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.

Konzultant diplomové práce:

Praha, 2021

Abstrakt

Návrh ontologií v jazyce OWL je při větším rozsahu poměrně nepřehledný. Práce se zabývá možnostmi rozšíření funkcionality PURO Modeleru vyvinutého pro práci s ontologiemi v grafickém prostředí. Cílem práce je prozkoumání možností vývoje aplikace podporující návrh složitějších ontologií v několika ohledech. Zásadní funkcionalitou je možnost slučování dílčích modelů vytvořených v jazyce PURO a také možnosti zlepšení přehlednosti navrhované ontologie.

Klíčová slova: PURO, OWL, ontologické inženýrství, grafické modelování

Obsah

[1 Sémantický web 6](#_Toc87887740)

[1.1 Ontologie 7](#_Toc87887741)

[1.1.1 Druhy ontologií 8](#_Toc87887742)

[1.2 RDF 8](#_Toc87887743)

[1.3 RDFS 9](#_Toc87887744)

[1.4 OWL 11](#_Toc87887745)

[1.5 PURO 11](#_Toc87887746)

[1.6 Mapování ontologií 12](#_Toc87887747)

[1.6.1 Druhy mapování 12](#_Toc87887748)

[1.6.2 Vyhodnocování sloučení ontologií 12](#_Toc87887749)

[2 Modelovací nástroje 18](#_Toc87887750)

[3 PURO Modeler 18](#_Toc87887751)

[4 Slučování modelů 19](#_Toc87887752)

[4.1 Skóre podobnosti entit 20](#_Toc87887753)

[4.2 Okolí uzlu 21](#_Toc87887754)

[4.3 Nalezení souvislého grafu 23](#_Toc87887755)

[5 Grafická reprezentace 24](#_Toc87887756)

[5.1 Skupiny 25](#_Toc87887757)

[5.2 Třídy 30](#_Toc87887758)

[6 Dostupné knihovny pro grafickou reprezentaci 31](#_Toc87887759)

[6.1 Požadavky 32](#_Toc87887760)

[6.1.1 Podpora různých grafických elementů 32](#_Toc87887761)

[6.1.2 Podmiňování tvorby vztahů 32](#_Toc87887762)

[6.1.3 Kontextová menu 32](#_Toc87887763)

[6.1.4 Složky a skupiny 33](#_Toc87887764)

[6.1.5 Formáty ukládání 33](#_Toc87887765)

[6.2 Knihovny 33](#_Toc87887766)

[6.2.1 GoJS 33](#_Toc87887767)

[6.2.2 mxGraph 34](#_Toc87887768)

[6.2.3 JavaScript InfoVis Toolkit 34](#_Toc87887769)

[6.2.4 Vis.JS 34](#_Toc87887770)

[7 Prerekvizity slučování 34](#_Toc87887771)

[7.1 Import 35](#_Toc87887772)

[7.1.1 Analýza 35](#_Toc87887773)

[7.1.2 Návrh 35](#_Toc87887774)

[7.1.3 Implementace 35](#_Toc87887775)

[8 Slučování tříd podle cíle 36](#_Toc87887776)

[8.1 Analýza 36](#_Toc87887777)

[8.2 Návrh 37](#_Toc87887778)

[8.2.1 Nalezení uzlů v hierarchii tříd 39](#_Toc87887779)

[8.2.2 Vytvoření cest grafem 40](#_Toc87887780)

[8.2.3 Hledání shodných uzlů 40](#_Toc87887781)

[8.2.4 Průchod cestami v grafu dolů 40](#_Toc87887782)

[8.2.5 Průchod grafem nahoru 41](#_Toc87887783)

[8.2.6 Odstraňování tříd 41](#_Toc87887784)

[8.2.7 Odstraňování redundantních vztahů 43](#_Toc87887785)

[8.2.8 Konečné sloučení 46](#_Toc87887786)

[8.3 Implementace 47](#_Toc87887787)

# 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 Consorcium (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 znovu.

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:

<https://www.wordsapi.com/>

{"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

Další příklad ze zdroje https://rapidapi.com/twinword/api/word-dictionary/ nabízí informace odděleně na základě samostatných dotazů. Uživatel se tedy musí zvlášť dotázat na synonyma, části a další informace o slově:

Dotaz na asociace ke slovu „mask“:

"assoc\_word":["hide","hat",face"]

"assoc\_word\_ex":["hide","hat","face","veil","disguise","camouflage"]

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.

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, jako je vyučovaný předmět na škole, nápad... Dalším krokem je pak propojování těchto entit pomocí vztahů. Tím se vytváří síť, ze které lze vyvozovat 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.

## 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.

### Druhy ontologií

* Doménové ontologie
* Generické ontologie
* Úlohové ontologie
* Aplikační ontologie

## RDF

Resource Description Framework RDF slouží pro popis skutečností z výseče reálného světa a vztahů mezi nimi. Propojením těchto skutečností vztahy 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 sémantického webu. Zdroj může být zjednodušeně chápán jako koncept velmi podobný URL. Je však nutné popsat jejich odlišnosti:

URI (Uniform Resource Identifier) je řetězec znaků popisující abstraktní, či fyzický zdroj. Slouží jako popis identity, či popis způsobu, jakým lze zdroj získat, případně oboje.

URL (Uniform Resource Locator) je podmnožinou URI. Rozšiřuje ho pomocí přímého popisu toho, jak je možné dostat se ke zdroji.

IRI přímo nerozšiřuje koncept URI. Podporuje rozsáhlejší znakovou sadu a slouží primárně k identifikaci zdroje. Dle definice je každé URI zároveň i IRI. Každé IRI může být mapováno právě na jedno URI. Více IRI však může být mapováno na jedno URI. Proto není zaručeno, že konverze z URI na IRI pokaždé vytvoří původní IRI.

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>

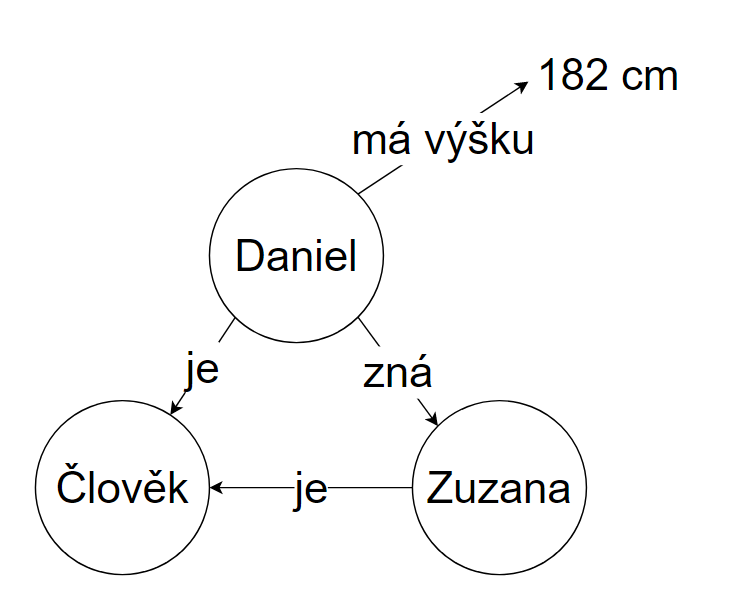
<Zuzana> <je> <člověk>

<Daniel> <má výšku> <182 cm>

<Daniel> <zná> <Zuzana>

Na příkladu lze pozorovat použití literálu pro popis výšky. Objekt „182 cm“ nelze jednoznačně identifikovat, jelikož se jedná v podstatě 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:



V tomto neformálním popisu 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 seznámen 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>

## RDFS

V samotném RDF lze popsat rozsáhlá fakta o entitách. Ze vztahů mezi nimi však nelze automaticky v podstatě nic vyvozovat. Pro rozsáhlejší podporu definic v RDF bylo vyvinuto RDFS, tedy RDF Schema. Hlavní koncepty používané v RDFS jsou následující:

|  |  |
| --- | --- |
| 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. 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.

## OWL

Owl (Web Ontology Language) je jazyk který rozšiřuje slovník popisu ontologií.

## PURO

PURO je dalším jazykem pro podporu vývoje ontologií. Hlavním cílem je vývoj modelu, nazývaného Ontology Background Model (OBM). Tento model by měl popisovat určitou oblast reálného světa na konkrétních příkladech „tak, jak jsou“. Samotný model však neslouží jako konečná ontologie, čí 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. 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 konkrétnosti ne. Objekty jsou entity se svojí 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.

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:
  + Konkrétní objekt se tímto způsobem odkazuje na třídu pro vyjádření instance.
  + Třída se odkazuje na jinou třídu. Toto je instancování třídy, což vede k existenci tříd různých úrovní. Třída může být instancí pouze třídy vyšší úrovně.
* B-subtypeOf:
  + Je vztahem dvou tříd na stejné úrovni. Účelem je vyjádření granularity. Subtype, tedy podtyp, je konkrétnějším vyjádřením obecnější třídy. Například člověk je podtyp savce.

## Mapování ontologií

Jedná se o proces, jehož cílem je provázání dvou a více ontologií přes určité spojitosti. Důvodem pro nutnost mapování a častou náročností tohoto procesu je rozdílnost mezi ontologiemi. 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. Doména – 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.

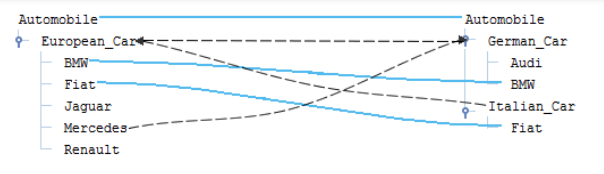
### Druhy mapování

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

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

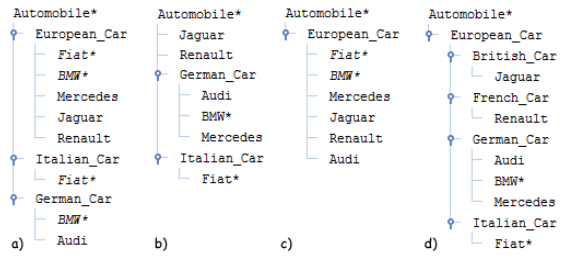
Dalším typem srovnávání je pak porovnávání entit z hlediska jejich vztahu s okolím. Takovou metodou je například grafová metoda, reprezentující ontologii jako graf. Srovnáváním hran a uzlů dvou ontologií lze vyvozovat podobnosti.

### Vyhodnocování sloučení ontologií



Na obrázku 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 stejné úrovni. To samé platí pro Německé auto.

Lze uvažovat následující čtyři typy slučování:



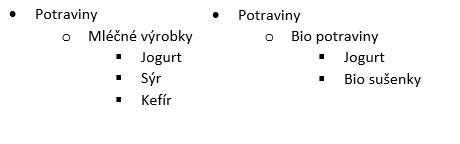
1. Plné sloučení, b) Slučování podle cíle, c) podle zdroje, d) expertní slučování

Plné sloučení zachovává všechny vztahy. Je zde však patrná redundance. 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.

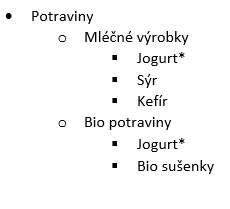
<evropské auto> <vyrobeno v> <Evropa>

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“, ačkoli je zjevné že by ho mělo mít. Zároveň však nemusí být automaticky žádoucí tento vztah přenést přes BMW na Audi díky stejné nadtřídě.

Mějme následující příklad:



Při plném sloučení by vznikl následující stav:



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

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

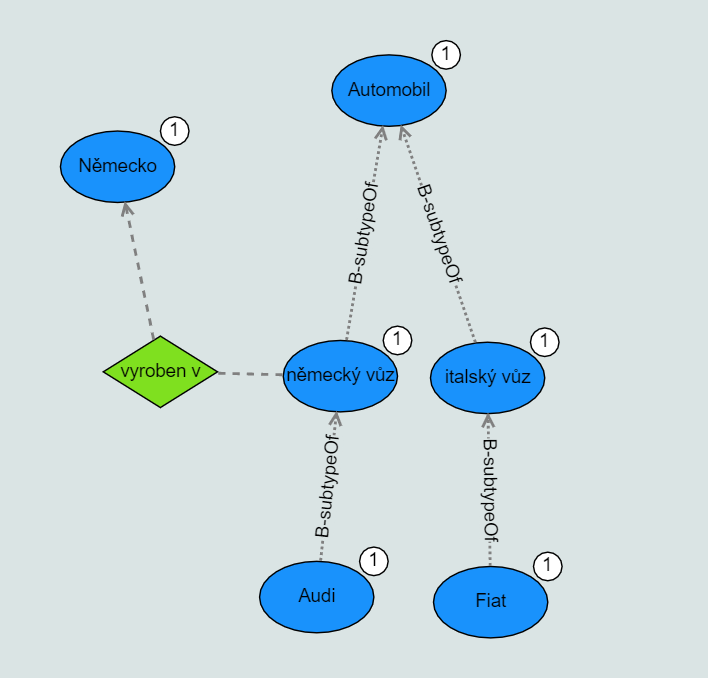
Pokud bychom pak automaticky tento vztah přes jogurt přenesli na bio sušenky, vznikla by pravděpodobně nežádoucí situace.

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 naprosto 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.

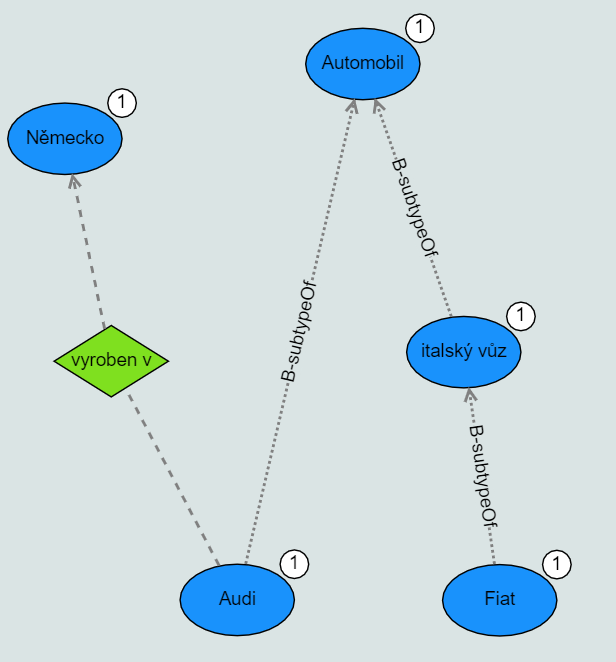
Slučování podle zdroje vyžaduje taky tento přenos vlastností vyšší třídy na podtřídy. Například vlastnost „německé auto je vyrobené v Německu“.

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

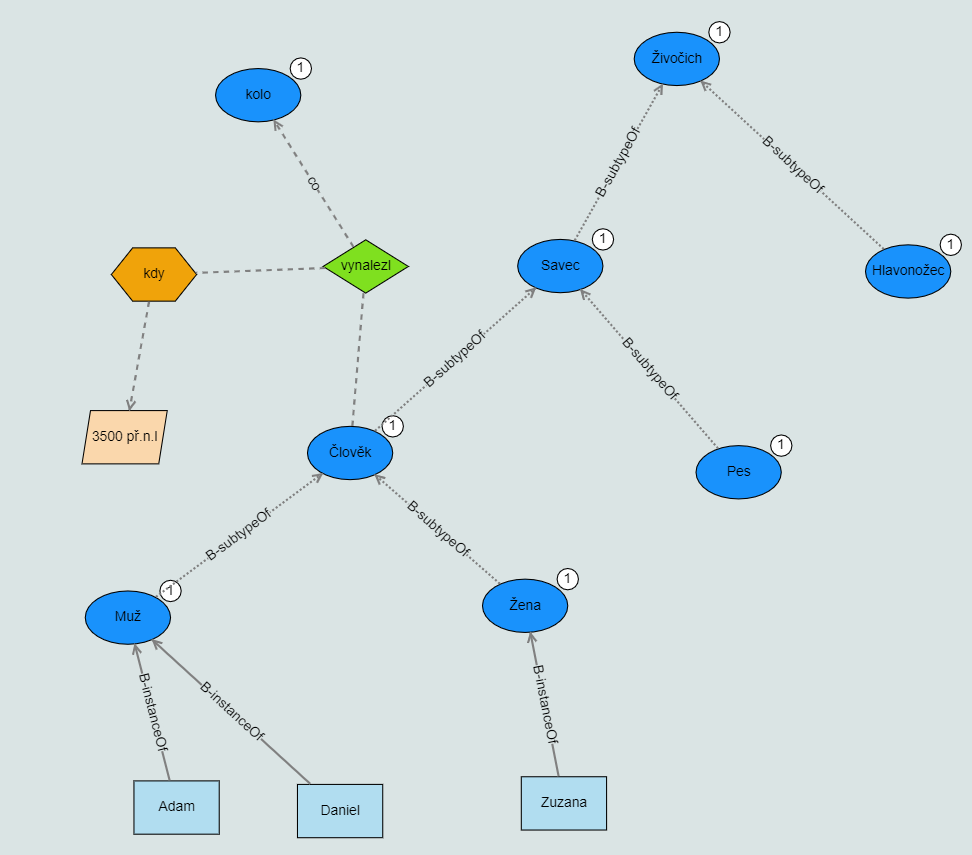
U slučování, při kterém dochází k odstranění některé ze tříd je nutné uvažovat o dvou směrech přenosu propojení s okolím na související třídy. Výše zmíněný proces se totiž týká pouze přenosu vztahů na podtřídy. Tyto vztahy totiž nelze přenést opačným směrem k nadřazené třídě.



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

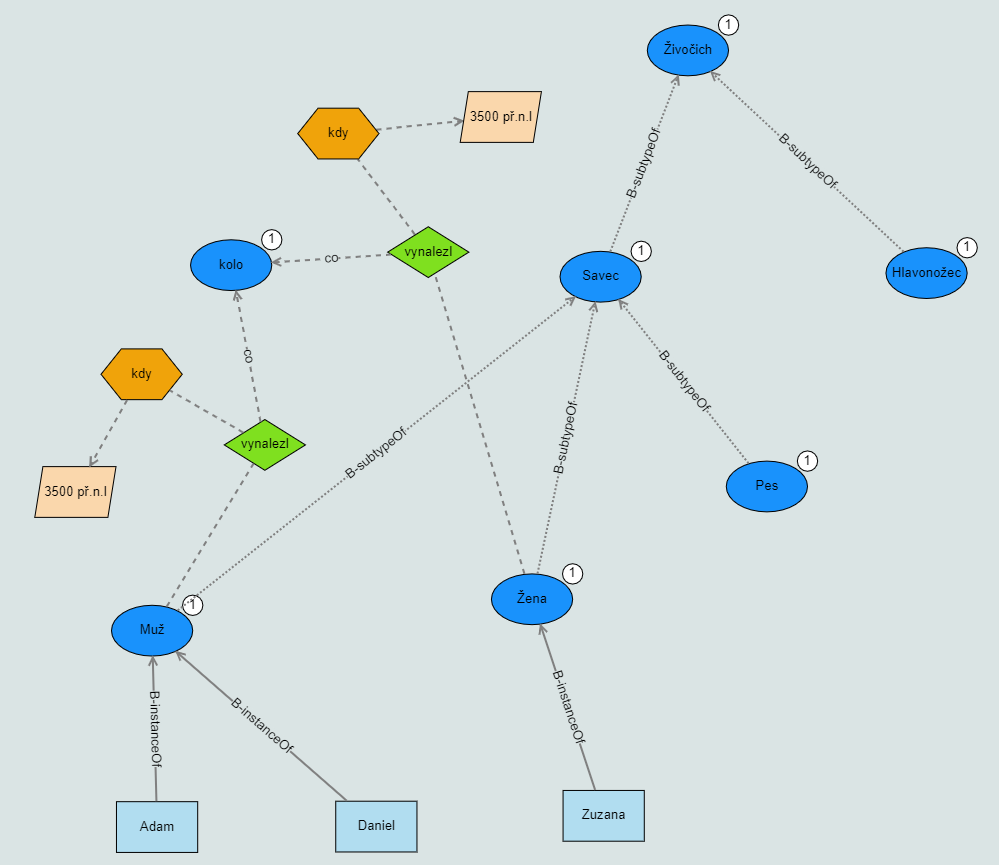


Další zmíněnou problematikou je pak přenos opačným směrem z podtřídy na nadřazenou třídu. Třídy totiž mohou mít instance. Při smazání třídy je pak nutno určit, k jaké jiné třídě má instance patřit.



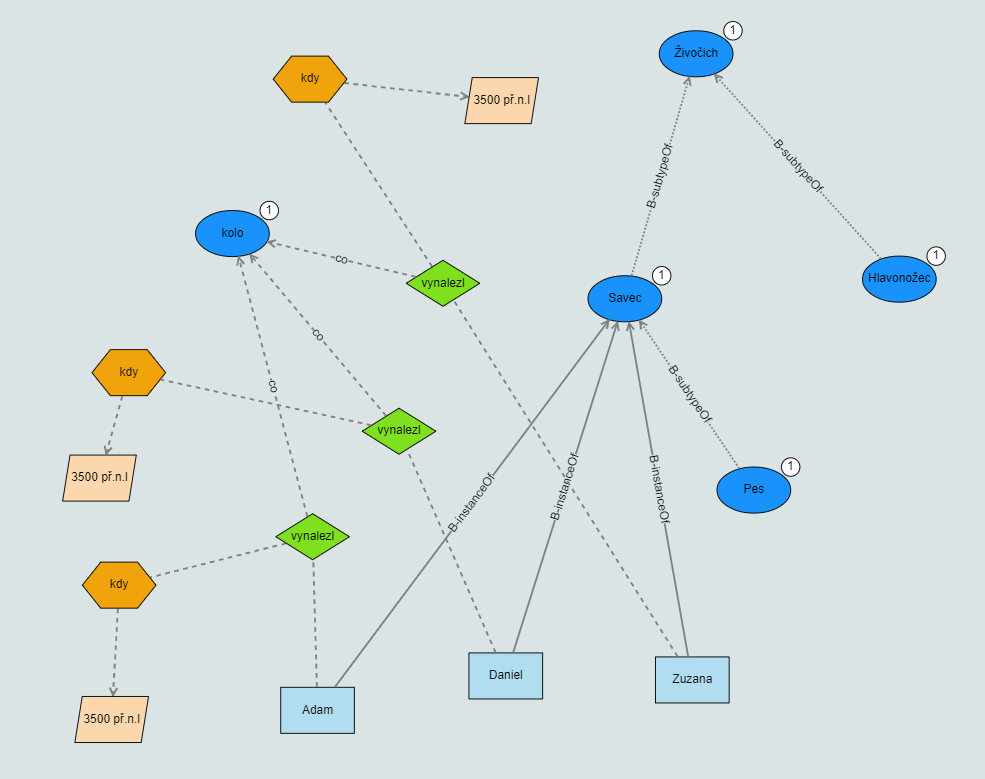
Na obrázku je popsána jednoduchá hierarchie živočichů. Instancí jsou pak muž Adam, Daniel a žena Zuzana. Dále je zde pro ilustraci vytvořen vztah popisující vynález kola. Při smazání některé ze tříd je pak nutné nejen přenést vztahy ale i vyjádření hierarchie. Pro názornost budou následující kroky postupně 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 směřována k uzlu „muž“ a druhý duplikát musí pro tuto hranu směřovat na uzel „žena“.
2. Hrany grafu vyjadřující hierarchii tříd se musí od „muž“ a „žena“ přesměrovat ze smazaného uzlu člověk na uzel „savec“.



Obrázek 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 dědit 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.



Je zřejmé, že na jedné straně dochází ke zjednodušování modelu z pohledu počtu uzlů tříd, avšak narůstá počet vztahů a model se při větším počtu uzlů může velmi rychle stát nepřehledným. Obecnější otázkou 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 opět nebylo žádoucí.

# Modelovací nástroje

# PURO Modeler

Cílem této práce je rozšířit funkcionalitu grafického modelovacího nástroje pro tvorbu ontologií PURO Modeler. V jeho současné podobě umožňuje navrhovat ontologie pomocí druhů 5 entit a vztahů mezi nimi.

Grafická reprezentace entit a vztahů umožňuje přehlednější práci, než je tomu u textových nástrojů. Vytvořené modely, nazývané OBM, lze převést v nástroji OBOWLMorph do OWL.

PURO Modeler má však určitá omezení. V prostředí lze vyvíjet jediný model. Ten je reprezentován uzly a vztahy. Z podstaty otevřenosti ontologií nelze předpokládat příliš mnoho omezení. V grafu nelze zavrhnout cykly, či předpokládat tvar stromu. Původní práce proto předpokládá, že modely o více než přibližně 50 uzlech se stávají nepřehlednými. S takovým omezením nelze předpokládat tvorbu rozsáhlejších ontologií.

Nabízí se proto představa vývoje menších částí ontologie a jejich následné slučování.

# Slučování modelů

Pro sloučení modelů je nutné rozlišovat podstatu jednotlivých entit, které se dělí na pět druhů:

* b-type: Analogie k OWL třídě
* b-object: OWL Instance
* b-relation: vztah mezi entitami
* b-attribute: vlastnost
* b-valuation: vyjádření kvantitativní hodnoty pro b-type, nebo b-object

Při slučování je nutné počítat s otevřeností ontologií. Předem lze tedy uvažovat pouze malé množství omezení.

Slučování modelů PURO má za cíl identifikaci párů shodných entit napříč modely a jejich spojení v jednu entitu. Společně s tím je nutné spojit i vztahy tohoto páru a připojit případné navazující uzly.

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. Ontologie 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 v nadřazené třídě.

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.

## Skóre podobnosti entit

V předchozí kapitole byla popsána potřeba tvorby systému pro vyhodnocení podobnosti dvou entit. Míra podobnosti entit pak může sloužit pro jejich automatické sloučení. 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.

Touto logikou však nelze postupovat u všech typů entit a je nutné respektovat určitou hierarchii. Lze předpokládat, že například třída, tedy b-type by měla mít unikátní název. Podobně lze postupovat u objektů, tedy instancí tříd. Zde je však nutné počítat s výše uvedenými příklady shodnosti. Název firmy a jméno člověka může být shodné. Jejich třída je však pravděpodobně odlišná 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 třídami 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 každé třídy 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 jiné třídě, 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 zamyšlení, zdali je možné, aby atribut byl ve vztahu s více hodnotami.
2. B-attribute je závislý na relaci, objektu, či třídě.
3. Relace je závislá na objektu, či třídě.
4. Objekt je instance třídy.

Z opačného konce hierarchie lze identifikovat pomyslné okolí objektů a tříd.

1. Pro každou třídu ve slučovaném modelu nalézt ekvivalent třídy dostatečně podobného názvu v původním modelu.
2. Pro každý objekt příslušící této třídě nalézt ekvivalent podobného názvu v původním modelu a identifikovat okolí objektu. Toto okolí zahrnuje všechny atributy, relace a hodnoty připojené bez přerušení k objektu.
3. Pro každou relaci, atribut a hodnotu nalézt stejný název a typ v okolí objektu v původním modelu.

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“

## 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.

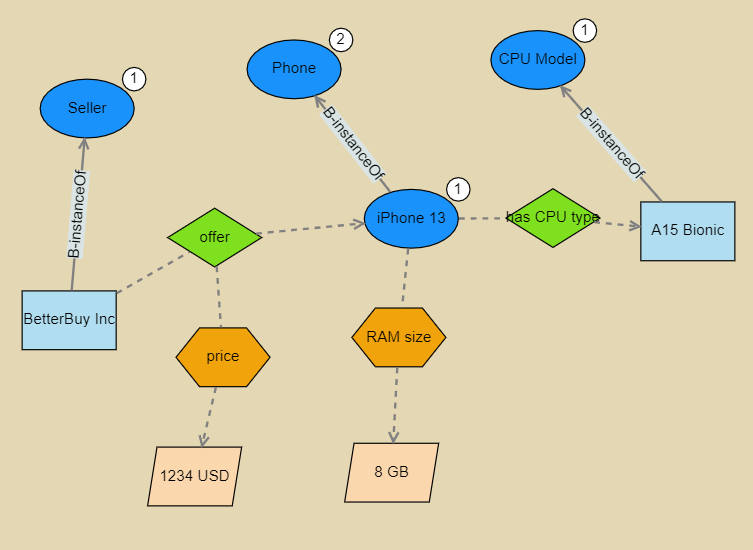
Z problematiky uvedené výše lze však pro toto srovnávání brát v úvahu pouze třídy a objekty. U všech ostatních entit nelze ze samotného názvu vyvozovat slučování. Proto je nutné tyto uzly porovnávat v kontextu okolí nadřazeného uzlu.

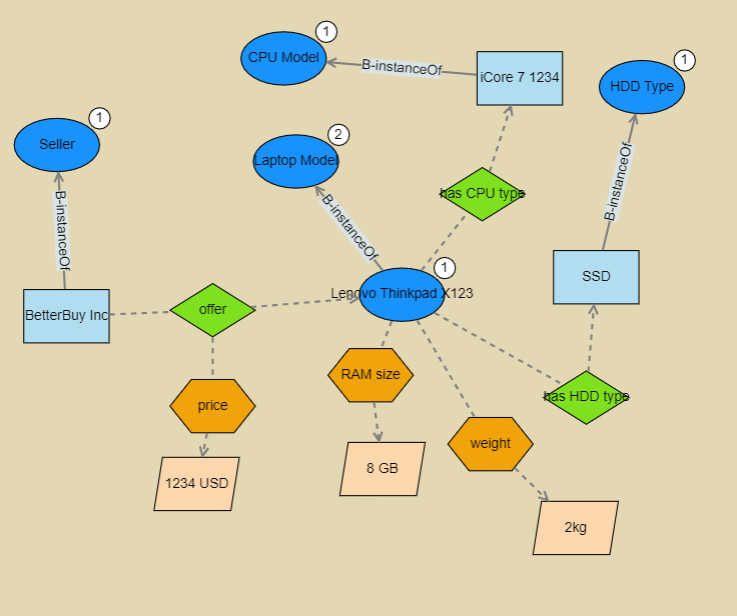
Okolí uzlu kategorie třída, či objekt lze nalézt pomocí následujícího algoritmu:

1. Vytvoř pole *„okolí“*, které bude obsahovat uzly z okolí a vlož do něj samotný uzel, kterému se okolí hledá.
2. Dokud je ještě co přidávat, tedy pokud od posledního cyklu proběhla změna v poli *okolí*, prováděj následující kroky:
   1. Pro každý uzel v poli *okolí* vyhledej připojené uzly.
   2. Pokud tyto připojené uzly náleží k jiné kategorii než třída, 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 uzlů v původním i slučovaném grafu. Pokud pak bude v tomto okolí nalezen uzel nižší kategorie, lze předpokládat, že se jedná o tentýž uzel. Při slučování dvou uzlů kategorie třída, či objekt je pak jejich okolí se stejnými, či podobnými názvy taky sloučeno.

Z tohoto postupu by se mohlo zdát, že mohou nastat nežádoucí situace, kdy bude sloučen uzel, který však má být unikátní.





Výše vyobrazené modely popisují nabídky telefonu a notebooku. Z obou modelů je patrné, že mnoho uzlů 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 třída „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 třída 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 třídám „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 každé třídy 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 pár 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.

## 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 je graf není souvislý, lze jeho části, které souvislé jsou, 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 pár 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 uzlu.
* 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:
   1. Vytvoř dočasné pole *„graf“*, popisující uzly v komponentě souvislosti.
   2. Dokud se počet uzlů v poli *graf* nepřestane zvyšovat, prováděj následující:
      * Pro každý uzel v poli *graf* vyhledej připojené uzly. Pokud tyto uzly ještě nejsou přidány do pole *graf*, přidej je.
   3. Přidej pole *graf* do pole *podgrafy*.
   4. Odeber všechny uzly obsažené v poli *graf* z pole *původní.*

Pole podgrafů pak obsahuje jednotlivé komponenty souvislosti. Z jejich množství a vztahů lze pak vyvozovat úspešnost sloučení jednotlivých modelů podle logiky uvedené výše.

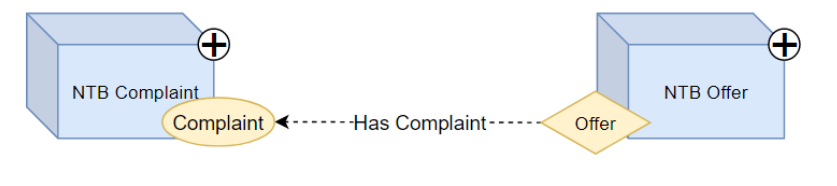
# Grafická reprezentace

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

Původní PURO Modeler se stává nepřehledným při tvorbě rozsáhlejších 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.

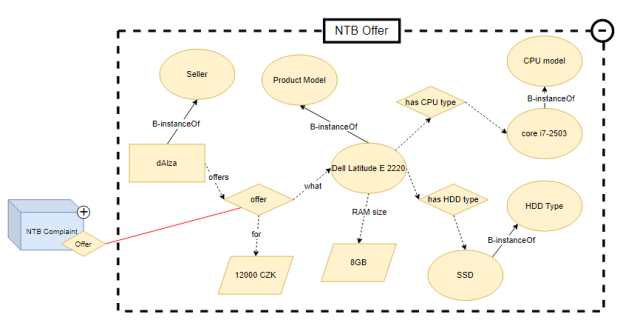
## Skupiny

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.

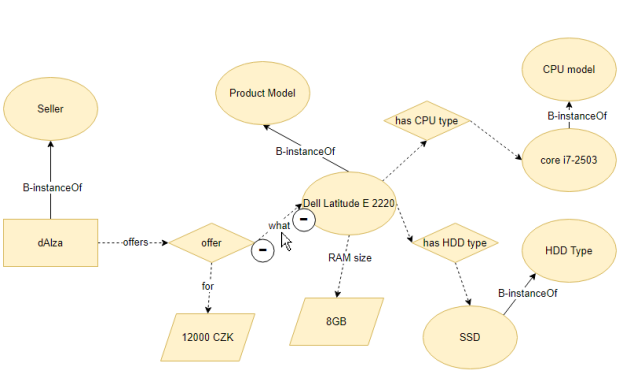


Na obrázku výše lze vidět návrh propojení dvou složek. NTB Offer vyjadřuje minimalizovaný graf obsahující informace o notebooku. NTB Complaint zahrnuje graf popisující reklamaci. Při jejich propojení byly zvoleny dva rozdílné uzly, které na sebe logicky navazují. Uživatel má možnost oba grafy nezávisle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

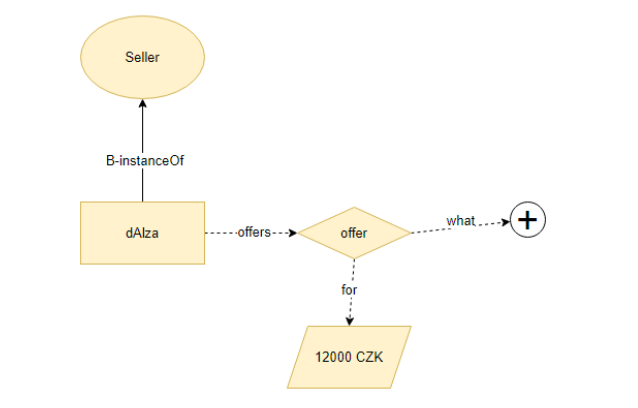
Přístup na obrázku výše podporuje tvorbu shodné entity na více místech, což nemusí být na první pohled tak intuitivní jako předchozí možnost. Výhodou však je, že vztah mezi výstupními body pouze vyjadřuje jejich propojení a zároveň jejich shodnost a do konečné ontologie se nepřenese. Při maximalizaci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



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.

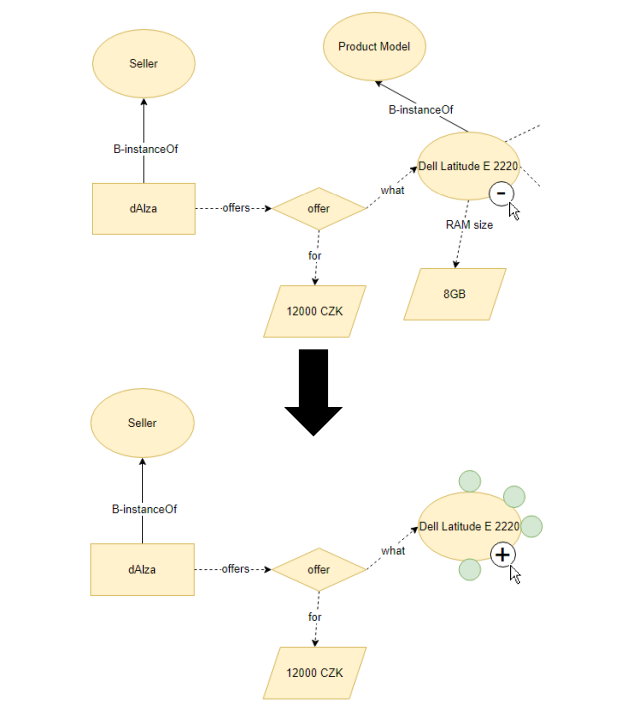


Na obrázku je vidět situace po najetí kurzorem na vztah mezi dvěma entitami. Zobrazí se dvě ikony umožňující minimalizaci jedné ze stran vztahu.



Obrázek výše 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.

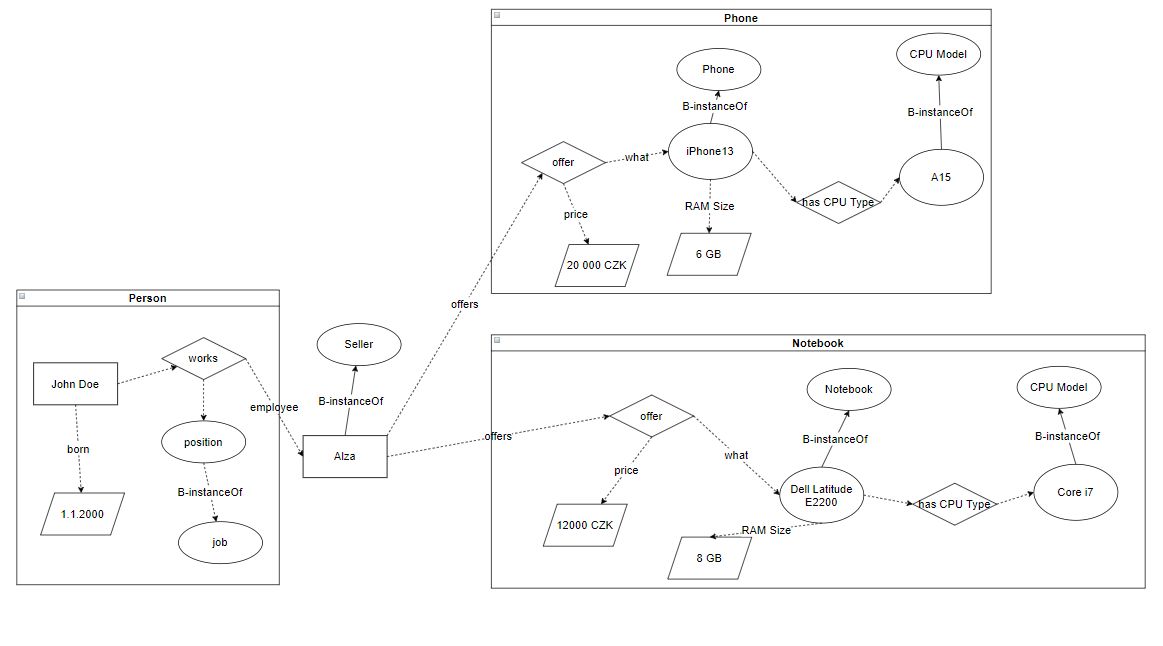


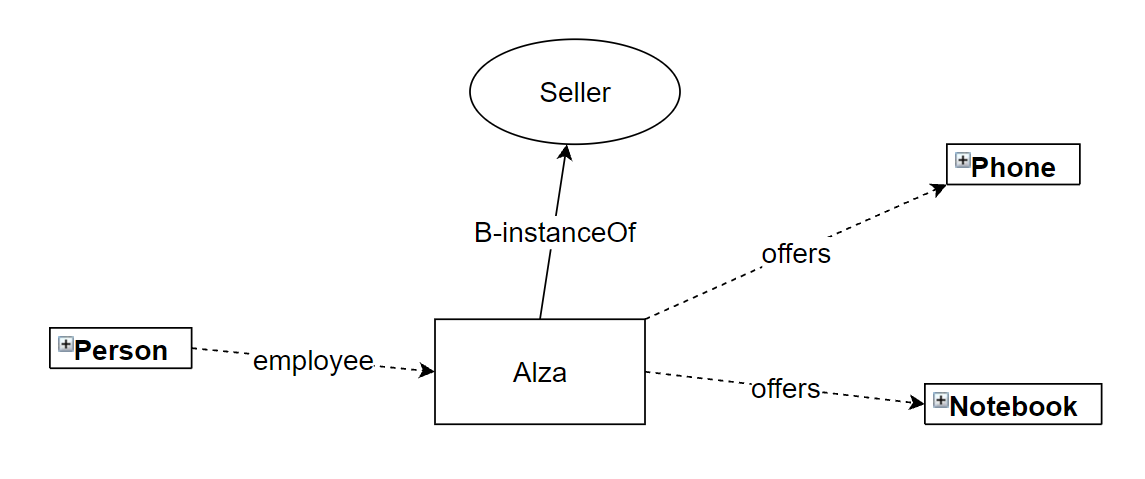
Faktem zůstává, že jakýkoli přístup neumožňující zobrazení pouze určité části sloučeného OBM při kontrole před převodem do ontologie pomocí nástroje OBOWLMorph však naráží znovu na původní omezení PURO Modeleru kvůli nepřehlednosti. Ostatně i samotný náhled v OBOWLMorph je omezen počtem uzlů ze stejného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

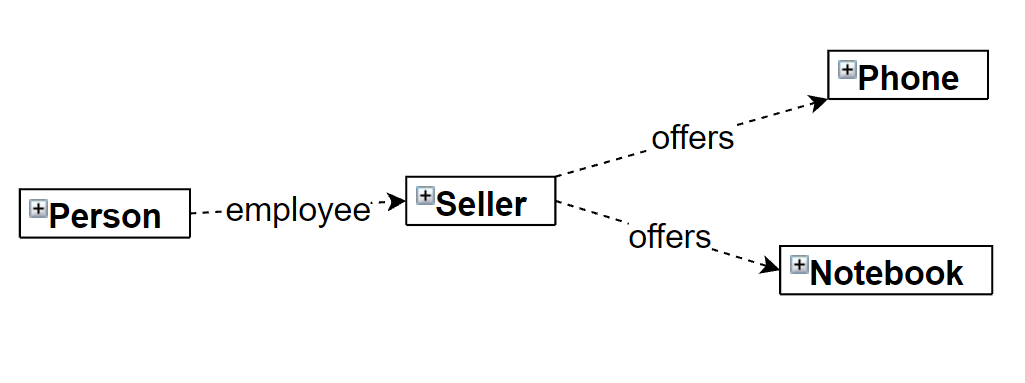
Uživatel by například mohl vybrat konkrétní uzly, které mají být minimalizovány. V tomto přístupu by nemusela být vynucována žádná omezení a předpokládalo by se, že tvůrce ontologie ví nejlépe, do jakých podskupin uzly shlukovat.

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.





Na obrázku je příklad tří skupin před a po minimalizaci. Každá skupina má pouze jeden uzel, který je propojen s okolím. Proto jsou vtahy při zjednodušeném náhledu stále patrné a přehledné.



Při zabalení posledních dvou uzlů 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.

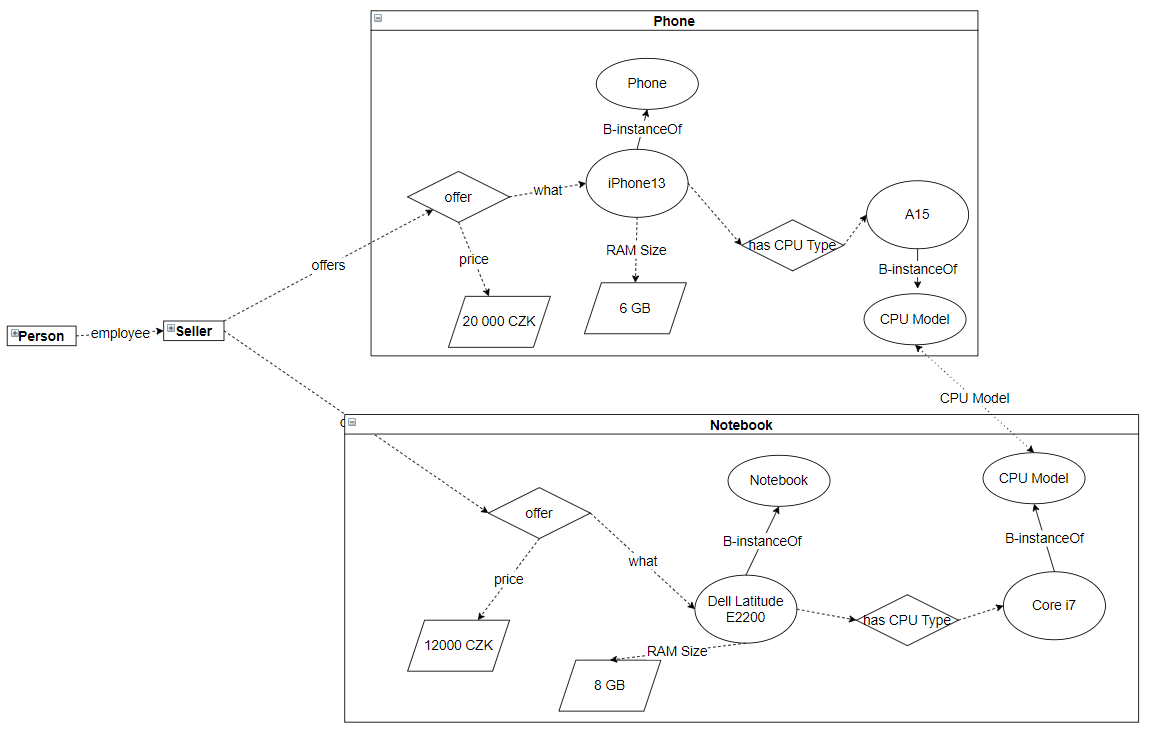
## Třídy

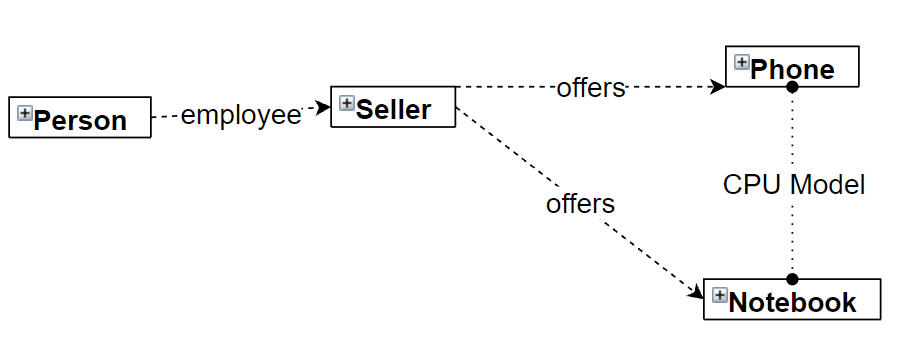
Z obrázků výše bylo patrné, že notebook a telefon nesou informaci o procesoru. Konkrétní procesor náleží třídě CPU Model. Tato třída by v modelu měla být zastoupena pouze jednou, protože v obou případech označuje to samé. Uživatel by však měl mít možnost rozhodnout se, zdali tuto třídu bude reprezentovat 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 nové třídy byla uživateli poskytnuta možnost tento uzel spřáhnout s již existující třídou. 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 třídy „CPU Model“, ale při minimalizaci skupin se tento vztah vytrácí.

Jedním z řešení je nabídnout uživateli možnost zobrazit dočasný propoj mezi těmito třídami. Linka mezi třídami by tak vyjadřovala, že jsou shodné. I při zabalení skupin by tak bylo možné vyvodit vztah přes tuto třídu.





Na obrázcích výše je patrný vztah mezi notebookem a telefonem přes třídu CPU Model. Tento vztah by bylo vhodné zobrazovat pouze na přání uživatele.

# 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 PRUO 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 funkcionalitu takové knihovny.

## 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 uzlů, změna velikosti náhledu, podpora shlukování a pokud možno replikace většiny funkcí z PURO Modeleru.

### 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 uzlů.

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

### 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 by bylo vhodné omezit počet propojení vycházejících z, či do uzlu.

### Kontextová menu

Při vytváření vtahu 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 modelů.

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.

### 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.

### 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.

## 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.

### GoJS

<https://gojs.net/>

GoJS 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.

### mxGraph

<https://jgraph.github.io/mxgraph/>

mxGraph je JavaScript knihovna podporující tvorbu grafů a diagramů.

Knihovna je dostupná pod licencí Apache 2.0.

### JavaScript InfoVis Toolkit

<http://philogb.github.io/jit/>

Nástroj pro vytváření vizualizací dat. MIT licence

### Vis.JS

<https://visjs.org/>

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

Dokumentace této knihovny není tak přehledná jako v případě GoJS.

Apache 2.0 licence

# Prerekvizity slučování

Pro základní funkcionalitu slučování modelů PURO není přímo nutné graficky reprezentovat jednotlivé modely a vztahy mezi nimi. Využitím jednoduché textové nabídky pro sloučení dvou, či více modelů by mohl být celý proces velmi rychlý a aplikace by pak sloužila jako prostý nástroj pro sloučení modelů vytvořených v původním PURO Modeleru. Požadavkem na uživatele je pak znalost všech slučovaných grafů, protože aplikace by poskytovala pouze omezené informace. Vhodnost sloučení jednotlivých uzlů by vycházela sice ze stejné logiky, která je v práci uvedena výše.

V nejjednodušší podobě by uživatel mohl do aplikace importovat několik souborů získaných z PURO Modeleru a získat sloučený model. Nastavením prahu skóre podobnosti entit by pak mohl určit, jakou nejnižší podobnost musí mít uzly, aby byly sloučeny.

Určitým rozšířením by pak mohl být seznam uzlů a k nim příslušící nabídka potenciálních párů. Díky tomu by pak měl uživatel větší kontrolu nad výsledkem.

Celý proces a jeho výsledek by pak bylo nutno konzultovat s původním editorem.

## 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 vtahy 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 stejné 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.

### Implementace

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:
   1. Uchovej původní ID uzlu jako novou proměnnou *původníID*.
   2. Nahraď jeho ID číslem *novéID*.
   3. Přičti k *novéID* jedničku.
3. Pro každý propoj importovaného grafu:
   1. Vyhledej v uzlech importovaného grafu *původníID*, které souhlasí s ID počátku propoje.
      1. Nahraď ID počátku propoje proměnnou *novéID* nalezeného uzlu.
   2. Vyhledej v uzlech importovaného grafu *původníID*, které souhlasí s ID konce propoje.
      1. Nahraď ID konce propoje proměnnou *novéID* nalezeného uzlu.

Tímto je zaručena jedinečnost každého uzlu a jeho propojení.

# Slučování tříd podle cíle

## Analýza

Slučování z hlediska tříd bylo již popsáno 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“. Vývojáři 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:

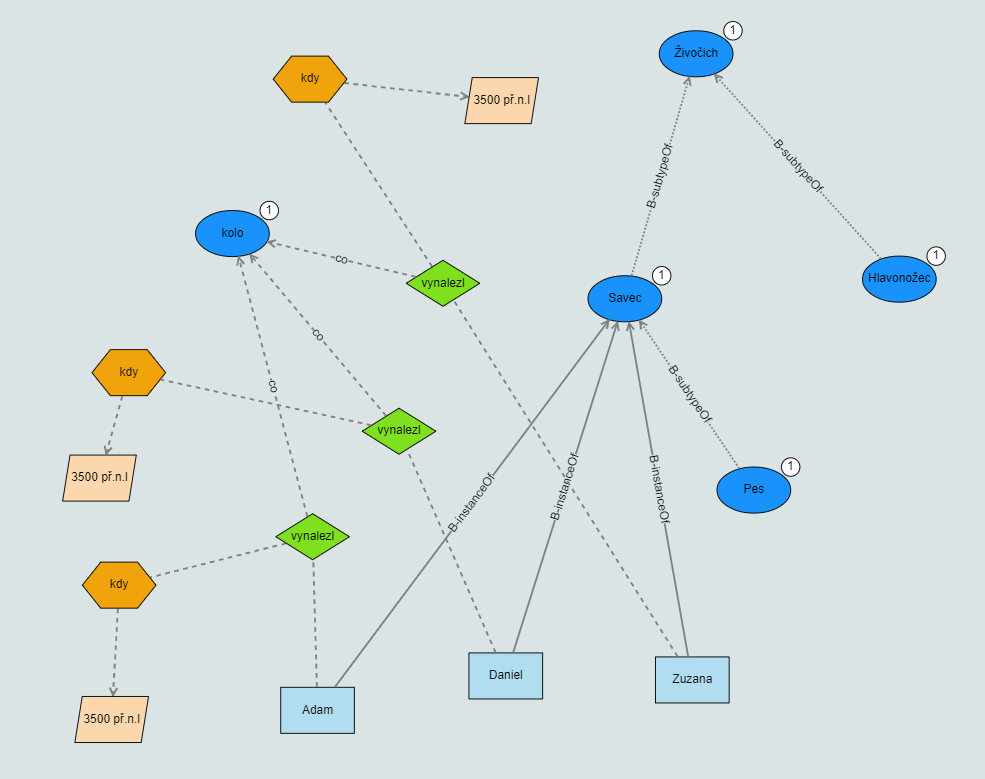
* + - Slučování podle cíle: Zachovává třídy, které jsou nalezeny v cílovém modelu
    - Podle zdroje: Zachovává třídy nalezené v původním modelu.
* Expertní slučování

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 navrhnout 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.

## Návrh

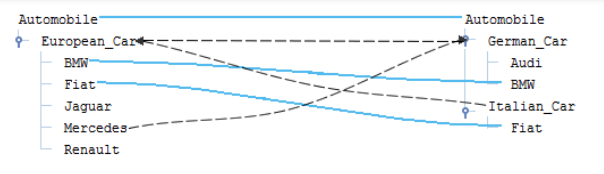
V kapitole věnující se slučování je popsán případ, kdy dochází k mazání uzlů v hierarchii modelující vztah muže a ženy k obecné třídě živočich. Konečným výsledkem tohoto slučování byla situace, ve které tři instance Adam, Daniel a Zuzana zdědily vztah popisující vynález kola.



Již v teoretické kapitole bylo zmíněno riziko tohoto postupu. Při převodu OBM z PURO Modeleru do jazyku OWL dochází totiž k naprostému zahození instancí tříd. Z tohoto důvodu se pak všechny atributy a relace přenášejí na třídu, ze které instance vychází. V OWL by bylo pak popsáno, že savec vynalezl kolo, což je do určité míry v pořádku. Přenosem tohoto vztahu na třídu „pes“ však vzniká nežádoucí situace.

Instance třídy 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 třídy, na níž se instance odkazuje, došlo by při převodu do OWL k přenosu těchto vztahů na jakoukoli nadřazenou třídu, jejíž instancí se nyní uzel stal. Lze očekávat, že každá instance třídy 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 podtřídy často popisují fakta nezbytná k následnému využití modelu. Příkladem může být obrázek, který v předchozích kapitolách sloužil k popisu situace slučování ontologie s automobily.



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 podtříd, jelikož v cílovém modelu existují. Zbytek uzlů by se převedl přímo pod třídu „automobil“. Z příkladu je však zřejmé, že zachování uzlů 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 uzlů tříd při asymetrickém slučování:

* Nelze smazat třídu která má přímo napojené instance a nemá nadřazenou třídu. Toto by vedlo ke ztrátě informace při převodu do OWL, jelikož instance se nepřenášejí.
* Při smazání třídy s přímo napojenými instancemi by se jakýkoli vztah instance s okolím přenášel na nadtřídy odstraněného uzlu. Toto může vést k mnoha nelogickým situacím. Není tedy vůbec možné smazat třídy s instancemi.
* Není možné smazat třídu která by vztahy přenášela přímo na svoji instanci. Pokud má třída přímé uzly instancí a zároveň má sama atribut, či relaci, není možné ji smazat.
* Nejnižší třídy nemusí být vhodné smazat, jelikož popisují detailní a pravděpodobně žádoucí informace.

Rozborem těchto omezení lze vyvodit, že nejnižší třídy v hierarchii většinou není vhodné smazat. Za určitých situací to vůbec není možné (instance by neměla třídu), nebo by byly při převodu do OWL vztahy instancí nelogicky přeneseny na nadřazenou třídu. Případně odstranění nemusí být žádoucí, jelikož by model přišel o vyjadřovací schopnost a násobení vztahů na podtřídy by vytvářelo nepřehledný model. Ořezáním výše uvedených podmínek ve své podstatě vyplývá, že smazat nejnižší třídu 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 třídy smazat či ne.

Podobnou situaci lze pak sledovat i u mazání tříd, které mají spoustu podtříd. Ty by totiž dědily vlastnosti své odstraňované nadtřídy, což by mohlo vést k nepřehlednému modelu. Uživatel by měl mít opět možnost rozhodnout například o tom, kolik musí mít třída podtříd, aby nedošlo k jejímu smazání.

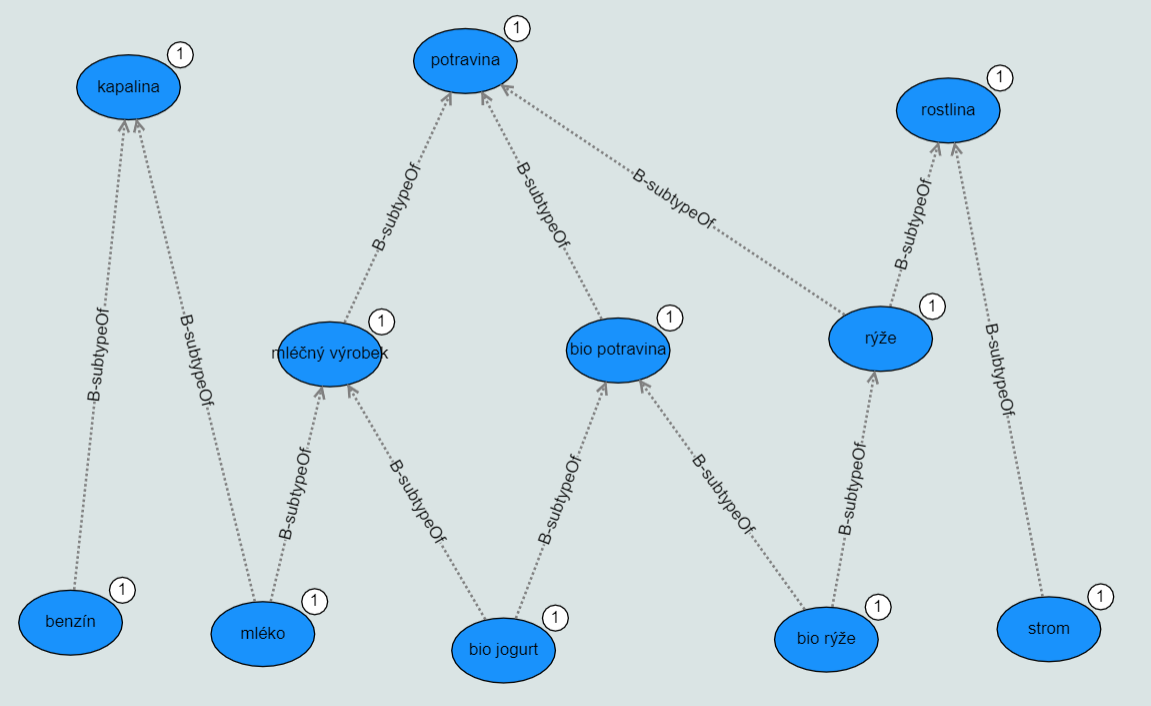
Následující návrh řešení se věnuje slučování podle cíle, který je však možné využít i pro slučování podle zdroje pouhým vyměněním pořadí modelů.

### Nalezení uzlů v hierarchii tříd

Prvním krokem je průchod všemi uzly tříd ve zdrojovém modelu, které jsou zahrnuty do hierarchie podtříd. 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 tříd, které jsou instancí, jejichž tvorba je v jazyce PURO možná.

Dále je nutné vyvodit které uzly jsou třídami na vrcholu grafu. Z podstaty konceptu podtříd vyplývá určitá dědičnost a hierarchie, lze tedy vyloučit cykly. Díky tomu je pak možné usuzovat, že v určitých podmnožinách bude mít tento graf tříd pro určité množství uzlů jediný vrchol, nejedná se však o orientovaný graf, protože listy grafu mohou mít k vrcholu více cest. Vyplývá to z volnosti při určování podtříd. Jakýkoli uzel totiž může být podtřídou více uzlů, pokud to nepovede k cyklickému grafu.

Výsledným grafem může být například následující hierarchie:



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 správně. Je zde zároveň patrná výše popsaná situace, kdy uzel náleží k více nadtřídám. Například „mléko“ je zároveň mléčný výrobek i kapalina. Proto je možné se z něj dostat ke dvěma vrcholovým uzlům.

Pro všechny vrcholové uzly tříd ale platí, že do nich nevstupuje žádná hrana grafu typu „b-subtypeOf“. Toto je tedy nutnou podmínkou 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 šířky. 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:

* nadtřída: potravina
* podtřída: 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.

### Hledání shodných uzlů

Dalším krokem je hledání shodných uzlů v cílovém grafu. Toto platí pro každý uzel, který je zahrnut do hierarchie. Pokud tedy uzel má nadtřídu, či podtřídu, je zahrnut do porovnávání. Pro každý vhodný uzel je pak procházen celý cílový graf. Pokud je nalezen shodný uzel na základě textového řetězce, sémantiky a typu, je uzlu ze zdrojového grafu tato informace uložena.

### Hledání kandidátů na odstranění

Při asymetrickém slučování nelze jednoduše vymazat všechny třídy, kterým nebyla nalezena ekvivalentní třída v původním grafu. U každého uzlu je nutné vyhodnotit následující:

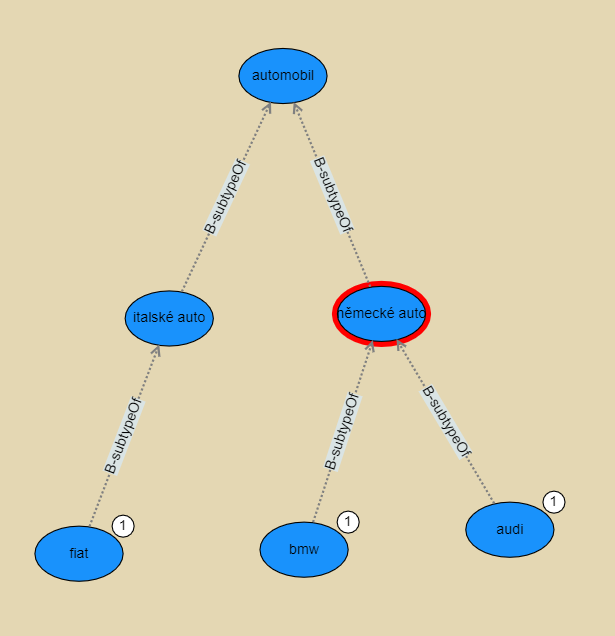
* Pokud má třída instance, není možné ji smazat. Vztahy instancí by se při převodu do OWL přímo přenesly na třídu, která by zůstala. Toto by mohlo vést k nelogickým situacím.
* Pokud je třída nalezena i v původním grafu, není možné ji smazat
* Pokud je třída instancí jiné třídy, není vhodné ji mazat. Vztahy by byly defacto ztraceny, jelikož by nemohly být přeneseny na třídu nad sebou.
* Uživateli by měla být dána možnost odstranit nejnižší třídy, které samy nemají instance a nebyly nalezeny v původním grafu. Tyto třídy ale mohou být napojeny na důležité relace, či atributy. Proto by mělo být rozhodnutí přenecháno na uživateli.

Pokud třída 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 každou třídu může existovat určitý počet podtříd, které budou přímo dědit, pokud bude třída smazána. Pokud uživatel nastaví práh, nesmaže se třída, u které by počet přímo dědících podtříd překonal tento práh.

### Průchod cestami nahoru

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



Pokud by došlo ke smazání třídy „německé auto“, je nutné třidě „automobil“ uložit uzel „italské auto“, „bmw“ a „audi“ jako uzly, které budou dědit za předpokladu, že dojde i ke smazání uzlu „automobil“.

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

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

### Odstraňování tříd

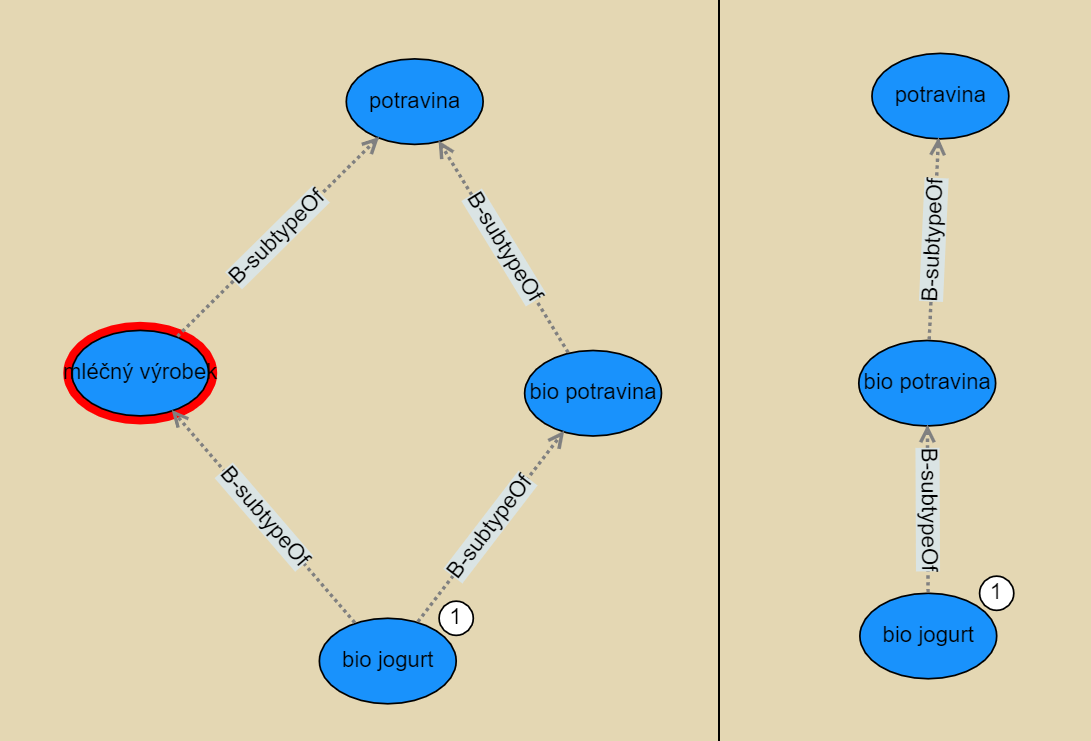
Pro každý uzel třídy, který byl určen ke smazání je nutné provést následující:

1. Přenést propojení s nadtřídou na každý uzel podtřídy.
2. Distribuovat atributy a relace mezi všechny podtřídy.

V prvním případě je nutné projít každý uzel podtřídy a přepojit je na každou nadtřídu, kterou odstraňovaný uzel má.

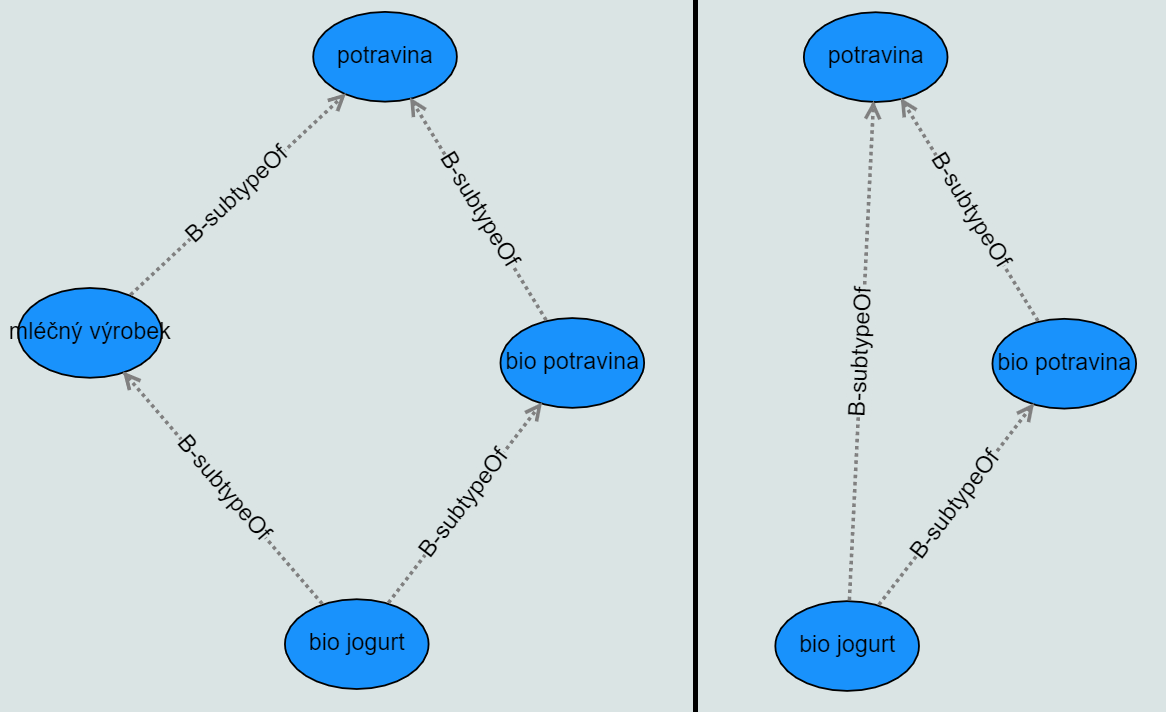
Dále je nutné vzít přímé okolí odstraňovaného uzlu, tedy všechny atributy a relace s okolím. Tyto relace je poté třeba rozkopírovat pro každou podtřídu a přepojit na ně hrany grafu, které původně příslušely odstraňovanému uzlu.

Uzel pak již může být odstraněn. Po odstranění všech uzlů je pak nutné znovu vyhodnotit cesty pro uzly a jejich hierarchii. Toto je nutné zejména kvůli redundanci vztahů, která může vzniknout odstraněním tříd a s tím souvisejícím přenosem vztahů s nadtřídami na podtřídy.



Na obrázku je zjednodušený model z příkladu výše. Jedná se o situaci před sloučením s cílovým grafem vpravo. Červeně je označen uzel, který nebyl v cílovém grafu nalezen a bude proto odstraněn. Při odstranění uzlů je vždy přenesen vztah s nadtřídou na uzel pod nimi.

Na následujícím obrázku je proces odstranění uzlu „mléčný výrobek“.



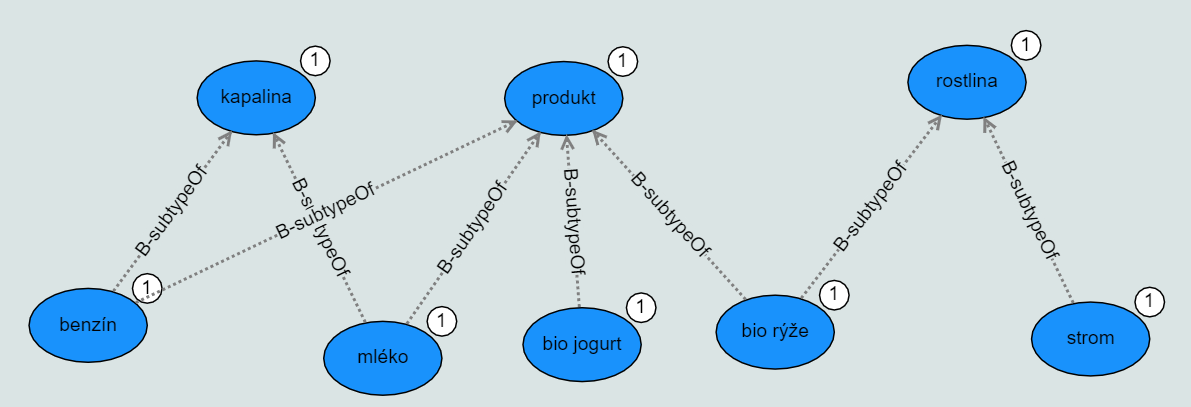
Na obrázku již není zdrojový a cílový model, ale popis situace před a po smazání uzlu „mléčný výrobek“

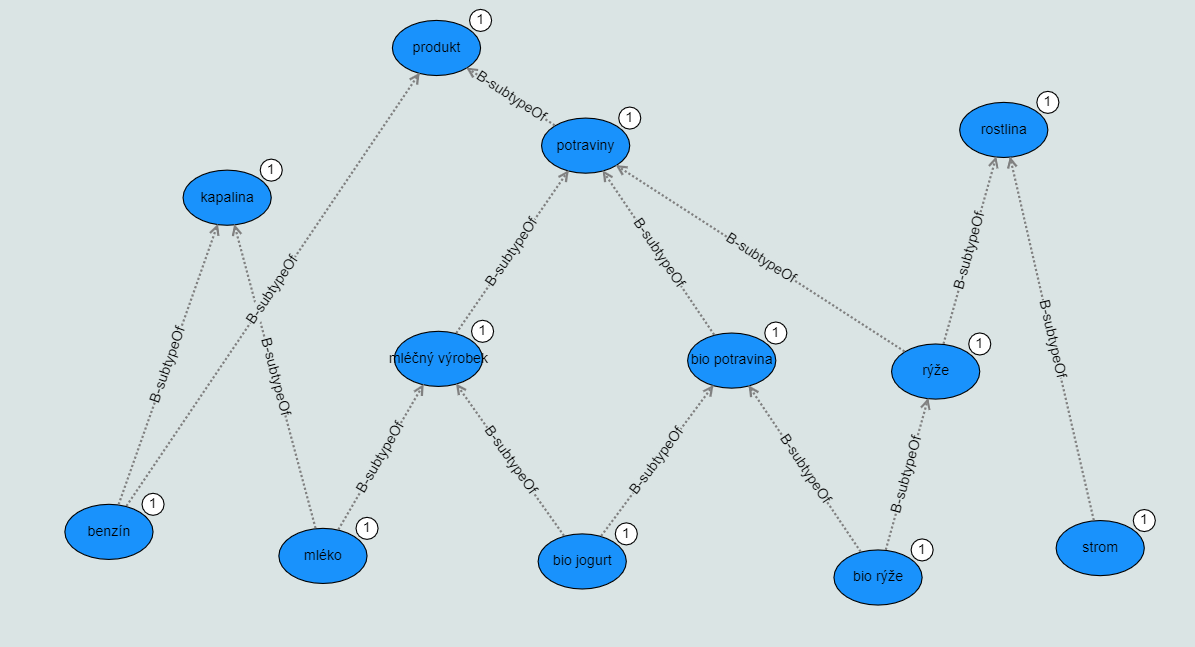
Uzel „mléčný výrobek“ byl odstraněn a jeho příslušnost k nadtřídě „potravina“ byla přenesena přímo na uzel „bio jogurt“. Je zjevné, že bio jogurt je zároveň bio potravinou i potravinou. Přímý vztah s potravinou však není nutný, protože vyplývá z příslušnosti ke tříde bio potravina.

Je tedy žádoucí tyto redundantní vztahy odstraňovat pro přehlednost modelu. Z konečné situace popsané na obrázku výše vyplývá, že je vhodné odstraňovat ty vztahy s nadtřídou, které vedou nejkratší cestou.

### Odstraňování redundantních vztahů

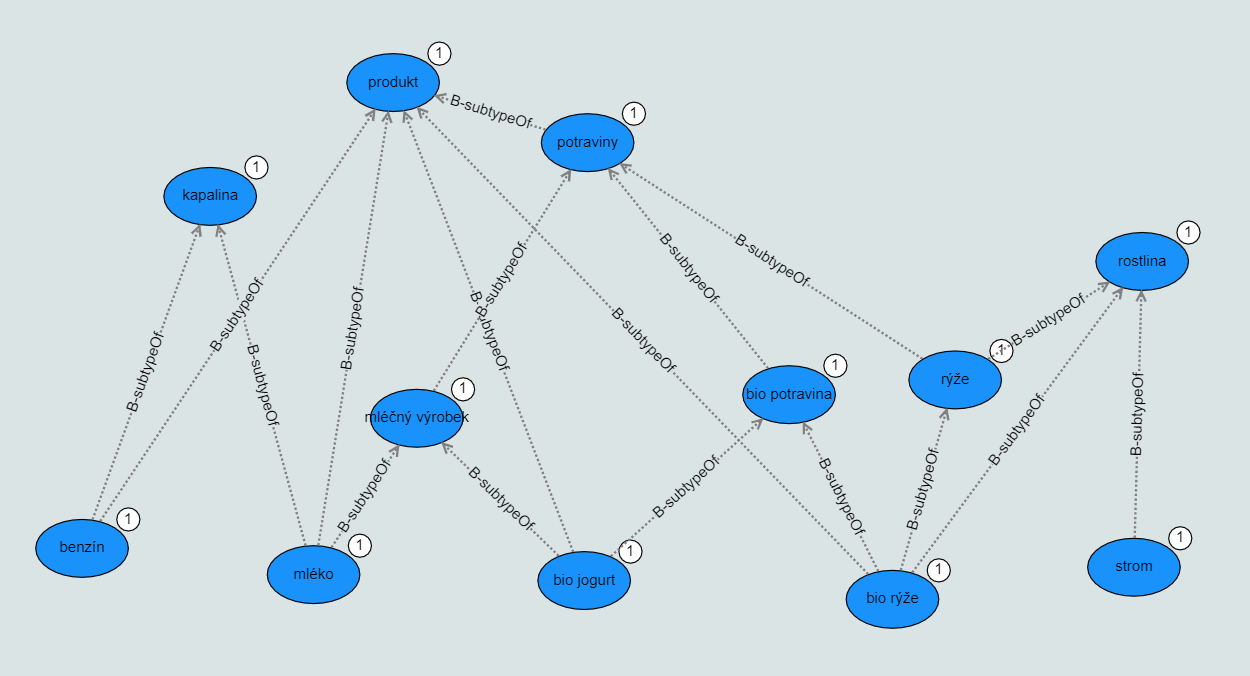
Pro názornou ilustraci odstraňování redundantních vztahů v hierarchii byl vytvořen ukázkový model, který je zjednodušením modelu, se kterým bude slučován:





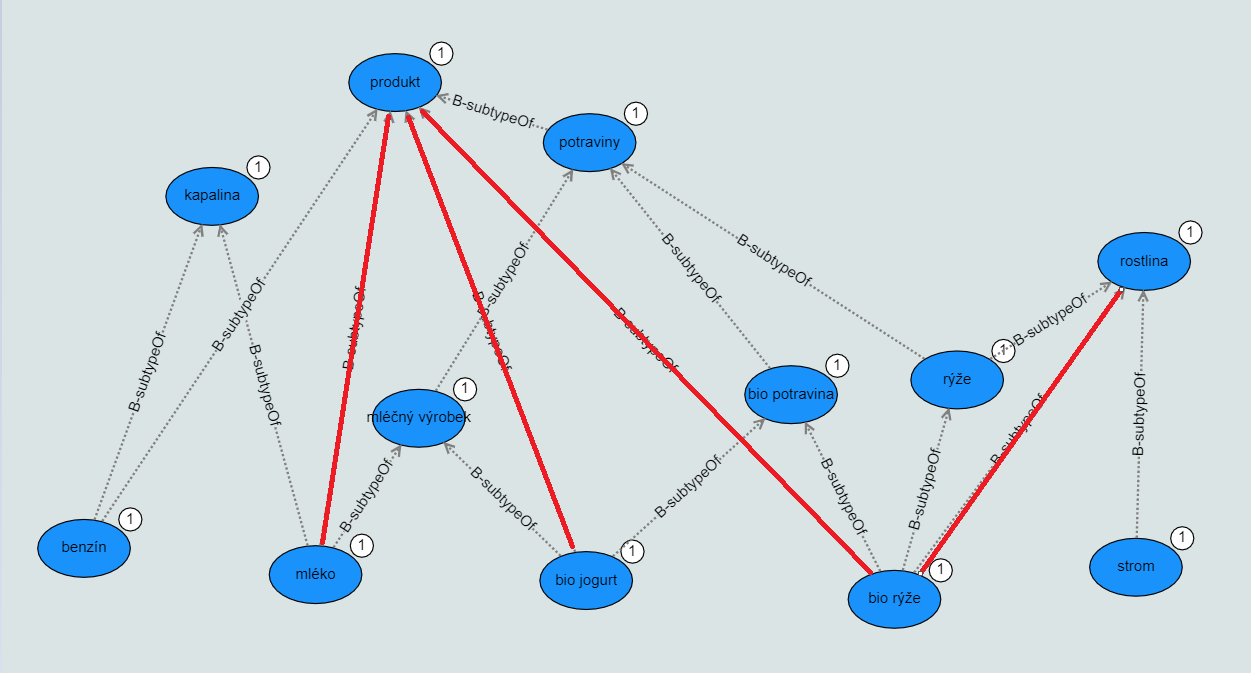
Druhý model, tedy cílový, obsahuje stejné uzly, jako výchozí model. Je však zřejmé, že některé třídy v prvním modelu spolu souvisejí přímo, ačkoli v cílovém modelu mají mezikrok. Nedojde tedy k odstranění žádných tříd, ale některé hrany grafu budou duplikovány.

Výsledek sloučení těchto dvou modelů je následující:



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

Na obrázku níže jsou vyznačené hrany grafu, které je možné odstranit.



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 každé své nejvyšší třídě. Mohla by nastat situace, kdy uzel cestu vede přes průchozí třídy více způsoby, na tyto situace je také nutno pamatovat.

Pro každý uzel se tedy opět prochází seznam cest k nejvyšší třídě, vytvořený v kapitole věnující se hierarchii tříd. 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 třídy „produkt“ na první pohled duplikátně přes třídy „bio potravina“ a „rýže“, nelze jednu z nich smazat. Každá cesta totiž vyjadřuje unikátní vztah s nadřazenými třídami. Tyto třídy by totiž mohly mít relace či atributy, které by pak díky dědičnosti přenášely na třídy pod sebou. Ačkoli tedy tyto cesty vedou ke stejné nejvyšší třídě, vedou k ní jiným způsobem a vytváří tak specifické vztahy pro nižší třídy.

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 tříd a přenos jejich vztahů a hierarchie na podtřídy je výsledkem graf, který již má každou třídu buďto shodnou s cílovým grafem, nebo již není žádoucí žádný z uzlů třídy odebírat. Poté již lze oba grafy slučovat stejným postupem, jako je tomu při plném slučování.

## Implementace

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

1. Pro každý uzel zjisti, jestli z něj, či do něj vede link typu „b-subtypeOf“. Pokud link b-subtypeOf vede pouze do uzlu, znamená to, že patří k nejvyšší třídě. Pokud patří, pokračuj:
   1. Vytvoř prázdné pole „strom“, které bude obsahovat uzly patřící v hierarchii pod tento uzel nejvyšší třídy. Přidej do něj právě procházený uzel nejvyšší třídy.
   2. Vytvoř pole „uzlyÚrovně“ obsahující uzly, které se nacházejí v určité úrovni, která se právě prochází. Přidej do něj právě procházený uzel nejvyšší třídy.
   3. Vytvoř proměnnou „úroveň“ vyjadřující číselně od nejvyššího uzlu (0) postupně číslo každé úrovně. Přiřaď jí číslo -1
   4. Pokud se délka pole „uzlyÚrovně“ nerovná nule, prováděj následující:
      1. Vytvoř pole „budoucíÚroveň“.
      2. Přičti k „úroveň“ jedničku.
      3. Pro každý uzel „výchozí“ v poli „uzlyÚrovně“ proveď 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 nadtřídou uzlu „výchozí“) proveď následující pouze pokud je uzel „související“ nalezen v poli „strom“:
            1. Uzlu „výchozí“ přidej do jeho pole nadřazených uzlů uzel „související“
            2. 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 podtřídou uzlu „výchozí“) proveď následující:
            1. Vlož „související“ do pole „strom“
            2. Vlož „související“ do pole „budoucíÚroveň“
            3. Vlož uzlu „výchozí“ do jeho vlastního pole „poduzly“ uzel „související“ a zároveň informaci o posledním nadřazeném uzlu z pole nadřazených uzlů.
      4. Nahraď pole „uzlyÚrovně“ polem „budoucíÚroveň“

Tento algoritmus v každému uzlu nejvyšší třídy vytvoří pomyslný strom. V tomto stromě pak prohledává do šířky 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 nejvyšší třídy.

Následujícím krokem je určení uzlů tříd k odstranění:

1. Pro každý uzel z hierarchie tříd proveď následující:
   1. Pokud je uzel nalezen v původním grafu, ulož do jeho proměnné „kandidát“ hodnotu „false“
   2. Pokud je uzel instancí, nebo má sám instance, ulož do jeho proměnné „kandidát“ hodnotu „false“
   3. Pokud uzel nemá podtřídy a ani instance, kontroluj, zdali uživatel chce i nejnižší třídy bez instancí zachovat:
      1. Pokud ano ulož do jeho proměnné „kandidát“ hodnotu „false“.
      2. Jinak ulož „true“

Následuje algoritmus vyhodnocující počet závislých podtříd:

1. Pro každou cestu každého nejnižšího uzlu procházej od nejnižší třídy po nejvyšší:
   1. Pokud má uzel po cestě „uzelCesty“ hodnotu kandidát = true, proveď následující:
      1. 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.
      2. Pro každou podtřídu uzlu uzelCesty zkontroluj hodnotu kandidát. Pokud je true, přidej všechny uzly z pole závislé uzly náležící podtřídě do pole závislé uzly náležící uzlu uzelCesty

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

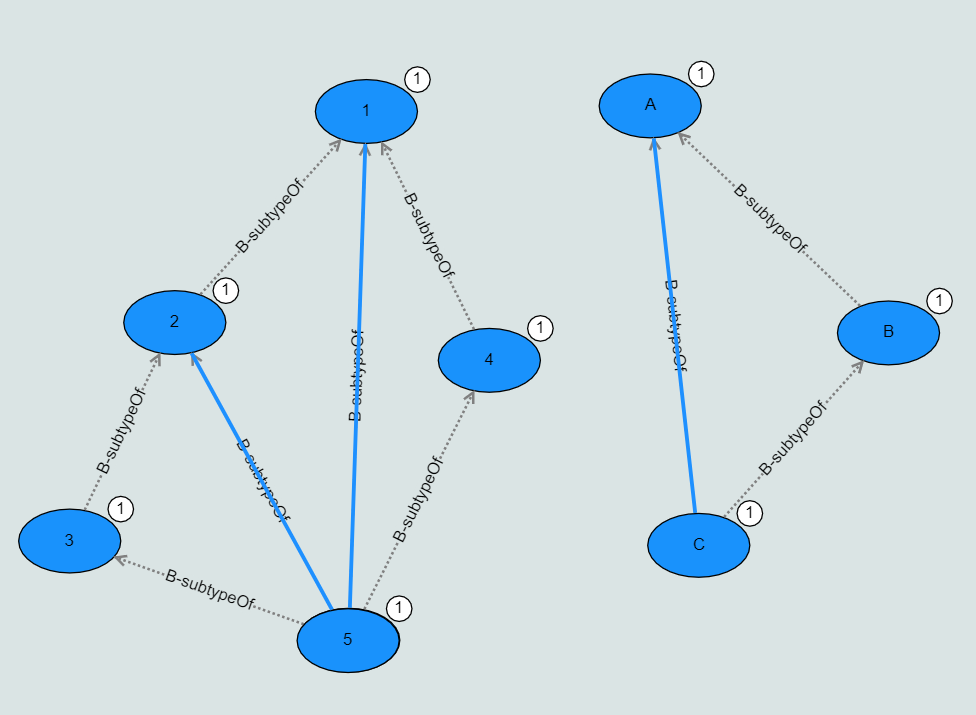
Pro odstranění libovolného uzlu třídy je nutné následující:

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

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

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:
   1. Pro každý uzel v poli *okolí* vyhledej připojené uzly.
   2. Pokud tyto připojené uzly náleží k jiné kategorii než třída, či objekt, a pokud ještě samy nejsou zahrnuté v poli *okolí*, přidej je do tohoto pole.
3. Odstraň z okolí odstraňovanýUzel
4. Pro každou podtřídu potomek odstraňovaného uzlu odstraňovanýUzel proveď následující:
   1. Zkopíruj pole okolí jako dočasnéOkolí bez referencí na původní uzly v poli.
   2. Pro každý uzel dočasnéOkolí získej všechny linky, které jsou na něj napojeny.
   3. Uzly i linky přidej do objektu dočasnýGraf.
   4. Přeindexuj dočasnýGraf podle algoritmu z kapitoly 7.1.3. Důvodem je, aby po přidání tohoto dočasného grafu do původního grafu nedošlo ke kolizím identifikátorů.
   5. Pro každý link z dočasného grafu proveď následující:
      1. Pokud se id příchozího nebo odchozího uzlu rovná id uzlu odstraňovanýUzel, nahraď ho id uzlu potomek.
   6. Přidej dočasnýGraf do původního grafu.
5. Odstraň uzel odstraňovanýUzel

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



Pro ilustraci slouží obrázek výše. 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 stejnému nejvyššímu uzlu jako ostatní cesty, ale uzel 5 může od uzlu 4 dědit důležité vlastnosti.