

Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky



**Slučování modelů PURO pro tvorbu ontologií
v OWL**

DIPLOMOVÁ PRÁCE

Studijní program: Aplikovaná informatika

Studijní obor: Znalostní a webové technologie

Autor: Bc. Daniel Bedrníček

Vedoucí diplomové práce: Ing. Marek Dudáš, Ph.D.

Praha, prosinec 2021

Prohlášení

Prohlašuji, že jsem diplomovou práci „Slučování modelů PURO pro tvorbu ontologií v OWL“ vypracoval samostatně za použití v práci uvedených pramenů a literatury.

V Praze dne 6. prosince 2021

Bc. Daniel Bedrníček

Poděkování

Tímto bych rád poděkoval Ing. Marku Dudášovi, Ph.D. za odborné vedení této diplomové práce, a také za jeho trpělivost, rady a cenné připomínky k obsahové struktuře této práce. Dále bych rád poděkoval své rodině a přítelkyni Zuzaně Štěpánkové za podporu a zázemí při psaní této práce.

Abstrakt

Práce se zabývá možnostmi rozšíření funkcionality PURO Modeleru vyvinutého pro tvorbu modelů v jazyce PURO. Cílem práce je prozkoumání možností vývoje aplikace podporující návrh složitějších modelů v několika ohledech. Zásadní funkcionalitou je možnost slučování dílčích modelů vytvořených v jazyce PURO a také možnosti zlepšení přehlednosti zobrazení výsledného sloučeného PURO modelu. Práce čtenáře seznamuje s oblastí ontologií a sémantického webu. Jsou popsány postupy mapování a slučování ontologií a ER modelů. Tyto postupy jsou pak vztahovány na problematiku slučování PURO modelů. Autor identifikuje klíčové problémy při slučování všech druhů entit v jazyce PURO a navrhuje možná řešení. Pro přehlednější zobrazování modelů se autor inspiruje tématem vizualizace velkých grafů a hledá možné využití obecných postupů pro využití ve své aplikaci. Autor dále porovnává několik knihoven pro zobrazování grafů ve webové aplikaci. Ve vybrané knihovně poté implementuje navržené algoritmy pro slučování modelů. Poté je demonstrována funkčnost aplikace vytvořením dílčích modelů na základě ontologie FOAF a jejich následným sloučením v jeden větší model.

Klíčová slova

entity relationship model, grafické modelování, ontologické inženýrství, OWL, PURO.

Abstract

Diploma thesis describes possibilities of expanding current functionalities of PURO Modeler developed for creation of models in PURO language. Goal of the thesis is to describe a possible development of application for creating more extensive models. Key feature is merging of several smaller models described in PURO language and displaying merged model in a well-arranged way. Thesis describes an area of ontology and semantic web. Mapping and merging of ontologies and entity relationship models is described and these principles are then applied on merging of PURO models. Author identifies main problems with merging of PURO models and describes a solution. Methods for visualization of large graphs are described. Several methods are identified to aid with a visualization of large PURO models. Author then compares several libraries used to visualize graphs and chooses one for implementation of the web application. Functionality of the application is then demonstrated by recreating FOAF ontology in several smaller PURO models which are then merged into one larger model.

Keywords

Entity relationship model, ontology engineering, OWL, PURO, visual modeling.

Obsah

Úvod	11
1 Ontologie a sémantický web	12
1.1 Ontologie	14
1.2 RDF	14
1.3 RDFS	15
1.4 OWL	16
1.5 PURO	17
2 Existující přístupy ke slučování modelů.....	18
2.1 Mapování ontologií	18
2.1.1 Možné postupy mapování.....	18
2.2 Sloučení ontologií.....	20
2.3 Párování instancí	25
2.4 Slučování Entity Relationship modelů	26
3 Vizualizace velkých grafů	31
3.1 Použitelná rozložení	31
3.1.1 Rozložení Force-directed.....	32
3.1.2 Shlukové rozložení	32
3.1.3 Lineární rozložení.....	33
3.1.4 Kruhové rozložení	33
3.1.5 Maticové rozložení	33
3.1.6 Stromové rozložení.....	34
3.2 Doporučené techniky.....	35
4 Problematika slučování v PURO.....	36
4.1 Mapování termů	38
4.1.1 Mapování relací	38
4.1.2 Mapování atributů	39
4.1.3 Mapování instancí	39
4.2 Slučování instancí.....	40
4.3 Skóre podobnosti entit.....	42
4.4 Okolí uzlu	43
4.5 Nalezení souvislého grafu	45
5 Grafická reprezentace	47
5.1 Skupiny.....	47

5.2 B-typy	51
6 Dostupné knihovny pro grafickou reprezentaci.....	53
6.1 Požadavky	53
6.1.1 Podpora různých grafických elementů	53
6.1.2 Podmiňování tvorby vztahů.....	53
6.1.3 Kontextová menu	54
6.1.4 Složky a skupiny	54
6.1.5 Formáty ukládání.....	54
6.2 Knihovny	54
6.2.1 mxGraph.....	54
6.2.2 VisJS.....	55
6.2.3 GoJS	55
6.2.4 Srovnání knihoven.....	57
7 Návrh algoritmů pro slučování PURO Modelů.....	58
7.1 Prerekvizity slučování	59
7.1.1 Import.....	60
7.1.2 Mapování.....	61
7.2 Slučování.....	62
7.2.1 Analýza.....	62
7.2.2 Návrh.....	62
8 Implementace uživatelského rozhraní	76
9 Případová studie	81
Závěr.....	88
Použitá literatura.....	90

Seznam obrázků

Obr. 1: Neformální grafická reprezentace RDF.	15
Obr. 2: Příklad mapování dvou ontologií [10].	18
Obr. 3: Příklady slučování: a) plné, b) podle cíle, c) podle zdroje, d) expertní [10].	21
Obr. 4: Příklad relace mezi b-typy.	22
Obr. 5: Přenos relace na podtyp.	23
Obr. 6: Příklad hierarchie v PURO.	23
Obr. 7: Příklad dědičnosti relací.	24
Obr. 8: Přenos relací na instance.	25
Obr. 9: Elementy ER diagramů [17].	27
Obr. 10: Příklad kompozice.	28
Obr. 11: Příklad modelů (M2, M4 a M7) z různých zdrojů [18].	28
Obr. 12: Protkávání modelu M7 do modelů M2 a M4 [18].	29
Obr. 13: Slučování sad [19].	29
Obr. 14: Homomorfismy při slučování dvou grafů [19].	30
Obr. 15: Neuspořádané zobrazení velkého grafu [21].	31
Obr. 16: Force-directed rozložení [21].	32
Obr. 17: Shlukové rozložení [21].	32
Obr. 18: Lineární rozložení [21].	33
Obr. 19: Kruhové rozložení [21].	33
Obr. 20: Maticové rozložení [21].	34
Obr. 21: Stromové rozložení [21].	34
Obr. 22: Model nabídky notebooku.	36
Obr. 23: Model nabídky telefonu.	36
Obr. 24: Model popisující člověka.	37
Obr. 25: Sloučení tří modelů.	37
Obr. 26: Příklad shodných relací.	39
Obr. 27: Příklad sloučení instancí do obecné entity.	40
Obr. 28: Sloučení instancí a linků k relacím.	41
Obr. 29: Příklad cyklického vztahu.	41
Obr. 30: Příklad okolí uzlu.	43
Obr. 31: OBM popisující nabídku telefonu.	44
Obr. 32: OBM popisující nabídku notebooku.	44
Obr. 33: Propojení složek přes různé entity.	47
Obr. 34: Propojení složek přes shodnou entitu.	47
Obr. 35: Příklad rozbalené složky.	48
Obr. 36: Nabídka minimalizace.	48
Obr. 37: Minimalizace části grafu.	49
Obr. 38: Minimalizace odchozích propojení.	49
Obr. 39: Vztahy entit mezi složkami.	50
Obr. 40: Vztahy entit po zabalení složek.	50
Obr. 41: Zabalení všech entit do složek.	51
Obr. 42: Vyjádření shodnosti b-typu.	52
Obr. 43: Vyjádření shodnosti b-typu mezi zabalenými složkami.	52

Obr. 44: Schéma importu, mapování a slučování OBM	58
Obr. 45: Výběr prahu míry textové podobnosti.....	59
Obr. 46: Přenos relací na instance. (kopie Obr. 8)	63
Obr. 47: Příklad mapování dvou ontologií [10]. Kopie Obr. 2	63
Obr. 48: Hierarchie b-typů	65
Obr. 49: Příklad potenciálního smazání b-typu v hierarchii.....	67
Obr. 50: Situace přes sloučením podle zdroje	68
Obr. 51: Proces odstranění uzlu v grafu	68
Obr. 52: Složitější model produktů	69
Obr. 53: Jednodušší model produktů.....	69
Obr. 54: Výsledné sloučení modelů produktů.....	70
Obr. 55: Redundantní hrany grafu.....	70
Obr. 56: Ilustrace redundantních cest grafem.....	74
Obr. 57: Náhled grafického prostředí aplikace.....	76
Obr. 58: Implementace vyhledávání entit	76
Obr. 59: Grafická implementace mapování dvou modelů.....	77
Obr. 60: Výběr shodné entity	77
Obr. 61: Shlukování entit a kontextové menu	78
Obr. 62: Minimalizované shluky entit.....	78
Obr. 63: Graf s nepropojenými částmi	79
Obr. 64: Nepropojené části grafu ve shlucích	79
Obr. 65: Model popisující organizace a skupiny	83
Obr. 66: Model popisující osoby	83
Obr. 67: Model vztahů mezi osobou a dalšími entitami.....	84
Obr. 68: Náhled slučování dvou modelů	84
Obr. 69: Příklad kontextového menu.....	85
Obr. 70: Výsledek sloučení tří modelů.....	86
Obr. 71: Výsledný model s použitím obecných instancí	87

Seznam zkratek

API	Application Programming Interface. Určuje způsob volání funkcí knihovny.
ER model	Entity relationship model. Popis entit a vztahů mezi nimi.
FOAF	Friend of a friend. Ontologie popisu osob, jejich aktivit a vztahů k lidem a objektům.
HTML	Hypertext Markup Language. Značkovací jazyk pro tvorbu webových stránek.
ID	Identifikátor určující jedinečnost záznamu.
IRI	Internationalized Resource Identifier. Slouží k jednoznačné identifikaci zdroje.
JSON	JavaScript Object Notation. Formát pro přenos dat agregovaných v polích, či objektech.
OBM	Ontology Background Model. Model vytvořený v jazyce PURO.
OWL	Web Ontology Language je jazyk pro tvorbu ontologií. Rozšiřuje RDFS.
PURO	Particular - Universal, Relationship - Object. Jazyk pro podporu vývoje ontologií.
RDF	Resource Description Framework. Specifikace pro modelování informací ve strojově i lidsky čitelné podobě.
RDFS	RDF Schema. Slouží pro rozsáhlejší podporu definic v RDF.
Tf-idf	Term frequency–inverse document frequency. Metodika hodnocení relevance při vyhledávání textu.
URL	Uniform Resource Locator. Přímý popis způsobu, jakým se lze dostat ke zdroji (např. webová stránka).
XML	Extensible Markup Language. Značkovací jazyk pro strojově čitelné kódování dokumentů.

Úvod

Jednou ze základních struktur sémantického webu jsou ontologie, které slouží jako schémata k popisu dat. Tvorba ontologií běžně probíhá přímo v jejich zdrojovém formátu. Různé formy vizualizace takto vytvářených struktur respektují tento formát a při návrhu tak neexistuje jednoduchá možnost určité formy abstrakce usnadňující tvůrce ontologie orientaci v doméně, kterou se snaží ontologií popisovat. Aplikace PURO Modeler, navržená na principech jazyka PURO, slouží jako nástroj pro usnadnění vývoje ontologií. Jazyk PURO na rozdíl od běžně používaného OWL, který je jedním z jazyků pro návrh ontologií, nevynucuje mnohá omezení. Tvůrce ontologie může model navrhovat v mnohem přirozenějších strukturách a získat lepší přehled o doméně, kterou popisuje. Po vývoji v aplikaci PURO Modeler pak může uživatel převést celý model do jazyka OWL a tím vytvořit určitý základ budované ontologie.

Autor PURO Modeleru ve své disertační práci zmiňuje určitá omezení aplikace. Problémem je zejména nepřehlednost při návrhu větších modelů. Ve zobrazeném grafu je od určitého počtu uzlů obtížnější orientace znemožňující tvorbu rozsáhlejších struktur. Je proto nutné rozšířit aplikaci o funkcionality, které by zpřehlednily vytvářený model a umožnily návrh na sobě nezávislých částí. Následným sloučením těchto částí by vznikl kompletní model s potenciálně menším množstvím chyb, než kolik by jich jinak mohlo vzniknout ve velkém a nepřehledném shluku uzlů. Takto sloučený model by pak bylo možné převést do jazyka OWL.

Hlavním cílem diplomové práce je analýza, návrh a implementace aplikace sloužící jako demonstrace postupů slučování modelů vytvořených v jazyce PURO. Součástí práce je rozbor problematiky slučování PURO modelů a vyvozování souvislostí s obecným slučováním a mapováním ontologií. Aplikace má za cíl rozvinutí funkcionalit původního PURO Modeleru [1]. Aplikace by měla rozšiřovat možnosti původní aplikace o slučování individuálně vyvinutých modelů a reimplementovat uživatelské rozhraní pro vývoj modelu na základě funkcionalit PURO Modeleru. Dalším cílem je navázat na původní práci v oblasti využití technik vizualizace velkých grafů pro lepší přehlednost při vývoji rozsáhlejších PURO modelů. Výsledkem by měl být přehledný sloučený model, který je možné exportovat do původní aplikace pro převod do jazyka OWL. Práce si neklade za cíl celkové nahrazení původní aplikace, protože nebude reimplementovat nástroj OBOWL Morph, který slouží pro převod PURO modelu do OWL.

1 Ontologie a sémantický web

Sémantický web je koncept rozšiřující funkcionalitu World Wide Web o možnost poskytování strojově čitelných informací pomocí standardizovaných postupů spravovaných organizací World Wide Web Consortium (W3C).

Základním způsobem poskytování dat na WWW je sdílení HTML dokumentů. Tyto dokumenty obsahují data strukturovaná značkovacím jazykem HTML. Pokud je dokument zobrazen v kompatibilním prohlížeči, je uživateli nabídnut lidsky čitelný obsah. Takový obsah je sice srozumitelný pro člověka, ale naprosto nevhodný pro strojové zpracování.

Obsah je možné sdílet pomocí různých dalších formátů, jako je JSON, či XML, avšak pro jejich strukturu není žádný univerzální standard. Vývojář se pak musí vždy seznámit s podobou dat z každého jednotlivého zdroje. Z toho vyplývajícím problémem pak je nekonzistence dat napříč různými zdroji. Napojování takových dat je pak pokaždé unikátní a nelze pravděpodobně použít znova.

Proto má spousta služeb poskytující API vlastní dokumentaci a příklady pro použití různých dotazů. Často taky obsahují příklady výstupů, aby se vývojář mohl seznámit s použitým formátem a byl schopen tento výstup zakomponovat do svého kódu.

Pro demonstraci rozdílnosti výsledků následuje příklad zkrácených výstupu dvou rozdílných API při dotazu na význam slova „mask“. Volba slova z příkladu je omezena webem rapidapi.com, kde je bez členství možné zobrazit pouze jeden předdefinovaný dotaz [2]:

První příklad datové struktury definice slova¹.

```
{"word": "mask",
  "results": [{"definition": "a party of guests wearing costumes
and masks",
    "partOfSpeech": "noun",
    "synonyms": ["masque", "masquerade", "masquerade party"],
    "typeOf": ["party"],
    "hasTypes": ["fancy-dress ball", "masked ball", "masquerade
ball"]},
   {"definition": "shield from light",
    "partOfSpeech": "verb",
    "synonyms": ["block out"],
    "typeOf": ["cover"]}],
  "pronunciation": {"all": "mæsk"},
  "frequency": 4.29}
```

¹Získané z API dostupné na: <https://www.wordsapi.com/>

Další příklad nabízí informace odděleně na základě samostatných dotazů². Uživatel se tedy musí zvlášť dotázat na synonyma, části a další informace o slově [3].

Dotaz na asociace ke slovu „mask“:

```
"assoc_word": ["hide", "hat", "face"]
"assoc_word_ex": ["hide", "hat", "face", "veil", "disguise", "camoufl
age"]
```

Dotaz na definici slova “mask”:

```
"noun": "(nou) a covering to disguise or conceal the face
(nou) activity that tries to conceal something
(nou) a party of guests wearing costumes and masks
(nou) a protective covering worn over the face"
"verb": "(vrb) hide under a false appearance
(vrb) put a mask on or cover with a mask
(vrb) make unrecognizable
(vrb) cover with a sauce
(vrb) shield from light"
```

Z příkladů je zřejmé, že zdroj informací určuje následnou podobu zpracovávání dat a není jednoduše možné tyto zdroje kombinovat. Při každém novém zdroji je třeba lidský vstup a úprava kódu. Tvůrce aplikace je pak nucen prostudovat dokumentaci a sledovat případné úpravy ve formátu dat na straně zdroje [3].

Cílem sémantického webu tak, jak ho představil Tim Berners-Lee se spolupracovníky je možnost publikování dat ve strojově čitelné podobě za použití standardů. Tím by bylo možné získávat data univerzálním postupem bez rizika rozdílné struktury napříč zdroji. S tím se pojí možnost relativně jednoduchého slučování takových dat. Uživatel by pak nebyl odkázán na mnoho různých dokumentací a stačilo by mu znát principy ontologií, které jsou ve své podstatě neměnné. Další výhodou sémantického webu je používání identifikátorů. Každý popsaný objekt je pak popsán jednoznačně a je možné se na něj odkazovat.

Sémantický web tedy stojí na principu uceleného formátu, který umožňuje jednoznačně označit entitu. Tou může být například konkrétní člověk, produkt, nebo i abstraktní koncept. Dalším krokem je pak propojování těchto entit pomocí vztahů. Tím se vytváří síť, ze které lze vyvzakovat strukturu popisované skutečnosti a je tak možné opětovně používat ty samé zdroje a koncepty bez vytváření redundancí. Dalším důležitým aspektem je pak možnost publikování těchto dat a jejich sdílení s ostatními. S tím souvisí opětovné získávání těchto dat pomocí dotazovacího jazyka [4].

² Získané ze zdroje dostupného na: <https://rapidapi.com/twinword/api/word-dictionary/>

1.1 Ontologie

Ontologie je ve své podstatě popis určité domény. Snahou je formální zápis zvolené výseče reálného světa s jeho entitami a vztahy. Tím je vytvořena zjednodušená a strojově čitelná struktura vybrané části reality v rozsahu potřebném pro využití v informačních systémech.

Ontologie popisují koncepty a vztahy zahrnuté v popisované doméně. Pojmy jsou charakterizovány pro smysluplný rozsah použití a zahrnují možné vztahy a případná omezení. Obecně lze pro ontologii použít výraz „slovník“. Neexistuje přesná definice, která by oddělovala slovník od ontologií, ale obecně je přijímáno, že ontologie je rigidnější a komplexnější struktura, kdežto slovník je volnější.

Ontologie je použitelná pro organizování znalostí a pro udržování terminologie napříč různými datasety.

V oblasti zdravotnictví může lékař využít ontologie pro popis nemocí, různých symptomů a způsobu léčby. Farmaceutická firma může takto reprezentovat různé léky, jejich interakce a vedlejší účinky. Kombinací těchto ontologií je pak možné na základě údajů o pacientovi vhodně nabízet léčbu.

Příkladem udržování terminologie může být knihovní systém, který udržuje strukturu popisu znalostí pro vhodnou organizaci dat. Například různé zdroje dat mohou popisovat autora knihy jako „autor“, či „tvůrce“. Pokud ontologie popisuje vztah mezi těmito dvěma pojmy, je možné do výsledné databáze zahrnout data z obou zdrojů bez ztráty významu [5].

1.2 RDF

Resource Description Framework RDF slouží pro popis skutečností z výseče reálného světa a vztahů mezi nimi. Propojením těchto skutečností vzniká grafová struktura, kdy entity a literály tvoří uzly grafu a vztahy jsou hranami grafu. Základním konceptem v RDF je triplet, tedy trojice. Ten popisuje vztah – predikát mezi dvěma entitami: subjektem a objektem. Objekt pak může být i literál, tedy například číselná hodnota, textový řetězec, datum.

<subjekt> <predikát> <objekt>

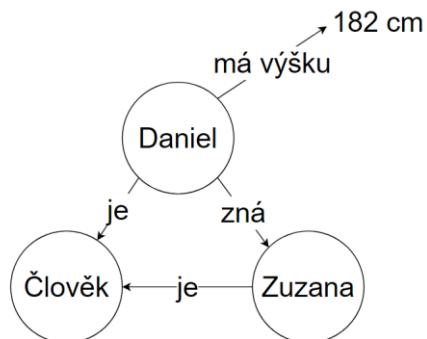
Pro popis těchto skutečností slouží IRI (Internationalized Resource Identifier) – unikátní identifikátor ukazující na zdroj. Synonymem zdroje je „entita“. Tato entita může být cokoli, jak bylo výše popsáno v kapitole 1. Zdroj může být zjednodušeně chápán jako koncept velmi podobný URL [6].

Kombinací tripletů je možné vytvářet poměrně složité vztahy mezi entitami a literály. Příkladem může být jednoduchý popis vztahu dvou osob. Popsaný neformálně bez použití IRI:

```
<Daniel> <je> <člověk>
<Zuzana> <je> <člověk>
<Daniel> <má výšku> <182 cm>
<Daniel> <zna> <Zuzana>
```

Na příkladu lze pozorovat použití literálu pro popis výšky. Objekt „182 cm“ nelze jednoznačně identifikovat, jelikož se jedná o abstraktní koncept. Zároveň z příkladu vyplývá, že jedna

entita může být použita jako subjekt i jako predikát, a to ve více tripletech. Takovýto vztah lze popsat i vizuálně jako graf s uzly a hranami (viz Obr. 1).



Obr. 1: Neformální grafická reprezentace RDF.

V tomto neformálním popisu na Obr. 1 zatím nebylo použito IRI. Bez unikátního identifikátoru by velmi rychle mohla nastat situace, kdy budou mít dvě entity stejné jméno. Pokud bude ontologie popisovat dvě osoby se jménem Daniel, je třeba tyto osoby odlišit. Bez toho by například nebylo možné vyjádřit, která z osob zná jinou osobu. IRI navíc nemusí sloužit pouze jako unikátní identifikátor, ale i jako popis obecných vlastností entity. Lze například vyjádřit, že Daniel žije v Praze. Praha pak díky IRI nemusí být pouhým textovým popisem. Každý uživatel ontologie nemusí být třeba seznamen s tím, co slovo Praha znamená. IRI odkazující se na město Praha pak může sloužit pro rozšíření vyjadřovací schopnosti ontologie přes odkaz na informace o entitě. Tento odkaz popisuje například plochu města, nebo počet obyvatel:

```
<http://example.org/Dan>
<žije_v>
<https://dbpedia.org/page/Prague>
```

1.3 RDFS

V samotném RDF lze popsat rozsáhlá fakta o entitách. Ze vztahů mezi nimi však nelze automaticky v podstatě nic vyvzakovat. Pro rozsáhlejší podporu definic v RDF bylo vyvinuto RDFS, tedy RDF Schema [7]. Hlavní koncepty používané v RDFS jsou následující popsány v Tab. 1.

Tab. 1: Hlavní konstrukty v RDFS a jejich popis [7].

Konstrukt	Popis
Class	Class neboli třída, slouží pro vyjádření obecniny, na kterou se mohou ostatní entity odkazovat. Příklady výše lze pak například rozšířit o fakt, že člověk je třída.
Property	Slouží k popisu vlastnosti. Například predikát „zná“ je vlastností.
Type	Vyjadřuje, že entita je instancí nějaké třídy, či vlastnosti. Daniel je člověk. Je tedy instancí třídy člověk.
SubClassOf	Koncept podtříd slouží k vyjádření hierarchie mezi třídami. Může existovat třída „muž“. Je nasnadě, že každý muž je zároveň člověkem a dědí všechny charakteristiky a vlastnosti člověka.
SubPropertyOf	Vyjadřuje podobný koncept jako „subClassOf“. Například vlastnost „je dobrý kamarád“ je rozšířením vlastnosti „je kamarád“.
Domain	Doména slouží pro rozšíření popisu vlastnosti. Defacto popisuje, jaká entita může být subjektem ve vztahu. Například vztah „je autor“ má doménu „člověk“.
Range	Je opakem domény. Je možné přeložit ho jako „rozsah“ Popisuje, jaká entita může být objektem ve vztahu. Vztah „je autor“ pak má rozsah „kniha“.

Díky těmto konstruktům lze popisovat mnohem více skutečností a vyvozovat z nich další fakta [7]. Pro názorný příklad poslouží následující neformální popis:

```
<člověk> <type> <class>
<člověk> <dýchá> <vzduch>
<muž> <subClassOf> <člověk>
<Daniel> <type> <muž>
```

Z výše uvedených tripletů lze díky podtřídě vyvodit, že každý muž je člověk. Člověk dýchá vzduch, a tedy i každý muž dýchá vzduch. Daniel tedy dýchá vzduch.

1.4 OWL

Owl (Web Ontology Language) je jazyk který rozšiřuje jazyk RDFS a slouží pro tvorbu ontologií. Jeho vyjadřovací schopnosti jsou v porovnání s RDFS bohatší například o popis disjunktnosti, či rovnosti dvou entit. V OWL je možné popisovat symetrii vztahů, či kardinalitu. Pro formální logiku, kterou OWL používá je možné odvozovat z původních dat nové znalosti [8].

1.5 PURO

PURO je jazykem pro podporu vývoje ontologií. Účelem jazyka je tvorba modelů nazývaných Ontology Background Model (OBM). Tyto modely by měly popisovat určitou oblast reálného světa na konkrétních příkladech „tak, jak jsou“. Samotný model však neslouží jako konečná ontologie, či schéma. Jazyk je navržen tak, aby umožnil vývojáři tvořit modely blízké OWL, avšak bez mnohých omezení. PURO umožňuje například tvořit třídy jako instance tříd a vytvářet vztahy s více než dvěma entitami, což přímo ve struktuře tripletů v OWL není možné. Návrhář si tak může rozmyslet strukturu budoucí ontologie bez těchto omezení na určitém konkrétním příkladu. Paralelou může být tvorba relačních databází, kde vývojář nejdříve navrhuje ER model, aby získal určitý nadhled nad modelovanou oblastí předtím, než započne samotnou implementaci. Vytvořený model (OBM) pak popisuje s určitou mírou granularity, tedy detailu, vybranou oblast reálného světa.

PURO rozlišuje mezi konkrétním (Particular) a obecným (Universal). Zároveň platí rozlišení mezi vztahy (Relationship) a objekty (Object). Z této logiky také vychází název jazyka. Tyto rozlišení vedou k dalším pravidlům. Obecniny mohou mít instance, ale jednotliviny ne. Objekty jsou entity se svou vlastní identitou. Jednoduše řečeno o nich „lze mluvit“. Vztahy jsou však abstraktním konstruktem a pokud je o nich třeba mluvit konkrétně, vždy dojde i na popis objektů, kterých se vztah týká. Nelze například, aby existoval vztah „přítel“ bez toho, aby se týkal dvou osob [1].

Jazyk popisuje následující typy, resp. termy:

- B-type: Obecný objekt. Ekvivalent k třídě v OWL
- B-object: Konkrétní objekt. Analogie instancí v OWL
- B-relationship: Konkrétní vztah. Analogie přiřazení „objektové vlastnosti“ v OWL
- B-valuation: Přiřazení textové, nebo kvantitativní vlastnosti třídě, či objektu. Ekvivalent přiřazení datové vlastnosti v OWL
- B-relation: Obecný vztah. Objektová vlastnost v OWL
- B-attribute: Obecné pojmenování kvantitativní, či textové vlastnosti. Datová vlastnost v OWL

Mezi termy mohou panovat určité vztahy. Důležité jsou zejména B-instanceOf a B-subtypeOf:

- B-instanceOf:
 - Vztah popisující, že konkrétní b-objekt nebo b-typ je instancí nějakého b-typu.
 - B-type se odkazuje na jiný b-type. Toto je instancování, což vede k existenci tří různých úrovní. Třída může být instancí pouze třídy vyšší úrovně.
- B-subtypeOf:
 - Je vztahem dvou tříd na stejně úrovni. Účelem je vyjádření granularity. Subtype, tedy podtyp, je konkrétnějším vyjádřením obecnější třídy. Například člověk je podtyp savce.

2 Existující přístupy ke slučování modelů

Pro slučování PURO modelů v současné době neexistuje žádné řešení. PURO Modeler byl navržen pro vytváření jednoduchých grafů, ze kterých jsou pak vytvářeny ontologie v OWL. Pro rozsáhlější modely není vhodný. Autor původního PURO Modeleru zmiňuje ve své disertační práci tato omezení a v diskusi uvažuje nad budoucími možnostmi rozšíření aplikace. V principu jsou PURO modely podobné ontologiím a entity relationship modelům. Jejich slučování se věnuje řada prací a již navrženými postupy by bylo možné se inspirovat pro tvorbu aplikace pro slučování PURO Modelů.

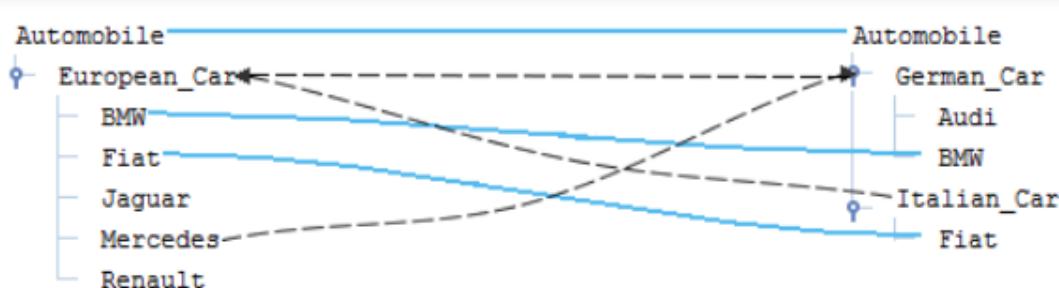
Mapování ontologií se obecně věnuje především nalézání shodných entit bez toho, aby nutně docházelo ke slučování. V ontologiích se povětšinou řeší podobnost tříd, které tvoří hlavní strukturu. Ve slučování PURO Modelů je však cílem sloučení nejen tříd, reprezentovaných ekvivalentním termínem b-type, ale i samotných instancí. Přesto je však vhodné hledat inspirace právě ve vyhledávání shodných entit podobně, jako je tomu v mapování ontologií především pro slučování b-typů, které podobně jako třídy v ontologii tvoří určité hierarchie.

2.1 Mapování ontologií

Jedná se o proces, jehož cílem je provázání dvou a více ontologií přes určité spojitosti. Důvodem pro nutnost mapování je potřeba sloučení dvou systémů používajících různé ontologie. Ty jsou vytvářeny většinou nezávisle na sobě různými vývojáři. Proto není nijak zaručena naprostá návaznost. Vytváření ontologií je závislé na úhlu pohledu a na požadavcích aplikace, tedy na konečném praktickém využití. Domény – určitá oblast, která je popisována, nemusí být ve slučovaných ontologiích v úplném překryvu. Zároveň nemusí být shodná jejich granularita, tedy hloubka, do které je ontologie popisována. Jedna ontologie může například popisovat prodej aut. Druhá ontologie pak prodej jednotlivých dílů automobilu. Toto mapování pak zjevně není triviální proces [9].

2.1.1 Možné postupy mapování

Na Obr. 2 je příklad možné ontologie popisující stejnou doménu, avšak s různou hierarchií a s jinými entitami. Některé z entit jsou však shodné a ty je při mapování nutné nalézt [10].



Obr. 2: Příklad mapování dvou ontologií [10].

Mapování je založeno na množství metod, jejichž cílem je nalezení shodných entit. Jednou z kategorií těchto technik je vyhledávání shodných entit bez ohledu na vztahy s okolím:

- Porovnávání textového řetězce – názvu.
- Sémantické porovnávání.
- Porovnávání atributů, typu...
- Napojení na zdroj – jinou ontologii

Dalším typem srovnávání je pak porovnávání entit z hlediska jejich vztahu s okolím. Takovou metodou je například grafová metoda, reprezentující ontologii jako graf. Srovnáváním hran a uzel dvou ontologií lze vyvozovat podobnosti.

Lze identifikovat pět hlavních úskalí při mapování ontologií:

- Koncepty mají různé názvy
- Jsou přítomny pouze v jedné z ontologií
- Mohou být podobné, ale ne shodné
- Mohou mít podobnou notaci, ale jinou sémantiku
- Ontologie nemusí popisovat automaticky předpokládaná fakta

Důležitým krokem je pak vyhodnocování shodnosti entit ve dvou různých ontologiích. V následující sekci je výčet vlastností entit využitelných pro porovnávání v kontextu této práce. Ke každé charakteristice je pak přiřazen algoritmus umožňující porovnávání:

- Lingvistické vlastnosti
 - Název (Levenštejnova vzdálenost, podobnost synonym)
 - Dokumentace: krátký textový popis entity (podobnost dokumentu)
- Strukturální vlastnosti
 - Hierarchie: Umístění entity v hierarchii (namepath)
 - Vztahy: Propojení s ostatními entitami (Názvy potomků)
 - Atributy: Vlastnosti entity (Názvy vlastností)

Popis výše uvedených algoritmů:

- **Levenštejnova vzdálenost:** Je definována jako nejnižší počet operací mezi dvěma textovými řetězci nutný k převodu jednoho řetězce na druhý. Operacemi může být odstranění, vložení a nahrazení znaku za jiný.
- **Podobnost synonym:** Měří, jak jsou si dvě entity podobné. Může vycházet z hierarchie Is-a, někdy označovanou jako ISA. Jedná se o hierarchii popisující vztahy mezi pojmy. Tyto vztahy jsou hyperonymie / hyponymie, holonyma / meronyma a vztahy mezi koncepty a objekty.
- **Podobnost dokumentu (Tf-idf):** Algoritmus hodnotí, zdali mají dvě entity dostatečně shodnou dokumentaci na základě hodnocení podobnosti těchto dokumentů. K tomu může sloužit metodika tf-idf vyhledávající četnost slov a jejich relevanci v dokumentech.
- **Namepath:** Slouží k vyhodnocení celkové cesty entity v grafu. V ontologii se jedná o cestu od kořene k entitě.
- **Názvy potomků:** Vyhodnocuje překryv názvu potomků dvou entit. Pokud je překryv potomků nad určitý prah, jsou entity vyhodnoceny jako shodné.

- **Názvy vlastností:** V principu se jedná o shodný algoritmus jako při porovnání názvu potomků [11].

Identifikace rozdílů mezi ontologiemi na úrovni modelu je možné rozdělit do následujících dvou kategorií:

- **Rozsah:** Dvě třídy mohou popisovat podobný koncept a mohou mít určitý překryv v instancích, ačkoliv se nejedná o přesně ty samé třídy [12]. Problém lze demonstrovat na příkladu zaměstnanců. Struktura může v jedné ontologii pracovat se zaměstnaneckou strukturou z pohledu jedné firmy (vlastní zaměstnanci, bývalí zaměstnanci), nebo se strukturou popisující externí pracovníky, konzultanty apod. [13].
- **Granularita a pokrytí:** Jedná se o rozdíl v tom, do jaké „hloubky“ ontologie při popisu zacházejí. Jedna z ontologií například může popisovat automobily, ale ne nákladní vozy. Další z nich popisuje nákladní vozy, ale pouze v několika málo kategoriích. Třetí pak popisuje nákladní vozy do mnohem větších detailů [14].

Další rozdělení je možné vyvodit podle modelovacích stylů:

- **Schéma:** Modely popisující podobnou oblast se mohou lišit ve způsobu, jakým jsou určité koncepty popisovány. Různé vztahy nemusí být shodně modelovány. Například určité události mohou být chápány jako periodické opakování, v jiné ontologii se pak tyto události modelují jako individuální body v čase.
- **Popis konceptů:** Různé ontologie mohou popisovat jinak strukturu hierarchie, či vyjadřovat shodnost, či rozdílnost konceptů jinak. Příkladem může být popis publikací. Diplomová práce může být popsána více způsoby jako:
 - diplomová práce < školní práce < vědecká publikace < publikace
 - diplomová práce < závěrečná práce < školní práce < vědecká publikace < publikace

Terminologické rozdíly:

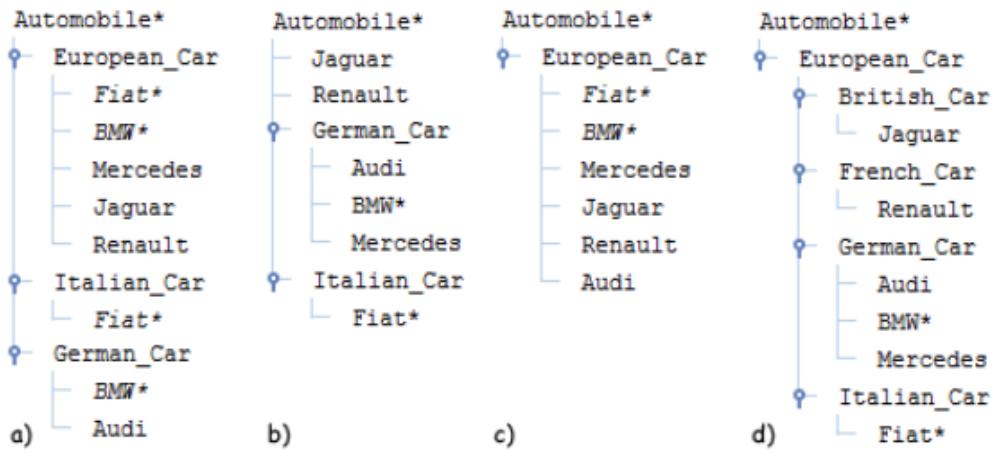
- **Synonyma:** V různých ontologiích je možné popisovat stejné entity jinými názvy. Automobil je tak možné nazvat „autem“, či „vozem“. Jedním z možných řešení je použití tezauru.
- **Homonyma:** Entity popisované stejným výrazem nemusí být automaticky tím samým konceptem. Například „kolej“ jako ubytovna pro studenty a zároveň vlaková dráha. Bez dalšího kontextu je velmi obtížné tyto entity rozeznat a často je pro rozeznání nutný vstup od uživatele[12].

2.2 Sloučení ontologií

Mapování ontologií je prvním krokem k nalezení ekvivalentních entit. V určitých situacích může postačit pro orientaci v podobnosti dvou ontologií. Dalšímo možným krokem je pak sloučení ontologií. Z metodik slučování lze pak čerpat inspiraci pro slučování hierarchií b-typů v PURO jazyce.

Na Obr. 2 (viz kapitola 2.1) je znázorněno slučování automobilů. Třída automobil je shodná, stejně tak vůz BMW a Fiat. Z hierarchie vyplývá, že pokud je Fiat v prvním příkladě Evropské auto a v druhém Italské auto, budou třídy Evropské a Italské auto na stejně úrovni. To samé platí pro Německé auto [10].

Lze uvažovat následující čtyři typy slučování (viz Obr. 3).



Obr. 3: Příklady slučování: a) plné, b) podle cíle, c) podle zdroje, d) expertní [10].

Plné sloučení zachovává všechny vztahy. Je zde však patrná redundancy. Fiat i BWM jsou podtřídou dvou kategorií. Toto není z podstaty věci špatně, ale vznikají nežádoucí jevy. Například Mercedes by bylo lepší umístit do kategorie německých aut, avšak automatizovaný systém nemá tuto skutečnost bez dalších informací podle čeho vyvozovat. Stejně tak je zjevné, že italská a německá auta by měla být podskupinou aut evropských. Kategorie „evropské auto“ by totiž mohla mít následující vztah, který je sice zjevný z názvu, ale pro ilustraci je vhodný.

<evropské auto> <vyrobeno v> <Evropa>

Při plném sloučení pak tuto informaci zdědí všechna auta v této kategorii, včetně BWM a Fiatu. Italská a Německá auta tento vztah však nemají, proto Audi nebude mít vztah „vyrobeno v Evropě“, ačkoliv je zjevné že by ho mělo mít. Zároveň však nemusí být automaticky žádoucí tento vztah přenést přes BMW na Audi díky stejné nadřídě [10].

Mějme následující příklad:

- Potraviny
 - Mléčné výrobky
 - Jogurt
 - Sýr
 - Kefír
 - Bio potraviny
 - Jogurt
 - Bio sušenky

Při plném sloučení by vznikl následující stav:

- Potraviny
 - Mléčné výrobky
 - Jogurt*
 - Sýr
 - Kefír
 - Bio potraviny
 - Jogurt*
 - Bio sušenky

Mléčné výrobky z první ontologie by mohly mít například následující vztah:

<mléčné výrobky> <obsahuje alergen> <7. mléko a výrobky z něj>

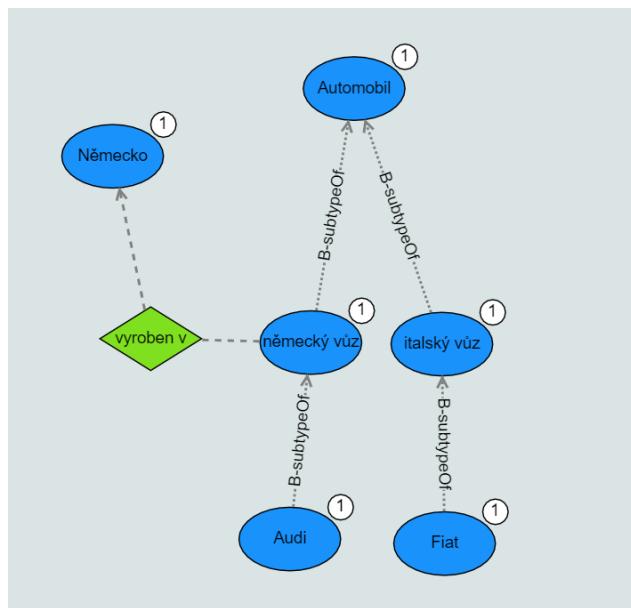
Pokud bychom pak automaticky tento vztah přes jogurt přenesli na bio sušenky, vznikla by pravděpodobně nežádoucí situace.

Zaměřme se opět na příklady s auty, konkrétně slučování podle cíle a opět předpokládejme, že evropská auta mají vztah „vyrobeno v Evropě“. V tomto příkladě se naprostě ruší třída „evropská auta“, což vede k otázce, jak pracovat s výše zmíněným vztahem. Bylo by možné ho například přenést přímo na podtřídy, tedy konkrétní značky aut. Opět by však nebylo možné tímto vztahem pokrýt i Audi.

Slučování podle zdroje vyžaduje taky tento přenos vlastností vyšší třídy na podtřídy. Například vlastnost „německé auto je vyrobené v Německu“.

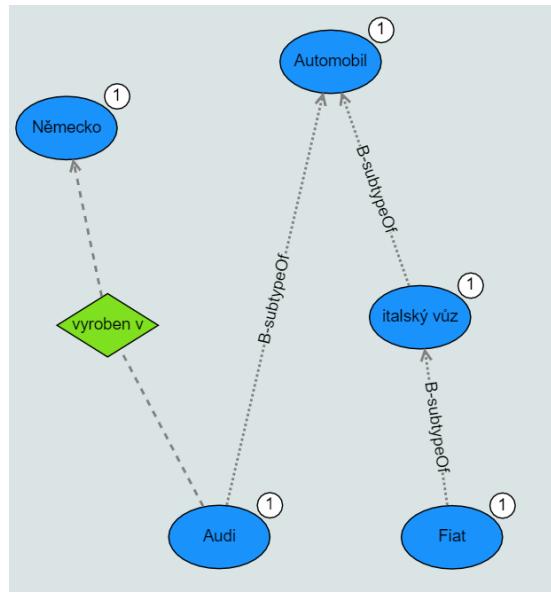
V případech slučování podle cíle a podle zdroje jsou vytvářeny přenosem vztahu z třídy na podtřídy ve své podstatě taky určité formy redundancy, jelikož původně dědičný vztah je nyní zapsán pro každou značku automobilu zvlášť.

Při slučování b-typů v PURO lze hledat inspiraci právě ve slučování ontologií. Zejména u slučování, při kterém dochází k odstranění některé ze tříd je nutné uvažovat o dvou směrech přenosu propojení s okolím na související třídy. Výše zmíněný proces se totiž týká pouze přenosu vztahů na podtřídy. Tyto vztahy totiž nelze přenést opačným směrem k nadřazené třídě. Podobné situace mohou nastat u hierarchie b-typů v modelu PURO a přenosy vztahů je tedy při slučování nutné řešit.



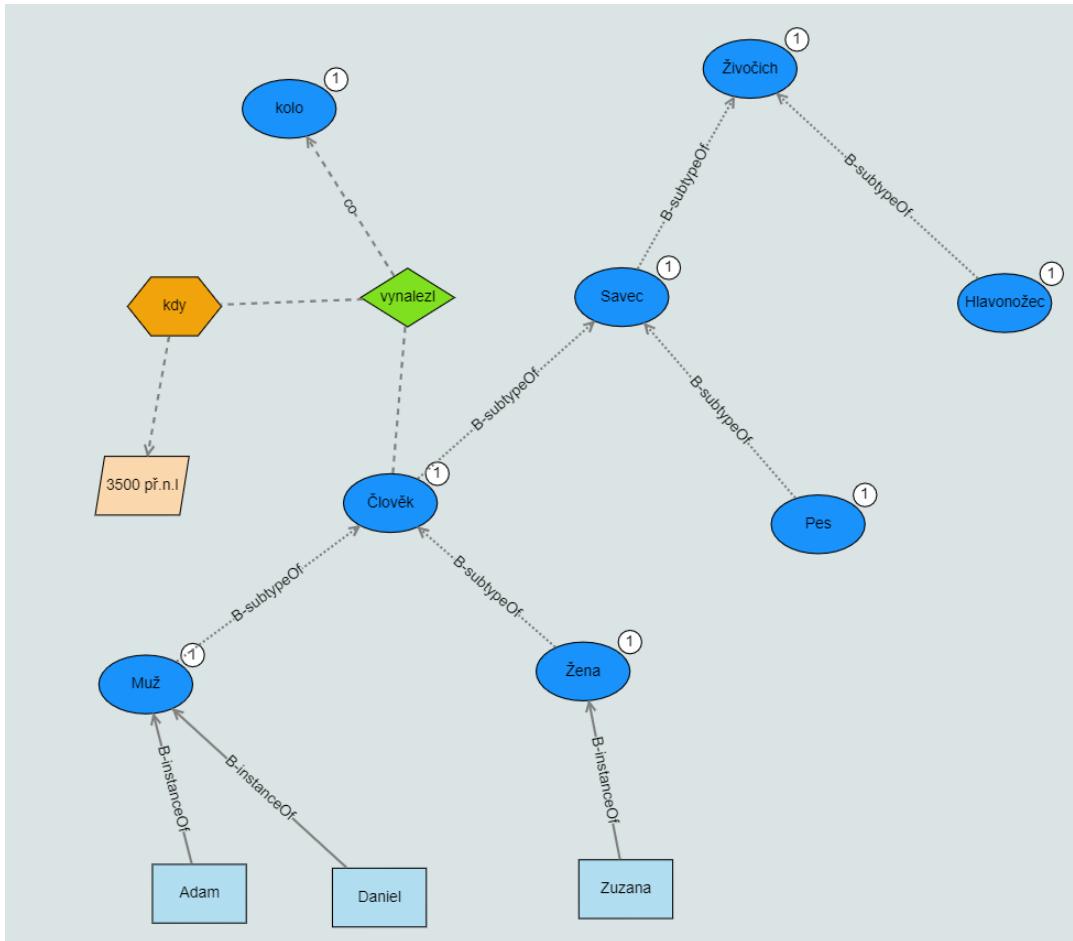
Obr. 4: Příklad relace mezi b-typy.

Na Obr. 4 je popsána situace, kdy je německý vůz vyroben v Německu. Při smazání třídy „německý vůz“ je tudíž tento vztah zjevně nutné přenést přímo na b-typ Audi, nikoli na b-typ automobil. Pokud by se tak stalo, znamenalo by, že i Fiat je vyroben v Německu. Správný výsledek je pak patrný na obrázku níže (viz Obr. 5). Byl přenesen nejen vztah „vyroben v“ ale i odkaz na nadřídu automobil.



Obr. 5: Přenos relace na podtyp.

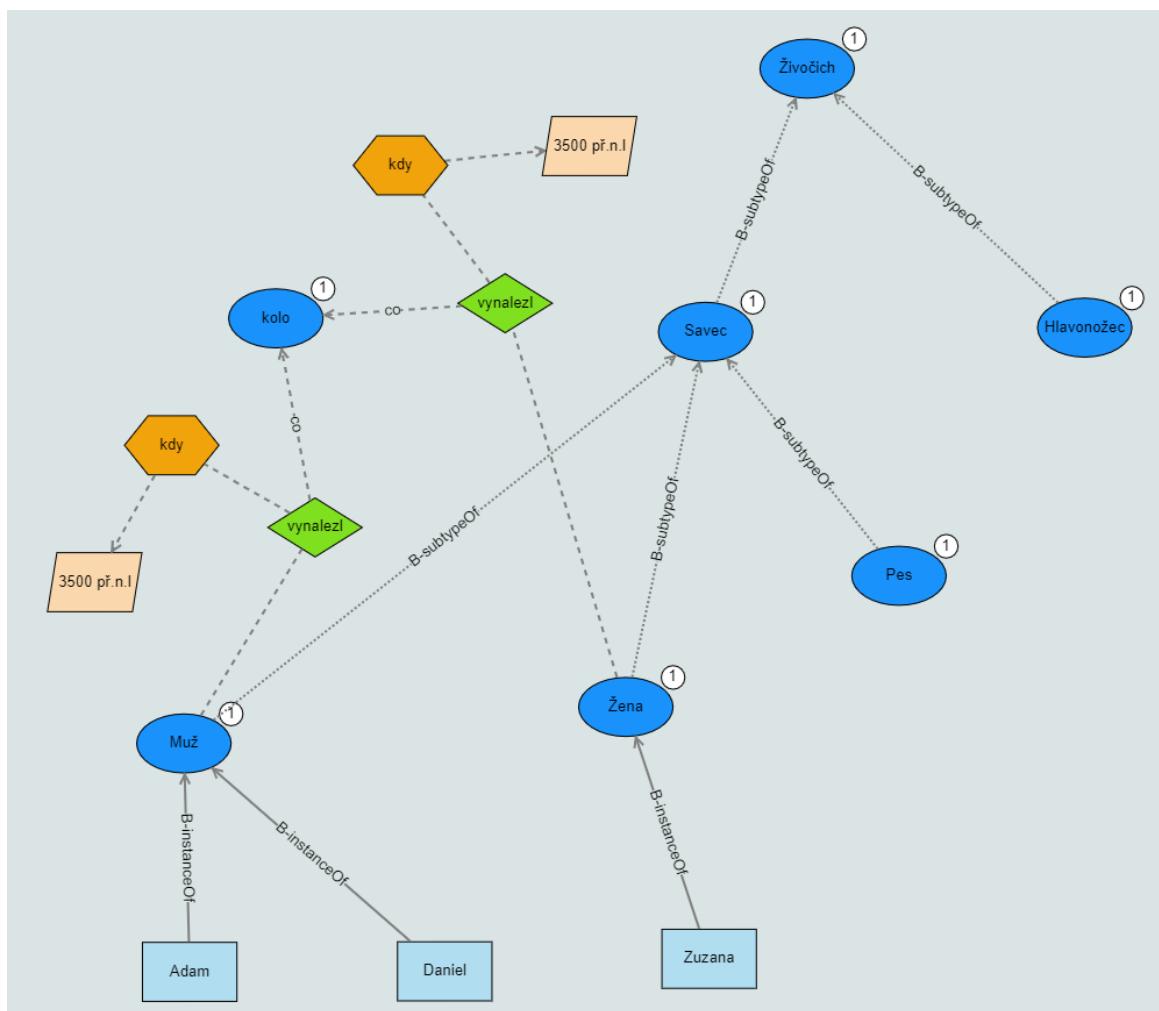
Další zmíněnou problematikou je pak přenos opačným směrem z podtřídy na nadřazenou třídu. Třídy totiž mohou mít instance. Při smazání třídy je pak nutno určit, k jaké jiné třídě má instance patřit.



Obr. 6: Příklad hierarchie v PURO.

Na Obr. 6 je popsána jednoduchá hierarchie živočichů. Instancí jsou pak muž Adam, Daniel a žena Zuzana. Dále je zde pro ilustraci vytvořen vztah popisující vynález kola. Při smazání některé ze tříd je pak nutné nejen přenést vztahy ale i vyjádření hierarchie. Pro názornost budou následující kroky postupně popisovat situaci, při níž dojde ke smazání uzlu člověk a poté uzlů muž a žena:

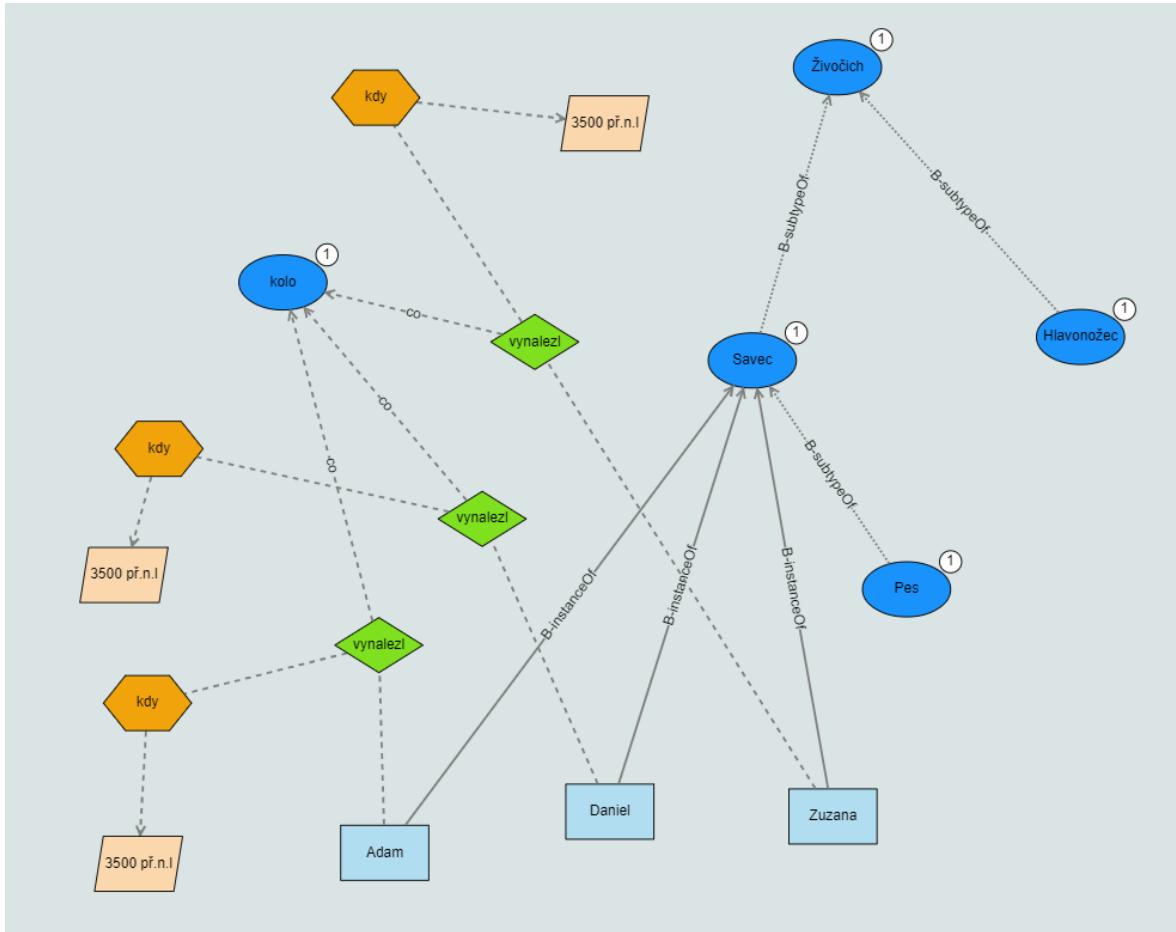
1. Smazán uzel člověk. Uzly „vynalezl“, „kdy“ a „3500 př.n.l.“ musí být duplikovány i s vnitřními hranami a s hranou směřující na uzel „kolo“. Hrana vycházející z uzlu „člověk“ musí být pro jeden duplikát směrována k uzlu „muž“ a druhý duplikát musí pro tuto hranu směrovat na uzel „žena“.
2. Hrany grafu vyjadřující hierarchii tříd se musí od „muž“ a „žena“ přesměrovat ze smazaného uzlu člověk na uzel „savec“.



Obr. 7: Příklad dědičnosti relací.

Obr. 7 popisuje stav po smazání uzlu „člověk“. Vztahy byly duplikovány a přeneseny na „muž“ a „žena“. Třída kolo nebyla duplikována, protože vztah se v obou případech týká obecné třídy „kolo“. Třídy „muž“ a „žena“ jsou nyní přímo podtřídou „savec“, což je logicky v pořádku. Pokud by například třída savec měla vlastnost „je teplokrevný“, dědily by ho třídy muž a žena i před smazáním třídy „člověk“. Na tomto příkladě je také zjevné, proč se vztahy musí dělit směrem dolů. Pokud by tomu bylo naopak, bylo by pak validní tvrzení, že „savec vynalezl kolo“. Přenesením by pak i pes vynalezl kolo.

Pokud dojde k dalšímu smazání tříd, konkrétně „muž“ a „žena“, je nutné vztahy přenést přímo na instance. Ty se pak stanou přímými instancemi třídy „savec“. Vzhledem k tomu, že muž má dvě instance, je nutné opět duplikovat vztah vynalezení kola a přenést ho na instance (viz Obr. 8).



Obr. 8: Přenos relací na instance.

Je zřejmé, že na jedné straně dochází ke zjednodušování modelu z pohledu počtu uzelů tříd, avšak narůstá počet vztahů a model se při větším počtu uzelů může velmi rychle stát nepřehledným. Specifickou otázkou pro modelování v PURO pak je, zdali je pravdou, že Adam, Daniel a Zuzana vynalezli kolo 3500 let př.n.l. V nástroji PURO se totiž modelují situace popisující jednoduché příklady. Při převodu do OWL se pak samotné instance nepřenášejí. Proto by byl výsledným vztahem „savec vynalezl oheň“. V takovém případě se pak nabízí otázka, zdali je vůbec možné přenášet vztahy při mazání tříd přímo na instance. Ve výsledném modelu bez instancí by totiž bylo možné tvrdit, že pes vynalezl oheň, což by nebylo žádoucí.

2.3 Párování instancí

Obecně označované jako Instance matching. Jedná se o jeden z procesů hledání podobnosti mezi dvěma sadami dat obsahujících entity v podobě instancí ve formátu RDF. Cílem je nalézt v těchto odlišných datech shodné instance, které by ve výsledném sloučeném modelu měly být zastoupeny pouze jednou. Je tedy nutné na základě vhodně zvolených technik posoudit míru podobnosti dvou entit.

Informace vhodné k párování instancí je možné rozdělit do následujících kategorií [15]:

- **Identifikátor:** Pokud je identifikátor v podobě IRI stejný, lze považovat instance za shodné.
- **Meta informace:** Porovnává informace o uzlu, tedy vlastnosti uzlu, třídy a další popisy.
- **Název:** Název uzlu vycházející například z RDFS: label, nebo vlastnosti z jiných ontologií (FOAF: name), či část řetězce IRI, kterou lze považovat za název. Název je pro unikátní označování objektů velmi blízké lidskému uvažování a často je tak možné posuzovat shodnost dvou entit.
- **Popisné vlastnosti:** Popis entity v přirozeném jazyce. Může se jednat o komentář, či delší text, který je možné s jiným popisem porovnávat podobností dokumentu. Může se jednat například o RDFS: comment.
- **Charakteristiky:** Jedná se o vlastností přímo nepopisující entitu názvem, či textovým dokumentem. Například vlastnost „je muž“ u instance člověka lze chápat jako omezení při párování s jinou instancí, která je „je žena“. Vlastnost „e-mail“ může být postačující pro vyvození shodnosti dvou instancí.
- **Sousedé:** Objektové vlastnosti odkazující na vztahy s ostatními instancemi lze použít pro párování instancí. Lze předpokládat že potenciálně shodné entity budou mít podobné sousední entity. Tento přístup je vhodný v situacích, kdy není pro entitu dostatek popisných vlastností a charakteristik [15].

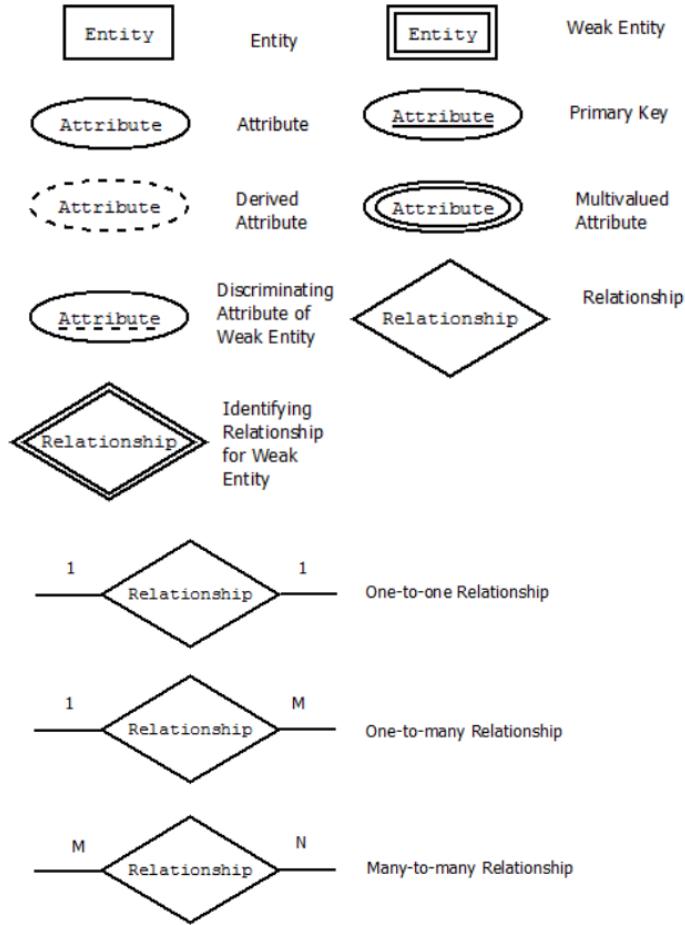
2.4 Slučování Entity Relationship modelů

ER modely neboli entitově vztahové modely, slouží k abstraktnímu a konceptuálnímu znázornění databáze [16]. Jedná se o vizuální reprezentaci určité oblasti pomocí popisu objektů a vztahů mezi nimi.

V ER modelu existují tři základní elementy [17]:

- Entita
- Atribut
- Vztah

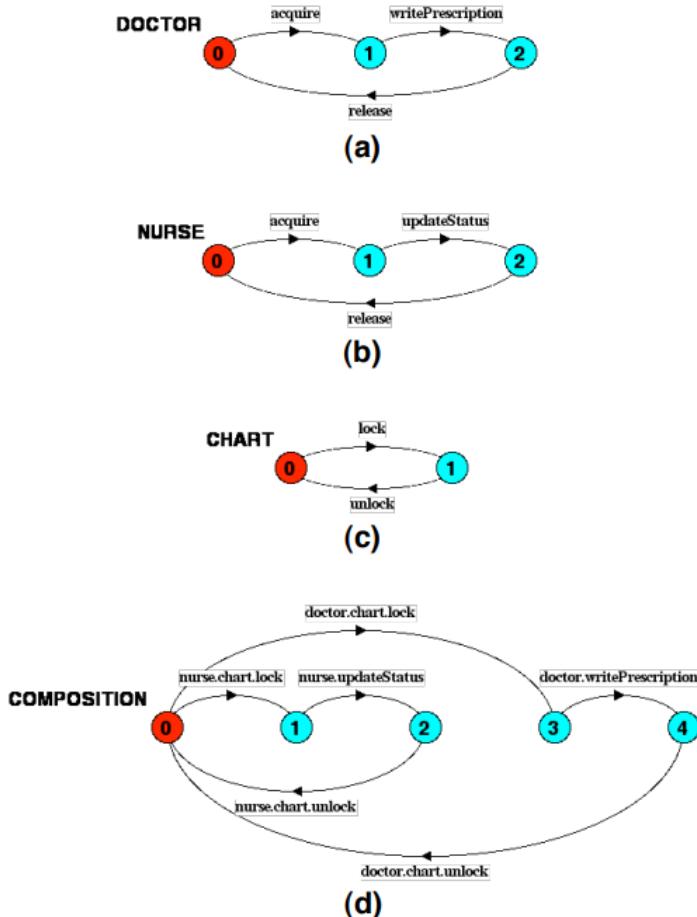
Tyto elementy jsou rozšiřovány různými podtypy (viz Obr. 9).



Obr. 9: Elementy ER diagramů [17].

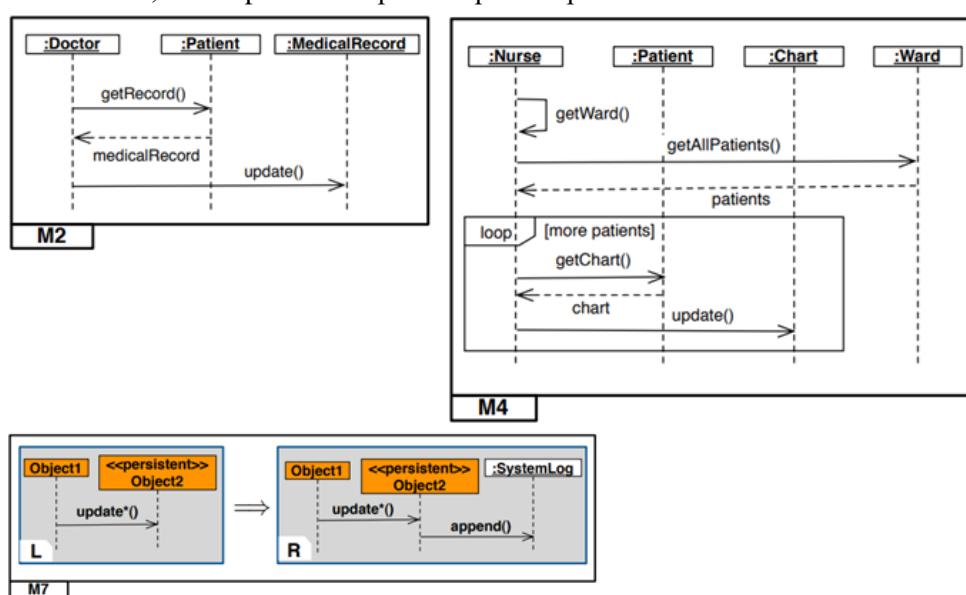
Při slučování ER modelů lze rozlišit tři klíčové operace: slučování, kompozice a protkávání [18].

- Cílem slučování modelů je spojit vstupní modely sjednocením jejich přesahů. V některých případech mohou být překrývající se charakteristiky vstupních modelů v rozporu. Stávající přístupy slučování se v řešení takových případů liší: některé přístupy vyžadují, aby byly slučovány pouze konzistentní modely, což znamená, že nekonzistentní modely musí být opraveny před nebo během sloučení. V jiných přístupech se tolerují nekonzistence a jsou explicitně reprezentovány ve výsledném sloučeném modelu.
- Kompozice se týká procesu sestavování sady autonomních, ale vzájemně se ovlivňujících modelů, které zachycují různé součásti systému. Příklad kompozice je na Obr. 10.



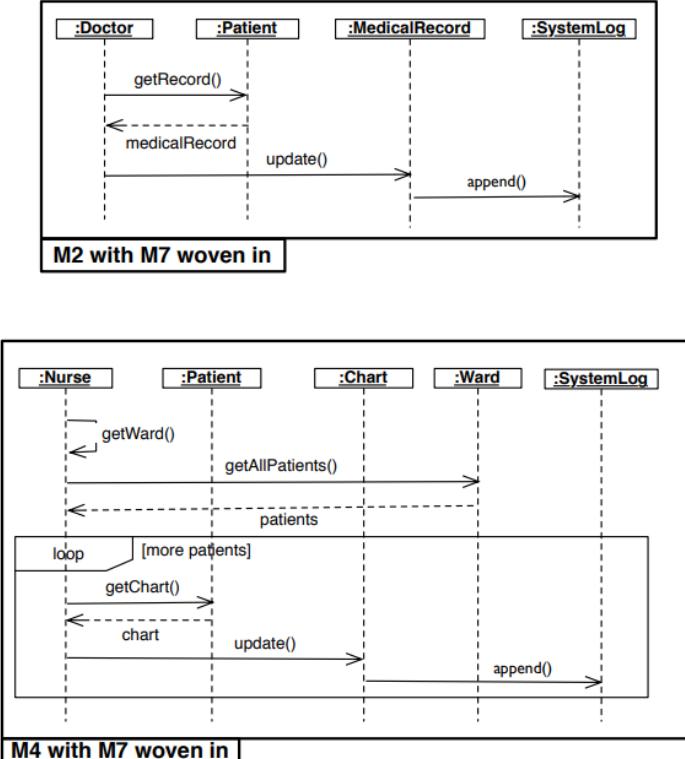
Obr. 10: Příklad kompozice.

- Protkávání se používá k začlenění průrezových problémů do základního systému. Operátory protkávání mohou být implementovány různými způsoby v závislosti na povaze základního systému a na tom, zda se protkávání provádí při komplikaci nebo za běhu.



Obr. 11: Příklad modelů (M2, M4 a M7) z různých zdrojů [18].

Obr. 11 popisuje modely sekvenčních diagramů. Model M2 zobrazuje transakce mezi doktorem, pacientem a lékařským záznamem. Model M4 popisuje každodenní prohlídku pacienta. Sestra vyhodnocuje zdravotní stav a ten zaznamenává do karty pacienta. Model M7 popisuje zaznamenávání každé změny v datech. Obr. 12 pak popisuje výsledek protkávání modelů.



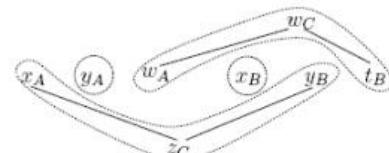
Obr. 12: Protkávání modelu M7 do modelů M2 a M4 [18].

Pro slučování je možné navrhnout algoritmus využitelný pro jakýkoli jazyk založený na grafové reprezentaci. Algoritmus se dělí na dvě hlavní části: slučování sad a slučování grafů [19].

$$A = \{x_A, y_A, w_A\} \quad B = \{x_B, y_B, t_B\}$$

$$C = \{z_C, w_C\}$$

(a)



(b)

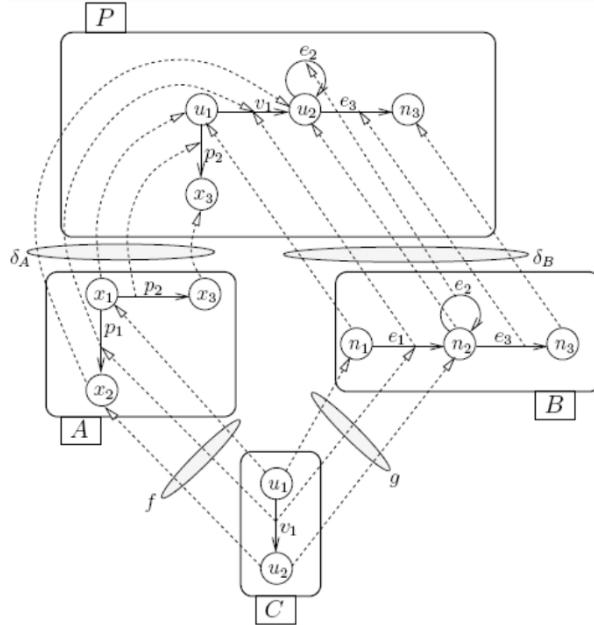
$$P = \{\{x_A, y_B, z_C\}, \{y_A\}, \{w_A, t_B, w_C\}, \{x_B\}\}$$

(c)

Obr. 13: Slučování sad [19].

Obr. 13 popisuje fáze slučování sad. (a) popisuje diagram propojení, (b) graf sjednocení s propojenými komponentami a (c) je sloučená sada [19].

Mehrdad využívá homomorfismus k označování mapování dvou grafů. Homomorfismus lze v tomto případě chápát jako mapování jednoho grafu do druhého při zachování struktury. Jako mezikrok je použit diagram propojení, jehož objekty jsou slučované grafy a jejich propojení jsou homomorfismy [19, 20].

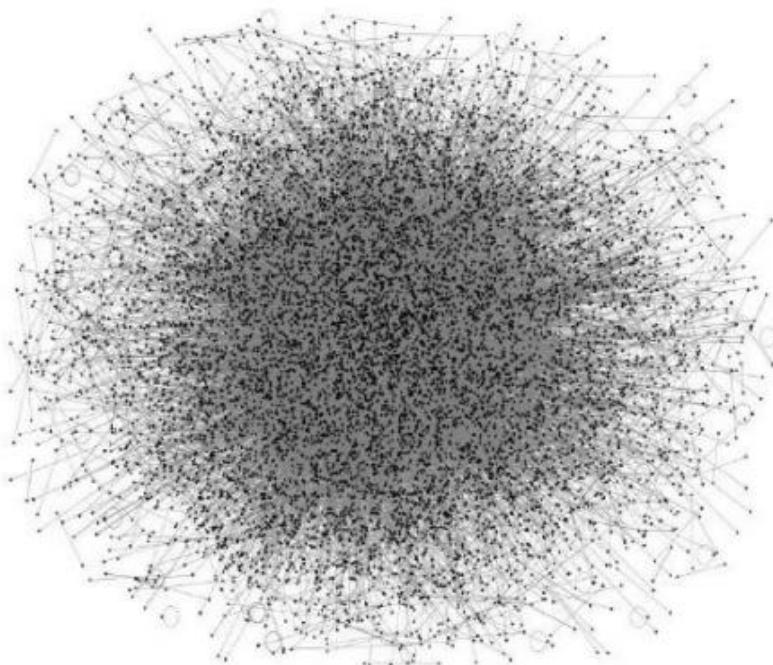


Obr. 14: Homomorfismy při slučování dvou grafů [19].

Na Obr. 14 jsou homomorfismy reprezentovány čárkovanými linkami a grafy jsou zde jako samostatné objekty. $\delta\mathbf{A}$ a $\delta\mathbf{B}$ vyjadřují, jak jsou grafy A a B reprezentovány v grafu P [19].

3 Vizualizace velkých grafů

Obecně je vhodné vyvýjený model vizualizovat pomocí grafu. Entity jsou pak zobrazovány jako uzly grafu a vztahy mezi nimi jsou zobrazeny uzly, či hranami. Při rostoucím počtu entit v modelu může nastat situace, kdy se stává model nepřehledným. Techniky pro vizualizaci velkých grafů mají za cíl organizování uzel a hran do přehledné podoby a vhodné zobrazování relevantních částí modelu [21].



Obr. 15: Neuspořádané zobrazení velkého grafu [21].

Na Obr. 15 je příklad velkého množství entit propojených uzly. Je zřejmé, že orientace v tomto grafu není jednoduchá. Při vývoji PURO modelu může nastat podobně chaotická situace, jelikož nelze automaticky předpokládat, že budou entity nutně propojeny pouze v rámci nějaké omezené oblasti. Je nutné uvažovat, že mnoho entit může být navzájem propojeno potenciálně s každou další entitou. Je nasnadě, že i mnohem menší množství uzel se při vývoji v PURO stane nepřehledné, protože používané zobrazovací zařízení má omezenou velikost a uživatel pravděpodobně potřebuje zobrazovat uzly s textovými popisky [21].

3.1 Použitelná rozložení

Vhodné rozložení uzel v grafu je možné aplikovat na model PURO, který je ve své vizuální reprezentaci také grafem. Správně zvolená technika může tvůrci modelu usnadnit návrh rozsáhlějších grafů.

Lze definovat šest základních typů uspořádání použitelných v různých situacích na základě charakteru dat [21].

3.1.1 Rozložení Force-directed

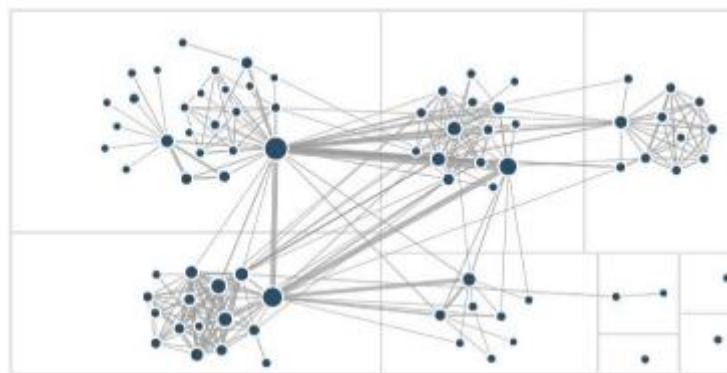
Toto řešení je obtížně implementovatelné, ale nabízí poměrně intuitivní a přehledné zobrazení. Cílem je od sebe odpuzovat jednotlivé uzly na základě definovaného „náboje“. Chování lze přirovnat k odpuzování objektů vlivem statické elektřiny. Parametrem pro odpuzování je pak výše zmíněný náboj, maximální, či minimální délka hrany a její elasticita, tedy snaha vrátit se do původního stavu. Obecně je nutné pro toto rozložení iterativně provádět kroky, ve kterých se prochází každý uzel grafu a vyhodnocuje se jeho poloha před další iterací výpočtu. Je vhodné určit maximální počet iterací, či minimální změnu, při které již bude rozložení považováno za stabilní. Implementace je tedy poměrně obtížná a potenciálně výpočetně náročná. Toto rozložení je vyobrazeno na Obr. 16 [21].



Obr. 16: Force-directed rozložení [21].

3.1.2 Shlukové rozložení

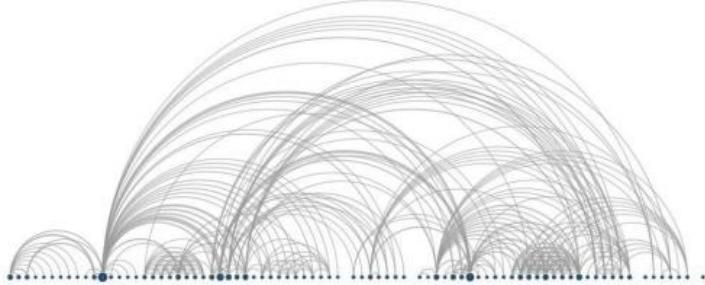
Rozložení slouží pro vizualizaci grafů, jejichž uzly mají mezi sebou nějaké společné vlastnosti. Kombinací výše uvedeného force-directed rozložení a principů shlukování lze seskupit podobné uzly do semknutých skupin (viz Obr. 17) [21]. V kontextu PURO modelů by bylo možné seskupovat entity na základě podobných atributů a relací. Shlukování na základě názvu b-typů a b-objektů, či na základě shodnosti termů by pravděpodobně nebylo žádoucí.



Obr. 17: Shlukové rozložení [21].

3.1.3 Lineární rozložení

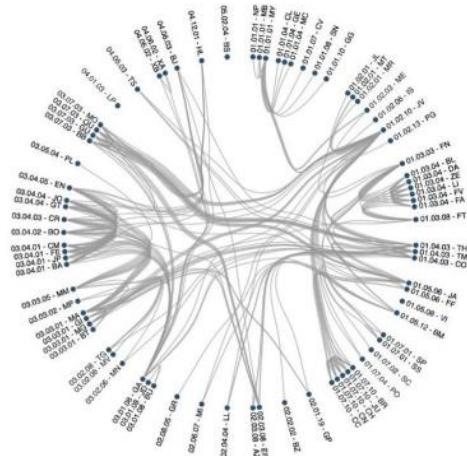
Rozložení pokládá všechny uzly grafu vedle sebe na osu (viz Obr. 18). Hrany grafu jsou pak reprezentovány obloukem mezi dvěma uzly. Pro zlepšení přehlednosti je možné využívat shlukování. Shluky jsou pak mezi sebou vizuálně odděleny například větší mezerou [21].



Obr. 18: Lineární rozložení [21].

3.1.4 Kruhové rozložení

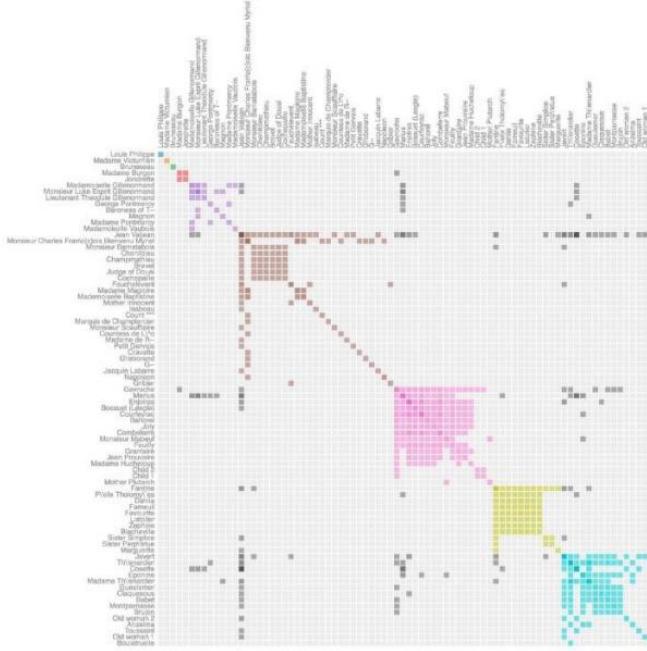
Toto rozložení je založeno na podobném principu jako lineární rozložení. Entity jsou položeny na kružnici a hrany procházejí mezi uzly vnitřkem kružnice (viz Obr. 19). Opět je možné používat shlukování pro větší přehlednost. Křížení hran podobně jako v lineárním rozložení vytváří poměrně nepřehledné oblasti [21].



Obr. 19: Kruhové rozložení [21].

3.1.5 Maticové rozložení

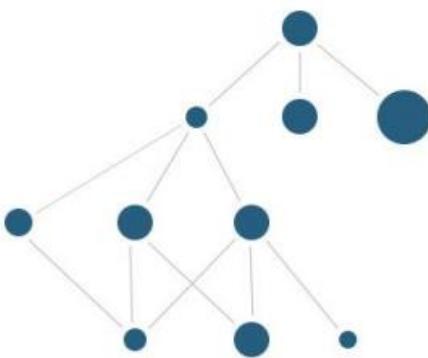
Toto rozložení je pro určité datové struktury poměrně vhodné, protože respektuje strukturu řádků a sloupců a nabízí tedy pro člověka přirozeně čitelnou tabulku (viz Obr. 20). Každá entita je vyobrazena v řádku i sloupci a v průsečíku dvou entit je buňka jako reprezentace hrany grafu. Nevýhodou je velikost při aplikaci na velké množství dat a potenciálně řídké zastoupení buněk, které reprezentují existující vztah. Maticové rozložení tak může mít rozsáhlé nevyužité plochy. Zobrazení je však velmi efektivní pro vizualizace, kdy mají všechny uzly vzájemně mezi sebou hrany, což by v ostatních zobrazeních vedlo k nepřehlednému křížení [21].



Obr. 20: Maticové rozložení [21].

3.1.6 Stromové rozložení

Toto rozložení umisťuje uzly do úrovní na základě určité hierarchie (viz Obr. 21). Je vhodné pro zobrazování grafů, které mají mezi uzly orientované hrany. Jako příklad může sloužit rodokmen, či vyjádření hierarchie podtříd [21]. V PURO modelu může být takové rozložení potenciálně vhodné pro zobrazování struktury b-typů na základě vztahu b-subtypeOf. Pokud jsou mezi entitami i jiné vztahy, které nevyjadřují hierarchii, může docházet k nepřehlednému hřízení hran. Strom může též růst do šírky, či výšky v závislosti na množství uzel stejně úrovni a vytvářet tak velmi úzkou strukturu, která je obtížně zobrazitelná.



Obr. 21: Stromové rozložení [21].

3.2 Doporučené techniky

Doporučené techniky úpravy grafů jsou:

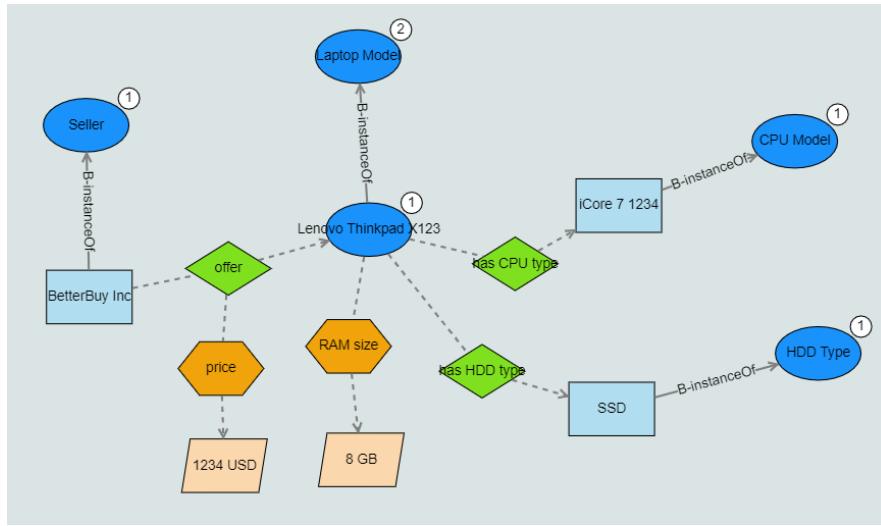
- **Minimalizace křížení:** Hrany by se, pokud možno, neměly vzájemně křížit. V případě, že křížení nelze zabránit, by měl být graf zobrazen tak, aby se hrany křížily co nejméně.
- **Minimalizace křivek:** Hrany by měly být v největším počtu případů přímé. Oblouky a smyčky by se měly používat pouze v nutných případech.
- **Minimalizace plochy:** Zobrazovaný graf by měl pokrývat co nejmenší plochu.
- **Minimalizace délky:** Pro lepší přehlednost by měla být každá hrana grafu co nejkratší.
- **Maximalizace úhlu:** Úhel mezi více hranami by měl být co největší.
- **Symetrie:** Graf by měl být, pokud možno, zobrazen co nejvíce symetricky.
- **Shlukování:** V rozsáhlém grafu je vhodné seskupovat uzly do skupin. Seskupování je možné provádět například na základě společných vlastností uzlů [21].

Následující čtyři metody umožňují zobrazovat různé míry detailu v grafu [21]:

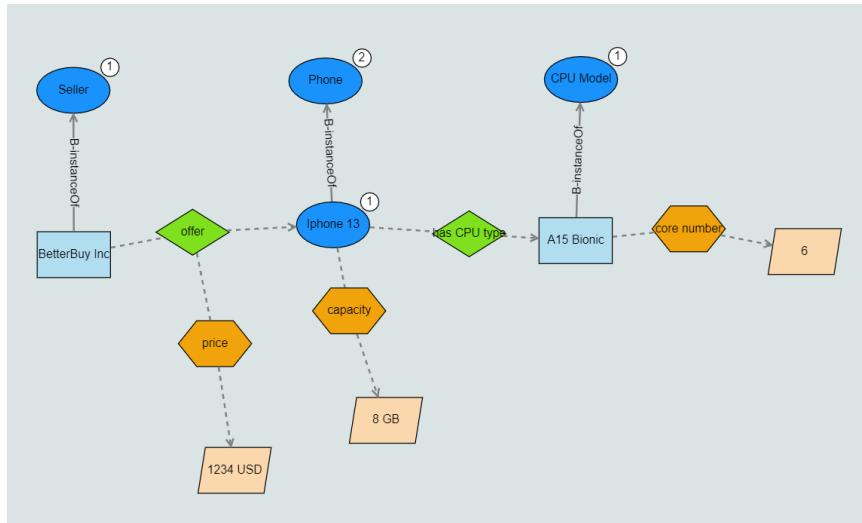
- **Přiblížování a oddalování:** Na základě přiblížení grafu jsou zobrazovány různě granulární informace. Pokud je graf oddálen, nemusí být například zobrazeny zbytečně detailní popisy uzlů a hran, nebo mohou být přímo vynechány. Tuto techniku by bylo možné v PURO aplikovat například na atributy, valuace a relace. Při určitém oddálení by se tyto uzly mohly změnit, či zmizet. A nabídnout uživateli pouze přehled o tom, že typy a objekty mezi sebou mají nějaký vztah.
- **Rozbalování a sbalování po kliknutí:** Technika využívá zobrazování a minimalizace určitých rozšiřujících informací o uzlu. Tyto informace nemusí být zobrazeny neustále, aby zbytečně nezahľtily graf. Uživatel by pak tyto data získal kliknutím na určitý uzel grafu. Autor této práce rozvíjí myšlenku o používání kontextového menu, které uživateli nabídne při slučování dvou PURO modelů možnosti dostupné pro konkrétní uzel. Toto menu se pak zobrazuje vždy na základě vybraného uzlu. Další možností je skrytí nepotřebných hran grafu a jejich zobrazení pouze při vybrání relevantního uzlu. V PURO modelu by například nebylo nutné po celou dobu zobrazovat hranu vyjadřující B-instanceOf. Uživatel by měl možnost zvolit si skrytí těchto hran a až při vybrání instance, či b-typu by se relevantní hrany dočasně zobrazily. Autor vychází z předpokladu, že pokud se uživatel právě nevěnuje oblasti modelu popisující tyto instance, nejsou hrany této části grafu relevantní.
- **Barevné odlišení po kliknutí:** Při práci s určitým uzlem by se barevně zvýraznila oblast relevantní pro tento uzel. Uživatel by pak mohl bez dalších úprav zobrazení grafu přehledně pracovat pouze s barevně odlišenou relevantní oblastí. Tato technika by byla v PURO vhodná například pro zobrazení instancí b-typu, či určitého okolí uzlu. Případně pro zvýraznění částí grafu, které spolu nejsou nijak propojeny. Uživatel by pak byl obeznámen s tím, že vytvořený model není kompletně propojen a mohl by se tedy soustředit na propojení dvou barevně odlišených oblastí.
- **Zobrazení podmnožiny:** Tato technika pracuje se zobrazením určitého uzlu na základě jeho výběru například ze seznamu elementů. Implementovat lze pomocí skrytí nevybraných uzlů, zvýraznění vybraného uzlu, či zobrazení v novém okně. V PURO modeleru by tato technika mohla sloužit k vyhledávání uzlu na základě názvu a termu. Případně při slučování pro označení podobných uzlů napříč více grafy.

4 Problematika slučování v PURO

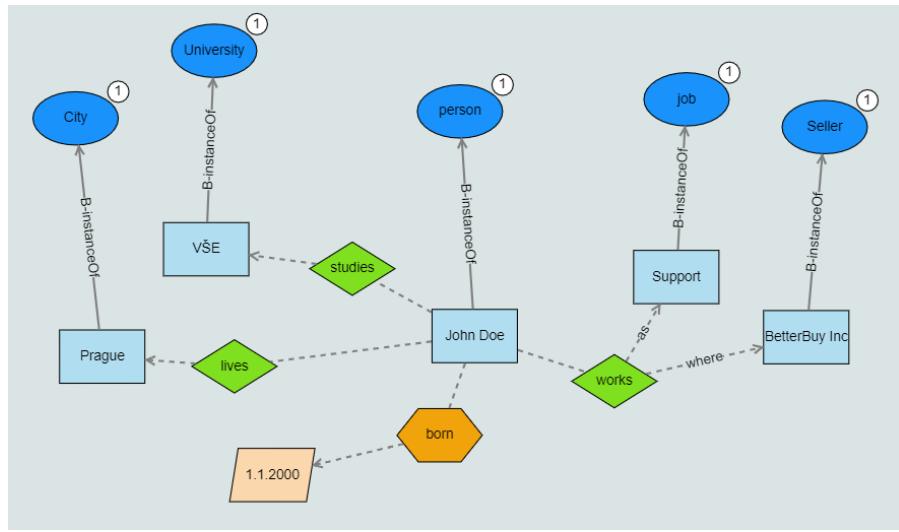
Z výše uvedených algoritmů a metodik lze pro slučování v PURO vyvzovat určité paralely. Obecně však nelze přímo převzít postupy a aplikovat je na tento jazyk. Je třeba rozlišovat mezi ontologiemi a PURO. PURO je model zachycující určitou oblast reality na konkrétních příkladech pomocí vztahů mezi entitami. Výsledkem je tedy model, který je potenciálně možné převést do OWL, a tedy ve výsledku vytvořit ontologii. Ve skutečnosti však ontologií, ani jakoukoli databází, není.



Obr. 22: Model nabídky notebooku.



Obr. 23: Model nabídky telefonu.

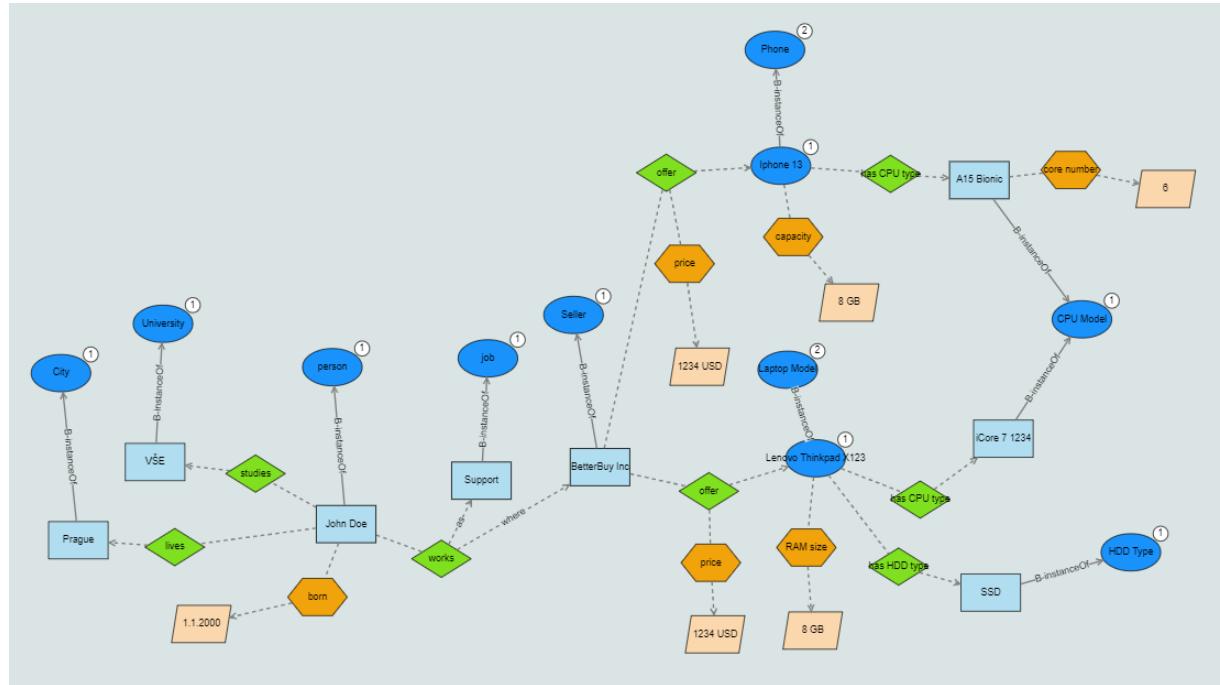


Obr. 24: Model popisující člověka.

Výše jsou popsány tři modely, které popisují prodejce elektroniky a jím nabízené produkty v podobě notebooku (Obr. 22) a telefonu (Obr. 23). V posledním modelu (Obr. 24) je popsána osoba, která u prodejce elektroniky pracuje.

Tyto modely jsou ve svém rozsahu malé, ale i zde je možné pozorovat odlišné oblasti, které se tvůrce modelu snaží popsat. Je možné, že by například model popisující osobu byl mnohem rozsáhlejší a zahrnoval by spoustu údajů o rodinných příslušnících, či informací o městě, ve kterém osoba žije.

Při slučování těchto modelů je nutné identifikovat podobné entity, zejména pak entitu prodejce. Dále je nutné sloučit b-type „CPU Model“, který sdílí model popisující telefon a notebook.



Obr. 25: Sloučení tří modelů.

Na Obr. 25 je možná podoba sloučených modelů. Za povšimnutí stojí například relace „offer“ která je vytvořena ve dvou exemplářích pro oba produkty. V souvislosti s tím je nutné vytvořit pro obě relace nabídky atribut „cena“ a jeho hodnotu.

4.1 Mapování termů

Problematická je aplikace obecně využívaných metod na porovnávání přímých vlastností uzelů. PURO pro popis entit používá pouze název. Využití názvu jako řetězce znaků pro srovnávání je poměrně omezené a nemusí být vždy spolehlivé. Synonymie, homonymie a další jevy v přirozeném jazyce včetně překlepů může vést ke špatné identifikaci párů. Použití IRI může v ontologích vést ke spolehlivému nalezení shodné entity. V PURO jazyce se proto musí vývojář soustředit na konzistentní používání stejných výrazů ve všech modelech. Pokud se však pamatuje na tento fakt při vyvíjení většího množství modelů, pak je jejich následné slučování spolehlivější.

Pro nalezení párů entit je nutné popsat předpoklady, ze kterých lze při vyhledávání vycházet. Následující skutečnosti jsou silným předpokladem pro vytvoření páru:

- Entity mají shodný, nebo velice podobný název.
- Entity jsou stejného typu.
- Vztahy entit s okolím jsou podobné.

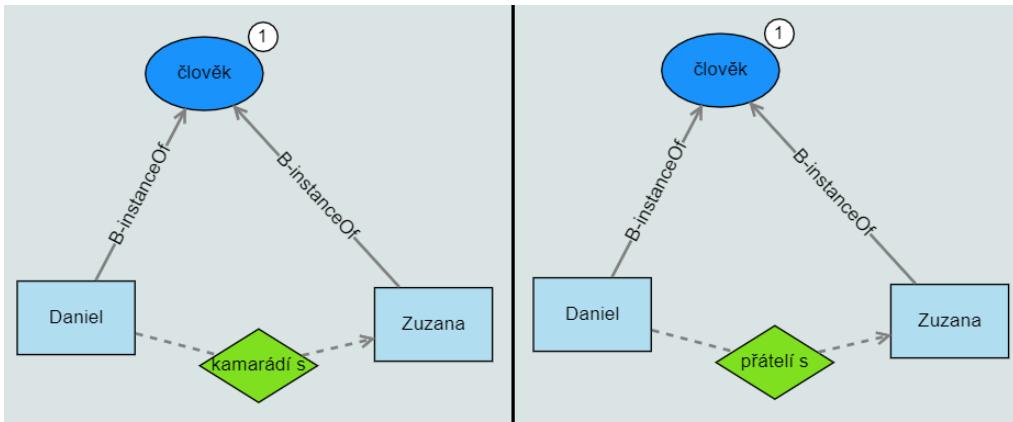
Je zřejmé, že tyto předpoklady jsou jednotlivě nespolehlivé. Jistě může existovat například firma nesoucí jméno svého zakladatele, přičemž obě entity jsou typu b-object. Tato nejednoznačnost může nastat i napříč druhy entit. Vlastnost b-attribute označující váhu člověka může být popsána slovem Váha. Stejné slovo však může označovat b-type, například nástroj k vážení.

Vztahy s okolím také neznamenají nutně shodnost, či podobnost. Model popisující nabídku elektroniky v obchodě může telefonu i počítači přiřazovat spoustu shodných vztahů. Oba mohou mít cenu, váhu, typ procesoru, velikost obrazovky a podobně. Lišit se mohou třeba i v jediné entitě, například b-typu, kterého jsou instancí.

Proto jsou výše uvedené skutečnosti samy o sobě nedostačující a pro vyhodnocení páru je třeba brát v potaz všechny najednou. Pro vyhodnocení lze navrhnout skórovací systém, jehož výstupem je míra podobnosti dvou entit.

4.1.1 Mapování relací

Mapování termů je závislé přímo na jejich příslušnosti k b-objektu, či b-typu. Pokud však mají například dvě relace jiný název, avšak stejný význam, není jednoduché tyto uzly identifikovat jako páry.



Obr. 26: Příklad shodných relací.

Výše uvedené modely na Obr. 26 popisují totožné situace, avšak při mapování není pro porovnání relací v podstatě z čeho vycházet. Bylo by chybné vyvozovat shodnost pouze z faktu, že relace se v obou případech týká dvou lidí. Pokud by se název relace přepsal na „je rodič“ je zjevné, že by pak sloučení relace „přátelí s“ a „je rodič“ vedlo ke ztrátě určité skutečnosti, kterou model popisuje. Zároveň je možné, že vývojář má zájem o zachování obou relací, jelikož je možné pozorovat určitou jemnou rozdílnost v expresivitě popisu vztahu mezi lidmi, kteří jsou přátelé, či kamarádi.

4.1.2 Mapování atributů

Stejná problematika se také týká mapování b-atributů v PURO. Jednoduší situace je však v b-valuacích, které slouží pouze jako pomůcka pro orientaci v modelu. Pokud tedy bude ve dvou modelech rozdílná hodnota nějakého atributu, není třeba řešit jejich podobnost. Důvodem je fakt, že se konkrétní hodnoty nepřenáší do výsledného OWL. Takže například konkrétní hodnota „datum narození“ může být ve dvou modelech rozdílná, ale klíčový je pouze fakt, že osoba má datum narození a konkrétní den je irrelevantní.

Z opačného pohledu by se mohlo zdát, že b-valuace jakožto konkrétní hodnota b-atributu může být užitečná pro vyvozování shodnosti dvou různě pojmenovaných atributů. Pokud bude mít konkrétní instance osoby ve dvou modelech různě pojmenované atributy, mohla by se jejich shodnost vyvozovat z konkrétní hodnoty. Například konkrétní shodné datum u atributu „den narození“ a „datum narození“ může znamenat, že se v obou případech jedná o tentýž atribut. Může být ale pravděpodobné, že tvůrce modelu použije hodnotu datumu jako pouhou reprezentaci a datum se tedy bude lišit. Poté může nastat situace, kdy jeden atribut bude třeba popisovat datum narození a druhý atribut například datum nástupu do práce. Pouhý fakt, že oba atributy mohou mít jako hodnotu datum, není dostačující.

4.1.3 Mapování instancí

PURO jazyk je specifický tím, že instance použité v modelu jsou pouze určitou omezenou reprezentací reálného světa. Slouží pouze jako pomůcka pro vývojáře k ujasnění vztahů a struktur v potenciální ontologii. Vzhledem k vynechání instancí při převodu do OWL je mnoho problémů možné do určité míry abstrahovat.

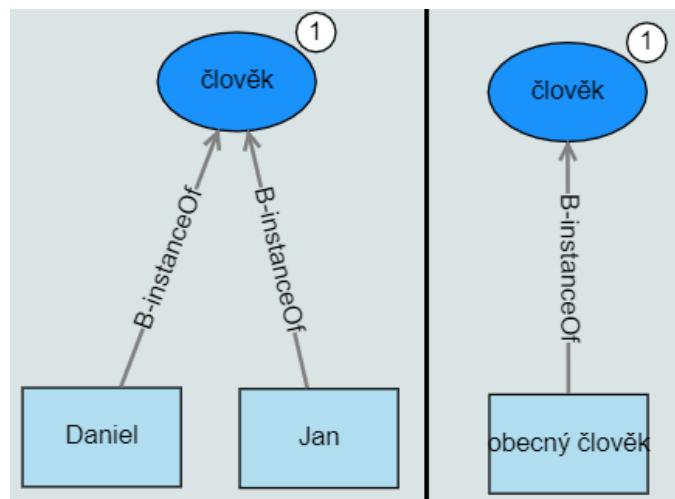
Možné přístupy porovnávání instancí v PURO

Autor identifikuje následující situace a možné přístupy při porovnávání instancí v rámci modelů PURO:

- **Pesimistický:** Pokud jsou dvě instance v jakémkoli aspektu (popis, atribut, relace, či b-typ) odlišné, není možné je sloučit. Tento přístup vede přirozeně k větší redundancii a znepřehlednění výsledného sloučeného modelu. Je zde však mnohem menší nutnost lidského zásahu při slučování, protože není třeba, aby uživatel při slučování jakkoli posuzoval, jaká míra podobnosti dvou entit je dostatečná pro sloučení.
- **Optimistický:** Pokud dvě instance sdílí alespoň nějakou sadu vlastností, je nutné zvažovat jejich shodnost. Pokud je provedena identifikace instancí správně, vede tento postup k menší redundancii a výsledkem je přehlednější model o méně instancích.
- **Triviální:** Tento postup vychází z předpokladu, že jsou instance ve své podstatě pouze pomůckou pro vývojáře a po převodu do OWL jsou irrelevantní. Pokud jsou dvě entity instancí naprosto shodných b-typů, lze ignorovat jejich individuální vyjadřovací schopnost a je možné je sloučit v jednu obecnou entitu.

4.2 Slučování instancí

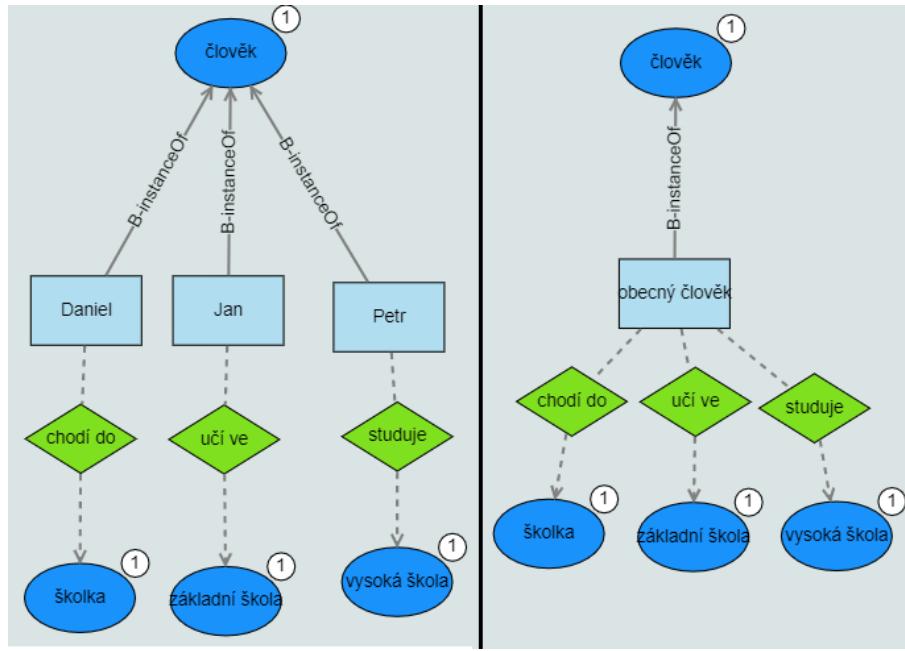
Při návrhu modelu může nastat situace, kdy bude velká část instancí náležet stejnemu b-typu. Autor modelu může pro každou instanci uvažovat nějaké specifické atributy a relace a tím popisovat určitou část pokrývané domény. Pro účely pochopení situací může být vhodné mít instance individuálně popsané. Tvůrce modelu může chtít například popsát vztahy v rodině. Lze očekávat, že takové vztahy se budou týkat mnoha osob, které mezi sebou mají specifické relace, jako například „je otec“, „je bratr“. Při větším množství instancí je sice každá relace lépe čitelná, ale pokud se poté tvůrce modelu rozhodne rozvinout popisovanou situaci o další domény, mohlo by být vhodnější mít jednu univerzální osobu, jakožto jedinou instanci člověka.



Obr. 27: Příklad sloučení instancí do obecné entity

Sloučením instancí pod jednu obecnou entitu (viz Obr. 27) je zjednodušen výsledný model. Jakékoli atributy či relace příslušící jednotlivým instancím by pak zdědila výsledná obecná instance a přeneseně pak i b-type jehož je instancí. Tento přístup sice vede k vizuálně přehlednějšímu modelu, ale v určitých situacích může vést k vyšším nárokům na pochopení modelu.

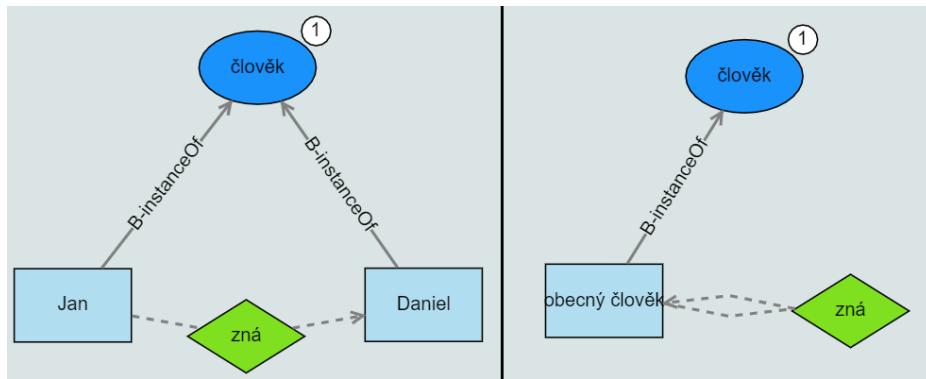
Následující příklad na Obr. 28 popisuje instance osob, jejichž relace s okolím není možné v reálném světě kombinovat, avšak jejich sloučení by z hlediska modelu bylo správně.



Obr. 28: Sloučení instancí a linků k relacím.

Je pravděpodobné, že osoba chodící do školky nemůže zároveň studovat vysokou školu, avšak při sloučení do jedné instance model přesně takovou situaci popisuje. Model tedy ztrácí potenciální vyjadřovací schopnost. Na příkladu je stále velmi dobře patrné, že obecný člověk je pouze reprezentací všech lidí a jejich možných vlastností. Při převodu do OWL jsou vlastnosti přeneseny přímo na entitu „člověk“. Avšak při určitých situacích by mohla nastat situace, kdy model nepopisuje realitu tak, jak autor zamýšlel.

Při situacích, kdy jsou spolu dvě instance shodného b-typu ve vztahu může nastat problém se schopností PURO popisovat cyklické vztahy. Pokud model vyjadřuje, že jedna osoba zná druhou, bude výsledkem sloučení v obecnou instanci situace, kdy obecný člověk zná sám sebe.



Obr. 29: Příklad cyklického vztahu

V této specifické situaci na Obr. 29 se jedná o přehledný a čitelný vztah. Při převodu do OWL se vytvoří validní relace vyjadřující, že člověk může znát člověka. Avšak tato přehlednost může být do velké míry dána přirozenou doménovou znalostí vztahů mezi lidmi.

4.3 Skóre podobnosti entit

V kapitole 4.1 byla popsána potřeba tvorby systému pro vyhodnocení podobnosti dvou entit. Míra podobnosti entit pak může sloužit pro informování tvůrce modelu o tom, že jsou si entity mezi dvěma modely dostatečně podobné. Předpokladem je vhodně zadaný práh této míry. Uživatel by měl mít možnost tento práh upravovat na základě pozorovaného sloučení. Vhodné je též uživateli nabídnout manuální úpravu každého uzlu slučovaného modelu. Pro každý uzel by pak bylo možné vybrat, zdali má být unikátní, nebo má být sloučen s nějakým uzlem v původním modelu.

Zjevně nejsilnějším předpokladem pro sloučení je shodnost typu u obou entit, dále pak stejné jméno a v poslední řadě jejich okolí.

Toto logikou však nelze postupovat u všech typů entit a je nutné respektovat určitou hierarchii. Lze předpokládat, že například b-type by měl mít unikátní název. Podobně lze postupovat u objektů, tedy instancí b-typů. Zde je však nutné počítat s výše uvedenými příklady shodnosti. Název firmy a jméno člověka může být shodné. Jejich b-typ je však pravděpodobně odlišný a z toho lze pak usuzovat, že se nejedná o stejnou entitu. Nasnadě je také možnost shodného jména například u lidí. Zde je pak nutné posuzovat jejich okolí a vyvodit potenciální sloučení. Ostatní entity pak vycházejí ze vztahů mezi typy a objekty a jejich unikátnost napříč modelem není nijak zaručena. Například atribut „váha“ tak může příslušet nepřebernému množství objektů. Cena může vycházet z relace mezi prodejcem a mnoha různými produkty. U entity b-valuation pak v podstatě vůbec nelze spoléhat na vyhledávání shodnosti pomocí typu a názvu. Vzhledem k tomu, že se jedná například o konkrétní cenu, váhu, rozměry, či datum, lze usuzovat, že shodných entit k různým atributům může být nepřeberné množství.

Proto je nutné uvažovat u každého typu a objektu o určitém okolí. Tyto entity jsou totiž stále velkou měrou unikátní. Například konkrétní člověk, jakožto objekt má atributy váhu, či výšku. Stejné atributy má však třeba notebook. Tyto atributy se stejnými názvy tedy nejsou vůbec shodné proto, že přísluší jiné nadřazené entitě.

Podobně relace shodného názvu i podstaty nejsou shodné kvůli příslušnosti k jinému typu, či objektu. Příkladem může být „místo narození“ pro různé osoby.

Slučování modelů tedy vychází z následujících základních závislostí:

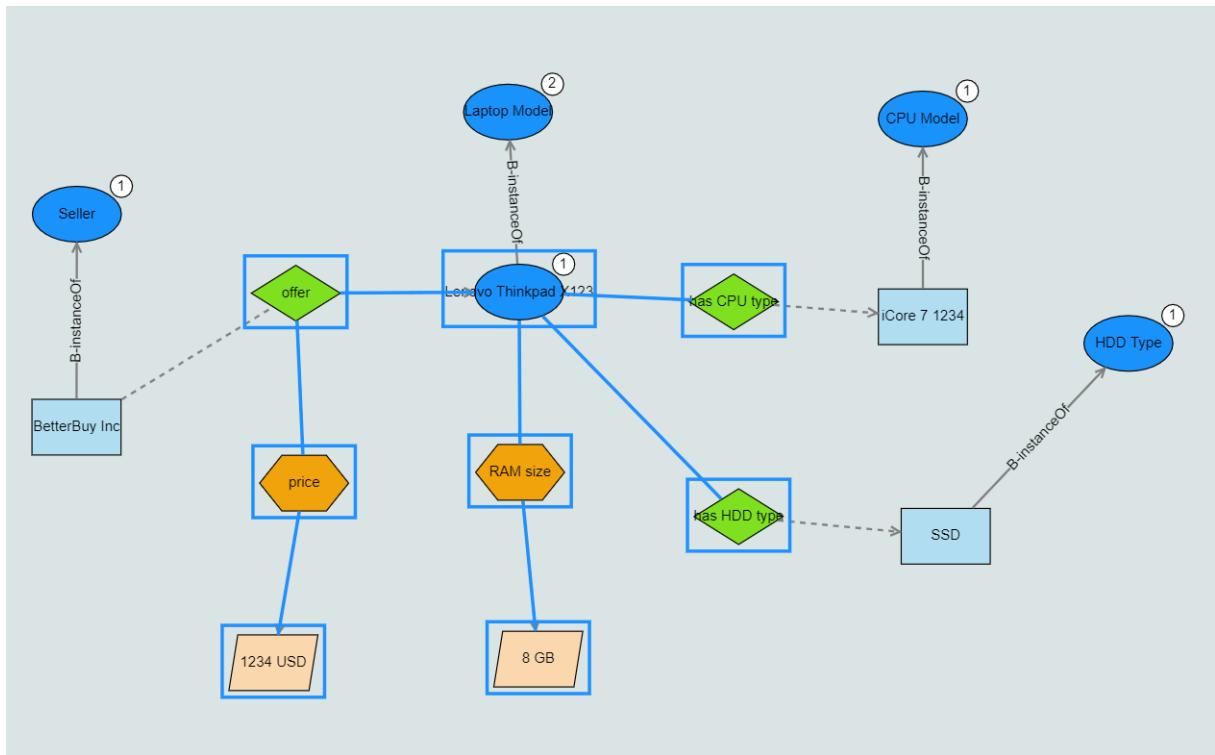
1. B-valuation nemůže existovat bez atributu. Zároveň je nutné zamýšlení, zdali je možné, aby atribut byl ve vztahu s více hodnotami.
2. B-attribute je závislý na relaci, objektu, či typu.
3. Relace je závislá na objektu, či typu.
4. Objekt je instance typu.

Pro vyhledávání ekvivalentních entit je nutné uvědomovat si tuto podřazenost atributů, hodnot a relací. Z tohoto důvodu lze vytvořit abstraktní koncept, který bude ve zbytku práce označován jako „okolí uzlu“

4.4 Okolí uzlu

Vyhledávání párů entit napříč modely lze v nejjednodušší podobě uvažovat jako srovnávání podobnosti názvů za předpokladu shodného typu entity. Je tedy nutné každý uzel ze slučovaného grafu porovnat s každým uzlem stejného typu v původním grafu. Pokud je název dostatečně shodný, je možné uvažovat o sloučení.

Z problematiky uvedené výše lze však pro toto srovnávání brát v úvahu pouze typy a objekty. U všech ostatních entit nelze ze samotného názvu vyvozovat slučování. Proto je nutné tyto uzly porovnávat v kontextu okolí nadřazeného uzlu.



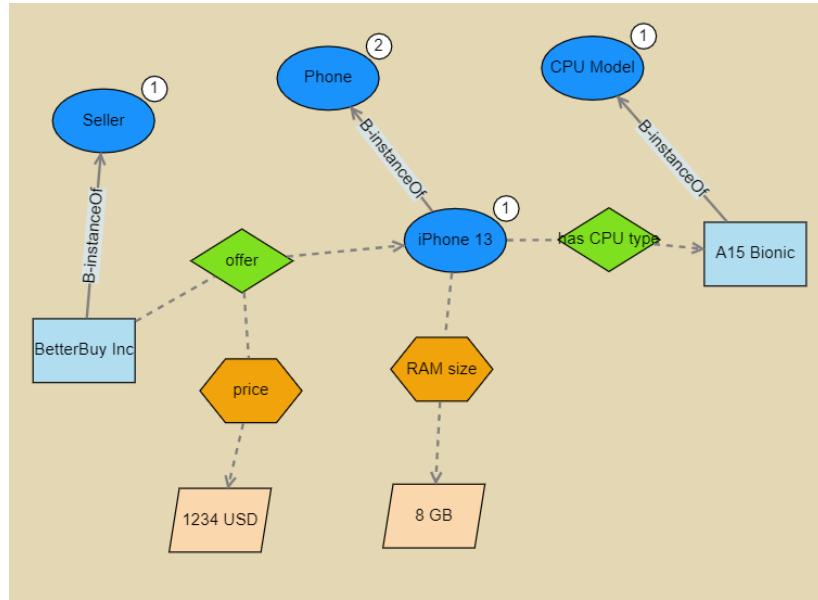
Obr. 30: Příklad okolí uzlu

Na Obr. 30 je označeno okolí uzlu „Lenovo Thinkpad X123“. Všechny přímo napojené relace, atributy a valuace jsou bez přiřazeného b-typu, či b-objektu defacto nepotřebné. Například atribut „velikost RAM“ by nemusel bez popisovaného notebooku existovat. Stejně tak by relace „nabídka“ neměla smysl bez nabízeného produktu, nebo entity, která produkt nabízí.

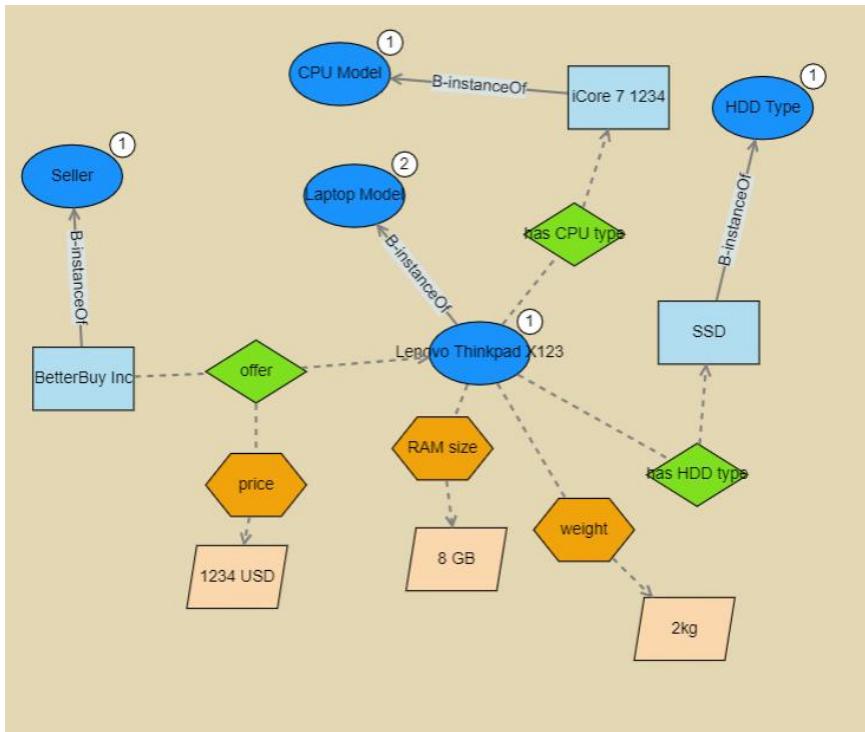
Okolí uzlu kategorie typ, či objekt lze nalézt pomocí následujícího algoritmu. Ten vyhledává okolí pro určitý uzel U:

1. Vytvoř pole okolí a vlož do něj uzel U.
2. Dokud je ještě co přidávat, tedy pokud od posledního cyklu proběhla změna v poli *okolí*, prováděj následující kroky:
 - a. Pro každý uzel v poli *okolí* vyhledej připojené uzly.
 - b. Pokud tyto připojené uzly náleží k jiné kategorii než typ, či objekt, a pokud ještě samy nejsou zahrnuté v poli *okolí*, přidej je do tohoto pole.

Takto lze vyhledat okolí pro oba v potenciálním páru uzelů v původním i slučovaném grafu. Pokud pak bude v tomto okolí nalezen uzel reprezentující atribut, relaci, či valuaci, lze předpokládat, že se jedná o tentýž uzel. Při slučování dvou uzelů kategorie typ, či objekt je pak jejich okolí se stejnými, či podobnými názvy taky sloučeno.



Obr. 31: OBM popisující nabídku telefonu



Obr. 32: OBM popisující nabídku notebooku

Výše vyobrazené modely popisují nabídky telefonu (Obr. 31) a notebooku (Obr. 32). Z obou modelů je patrné, že mnoho uzelů je shodných a podle prosté logiky vyhledávání stejné kategorie a názvu uzlu by pak sloučení nedopadlo podle představ. Je možné předpokládat, že prodejce je v obou případech opravdu

shodný. Stejně tak typ „CPU Model“ je v obou případech totožná. Zbytek uzlů, jako je váha, velikost RAM, či vztahy „nabídka“ s prodejcem by však měl zůstat unikátní.

Vytvořením okolí uzlů kategorie objekt a typ lze této situaci předejít. Nižší uzly budou totiž navzájem porovnávány pouze pokud jejich nadřazený uzel bude shodný s uzlem v druhém modelu.

Díky porovnávání uzlů na základě okolí se tedy nestane, aby se uzly kategorie b-valuation, b-attribute, či b-relation náležící typům „Lenovo Thinkpad X123“ a „iPhone 13“ dostaly do situace, kdy by byly porovnávány s ekvivalentem v druhém modelu.

Uzel „offer“, který popisuje vztah nabídky notebooku, či telefonu prodejcem je však stále problematický. Tento vztah je rozšířen atributem cena. Cena notebooku a telefonu by spolu zjevně neměla nijak souviset, ačkoli je v obou příkladech stejná pro demonstraci faktu, že přes stejnou cenu nelze uvažovat o jakémkoli vztahu.

Pokud by se tento uzel nabídky jednoduše sloučil a měl propojení k obou nabízeným produktům, nezbyl by prostor pro definici různých cen. V algoritmu slučování lze však jednoduše porovnávat okolí každého typu a objektu nezávisle. Pokud by tedy algoritmus nejdříve porovnával uzel prodejce, zajisté by vyhodnotil uzel nabídky jako již existující a tento by poté byl určen jako kandidát na sloučení s původním. Po určitém počtu cyklů by však algoritmus začal porovnávat okolí uzlu produktu. Pro něj by opět vyhledával ekvivalent uzlu nabídky, a protože je produkt unikátní, nebylo by možné nalézt páry pro uzel nabídky. Tento uzel, v předchozích krocích vyhodnocen jako kandidát na sloučení, by byl změněn na unikátní uzel.

Pokud by však algoritmus postupoval opačně, byl by uzel nabídky nejdříve určen jako unikátní a později jako slučovaný uzel. Proto je nutné vytvořit pole přidávaných uzlů. Pokud je jednou jakýkoli uzel přidán do tohoto pole, nemůže již být nikdy vyhodnocen jako uzel slučovaný. Tímto postupem je zajištěno, že nabídka bude vždy unikátní pro jakýkoli produkt.

4.5 Nalezení souvislého grafu

Pro vyhodnocení úspěšnosti spojení dvou modelů může posloužit hledání nesouvislého grafu. Nesouvislý graf je v případě PURO modelu takový, ve kterém se nelze dostat z jeho libovolného uzlu do jiného libovolného uzlu v celém grafu. Tento proces není přímo nutný v samotném slučování modelů, ale může poskytnout uživateli aplikace dobré informace o výsledku slučování dvou modelů.

Pokud graf není souvislý, lze jeho části, které souvislé jsou, obecně nazývat komponentami souvislosti.

Z podstaty nesouvislého grafu lze předpokládat následující skutečnosti:

- Pokud je výsledkem slučování určitého množství modelů nesouvislý graf o stejném počtu komponent souvislosti jako byl počet slučovaných modelů a pokud jsou všechny uzly každé z komponent souvislosti shodné alespoň s jedním slučovaným modelem, pak nebyl napříč modely určen ke sloučení žádný páry uzlů.
 - Požadavek na shodnost komponent souvislosti s alespoň jedním slučovaným modelem, ne však právě s jedním, vychází z předpokladu, že některé slučované modely mohou být shodné v každém uzel.

- Pokud je počet komponent souvislosti různý od počtu slučovaných modelů, avšak uzly těchto komponent jsou shodné právě s jednou kombinací průniků slučovaných modelů, pak proběhlo sloučení pouze částečně.
- Pokud byly nalezeny nesouvislé grafy a jejich uzly nejsou shodné s žádným slučovaným modelem, či jejich kombinací, pak došlo k nalezení nějakých párů uzlů ke sloučení, ale v procesu slučování došlo ke ztrátě nějakého propojení mezi uzly.

Tyto skutečnosti však vycházejí z předpokladu, že každý ze slučovaných modelů byl sám souvislým grafem.

Algoritmus pro vyhledávání komponent souvislosti je následující:

1. Vytvoř pole „*původní*“ se všemi uzly grafu.
2. Vytvoř pole „*podgrafy*“, které bude obsahovat jednotlivé komponenty souvislosti.
3. Pokud se v poli *původní* nachází nějaký uzel, prováděj následující:
 - a. Vytvoř dočasné pole „*graf*“, popisující uzly v komponentě souvislosti.
 - b. Dokud se počet uzlů v poli *graf* neprestane zvyšovat, prováděj následující:
 - Pro každý uzel v poli *graf* vyhledej připojené uzly. Pokud tyto uzly ještě nejsou přidány do pole *graf*, přidej je.
 - c. Přidej pole *graf* do pole *podgrafy*.
 - d. Odeber všechny uzly obsažené v poli *graf* z pole *původní*.

Pole podgrafů pak obsahuje jednotlivé komponenty souvislosti. Z jejich množství a vztahů lze pak vyvzovat úspěšnost sloučení jednotlivých modelů. Pokud je nalezených komponent souvislosti více, nejspíš nastala při slučování chyba

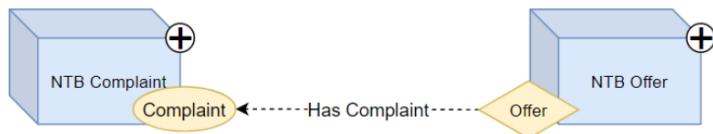
5 Grafická reprezentace

Součástí práce je návrh metod pro zjednodušení grafické reprezentace PURO modelu.

Původní PURO Modeler se stává nepřehledným při tvorbě rozsáhlých grafů a nenabízí žádnou možnost, jak model automaticky zjednodušit. Slučování dílčích nezávisle vytvořených modelů je zásadní způsob, jak se tomuto problému vyhnout. Pomocí vhodně navržených grafických elementů lze však předejít nutnosti slučování.

5.1 Skupiny

Jedním ze zásadních zlepšení přehlednosti je tvorba skupin uzlů. Způsobem, jak zpřehlednit návrh OBM v prostředí PURO Modeleru by mohla být implementace možnosti minimalizace částí grafu. U entity, na kterou jsou napojeny další uzly by se mohly minimalizovat uzly s odchozím, či příchozím vztahem.



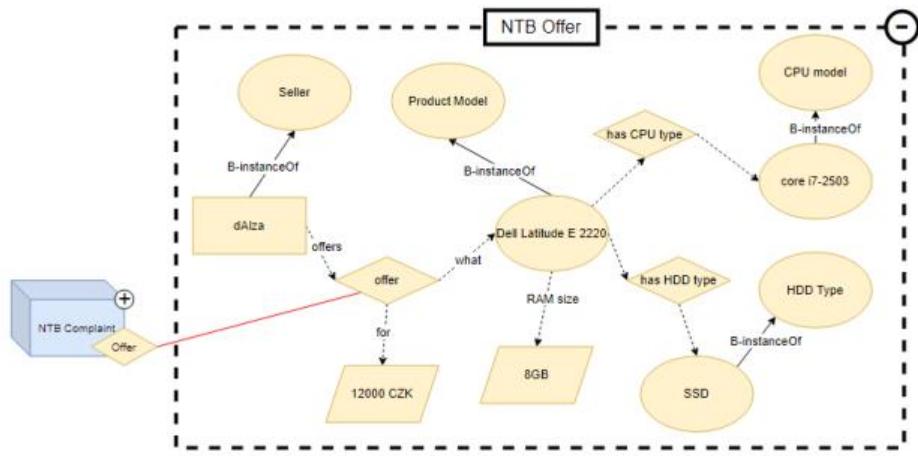
Obr. 33: Propojení složek přes různé entity

Na Obr. 33 lze vidět návrh propojení dvou složek. NTB Offer vyjadřuje minimalizovaný graf obsahující informace o notebooku. NTB Complaint zahrnuje graf popisující reklamaci. Při jejich propojení byly zvoleny dva rozdílné uzly, které na sebe logicky navazují. Uživatel má možnost oba grafy nezávisle maximalizovat a upravit. Další možnosti sloučení více modelů je přímé propojení stejných entit, které se vyskytují v obou modelech.



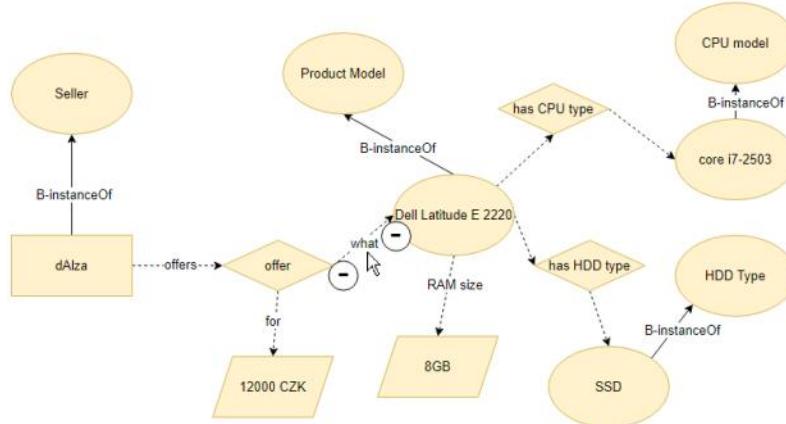
Obr. 34: Propojení složek přes shodnou entitu

Přístup na Obr. 34 podporuje tvorbu shodné entity na více místech, což nemusí být na první pohled tak intuitivní jako předchozí možnost. Výhodou však je, že vztah mezi výstupními body pouze vyjadřuje jejich propojení a zároveň jejich shodnost a do konečné ontologie se nepřenese. Při maximalizaci takto spojených modelů by tato propojovací entita byla sloučena v jednu do té doby, než by byly oba modely opět minimalizovány. Při maximalizaci jedné ze složek by propoj vstupoval do této složky (viz Obr. 35)



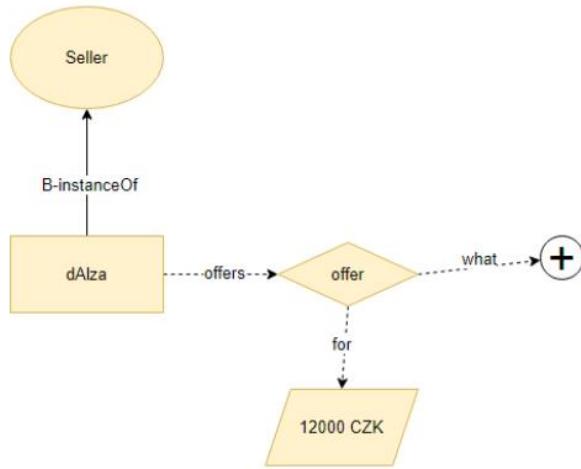
Obr. 35: Příklad rozbalené složky

Jakou formou by minimalizace probíhala je otázkou uživatelského komfortu a přehlednosti. Jednou z možností jsou ikony, zobrazující se při najetí kurzoru na linku propojení dvou entit vyjadřující vztah. Uzel na příslušném konci tohoto propojení by se minimalizoval a skryty by byly všechny ostatní uzly propojeny s tímto uzlem, krom entity na lince propojení, na které minimalizace nastala.



Obr. 36: Nabídka minimalizace

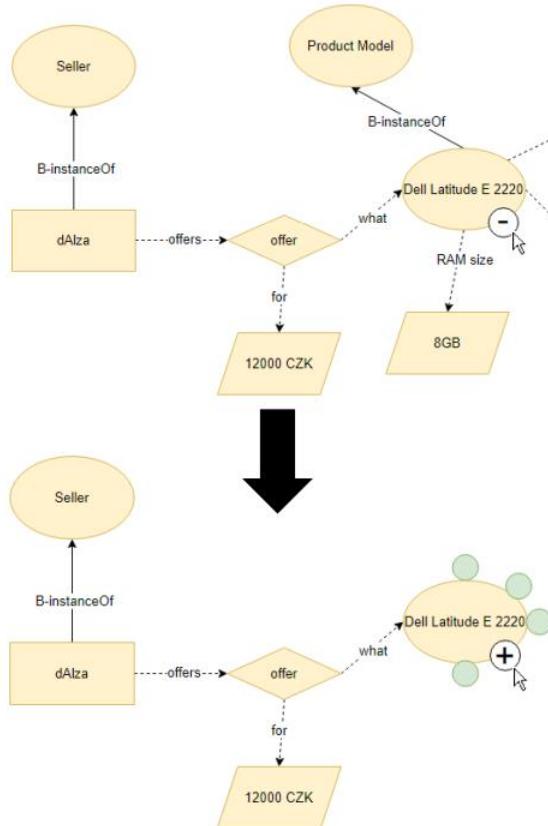
Na Obr. 36 je vidět situace po najetí kurzorem na vztah mezi dvěma entitami. Zobrazí se dvě ikony umožňující minimalizaci jedné ze stran vztahu.



Obr. 37: Minimalizace části grafu

Obr. 37 popisuje stav, kdy je část OBM minimalizována. Uzel s entitou notebooku je vyjádřen pouze jako ikona se znaménkem, vyjadřující minimalizaci. Vztah mezi uzlem offer a notebookem je však zachován pro přehlednost.

Další možností by mohla být minimalizace všech odchozích propojení a uzlů. Při najetí kurzorem na uzel by se zobrazila ikona umožňující minimalizaci. Po kliknutí by byly odchozí vztahy nahrazeny pouhými body vyjadřujícími existenci těchto propojení (viz Obr. 38).



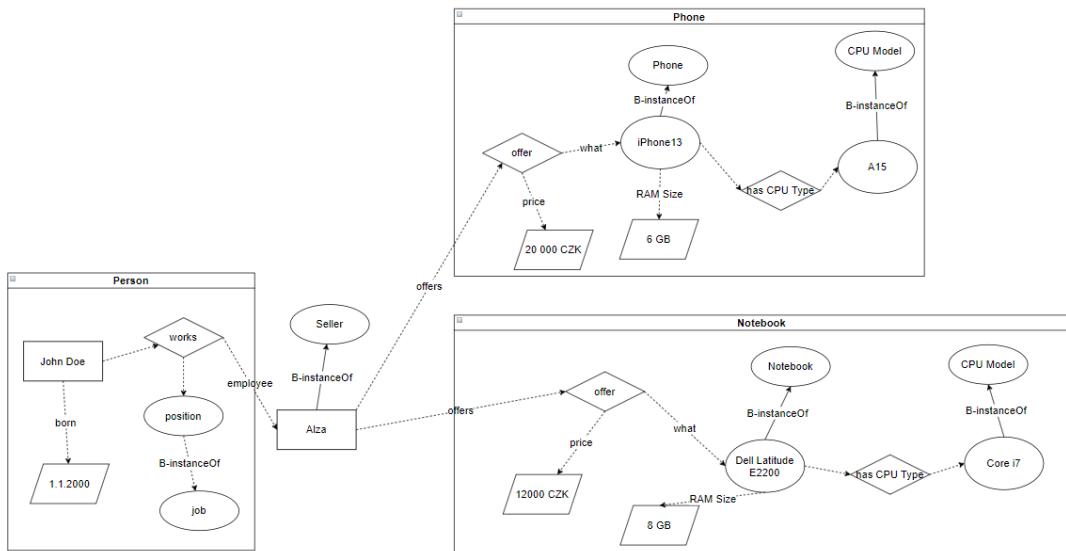
Obr. 38: Minimalizace odchozích propojení

Faktem zůstává, že jakýkoli přístup neumožňující zobrazení pouze určité části sloučeného OBM při kontrole před převodem do ontologie pomocí nástroje OBOWL Morph však narází znovu na původní omezení PURO Modeleru kvůli nepřehlednosti. Ostatně i samotný náhled v OBOWL Morph je omezen počtem uzlů ze stejného důvodu.

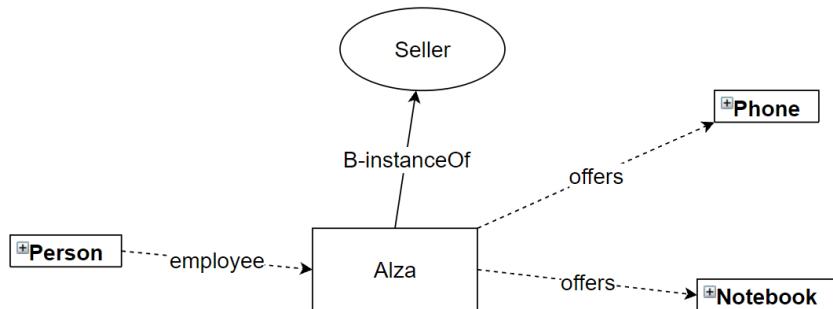
Všechny výše uvedené přístupy zároveň předpokládají, že minimalizovaná část není propojena se zbytkem modelu ve více než jednom uzlu. Toto velmi omezuje využití v rozsáhlých ontologiích. Proto by měla být minimalizace v co největší míře v rukou uživatele.

Uživatel by například mohl vybrat konkrétní uzly, které mají být minimalizovány. V tomto přístupu by nemusela být vynucována žádná omezení a předpokládalo by se, že tvůrce ontologie ví nejlépe, do jakých podskupin uzly shlukovat.

V tomto případě je nutné uvažovat několik způsobů, jak je možné reprezentovat vztahy mezi skupinami. Do skupiny by totiž bylo možné zahrnout více uzlů, jejichž propoj skupinu opouští.

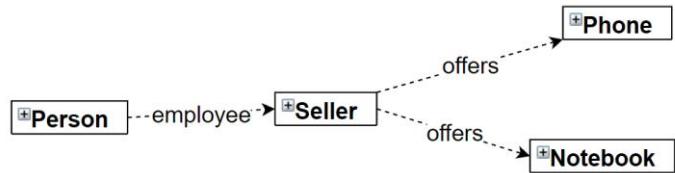


Obr. 39: Vztahy entit mezi složkami



Obr. 40: Vztahy entit po zabalení složek

Výše je příklad tří skupin před (Obr. 39) a po minimalizaci (Obr. 40). Každá skupina má pouze jeden uzel, který je propojen s okolím. Proto jsou vztahy při zjednodušeném náhledu stále patrné a přehledné.



Obr. 41: Zabalení všech entit do složek

Při zabalení posledních dvou uzlů (viz Obr. 41) lze již jen těžko usuzovat, který uzel z minimalizovaných skupin je ve vztahu vycházejícím mimo skupiny. Tvůrce ontologie si dost možná pamatuje jednotlivé vztahy a ostatní uživatelé si mohou skupiny rozbalit. Otázkou je, zdali by bylo vhodné tyto vztahy lépe označit, například ikonou reprezentující uzel, ke kterému vztah náleží.

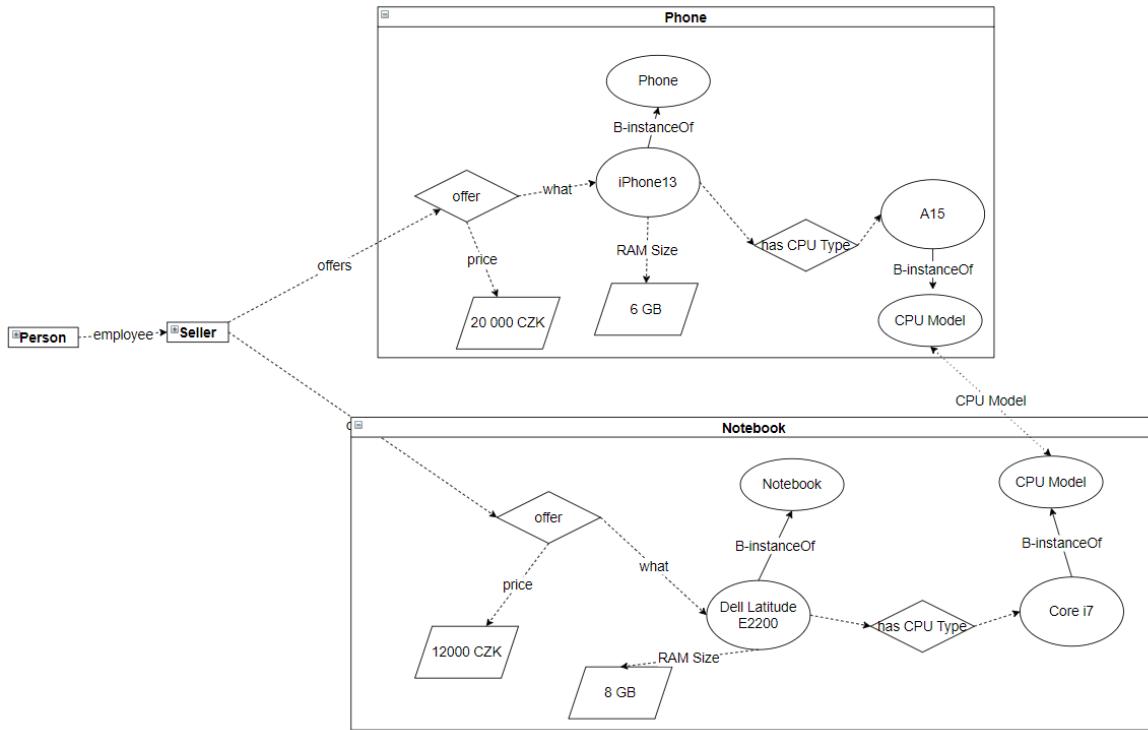
5.2 B-typy

Z obrázků výše bylo patrné, že notebook a telefon nesou informaci o procesoru. Konkrétní procesor náleží typu CPU Model. Tento typ by v modelu měl být zastoupen pouze jednou, protože v obou případech označuje to samé. Uživatel by však měl mít možnost rozhodnout se, zdali tento typ bude reprezentovat v modelu jeden uzel, nebo více uzlů, které se však při transformaci do OWL sloučí v jeden.

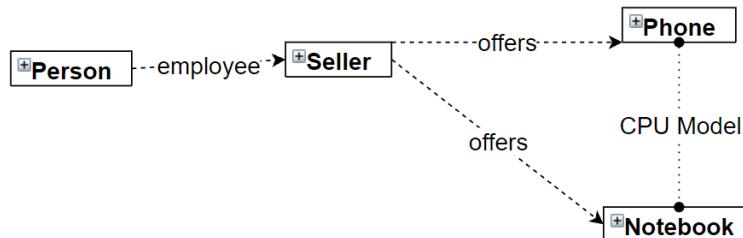
V průběhu návrhu modelu by pak při tvorbě nového typu byla uživateli poskytnuta možnost tento uzel spřáhnout s již existujícím typem. Uživatel by pak měl možnost volby mezi uzly graficky reprezentovanými nezávislými ikonami a sloučením těchto ikon v jeden uzel.

První případ zabraňuje přílišnému zahlcení linkami vztahů mezi uzly, avšak z grafické reprezentace modelu se vytrácí informace o případných vztazích. Například mezi notebookem a telefonem z předchozích příkladů je patrný vztah přes uzel typu „CPU Model“, ale při minimalizaci skupin se tento vztah vytrácí.

Jedním z řešení je nabídnout uživateli možnost zobrazit dočasný propoj mezi téměř typy. Linka mezi typy by tak vyjadřovala, že jsou shodné. I při zabalení skupin by tak bylo možné vyvodit vztah přes tento typ.



Obr. 42: Výjádření shodnosti b-typu



Obr. 43: Výjádření shodnosti b-typu mezi zabalenými složkami

Na obrázcích výše je patrný vztah mezi složkou zahrnující notebook a telefon přes typ CPU Model před (Obr. 42) a po zabalení složek (Obr. 43). Tento vztah by bylo vhodné zobrazovat pouze na přání uživatele.

6 Dostupné knihovny pro grafickou reprezentaci

Samotné slučování PURO modelů by se mohlo obejít bez aplikace s grafickou reprezentací grafů. Pro různé úpravy modelů a pro případný vývoj aplikace umožňující zjednodušování náhledu PURO modelů na základě myšlenek z předchozích kapitol je však nutné navrhnout, či využít již existující knihovny pro podporu zobrazování diagramů. Vzhledem k tématu této práce by bylo nadbytečné vyvíjet tyto knihovny od začátku. Je proto nutné vybrat již existující prostředí. K tomu je nutné shrnout požadavky na funkcionality takové knihovny.

6.1 Požadavky

Pro podporu vytváření, úpravy a slučování PURO Modelů v grafickém prostředí je nutné zobrazovat uzly a vztahy mezi nimi. Pro zobrazování dvou slučovaných modelů by mohl teoreticky postačovat statický obrázek vykreslený na základě vstupních hodnot. Jakýkoli rozsáhlejší graf by se však při statickém zobrazení stával velmi rychle nepřehledným. Zároveň by statický náhled neumožňoval aplikaci návrhů na zlepšení PURO Modeleru po stránce jeho grafické reprezentace. Proto je nutné zobrazovat objekty dynamicky s různými funkcemi jako je přesouvání uzelů, změna velikosti náhledu, podpora shlukování a pokud možno replikace většiny funkcí z PURO Modeleru.

6.1.1 Podpora různých grafických elementů

Vytvářené uzly spadají do jedné z pěti kategorií, utvářejících jazyk PURO. Je vhodné tyto uzly reprezentovat, pokud možno stejně, jako tomu bylo v původním PURO Modeleru. Je tedy nutné, aby knihovna umožňovala zobrazování různých tvarů a barev uzelů.

Vztahy mezi uzly mohou být též různé a jsou definovány jazykem PURO. Knihovna by měla podporovat různé vzhledy těchto vztahů.

6.1.2 Podmiňování tvorby vztahů

Existence vztahu je podmíněna zahrnutými uzly. Některé vztahy nelze vytvořit mezi libovolnou kombinací kategorií entit. Proto je nutné při vytvoření nového propojení zohlednit, jaký je výchozí a cílový uzel. Některé uzly mohou mít více druhů vztahů, takové je pak nutno uživateli vhodně nabízet.

Zároveň je vhodné ohlídat, zdali je vůbec možné vztah mezi dvěma uzly vytvořit. Uživatel by měl být s jazykem PURO seznámen a měl by tedy být schopen posoudit, jakého druhu je vztah, který mezi dvěma uzly vytváří. Lepší je však zcela zabránit možnosti vytváření vztahů, které jazykem PURO nejsou podporovány.

Dále je vhodné omezit počet propojení vycházejících z, či do uzlu.

6.1.3 Kontextová menu

Při vytváření vztahu mezi dvěma entitami nemusí být zřejmé, o jakou kategorii vztahu se má jednat. V tomto případě je nutné nabídnout uživateli validní možnosti. Vhodným řešením je zobrazení kontextové nabídky potenciálních kategorií vztahů právě při události vytváření nového propojení.

Podobně lze řešit nabídku potenciálních párů uzlu při slučování více modelů. Uživatel by pak měl možnost zobrazit vhodné kandidáty na sloučení pro konkrétní uzel slučovaného modelu. Tento způsob je zřejmě přehlednější než pouhý seznam uzlů s přiřazenou nabídkou, jako by tomu bylo v případě slučování modelů bez grafické reprezentace uzlů.

Dalším vhodným využitím kontextových nabídek je třeba vytváření nového uzlu. Intuitivním a uživatelsky přístupným řešením může být kliknutí na prázdný prostor diagramu, kde se pak zobrazí nabídka potenciálních uzlů. Tato funkce může být doplněna podoknem na okraji diagramu se stejnou nabídkou. Uzel by pak mohl být vytvořen přesunutím z této nabídky. Takové okno se obecně nazývá paleta a grafická knihovna by ho měla v ideálním případě podporovat.

6.1.4 Složky a skupiny

Důležitým rozšířením původního PURO Modeleru je rozřazování částí modelu do složek, či skupin. Je tedy nutné graficky rozlišit uzly grafu, které spadají pod uživatelem vytvořenou skupinu. Takovéto skupiny by pak mělo být možné minimalizovat a zvýšit tak přehlednost celého modelu.

6.1.5 Formáty ukládání

Pro ukládání a export vytvořených modelů je také žádoucí podpora jednoduchého a přehledného výstupu. Předpokládá se formát, který odlišuje název modelu a alespoň dvě podskupiny, nesoucí informace o uzlech a jejich vztazích.

6.2 Knihovny

Dostupných knihoven pro vykreslování grafů na webu je nespočet. Vzhledem k zaměření práce je vhodné využít takové prostředí, které by nativně podporovalo co největší množství výše uvedených požadavků.

6.2.1 mxGraph

mxGraph³ je JavaScript knihovna podporující tvorbu grafů a diagramů dostupná pod licencí Apache 2.0 [22]. Knihovna obsahuje přehledný návod a dokumentaci. Nevýhodou relativně malá podpora ze strany vývojářů. Uživatelský manuál je spíše přehledem vlastností knihovny a nevěnuje se funkcionalitám tolik do hloubky. Vyhledávání jednotlivých funkcí a tříd je velmi obtížné. Knihovna má na webové stránce Stack overflow označených přes 400 dotazů, ale velká část z nich je relativně málo navštěvovaná ostatními uživateli a mnoho dotazů není zodpovězeno [23].

³ Knihovna je dostupná na: <https://jgraph.github.io/mxgraph/>

Knihovna nabízí možnosti seskupování uzelů, avšak zobrazování takových skupin je velmi omezené. Autor nebyl schopen v dokumentaci dohledat popis průběžného ukládání změn v modelu, které by jednoduše umožňovalo vzít nějaké změny zpět. Zároveň autor nebyl schopen nalézt nativní podporu force directed rozložení, či dynamického přesunu uzelů. Při pohybu uzlem grafu není vizuálně patrná žádná změna, dokud uživatel přesun nedokončí, což je poměrně rušivé.

6.2.2 Vis.JS

Knihovna určená pro vizualizaci grafů a je dostupná pod licencí Apache 2.0. Umožňuje vykreslování sítí, grafů v 3D prostoru a klasických dvourozměrných grafů pro popis dat⁴. Na webu Vis.JS je dostupná spousta příkladů. Díky tomu lze jednoduše posoudit vhodnost této knihovny pro vizualizaci slučování PURO modelů.

Pro účely aplikace PURO modeleru by byla vhodná varianta knihovny Vis Network. Dokumentace této knihovny⁵ je přehlednější než v případě mxGraph a nabízí poměrně velké množství konkrétních příkladů. Oproti knihovně mxGraph nabízí dynamické zobrazování uzelů a jejich plynulé přesouvání. S tím se pojí podpora force directed rozložení, které zpřehledňuje náhled na model a usnadňuje orientaci.

Knihovna nabízí určitou formu shlukování uzelů a jejich zabalování do jednoho nového uzlu, nebyla však nalezena podpora rozbalení složky a zobrazení jejího obsahu. Není možné tvořit hierarchii složek a v této hierarchii postupně jednotlivé složky rozbalovat. Jedinou formou je tak zabalení do určitého množství uzelů do jednoho a poté případně jeho opětovné rozbalení. Uzel reprezentující shluk však zaniká. Autor nebyl schopen nalézt dokumentaci pro podporu kontextových menu.

6.2.3 GoJS

GoJS⁶ je knihovna v napsaná v čistém JavaScriptu. Umožňuje vytváření mnoha druhů diagramů a grafů. Výhodou je rozšiřitelnost a nezávislost na dalších knihovnách. V balíku je již zahrnuto množství užitečných funkcí.

Společnost spravující knihovnu zároveň provozuje vlastní diskusní fórum s nepřeberným množstvím vláken pro řešení problémů a pro sdílení užitečných postupů a funkcí. Součástí této knihovny je velmi přehledná a rozsáhlá dokumentace. Zároveň lze na stránkách nalézt podrobný návod, věnující se mnoha aspektům knihovny. Zde je možné dohledat příklady k prakticky všem zásadním funkcionalitám a objektům. Stránky obsahují mnoho demonstrací různých druhů diagramů s přehledným popisem a dostupnou dokumentací včetně kódu. Tímto způsobem se uživatel jednoduše dostane ke spolehlivým a ověřeným postupům přímo pro konkrétní využití. V neposlední řadě je pak dostupný základní návod na zprovoznění nejdůležitějších prvků. Pokud tedy vývojář potřebuje pro svůj projekt velmi přístupnou, ale zároveň v mnoha ohledech rozšiřitelnou knihovnu, je GoJS velmi dobrou volbou. Nevýhodou je placená licence pro komerční využití. Pro akademické účely je však zdarma.

⁴ Všechny varianty knihovny jsou dostupné na: <https://visjs.org/>

⁵ Dostupné na: <https://visjs.github.io/vis-network/docs/network/>

⁶ Dostupné na: <https://gojs.net/>

Základním prvkem vizualizace v této knihovně je třída `Diagram`. Při vytvoření instance diagramu je možné nadefinovat obecné vlastnosti diagramu.

Klíčové vlastnosti:

- **Používání klávesových zkratky:** Knihovna nativně podporuje používání klávesových zkratky pro většinu standardních operací. Toto chování lze rozšiřovat, zakazovat, či chování některých zkratky přepsat jinou vlastní funkcí. Využití zkratky pro mazání uzlů, přejmenovávání, či kopírování uzlů z clipboardu zvyšuje uživatelský komfort.
- **UndoManager:** Pro uživatelský komfort je vhodné využít třídu `UndoManager`. Tato třída sleduje změny v diagramu a dává uživateli možnost vzít zpět určité kroky při návrhu modelu. Vývojář nepotřebuje definovat obecné události, jako je změna názvu uzlu, vytvoření linku a přidání či odebrání uzlu. Tyto jsou sledovány automaticky. Pokud je ve vlastním kódu upravován diagram, je nutné tuto informaci třídě sdělit pomocí metody `commitTransaction`. Výhodou metody je, že je pod jednu transakci možné vložit více kroků naprogramovaných úprav. Například při odstraňování redundantních linků mezi uzly je možné všechny linky odstranit jako jednu transakci. Pokud se pak uživatel rozhodne vzít krok zpět, stačí mu k tomu jeden požadavek.
- **Force-directed rozložení:** Knihovna umožňuje velmi jednoduchou implementaci tohoto rozložení definicí „náboje“, maximální délky linku a počtem iterací. Uzly se pak přirozeně odpuzují a nedochází k jejich překryvu, což zvyšuje přehlednost grafu.
- **Sledování událostí:** Diagramu lze nadefinovat širokou škálu událostí, kterým lze pak přiřadit funkci. Pro příklad lze zmínit sledování, zdali byl vytvořen nový link, zdali byl vybrán uzel, či jestli byl upraven název uzlu. Poslední zmíněná událost je užitečná pro zajištění unikátních názvů pro uzly b-type.
- **Definice stylu:** Diagramu je možné nadefinovat různé styly uzlů a linků mezi nimi. Pro každý styl lze zvolit nejen vzhled, ale i specifické chování. Například při změně údajů o uzlu lze automaticky změnit barvu, tvar, okraje apod.
- **Vytváření skupin:** Velmi užitečným nástrojem je definování vzoru, podle kterého se budou vytvářet grafické reprezentace skupin uzlů. Vývojář tak může navrhnut různé styly boxů, do kterých pak budou umístěny vybrané uzly. Tyto skupiny je pak možné dynamicky minimalizovat a poskytnout tak lepší přehlednost celého modelu. Skupiny můžou vytvářet vlastní hierarchii zapouzdřením více skupin do nadřazené skupiny.

GoJS nabízí nespočet užitečných metod, jako jsou iterátory přes uzly a linky v grafu či vyhledávání skupin uzlů propojených s vybraným uzlem.

Diagramů lze vytvořit více a je možné mezi nimi přesouvat údaje. Pokud se tedy vývojář rozhodne sloučit dva modely, je mu nabídnuto prostředí se dvěma grafy, které jsou spravovány jinými diagramy. Témto diagramům lze nadefinovat jiné vlastnosti a lze tak například zabránit úpravám grafu v průběhu slučování.

6.2.4 Srovnání knihoven

Na základě výše uvedených charakteristik knihoven autor sestavil hodnocení jednotlivých vlastností (viz Tab. 2). Každá vlastnost je oceněna body na škále od 0 do 5 na základě rozsahu a kvality podpory vlastnosti a reflekují subjektivní názor autora. Nejnižší nula je v případě nenalezení podpory potřebné vlastnosti.

Tab. 2: Vyhodnocení vlastností knihoven.

Vlastnosti	mxGraph	Vis.JS	GoJS
Podpora vizuálního odlišení uzlů	3	3	5
Různé druhy linků dle kontextového menu	3	0	5
Kontextová menu	4	0	5
Tvorba složek	3	1	5
Vhodné formáty pro export	5	5	5
Podpora force directed rozložení	0	5	3
Dynamický přesun elementů	0	5	5
Podpora palety	3	0	5
Podpora vrácení změn v diagramu	0	0	5
Kvalita a počet příkladů	3	2	5
Přehlednost dokumentace	2	4	5
Suma:	26	25	53

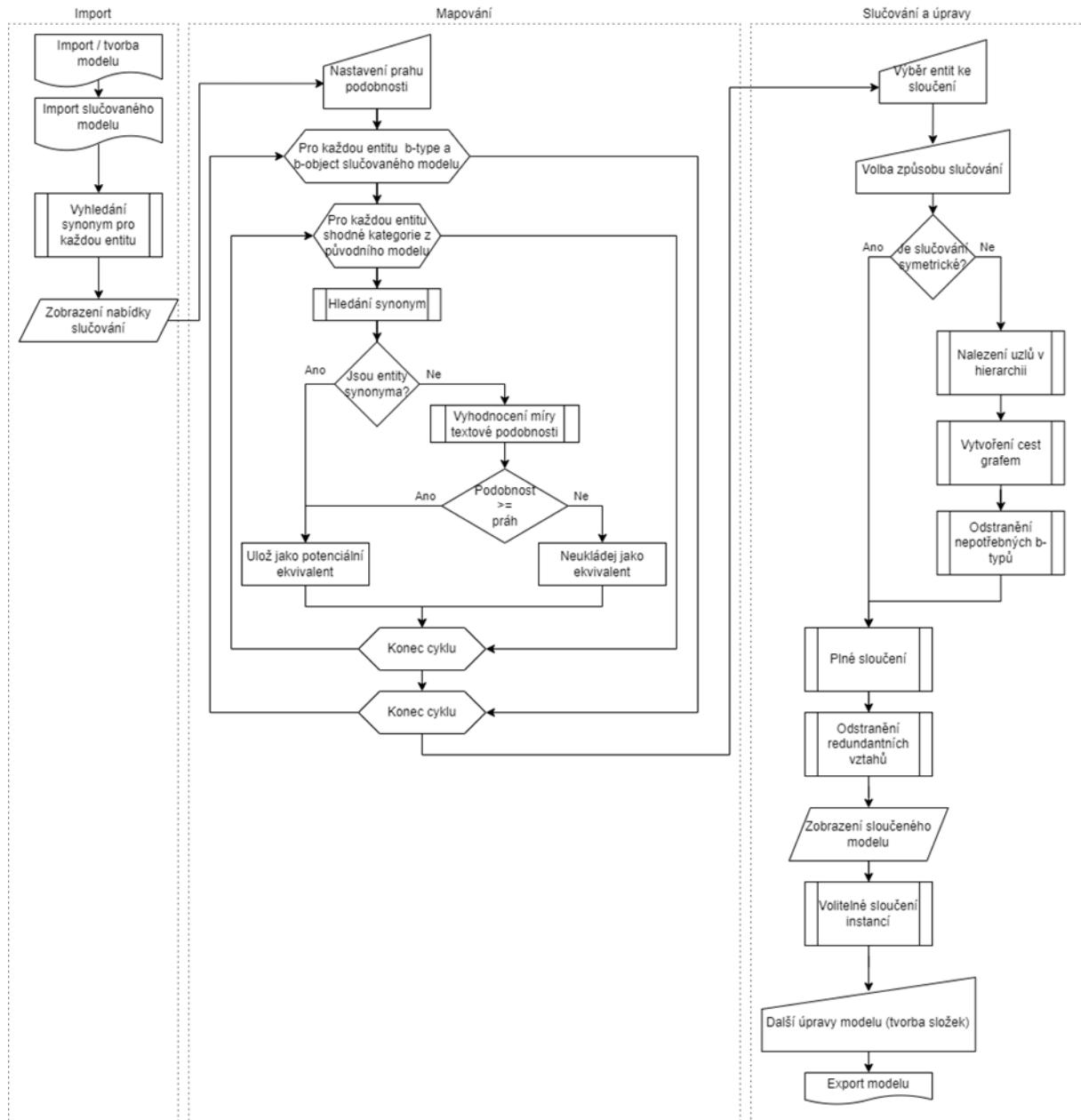
Na základě charakteristik knihoven si autor pro implementaci zvolil GoJS. Jedná se o placený produkt a je proto zřejmé, že podpora uživatele a kvalita funkcionalit bude na vysoké úrovni. Výhodou je však bezplatné použití pro akademické účely.

Knihovna mxGraph byla vyhodnocena jako dostatečná pro vizualizaci PURO modelu a pro zobrazování slučování dvou grafů. Nebyla však zvolena kvůli potenciálně složitější implementaci a méně přehledné dokumentaci. Kontextová menu a možnosti tvorby složek jsou oproti knihovně GoJS na mnohem nižší úrovni.

Vis.JS je užitečnou knihovnou pro dynamické zobrazování velkého množství uzlů. Přesun uzlů a interakce mezi nimi působí velmi moderně a knihovna se tak jeví jako vhodnou volbou pro interaktivní zobrazování již vytvořených modelů bez dalšího tvůrčího vstupu uživatele. Pro tvorbu nového modelu a strukturování do hierarchie složek v rozsáhlejším PURO modelu však není podle autorova hodnocení příliš vhodná.

7 Návrh algoritmů pro slučování PURO Modelů

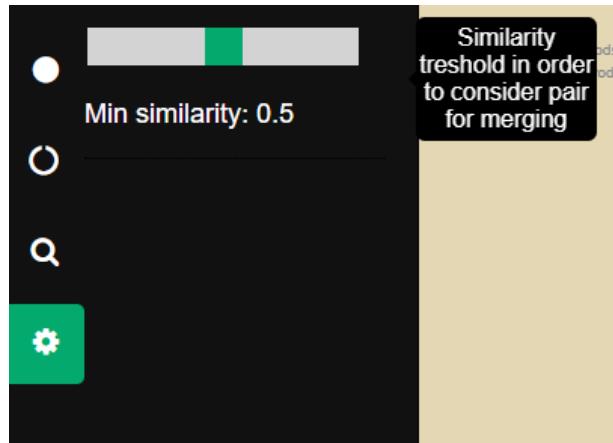
Na Obr. 44 je schéma popisující proces importu, mapování na základě textové podobnosti a synonymie a následné slučování⁷.



Obr. 44: Schéma importu, mapování a slučování OBM

⁷ Kód je dostupný na: <https://github.com/TheCandy/PUROMJoiner>

Schéma je rozděleno do tří částí. Ve fázi importu je nutné vytvořit, či importovat první model. Druhý model musí být již vytvořen a uložen. Po nahrání druhého modelu nastává fáze mapování. Uživatel nastavuje práh textové podobnosti dvou entit (viz Obr. 45).



Obr. 45: Výběr prahu míry textové podobnosti

Uživatel vybírá míru textové podobnosti od nuly do jedné. Číslo vyjadřuje Diceho míru podobnosti [24] vypočítanou algoritmem string-similarity, který je volně dostupný pod licencí ISC⁸. Pokud je podobnost větší nebo rovna prahu, budou entity vyhodnoceny jako potenciálně shodné.

Pokud je však mezi dvěma entitami nalezen vztah synonymie, je porovnávání textového řetězce ignorováno a entita je nabízena jako potenciálně shodná i přes nesplnění prahu minimální podobnosti textu.

Uživatel poté pro entity ve slučovaném modelu vybírá ze seznamu potenciálně shodných entit původního modelu takovou, která má být sloučena. Zároveň je možné entitu ponechat jako unikátní (viz Obr. 60 v kapitole 8).

Poté už je možné přejít k samotnému sloučení modelů, které probíhá automaticky (viz kapitola 7.2).

7.1 Prerekvizity slučování

Před samotným slučováním je nutné vytvořit, či importovat první model a následně naimportovat i druhý, tedy slučovaný model. Druhý model by měl být již vytvořený. Model může být importován z původního PURO Modeleru, či ze souboru uloženého z aplikace (vyvíjené v rámci této práce).

Tyto modely mohou být vytvořeny nezávisle a je tedy nutné předejít případné kolizi identifikátorů entit. Entity jsou totiž popisovány unikátním identifikátorem a ten zároveň slouží pro vyjádření propojení s jinou entitou (více v kapitole 7.1.1)

Při importu druhého modelu je zároveň nutné vyhledat shodné entity na základě textové podobnosti a případné synonymie. Tyto shodné entity jsou pak nabízeny uživateli. Ten může prohlédnout obě entity v grafickém prostředí a poté usoudit, zdali se opravdu mají sloučit.

⁸ Dostupné na : <https://www.npmjs.com/package/string-similarity>

7.1.1 Import

Analýza

Importování modelů z PURO Modeleru je prvním krokem ve slučování dvou grafů. Přímá komunikace s původní aplikací je nad rámec této práce, proto budou použity exportované soubory ve formátu JSON. Předpokládá se vstup alespoň dvou souborů, které obsahují data o uzlech a propojeních.

Import proběhne vybráním dvou souborů. Záleží na samotné implementaci, zdali bude jeden z modelů nabízen jako hlavní. Při plném slučování na pořadí nezáleží.

Návrh

Již při samotném importu dvou modelů je nutné pracovat s předpokladem, že uzly obou grafů mají jedinečný identifikátor (dále ID). ID slouží k jednoznačnému označení uzlu a zároveň je využito jako informace pro vztahy mezi uzly. Vztahy jsou v nejjednodušší podobně reprezentovány jako pole dvou čísel. První číslo je ID počátečního uzlu, druhé číslo označuje uzel cílový.

Lze předpokládat, že původní aplikace vytváří pro každý nový uzel ID, které není v kolizi s ostatními. Avšak při vývoji několika modelů odděleně bude s největší pravděpodobností mnoho uzlů napříč slučovanými modely sdílet stejně ID. Je sice možné, že původní aplikace pamatuje na kolize uzlů se stejnými ID a jednoduše by některému z problematických uzlů přiřadila ID nové. Problematické jsou ale vztahy mezi uzly. Ty jsou, jak již bylo zmíněno výše, definovány dvěma ID. Pro původní aplikaci je v podstatě nemožné určit, kterému z kolizních uzlů tento propoj původně příslušel.

Z důvodu kolize identifikátorů uzlů je nutné navrhnout pro druhý a každý další importovaný model algoritmus, který každému uzlu a jeho odkazu v relevantních propojeních vytvoří ID nové:

1. Projdi všechny uzly prvního grafu a k nejvyššímu nalezenému ID přičti jedničku. Toto bude nejnižší ID importovaného grafu (*novéID*).
2. Pro každý uzel importovaného grafu:
 - a. Uchovaj původní ID uzlu jako novou proměnnou *původníID*.
 - b. Nahraď jeho ID číslem *novéID*.
 - c. Přičti k *novéID* jedničku.
3. Pro každý propoj importovaného grafu:
 - a. Vyhledej v uzlech importovaného grafu *původníID*, které souhlasí s ID počátku propoje.
 - i. Nahraď ID počátku propoje proměnnou *novéID* nalezeného uzlu.
 - b. Vyhledej v uzlech importovaného grafu *původníID*, které souhlasí s ID konce propoje.
 - i. Nahraď ID konce propoje proměnnou *novéID* nalezeného uzlu.

Tímto je zaručena jedinečnost každého uzlu a jeho propojení.

7.1.2 Mapování

Analýza

Pro nabízení potenciálně shodných entit je nutné vyhodnocovat míru podobnosti textových řetězců názvů a zároveň zjistit, zdali jsou synonymy. Postup reflektuje diagram popsaný na Obr. 44.

Před samotným sloučením je tedy vhodné nabízet uživateli entity, které by mohly být potenciálně shodné. Zvolením těchto entit by uživatel rozhodoval o jejich sloučení.

Návrh

Při importu původního i slučovaného modelu je třeba projít každou entitu modelu a pokusit se její název vyhledat v seznamu slov s přiřazenými synonymy. Při vkládání druhého modelu ke slučování je pak třeba porovnávat seznam synonym každé entity ze slučovaného modelu se seznamem synonym entit z původního modelu. Pokud bude synonymie nalezena, není již nutné brát ohled na potenciální textovou podobnost. Je zřejmé, že například slovo „člověk“ a „osoba“ jsou synonyma, ale podobnost textového řetězce je velmi nízká. Synonymie se tedy jeví jako důležitější než porovnávání samotného textu a nabízení entit, jejichž názvy jsou synonyma je prakticky vždy žádoucí.

Při vkládání druhého modelu se zároveň na základě zvoleného prahu textové podobnosti porovnají všechny entity slučovaného a původního modelu.

Pokud je textová podobnost vyšší nebo rovna prahu, nebo pokud se jedná o synonymum, je entita z původního grafu zvolena jako potenciálně shodná s entitou slučovaného grafu. Potenciálně shodné entity jsou pak uživateli nabízeny jako seznam v kontextovém menu při kliknutí na entitu ze slučovaného grafu. Z tohoto seznamu může uživatel vybírat, se kterou entitou se právě nahlížená entita sloučí.

Pro vyhledávání synonym byla použita offline databáze WordNet⁹. Databáze je uložena ve formátu JSON a při spuštění aplikace je nahrána do objektu nazvaného Synonyma. Při importu modelu je procházena každá entita a v objektu Synonyma je vyhledáváno slovo na základě názvu entity. Příslušná synonyma jsou nahrána do pole synonym entity. Při změně názvu entity je tato synonymie opět aktualizována.

Při importu druhého modelu je spuštěn algoritmus, který porovnává každou entitu slučovaného modelu s každou entitou původního modelu. Předpokladem je, že entity jsou shodné kategorie. Například b-typ je porovnáván pouze s b-typem.

Nejprve je zjištěno, zdali pole synonym obou porovnávaných entit obsahuje alespoň nějaký shodný záznam. Dále je porovnána textová podobnost názvů entit.

Pokud je nalezeno synonymum, či pokud je podobnost vyšší nebo rovna nastavenému prahu, je porovnávaná entita z původního modelu uložena do seznamu potenciálně shodných entit pro entitu ze slučovaného modelu. Tento seznam potenciálně shodných entit je pak nabízen uživateli (viz Obr. 60 v kapitole 8).

⁹ Dostupné na: <https://wordnet.princeton.edu/>

7.2 Slučování

7.2.1 Analýza

Slučování z hlediska tříd bylo již popsáno v kapitole 2.2 jako proces zahrnující čtyři druhy přístupů:

- Plné sloučení

Při plném sloučení dochází k zachování všech tříd. Dojde tedy pouze k propojení vztahů a sloučení shodných uzlů. Tento způsob není destruktivní a zachovává všechny údaje z obou původních modelů. Vede však k nejvyššímu počtu redundancí.

- Asymetrické slučování

Asymetrickým slučováním se myslí takové, jehož výstupem je model, který respektuje hierarchii pouze v cílovém, nebo výchozím modelu. Motivací může být například vhodnější hierarchie tříd v jednom z modelů. Vývojář například nemusí mít zájem o větší granularitu a s tím rostoucí nepřehlednost. Obecný popis této problematiky se věnoval různým druhům automobilů. Ve výsledném modelu například není třeba informace o třídách jako je „německý automobil“, či „italský automobil“. Vývojář dost možná pro jeho konkrétní využití naprosto postačuje fakt, že každá značka automobilu je podtřídou „automobil“. Tento druh slučování má dva typy, které jsou však ve své podstatě opakem a záleží pouze na tom, který model je cílový:

- Slučování podle cíle: Zachovává třídy, které jsou nalezeny v cílovém modelu
- Podle zdroje: Zachovává třídy nalezené v původním modelu.

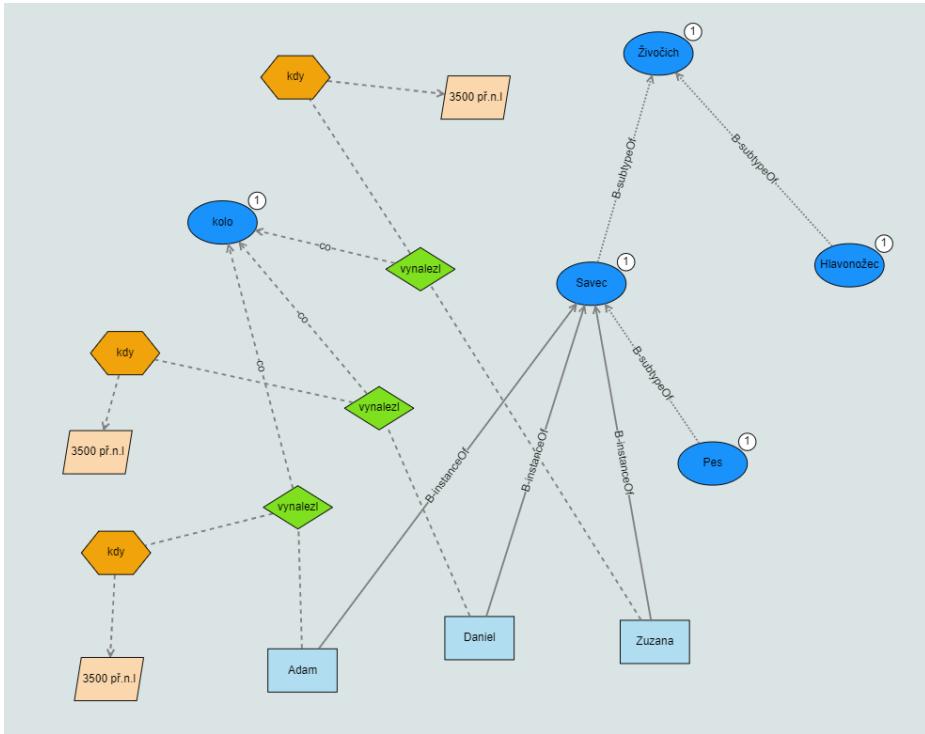
- Expertní slučování

Tento druh slučování nemůže být z podstaty automatizovaný jednoduchým algoritmem. Vývojář dotvoří potřebné třídy na základně svých znalostí domény. Při větším modelu to může klást velké nároky na schopnosti a čas vývojáře.

Cílem práce je navrhnut a implementovat metody, které by vedly k úspěšnému slučování. Vhodné je pak nabídnou uživateli plné i asymetrické sloučení. Vzhledem k zaměnitelnosti slučování podle cíle a zdroje bude navržen postup pouze pro jeden z nich.

7.2.2 Návrh

V kapitole 2.2 venující se slučování je popsán případ, kdy dochází k mazání uzlů v hierarchii modelující vztah muže a ženy k obecnému typu živočichů. Konečným výsledkem tohoto slučování byla situace, ve které tři instance Adam, Daniel a Zuzana zdědily vztah popisující vynález kola (viz Obr. 46).

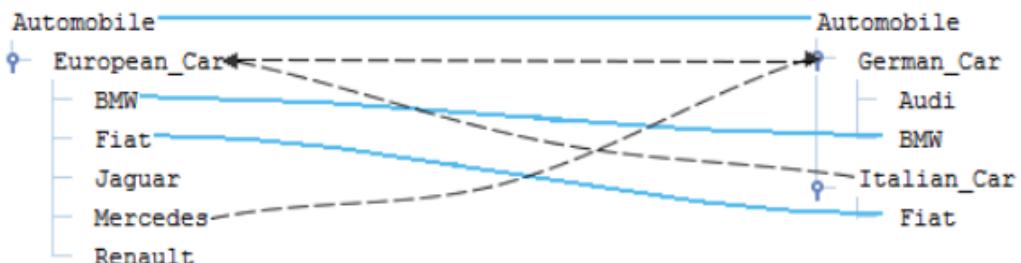


Obr. 46: Přenos relací na instance. (kopie Obr. 8)

V kapitole 2.2 bylo také zmíněno riziko tohoto postupu. Při převodu OBM z PURO Modeleru do jazyku OWL dochází totiž k naprostému zahodení instancí typů. Z tohoto důvodu se pak všechny atributy a relace přenášejí na typ, ze které instance vychází. V PURO by to pak znamenalo, že savec vynalezl kolo, což je do určité míry v porádku. Přenosem tohoto vztahu na typ „pes“ však vzniká nežádoucí situace.

Instance již může mít vytvořené vztahy s okolím. Lze očekávat, že jsou tyto vztahy logicky správně, jelikož jsou manuálně vytvořeny vývojářem. Pokud však dojde ke smazání typu, na nějž se instance odkazuje, došlo by při převodu do OWL k přenosu těchto vztahů na jakýkoli nadřazený typ, jejíž instancí se nyní uzel stal. Lze očekávat, že každá instance typu bude mít nějaké vztahy s okolím, jinak by totiž nebylo smysluplné ji vytvářet, vzhledem k tomu, že při převodu do OWL stejně dojde k jejímu smazání a slouží tak pouze k popisu modelované situace.

Dalším důležitým faktem je, že nejnižší podtypy často popisují fakta nezbytná k následnému využití modelu. Příkladem může být obrázek, který v kapitole 2.2 sloužil k popisu situace slučování ontologie s automobily (Obr. 47).



Obr. 47: Příklad mapování dvou ontologií [10]. Kopie Obr. 2

Pokud by se na obrázku popisované modely slučovaly podle cíle, nebylo by pak potřeba pracovat s uzlem „evropské auto“, nalezené uzly Fiat a BWM by se převedly do příslušných podtypů, jelikož v cílovém modelu existují. Zbytek uzelů by se převedl přímo pod typ „automobil“. Z příkladu je však zřejmé, že zachování uzelů popisujících značky automobilů je žádoucí, jelikož vyjadřují potřebnou cílovou granularitu, či vyjadřovací schopnost modelu.

Autor proto na základě těchto myšlenek navrhuje určitá omezení v možnosti odstraňování uzelů typů při asymetrickém slučování:

- Nelze smazat typ který má přímo napojené instance a nemá nadřazený typ. Toto by vedlo ke ztrátě informace při převodu do OWL, jelikož instance se nepřenáší.
- Při smazání typu s přímo napojenými instancemi by se jakýkoli vztah instance s okolím přenášel na nadtypy odstraněného uzlu. Toto může vést k mnoha nelogickým situacím. Není tedy vůbec možné smazat typy s instancemi.
- Není možné smazat typ který by vztahy přenášel přímo na svoji instanci. Pokud má typ přímé uzly instance a zároveň má sám atribut, či relaci, není možné ho smazat.
- Nejnižší typy nemusí být vhodné smazat, jelikož popisují detailní a pravděpodobně žádoucí informace.

Rozborem těchto omezení lze vyvodit, že nejnižší typy v hierarchii většinou není vhodné smazat. Za určitých situací to vůbec není možné (instance by neměla typ), nebo by byly při převodu do OWL vztahy instance nelogicky přeneseny na nadřazený typ. Případně odstranění nemusí být žádoucí, jelikož by model přišel o vyjadřovací schopnost a násobení vztahů na podtypy by vytvářelo nepřehledný model. Ořezáním výše uvedených podmínek ve své podstatě vyplývá, že smazat nejnižší typ v hierarchii je možné pouze za předpokladu, že nemá instance.

I v situaci, kdy by nejnižší uzel bylo možné smazat je však nutné se zamyslet nad vhodností tohoto řešení. Jak již bylo uvedeno, nejnižší uzly poskytují určitou granularitu a vyjadřovací schopnost modelu. Bylo by tedy vhodné nechat na uživateli poslední rozhodnutí, zdali chce nejnižší typy smazat či ne.

Podobnou situaci lze pak sledovat i u mazání typů, které mají spoustu podtypů. Ty by totiž dědily vlastnosti svého odstraňovaného nadtypu, což by mohlo vést k nepřehlednému modelu. Uživatel by měl mít opět možnost rozhodnout například o tom, kolik musí mít typ podtypů, aby nedošlo k jeho smazání.

Následující návrh řešení popisuje slučování podle zdroje. Algoritmy je však možné využít i pro slučování podle cíle pouhým vyměněním pořadí modelů.

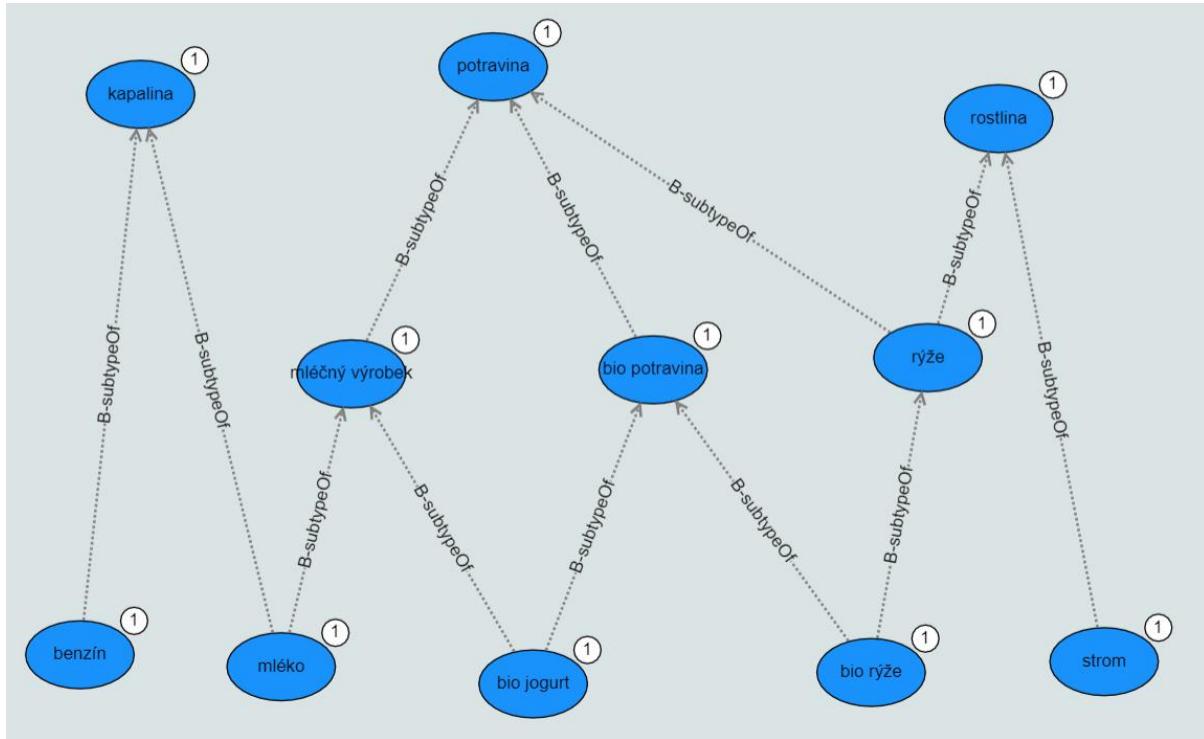
Nalezení uzelů v hierarchii typů

Prvním krokem je průchod všemi uzly typů v modelu, které jsou zahrnuty do hierarchie podtypů. Je tedy nutné pro každý uzel termu „b-type“ prohledat připojené hrany grafu bez ohledu na směr a zjistit, zdali jsou kategorie „b-subtypeOf“. Takový uzel je pak s jistotou zahrnut do hierarchie. Důvodem je ignorování typů, které jsou instance, jejichž tvorba je v jazyce PURO možná.

Dále je nutné vyvodit které uzly jsou typem na vrcholu grafu. Z podstaty konceptu podtypů vyplývá určitá dědičnost a hierarchie, lze tedy vyloučit cykly. Díky tomu je pak možné usuzovat, že v určitých podmnožinách bude mít tento graf typy pro určité množství uzelů jediný vrchol, nejedná se však

o orientovaný graf, protože listy grafu mohou mít k vrcholu více cest. Vyplývá to z volnosti při určování podtypů. Jakýkoli uzel totiž může být podtypem více uzlů, pokud to nepovede k cyklickému grafu.

Výsledným grafem může být například následující hierarchie (Obr. 48):



Obr. 48: Hierarchie b-typů

Zde je možné určit několik vrcholů. Jsou jimi „kapalina“, „potravina“ a „plodina“.

Potravina se několika cestami může dostat k nejnižším uzlům „mléko“, „bio jogurt“ a „bio rýže“, avšak ne k uzlu „benzín“ a „strom“, což je fakticky správné. Je zde zároveň patrná výše popsaná situace, kdy uzel náleží k více nadtypům. Například „mléko“ je zároveň mléčný výrobek i kapalina. Proto je možné se z něj dostat ke dvěma vrcholovým uzlům.

Pro všechny vrcholové uzly typů ale platí, že do nich nevstupuje žádná hrana grafu typu „b-subtypeOf“. Toto je tedy nutnou podmírkou pro určení nejvyšších uzlů.

Vytvoření cest grafem

Následným krokem po identifikaci vrcholových uzlů je pak průchod grafem pro nalezení pomyslných „podstromů“ vycházejících z každého vrcholového uzlu. Tento průchod bude v podstatě reflektovat průchod stromu do šírky. Pro každý uzel je pak nutné zjistit, na který uzel navazuje. Vrcholové uzly nenavazují na žádný, postupně však ke každému uzlu bude přibývat seznam uzlů, které jsou před ním. Toto musí být vytvářeno s ohledem na to, který vrcholový uzel se právě prochází. Výsledkem pak bude, že každý uzel bude mít více seznamů uzlů, přes který se dostanou ke každému vrcholu, se kterým souvisejí.

Uzel „mléko“ pak například bude mít dva seznamy:

- potravina -> mléčný výrobek -> mléko
- kapalina -> mléko

Bio rýže pak:

- potravina -> bio potravina -> bio rýže
- rostlina -> rýže -> bio rýže

Zároveň je vhodné každému uzlu přiřadit, které uzly jsou jeho přímým následníkem a předchůdcem, aby již dále nebylo nutné tuto skutečnost vyvozovat ze samotných hran grafu.

Například pro uzel bio potravina to pak bude následující situace:

- nadtyp: potravina
- podtyp: bio jogurt, bio rýže

V tomto případě je graf poměrně malý a je na první pohled jasné, které uzly jsou nadřazené a podřazené, je však nasnadě, že rozsáhlejší modely mohou být málo přehledné a tato uložená informace zefektivní další kroky v procesu slučování.

Shodnost entit

Dalším krokem je vyhodnocení mapování shodných uzlů mezi modely. Toto platí pro každý uzel, který je zahrnut do hierarchie. Pokud tedy uzel má nadtyp, či podtyp, je zahrnut do porovnávání. Pokud je nalezen uživatelem vybraný shodný uzel na základě textového řetězce, sémantiky a typu, je uzlu z cílového grafu tato informace uložena.

Hledání kandidátů na odstranění

Při asymetrickém slučování nelze jednoduše vymazat všechny typy, kterým nebyl nalezen ekvivalentní typ v původním grafu. U každého uzlu je nutné vyhodnotit následující:

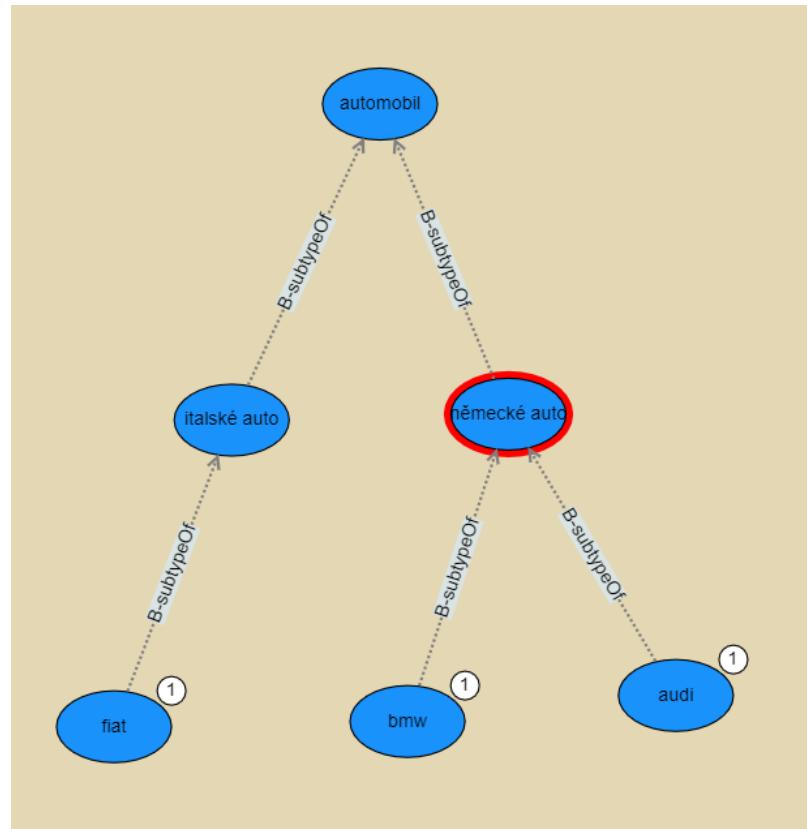
- Pokud má typ instance, není možné ho smazat. Vztahy instancí by se při převodu do OWL přímo přenesly na typ, který by zůstal. Toto by mohlo vést k neologickým situacím.
- Pokud je typ nalezen i v původním grafu, není možné ho smazat
- Pokud je typ instancí jiného typu, není vhodné ho mazat. Vztahy by byly defacto ztraceny, jelikož by nemohly být přeneseny na typ nad sebou.
- Uživateli by měla být dána možnost odstranit nejnižší typy, které samy nemají instance a nebyly nalezeny v původním grafu. Tyto typy ale mohou být napojeny na důležité relace, či atributy. Proto by mělo být rozhodnutí přenecháno na uživateli.

Pokud typ nesplní žádnou z výše popsaných podmínek, je určena jako kandidát na smazání.

- Uživateli je vhodné nabídnout možnost zvolit číselnou hodnotu, která by reprezentovala určitý práh. Pro každý typ může existovat určitý počet podtypů, které budou přímo dědit, pokud bude typ smazán. Pokud uživatel nastaví práh, nesmaže se typ, u kterého by počet přímo dědících podtypů překonal tento práh.

Průchod cestami nahoru

Při průchodu cestami nahoru je nutné projít každou cestu každého nejnižšího uzlu. V tomto průchodu je nutné vyhodnotit, kolik typů je přímo závislých na smazání konkrétního uzlu. Každému uzlu je pak přiřazen seznam uzlů, které budou přímo dědit při jeho smazání.



Obr. 49: Příklad potenciálního smazání b-typu v hierarchii

Pokud by na Obr. 49 došlo ke smazání typu „německé auto“, je nutné typu „automobil“ uložit uzel „italské auto“, „bmw“ a „audi“ jako uzly, které budou dědit jeho relace a atributy za předpokladu, že dojde i ke smazání uzlu „automobil“.

Proto je každá cesta procházena odspodu a dalším uzlům se přidávají do pole „závislých uzel“ vždy všechny závislé uzly přímých podtypů, které budou smazány.

Poté se opět projdou všechny uzly typů a odstraní se ty, které byly určeny ke smazání a počet „závislých uzel“ nepřekročí práh zvolený uživatelem.

Odstraňování typů

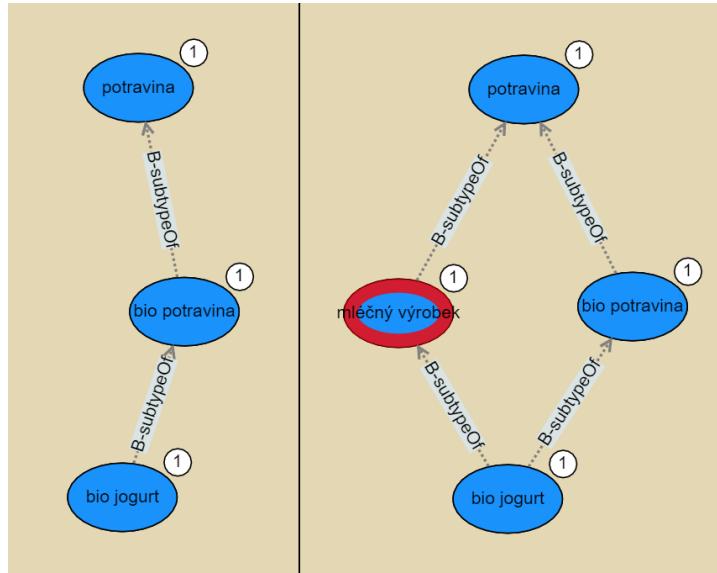
Pro každý uzel typu, který byl určen ke smazání je nutné provést následující:

1. Přenést propojení s nadtypem na každý uzel podtypu.
2. Distribuovat atributy a relace mezi všechny podtypy.

V prvním případě je nutné projít každý uzel podtypu a přepojit ho na každý nadtyp, který odstraňovaný uzel má.

Dále je nutné vzít přímé okolí odstraňovaného uzlu, tedy všechny atributy a relace s okolím. Tyto relace je poté třeba rozkopírovat pro každý podtyp a přepojit na ně hrany grafu, které původně příslušely odstraňovanému uzlu.

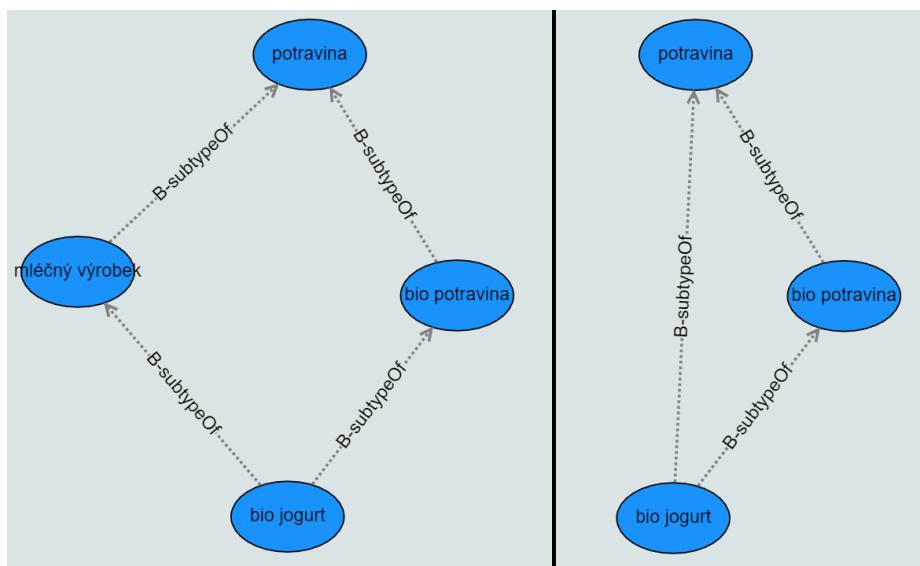
Uzel pak již může být odstraněn. Po odstranění všech uzel je pak nutné znovu vyhodnotit cesty pro uzel a jejich hierarchii. Toto je nutné zejména kvůli redundanci vztahů, která může vzniknout odstraněním typů a s tím souvisejícím přenosem vztahů s nadtypy na podtypy.



Obr. 50: Situace přes sloučením podle zdroje

Obr. 50 popisuje situaci před sloučením cílového modelu vpravo se zdrojovým modelem vlevo. Červeně je označen uzel, který nebyl ve zdrojovém grafu nalezen a bude proto odstraněn. Při odstranění uzel je vždy přenesen vztah s nadtypem na uzel pod nimi.

Na Obr. 51 je proces odstranění uzlu „mléčný výrobek“.



Obr. 51: Proces odstranění uzlu v grafu

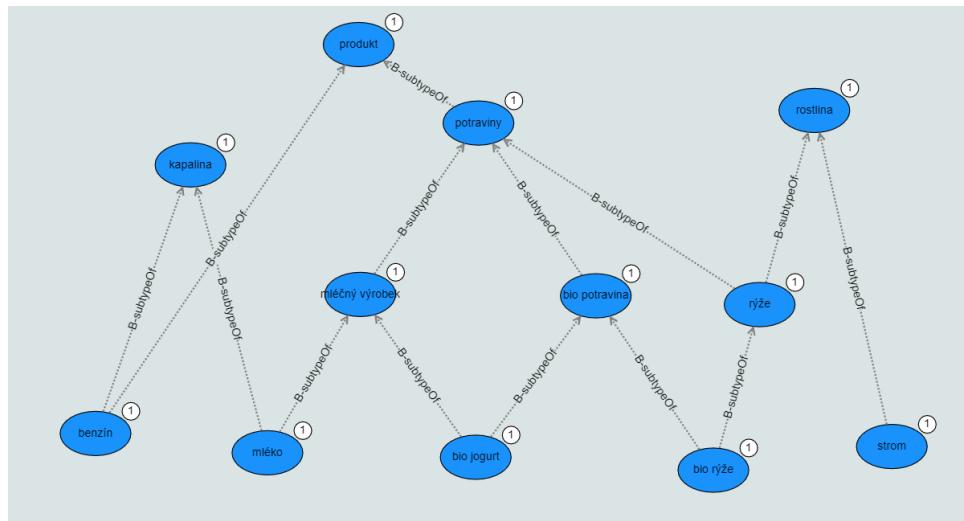
Zde již není popisován zdrojový a cílový model, ale popis situace před a po smazání uzlu „mléčný výrobek“

Uzel „mléčný výrobek“ byl odstraněn a jeho příslušnost k nadtypu „potravina“ byla přenesena přímo na uzel „bio jogurt“. Je zjevné, že bio jogurt je zároveň bio potravinou i potravinou. Přímý vztah s potravinou však není nutný, protože vyplývá z příslušnosti k typu bio potravina.

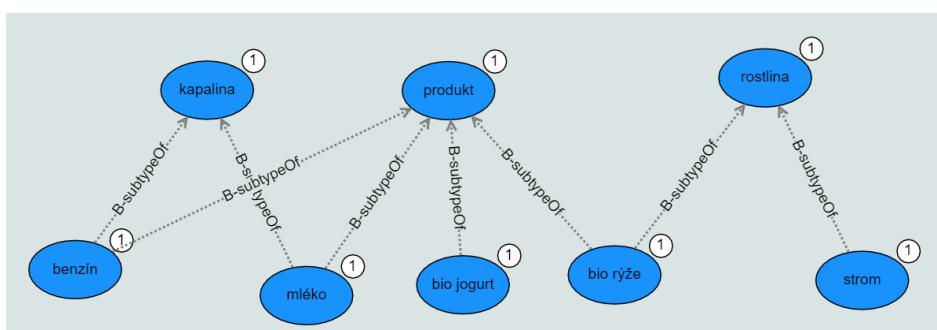
Je tedy žádoucí tyto redundantní vztahy odstraňovat pro přehlednost modelu. Z konečné situace popsанé na Obr. 51 vyplývá, že je vhodné odstraňovat ty vztahy s nadtypem, které vedou nejkratší cestou.

Odstraňování redundantních vztahů

Pro názornou ilustraci odstraňování redundantních vztahů v hierarchii byly vytvořeny dva modely. Na Obr. 52 je vyobrazen složitější model, popisující produkty. Model na Obr. 53 je zjednodušením předchozího modelu.



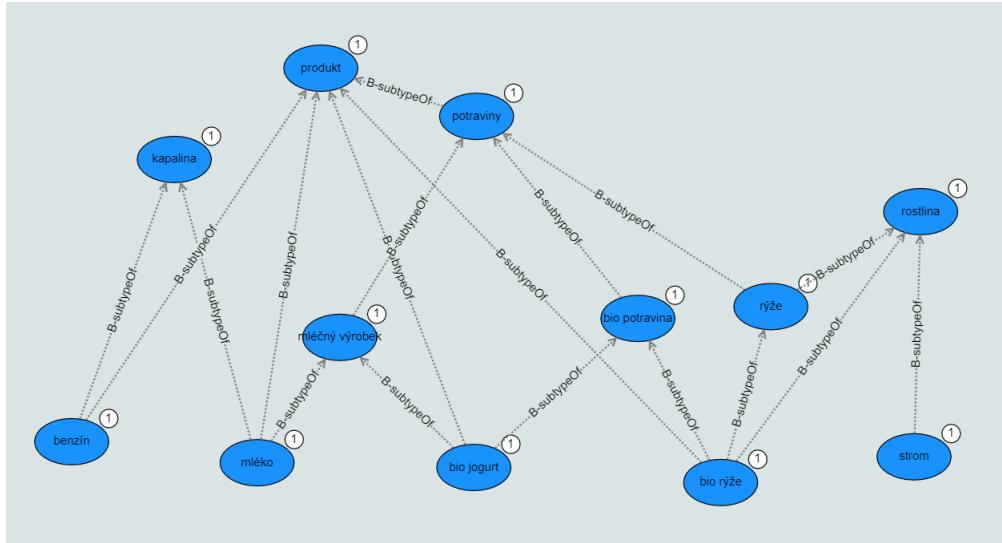
Obr. 52: Složitější model produktů



Obr. 53: Jednodušší model produktů

První model, tedy zdrojový, obsahuje stejné uzly, jako cílový model. Je však zřejmé, že některé typy ve druhém modelu spolu souvisejí přímo, ačkoli v původním modelu mají mezírok. Nedojde tedy k odstranění žádných typů, ale některé hrany grafu budou duplikovány.

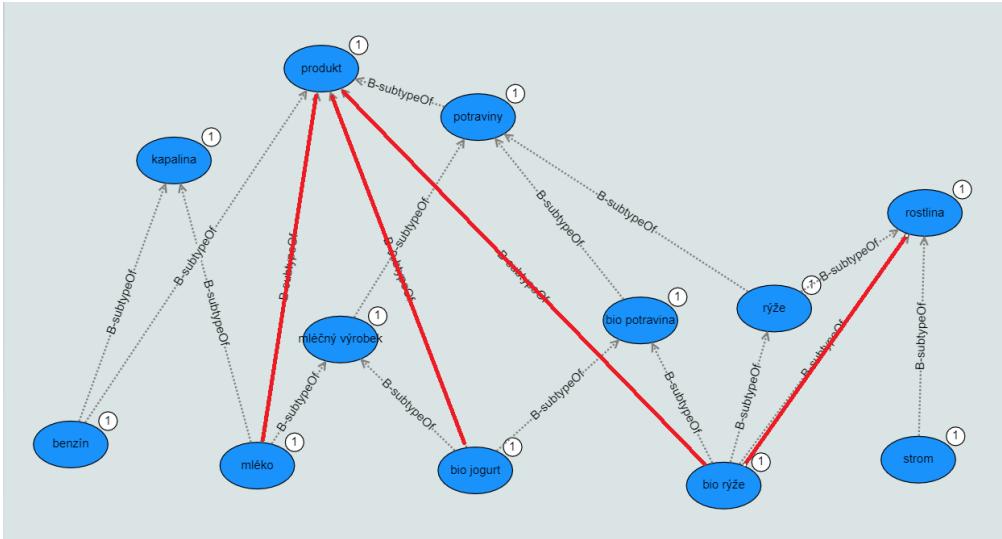
Výsledek sloučení těchto dvou modelů je následující:



Obr. 54: Výsledné sloučení modelů produktů

Přes nepřehlednost hran grafu na Obr. 54 je možné pozorovat, že některé cesty jsou přebytečné. Například fakt, že bio rýže je rostlinou vyplývá z toho, že bio rýže je podtypem rýže a ta je podtypem rostliny. Bio jogurt je produktem už z příslušnosti k bio potravině, která je potravinou a tedy produktem.

Na Obr. 55 jsou vyznačené hrany grafu, které je možné odstranit.



Obr. 55: Redundantní hrany grafu

Člověk může poměrně jednoduše vyvodit, které hrany je nutné odstranit na základě znalosti domény. Algoritmus však musí postupovat bez těchto znalostí. Je tedy nutné brát v potaz každou cestu, kterou se každý z uzlů dostane ke každému svému nejvyššímu typu. Mohla by nastat situace, kdy uzel cestu vede přes průchozí typy více způsoby, na tyto situace je také nutno pamatovat.

Pro každý uzel se tedy opět prochází seznam cest k nejvyššímu typu vytvořený v rámci prohledávání hierarchií typů. Například pro uzel rýže budou cesty následující:

- produkt -> potraviny -> bio potravina -> bio rýže
- produkt -> potraviny -> rýže -> bio rýže
- produkt -> bio rýže
- rostlina -> bio rýže
- rostlina -> rýže -> bio rýže

Některé cesty v tomto seznamu je nutné odstranit. V tomto příkladu se vždy bude jednat o cestu s jednou hranou grafu. Avšak není to nezbytnou podmínkou. Cesta může být delší, a přesto může být redundantní. Zároveň je nutné poznamenat, že ačkoli do uzlu bio rýže vede cesta z typu „produkt“ na první pohled duplikátně přes typy „bio potravina“ a „rýže“, nelze jednu z nich smazat. Každá cesta totiž vyjadřuje unikátní vztah s nadřazenými typy. Tyto typy by totiž mohly mít relace či atributy, které by pak díky dědičnosti přenášely na typy pod sebou. Ačkoli tedy tyto cesty vedou ke stejně nejvyššímu typu, vedou k němu jiným způsobem a vytváří tak specifické vztahy pro nižší typy.

Je tedy při odstraňování nutné zvážit každou specifickou cestu a odstraňovat vždy tu nejkratší, která vyjadřuje to samé, jako nějaká z delších cest.

Konečné sloučení

Po odstranění nežádoucích typů a přenos jejich vztahů a hierarchie na podtypy je výsledkem graf, který již má každý typ buďto shodný s cílovým grafem, nebo již není žádoucí žádný z uzlů typů odebírat. Poté již lze oba grafy slučovat stejným postupem, jako je tomu při plném slučování.

Při tomto slučování jsou procházeny uzly kategorie b-typ a b-objekt v cílovém grafu. Pro každý z uzlů je nutné zjistit, jestli má být sloučen se svým ekvivalentem, či jestli má být v konečném grafu unikátní.

Pokud má být uzel unikátní, je třeba přenést i jeho okolí popsané v kapitole 4.4. Pro každý uzel z okolí je nutné zjistit, jestli už jejich přidání není určeno jiným uzlem b-typu, či b-objektu. Tímto se předejde duplikátům uzlů a linků.

Pokud má být uzel sloučen, je nutné porovnat jeho okolí s okolím ekvivalentního uzlu z původního grafu. Pro každý takovýto uzel z okolí je třeba vyhodnotit, zdali se již v grafu nachází. Pokud ano, nesmí být znova přidán, jelikož by tak mohlo dojít k duplikaci atributů a relací.

Postup popisu algoritmů bude reflektovat podkapitoly výše. Prvním krokem je tedy algoritmus pro průchod grafu a vytvoření hierarchie:

1. Pro každý uzel U zjisti, jestli z něj, či do něj vede link typu „b-subtypeOf“. Pokud link b-subtypeOf vede pouze do uzlu U, znamená to, že patří k nejvyššímu typu. Pokud patří, pokračuj:
 - a. Vytvoř prázdné pole „strom“, které bude obsahovat uzly patřící v hierarchii pod tento uzel U. Přidej do něj uzel U.
 - b. Vytvoř pole „uzlyÚrovne“ obsahující uzly, které se nacházejí v určité úrovni, která se právě prochází. Přidej do něj uzel U.
 - c. Vytvoř proměnnou „úroveň“ vyjadřující číselně od nejvyššího uzlu U (0) postupně číslo každé úrovně. Přiřaď jí číslo -1
 - d. Pokud se délka pole „uzlyÚrovne“ nerovná nule, prováděj následující:
 - i. Vytvoř pole „budoucíÚroveň“.
 - ii. Přičti k „úroveň“ jedničku.
 - iii. Pro každý uzel „výchozí“ v poli „uzlyÚrovne“ proved’ následující:
 1. Pro každý uzel „související“, který je s uzlem „výchozí“ propojen linkem kategorie „b-subtypeOf“ a kde je uzel „související“ brán jako cílový uzel („související“ je tedy nadtypem uzlu „výchozí“) proved’ následující pouze pokud je uzel „související“ nalezen v poli „strom“:
 - a. Uzlu „výchozí“ přidej do jeho pole nadřazených uzlů uzel „související“
 - b. Vezmi pole „cesta“ uzlu „související“, pokud nějakou má, a přiřaď toto pole do téhož pole uzlu „výchozí“. Přidej k tomuto poli i samotný uzel „související“.
 2. Pro každý uzel „související“, který je s uzlem „výchozí“ propojen linkem kategorie „b-subtypeOf“ a kde je uzel „související“ brán jako startovní uzel („související“ je tedy podtypem uzlu „výchozí“) proved’ následující:
 - a. Vlož „související“ do pole „strom“
 - b. Vlož „související“ do pole „budoucíÚroveň“
 - c. Vlož uzlu „výchozí“ do jeho vlastního pole „poduzly“ uzel „související“ a zároveň informaci o posledním nadřazeném uzlu z pole nadřazených uzlů.
 - iv. Nahraď pole „uzlyÚrovne“ polem „budoucíÚroveň“

Tento algoritmus v každém uzlu nejvyššího typu vytvoří pomyslný strom. V tomto stromě pak prohledává do šírky a v každé úrovni všem uzlům uloží informaci o nadřazeném a podřazeném uzlu. Zároveň všem uzlům ukládá informaci o všech cestách, které vedou k uzlu nejvyššího typu.

Následujícím krokem je určení uzlů typů k odstranění:

1. Pro každý uzel U z hierarchie typů proved’ následující:
 - a. Pokud je uzel U nalezen v původním grafu, ulož do jeho proměnné „kandidát“ hodnotu „false“
 - b. Pokud je uzel U instancí, nebo má sám instance, ulož do jeho proměnné „kandidát“ hodnotu „false“

- c. Pokud uzel U nemá podtypy a ani instance, kontroluj, zdali uživatel chce i nejnižší typy bez instancí zachovat:
 - i. Pokud ano ulož do jeho proměnné „kandidát“ hodnotu „false“.
 - ii. Jinak ulož „true“

Následuje algoritmus vyhodnocující počet závislých podtypů:

1. Pro každou cestu každého nejnižšího uzlu U procházej od nejnižšího typu po nejvyšší:
 - a. Pokud má uzel U po cestě „uzelCesty“ hodnotu kandidát = true, proved' následující:
 - i. Vytvoř pole „závislé uzly“. Toto pole bude udržovat pouze unikátní hodnoty. Pokud do něj byl uzel již přidán, nebude přidán znovu. Toto zabraňuje duplikátům, jelikož některé uzly můžou být procházeny víckrát.
 - ii. Pro každý podtyp uzlu uzelCesty zkонтroluj hodnotu kandidát. Pokud je true, přidej všechny uzly z pole závislé uzly náležící podtypu do pole závislé uzly náležící uzlu uzelCesty

Poté je každý uzel znova vyhodnocen. Pokud má hodnotu kandidát = true a počet uzlů v poli závislé uzly nepřekračuje uživatelem zvolenou mez, dojde k odstranění uzlu.

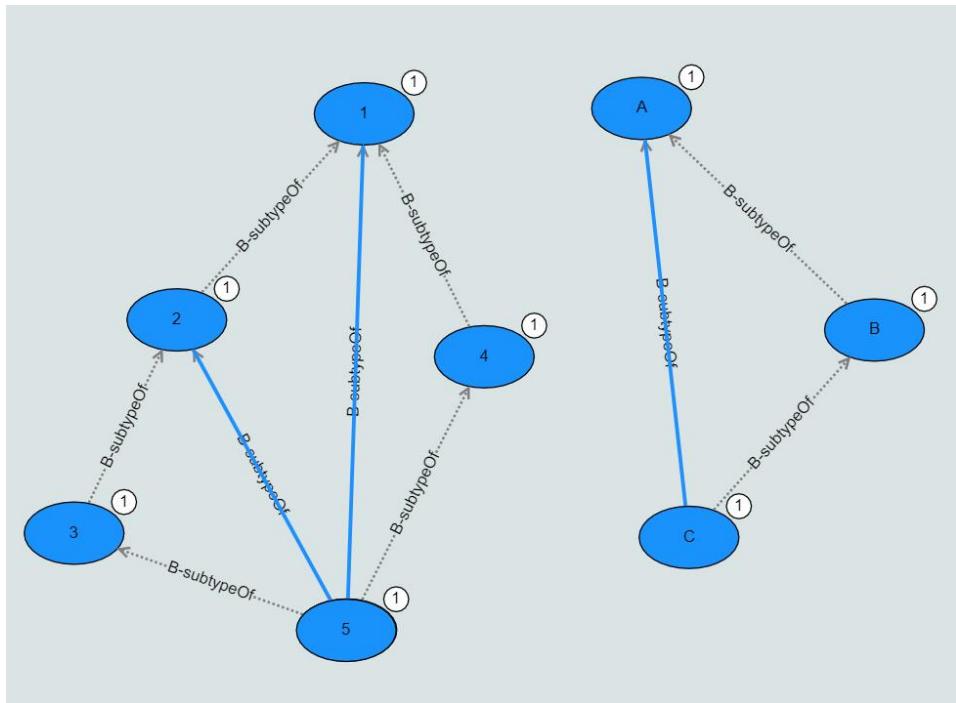
Pro odstranění libovolného uzlu typu je nutné následující:

1. Vezmi každý záznam z pole „poduzly“. Každý ze záznamů obsahuje uzel nižšího a uzel vyššího typu, které mají být propojeny po smazání vybraného uzlu.
 - a. Vytvoř link kategorie „b-subtypeOf“, který propojuje uzel nižšího a vyššího typu.

Dále je nutné kopírovat a propojit „okolí“, tedy relace a atributy, pro každý uzel nižšího typu:

1. Vytvoř pole „okolí“, které bude obsahovat uzly z okolí a vlož do něj samotný uzel odstraňovanýUzel.
2. Dokud je ještě co přidávat, tedy pokud od posledního cyklu proběhla změna v poli okolí, prováděj následující kroky:
 - a. Pro každý uzel v poli okolí vyhledej připojené uzly.
 - b. Pokud tyto připojené uzly náleží k jiné kategorii než typ, či objekt, a pokud ještě samy nejsou zahrnuté v poli okolí, přidej je do tohoto pole.
3. Odstraň z okolí odstraňovanýUzel
4. Pro každý podtyp potomek odstraňovaného uzlu odstraňovanýUzel proved' následující:
 - a. Zkopíruj pole okolí jako dočasnéOkolí bez referencí na původní uzly v poli.
 - b. Pro každý uzel dočasnéOkolí získej všechny linky, které jsou na něj napojeny.
 - c. Uzly i linky přidej do objektu dočasnýGraf.
 - d. Přeindexuj dočasnýGraf podle algoritmu z kapitoly 7.1.1. Důvodem je, aby po přidání tohoto dočasného grafu do původního grafu nedošlo ke kolizím identifikátorů.
 - e. Pro každý link z dočasného grafu proved' následující:
 - i. Pokud se id příchozího nebo odchozího uzlu rovná id uzlu odstraňovanýUzel, nahrad' ho id uzlu potomek.
 - f. Přidej dočasnýGraf do původního grafu.
5. Odstraň uzel odstraňovanýUzel

Posledním krokem je odstranění jakéhokoli redundantního propojení mezi libovolnými dvěma uzly.



Obr. 56: Ilustrace redundantních cest grafem

Pro ilustraci slouží Obr. 56. Dále je pro názornost uveden seznam cest dvou nejnižších uzlů.

Pro uzel C

[A]

[A, B]

Pro uzel 5

[1, 4]

[1]

[1, 2]

[1, 2, 3]

Je nutné vyhodnotit, které cesty jsou zbytečné. Je zjevné, že se jedná o cesty [A], [1] a [1, 2]. Cesta [1,4] sice vede ke stejněmu nejvyššímu uzlu jako ostatní cesty, ale uzel 5 může od uzel 4 dělit důležité vlastnosti.

Algoritmus vyhledávání redundantních cest probíhá následovně:

1. Vytvoř pole všech cest „všechnyCesty“.
2. Dokud není pole všechnyCesty prázdné, prováděj následující:
 - a. Najdi nejdelší cestu „nejCesta“ a odstraň ji z pole všech cest.
 - b. Vytvoř pole „keSmazání“ a nahraj do něj pole všechnyCesty.
 - c. Proveď cyklus s iterátorem „i“ od 0 do délky nejCesta:
 - i. Pro každou cestu v poli keSmazání proved’ následující:
 1. Pokud je délka cesty o jednu kratší, než i, odstraň ji z modelu, i ze seznamu keSmazání a všechnyCesty.

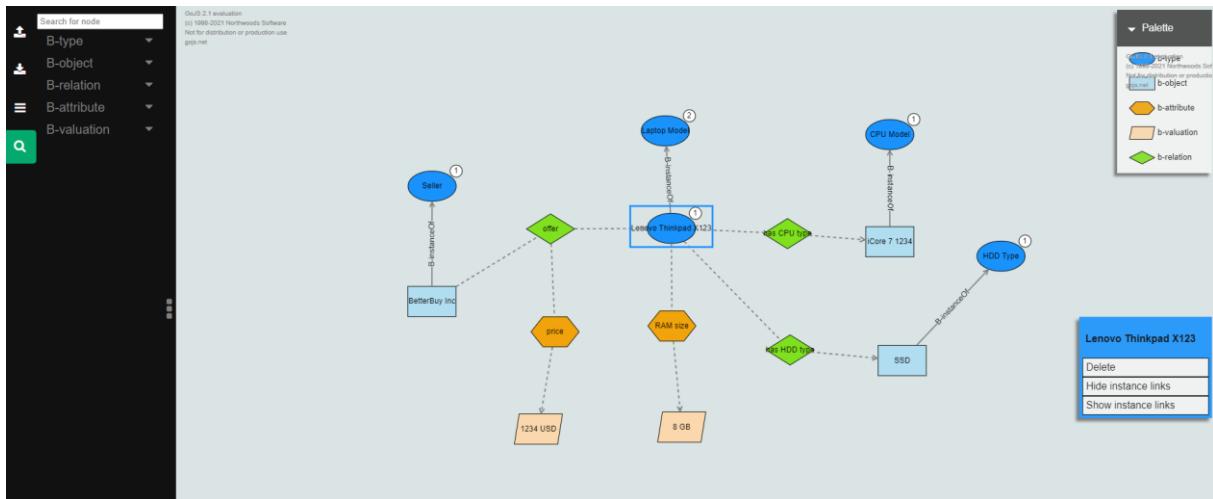
2. Pokud délka není o jednu kratší a uzel na pozici i v cestě se nerovná uzlu na pozici i v cestě nejCesta, odstraň cestu z pole keSmazání.

Pro výše uvedený příklad to znamená následující:

1. Nejdříve se vybere cesta [1,2,3].
2. Cesty [A] a [A,B] se již v prvním uzlu A nerovnají uzlu 1 a jsou z porovnávání vyřazeny, ostatní cesty mají jako první uzel 1 a zůstávají.
3. V druhém cyklu se porovnává uzel 2 a zároveň se kontroluje, jestli nějaká z cest není kratší než 2 (jelikož se jedná o druhý cyklus). Cesta pouze s uzlem 1 je kratší a je smazána. Cesta [1,4] má na druhém místě 4 a dále se neporovnává. Ostatní cesty pokračují.
4. Ve třetím cyklu se porovnává cesta [1,2]. Cesta je kratší než 3 a bude smazána.

Cyklus dále probíhá pro všechny ostatní cesty, které nebyly smazány. Takto se například odstraní cesta [A].

8 Implementace uživatelského rozhraní

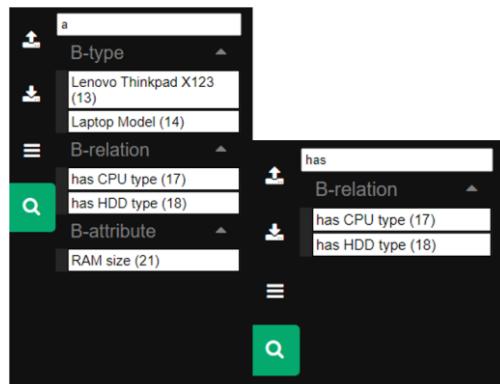


Obr. 57: Náhled grafického prostředí aplikace

Grafické prostředí na Obr. 57 reflektuje původní PURO Modeler ve snaze o jeho reimplementaci. Důvodem vytvoření nové aplikace je přehlednější a jednodušší implementace slučování. Autor také využil funkcionalit GoJS, jako je například vytváření skupin uzlů¹⁰.

V aplikaci má uživatel k dispozici paletu s entitami všech pěti termů definovaných v PURO. Na levé straně obrazovky je menu s výběrem kategorií pro obsluhu aplikace. Jedná se především o nástroje na vyhledávání uzlů v grafu, či import a export modelů. V pravé části je krom palety dostupné i kontextové menu, umožňující přehlednou úpravu jednotlivých uzlů. Kontextové menu mění barvu v závislosti na vybraném uzlu, aby bylo lépe rozeznatelné, o jakou kategorii uzlu se jedná. Kontextové menu je vytvořené autorem a je nezávislé na knihovně GoJS.

Vyhledávání je rozděleno do kategorií podle termů. Při vyhledávání jsou nabízeny pouze relevantní kategorie uzlů. Při kliknutí na kolonku je příslušný uzel označen a obrazovka je přesunuta k uzlu.

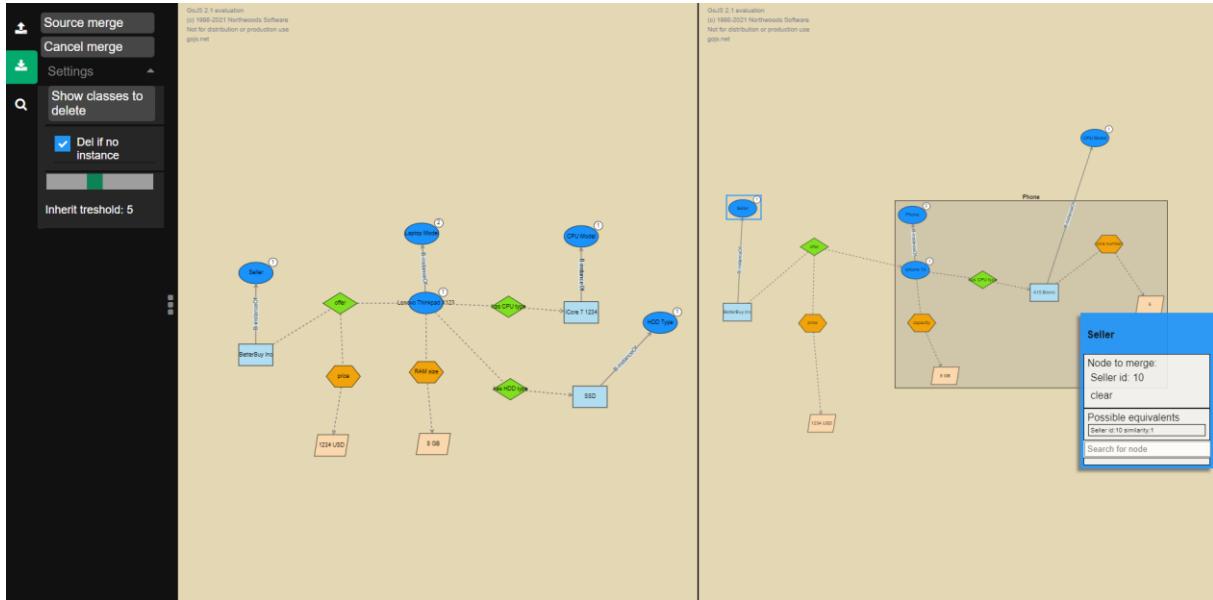


Obr. 58: Implementace vyhledávání entit

¹⁰ Aplikace je dostupná na: <https://thecandy.github.io/PUROMJoiner/>

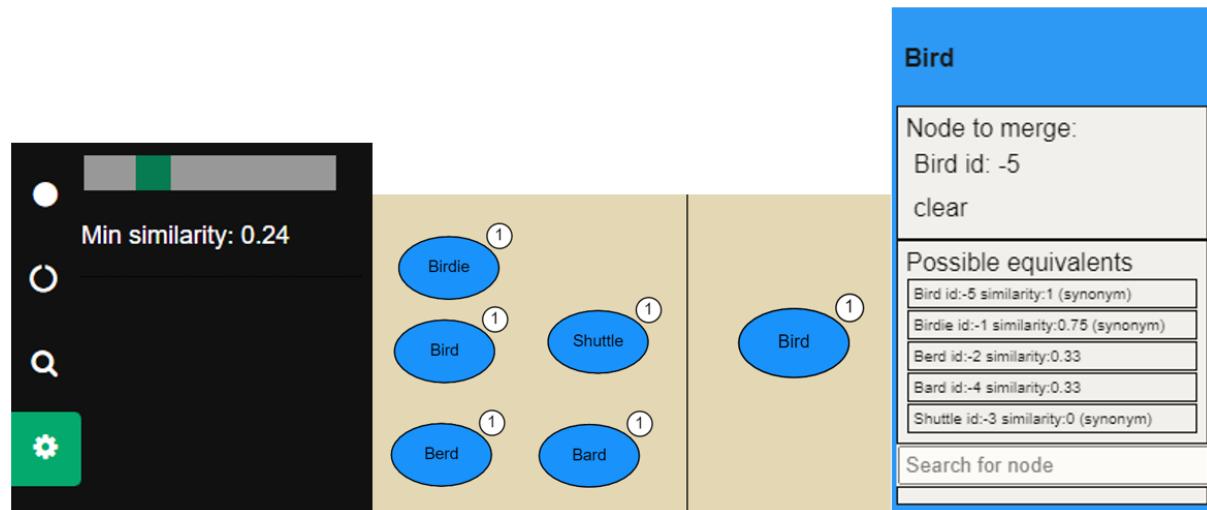
Obr. 58 ilustruje příklad vyhledávání entit. V levé části je nabídka potenciálních entit vzhledem k vyhledávanému textu rozsáhlejší. Postupným rozšířením dotazu se zužuje nabídka entit a jejich kategorií.

Pro slučování dvou modelů bylo navrženo současně zobrazování dvou grafů vedle sebe:



Obr. 59: Grafická implementace mapování dvou modelů.

Na Obr. 59 je původní graf na levé straně a slučovaný na pravé. V menu na levé straně lze vybírat obecné možnosti pro sloučení. Na obrázku jsou zobrazeny možnosti pro slučování podle zdroje. Kontextové menu při slučování slouží pro výběr potenciálních páru.



Obr. 60: Výběr shodné entity

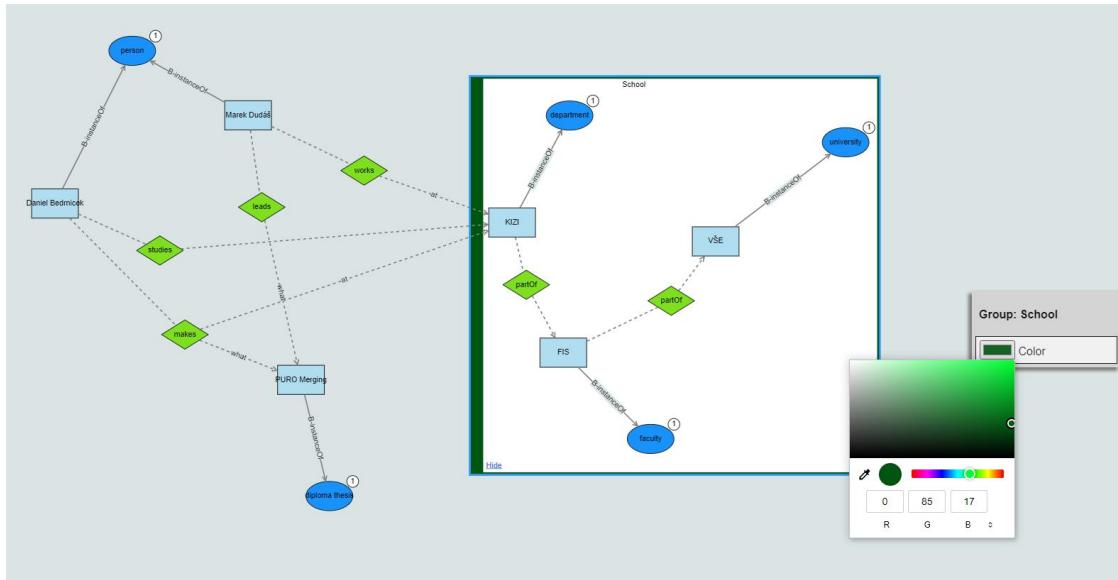
Obr. 60 je rozdělen do tří částí, které zobrazují: a) nastavení minimální podobnosti textu (vlevo) b) náhled slučovaných modelů (uprostřed) c) kontextové menu elementu (vpravo).

Část b) na Obr. 60 popisuje původní model s pěti různými entitami a slučovaný model s entitou „Bird“. V kontextovém menu c) jsou uvedeny všechny potenciálně shodné entity. I přes nesplnění minimální

podobnosti textu a) nastavenou na hodnotu 0,24 je nabízen uzel „Shuttle“, jelikož se jedná o synonymum.

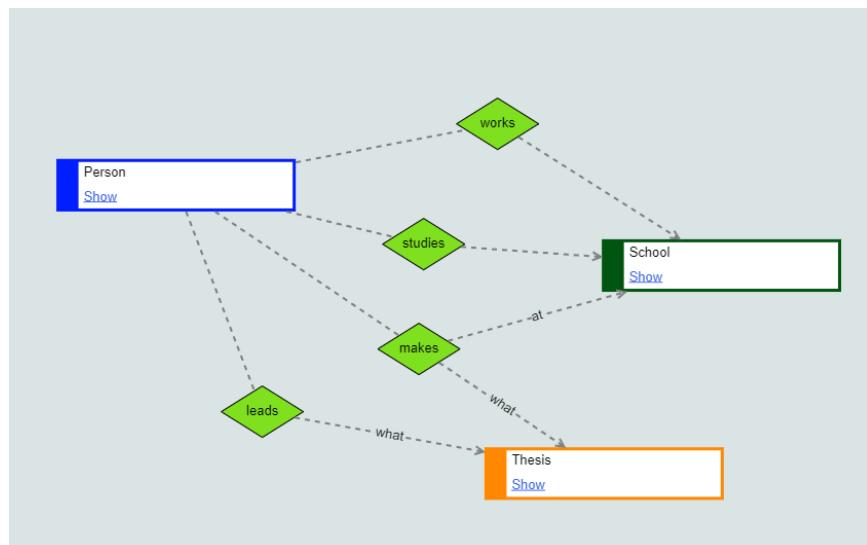
Uživatel má možnost vybírat z nabídky potenciálních ekvivalentů a případně i vyhledávat jakýkoli uzel stejné kategorie v případě, že nebyla jeho podobnost dostatečná.

Shlukování entit probíhá pomocí klávesové zkratky (ctrl + g). Výsledkem je box obalující entity (viz Obr. 61).



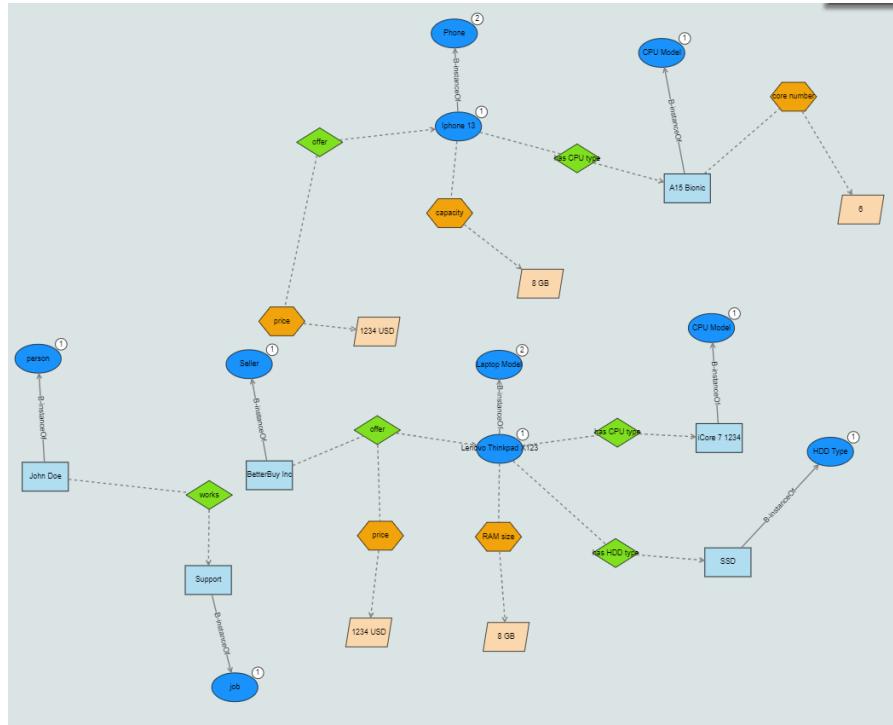
Obr. 61: Shlukování entit a kontextové menu

Výsledné skupině je možné měnit název a v kontextovém menu upravovat barvu pruhu pro lepší přehlednost. Při vytvoření a zabalení skupin na Obr. 62 vzniká výrazně přehlednější situace, která je při správně zvoleném názvu stále dobře čitelná.



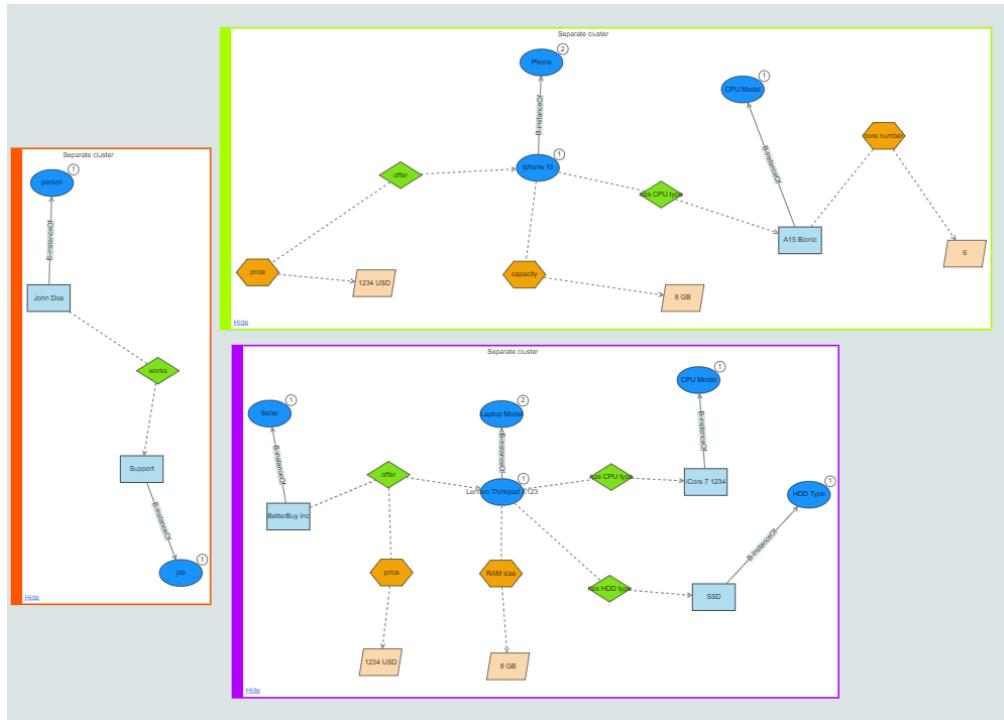
Obr. 62: Minimalizované shluky entit

Skupiny je zároveň možné využít při vizualizaci výsledků hledání souvislého grafu. Všechny uzly každé komponenty souvislosti mohou být zabalenы do individuálních boxů.



Obr. 63: Graf s nepropojenými částmi

Obr. 63 popisuje graf se záměrně vynechanými vztahy. Uzly, kterým byly pro účely příkladu odstraněny linky jsou přesunuty blízko k sobě, aby byla demonstrována užitečnost funkcionality vyhledávání komponent souvislosti v grafu.



Obr. 64: Nepropojené části grafu ve shlucích

Obr. 64 ilustruje výsledek tohoto vyhledávání a zapouzdření nepropojených grafů. Každému boxu je pro rozlišení přiřazena náhodná barva. Uživatel pak může tyto nepropojené uzly vyhledat, propojit a případně odstranit vytvořené boxy. Skupiny stejně jako uzly podléhají force-directed rozložení a jsou proto rozmístěny tak, aby se nepřekrývaly. Zároveň je možné grafy postupně propojovat a opakovat spouštět kontrolu, dokud nezůstane jediná box se skupinou uzelů.

9 Případová studie

Pro ověření funkčnosti aplikace se autor rozhodl reprodukovat již existující ontologii. Účelem je demonstrace toho, jak by potenciální ontologie mohla vznikat na základě PURO modelu vytvořeného z konkrétních příkladů z reálného světa. Vyvinutá aplikace umožňuje slučovat modely, a proto je výhodné vytvářet základ budoucí ontologie po částech. Zvýší se tím přehlednost při návrhu modelu a tvůrce se pak může soustředit pokaždé pouze na jednu vymezenou oblast. Při slučování modelů je naopak možné zaměřit se na oblasti, které by se napříč modely mohly potenciálně překrývat.

Vývoj OBM by tak bylo možné rozdělit na dvě fáze:

- Vývoj jednotlivých modelů
- Postupné slučování

Přinosem tohoto rozdělení vývoje na návrh jednotlivých částí a následné slučování by mohlo být lepší soustředění uživatele a minimalizování chyb způsobených nepřehledným modelem. Úzké zaměření vždy na jednu oblast tvorby a poté na úzkou oblast překryvu by mohlo vést k menší nepřehlednosti.

Autor pro reprodukci vybral ontologii FOAF (Friend of a Friend) [25]. Obecně se ontologie zabývá popisem vlastností lidí, skupin a dokumentů a vztahů mezi nimi. Ve své základní podobě není omezená kontextem současné doby a je proto možné tuto ontologii využít i pro popis historických osob.

Toto jádro (Core) je tvořeno následující základní sadou tříd a vlastností [25]. V závorece jsou vyjádřeny vztahy opačným směrem (např. popisuje – je popisován).

Třídy:

- **Agent:** Entity, které něco dělají
 - **Person:** Člověk
 - **Group:** Skupina. Sada individuálních agentů (komunita, skupina lidí). Skupina sestává z agentů a sama je agentem.
 - **Organization:** Organizace je do určité míry podobná skupině, ale je více formální a trvalá. Jedná se o firmy, společnosti apod.
- **Project:** Jakýkoli formální, či neformální projekt
- **Document:** Fyzický, či elektronický dokument
 - **Image:** Specifický dokument v podobě obrázku

Základní vlastnosti:

- **name:** jméno (obecně všech věcí)
- **title:** společenské a akademické tituly (pan, paní, Ing.)
- **img:** vztah mezi obrázkem a osobou kterou popisuje.
- **depiction (depicts):** odkazuje na obrázek popisující entitu.

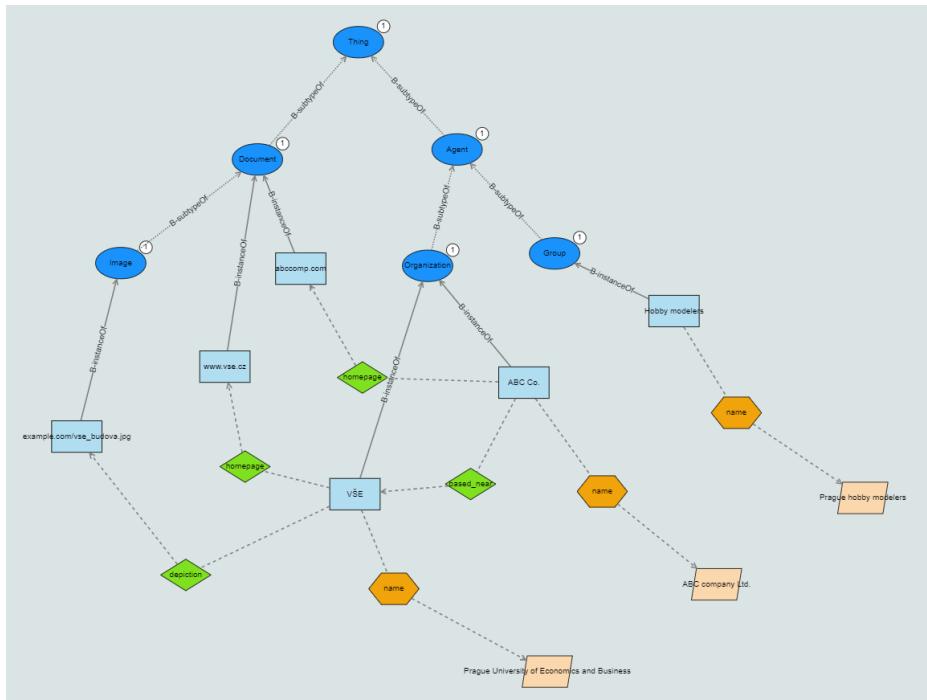
- **familyName:** Příjmení člověka
- **givenName:** Jméno člověka
- **knows:** člověk zná člověka
- **based_near:** geografická blízkost dvou entit
- **age:** doba existence agenta
- **made (maker):** Agent vytvořil věc
- **primaryTopic (primaryTopicOf):** věc, která je hlavním tématem dokumentu
- **member:** agent je členem skupiny

FOAF může být rozšířen další terminologií v podobě Social web, tedy vztahů týkajících se sociálních sítí a uživatelských účtů a entit na internetu. Zajímavými vlastnostmi jsou například schoolHomepage a workplaceHomepage. Tyto vlastnosti odkazují na dokument, který je webovou stránkou organizací, ve kterých člověk pracuje, nebo kam chodí do školy. Organizace by tedy měla mít vlastnost homepage, která odkazuje na webovou stránku, která je třídou dokumentu. Další zajímavou vlastností je currentProject, odkazující na dokument, který je současným projektem nějakého člověka, například diplomová práce.

V dokumentaci FOAF se popisuje problematický vztah mezi třídou Document a Project. V současné době není ustanovenno, jak propojovat dokumenty, které jsou projektem s třídou Project. Zároveň nejsou specifikované spojení mezi třídou Project a vlastnostmi currentProject a pastProject [25]. Autor si dovoluje pro názornost případové studie použít currentProject jako vztah mezi člověkem a projektem. Autor zároveň použije ve FOAF neexistující vztah hasDocument, odkazující na dokument projektu.

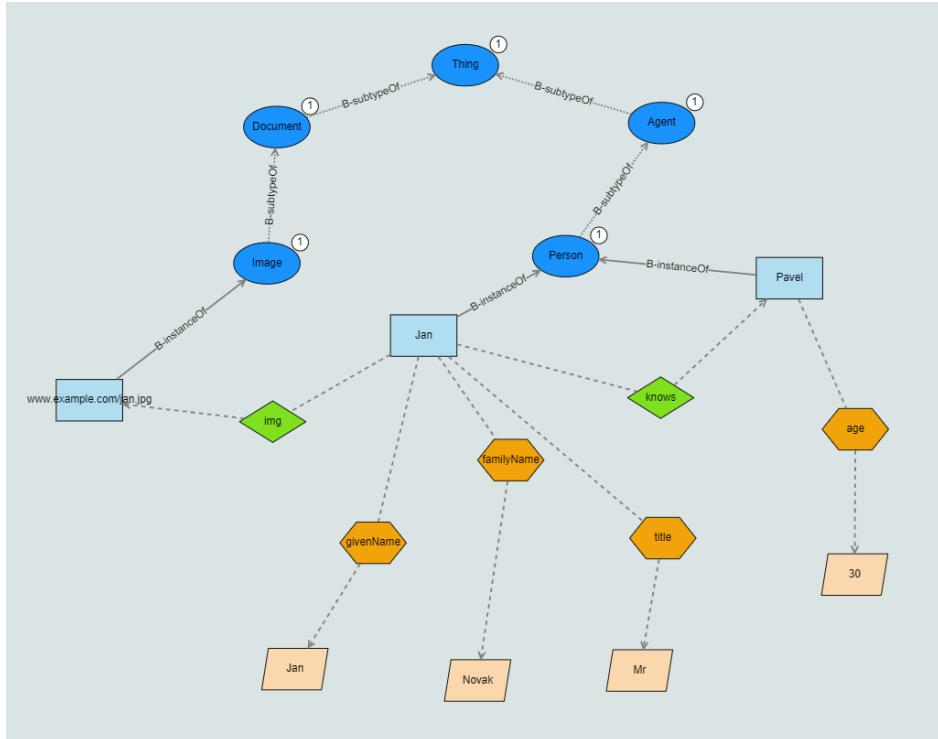
Autor vytvořil tři PURO modely na základě FOAF ontologie:

1. Model popisující organizace a skupiny (viz Obr. 65)
2. Model popisující osoby a jejich vlastnosti (viz Obr. 66)
3. Model vztahů mezi osobou a dalšími entitami. (viz Obr. 67)



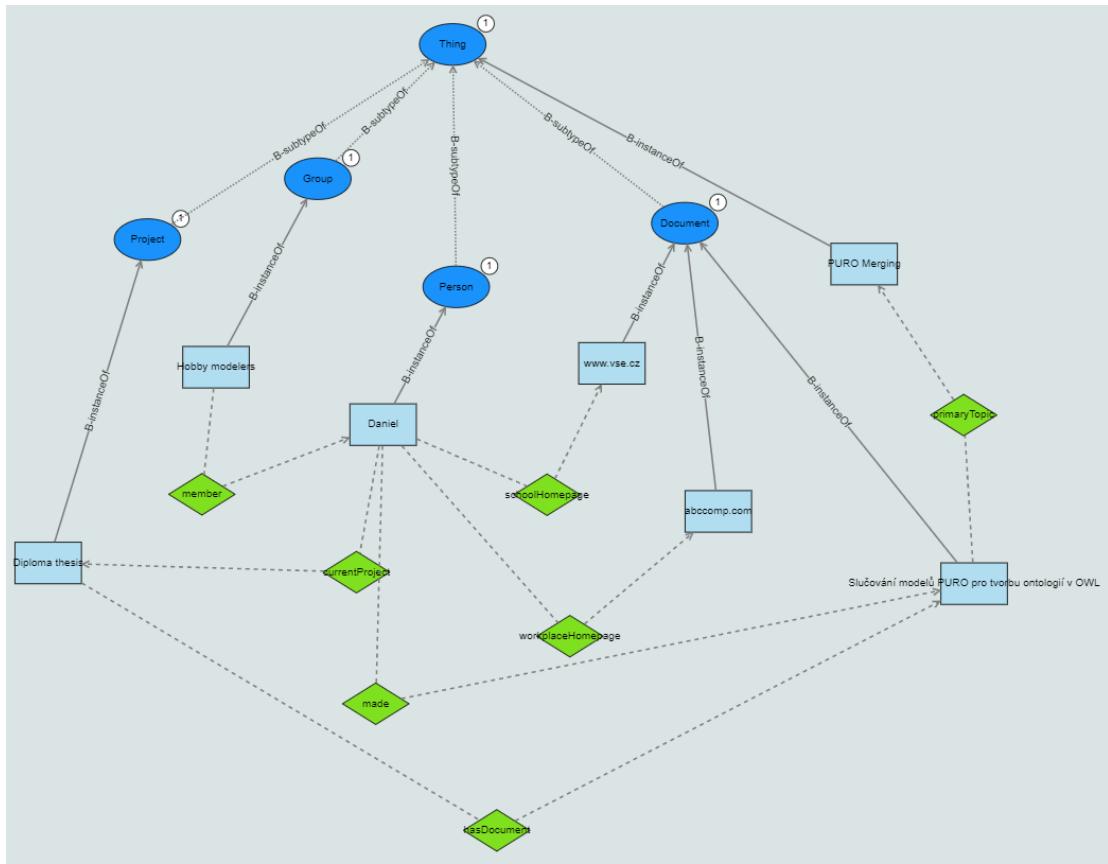
Obr. 65: Model popisující organizace a skupiny

Obr. 65 ilustruje model vztahů entit školy a firmy jako organizace. Dále je zde modelářská skupina jakožto skupina. Entity mají atribut název. Dále je zde pak možnost vyjádřit vztah s webovou stránkou organizace a vztah s obrázkem pro vizuální popis organizace (např. fotografie budovy školy).



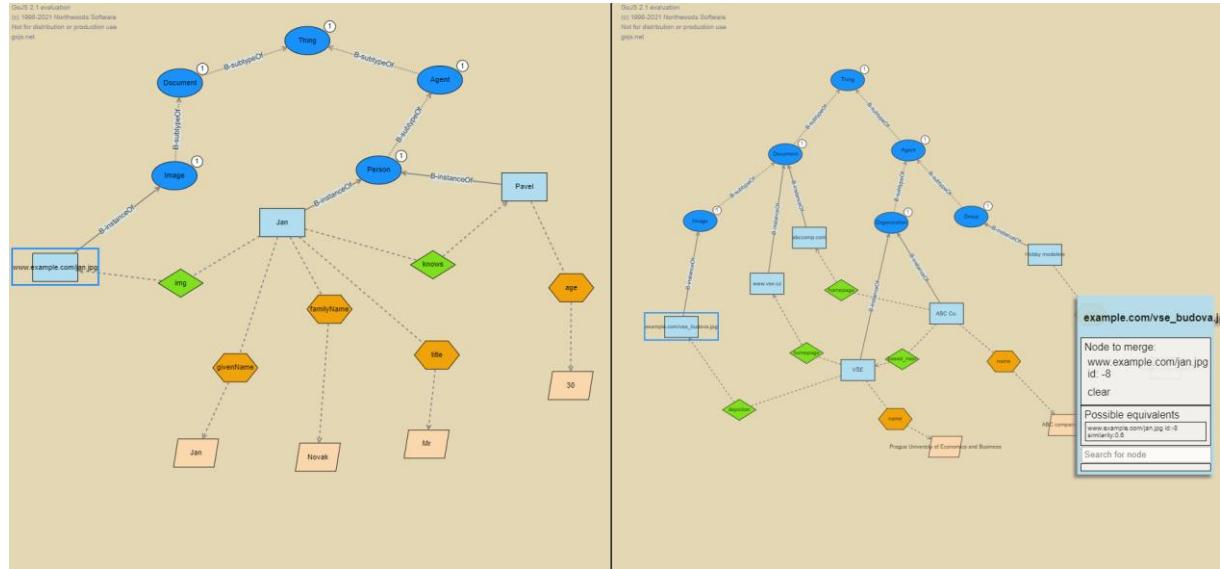
Obr. 66: Model popisující osoby

Obr. 66 popisuje osoby a jejich atributy: jméno, příjmení, titul a věk. Dále jsou zde relace „zná“ pro vyjádření vztahu mezi dvěma osobami a obrázek, který je fotografií člověka.



Obr. 67: Model vztahů mezi osobou a dalšími entitami

Poslední model na Obr. 67 popisuje člověka, který chodí do školy, pracuje ve firmě a tvoří projekt. Tento projekt má svůj dokument, který má nějaké hlavní téma. Zároveň je osoba členem skupiny.



Obr. 68: Náhled slučování dvou modelů

Na Obr. 68 je náhled slučování dvou výše uvedených modelů. Při slučování má autor možnost rozhodovat, které entity jsou shodné a mají být sloučeny (viz Obr. 69).

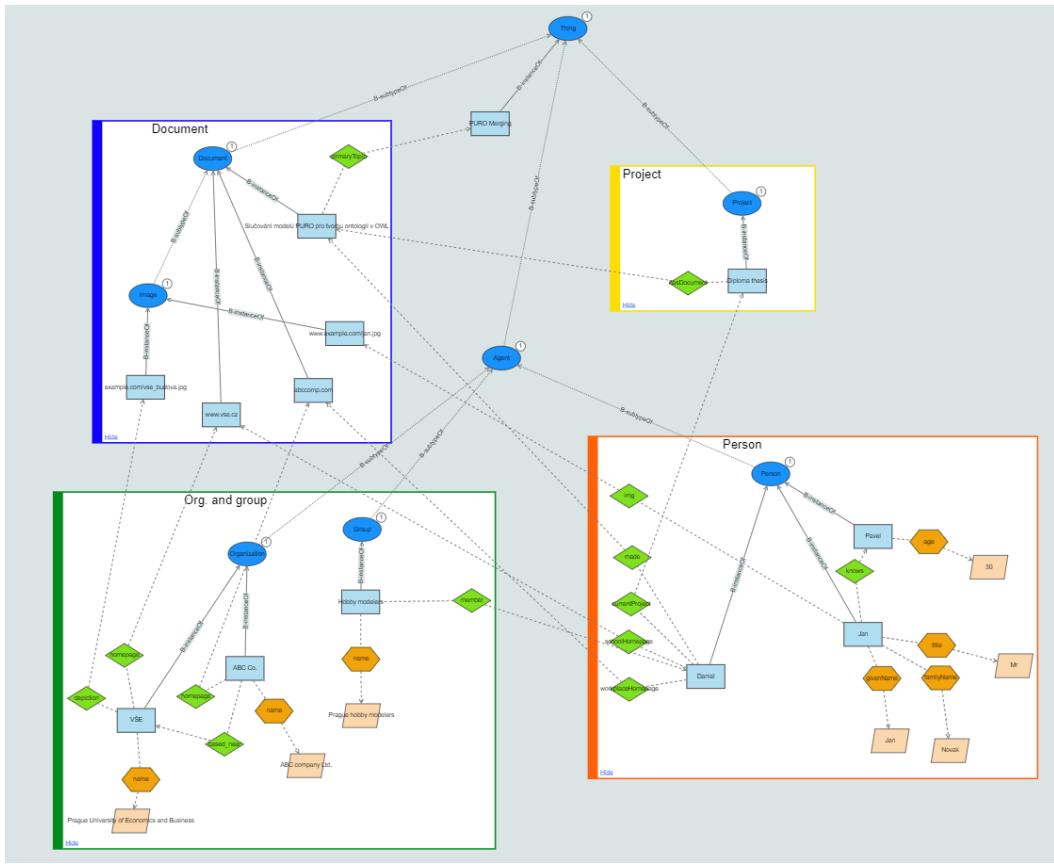


Obr. 69: Příklad kontextového menu

Toto kontextové menu je zobrazeno, protože uživatel vybral uzel b-objektu popisujícího url obrázku školní budovy. V možných ekvivalentech je nabízen b-objekt obrázku člověka. Podobnost byla vyhodnocena jako 0,6 na škále od 0 do 1. Uživatel měl zvolen nižší prah než 0,6 pro nutnou podobnost dvou entit, a proto byl uzel vyhodnocen jako vhodný ke sloučení. Na základě zvoleného prahu podobnosti dvou entit lze toto chování upravovat. Pokud uživatel nemá zájem tyto entity sloučit, může tak učinit pomocí tlačítka „clear“.

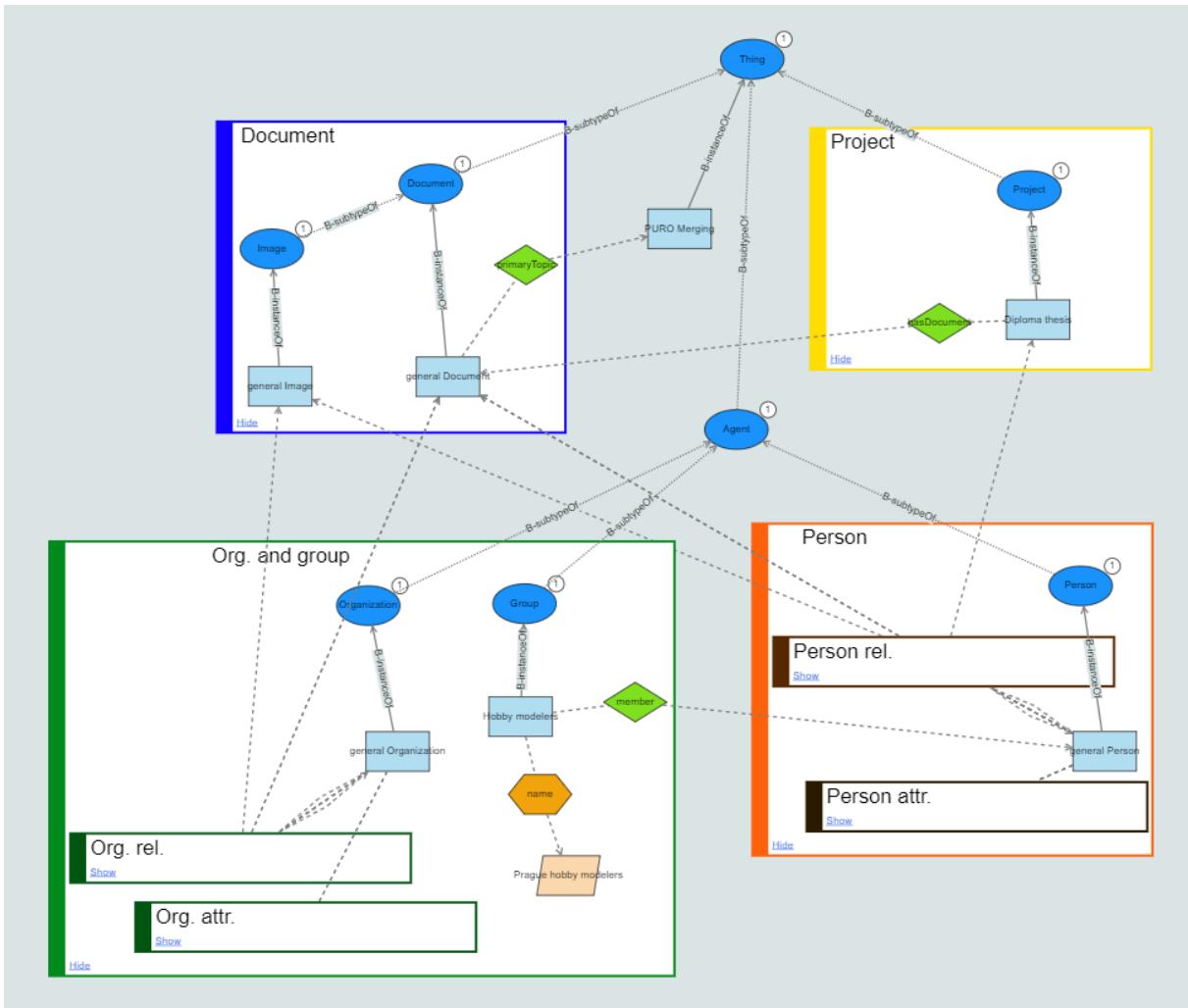
Autor ve všech případech slučoval pouze entity, které byly textově naprostě shodné. Jedná se o následující entity:

- B-typy
 - Thing
 - Person
 - Group
 - Document
 - Image
 - Agent
- B-objekty
 - Hobby modelers
 - abccomp.com
 - www.vse.cz



Obr. 70: Výsledek sloučení tří modelů.

Na Obr. 70 je náhled modelu po plném sloučení všech tří původních částí a po následném přesunutí uzlů a zabalení relevantních entit do logických skupin. V této fázi má autor modelu možnost sloučit všechny instance shodných tříd do jedné. Například v případě popisu člověka není potřebné mít tři instance, které se nakonec stejně do výsledné ontologie v OWL nepřenesou. Při exportu do nástroje OBOWLmorph je tedy žádoucí mít co nejmenší počet uzlů, jelikož nástroj nepodporuje seskupování uzlů.



Obr. 71: Výsledný model s použitím obecných instancí

Model na Obr. 71 má již sloučené uzly instancí a zabaleny uzly atributů a relací. V této podobě je model mnohem přehlednější, než tomu bylo původně. Takto si může autor modelu promyslet, zdali je vše propojené korektně a poté přistoupit k exportu do původního PURO Modeleru pro následný převod do OWL.

Závěr

Práce seznamuje čtenáře s jazykem PURO a s jeho využitím pro tvorbu modelů, které uživateli usnadňují orientaci v modelované doméně. Tyto modely slouží jako základ pro budoucí ontologii, jejíž vývoj v běžných jazycích jako je OWL nenabízí uživateli takovou možnost abstrakce jako jazyk PURO. PURO jazyk je srovnáván s ER modelem, který může zastávat podobnou funkci například při vývoji relačních databází. Tvůrce ontologie tak má možnost lépe se seznámit se všemi aspekty, které je nutné do ontologie zahrnout.

Práce popisuje problémové zobrazování větších modelů v původním PURO Modeleru a nastiňuje nutnost tvorby funkcionalit, které by zjednodušovaly náhled na celkový model. Zároveň s tím je popsána myšlenka tvorby menších individuálních modelů. Tyto modely by bylo možné vyvíjet nezávisle na sobě v mnohem přehlednější formě a následně sloučit do jednoho velkého modelu.

V souvislosti s tím je uvedena obecná teorie sémantického webu, ontologií a jazyků používaných pro popis ontologií. Dále práce popisuje problematiku mapování a slučování ontologií. Účelem je hledání využitelných postupů slučování z oblasti ontologií pro jejich aplikaci ve slučování PURO modelů. Stejnemu cíli, tedy hledání využitelných postupů, se věnuje kapitola 2.4.

Čtenář je dále seznámen s metodami vizualizace velkých grafů. V této oblasti práce identifikuje využitelné metody a funkcionality pro a zpřehlednění vizualizace PURO modelů. Jsou zde popsány konkrétní problémy, které mohou ve zobrazování PURO modelu nastat a postupy, jakými lze situace řešit.

Autor popsal problematiku slučování v PURO modelu na základě nalezených paralel se slučováním ontologií a ER modelů. Byly identifikovány problémy při slučování různých kategorií entit a možné řešení v podobě slučování na základě určité hierarchie těchto kategorií. Bylo definováno okolí entit b-typu a b-objektu, které pro tyto entity v grafu vyhledává uzly atributů, relací a valuací přímo napojených na b-typ, či b-objekt.

Práce popisuje nutnost výběru grafické knihovny pro zobrazování modelů. Jsou identifikovány nutné funkcionality a na základě toho je vybrána knihovna GoJS. Čtenář je poté obeznámen s klíčovými třídami a vlastnostmi knihovny.

Autor navrhuje algoritmy nutné pro úspěšné slučování dvou modelů. Tyto algoritmy implementuje¹¹ spolu s grafickým rozhraním vhodným pro zobrazování rozsáhlejších modelů a pro jejich slučování do webové aplikace¹².

Práce demonstруje funkčnost této aplikace reproducí ontologie FOAF v dílčích modelech PURO. Jejich následným sloučením a vhodným seskupením vzniká přehledný model, který je možné přenést do aplikace OBOWL Morph. V této aplikaci vzniká převodem do jazyka OWL základ ontologie.

¹¹ Kód aplikace dostupný na: <https://github.com/TheCandy/PUROMJoiner>

¹² Aplikace je dostupná na: <https://thecandy.github.io/PUROMJoiner/>

Výsledná aplikace není nijak propojená s původním PURO Modelerem. Aplikace sice nahrazuje modelovací část, ale je stále závislá na původním nástroji OBOWL Morph, ve kterém se model převádí do OWL. Možným řešením je rozšíření aplikace o možnost převodu modelu do jazyka OWL a tím kompletně nahradit původní nástroj. Současný OBOWL Morph totiž nepodporuje shlukování uzlů do skupin a další funkcionality, které byly pro novou aplikaci implementovány. Uživatel tak sice může v nové aplikaci pracovat s relativně přehledným modelem, ale při následném převodu do OWL je mu nabídnut neseskupený a potenciálně chaotický graf.

Většina funkcionalit aplikace byla záměrně vyvíjena nezávisle na použité knihovně GoJS. Tato knihovna je sice pro akademické účely zdarma, ale přesto by bylo vhodné aplikaci potenciálně reimplementovat ve volně dostupné knihovně. Slučování modelů a entit je totiž založeno na datových strukturách a funkcích dostupných v JavaScriptu. Použitá knihovna GoJS tak primárně slouží pro vizualizaci uzlů, podporu seskupování a sledování událostí při úpravě grafu.

Aplikace by mohla podporovat ukládání modelů na server v rámci účtu uživatele. Poté by bylo možné našeptávat možné názvy uzlů na základě již vytvořených modelů, případně podporovat úpravu modelů více uživateli najednou.

Dalším vhodným rozšířením by mohla být komplexnější logika porovnávání dvou potenciálně podobných uzlů na základě heuristiky, či strojového učení.

Použitá literatura

- [1] DUDÁŠ, Marek. *Podpora vývoje ontologií pomocí grafických modelovacích nástrojů*. B.m., 2018. Vysoká škola ekonomická v Praze.
- [2] WordsAPI [online]. [vid. 2021-08-25]. Dostupné z: <https://www.wordsapi.com/>
- [3] Word Dictionary API Documentation [online]. [vid. 2021-10-05]. Dostupné z: <https://rapidapi.com/twinword/api/word-dictionary/>
- [4] BERNERS-LEE, Tim, James HENDLER a Ora LASSILA. *The semantic web* [online]. 2001. ISSN 00368733. Dostupné z: doi:10.1038/scientificamerican0501-34
- [5] Ontologies - W3C [online]. [vid. 2021-07-21]. Dostupné z: <http://www.w3.org/standards/semanticweb/ontology.html>
- [6] RDF Primer [online]. [vid. 2021-10-26]. Dostupné z: <https://www.w3.org/TR/rdf-primer/?fbclid=IwAR1eSXeiW0SiuTOe7RtJsk3q0GSBULb9RFfuInqrjOflfKSaX7j1zz1qOOA>
- [7] RDF Schema 1.1 [online]. [vid. 2021-09-19]. Dostupné z: <https://www.w3.org/TR/rdf-schema/>
- [8] OWL Web Ontology Language - Overview [online]. [vid. 2021-11-01]. Dostupné z: <https://www.w3.org/TR/owl-features/>
- [9] SHVAIKO, Pavel a Jérôme EUZENAT. Ontology matching: State of the art and future challenges. *IEEE Transactions on Knowledge and Data Engineering, Institute of Electrical and Electronics Engineers* [online]. 2013, **25**(1), 158–176. ISSN 10414347. Dostupné z: doi:10.1109/TKDE.2011.253
- [10] RAUNICH, Salvatore a Erhard RAHM. Towards a benchmark for ontology merging. In: *On the Move to Meaningful Internet Systems: OTM 2012 Workshops* [online]. 2012, s. 124–133. ISBN 978-3-642-33617-1. Dostupné z: doi:10.1007/978-3-642-33618-8_20
- [11] MARTINEZ-GIL, Jorge a José ALDANA-MONTES. An overview of current ontology meta-matching solutions. *The Knowledge Engineering Review* [online]. 2012, **27**(04), 393–412. Dostupné z: doi:10.1017/s0269888912000288
- [12] KLEIN, Michel. Combining and relating ontologies: an analysis of problems and solutions. *Ontologies and Information Sharing* [online]. 2001, **47**(1), 53–62. Dostupné z: doi:citeulike-article-id:3936453
- [13] WIEDERHOLD, Gio. Mediators in the Architecture of Future Information Systems. *Computer* [online]. 1992, **25**(3), 38–49. Dostupné z: doi:10.1109/2.121508
- [14] CHALUPSKY, Hans. OntoMorph: A translation system for symbolic logic. In: *In Principles of Knowledge Representation and Reasoning* [online]. 2000, s. 471–482. Dostupné z: <https://www.isi.edu/~hans/publications/KR2000.pdf>
- [15] LI, Juanzi, Zhichun WANG, Xiao ZHANG a Jie TANG. Large scale instance matching via multiple indexes and candidate selection. *Knowledge-Based Systems* [online]. 2013, **50**, 112–120. Dostupné z: doi:10.1016/j.knosys.2013.06.004
- [16] CHEN, Peter Pin Shan. The entity-relationship model—toward a unified view of data. *ACM*

Transactions on Database Systems [online]. 1976, 1(1), 9–36. Dostupné z: doi:<https://doi.org/10.1145/320434.320440>

- [17] HE, Wentao. An Implementation of Entity-Relationship Diagram Merging. 2008.
- [18] CHECHIK, Marsha, Shiva NEJATI a Mehrdad SABETZADEH. A relationship-based approach to model integration. *Innovations in Systems and Software Engineering* [online]. 2012, 8(1), 3–18. ISSN 1614-5046. Dostupné z: doi:[10.1007/s11334-011-0155-2](https://doi.org/10.1007/s11334-011-0155-2)
- [19] SABETZADEH, Mehrdad. *Merging and consistency checking of distributed models* [online]. B.m., 2008. University of Toronto. Dostupné z: https://tspace.library.utoronto.ca/bitstream/1807/17256/1/Sabetzadeh_Mehrdad_A_200811_PhD_thesis.pdf
- [20] *homomorphism* [online]. [vid. 2021-11-27]. Dostupné z: <https://www.britannica.com/science/homomorphism>
- [21] SUK, Pavel. *Využití technik vizualizace velkých grafů pro zobrazení schémat RDF datasetů*. B.m., 2018. Vysoká škola ekonomická v Praze Fakulta.
- [22] *APACHE LICENSE, VERSION 2.0* [online]. [vid. 2021-11-22]. Dostupné z: <https://www.apache.org/licenses/LICENSE-2.0>
- [23] *Questions tagged [mxgraph]* [online]. [vid. 2021-11-02]. Dostupné z: <https://stackoverflow.com/questions/tagged/mxgraph>
- [24] DICE, Lee R. Measures of the Amount of Ecologic Association Between Species. *Ecology* [online]. 1945, 26(3), 297–302. Dostupné z: doi:[doi:10.2307/1932409](https://doi.org/10.2307/1932409)
- [25] *FOAF Vocabulary Specification 0.99* [online]. [vid. 2021-11-28]. Dostupné z: <http://xmlns.com/foaf/spec/>