roce 2000.Tato ontologie je základem pro různé počítačové systémy na zpracování informací. [13, vlastní překlad]



Obrázek 3.3 Příklad jednoduché vyšší ontologie. [7, vlastní překlad]

3.3.2 Doménové ontologie

Poskytují formální reprezentaci specifické domény. Jsou esenciální pro interoperabilitu mezi softwarovými aplikacemi. Vědci v mnoha různých oblastech uznali potřebu doménových ontologií pro jasné definování specializovaných pojmů v dané oblasti. [14, vlastní překlad]

3.3.3 Ontologie problematiky

Popisují slovník potřebný k plnění obecných úkolů nebo činností tím, že se specializují na pojmy poskytované upper level ontologií. [3, vlastní překlad]

3.3.4 Aplikační ontologie

Popisují slovník konkrétní aplikace, jejíž koncepty obecně odpovídají rolím, které v dané oblasti vykonávají entity při provádění nějakého úkolu nebo aktivity. [3, vlastní překlad]

3.4 Unified Foundational Ontology

Unified Foundational Ontology je jednou z upper level ontologií (nebo také foundational ontologies), která byla vytvořena před dvaceti lety italsko-brazilským vědcem Giancarlem Guizzardim a jeho společníky. [15, vlastní překlad]

UFO je tvořena spojením dvou dříve existujících upper level ontologií známých jako General Formal Ontology (GFO) a Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) (více viz 3.3.1.1). Cílem této syntézy bylo vytvořit upper level ontologii(více viz 3.3.1), která je přizpůsobená pro aplikace na konceptuální modelování. [2, vlastní překlad]

UFO byla použita jako základ při navrhování ontologického modelovacího jazyka zvaného OntoUML(více viz 5), který byl vytvořen Giancarlem Guizzardim. [15, vlastní překlad]

3.4.1 Klasifikace UFO

UFO se dělí do tří vrstev, které se zabývají různými aspekty reality. [16, vlastní překlad]

3.4.1.1 UFO-A

Jedná se o ontologii endurantů, neboli entit, které můžeme vnímat jako kompletní koncept v jakémkoliv úseku času. Zabývá se aspekty strukturálního a konceptuálního modelování. Je organizována do čtyř kategorií, které se skládají z teorie typů a taxonomických struktur propojených s teorií objektových identifikátorů, relací částí a celků, konkrétních vnitřních vlastností a atributů a jejich hodnot, konkrétních relačních vlastností a relací a rolí. [16, vlastní překlad]

3.4.1.2 UFO-B

Je ontologie perdurantů. Perdurant je entita, na kterou když se podíváme v jakémkoliv úseku času, tak jí existuje pouze část. Také se dá označit jako ontologie událostí a procesů. Jedná s koncepty jakými jsou například perdurantská mereologie, dočasné uspořádání perdurantů, účastnění se objektů mezi penduranty, příčinná souvislost, změna a propojení mezi událostmi a enduranty skrz dispozice. [16, vlastní překlad]

3.4.1.3 UFO-C

Jde ontologii záměrných a sociálních entit, která je zkonstruovaná na UFO-A a UFO-B. Zabývá se pojmy jako přesvědčení, touhy, záměry, cíle, akce, závazky, tvrzení, sociální role, sociální konkrétní relační komplexy a další. [16, vlastní překlad]

3.4.2 Mikroteorie

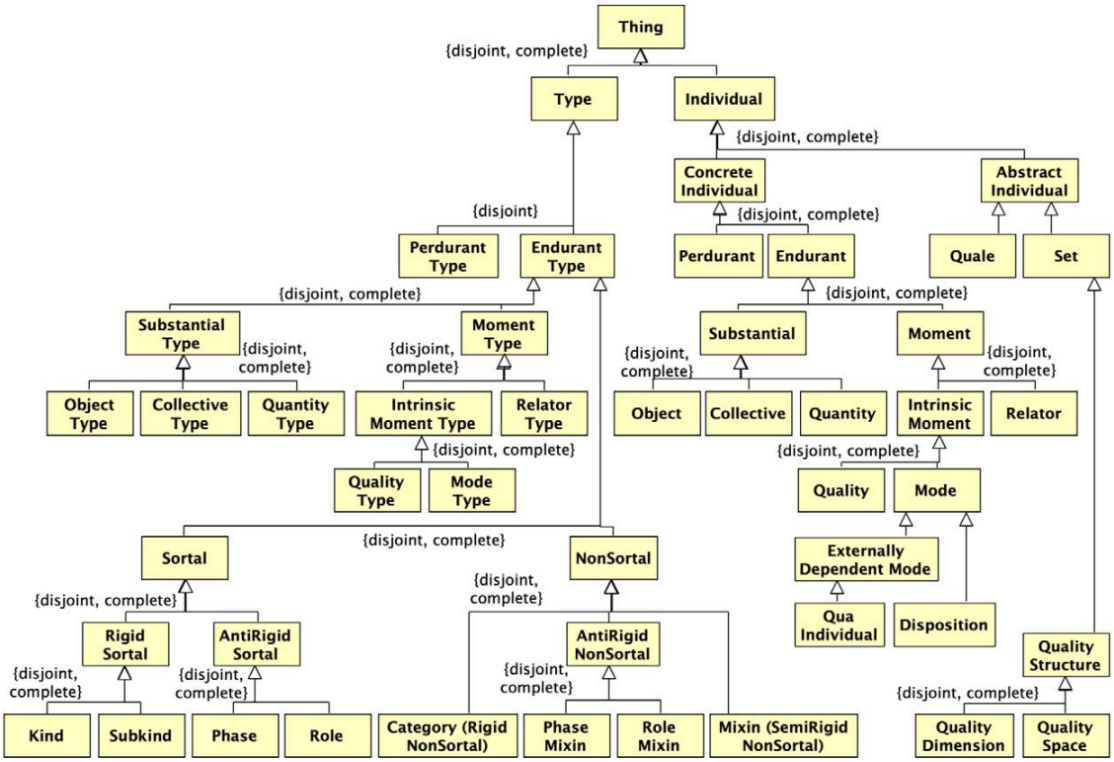
UFO byla vyvinuta konzistentním spojováním řady různých teorií, které vznikly z oblastí, jako jsou například formální ontologie ve filozofii, kognitivní věda, lingvistika a filozofie logiky. Zahrnuje množství mikroteorií zabývajících se základními pojmy koncepčního modelování. [16, vlastní překlad]

Tyto mikroteorie jsou:

- „*teorie typů a taxanomických struktur, která je propojená s teorií identifikátorů objektů a zahrnuje formální sémantiku v sortální kvantifikované modální logice*“
- „*teorie relací mezi části a celky*“
- „*teorie konkrétních vnitřních vlastností, atributů a hodnot atributů, které zahrnují pohled na typy dat jako sémantické odkazové struktury*“
- „*teorie konkrétních relačních vlastností a relací, včetně návrhu aby Weak Truthmaking propojovalo konkrétní vlastnosti s návrhy*“
- „*teorie rolí*“

- „*teorie událostí, včetně aspektů jako mereologie událostí, časové uspořádání událostí, účast objektů na událostech, kauzalita, změna a spojení mezi událostmi a enduranty prostřednictvím dispozic*“
- „*teorie víceúrovňového modelování*“

[15, vlastní překlad]



Obrázek 3.4 Taxonomie UFO. [15]

Kapitola 4
UML

„UML je standardní jazyk pro specifikaci, vizualizaci, konstrukci a dokumentaci všech artefaktů softwarového systému.“ [17, vlastní překlad]

Nejdříve nazývaný jako Unified Method, Unified Modeling Language byl vytvořen v roce 1995 Jimem Rumbaughem a Grady Boochem ve firmě Rational a později se k vývoji přidal Ivar Jacobson. V roce 1997 bylo UML předloženo Object Management Group k standardizaci. Na rozdíl od všeobecného přesvědčení Rational nevlastní UML, i když na něm nadále pracují. UML je vlastněn OMG. [17, vlastní překlad]

4.1 Klasifikace UML diagramů

UML můžeme definovat jako kolekci grafických modelů nebo diagramů, které vyjadřují různé vlastnosti objektově orientovaného návrhu. Dva nejdůležitější typy grafických diagramů jsou strukturální a behaviorální diagramy. [18, vlastní překlad]

4.1.1 Strukturální diagramy

Strukturální diagramy vyjadřují informace o struktuře tříd a objektů v rámci systému, jako jsou jejich vzájemné vztahy, atributy a omezení. [18, vlastní překlad]

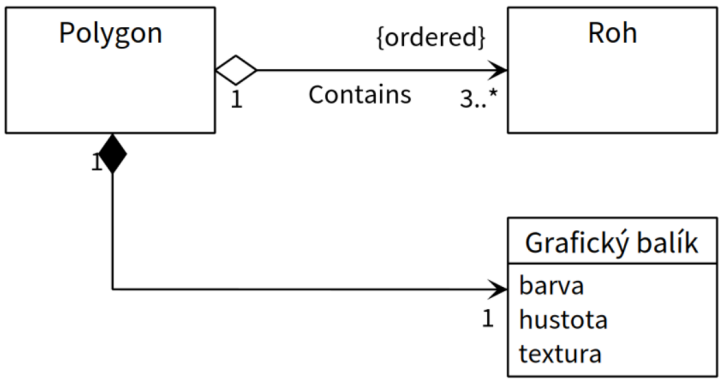
Druhy strukturálních diagramů:

- Class Diagram
- Component Diagram
- Composite Structure Diagram
- Deployment Diagram
- Object Diagram
- Package Diagram
- Profile Diagram

[19]

4.1.1.1 Class Diagram

Nejdůležitějším strukturálním diagramem a centrálním modelem celého UML je class diagram. Class diagram je dvou-dimenzionální model, který obsahuje následující komponenty: kolekci typů nebo tříd, set hodnot, popisy operací tříd, set vztahů a generalizační hierarchie mezi třídami nebo typy. [18, vlastní překlad]



Obrázek 4.1 Příklad class diagramu. [18, vlastní překlad]

4.1.2 Behaviorální diagramy

Behaviorální diagramy vyjadřují informace o dynamickém chování programu, který je navrhován. Typicky tyto diagramy popisují chování z hlediska změny stavu a předávání zpráv, které musí nastat na nějakou systémovou událost. [18, vlastní překlad]

Druhy behaviorálních diagramů:

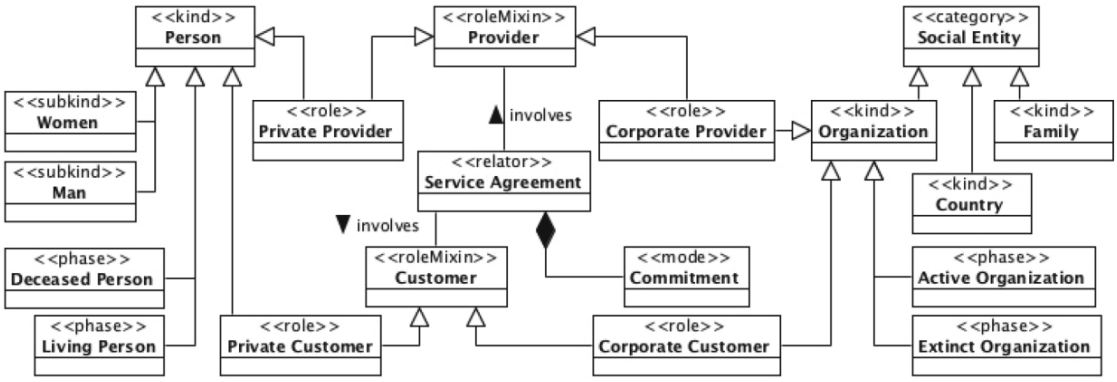
- Activity Diagram
- State Machine Diagram
- Use Case Diagram
- Communication Diagram
- Interaction Overview Diagram
- Sequence Diagram
- Timing Diagram

[19]

Kapitola 5
OntoUML

OntoUML je ontologicky založený konceptuální modelovací jazyk, navržený podle pravidel upper level ontologie Unified Foundational Ontology (více viz 3.4) , vytvořený Giancarlem Guizzardim v roce 2005 jako rozšíření jazyka UML (více viz 4). [20, vlastní překlad]

Na rozdíl od jiných rozšíření UML nestaví OntoUML na ontologicky vágním pojmu *třídy* jazyka UML, ale představuje úplný systém, nezávislý na původních prvcích jazyka UML. Používá některé aspekty UML jako třídy, ale opomíjí řadu dalších problematických konceptů, například agregace a kompozice, a nahrazuje je vlastními ontologicky správnými pojmy. [20, vlastní překlad]



Obrázek 5.1 Příklad OntoUML modelu. [21]

5.1 Princip identity

Entita může mít princip identity, který explicitně vyjadřuje vlastnosti, které žádné dvě různé instance té entity nemohou mít společné, protože takové vlastnosti jednoznačně identifikují každou instanci té entity. Entity se v OntoUML mohou dělit podle principu identity na dvě skupiny: [21, vlastní překlad]

- Sortály - Mají princip identity.
- Non-sortály - Nemají princip identity.

[21, vlastní překlad]

5.2 Sortály

Sortální typy jsou ty, které buď poskytují, nebo přebírají jednotný princip identity pro své instance. V kategorii sortálů také rozlišujeme mezi rigidními a anti-rigidními sortály. [21, vlastní překlad]

5.2.1 Rigidita sortálů

Sortály se v OntoUML mohou dále dělit podle rigidity, a to do dvou skupin:

- Rigidní
- Anti-rigidní

[21, vlastní překlad]

5.2.2 Druhy sortálů

5.2.2.1 Rigidní sortály

„Rigidní sortál je takový, který nutně klasifikuje své instance (v modálním smyslu), tj. instance tohoto typu jimi nemohou přestat být, aniž by přestaly existovat.“ [21, vlastní překlad]

- «Kind»

  «Kind» se používá ve valné většině OntoUML modelů. Poskytuje identitu, tedy nemůže mít jinou entitu, která poskytuje identitu jako super-typ. Jako příklady entity «Kind» lze uvést auto nebo počítač. [22, vlastní překlad]

- «Subkind»

  Tato entita je jediným rigidním sortálem, který nemá vlastní identitu, ale musí ji zdědit. «Subkind» reprezentuje rigidní specializaci jakýchkoliv poskytovatelů identity (tedy všech ostatních rigidních sortálů a také aspektů). Například červené víno, muž a vidlička. [22, vlastní překlad]

- «Relator»

  «Relator» představuje určité věci, které musí existovat, aby dvě nebo více jiných entit mohly být propojené pomocí materiálních relací (více viz 5.4.2.2). Například zaměstnání nebo sňatek. [22, vlastní překlad]

Obrázek 5.2 Příklad použití «Relator». [23, vlastní překlad]

- «Collective»

  «Collective» reprezentuje rigidní koncepty, které poskytují identitu jejich instancím. Hlavní charakteristikou je jeho vnitřní homogenní struktura. Mezi příklady patří kapela a rodina. [22, vlastní překlad]

- «Quantity»

  «Quantity» je podobná «Collective» tím, že reprezentuje rigidní koncepty, které poskytují identitu jejím instancím, ale na rozdíl od «Collective» reprezentuje nepočítatelné věci, jako například písek a voda. [22, vlastní překlad]

5.2.2.2 Aspekty

Aspekty jsou součástí rigidních sortálů, ale na rozdíl od ostatních rigidních sortálů jsou existenčně závislé na svém nositeli (bearer). Tato vazba se nazývá inherence. [23, vlastní překlad]

- «Mode»

  «Mode» je typ vnitřní vlastnosti, která nemá žádnou strukturovanou hodnotu. «Mode» je existenčně závislý na svém nositeli (bearer). Mezi příklady patří schopnost nebo nemoc. Každý «Mode» musí být přímo nebo nepřímo připojený k «Characterization» vztahu (více viz 5.4.2.4). [22, vlastní překlad]

- «Quality»

  «Quality» je typ vnitřní vlastnosti, která má strukturovanou hodnotu. Každá «Quality» je závislá na věcech, které charakterizuje. Tento typ entity se dělí do tří skupin podle konceptu, který charakterizují, a to «PerceivableQuality», «NonPerceivableQuality» a «NominalQuality». Mezi příklady entit, které můžeme označit tímto typem, patří výška, hodnota nebo číslo kreditní karty. Každá «Quality» musí být přímo nebo nepřímo připojená k «Characterization» vztahu (více viz 5.4.2.4). (viz obrázek 5.7) [22, vlastní překlad]

5.2.2.3 Anti-rigidní sortály

„*Anti-rigidita charakterizuje sortál, jehož instance se mohou pohybovat dovnitř a ven z jeho rozšíření, aniž by se změnila jejich identita.*“ [21, vlastní překlad]

- «Role»

  «Role» je entita, která reprezentuje anti-rigidní specializace poskytovatelů identit (aspektů a rigidních sortálů kromě «Subkind»), které jsou vytvořeny relačních kontextech. Mezi příklady entit «Role» patří student, pacient nebo cestovatel. [22, vlastní překlad]

- «Phase»

  «Phase», podobně jako «Role», reprezentuje anti-rigidní specializace poskytovatelů identit, ale na rozdíl od «Role» tyto specializace musí být vytvořené změnami ve vnitřních vlastnostech. Tedy například věk člověka nebo barva objektu. [22, vlastní překlad]

5.3 Non-sortály

Non-sortály neposkytují princip identity pro jejich instance. Klasifikují věci, které mají společné vlastnosti, ale řídí se podle odlišných principů identity. [21, vlastní překlad]

5.3.1 Rigidita non-sortálů

Non-sortály se v OntoUML mohou dále dělit podle rigidity do tří skupin:

- Rigidní
- Anti-rigidní
- Semi-rigidní

[21, vlastní překlad]

5.3.2 Druhy non-sortálů

5.3.2.1 Rigidní non-sortál

„*Rigidní non-sortál je disperzní typ, který agreguje esenciální vlastnosti, které jsou společné pro různé rigidní sortály (např. fyzické objekty agregují základní vlastnosti stolů, aut a sklenic).*“ [24, vlastní překlad]

- «Category»

  «Category» entita nepotřebuje závislost, aby mohla být definována. Používá se k připojení základních vlastností k jednotlivcům, kteří se řídí různými principy identity. Například koncepty objekt nebo přístroj se mohou definovat jako «Category». [22, vlastní překlad]

5.3.2.2 Anti-rigidní non-sortál

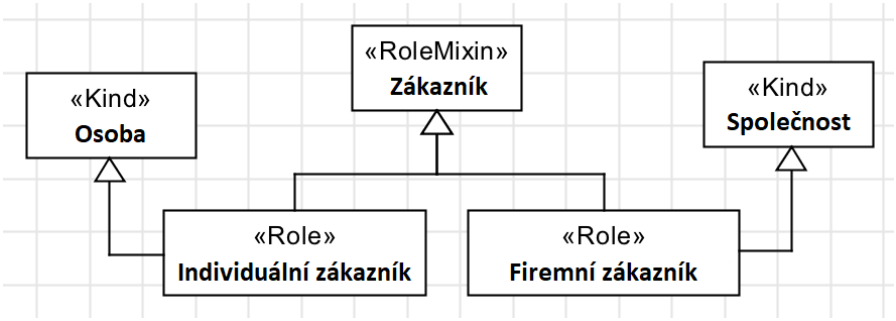
„*Anti-rigidní non-sortál je disperzní typ, který agreguje vlastnosti, které jsou společné pro různé role (např. typ zákazník, který agreguje vlastnosti individuálních zákazníků a firemních zákazníků).*“ [24, vlastní překlad]

- «PhaseMixin»

  ≪PhaseMixin≫ je ekvivalentní «Phase», s tím rozdílem, že je pro entity, které agregují instance s různými principy identity. [22, vlastní překlad]

- «RoleMixin»

  ≪RoleMixin≫ je ekvivalentní «Role», s tím rozdílem, že je pro entity, které agregují instance s různými principy identity. [22, vlastní překlad]



Obrázek 5.3 Příklad použití «RoleMixin». [22, vlastní překlad]

5.3.2.3 Semi-rigidní non-sortál

„*Non-sortál je semi-rigidní, pokud je rigidní pro některé jednotlivce a anti-rigidní pro jiné.*“ [25, vlastní překlad]

- «Mixin»

  «Mixin» je v OntoUML jediný typ entity, který je semi-rigidní. Charakterizuje entity s rozdílnými principy identity. [22, vlastní překlad]

5.4 Relace (Vztahy)

Relace neboli vztahy jsou entity stejně jako sortály nebo non-sortály, ale na rozdíl od nich, se používají ke spojovaní ostatních entit do většího celku, nebo k naznačení vztahů mezi jednotlivými entitami. Relace se rozdělují do tří kategorií: [24, vlastní překlad]

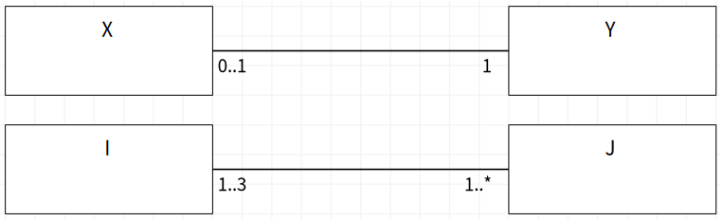
- Asociace
- Generalizace
- Agregace mezi částí a celkem

[24, vlastní překlad]

5.4.1 Multiplicita

Multiplicita vztahů nám udává možný počet instancí entit na každém konci daného vztahu. Multiplicita se značí u všech typů znaků až na generalizaci.

Některé vztahy mají předem určený počet na počáteční nebo konečné straně vztahu a nebo na obou stranách zároveň (např. SubQuantityOf a Containment). Počet instancí se určuje těmito anotacemi: *0..1, 1..1, 0..*, 1..**. Zde znak hvězdičky představuje nějaký neznámý počet, ale může být nahrazen i jakýmkoliv přirozeným číslem. [24, vlastní překlad]



Obrázek 5.4 Příklad multiplicity relací.

5.4.2 Asociace

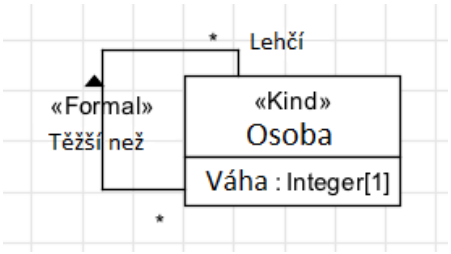
Asociace se používají k definování vztahů mezi instancemi různých tříd. [26, vlastní překlad] Jedná se o jednoduchý vztah mezi dvěma entitami. Existuje několik druhů asociací:

- Formální relace
- Materiální relace
- Mediace (Mediation)
- Charakterizace (Characterization)
- Derivace (Derivation)
- Strukturace (Structuration)

[22, vlastní překlad]

5.4.2.1 Formální relace

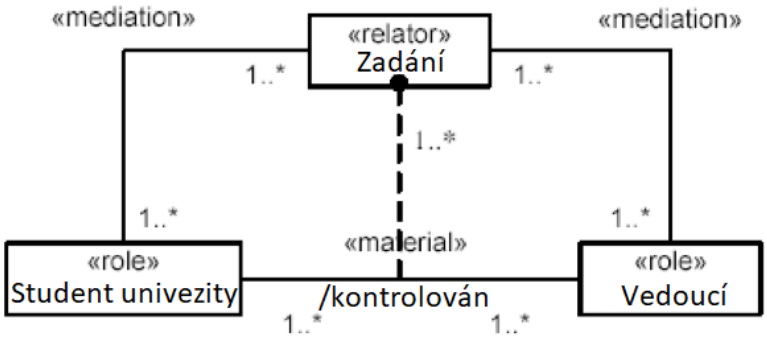
Formální relace existují mezi dvěma entitami samostatně bez entity «Relator». Můžou také popisovat vztah, co má entita sama se sebou. Vždy jsme schopni identifikovat vnitřní atributy entit, které relace definují. Mezi příklady: patří být vyšší než, větší než nebo být částí. Do formálních relací ale také patří kromě jiného vztahy vzniku, inherence, kvality, asociace, závislosti. Tedy vztahy, které tvoří matematickou nadstavbu. [24, vlastní překlad]



Obrázek 5.5 Příklad formální relace. [22, vlastní překlad]

5.4.2.2 Materiální relace

Materiální relace mají, na rozdíl od formálních, svoji vlastní materiální strukturu. Pro propojení entit pomocí materiálních relací je potřeba použít entitu «Relator» (více viz 5.2.2.1). Entity propojené materiálními relacemi získají některé další atributy, které jsou existenčně závislé na samotných entitách. [24, vlastní překlad]



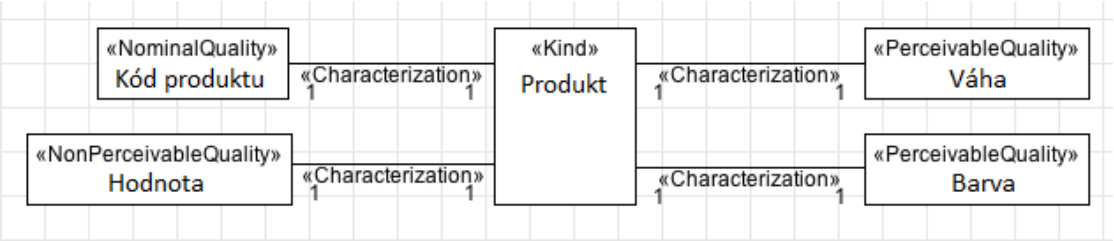
Obrázek 5.6 Příklad materiální relace a mediace. [22, vlastní překlad]

5.4.2.3 Mediace (Mediation)

Tato relace definuje vztah mezi «Relator» a entitami, které spojuje (viz obrázek 5.6). [24, vlastní překlad]

5.4.2.4 Charakterizace (Characterization)

Tato entita je vztah mezi nositelem (bearer type) a jeho vlastností (aspektem), tedy je určen pro propojení entit «Quality» nebo «Mode» s ostatními entitami (více viz 5.2.2.2). [24, vlastní překlad]



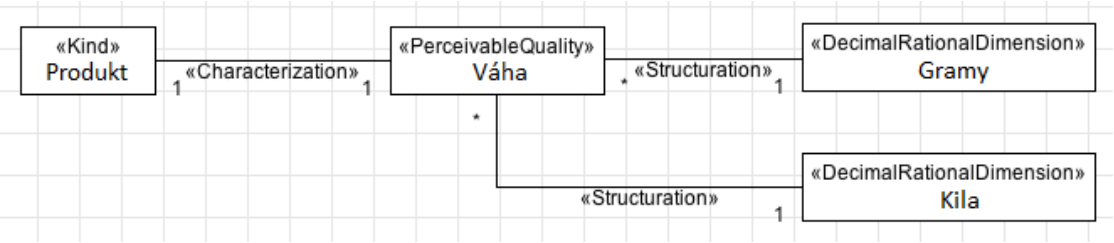
Obrázek 5.7 Příklad charakterizace. [22, vlastní překlad]

5.4.2.5 Derivace (Derivation)

Materiální relace mohou být zcela odvozené pomocí asociace derivace z «Relator» a odpovídajících relací mediace. [24, vlastní překlad]

5.4.2.6 Strukturace (Structuration)

Vztah strukturace umožňuje strukturaci entity «Quality». [24, vlastní překlad]



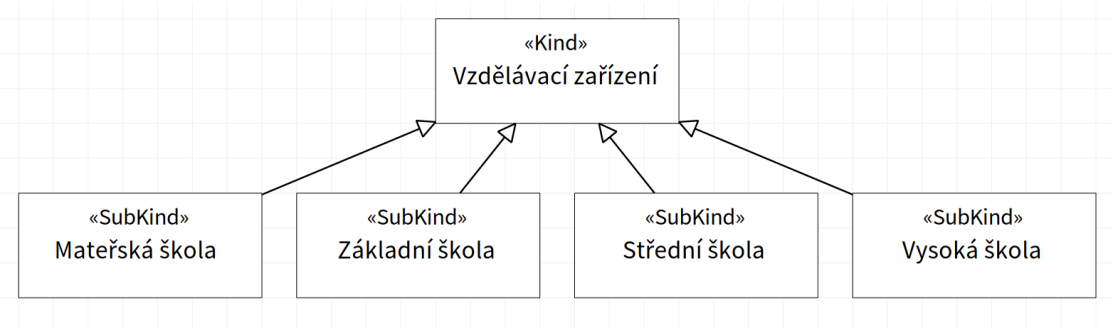
Obrázek 5.8 Příklad strukturace. [22, vlastní překlad]

5.4.3 Generalizace

Generalizace je taxonomický vztah mezi obecnou entitou a dalšími entitami, které jsou jejími specifikacemi. Tyto specifické entity dědí vlastnosti obecné entity. [27, vlastní překlad]

Tento vztah se používá ve chvílích, kdy máme jednu nebo více specifických entit, které mají společné vlastnosti mezi sebou a nějakou obecnou entitou. Pomocí generalizace každá specifická entita zdědí vlastnosti obecné entity a přidá svoje unikátní vlastnosti, takže tyto specifické entity mají jak svoje vlastnosti, tak i vlastnosti obecné entity. [26, 27, vlastní překlad]

Specifické entity jedné obecné mohou tvořit generalizační set (*GeneralizationSet*) za účelem definování skupiny specifických tříd s podobným významem. [26, 27, vlastní překlad]



Obrázek 5.9 Příklad generalizace.

5.4.4 Agregace mezi částí a celkem

Poslední kategorií relací jsou agregace mezi částí a celkem. Jedná se o relace, které se dají použít pouze v kontextu částí a celku. K popisu vztahů mezi částí a celkem lze použít i formální asociace upravené pomocí speciálních anotací. (Více o části a celku viz 5.5)

Skupina relací specializovaných pouze pro podrobný popis vztahů mezi částí a celkem obsahuje těchto pět vztahů:

- Containment
- ComponentOf
- MemberOf
- SubCollectionOf
- SubQuantityOf

[22]

5.4.4.1 Containment

Containment je vztah mezi nádobou a jejím obsahem. Tento vztah je určený pouze pro rigidní sortál «Quantity» (více viz 5.2.2.1). Například sud obsahuje pivo. Multiplicita (více viz 5.4.1) tohoto vztahu musí být vždy jedna ku jedné. [24, vlastní překlad]

5.4.4.2 ComponentOf

ComponentOf je vztah mezi dvěma celky, tedy entity na obou koncích tohoto vztahu musí být funkčními komplexy. Například: motor je součástí automobilu nebo srdce je součástí oběhového systému. Tento vztah se řídí principem slabé suplementace (více viz 5.5.0.1). [24, vlastní překlad]

5.4.4.3 MemberOf

MemberOf je vztah mezi nějakým funkčním komplexem, nebo «Collective» (více viz 5.2.2.1), který reprezentuje část, a «Collective», který reprezentuje celek. Mezi příklady patří: strom je součástí lesa, karta je součástí balíčku karet. MemberOf funguje podle slabého principu suplementace (více viz 5.5.0.1). [24, vlastní překlad]

5.4.4.4 SubCollectionOf

SubCollectionOf je relace, která na obou koncích musí mít «Collective» (více viz 5.2.2.1) a také musí mít vždy multiplicitu (více viz 5.4.1) jedna ku jedné, stejně jako agregace části a celku Containment. Příklad použití této relace: všichni žolíci jsou součástí balíčku karet, kolekce vidliček je součástí příborů. [24, vlastní překlad]

5.4.4.5 SubQuantityOf

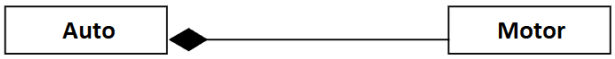
Podobně jako subCollectionOf je subQuantityOf také vztah mezi dvěma stejnými entitami s tím rozdílem, že to jsou entity typu «Quantity» (více viz 5.2.2.1). Stejně jako dříve zmíněné vztahy subQuantityOf musí mít multiplicitu (více viz 5.4.1) jedna ku jedné. Tento vztah by se použil v následujících příkladech: alkohol je součástí vína, kofein je součástí kávy. [24, vlastní překlad]

5.5 Část a celek (part-whole)

Samotný koncept části a celku a potom následné relace mezi nimi jsou součástí prakticky všech objektově orientovaných modelovacích jazyků. Jsou potřebné k popisu a jednoduchému pochopení složitých vztahů mezi jednotlivými entitami. Název filozoficko-vědního oboru, který se zabývá studiem vztahů částí a celku, je mereologie. [28, vlastní překlad]

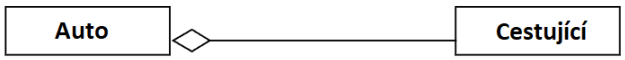
UML (více viz 4) identifikuje vlastnosti částí a celků nepřesně, pouze pomocí agregace a kompozice: [24, vlastní překlad]

- „*Motor je nedílnou součástí auta. (kompozice)*“ [23]



Obrázek 5.10 Příklad kompozice v UML. [23, vlastní překlad]

- „*Pasažéři existují i bez auta. (agregace)*“ [23]



Obrázek 5.11 Příklad agregace v UML. [23, vlastní překlad]

„*Pouze takovéto rozlišení je však zjednodušující a nejednoznačné pro ontologické modelování. Je třeba se zabývat zejména oběma směry (mezi Částí a Celkem) a rozlišovat přesněji "sílu"vzájemné povinnosti.*“ [23]

OntoUML rozvíjí a rozšiřuje vztahy mezi částí a celkem za účelem zpřesnění jejich vztahů. Proto kromě dříve zmíněné agregace a kompozice také zavádí pojmy ohledně multiplicity vztahu: [24, vlastní překlad]

- Povinná část (Mandatory part)

  „*Instance x je povinná část (mandatory part) jiné instance y, pokud y je genericky závislá na entitě T, jež x instancuje, a instance x je nezbytně částí instance y.*“ [23]

- Povinný celek (Mandatory whole)

  „*Instance y je povinný celek (mandatory whole) pro jinou instanci x, pokud x je genericky závislá na entitě T, který y instancuje, a instance x je nezbytně částí instance entity T.*“ [23]

- Nepovinná část (Voluntary part)

  „*Nepovinné části jsou části, které celek může postrádat bez vlivu na jeho klasifikaci či identitu.*“ [23]

- Nepovinný celek (Voluntary whole)

  „*Část může existovat bez celku.*“ [23]

[24]

Dále řadě rozlišujeme rozdíly v ontologické závislosti (ontological dependence), které jsou součástí charakteristiky rozdělitelnosti (separability): [24, vlastní překlad]

- Generická závislost (Generic dependence)

  „*Instance y je genericky závislá (generically dependent) na entitě T, pokud pro existenci y je nutné, aby existovala i instance T.*“ [23]

- Existenční závislost (Existential dependence)

  „*Instance x je existenčně závislá (existentially dependent) na jiné instanci y, pokud existence instance x vyžaduje existenci určité specifické instance y.*“ [23]

[24]

V poslední řadě také v OntoUML podrobně rozlišujeme v různé typy celků a jejich částí a na to používáme tyto anotace:

- {essential}
- {inseparable}
- {immutable part}
- {immutable whole}
- {disjoint}
- {complete}

[24]

Některé jednotlivé anotace můžeme kombinovat mezi sebou pro ještě větší upřesnění daného vztahu. [24]

5.5.0.1 Slabý princip suplementace (weak supplementation principle)

Slabý princip suplementace se týká některých relací částí a celku. Jednoduše říká, že aby entita tvořila celek, musí mít alespoň dvě samostatné části. Například pokud je auto tvořeno právě jedním motorem, aby se stalo celkem, musí obsahovat alespoň jeden další díl (např. kufr). [29, vlastní překlad]

Kapitola 6

OpenPonk

OpenPonk je open-source modelovací platforma implementovaná v dynamickém prostředí *Pharo*, které je vyvíjeno v objektově orientovaném jazyce SmallTalk, a díky tomu je jednoduchá na rozvíjení, přidávání funkcionalit a tvorbu pluginů. OpenPonk byl vytvořen výzkumnou skupinou Centrum pro Konceptuální modelování a Implementace (*Centre for Conceptual Modelling and Implementation - CCMi*). Vedoucí této bakalářské práce doc. Ing. Robert Pergl, Ph.D., je vedoucím výzkumné skupiny CCMi. Tento nástroj je užitečný nejenom pro studenty, ale i pro výzkumníky a odborníky v modelování. [30, 31, vlastní překlad]

OpenPonk nabízí velké množství nástrojů na tvorbu různých typů modelů. Těmito modely jsou:

- UML
- OntoUML
- BORM
- FSM (Konečné automaty)
- BPMN
- Petriho sítě

[31, vlastní překlad]

Kapitola 7
PEG.js

Jedná se o nástroj vytvořený Davidem Majdou, který generuje parsery v jazyce JavaScript podle svojí vlastní gramatiky. PEG.js může být využíván jak v online internetové verzi, tak i své ve vlastní aplikační verzi. [32, vlastní překlad]

„*Může být použit ke zpracování komplexních dat nebo počítačových jazyků a ke snadnému vytváření transformátorů, interpereterů, kompilátorů a jiných nástrojů.*“ [32, vlastní překlad]

7.1 Gramatika

Gramatika v PEG.js se používá k jednoduchému a hlavně krátkému popisu funkcionalit, které má finální vygenerovaný parser vykonávat. Pomocí využití nástroje PEG.js a jeho příslušné gramatiky jsou uživatelé schopni podstatně snížit nároky na čas a vědomosti, které jsou potřeba k vytvoření požadovaného parseru.

Vytvořená gramatika musí podrobně popisovat očekávaný vstup a specifikovat, co má výsledný parser vrátit jako výstup v různých situacích. Z výsledné gramatiky PEG.js vygeneruje parser jako JavaScript objekt s jednoduchým API. [32, vlastní překlad]

Výpis kódu 7.1 Příklad PEG.js gramatiky na zpracovávání jednoduché aritmetiky.

```
Expression
  = head:Term tail:(_ ("+" / "-") _ Term)* {
      return tail.reduce(function(result, element) {
        if (element[1] === "+") { return result + element[3]; }
        if (element[1] === "-") { return result - element[3]; }
      }, head);}
Term
  = head:Factor tail:(_ ("*" / "/") _ Factor)* {
      return tail.reduce(function(result, element) {
        if (element[1] === "*") { return result * element[3]; }
        if (element[1] === "/") { return result / element[3]; }
      }, head);}
Factor
  = "(" _ expr:Expression _ ")" { return expr; }
  / Integer
Integer "integer"
  = _ [0-9]+ { return parseInt(text(), 10); }
_ "whitespace"
  = [ \t\n\r]*
```

[32]

Část II

Praktická část

Kapitola 8

Ontologické konceptuální modely

Primárním cílem této bakalářské práce je tvorba ontologických konceptuálních modelů propojených pomocí mapovacích pravidel s datovými sadami vztahujícími se k určitým klíčovým doménám. Tato bakalářská práce se realizovala pod vedením doc. Ing. Roberta Pergla, Ph.D., v modelovacím týmu vedeným Bc. Terezou Machačovou v projektu Nest Big Data Arena firmy Remmark a. s. Výsledné modely byly nahrány do Datové platformy, kde budou přístupné uživatelům.

Modely byly vytvořené v platformě OpenPonk (více viz 6). Tento nástroj byl vybrán z důvodu jednoduchosti operování a vlastností, které nabízí. Dále kvůli získaným znalostem z bakalářského předmětu BI-KOM, kde byl používaný skrze celou výuku. V neposlední řadě byl nástroj OpenPonk vybrán z důvodu existence rozšíření OpeNest vytvořené v rámci projektu Nest Big Data Arena. Rozšíření OpeNest umožňuje tvorbu konceptuálních ontologických modelů přesně podle potřeby projektu NBDA, například tvorbou datových entit a zpracováním mapovacích pravidel, které nejsou součástí vanilla OntoUML (více viz 5). Dále toto rozšíření umožňuje přímý přístup do produkční databáze Datové Platformy a tím velice zjednodušuje tvorbu modelů. Také nabízí rozšířené možnosti formátů, ve kterých lze modely exportovat.

Cílem práce bylo vytvoření ontologických konceptuálních modelů propojených s příslušnými datovými sadami pomocí datových entit a mapovacích pravidel. Bylo zároveň nutné zachovat sémantickou interoperabilitu (více viz 8.1) jak mezi modely, které vznikly jako výsledek této práce, tak i mezi modely, které již existovaly v Datové platformě. Proto součástí tvorby modelů byla nutná precizní kontrola již existujících ontologických konceptuálních modelů.

Proces tvorby ontologických konceptuálních modelů se dá rozdělit do čtyř fází:

1. Ontologická analýza datových sad
2. Tvorba ontologických konceptuálních modelů
3. Propojení modelů s datovými sadami
4. Tvorba mapovacích pravidel

V této části bakalářské práce bylo vytvořeno šest ontologických konceptuálních modelů propojených s příslušnými datovými sadami pomocí mapovacích pravidel a datových entit. Následují názvy, identifikátory a popisy těchto modelů a jejich datových sad:

- Hosté a přenocování v hotelích podle zemí

  (hoste-a-prenocovani-v-hotelich-podle-zemi)

  Tento model se týká cestovního ruchu, konkrétně turismu v České republice. Příslušná datová sada obsahuje různé statistické údaje, například počet turistů a počet nocí strávených

v různých hromadných ubytovacích zařízeních. U turistů se také rozlišuje země trvalého pobytu, odkud do České republiky přicestoval. [33] Tato datová sada byla poskytnuta Českým statistickým úřadem.

- Naděje dožití v okresech a správních obvodech ORP

  (nadeje-doziti-v-okresech-a-spravnich-obvodech-orp)

  Tématem tohoto modelu je zdraví společnosti. Datová sada se zaměřuje na statistické údaje ohledně délky života obyvatel v okresech a správních obvodech obcí s rozšířenou působností (SO ORP). [33] Tato datová sada byla poskytnuta Českým statistickým úřadem.

- Přehled o počtu zaměstnavatelů, pojištěnců a pojistných vztahů podle okresů

  (pocty-zamestnavatelu-a-pojistencu)

  Model vyobrazuje entity a vztahy ohledně hospodářství. Datová sada, se kterou je tento model propojený, obsahuje údaje o zaměstnavatelích a pojištěncích. [34] Tato datová sada byla poskytnuta Českou správou sociálního zabezpečení.

- Příjmy domácností zaměstnanců a důchodců

  (prijmy-domacnosti-zamestnancu-a-duchodcu)

  Klíčovým tématem toho modelu jsou finance. Model byl vytvořený tak, aby bylo možné ho propojit s datovou sadou obsahující statistické údaje o příjmech a výdajích různých druhů domácností. Příjmy a výdaje jsou rozmodelované do podrobných kategorií. [33] Tato datová sada byla poskytnuta Českým statistickým úřadem.

- Školy a školská zařízení

  (skoly-a-skolska-zarizeni)

  Vzdělávání je centrálním motivem toho modelu. Jeho datová sada pracuje se statistickými údaji, například ohledně počtu různých druhů školských zařízení nebo počtu žáků. Tyto statistiky jsou rozdělené podle krajů České republiky. [33] Tato datová sada byla poskytnuta Českým statistickým úřadem.

- Vybavenost domácností informačními a komunikačními technologiemi

  (vybavenost-domacnosti-technologiemi)

  Tento model vyobrazuje část oblasti týkající se společnosti a populace. S ním propojená datová sada bude obsahovat statistické údaje o různém vybavení různých typů domácností elektronikou a podobnými technologiemi. [33] Tato datová sada byla poskytnuta Českým statistickým úřadem.

Vypracované modely jsou obsažené v příloze ve formátech json, opp a png.

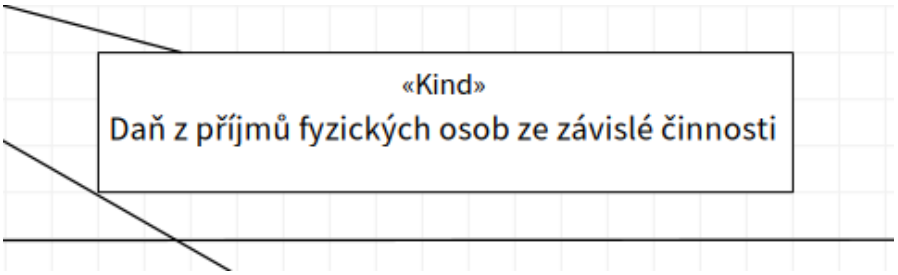
8.1 Sémantická interoperabilita

Zachování a rozšíření sémantické interoperability mezi rozsáhlými a odbornými oblastmi je jedním z hlavních důvodů pro vznik Datové platformy.

Sémantická interoperabilita zaručuje přímé propojení ontologických konceptuálních modelů do větších celků skrze identické entity, které se nachází v jednotlivých modelech. Tato vlastnost modelů v Datové Platformě zjednodušuje práci s daty z různých oblastí, které se překrývají. Například tedy nenastane situace, že máme dvě různé entity, které představují stejný koncept.

U konceptuálních modelů v Datové platformě je sémantická interoperabilita zaručena jak názvem dané entity, tak i jejím stereotypem. Jakmile bude jeden z těchto požadavků nedodržen, sémantická interoperabilita bude přerušena a místo propojení modelů do většího celku vzniknou duplicitní entity.

Obrázek 8.1 Příklad entity z modelu ”Příjmy domácností zaměstnanců a důchodců”.



Obrázek 8.2 Příklad entity z modelu ”Daňová statistika”.

Na příkladech vidíme dvě stejné entity jak v názvu a stereotypu ze dvou různých modelů. Modely ”Daňová statistika”a ”Příjmy domácností zaměstnanců a důchodců”jsou propojeny skrze entitu ”Daň z příjmů fyzických osob ze závislé činnosti”se stereotypem «Kind». Modely můžou být propojené skrze jednu nebo více entit a zároveň některé modely nemusí být propojené vůbec.

8.2 Ontologická analýza datových sad

Ontologická analýza datových sad a jejich příslušných klíčových domén je prvním krokem k tvorbě ontologických konceptuálních modelů.

Datová sada se skládá z atributů, precizního popisu těchto atributů a nakonec z obecného popisu oblasti, kterou datová sada představuje. Datové sady jsou poskytnuty externím zákazníkem a zpracovány jiným týmem v projektu Nest Big Data Arena. Zpracování datových sad není součástí této bakalářské práce.

Obecný popis datové sady poskytuje obecné informace ohledně oblasti, kterou datová sada popisuje, jako například téma, klíčová slova, název.

Dále datová sada obsahuje jednotlivé atributy, které obsahuje a které musí být korektně rozděleny do datových entit v konceptuálním modelu.

Nakonec datová sada obsahuje popis jednotlivých atributů. Tenhle popis pomůže při tvorbě mapovacích pravidel, případně při tvorbě samotného konceptuálního modelu.

Název	Datový typ	Popis
hodnota	Numeric	zjištěná hodnota údaje je zaokrouhlena na Kč; v případě, že se jedná o důvěrný údaj, je sloupec prázdný
ekakocd_kod	Numeric	kód položky číselníku pro ekonomickou aktivitu osoby v čele domácnosti charakterizuje druh domácnosti (např. domácnosti zaměstnanců, důchodců apod.); pokud je prázdný, jedná se o průměrné hodnoty za všechny druhy domácností; 4 - nezaměstnaní, 5 - samostatně činní, 6 - zaměstnanci, 11 - ostatní domácnosti, 14 - důchodci
ekakocd_txt	String	text druhu domácnosti podle ekonomické aktivity osoby v čele domácnosti; pokud je prázdný, jedná se o průměrné hodnoty za všechny druhy domácností. 220 - Ekonomická aktivita; 221 - Ekonomická aktivita – agregace
rok	Date(YYYY)	rok referenčního období ve formatu RRRR

Tabulka 8.1 Několik atributů z datové sady "Příjmy domácností zaměstnanců a důchodců".

Počet atributů se může mezi jednotlivými modely výrazně lišit, zároveň počet atributů není lineární s velikostí výsledného ontologického modelu.

Po pročtení popisů datové sady následuje důkladná analýza oblasti, kterou datová sada popisuje. Součástí této analýzy je precizní studium odborných názvů, dělení, struktur, atd. na internetových stránkách. Ke každé datové sadě je také přiřazena ukázková databáze, díky které je možné ještě dále upřesnit ontologický konceptuální model a poté mapovací pravidla v datových entitách daného modelu.

Je také nutné zkontrolovat popisy jednotlivých atributů v datové sadě podle internetových zdrojů, jelikož se může stát, že se v těchto popisech vyskytuje chyba. Dále je možné, že některé samotné atributy nevyhovují struktuře, kterou používá modelovací tým ve všech modelech. Kromě ostatních věcí i struktura některých atributů musí být jednotná, aby byla zachována a rozšířena sémantická interoperabilita (více viz 8.1).

V poslední řadě je nutné prohledat ontologické konceptuální modely, konkrétně tzv. "šablony", a zjistit, jestli nějaká nepokrývá naše téma.

8.2.1 Šablony

Šablony jsou modely, které nebyly vytvořeny za účelem přidání do produkční databáze modelů v Datové Platformě. Neobsahují žádné datové entity a tím pádem žádná mapovací pravidla. Není k nim propojená žádná datová sada.

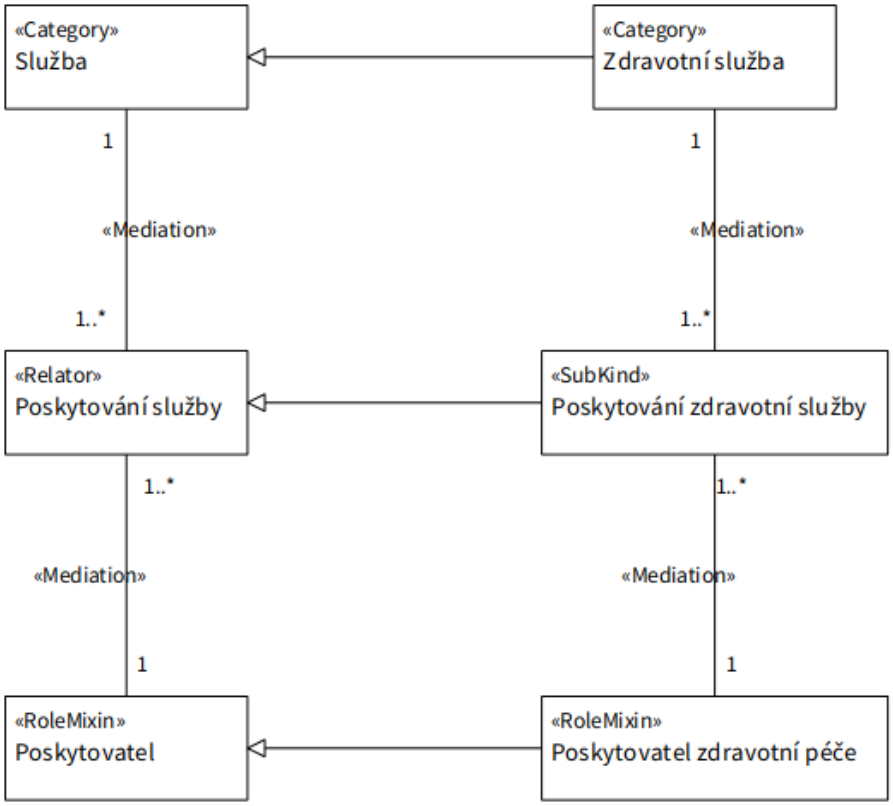
Tyto speciální ontologické konceptuální modely byly vytvořeny ke zjednodušení práce v modelovacím týmu. Pokrývají konkrétní oblasti, které se často objevují v různých datových sadách. Členům modelovacího týmu ulehčují práci tím, že místo modelování identické oblasti několikrát od začátku můžou použít tyto šablony, pokud existuje nějaká, která jejich požadovanou oblast pokrývá.

Používání těchto ontologických šablon také přispívá ke zjednodušení zachování sémantické interoperability mezi modely.

Příklady názvů některých šablon:

- Zdravotní služby
- Nemoci
- Státy EU

Na tvorbu modelů, které jsou výsledkem této bakalářské práce, nebyly použity žádné šablony z toho důvodu, že neexistovaly žádné, které by pokrývaly vyžadované oblasti. Zároveň výsledkem této bakalářské práce nejsou žádné šablony.



Obrázek 8.3 Ukázka šablony ”zdravotní služby”.

8.3 Tvorba ontologických konceptuálních modelů

Na ontologickou analýzu datových sad navazuje tvorba samotných ontologických konceptuálních modelů v jazyce OntoUML (více viz 5), v programu OpenPonk (více viz 6). Zde používáme informace získané při ontologické analýze ke tvorbě modelů.

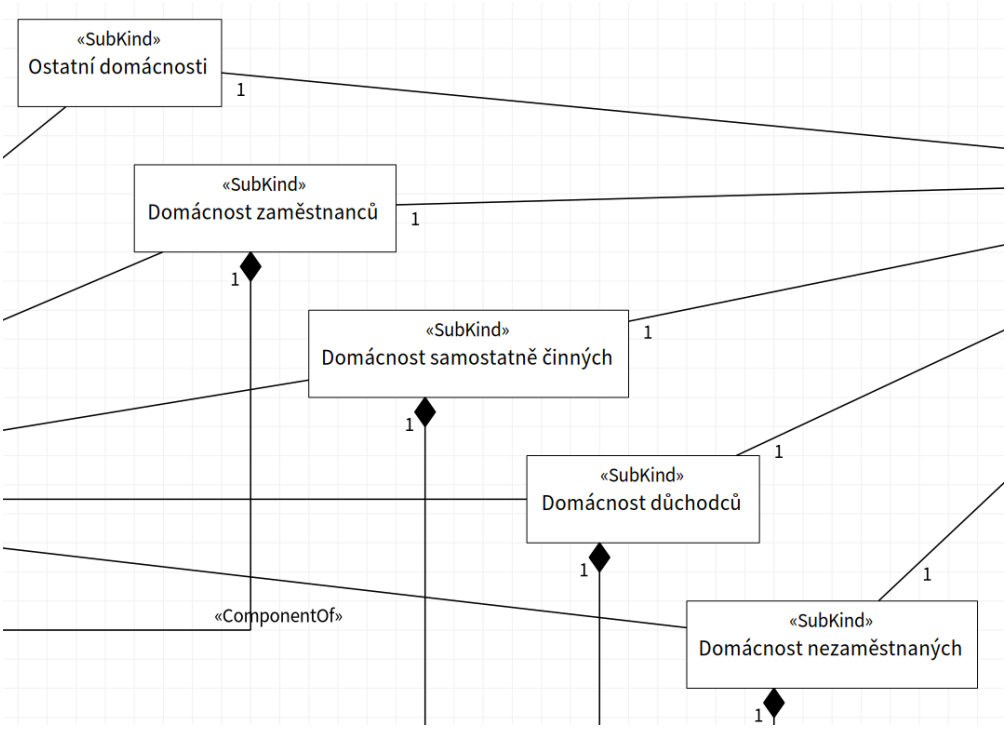
8.3.1 Postup tvorby modelů

V této sekci je popsán postup tvorby ontologických modelů s použitím příkladů z modelu "Příjmy domácností zaměstnanců a důchodců", který je součástí výsledku této bakalářské práce. Celý model se vyskytuje v příloze této bakalářské práce.

V průběhu tvorby modelu je nutné důkladně kontrolovat již existující modely v databázi Datové platformy, aby byla zachována sémantická interoperabilita (více viz 8.1). Tento proces velice zjednodušuje rozšíření OpenPonku zvané OpeNest (více viz 8), které kromě jiného umožňuje přímý a rychlý přístup k modelům v produkční databázi skrze OpenPonk. Ze stejného důvodu je také nutné udržovat komunikaci v modelovacím týmu, hlavně mezi členy, kteří pracují na datových sadách popisující podobné oblasti, jelikož tyto rozdělané modely, na kterých se ještě pracuje, nejsou v databázi Datové platformy a tím pádem k nim členové nemají mezi sebou přímý přístup.

1. Pokud existuje model šablony (více viz 8.2.1) vyhovující naší datové sadě, tak se začne model stavět na ní, pokud ne, musí se začít v od začátku. Konkrétně pro model "Příjmy domácností zaměstnanců a důchodců"neexistuje žádná použitelná šablona.

2. Namodelují se okrajové entity, tedy ty entity, které budou napojené na datové entity v dalším kroku.



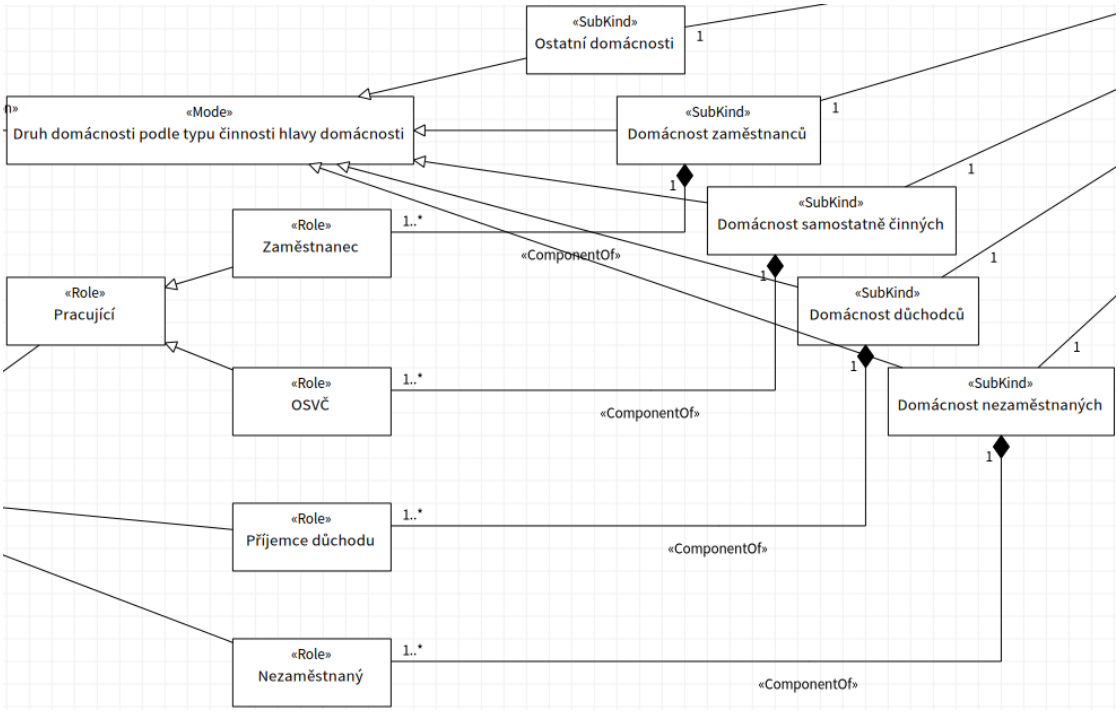
Obrázek 8.4 Příklad okrajových entit z modelu "Příjmy domácností zaměstnanců a důchodců".

Název a podobu okrajových entit zjistíme podle popisu atributů v datových sadách. Je důležité udělat analýzu přiložené databáze a také podrobnou analýzu na internetu, jelikož je možné, že popis atributů není úplný nebo korektní. V tomhle případě byl.

Název	Datový typ	Popis
ekakocd_kod	Numeric	kód položky číselníku pro ekonomickou aktivitu osoby v čele domácnosti charakterizuje druh domácnosti (např. domácnosti zaměstnanců, důchodců apod.); pokud je prázdný, jedná se o průměrné hodnoty za všechny druhy domácností; 4 - nezaměstnaní, 5 - samostatně činní, 6 - zaměstnanci, 11 - ostatní domácnosti, 14 - důchodci

Tabulka 8.2 Atribut z datové sady "Příjmy domácností zaměstnanců a důchodců".

3. Posledním krokem je namodelování vnitřních entit a vztahů ontologického modelu a propojení okrajových entit. V tomto procesu zase čerpáme z informací získaných při ontologické analýze a zároveň důkladně vnitřní entity porovnáváme s entitami z již existujících modelů, abychom neporušili sémantickou interoperabilitu (více viz 8.1).



Obrázek 8.5 Příklad vnitřních entit z modelu "Příjmy domácností zaměstnanců a důchodců".

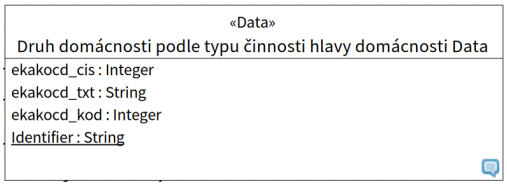
8.4 Propojení modelů s datovými sadami

Propojení modelů s datovými sadami je zprostředkováno pomocí speciálních data entit. Tyto data entity nejsou součástí vanilla OntoUML (více viz 5) ani OpenPonk (více viz 6). Do obojího jsou přidány pomocí rozšíření OpeNest (více viz 8), které je součástí projektu Nest Big Data Arena.

8.4.1 Postup tvorby data entit

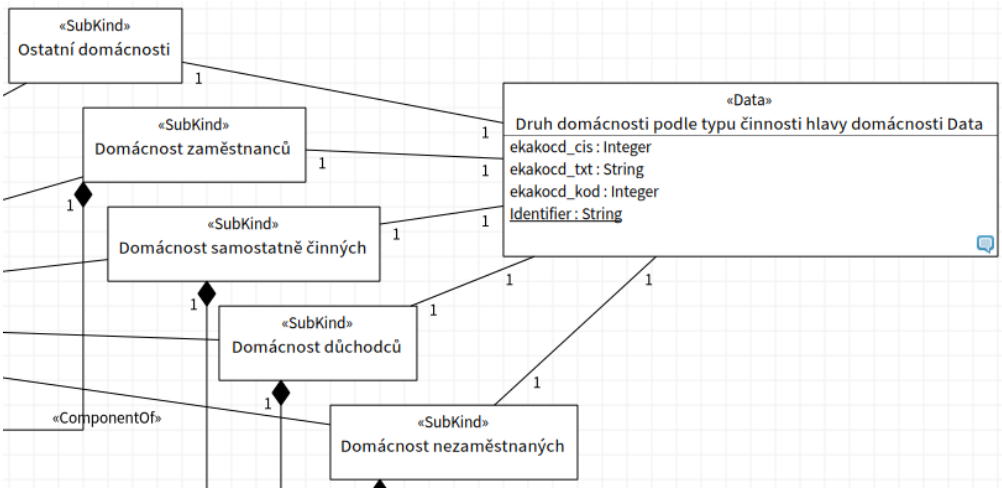
Postup propojení modelů s datovými sadami je důkladně popsán na příkladu z modelu "Příjmy domácností zaměstnanců a důchodců".

1. Datová entita se vytvoří jako jakákoliv jiná entita v nástroji OpenPonk. Každá datová entita může obsahovat jednu nebo více atributů zmíněných v popisu datové sady a také musí obsahovat atribut "identifier", který se popíše ve fázi tvorby mapovacích pravidel. Atributy se shromažďují v jednotlivých entitách, pokud mají mezi sebou nějakou souvislost. Poté se pojmenují podle skupiny okrajových entit (více viz 8.3.1), které jsou asociované s jednotlivými atributy, které datová entita obsahuje. Konec názvu datové entity musí vždy obsahovat slovo "Data".



Obrázek 8.6 Příklad datové entity z modelu "Příjmy domácností zaměstnanců a důchodců".

2. Datová entita se připojí jednoduchou 1:1 formální asociací (více viz 5.4.2) přímo na dříve zmíněnou skupinu okrajových entit, tedy tu skupinu entit, která souvisí s atributy, které daná datová entita obsahuje.



Obrázek 8.7 Příklad vazeb na datovou entity v modelu "Příjmy domácností zaměstnanců a důchodců".

Proces se opakuje, dokud všechny atributy zmíněné v datové sadě nejsou součástí datových entit v ontologického modelu.

8.5 Tvorba mapovacích pravidel

Po vyplnění všech atributů datové sady do datových entit ontologického konceptuálního modelu následuje vyplnění těchto datových entit pomocí datových pravidel.

Mapovací pravidla ještě více upřesňují strukturu jednotlivých dat, které model a jeho atributy reprezentují, neboli podle mapovacích pravidel se určí, které entity patří ke konkrétnímu vstupu dat. Tato pravidla se píší do popisu jednotlivých datových entit v programu OpenPonk.

Každá datová entita musí obsahovat kromě mapovacích pravidel atributů ještě speciální atribut "identifier"(Identifikátor). Popis tohoto atributu obsahuje název datové sady a používá se k identifikaci datové sady nebo ontologického modelu, ke kterému patří.

Mapovací pravidla mohou být velice rozsáhlá, proto je důležité je důkladně kontrolovat, aby nevznikla chyba. I tak se ale někdy může stát, že nějaká syntaktická chyba nebude odhalena a dostane se až do produkční databáze. Tento zásadní problém řeší druhý výstup mé bakalářské práce, syntaktický validátor (více viz 9), který automaticky zkontroluje syntaxi a tvar všech mapovacích pravidel v modelu.

Následuje ukázka části mapovacích pravidel z datové entity "Druh domácnosti podle typu činnosti hlavy domácnosti Data"(viz obrázek 8.6) z modelu "Příjmy domácností zaměstnanců a důchodců".

Výpis kódu 8.1 Příklad mapovacích pravidel z modelu "Příjmy domácností zaměstnanců a důchodců".

```
("ekakocd_kod" = 4) -> ("ekakocd_cis" -> "Domácnost nezaměstnaných");
("ekakocd_kod" = 5) -> ("ekakocd_cis" -> "Domácnost samostatně činných");
("ekakocd_kod" = 6) -> ("ekakocd_cis" -> "Domácnost zaměstnanců");
("ekakocd_kod" = 11) -> ("ekakocd_cis" -> "Ostatní domácnosti");
("ekakocd_kod" = 14) -> ("ekakocd_cis" -> "Domácnost důchodců");
("ekakocd_kod" = "") -> ("ekakocd_cis" -> "Domácnost nezaměstnaných" OR
"Domácnost samostatně činných" OR "Domácnost zaměstnanců" OR "Ostatní domácnosti" OR
"Domácnost důchodců");
```

8.5.1 Druhy mapovacích pravidel

Existují 3 různé druhy mapovacích pravidel a také další nástroje na rozšíření použitelnosti daných druhů. Tato pravidla se řídí podle různých struktur a návazností.

- Základní mapovací pravidla
- Mapování podle hodnoty
- Podmíněná mapovací pravidla

Zjednodušeně řečeno mapovací pravidla říkají, jaký **atribut** patří k jaké **entitě** při jaké **hodnotě atributu**.

8.5.1.1 Základní mapovací pravidla

Základní mapovací pravidlo je nejjednodušší, které existuje. Prostě a jasně říká, jaký **atribut** patří k jaké **entitě** bez ohledu na **hodnotu atributu**.

Výpis kódu 8.2 Tvar základního mapovacícho pravidla.

```
"atribut" -> "entita";
```

Základní mapovací pravidla jsou jednoduchá jak tvarově, tak i ve svojí funkci. Proto se moc často nepoužívají, většinou pouze na nejobecnější atributy. V ontologickém modelu ”Příjmy domácností zaměstnanců a důchodců”je pouze jedno jednoduché pravidlo, a to v datové entitě ”Příjmy domácností Data”.

Výpis kódu 8.3 Příklad základního mapovacího pravidla z modelu ”Příjmy domácností zaměstnanců a důchodců”.

```
"idhod" -> "Domácnost";
```

Toto pravidlo je ohledně atributu ”idhod”, které obsahuje id hodnotu jednotlivých záznamů do dané datové sady. Tato hodnota je určena k identifikaci jednotlivých záznamů. Pravidlo říká, že atribut ”idhod”patří k entitě ”Domácnost”za jakýchkoliv okolností.

8.5.1.2 Mapování podle hodnoty

Toto mapovací pravidlo už bere v potaz jednotlivé **hodnoty atributů**. Říká, že pokud má **atribut** nějakou určitou **hodnotu**, potom se mapuje na určitou **entitu**. Může být použita i negace.

Výpis kódu 8.4 Tvar základního mapovacícho pravidla”.

```
"atribut" = "hodnota" -> "entita";
"atribut" != "hodnota" -> "entita";
```

Hodnota může mít jakýkoliv tvar (Date, String, Numeric, regulární výraz), záleží na typu atributu. Tento typ pravidla je již hojně využívaný.

Výpis kódu 8.5 Příklad mapování podle hodnoty z modelu ”Příjmy domácností zaměstnanců a důchodců”.

```
"ekakocd_kod" = 4 -> "Domácnost nezaměstnaných";
```

Toto pravidlo říká, že pokud atribut ”ekakocd_kod”je rovný čtyřem, potom tento atribut se namapuje na entitu ”Domácnost nezaměstnaných”.

8.5.1.3 Podmíněná mapovací pravidla

Podmíněné pravidlo se využívá, když určení mapování atributu závisí na hodnotě jiného atributu. Tedy toto pravidlo říká, že pokud **atribut jedna** má nějakou **hodnotu**, potom **atribut dva** patří k určité **entitě**. Také lze použít negace stejně jako u mapování podle hodnoty.

Výpis kódu 8.6 Tvar podmíněného mapovacícho pravidla”.

```
("atribut1" = "hodnota") -> ("atribut2" -> "entita");
("atribut1" != "hodnota") -> ("atribut2" -> "entita");
```

Toto je další často využívaný druh mapovacích pravidel.

Výpis kódu 8.7 Příklad podmíněného mapovacího pravidla z modelu ”Příjmy domácností zaměstnanců a důchodců”.

```
("ekakocd_kod" = 4) -> ("ekakocd_cis" -> "Domácnost nezaměstnaných");
```

Pravidlo v příkladu říká, že pokud atribut ekakocd_kod má hodnotu čtyři, potom atribut ekakocd_cis patří k entitě s názvem ”Domácnost nezaměstnaných”.

8.5.2 Logické spojky

Mapovací pravidla jakéhokoliv druhu mohou dále obsahovat logické spojky.

- AND
- OR

Tyto logické spojky mohou být jak na straně předpokladu, tak na straně závěru a nejsou nijak omezené počtem. Používají se k rozšíření funkčnosti mapovacích pravidel a vyjádření vazby, která by bez logických spojek ztvárnit nešla.

8.5.2.1 AND

Logická spojka ”AND”se využívá, pokud atribut má být namapován na dvě nebo více entit najednou nebo při použití MEA anotací (více viz 8.5.3) nebo když nastanou obě situace naráz.

Výpis kódu 8.8 Příklad použití logické spojky ”AND”bez MEA anotací.

```
veg" = "clen" -> "Vegetarián" AND "Člen rodiny";
```

Model ”Příjmy domácností zaměstnanců a důchodců”neobsahuje mapovací pravidlo s logickou spojkou ”AND”, proto byl použit jiný příklad. Předešlá ukázka pravidla říká, že pokud má **atribut** ”veg”**hodnotu** ”clen”, potom je vázán k **entitám** ”Vegetarián”a ”Člen rodiny”zároveň.

Výpis kódu 8.9 Příklad použití logické spojky ”AND”s MEA anotacemi z modelu ”Příjmy domácností zaměstnanců a důchodců”.

```
("stapro_kod" = 6008) -> ("hodnota" -> "Hrubý peněžní příjem domácnosti" AND
"MEA_Průměr" AND "MEA_CZK" AND "MEA_Poměrová stupnice" AND "MEA_Finanční obnos");
```

Předešlý příklad ukazuje podmíněné mapovací pravidlo, kde v závěru byla použita logická spojka ”AND”k připojení několika MEA anotací(více viz 8.5.3).

8.5.2.2 OR

Logická spojka ”OR”se používá hlavně při situacích, když chceme vymezit skupinu entit, ke kterým může atribut patřit, ale zároveň nakonec může patřit pouze k jedné.

Výpis kódu 8.10 Příklad použití logické spojky ”OR”z modelu ”Příjmy domácností zaměstnanců a důchodců”.

```
"ekakocd_kod" = "" -> "Domácnost nezaměstnaných" OR "Domácnost samostatně činných"
OR "Domácnost zaměstnanců" OR "Ostatní domácnosti" OR "Domácnost důchodců";
```

Příklad 8.10 říká, že pokud je atribut ”ekakocd_kod”prázdný, potom patří k právě jedné ze zmíněných entit.

8.5.3 MEA anotace

MEA (neboli ”measurement”) anotace se používají pro upřesnění některých atributů a jejich hodnot, většinou u atributů, co vyjadřují nějakou podobu času jako interval, dobu trvání, atd. nebo nějakou číselnou hodnotu (jako v 8.9) jako počet, procento, atd. Pro jejich zakomponování do mapovacích pravidel se používají logické spojky (více viz 8.5.2).

Mezi stovky aktivně využívaných MEA anotací patří:

- MEA_Měsíc

  Přidává se k atributům, které vyjadřují nějakou podobu času v měsících.

- MEA_CZK

  Dává se do mapovacích pravidel k atributům, které se týkají měny korun českých.

- MEA_Finanční obnos

  Je u atributů, které vyjadřují nějakou informaci ohledně financí.

Počet MEA anotací není nijak omezen. U zmíněných příkladů by v mapovacích pravidlech anotaci ”MEA_CZK”vždy doprovázela minimálně anotace ”MEA_Finanční obnos”.

Kapitola 9
Syntaktický validátor

Sekundárním výstupem této bakalářské práce je syntaktický validátor. Tento program umožňuje členům modelovacího týmu společnosti Remmark a. s. zautomatizovat a zjednodušit velkou část procesu kontrolování korektnosti modelů, konkrétně kontrolu struktury a syntaktické správnosti mapovacích pravidel, a proto je aktivně využíván členy modelovacího týmu.

9.1 Nástroje

Syntaktický validátor je celý vytvořený v jazyce javaScript. Pro vygenerování části kódu, který obstarává tokenizaci vstupních dat, byla použita gramatika, kterou využívá jako vstup nástroj PEG.js (více viz 7).

Pro verzování a zálohování kódu při tvorbě programu a následnou distribuci programu mezi členy modelovacího týmu byl využit Gitlab.

9.2 Analýza požadavků

Požadavky, které má syntaktický analyzátor splňovat, byly vytvořeny pod vedením doc. Ing. Roberta Pergla, Ph.D., vedoucího této bakalářské práce.

9.2.1 Funkční požadavky

Funkční požadavky tohoto programu obsahují pouze kontrolu syntaxe a struktury mapovacích pravidel. Program nekontroluje samotnou korektnost významu daných mapovacích pravidel nebo ontologického konceptuálního modelu. Tento proces je natolik složitý, že vyžaduje přítomnost lidského faktoru.

- Kontrola struktury mapovacích pravidel

  Prvním z předem určených funkčních požadavků byla kontrola struktury mapovacích pravidel. Mapovací pravidla mají jasně daná pravidla pro jejich tvorbu a musí se nutně dodržovat pro správné plnění jejich funkce (více viz 8.4).

- Kontrola syntaxe prvků mapovacích pravidel

  Druhým funkčním požadavkem je kontrola syntaxe prvků mapovacích pravidel. Tento proces je vhodné automatizovat, jelikož vyžaduje hodně času a také jelikož se snadno může stát, že v mapovacích pravidlech bude zanechaná syntaktická chyba, která negativně ovlivní jejich funkčnost. Automatizace tohoto procesu by zajistila jejich správnost a zároveň by urychlila proces kontroly.

9.2.2 Nefunkční požadavky

Některé klasické kategorie nefunkčních požadavků byly opomenuty z důvodu povahy programu a jeho účelu.

- Rychlost

  Výsledná rychlost je důležitý parametr pro tento program. Je nutné zařídit vysokou rychlost programu vzhledem k vstupním datům, aby práce uživatele na konstrukci ontologického modelu nebyla zbrzděna a on mohl využít výstup tohoto programu k určení, jak bude dál pokračovat.

- Spolehlivost

  Dalším důležitým parametrem tohoto programu je jeho spolehlivost. Syntaktický validátor nesmí vynechávat chyby obsažené v modelech a na stejný vstup musí vždy vracet stejný výstup.

- Bezpečnost

  Jelikož Datová platforma, výsledek projektu Nest Big Data Arena, bude placená služba, je nutné, aby byla zařízena bezpečnost dat, se kterými se pracuje.

- Rozšiřitelnost a modifikovatelnost

  I když v případě tohoto programu jsou rozsáhlá rozšíření nepravděpodobná, tak je důležité zařídit, aby mohla být uskutečněná buď úplně bez zásahu do stávajícího kódu, nebo jen s velmi malým zásahem.

  Dále je pravděpodobné, že vznikne potřeba funkčnost syntaktického analyzátoru modifikovat. Struktura mapovacích pravidel a pravidla pro jejich tvorbu se můžou v budoucnosti měnit, a tak musí být snadné tyto změny zanést i do fungování syntaktického analyzátoru. To se zařídí přehlednou strukturou kódu a jeho důkladným okomentováním.

9.3 Vyhodnocení požadavků

Při tvorbě syntaktického validátoru bylo dosaženo všech funkčních i nefunkčních požadavků. Dále kromě dosažení všech předem stanovených funkčních požadavků byla přidána funkcionalita kontroly korektnosti a existence identifikátoru datových entit.

9.3.1 Vstupní data

Jako vstupní data tento program přijímá celé ontologické konceptuální modely (více viz 8) ve formátu JSON. Nativní formát ontologických modelů je oop, ale o konverzi z oop na JSON se jednoduše postará nástroj OpenPonk (více viz 6), který byl použit na tvorbu těchto modelů. Program nemá žádné grafické UI. Spouští se jednoduše pomocí příkazové řádky, kde příkaz má jeden parametr, a to název ontologického modelu ve formátu JSON.

Výpis kódu 9.1 Ukázka struktury příkazu na spuštění syntaktického validátoru.

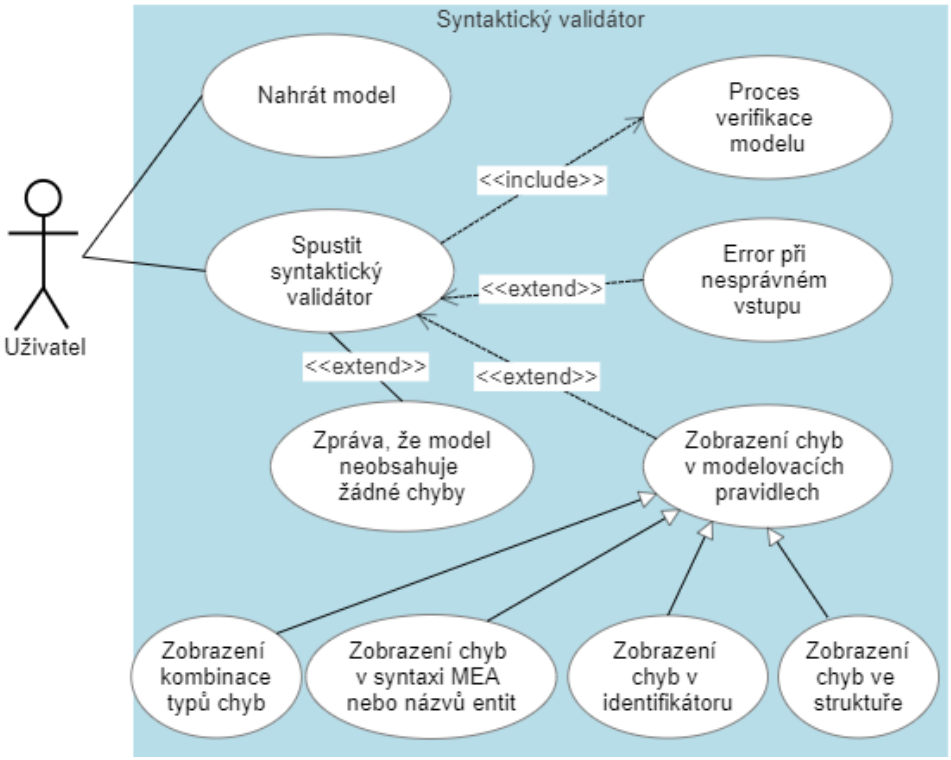
```
.\syntaxValidator-win.exe název_souboru.json
```

Ontologický model ve formátu JSON musí být ve složce "**modelsJSON**", která je na stejné úrovni jako exe soubor, aby byl syntaktickým analyzátorem nalazen.

9.3.2 Výstupní data

Výstupem programu je jasný textový popis typu chyby a její přesná lokace v ontologickém konceptuálním modelu. Tento výstup je vypsán do konzole, kde byl program spuštěn.

Kromě popisu jednotlivých chyb výstup obsahuje také celkový počet chyb, které byly nalezené.



Obrázek 9.1 Use case diagram syntaktického validátoru".

9.3.3 Funkční požadavky

9.3.3.1 Kontrola struktury mapovacích pravidel

Kontrola struktury mapovacích pravidel je zařízena kódem, který byl vygenerovaný z gramatiky (tato gramatika je dostupná ve zdrojových souborech programu, viz příloha) pomocí nástroje PEG.js (více viz 7). Tato část kódu také obstarává tokenizaci mapovacích pravidel.

Výpis kódu 9.2 Příklad výstupu při nalezené chybě ve struktuře mapovacích pravidel.

```
SyntaxError: Expected "->", "AND", "OR", or [ ] but "-" found.
    at entity: Ubytování v hotelu Data
    at mapping rule: "stapro_kod" != 2654 - "Počet turistů ubytovaných
    v hotelu";
```

Program odhalí jakékoliv chyby ve struktuře mapovacích pravidel, které by ovlivnily jejich funkčnost. Správně rozeznává všechny přítomné druhy mapovacích pravidel a jejich možná rozšíření pomocí MEA anotací (více viz 8.5.3) a logických spojek (více viz 8.5.2).

9.3.3.2 Kontrola syntaxe prvků mapovacích pravidel

Po tokenizaci vstupního souboru do struktury na jednotlivé entity, vztahy a datové entity se tato struktura dále zpracovává a rozděluje na menší díly, se kterými je jednodušší pracovat.

Dále se také shromáždí názvy normálních entit (ne datových) z celého ontologického konceptuálního modelu, aby se mohla kontrolovat korektnost syntaxe názvů entit v mapovacích pravidlech.

Výpis kódu 9.3 Příklad výstupu při nalezené chybě v názvu entity v mapovacím pravidle.

```
SyntaxError: Entity name in mapping rules does not match any
entities.
    at entity: Referenční období Data
    entity name not found: Referenční obdob
```

Všechny používané názvy MEA anotací se nachází v souboru "**namesMEA.js**", obsah tohoto souboru se použije k syntaktické kontrole MEA anotací.

Výpis kódu 9.4 Příklad výstupu při nalezené chybě v názvu MEA anotace v mapovacím pravidle.

```
SyntaxError: Measurement name (MEA_*) in mapping rules does not
match any existing MEA annotations.
    at entity: Referenční období Data
    measurement name not found: MEA_Časový okamži
```

9.3.3.3 Kontrola identifikátorů datových entit

Kontrola identifikátorů datových entit byla přidána jako funkční požadavek nad rámec zadání.

Každá datová entita musí mít svůj vlastní identifikátor. Identifikátor se přidává do datové entity jako statický atribut typu **String** s názvem **Identifier**. Tvar identifikátoru se vyplní v nástroji OpenPonk do komentáře atributu **Identifier**.

Výpis kódu 9.5 Příklad výstupu při chybějícím indentifikátoru v datové entitě.

```
SyntaxError: Identifier attribute is missing.
    at entity: Ubytovací zařízení Data
```

Program odhalí, když identifikátor úplně chybí, nebo nemá správné vlastnosti.

Výpis kódu 9.6 Příklad výstupu při nesprávném indentifikátoru v datové entitě.

```
SyntaxError: Identifier doesn't match file name.
    at entity: Referenční období Data
    is       : hoste-a-prenocovani-v-hotelich-podle-zem
    should be: hoste-a-prenocovani-v-hotelich-podle-zemi
```

Syntaktický validátor také rozezná a vypíše, pokud tvar identifikátoru je nesprávný.

9.3.4 Nefunkční požadavky

Všechny předem zadané nefunkční požadavky byly splněny.

9.3.4.1 Rychlost

Rychlost syntaktického validátoru byla otestována na nejrozsáhlejších ontologických modelech, které byly k dispozici. Rychlost procesu kontroly byla vysoká a nijak nebude zdržovat uživatele programu. Z toho důvodu nebyla po stresovém testování uskutečněna snaha o optimalizaci programu.

9.3.4.2 Spolehlivost

Spolehlivost a jednoznačnost výstupu syntaktického analyzátoru je zajištěna automatizovanými testy a rozsáhlým manuálním testováním.

Manuální testování bylo zajištěno osobní tvorbou konceptuálních modelů a jejich následnou kontrolou syntaktickým validátorem a také stejným využíváním členy modelovacího týmu v průběhu vývoje programu na rozpracovaných verzích.

Automatizované testy byly vytvořeny v podobě **unit testů** s JavaScriptovou knihovnou **Sinon**. **Unit testy** byly vytvořeny pro všechny funkce v programu. Automatizované testy jsou součástí programu v příloze této bakalářské práce.

9.3.4.3 Rozšiřitelnost a modifikovatelnost

Snadná rozšiřitelnost a modifikovatelnost je zajištěna korektním rozdělením tříd podle funkcí do správných souborů, podrobnou dokumentací každé funkce v programu, dokumentací kódu ve složitějších funkcích a strukturováním kódu tak, že každá funkce má právě jednu funkcionalitu.

Soubor "**main.js**" se stará o spuštění programu a nastavení některých proměnných. Soubor "**filePreparation.js**" zajišťuje zpracování vstupních dat do podoby, se kterou může zbytek programu jednoduše pracovat. Dále "**generatedParser.js**"space je vygenerovaná část kódu za pomoci PEG.js nástroje (více viz 7). Tato část obstarává tokenizaci vstupních dat. Funkce z "**validators.js**" kontrolují korektnost tokenizovaných částí vstupu, a pokud naleznou chybu, zavolají funkce ze souboru "**errorMsgs.js**", které zajišťují korektní výstup programu.

V poslední řadě soubor "**namesMEA.js**" obsahuje všechny používané názvy MEA anotací (více viz 8.5.3), tedy lze jednoduše zkontrolovat, které MEA anotace program rozeznává, a případně další přidat nebo nějaké odebrat.

9.3.4.4 Bezpečnost

Syntaktický validátor běží bez přístupu na internet na přístroji uživatele. Nehrozí tedy žádná přímá nebezpečí citlivým datům, které by mohl uživatel zpracovávat.

Kapitola 10
Závěr

V bakalářské práci jsem se nejprve zabýval ontologickou analýzou klíčových domén a jejich datových sad. Poté navazovala tvorba ontologických konceptuálních modelů a následně jejich propojení s příslušnými datovými sadami pomocí tvorby mapovacích pravidel. Posledním cílem této bakalářské práce byla implementace syntaktického validátoru pro kontrolu mapovacích pravidel pomocí rozšíření již existujícího parseru.

Prvnímu výstupu této bakalářské práce předcházela důkladná ontologická analýza domén a jejich datových sad, která spočívala ve vymezení obecného obsahu, specifikaci případných odborných názvů datových sad a nalezení způsobu propojení ontologického konceptuálního modelu s datovou sadou.

Díky ontologické analýze bylo možné vytvořit ontologické konceptuální modely. Tyto modely byly vytvořeny pomocí nástroje OpenPonk v modelovacím jazyce OntoUML. Do těchto modelů musely být přidány speciální konstrukty zvané Data entity.

Po zkonstruování kompletního ontologického konceptuálního modelu, byla do Data entit vložena mapovací pravidla. Tato pravidla zaručují propojení modelů s datovými sadami.

Posledním výstupem této bakalářské práce je syntaktický validátor mapovacích pravidel. Pro implementaci byl použit programovací jazyk Javascript a generátor parserů v Javascriptu PEG.js. Pro vygenerování parseru pomocí PEG.js bylo potřeba vytvořit gramatiku. Tato gramatika byla převzata z již existujícího parseru a byla rozšířena pro rozpoznání nových mapovacích pravidel a upravena pro přívětivější strukturu tokenizovaných dat. Tato vygenerovaná část kódu se používá ke kontrole obecné struktury mapovacích pravidel a tokenizaci vstupních dat. Dále bylo implementováno několik syntaktických kontrol mapovacích pravidel a kontroly existence a tvaru identifikátorů datových entit. Jako vstup tento program přijímá celý konceptuální model ve formátu JSON, který je možný jednoduše exportovat z nástroje OpenPonk, a výstupem je detailní popis charakteru chyb a jejich lokací.

Aktuálně je syntaktický validátor mapovacích pravidel samostatný program. V budoucnu by bylo dobré ho implementovat přímo do nástroje OpenPonk pro zjednodušení a zrychlení jeho využívání.

Bibliografie

1. HOFWEBER, Thomas. Logic and Ontology. In: ZALTA, Edward N.; NODELMAN, Uri (ed.). *The Stanford Encyclopedia of Philosophy* [`https://plato.stanford.edu/archives/sum2023/entries/logic-ontology/`]. Metaphysics Research Lab, Stanford University, 2023.
2. GUIZZARDI, Giancarlo; WAGNER, Gerd. A Unified Foundational Ontology and some Applications of it in Business Modeling. In: *CAiSE Workshops (3)*. 2004, s. 129–143.
3. BREITMAN, Karin Koogan; CASANOVA, Marco Antonio; TRUSZKOWSKI, Walter. Ontology in computer science. *Semantic Web: Concepts, Technologies and Applications*. 2007, s. 17–34.
4. GRUBER, Thomas R. A translation approach to portable ontology specifications. *Knowledge acquisition*. 1993, roč. 5, č. 2, s. 199–220.
5. F., Sowa John. *Principles of Ontology* [online]. [cit. 1997-03-12]. Dostupné z: `http://www-ksl.stanford.edu/onto-std/mailarchive/0136.html`.
6. GUARINO, Nicola; CARRARA, Massimiliano; GIARETTA, Pierdaniele. An ontology of meta-level categories. In: *Principles of Knowledge Representation and reasoning*. Elsevier, 1994, s. 270–280.
7. PERGL, Robert. *OntoUML: Základy, rigidní a anti-rigidní sortály* [online]. Department of Software Engineering Faculty of Information Technology Czech Technical University in Prague. [cit. 2021-05-02]. Dostupné z: `https://courses.fit.cvut.cz/BI-KOM/slides/html/lectures-czech/02-ontouml-basics-sortals.htm`.
8. GUARINO, Nicola. *Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy*. Sv. 46. IOS press, 1998.
9. HOEHNDORF, Robert. What is an upper level ontology? *Ontogenesis*. 2010.
10. NARDI, Julio Cesar; ALMEIDA FALBO, Ricardo de; ALMEIDA, João Paulo A; GUIZZARDI, Giancarlo; PIRES, Luis Ferreira; SINDEREN, Marten J van; GUARINO, Nicola. Towards a commitment-based reference ontology for services. In: *2013 17th IEEE International Enterprise Distributed Object Computing Conference*. IEEE, 2013, s. 175–184.
11. SMITH, Barry. On classifying material entities in Basic Formal Ontology. 2012.
12. GANGEMI, Aldo; GUARINO, Nicola; MASOLO, Claudio; OLTRAMARI, Alessandro; SCHNEIDER, Luc. Sweetening ontologies with DOLCE. In: *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web: 13th International Conference, EKAW 2002 Sigüenza, Spain, October 1–4, 2002 Proceedings 13*. Springer, 2002, s. 166–181.

13. NILES, Ian; PEASE, Adam. Towards a standard upper ontology. In: *Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001*. 2001, s. 2–9.

14. MCDANIEL, Melinda; STOREY, Veda C. Evaluating domain ontologies: clarification, classification, and challenges. *ACM Computing Surveys (CSUR)*. 2019, roč. 52, č. 4, s. 1–44.

15. GUIZZARDI, Giancarlo; BOTTI BENEVIDES, Alessander; FONSECA, Claudenir M.; PORELLO, Daniele; ALMEIDA, João P. A.; PRINCE SALES, Tiago. UFO: Unified Foundational Ontology. *Applied ontology*. 2022, roč. 17, č. 1, s. 167–210. ISBN 1570-5838.

16. GUIZZARDI, Giancarlo; WAGNER, Gerd; ALMEIDA, João Paulo Andrade; GUIZZARDI, Renata SS. Towards ontological foundations for conceptual modeling: The unified foundational ontology (UFO) story. *Applied ontology*. 2015, roč. 10, č. 3-4, s. 259–271.

17. QUATRANI, Terry; EVANGELIST, UML. Introduction to the Unified modeling language. *A technical discussion of UML*. 2003, roč. 6, č. 11, s. 03.

18. CLARK, Anthony; EVANS, Andy. Foundations of the Unified Modeling Language. In: *Proceedigs of the 2nd Northern Formal Methods Workshop*. Springer, 1997.

19. PADMANABHAN, Bharath. Unified Modeling Language (UML) Overview. *EECS810–Principles of Software Engineering*. 2012.

20. PERGL, Robert; SALES, Tiago Prince; RYBOLA, Zdeněk. Towards OntoUML for software engineering: from domain ontology to implementation model. In: *Model and Data Engineering: Third International Conference, MEDI 2013, Amantea, Italy, September 25-27, 2013. Proceedings 3*. Springer, 2013, s. 249–263.

21. GUIZZARDI, Giancarlo; FONSECA, Claudenir M; BENEVIDES, Alessander Botti; ALMEIDA, João Paulo A; PORELLO, Daniele; SALES, Tiago Prince. Endurant types in ontology-driven conceptual modeling: Towards OntoUML 2.0. In: *Conceptual Modeling: 37th International Conference, ER 2018, Xi'an, China, October 22–25, 2018, Proceedings 37*. Springer, 2018, s. 136–150.

22. M., Suchánek. *OntoUML specification* [online]. [cit. 2022-11-09]. Dostupné z: `https://ontouml.readthedocs.io/en/latest/index.html`.

23. PERGL, Robert. *OntoUML: Vztahy Celek-Část, typy agregací; Aspekty* [online]. Department of Software Engineering Faculty of Information Technology Czech Technical University in Prague. [cit. 2021-05-02]. Dostupné z: `https://courses.fit.cvut.cz/BI-KOM/slides/html/lectures-czech/04-ontouml-aggr-aspects.htm`.

24. GUIZZARDI, Giancarlo. Ontological foundations for structural conceptual models. 2005.

25. GUIZZARDI, Giancarlo. Ontological meta-properties of derived object types. In: *Advanced Information Systems Engineering: 24th International Conference, CAiSE 2012, Gdansk, Poland, June 25-29, 2012. Proceedings 24*. Springer, 2012, s. 318–333.

26. RYBOLA, Zdeněk; PERGL, Robert. Towards OntoUML for software engineering: transformation of rigid sortal types into relational databases. In: *2016 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2016, s. 1581–1591.

27. M., Suchánek. *OMG Unified Modeling Language (OMG UML) v2.5* [online]. [cit. 2023-06-15]. Dostupné z: `https://www.omg.org/spec/UML/2.5/PDF`.

28. GUIZZARDI, Giancarlo. The problem of transitivity of part-whole relations in conceptual modeling revisited. In: *Advanced Information Systems Engineering: 21st International Conference, CAiSE 2009, Amsterdam, The Netherlands, June 8-12, 2009. Proceedings 21*. Springer, 2009, s. 94–109.

29. BENEVIDES, Alessander Botti; GUIZZARDI, Giancarlo. A Model-Based Tool for Conceptual Modeling and Domain Ontology Engineering in OntoUML. In: FILIPE, Joaquim; CORDEIRO, José (ed.). *Enterprise Information Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, s. 528–538. ISBN 978-3-642-01347-8.

30. CCMI. *OpenPonk* [online]. 2016. [cit. 2023-06-20]. Dostupné z: `https://ccmi.fit.cvut.cz/nastroje/openponk/`.

31. CVUT, FIT. *OpenPonk modeling platform* [online]. 2022. [cit. 2023-06-20]. Dostupné z: `https://openponk.org/`.

32. D., Majda. *PEG.js Parser Generator for JavaScript* [online]. 2023. [cit. 2023-06-20]. Dostupné z: `https://pegjs.org/`.

33. *Český statistický úřad* [online]. 2023. [cit. 2023-06-21]. Dostupné z: `https://www.czso.cz/`.

34. *Česká správa sociálního zabezpečení - Otevřená data* [online]. 2023. [cit. 2023-06-21]. Dostupné z: `https://data.cssz.cz/zakladni-info`.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF