EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

Evidenčné číslo: 103004/I/2019/36069387448905476

Ontologické modelovanie pre tvorbu študijných programov na univerzitách

Diplomová práca

2019 **Bc. Patrik Balog**

EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

Ontologické modelovanie pre tvorbu študijných programov na univerzitách

Diplomová práca

Študijný program: Informačný manažment

Študijný odbor: Kvantitatívne metódy v ekonómii

Školiace pracovisko: Katedra aplikovanej informatiky

Vedúci záverečnej práce: RNDr. Eva Rakovská, PhD.

Bratislava 2019 **Bc. Patrik Balog**

Čestné vyhlásenie

Čestne vyhlasujem, že záverečnú prácu som vypracoval samostatne a že som uviedol všetku použitú literatúru.

Dátum: **.....**

(Podpis študenta)

Poďakovanie

Touto cestou by som chcel poďakovať vedúcej záverečnej práce RNDr. Eve Rakovskej, PhD. za odborné vedenie, ochotu, cenné rady a pripomienky, ktoré mi pomohli pri vypracovávaní tejto záverečnej práce.

Abstrakt

BALOG, Patrik: Ontologické modelovanie pre tvorbu študijných programov na univerzitách. – Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky; Katedra aplikovanej informatiky. - Vedúci záverečnej práce: RNDr. Eva Rakovská, PhD. – Bratislava: FHI, 2019, 65 s.

Cieľom záverečnej (diplomovej)práce je nájsť vhodný nástroj na modelovanie ontológie pre vybraný predmet štúdia na FHI, vytvoriť ontologický model jadra predmetu a ukázať jeho modifikovateľnosť. Práca je rozdelená do troch kapitol a obsahuje 38 obrázkov. Prvá kapitola sa zaoberá vymedzením základných teoretických pojmov v oblasti znalostného inžinierstva. Druhá kapitola sa venuje vymedzeniu cieľa, ktorý sme rozdelili na menšie čiastkové ciele a výberu metodiky použitej v práci. Tretia kapitola sa venuje popisu vývojového prostredia Protégé, ontologickému modelovaniu predmetu Umelá inteligencia, analýze jednotlivých algoritmov a nakoniec vizualizácii ontologického modelu. Výsledkom je vytvorená ontológia v prostredí Protégé, na ktorej demonštrovaná automatizovaná klasifikácia a poukázali sme na možnosť zlepšenia tvorby študijných predmetov s pomocou ontológií

Kľúčové slová:
Ontologické modelovanie, Doménová ontológia, Sémantický web, Modelovanie študijného programu

Abstract

BALOG, Patrik: Ontological Modelling for creating study programs at Universities. – University of Economics in Bratislava. Faculty of business informatics; Department of applied informatics. – Thesis supervisor: RNDr. Eva Rakovská, PhD. – Bratislava: FHI, 2019, 65 p.

The aim of the (diploma) thesis is to find a suitable tool for ontological modelling for the chosen subject of study at FHI, create ontological model of the core subject and show its modifiability. The thesis is divided into three chapters and include 38 pictures. The first chapter deals with a basic theoretical aspects of knowledge engineering. The second chapter is about defining main goal, which is divided into partial goals and selecting of the used methodology. The third chaper is focused on decsription of ontology editor Protégé, ontological modeling of study subject Artificial intelligence, analysis of individual algorithms and at the end is focused on ontology visualization.
The result is created ontology in Protégé enviroment, on which is demonstrated automatical classification and we refer to possibility to improve process of creation of subject with using ontology.

Key words:
Ontology modeling, Domain ontology, Semantic web, Modeling of study program

Obsah

Úvod

V dnešnej dobe je potrebné aby vysoká škola poskytovala kvalitné vzdelanie a neustále pracovala na zvyšovaní tejto kvality. Preto je potrebné, aby jej študijné programy boli zložené zo zmysluplných vyučovaných predmetov a tieto predmety na seba v istej miere nadväzovali.

Každý z týchto predmetov obsahuje svoju štruktúru učiva a je zrejmé, že niektoré vyučované témy súvisia s inými v rámci predmetu, ale aj s témami vyučovanými v iných predmetoch. Preto je dôležité rozhodnúť kedy v rámci študijného programu zasadiť predmet do výučby a taktiež kedy vyučovanú tému zasadiť do výučby v rámci predmetu. Ak sú študijné predmety a programy správne namodelované, tak výučba je efektívnejšia a študenti dosahujú lepšie výsledky, pretože sa predchádza prípadom, kedy študent ešte nemá potrebné vedomosti na absolvovanie predmetu, ktoré by inak získal až neskôr v rámci štúdia.

Jedným zo spôsobov ako zachytiť vzájomné vzťahy medzi vyučovanými témami v rámci nielen vyučovaného predmetu, ale aj za jeho hranicami v rámci študijného programu, ale kľudne aj v rámci univerzity je vytvorenie ontologického modelu.

V tejto diplomovej práci sa zaoberáme vytvorením ontologického modelu v prostredí Protégé. Modelujeme časť predmetu Umelá inteligencia a to oblasť Prehľadávania stavového priestoru, kde vytvárame modely, ktoré zachytávajú klasifikačné vzťahy medzi algoritmami.

V prvej časti práce sa snažíme popísať, čo je to ontológia, aké jazyky sa používajú pri zachytávaní, udržiavaní a prenose poznatkov a metódam vytvárania ontologických modelov.

V praktickej časti sa venujeme mapovaním poznatkov o jednotlivých algoritmoch. Tieto poznatky ďalej evidujeme v systéme Protégé. V práci sú obsiahnuté postupy, ako pracovať s poznatkami v Protégé a následné použitie automatickej klasifikácie nami evidovaných poznatok za pomoci HermiT OWL Reasonera.

1 Súčasný stav riešenej problematiky doma a v zahraničí

1.1 Dáta, informácie a znalosti

Na začiatok je potrebné charakterizovať význam slova znalosť. Je potrebné pochopiť, čo sú dáta, informácie a znalosti.

Dáta môžeme chápať, ako fakty a čísla, ktoré nie sú žiadnym spôsobom interpretované a neposkytujú ďalšie informácie o vzore, kontexte a podobne. Podľa Thierraufovej definície to sú neštruktúrované skutočnosti a čísla, ktoré nedávajú žiaden ďalší význam.[1]

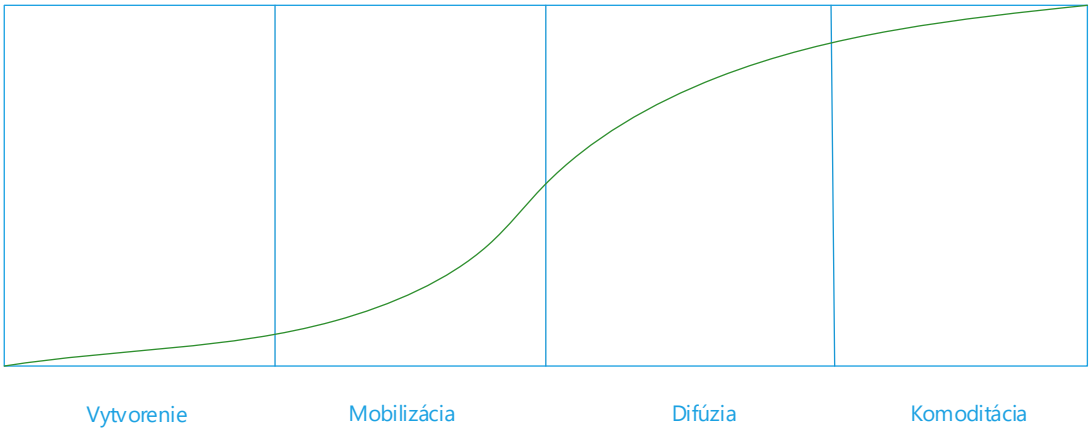
Aby sa dáta stali informáciami, tak musia byť kontextualizované, kategorizované, vypočítané. Informácie tak vytvárajú širší obraz. Sú to dáta s významom a účelom. Informácie sú v podstate odpoveďami na otázky, ktoré začínajú slovami kto, čo, kedy, koľko,... [1]

Informačné technológie prinášajú možnosti premeny dát na informácie, špeciálne vo veľkých firmách, kde sa spracováva obrovské množstvo údajov na rôznych oddeleniach, pričom človek je potrebný najmä pri konceptualizácii týchto údajov.

Znalosti môžeme popísať ako aplikovanie dát a informácií v praxi. Poskytujú odpovede na otázky typu Ako a Prečo.

1.2 Životný cyklus znalostí

Životný cyklus vedomostí môžeme reprezentovať ako krivku, ktorá je zobrazená na obrázku nižšie. Horizontálna os zobrazuje uplynutý čas a vertikálna os zobrazuje množstvo jedincov, ktorí majú prístup k vedomosti.



Obrázok 1: Životný cyklus znalostí [2]

Prvým krokom v cykle je vytvorenie znalosti, čo nie je nič viac, ako myšlienka jedinca, alebo skupiny jedincov. V tejto fáze je pravdepodobné, že ani samotný jedinec úplne nerozumie tejto vedomosti. Veľa z myšlienok sa ukážu ako neužitočné, alebo nevzbudia u jedinca záujem, ale podstatné sú tie ostatné, ktoré budú zrozumiteľnejšie formulované a lepšie chápané ich tvorcom. [2]

Následne sa myšlienka mobilizáciou pretvára na konkrétnejšiu a jej hodnota je stanovená procesom testovania a validácie. V tejto fáze už konkrétni jedinci dokážu v istej miere kodifikovať tiché vedomosti. Inak povedané. Fáza mobilizácie predstavuje kodifikáciu z dôvodu sprístupnenia vedomosti ďalším jedincom. [2]

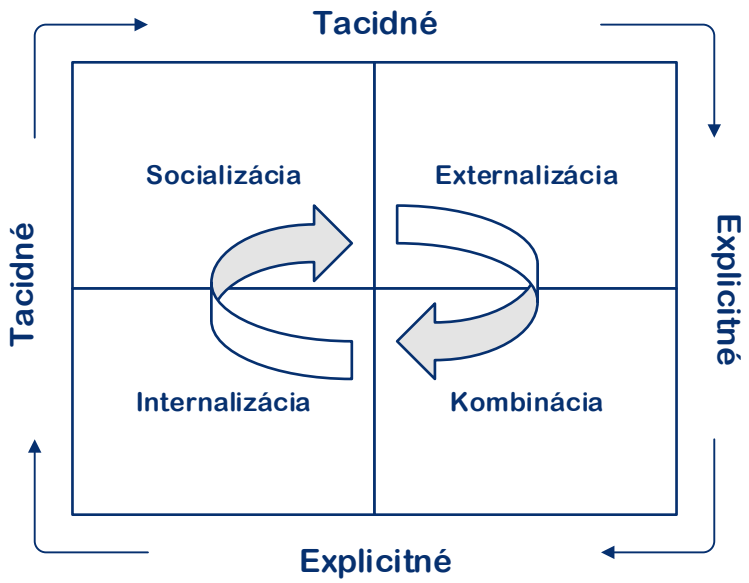
Ďalšia fáza sa nazýva difúzia. Znalosti sa stávajú pochopiteľné na relevantnom trhu. V tejto fáze môže každý získať tieto poznatky, ale stále existujú náklady na internalizáciu týchto nových vedomostí. [2]

Nakoniec prichádza etapa komoditácie. Vedomosti sa menia na bežné poznatky. Takéto poznatky sú bezplatné, inak povedané, sú bežne dostupné v knihách, na internete, vyučované vo vzdelávacích zariadeniach. [2]

1.2.1 SECI model

SECI model bol predstavený Nonakom a Takeuchim v roku 1996, ktorý predstavuje teóriu vytvárania poznatkov a ich prenosu. Predstavuje štyri spôsoby ako môžeme kombinovať a konvertovať typy vedomostí, ktoré ukazujú, ako sú vedomosti v organizáciách vytvárané a zdieľané. Model je založený na explicitných a tacitných vedomostiach. Tacitné vedomosti sú také vedomosti, ktoré nie sú vyjadrené. Sú to v podstate

vedomosti, ktoré máme v hlave. Explicitné vedomosti sú vedomosti, ktoré sú istou formou zachytené napríklad na papieri, alebo elektronicky.[3]



Obrázok 2: SECI model [3]

Prvým spôsobom je socializácia. Predstavuje zdieľanie tichých vedomostí prostredníctvom pozorovania, napodobňovania, praxe a účasťou vo formálnych a neformálnych komunitách. Tento proces zvyčajne prebieha vo fyzickom, alebo virtuálnom priestore, v ktorom prebieha sociálna interakcia danej komunity.

Ďalším spôsobom je externalizácia. Je to proces pretvárania tichých vedomostí na explicitné. Je považovaná za náročný, ale dôležitý konverzný mechanizmus. Tacitný poznatok je istým spôsobom kodifikovaný, napríklad zapísaním do dokumentov. Keďže tiché vedomosti je často prakticky nemožné kodifikovať, rozsah tohto spôsobu konverzie je diskutabilný.[3]

Tretím spôsobom je kombinácia. Explicitné vedomosti vnútri, ale aj mimo organizácie sme schopní zhromažďovať, kombinovať, upravovať a spracovávať, aby sme vytvorili nové zložitejšie a systematickejšie explicitné vedomosti a tieto sa potom ďalej rozširujú v rámci organizácie.[4]

Posledným spôsobom je internalizácia. Týmto spôsobom sa explicitné vedomosti, ktoré sú zdieľané aj mimo organizácie, konvertujú jednotlivcami na tiché vedomosti. Túto fázu môžeme tiež nazvať ako prax, kedy sú poznatky aplikované v konkrétnych situáciách a stávajú sa novými rutinami.[3]

Tvorenie poznatkov je kontinuálna činnosť interakcií medzi tacitnými a explicitnými poznatkami. Tieto štyri spôsoby interagujú v špirále vytvárania vedomostí. Táto špirála rastie spolu s organizačnou úrovňou a je impulzom k vytváraniu nových takýchto špirál.

1.3 Znalostný manažment

Riadenie poznatkov a ich efektivita sa v súčasnosti považujú za silné aktíva v konkurencieschopnosti bez ohľadu na trhový segment. Je to disciplína o tom, ako ľudia, procesy a technológie spoločne fungujú, aby poskytli správne informácie v správny čas, na správnom mieste, správnemu publiku a v správnej forme.

Znalostný manažment funguje takým spôsobom, aby sa dali individuálne a skupinové poznatky opakovane použiť. Slúži na konverziu dát a informácií pochádzajúcich z rôznych zdrojov na vedomosti.

Rozlišujeme dva hlavné druhy vedomostí: Explicitné a tacitné vedomosti. Explicitné znalosti sú formálne a systematické. Pre tento typ vedomostí je špecifické, že sú ľahko identifikovateľné, uchovateľné a zdieľateľné. Sú reprezentované formou dokumentov, obrazovo-zvukového obsahu a podobne. Ak vyhľadávame text na internete, tak poznatky, ktoré nájdeme sú vyjadrené explicitnou formou. Kľúčovým prvkom explicitných znalostí je, že sú jednoducho vyjadriteľné a zrozumiteľné.[5]

Druhým typom znalostí sú tacitné znalosti. Sú to také znalosti, ktoré sú získané osobnou skúsenosťou. Tieto znalosti sú závislé na jednotlivcovi a sú kombináciou skúseností a intuície. Ich reprezentácia je náročná, ale nie nemožná.

Tretím typom znalostí sú implicitné znalosti. Sú to znalosti uchovávané v nás, ale na rozdiel od tacitných znalostí sme tieto schopní zaznamenať externe.

1.4 Ontológia

Vývoj ontológií v zmysle špecifikácie pojmov a ich vzťahov v konkrétnej oblasti vykonávajú predovšetkým doménoví experti. V súčasnosti sú ontológie bežným prvkom world wide webu. Ontológie na webe majú rôznu úroveň. Bývajú používané pri kategorizácii rôznych webových stránok, ale aj v rámci samostatných webových stránok, napríklad pre kategorizáciu produktov.

Ontológia definuje bežný slovník pre zdieľanie informácií v doméne. Obsahuje definície základných pojmov a vzťahy medzi nimi. [6]

Dôvodmi pre vytvorenie ontológie môžu byť:

- Zdieľanie spoločného porozumenia štruktúry informácií medzi ľuďmi a softvérovými agentami,
- Umožnenie opätovného použitia znalostí v doménach,
- Vyjadrenie doménových predpokladov,
- Rozdelenie doménových a prevádzkových znalostí,
- Analýza doménových znalostí.

Jedným z najbežnejších cieľov vytvárania ontológií je zdieľanie spoločného významu štruktúry informácií medzi ľuďmi alebo softvérovými agentami. V praxi to môže znamenať, že ak viaceré internetové stránky zdieľajú a publikujú tú istú ontológiu, tak počítačové agenty môžu získavať a zhromažďovať informácie z týchto stránok. Takto získané informácie bývajú používané ako odpovede na dotazy od používateľov, alebo tiež ako vstupné údaje ďalších aplikácií. [6]

Vytváraním ontológií umožňujeme opätovné využívanie poznatkov v doméne. V princípe to znamená, že ak existuje ontológia, ktorú vieme použiť, tak ju nevytvárame odznova, ale ju použijeme pre svoju doménu. Ak vytvárame veľkú ontológiu, tak môžeme použiť viacero existujúcich parciálnych ontológií, ktoré popisujú jednotlivé časti tejto veľkej ontológie. Rovnako používame všeobecné ontológie, ako je UNSPSC (United Nations Standard Products and Services Code) ontológia a upravujeme ju tak, aby popísala našu doménu. [7]

Vytváraním explicitných predpokladov domény, čo predstavuje základ implementácie, umožňujeme ich jednoduchú zmenu v prípade, že naše vedomosti o doméne sa zmenia. Pokiaľ sú predpoklady o doméne pevne napísané v programovacom jazyku, tak takéto predpoklady sú ťažko dostupné, pochopiteľné a meniteľné pre ľudí ktorí nepoznajú použitý programovací jazyk. Explicitné špecifikácie doménových poznatkov sú užitočné pre nových používateľov, ktorí sa potrebujú naučiť význam výrazov v doméne.

Dôležitým dôvodom vzniku ontológií je oddelenie doménových znalostí od prevádzkových. Takto môžeme napríklad použiť rovnaký algoritmus na konfiguráciu počítačov a výťahov s tým rozdielom, že raz k nemu pridáme ontológiu komponentov počítača a raz ontológiu komponentov výťahov. [7]

V prípade, kedy disponujeme deklaratívnou špecifikáciou pojmov, tak sme schopní vykonávať analýzu znalostí o doméne. Formálna analýza pojmov je hodnotná , keď sa obe pokúšajú použiť existujúce ontológie a tie ďalej rozširovať.

Cieľom nemusí byť vypracovanie ontológie domény. Jej vytvorenie je podobné pre definovanie údajov a ich štruktúr, ktoré vstupujú do programov. Ontológie a znalostné bázy z nich sú používane v metódach na riešenie problémov, v doménovo nezávislých aplikáciách a softvérových agentoch. [6]

1.4.1 Čo je to ontológia?

V literatúre o umelej inteligencii nájdeme rôzne definície ontológie. Pre nás je ontológia formálnym explicitným opisom konceptov (tried) v konkrétnej oblasti, vlastností konceptov opisujúcich ich znaky a atribúty (sloty) a obmedzenia slotov. Ontológia so setom inštancií jednotlivých tried reprezentuje bázu vedomostí. Hranica, kde končí ontológia a začína báza znalostí je naozaj veľmi tenká.

Väčšina ontológií sa zameriava na triedy. Jednotlivé triedy opisujú koncepty v doméne. Trieda môže obsahovať podtriedy, ktoré obsahujú inštancie, ktoré sú podľa istého kritéria špecifickejšie, ako nadradená trieda. V triedach môžeme vytvárať podtriedy alternatívne podľa kritérií. [8]

Z praktického hľadiska vývoj ontológie znamená:

- Definíciu tried v ontológii,
- usporiadanie týchto tried v taxonomickej hierarchii (vzťah triedy a podtriedy),
- určenie vlastností a ich povolené hodnoty,
- naplnenie slotov jednotlivých inštancií.

Následne sme schopní vytvoriť bázu poznatkov pomocou definície jednotlivých inštancií tried a naplnením ich slotov informáciami.

1.4.2 Jednoduchá metodika znalostného inžinierstva

Pri vývoji ontológií nie je možné hovoriť o správnom spôsobe, alebo metodike. Popisujeme iteratívnu metódu vývoja ontológií. Na začiatok vytvoríme hrubú ontológiu a s použitím iteratívneho procesu, kde túto ontológiu prehodnocujeme a dopĺňaním detailov ju vylepšujeme. Počas tohto procesu sa diskutuje o projektantových rozhodnutiach, o výhodách a nevýhodách a dôsledkov rozdielnych rozhodnutí. [9]

Existuje niekoľko základných pravidiel v oblasti dizajnu ontológií, ktoré môžu vyzerať dogmaticky, ale v mnohých prípadoch sú schopné pomôcť pri navrhovaní. [9]

- Pre modelovanie domény nie je konkrétny správny spôsob. To optimálne riešenie je závisle od našej myslenej aplikácie a od predpokladaných rozšíreniach.
- Modelovanie ontológií je iteratívny proces.
- Triedy v ontológiách môžu byť blízke fyzickým, alebo logickým objektom a vzťahom v našej záujmovej doméne.

Zjednodušene vieme povedať, na aký účel bude ontológia použitá, ako veľmi bude všeobecná, alebo detailná vyvíjaná ontológia. Toto všetko závisí od rozhodnutí počas jej modelovania. Vždy máme niekoľko alternatív rozhodnutí a potrebujeme byť schopní medzi nimi určiť, ktoré z nich je lepšie pre konkrétnu úlohu, rozšíriteľnejšie, udržateľnejšie a intuitívnejšie. Dôležitým obmedzením je model reality a jednotlivé triedy v tejto ontológii majú túto realitu odrážať v čo najväčšej miere. Po vytvorení počiatočnej verzie ontológie ju následne zhodnocujeme a odstraňujeme chyby, čím revidujeme pôvodnú ontológiu. Toto práve predstavuje iteratívne vytváranie ontológie, ktoré zvyčajne prebieha počas celého životného cyklu ontológie. [9]

1.4.3 IDEF5

IDEF5 poskytuje metódu, ktorá je navrhnutá pre vytváranie, modifikáciu a udržiavanie ontológií. Štandardizované postupy, možnosť prezentácie informácií o ontológiách v prirodzenej forme a lepšie kvalitatívne výsledky sa sprístupňujú prostredníctvom IDEF5, čo zároveň redukuje náklady pre tieto činnosti.

Táto metodika v sebe obsahuje základné princípy ontologickej analýzy. Vykonáva sa skúmaním slovnej zásoby použitej pri diskusii o objektoch a procesoch tvoriacich doménu. Vytvárajú sa definície základných pojmov a popisujú sa logické spojitosti medzi týmito pojmami. Výsledkom analýzy je ontológia, ktorá je doménovým slovníkom s množinou definícií alebo axióm, ktoré obmedzujú význam pojmov, aby bola zabezpečená konzistentná interpretácia údajov používajúcich tento slovník. [10]

V každej oblasti nachádzame fenomény, ktoré ľudia rozlišujú ako objekty, asociácie a situácie. S použitím jazykových mechanizmov spájame s fenoménmi deskriptory. V ontologickom kontexte je vzťah jednoznačným deskriptorom, ktorý odkazuje na asociáciu

v reálnom svete.. Termín je definitívny deskriptor, ktorý odkazuje na objekt alebo situáciu, ktorá sa približuje situácii reálneho sveta. [10]

Počas vytvárania ontológie katalogizujeme deskriptory a vytvárame model domény. Z tohto dôvodu vykonávame tri úlohy: [10]

- Katalogizujeme pojmy,
- Zachytávame obmedzenia, ktorými určujeme, ako môžu byť výrazy použité na opisné vyjadrenia v doméne,
- Vytvárame model, ktorý môže generovať dodatočné opisné vyhlásenia.

O tomto modeli potom môžeme vyhlásiť, že reprezentuje sankcionované závery v danej doméne. Tiež vieme povedať, že charakterizuje správanie objektov a asociácií v doméne. Preto ontológiu môžeme prirovnať k slovníku údajov, ktorá ale obsahuje aj gramatiku a model správania sa domény.

Vývoj ontológie metódou IDEF5 sa skladá z piatich činností: [10]

- Organizácie a určením rozsahu – V tomto kroku určujeme účel, hľadisko a kontext projektu vývoja ontológie a prideľujeme úlohy jednotlivým členom projektu.
- Zber údajov – V tomto kroku prebieha zhromažďovanie prvotných údajov, ktoré sú potrebné pri vývoji ontológie.
- Analýza údajov – Tento krok zahŕňa analýzu údajov pre zjednodušenie extrakcie ontológie.
- Začiatočný vývoj ontológie – Tento krok reprezentuje vývojovú činnosť rozvíjania predbežnej ontológie s použitím zozbieraných údajov.
- Zdokonaľovanie a validácia ontológie – V poslednom kroku sa ontológia postupne vylepšuje a jej validáciou ukončujeme vývojový proces.

1.5 Vývoj ontológie v oblasti informatiky

Pri výbere formalizmu reprezentácie poznatkov pre kódovanie znalostných modelov je potrebné zvážiť viacero faktorov. Medzi najdôležitejšie patria expresivita jazyka, sémantika jazyka a úroveň matematickej presnosti poskytovanej konkrétnym jazykom.

Expresivita jazyka vyjadruje rozsah konštruktov ním poskytovaných pre správne a explicitné definovanie komponentov vedomostného modelu. [11]

Sémantika vyjadruje stupeň jednoznačnosti jazyka v jeho špecifikácii, preto je dôležitá pri výbere jazyka pre prezentáciu vedomostí za účelom modelovania.

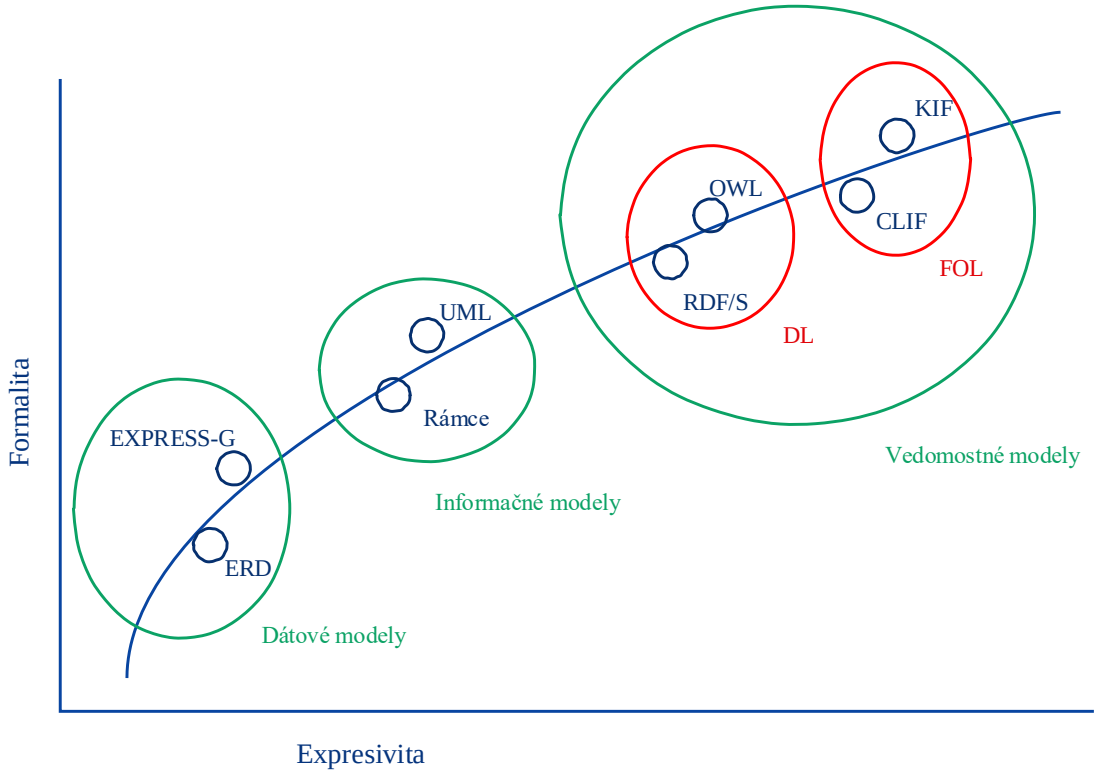
Ďalším faktorom je úroveň matematickej presnosti poskytovanej jazykom. Súvisí so stupňom satisfakcie a súdržnosti reprezentácie, teda ontologickej integrity a spoľahlivosti. Stupeň matematickej presnosti v ontológii je závislý od sémantiky zvoleného jazyka. [11]

Toto boli tri základné faktory. Okrem týchto svoju úlohu zohrávajú aj iné faktory, ako je napríklad skúsenosť vývojára ontológie a funkcie podporných nástrojov. [12]

Znalostný model, ktorý je vyjadrený matematicky prísnym spôsobom, nazývame ťažkou ontológiou. V porovnaní s ľahkými ontológiami, ktoré nie sú tak veľmi formálne a pri ktorých sú entity ľahko pochopiteľné, ťažké ontológie používajú okrem konštruktov ľahkých ontológií aj schopnosť zachytiť formálne axiómy a pravidlá, vďaka čomu lepšie vyjadrujú význam pojmov vo vytváraných vedomostných modeloch. [12]

Pre reprezentáciu dát, informácií a poznatkov existuje viacero jazykov, ktoré sa od seba odlišujú v stupni formálnosti a expresivity. Poznáme jazyky pre vytváranie dátových modelov, ako entitno-relačný diagram a EXPRESS-G, ďalej založené na modelovaní, ako je UML a rámce. Na druhej strane sú jazyky, pre reprezentáciu znalostí, ktoré využívajú formálnu logiku, ako sú Resource Description Framework (RDF) a Web Ontology Language (OWL). Tieto sú založené na opisnej logike. Tiež poznáme jazyky ako je Common Logic Interchange format (CL), Knowledge Interchange Format (KIF), ktoré sú postavené na logike prvého rádu. [13]

Na obrázku nižšie sú prehľadne zobrazené stupne formálnosti a expresivity jednotlivých jazykov.



Obrázok 3: Reprezentácie z pohľadu expresivity a formality [13]

1.5.1 Entitno-relačný model

Modelovanie vzťahov medzi entitami vyvinul Peter Chen, čo publikoval v roku 1976. [14] Entitno-relačný model vniká systematickou analýzou. Neurčuje procesy, slúži na reprezentáciu schémy doménových údajov v grafickej podobe, kde boxy reprezentujú jednotlivé typy entít a spojnice medzi nimi vyjadrujú ich vzájomné vzťahy. Entity sú okrem vzťahov opísané aj ich atribútmi, ktorých jeden, alebo kombinácia viacerých tvorí jednoznačný identifikátor, ktorý nazývame primárny kľúč. Entitno-relačný model je implementovaný ako relačná databáza. V takejto databáze vnímame jeden riadok tabuľky ako inštanciu typu entity a jednotlivé stĺpce vyjadrujú atribúty. Vzťahy vyjadrujeme tak, že primárny kľúč jednej tabuľky je uložený ako atribút v druhej tabuľke. Takýto údaj nazývame cudzí kľúč. V entitno-relačných modeloch rozlišujeme tri úrovne abstrakcie: Konceptuálna, logická a fyzická. [14]

- Konceptuálny dátový model

Cieľom konceptuálneho dátového modelu je vytvorenie jednotlivých typov entít, ich atribútov a definovanie ich vzájomných vzťahov medzi entitami. Táto úroveň dátového

modelovania sa nezaoberá detailmi databázovej štruktúry. Obsahuje tri základné prvky. Prvou je entita, ktorá predstavuje objekt reálneho sveta. Druhou je Atribút, ktorý predstavuje charakteristiku entity. Poslednou je Vzťah, ktorý vyjadruje závislosť alebo asociáciu medzi dvoma entitami. [14]

Konceptuálny dátový model ponúka pokrytie konceptov naprieč celou organizáciou. Tento dátový model sa vyvíja nezávisle od hardvérových špecifikácií. Zameranie je prednostne na reprezentáciu dát tak, ako sú vnímané používateľom v reálnom svete. Konceptuálne údajové modely poznáme ako doménové modely, ktoré vytvárajú slovník pre zainteresované strany prostredníctvom stanovenia základných pojmov a rozsahu.

- Logický dátový model

Logický dátový model pridáva k dátovým prvkom konceptuálneho modelu ich štruktúru a určuje vzťahy medzi nimi. Tento model poskytuje základ pre vytvorenie fyzického dátového modelu, pričom štruktúra modelovania ostáva generická. Na tejto úrovni modelovania upravujeme detaily konektorov, ktoré boli zvolené pre vzťahy medzi entitami a zatiaľ nedefinujeme primárne alebo sekundárne kľúče. Popisuje údaje potrebné pre špecifický projekt, ale môže integrovať s ďalšími logickými modelmi dát. Je vyvíjaný nezávisle od systému riadenia bázy dát. [14]

- Fyzický dátový model

Fyzický dátový model opisuje implementáciu dátového modelu v databáze. Poskytuje abstrakciu databázy a napomáha generovaniu schémy. S jeho pomocou vizualizujeme databázovú štruktúru. Pomáha pri modelovaní kľúčov, obmedzení, indexov, triggerov a ďalších funkcií systémov riadenia relačných databáz. Popisuje potrebu dát pre konkrétny projekt, ale môže byť integrovaný s inými fyzickými dátovými modelmi. Obsahuje vzťahy medzi tabuľkami týkajúce sa kardinality a existencie údajov. Vyvíjajú sa pre konkrétny systém riadenia bázy dát, dátového úložiska a technológiu použitú v súvisiacom projekte. Atribúty majú definované dátové typy, dĺžky a predvolené hodnoty. Sú definované primárne a cudzie kľúče, zobrazenia, indexy, prístupové roly a oprávnenia. [14]

1.5.2 EXPRESS-G

EXPRESS je jazyk používaný pre modelovanie dát definovaný normou ISO 10303-11. [15] V tomto jazyku vieme dátový model vyvíjať dvomi spôsobmi a to textovo a graficky. Pre človeka je vo väčšine prípadov ľahšie zrozumiteľné práve grafické vyjadrenie

nazývané EXPRESS-G, ale nedokáže reprezentovať všetky detaily, ktoré vieme definovať v textovej forme. Umožňuje definovať typy údajov so štrukturálnymi obmedzeniami a algoritmickými pravidlami. Najvýznamnejšou črtou je možnosť overenia populácie dátových typov, teda kontrola štrukturálnych a algoritmických pravidiel. [15]

EXPRESS-G je grafické modelovanie vyvinuté v rámci štandardu pre výmenu produktových modelových dát, skrátene STEP. S jeho pomocou definujeme triedy, ich atribúty a vzťahy medzi triedami. [16] Úzko súvisí s jazykom definovania údajov EXPRESS. To znamená, že všetko, čo je vyjadrené v EXPRESS-G vieme definovať aj v jazyku EXPRESS, ale v opačnom smere tento vzťah neplatí. [15]

1.5.3 Rámce

Marvin Minsky použil tento pojem, ako prvý v roku 1974, ako paradigmu pre pochopenie vizuálneho myslenia a spracovania prirodzeného jazyka. [17] Pojem rámec spočíva v tom, že vytvorí kontext pre problém, čím sa razantne ohraničí priestor vyhľadávania. [18]

Pri rámcoch je dôležité spomenúť sloty. Sloty môžeme prirovnať k atribútom tried v objektovo orientovanom modelovaní a reprezentácii vzťahov medzi entitami v entitno-relačnom modelovaní. Sloty môžu mať predvolené hodnoty, vyžadujúce zdokonalenie. To znamená, že sa vykonáva kontrola úlohy a to tak, že sa musí začať základnou rámcovou inštanciou a ďalej sa konkretizujú a vylepšujú ďalšie hodnoty. Spočiatku bol dôraz na opis statických údajov rámca. Tiež už existovali aj procedurálne možnosti, ako používanie spúšťačov pripojených k slotom. Spúšťač je procedurálny kód pripojený k slotu, ktorý môže napríklad meniť hodnotu slotu. [18] Rámce sú ontológie množín a podmnožín rámcových konceptov. Môžeme ich prirovnať hierarchickým triedam používaným v objektovo orientovaných jazykoch. Rámce explicitne a intuitívne zobrazujú poznatky, pričom objektovo orientované jazyky sa zameriavajú na zapuzdrenie a skrytie informácií. [18] V praxi sa použitie rámcov a objektovo orientovaných jazykov často prekrýva.

Rovnakým znakom s objektovými triedami je, že rámce sú tiež usporiadané v hierarchickej štruktúre. Špecializáciou rámca vzniká ďalší podradený rámec, ktorý dedí od nadradeného rámca všetky predvolené hodnoty, ale zároveň pridáva ďalšie sloty, alebo mení aspoň jednu predvolenú hodnotu pre tento rámec. [19] Značná časť výskumu rámcového jazyka vychádzala zo zistení v psychológii a riadila sa snažením vytvoriť nástroje pre

reprezentáciu poznatkov, ktoré odzrkadľujú ľuďmi používané vzory pri každodennom fungovaní. [19]

Jazyk KL-ONE je založený na rámcovej reprezentácii poznatkov. Formálna sémantika, ktorá sa používa aj v tomto jazyku priniesla rámcovým jazykom novú možnosť automatizovaného odvodzovania, ktoré nazývame klasifikátor. [20] Klasifikátor je mechanizmus, ktorý vykonáva analýzu deklarácií v rámcovom jazyku, čím môže automaticky odvodzovať ďalšie vzťahy a určiť nejednotné časti modelu. [21]

1.5.4 UML

Unified modeling language je jazyk pre modelovanie objektovo orientovaných systémov. UML poskytuje viacero techník modelovania nachádzajúcich sa vo väčšine moderných objektových metódach, ako sú objektové diagramy, stavové diagramy a diagramy interakcie objektov. Cieľom UML je poskytnúť štandardizovaný jazyk, ktorý nahrádza množstvo poznámok, diagramov a iných prezentačných nástrojov. [22]

Zaujímavým aspektom UML je , že existuje presný opis jeho sémantiky. Zámerom autorov bolo, aby vytvorili jednoznačný popis jazyka a zároveň umožnili jeho rozšírenie. [23]

UML je súbor grafických modelov, ktoré reprezentujú systémy objektovo orientovaným dizajnom. Dva najdôležitejšie typy modelov vytváraných prostredníctvom UML sú štrukturálne a behaviorálne modely. [23]

- Štrukturálne modely

Štrukturálny model sa používa pre modelovanie tried a ich inštancií v systéme, ich vzťahov, atribútov a obmedzení. Najpoužívanejším je diagram tried. Diagram tried pozostáva z viacerých komponentov: Z definície typov tried, množiny hodnôt, ktoré nadobúdajú inštancie tried, opisov operácií tried a ich parametrov, sadou asociačných vzťahov, generalizačných hierarchií. Práve generalizácia poskytuje spôsob, ako vytvárať dcérske triedy. [24]

- Behaviorálne modely

Behaviorálny model poskytuje informácie o správaní navrhovaného modelu. Opisuje správanie z hľadiska zmeny stavu a výmeny správ, ktoré sa vykonávajú pri konkrétnej udalosti. Príkladom takýchto modelov je napríklad stavový diagram, alebo diagram aktivít. [23]

1.5.5 RDF/S

RDF schéma dáva slovník pre modelovanie dát v resource description formáte. Je doplnená o niekoľko sprievodných dokumentov popisujúcich základné koncepty a abstraktnú syntax RDF, formálnu sémantiku RDF a rôzne konkrétne syntaxy pre RDF, ako sú Turtle, TriG, JSON-DL a iné. [25] RDF schéma je sémantické rozšírenie RDF. Táto schéma poskytuje mechanizmy na opis súvisiacich zdrojov a vzťahov medzi nimi. [25]

RDF schéma je systém tried a ich vlastností podobný objektovo orientovaným programovacím jazykom, ako je napríklad JAVA. Podstatným rozdielom je, že RDF schéma sa od nich líši tým, že triedy nedefinuje z hľadiska ich vlastností, ktoré majú svoje inštancie, ale opisuje vlastnosti z hľadiska tried zdrojov, na ktoré sa vzťahujú. [25] Pomocou RDF je jednoduché následne definovať ďalšie vlastnosti bez nutnosti opätovne definovať pôvodný popis tried. [25] Veľkou výhodou je, že umožňuje komukoľvek rozšíriť popis existujúcich zdrojov.

1.5.6 OWL

Použitie OWL nachádzame v prípadoch, kedy informácie obsiahnuté v dokumentoch neslúžia iba na prezentáciu ľuďom, ale tieto informácie sú spracovávané aplikáciami. Slúži na explicitné vyjadrenie významu pojmov a vzťahov medzi týmito pojmami. Toto vyjadrenie nazývame ontológia. Na rozdiel od XML, RDF a RDF-S poskytuje väčšie množstvo možností na vyjadrenie významu a sémantiky a preto presahuje tieto jazyky v schopnosti reprezentovať strojovo interpretovateľný obsah na webe. [26]

Použitie OWL je hlavne v sémantickom webe, kde poskytované informácie sú prístupné s explicitným významom a tým je jednoduchšie ich automatické strojové spracovanie. Sémantický web vychádza zo schopnosti XML definovať prispôsobené označovacie schémy a flexibilnom prístupe RDF k reprezentácii dát. Pre sémantický web sa vyžaduje ontologický jazyk, ktorý dokáže formálne opísať význam terminológie obsiahnutej vo webových dokumentoch. Pokiaľ očakávame od strojov vykonávanie užitočného odvodzovania z týchto dokumentov, tak nám nestačí základná sémantika RDF schém. [26]

Jazyk OWL bol navrhnutý, aby spĺňal túto požiadavku na webový ontologický jazyk. Je súčasťou W3C odporúčaní pre sémantický web. [26]

- XML poskytuje syntax pre štruktúrované dokumenty, ale nedokáže definovať sémantické obmedzenia týchto dokumentov.

- XML schéma je jazyk, ktorý umožňuje obmedziť štruktúru XML dokumentov a taktiež tento jazyk rozširuje o dátové typy.
- RDF predstavuje dátový model pre objekty a vzťahy medzi nimi, pričom poskytuje jednoduchú sémantiku pre tento dátový model a tieto dátové modely bývajú reprezentované prostredníctvom XML syntaxe.
- RDF schéma je slovník pre popis tried so sémantikou pre hierarchickú generalizáciu vlastností a tried.
- OWL je vylepšený slovník pre popis tried a ich vlastností. Poskytuje napríklad vyjadrenie vzťahov medzi triedami, kardinality, ekvivalencie.

OWL obsahuje tri podjazyky, ktoré sú určené pre konkrétne komunity implementátorov a používateľov. Sú to OWL Lite, OWL DL, OWL Full. [26]

OWL Lite najviac vyhovuje používateľom, ktorí potrebujú predovšetkým klasifikovať hierarchiu a jednoduché obmedzenia. OWL Lite ma zároveň nižšiu formálnu zložitosť ako OWL DL.

OWL DL je vyhovujúci pre používateľov, ktorí očakávajú maximálnu expresívnosť pri zachovaní výpočtovej úplnosti, čo znamená, že všetky závery sú vypočítateľné pri zachovaní rozhodovateľnosti, čo znamená, že všetky výpočty budú v určitom čase dokončené. OWL DL v sebe obsahuje všetky konštrukty jazyka OWL ale tieto sú použiteľné s určitými obmedzeniami. OWL DL získal svoje pomenovanie, pretože korešponduje s opisnou logikou (description logic - DL). [26]

Pre používateľov, ktorí požadujú maximálnu expresívnosť, ale zároveň chcú syntaktickú slobodu RDF, je určený jazyk OWL Full. OWL Full používateľom umožňuje používateľom ontológiu rozšíriť o význam preddefinovaného slovníka.

Každý z týchto jazykov vznikol rozšírením jeho jednoduchšieho predchodcu. Preto platia nasledujúce výroky:

- Každá OWL Lite ontológia je aj OWL DL ontológia.
- Každá UWL DL ontológia je aj OWL Full ontológia.
- Každé validné OWL Lite odvodenie je zároveň validným OWL DL odvodením.
- Každé validné OWL DL odvodenie je zároveň validným OWL Full odvodením.

Pred samotným procesom vývoja ontológie sa musíme zamyslieť, aké sú naše potreby a tomu prispôsobiť aj výber jazyka. V prípade rozhodovania medzi OWL Lite

a OWL DL si treba uvedomiť do akej miery požadujeme expresívne konštrukty poskytované jazykom OWL DL. Pokiaľ sa rozhodujeme medzi OWL DL a OWL FULL, tak berieme do úvahy, do akej miery vyžadujeme metamodelovacie vlastnosti RDF schémy. [26]

OWL Full môžeme chápať ako rozšírenie RDF, ale jazyky OWL DL a OWL Lite iba ako rozšírenia obmedzeného pohľadu na RDF. Z toho vyplýva, že každý OWL dokument je RDF dokumentom a každý RDF dokument je OWL Full dokumentom, ale nie všetky RDF dokumenty sú OWL Lite, alebo OWL DL dokumentmi. Pokiaľ je pre naše potreby postačujúca expresívnosť jazykov OWL Lite alebo OWL DL, tak je potrebné, aby RDF dokument spĺňal obmedzenia definované jazykmi OWL Lite a OWL DL

1.5.7 KIF

Knowledge interchange format, skrátene KIF, čo znamená formát pre výmenu znalostí medzi rôznymi systémami. Tento jazyk nie je určený primárne pre komunikáciu ľudskými používateľmi. Taktiež nie je používaný pre internú reprezentáciu znalostí v rámci jednotlivých programov. Ak program číta bázu znalostí vyjadrenú v KIF, tak tieto znalosti konvertuje a výpočty sa vykonávajú nad týmito už konvertovanými formami. V prípade, ak program vyžaduje komunikáciu s iným systémom, tak tieto znalosti konvertuje späť do KIF. [28]

Jeho účel vieme prirovnať k Postscript, ktorý sa používa pri textových a grafických formátovacích aplikáciách pri komunikácii. KIF nie je vnímaný ako špecializovaná reprezentácia vedomostí, ale je programovo čitateľným jazykom, čím zjednodušuje nezávislý vývoj programov, ktoré manipulujú s poznatkami.

Definícia KIF je veľmi podrobná. Spomeňme tri základné vlastnosti KIF: [28]

- Sémantika jazyka je deklaratívna. Umožňuje pochopenie významu výroku bez odvolávania sa na interpreter pre manipuláciu s týmto výrokom. Týmto sa KIF odlišuje od iných jazykov, ako sú Emycin a Prolog.
- Jazyk je logicky komplexný, čo zabezpečuje vyjadrenie viet v predikátovom výpočte. Týmto sa je odlišný od relačných databázových jazykov a jazykov podobných Prologu.
- Jazyk poskytuje reprezentáciu znalostí o reprezentácii znalostí. To poskytuje vykonanie rozhodnutí o reprezentácii poznatkov explicitne a umožňuje zaviesť nové konštrukty reprezentácie vedomostí bez nutnosti zmeny jazyka.

Okrem týchto pevne daných kritérií je dizajnovaný tak, aby maximalizoval ďalšie meradlá: [28]

- Hlavnou výkonnostnou požiadavkou pre KIF je preložiteľnosť. To znamená, že umožňuje preklad deklaratívnych vedomostných báz do a z typických jazykov pre reprezentáciu poznatkov.
- Ďalšou požiadavkou je čitateľnosť. Napriek tomu, že KIF nie je prioritne určený pre interakciu s ľudským používateľom, tak čitateľnosť zjednodušuje jeho použitie pri popise sémantiky jazyka.
- Treťou požiadavkou je použiteľnosť ako jazyka reprezentácie. Aj keď nie je určený na použitie v jednotlivých aplikáciách ako reprezentačný jazyk vedomostí, môžeme ho v prípade potreby na tento účel použiť.

1.5.8 CLIF

Common logic, skrátene CL, je framework pre skupinu logických jazykov založených na logike prvého rádu. Jeho použitie nachádzame vo prenose poznatkov medzi počítačovými systémami. [29] V roku 2001 Hayes a Menzel definovali všeobecnú teóriu modelu pre CL, ktorú v roku 2003 Hayes spolu s McBrideom použili pre definovanie sémantiky jazykov RDF/S a OWL. Okrem syntaxe a modelovej teórie, táto norma špecifikuje tri dialekty, ktoré vyjadrujú plnú sémantiku CL: [29]

- Common Logic Interchange format – CLIF,
- Conceptual Graph Interchange format – CGIF,
- Notáciu pre XL založenú na XML – XCL.

Vzhľadom k tomu, že sémantiky jazykov DRF a OWL sú založené na podmnožine sémantiky CL, môžeme povedať: Akýkoľvek výrok v RDF alebo OWL môže byť preložený do CLIF, CGIF, alebo XCL, ale z druhej strany iba časť výrokov v CL môže byť preložená do RDF alebo OWL. [30]

1.6 Odvodzovanie nových poznatkov v OWL

Hierarchia tried, ktorú navrhujeme v ontologickom modeli je označovaná, ako známa, alebo uplatňovaná hierarchia tried a to preto, lebo sa skladá z poznatkov, ktoré o nej vieme. Systémy pre vývoj ontológií zvyčajne poskytujú automatizované argumentačné služby, čím systém dokáže odvodzovať nové poznatky na základe vývojárom vytvorených

opisov tried a definícií, alebo vykonávať extrapoláciu založenú na entitách, ktoré sú deklarované v znalostnom modeli.

Jednou z týchto služieb je automatizovaná klasifikácia. Je užitočná pri vytváraní veľkých ontológií a overuje, či je popis tried správny. Týmto sa zjednodušuje udržiavanie vedomostného modelu a odhaľujú viditeľné chyby, inak povedané overuje sa udržiavanie konzistencie.

Dôležitou vecou je, že OWL sa zakladá na predpoklade otvoreného sveta, čo znamená, že ak výrok nie je explicitne deklarovaný ako pravdivý, tak to neznamená, že je nepravdivý. Opakom, ktorý sa používa pri databázových systémoch je predpoklad uzavretého sveta, kde ak niečo nie je pravda, tak je to nepravda.

V mnohých aplikáciách sú dokončené iba časti databáz a v iných častiach sú chýbajúce údaje. Tu sa stretávame s predpokladom čiastočne uzavretého sveta. Vzniká problém rozhodovania, či dotaz vráti všetky možné odpovede. Ak áno, tak dotaz sa nazýva kompletný. [31]

2 Cieľ a metodika práce

Cieľom diplomovej práce je nájsť vhodný nástroj na modelovanie ontológie pre vybraný predmet štúdia na FHI, vytvoriť ontologický model jadra predmetu a ukázať jeho modifikovateľnosť. Ako predmet štúdia sme zvolili predmet Umelá Inteligencia a zameriavame sa na jeho časť prehľadávania stavového priestoru. Pri vypracovávaní ontológie postupujeme podľa kombinácie metód jednoduchej metodiky znalostného inžinierstva a IDEF5. Výber metodiky sme prispôsobili zvolenému nástroju Protégé. Autori tohto nástroja odporúčajú použitie jednoduchej metodiky znalostného inžinierstva.

Cieľ práce sme rozdelili do viacerých menších cieľov. V teoretickej časti sme spracovali dostupné informácie o ontológiách a znalostnom inžinierstve.

V praktickej časti sa najprv venujeme náčrtu ontologického modelu. Analýzou dostupných informácií o štruktúre predmetu vytvárame náčrt ontologického modelu. Náčrt zobrazený v práci je výsledkom iteratívneho procesu vývoja. Tento náčrt sme priebežne upravovali tak, aby čo najviac zodpovedal realite. Treba podotknúť, že každý môže skutočnosti vnímať inak a výsledný náčrt je vždy závislý od predstavy a vnímania jeho autora.

Ďalej sa zaoberáme samotným prostredím Protégé,, ktoré sme si zvolili ako nástroj pre modelovanie ontológie. Použili sme jeho lokálnu inštaláciu a analýzou jeho fungovania a následnou syntézou týchto nami získaných poznatkov sme popísali komplexný postup vytvárania ontológií pomocou toho nástroja.

V ďalšom pokračovaní praktickej časti venujeme algoritmom prehľadávania stavového priestoru, kde vyhľadávame a analyzujeme jednotlivé atribúty algoritmov. Po vykonaní analýzy algoritmov sme jednotlivé algoritmy klasifikovali a tým zaradili do tried.

Po vytvorení ontológie na nej demonštrujeme odvodzovanie nových poznatkov prostredníctvom automatizovanej klasifikácie s použitím nástroja HermiT a poukazujeme, ako toto odvodzovanie môže pomôcť pri plánovaní procesu výučby.

V poslednej časti sa venujeme vizualizácii ontológie. K tomuto účelu používame rozšírenia Protégé OWLViz a OntoGraf.

3 Výsledky práce

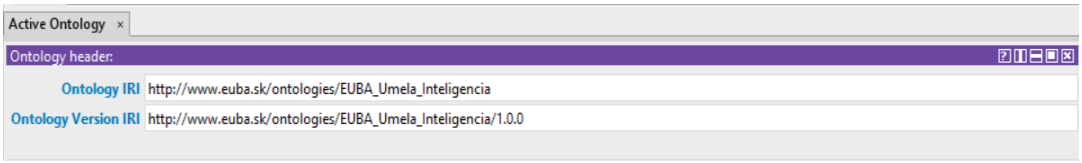
3.1.1.1 Protégé

Protégé je bezplatné, opensource vývojové prostredie používané pri vývoji ontológií, ktoré uľahčuje prácu v rámci životného cyklu vytvárania ontológií. Pre používanie máme dve možnosti. Jednou z nich je využitie hosťovaného prostredia Webprotégé, alebo môžeme použiť lokálnu inštaláciu prostredia.

Plne podporuje najnovší jazyk OWL2. Jeho konfigurovateľné prostredie je vhodné pre všetky úrovne používateľov, či už začiatočníkov, alebo expertov. Podporuje viacero formátov, ako RDF/XML, Turtle, OWL/XML, OBO a ďalšie pre nahrávanie a sťahovanie ontológií.

3.2 Vytváranie ontológie v prostredí Protégé

Po otvorení aplikácie Protégé sa nám zobrazí prázdny projekt. Prvé, čo musíme spraviť je definovať základné údaje o ontológii. Naša ontológia už má pridelené URI, čo znamená Uniform resource Identifier. Práve URI je unikátnym identifikátorom každej ontológie. V našom prípade vyvíjame ontológiu študijného predmetu Umelá inteligencia vyučovanom na Ekonomickej univerzite v Bratislave, preto sme URI nastavili hodnotu http://www.euba.sk/ontologies/EUBA_Umela_Inteligencia. Ontológia prechádza vývojom, počas ktorého vykonávame rôzne zmeny, preto je dôležité s akou verziou ontológie pracujeme. Verziu vidíme hneď pod Ontology URI v poli Ontology Verzion IRI.



Obrázok 4: IRI a verzia ontológie [vlastné spracovanie]

Vhodné je popísať, čoho modelom je naša ontológia. Na to slúži okno anotácie, kde vieme pridať popis ontológie ako komentár.

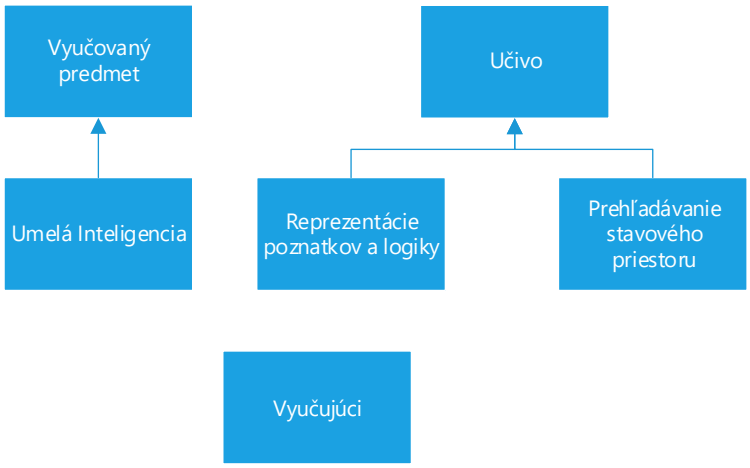
Obrázok 5: Komentár ontológie [vlastné spracovanie]

Spomeňme užitočnú možnosť, ktorú my v našej ontológii nepoužijeme. V prípade vývoja komplexnejšej ontológie, vieme ontológiu rozdeliť do viacerých menších ontológií a tie následne použiť a to tak, že ich do projektu importujeme. Jednotlivé parciálne ontológie importujeme na karte Active ontology v sekcii Ontology import.

3.2.1 Triedy

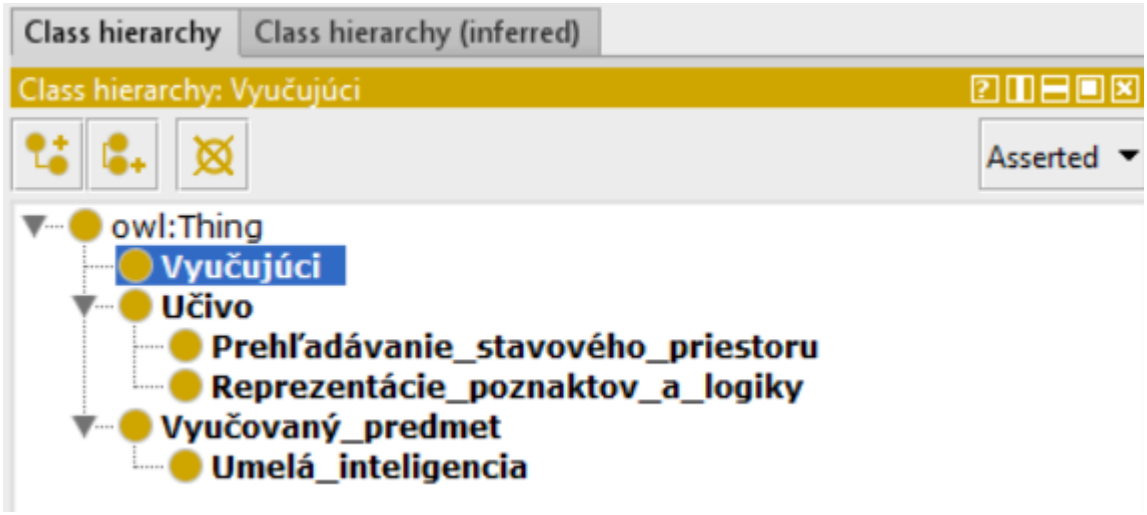
Základným stavebným prvkom v OWL sú triedy a ich hierarchia. Vytváranie a úpravu tried vykonávame na karte Classes. Na ľavej strane vidíme hierarchiu tried. Na pravej strane sa nachádza okno s anotáciami a pod nimi veľmi dôležitá časť Desription, v ktorej vytvárame popis OWL tried.

Prázdna ontológia v OWL obsahuje jednu triedu s názvom Thing, ktorá tvorí kmeň, z ktorého sa vetvia ostatné OWL podtriedy. Na začiatok pridávame 6 tried, ktoré vyjadrujú Umelú inteligenciu ako vyučovaný predmet, vyučujúcu, kde sú zamestnanci, ktorí vyučujú a učivo, v ktorom sú tematické okruhy vyučované v predmete. Pridáme ich v štruktúre, ako na obrázku nižšie.



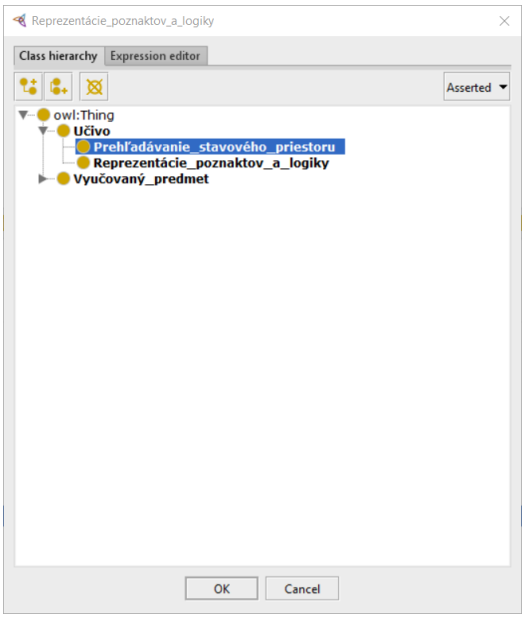
Obrázok 6: Základný náčrt ontológie

V prvom kroku vytvoríme triedu na nižšej úrovni v hierarchickej štruktúre, pretože zatiaľ existuje iba trieda Thing. Tým sme vytvorili podtriedu k nami zvolenej triede. Ďalej je možné triedy vytvárať aj na rovnakej úrovni v hierarchickej štruktúre, čim vytvoríme sesterskú triedu. Po vytvorení nami definovaných tried vyzerá strom tried ako na obrázku nižšie.



Obrázok 7: Triedy v ontológii [vlastné spracovanie]

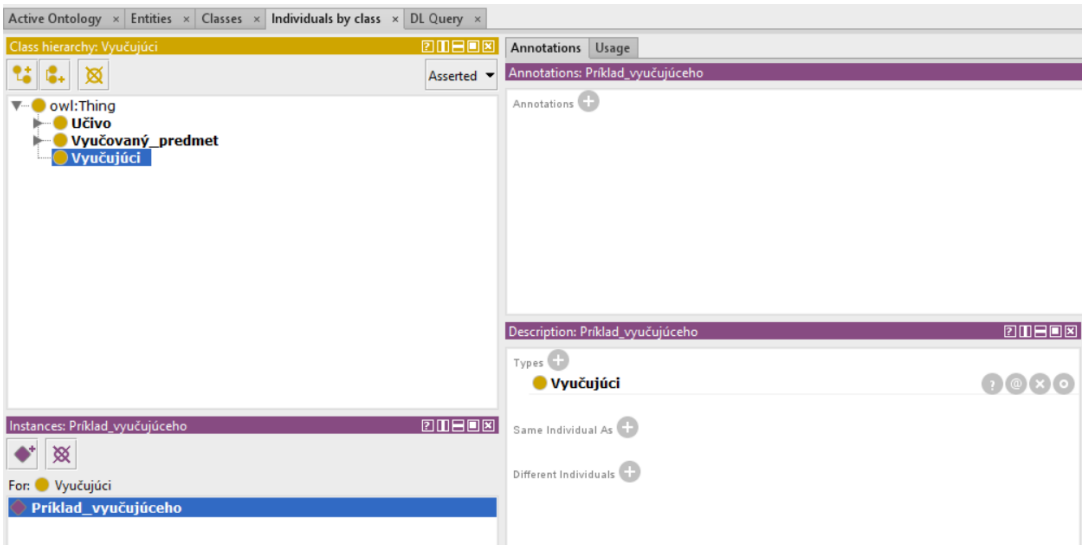
Zamerajme sa na triedy Prehľadávanie_stavového_priestoru a Reprezentácie_poznatkov_a_logiky. Predstavme si, že jednotlivec jednej z týchto tried predstavuje konkrétne učivo vyjadrené či už na papieru, alebo elektronicky. Je zrejmé, že prvok Prehľadávanie stavového priestoru nebude prvkom Reprezentácií poznatkov a logík a tento vzťah platí aj opačne. Tieto triedy sú teda disjunktné. Inak povedané, neexistuje medzi nimi žiaden prienik. Pre konkrétnu triedu v OWL to zadefinujeme pridaním Disjoint with, ktorý nájdeme v časti Description. Po kliknutí sa nám otvorí okno, v ktorom vyberieme s ktorou triedou je zvolená trieda disjunktná. Po potvrdení Protégé zadefinuje disjunktnosť aj pri práve vybraných triedach.



Obrázok 8: Disjunktnosť triedy [vlastné spracovanie]

3.2.2 *Inštancie tried*

V ontologickom modeli okrem tried vystupujú jedinci, ktorí predstavujú inštancie tried. Inštancie editujeme na karte Individuals by class. Opäť vidíme hierarchický strom tried. Zvolíme si triedu, do ktorej chceme pridať inštanciu. A v časti Instances vytvoríme novú. V časti Description si všimnime že Types je vyplnené práve zvolenou triedou, čo reprezentuje, ktorej triedy je inštancia súčasťou. Na obrázku nižšie vidíme, že sme vytvorili jedinca Príklad_vyučujúceho, ktorý je jedincom triedy Vyučujúci.

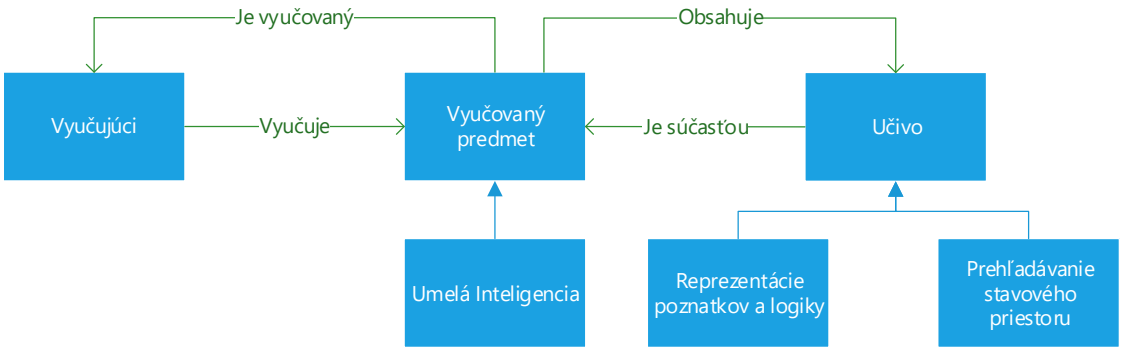


Obrázok 9: Inštancia triedy [vlastné spracovanie]

3.2.3 *Vzťahy*

Pre vyjadrenie vzťahov v OWL sa používa pojem vlastnosti. V OWL rozlišujeme dva hlavné druhy. Jedným sú vlastnosti objektu a druhým sú vlastnosti dátového typu. My sme zatiaľ použili tretí typ a pridanie komentára ako anotácie. Rozdielom medzi vlastnosťami objektovými a dátového typu je, že objektové vlastnosti vyjadrujú vzťahy medzi jedincami, zatiaľ čo vlastnosti dátového typu vyjadrujú asociáciu niečoho k dátovej hodnote.

V našom vizuálnom ontologickom modeli pridávame zatiaľ vzťahy medzi triedami Vyučujúci a vyučovaný predmet a vzťahy medzi Vyučovaným predmetom a Učivom. Vizuálny model ontológie je reprezentovaný nižšie.

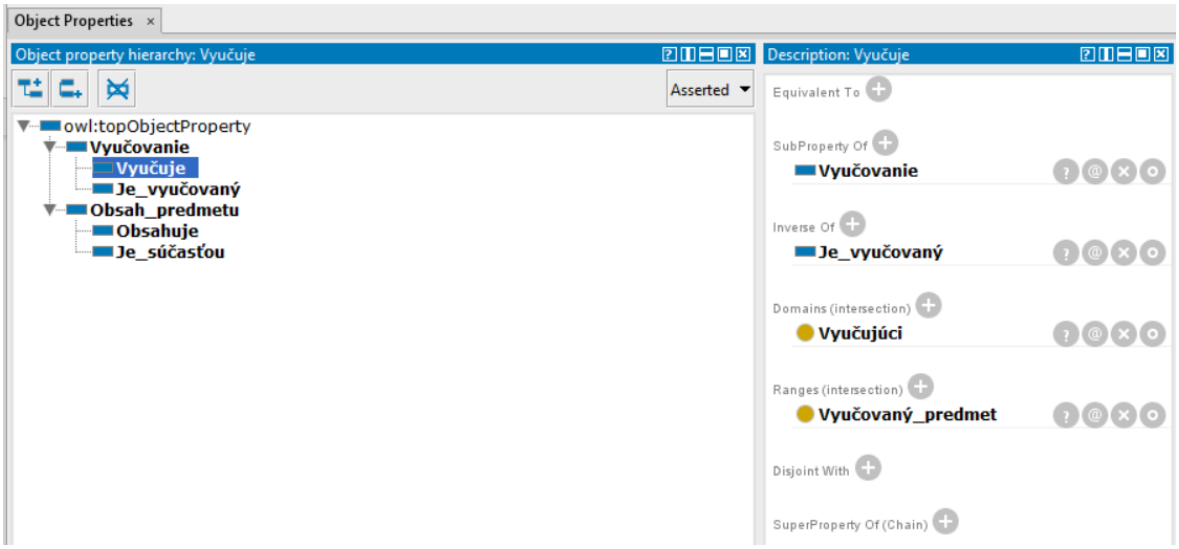


Obrázok 10: Zobrazenie vzťahov [Vlastné spracovanie]

V Protégé prostredí, definujeme vlastnosti objektu na karte Object properties. Na tejto karte, podobne ako na karte Classes, vidíme hierarchiu, ale s tým rozdielom, že ide o hierarchiu objektových vlastností. Znamená to, že nami vytvorené vlastnosti objektov môžeme kategorizovať, čím získavame prehľadnejšie usporiadanie. Pre demonštráciu si to ukážeme na príklade, kde vytvoríme vlastnosť objektu s názvom Obsah predmetu a pod túto objektovú vlastnosť zoskupíme tie, ktoré vyjadrujú vzťahy medzi triedami Vyučovaný predmet a Učivo, čiže vlastnosti objektov Je súčasťou a Obsahuje. Takto sme vytvorili samotné objektové vlastnosti, ale neviem o nich nič viac.

Teraz definujeme medzi ktorými triedami tieto objektové vlastnosti vytvárajú vzťah. Objektové vlastnosti majú v časti Description atribúty Domains a a Ranges. Všimnime si, že v našom vizuálnym OWL modeli je vyjadrenie vzťahov orientované. Pre objektovú vlastnosť Vyučuje je doménou práve trieda Vyučujúci a rozsahom je trieda Vyučovaný predmet.. Toto popisuje fakt, že inštancia triedy Vyučujúci vyučuje niektorú inštanciu triedy alebo podtriedy Vyučovaný predmet. My chápeme ako inštanciu konkrétne vyučovanie v akademickom roku. Preto je Umelá inteligencia vyjadrená ako podtrieda vyučovaného predmetu a nie ako inštancia tejto triedy. Preto Objektovej vlastnosti Vyučuje pridáme do domény triedu Vyučujúci a do rozsahu triedu Vyučovaný predmet.

Všimnime si, že objektová vlastnosť Je vyučovaný vyjadruje vzťah medzi tými istými triedami, ale opačne, teda vyjadruje to, že vyučovaný predmet je vyučovaný vyučujúcim. Toto znamená, že objektové vlastnosti Vyučuje a je vyučovaný sú vzájomne inverzné. Toto definujeme v časti description v atribúte Inverse of, kde po otvorení okna pre pridanie tejto definície vyberieme zo stromu inverznú vlastnosť objektu k práve vybranej. Na obrázku nižšie je zobrazené nastavenie objektovej vlastnosti Vyučuje.



Obrázok 11: Objektové vlastnosti [vlastné spracovanie]

V OWL definujeme ďalšie charakteristiky vlastností, ktoré nám napomáhajú k lepšiemu špecifikovaniu vzťahov v ontologickom modeli. Prvou charakteristikou, ktorou vieme vlastnosť označiť je funkčná. Znamená to, že vzťah medzi triedami je jedna k jednej. Inak povedané inštancia jednej triedy vyjadruje vzťah práve k jednej inštancii v druhej triede. Napríklad, keď vyjadrujeme bod mrazu vody v stupňoch Celzia a v stupňoch Fahrenheita, tak vieme, že tieto dve teploty reprezentujú toho istého jedinca. Na druhej strane poznáme inverznú funkčnú charakteristiku, čim vyjadríme, že jedincovi môže byť priradený iba jeden jedinec. Inak povedané, že ak nevieme, ak máme bod mrazu a k nemu viazané dve hodnoty v rôznych merných jednotkách, tak môžeme vyhlásiť, že tieto dve teploty sú rovnaké.

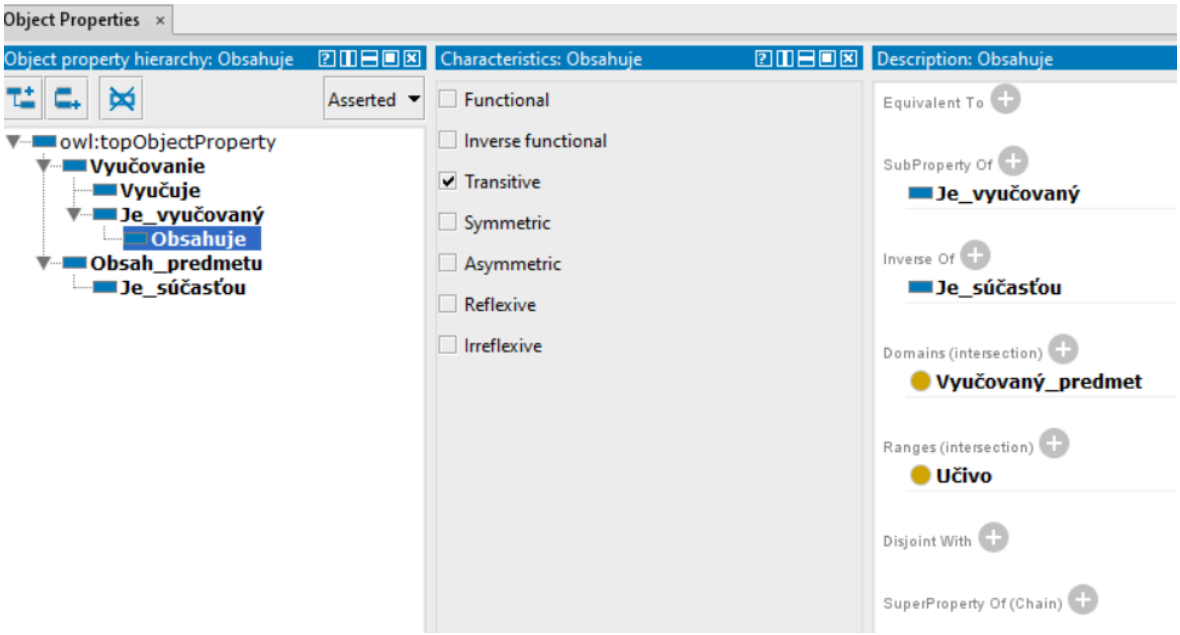
Ďalšou je tranzitívna charakteristika. Ak A vyvoláva B a B vyvoláva C, tak vieme povedať, že A vyvoláva C. Na príklade, ak mletie kávy vyvoláva spustenie kávovaru a spustenie kávovaru vyvoláva pitie kávy, tak vieme povedať, že mletie kávy vyvoláva pitie kávy.

Symetrickosť a asymetrickosť vzťahov je vyjadruje v akom smere platí vzťah. Ak je symetrický tak vyjadrujeme, že ak A je spojení s B prostredníctvom konkrétneho vzťahu, tak B je v spojení s A s použitím toho istého vzťahu. Na druhej strane asymetrickosť vyjadruje, že ak A je v spojení prostredníctvom vzťahu X, tak B nemôže byť v spojení s A prostredníctvom toho istého vzťahu.

Reflexívnosť znamená, že objekt okrem vzťahu s iným objektom, môže vo vzťahu aj sám so sebou. Naopak nereflexívnosť vyjadruje že objekt nemôže byť vo vzťahu so samým sebou.

V našom modeli nastavíme zatiaľ pre Objektovú vlastnosť Obsahuje tranzitívnosť, čo znamená, že ak učivo obsahuje ďalšie učivo, tak vyučovaný predmet obsahuje aj toto učivo. Na obrázku je vidieť Vzťah Obsahuje v rámci hierarchie vzťahov, spolu s jeho charakteristikou a opisom.

Vlastnosti dátového typu obmedzujú, aké hodnoty môže nadobúdať jedinci v ontologickom modeli. Môžeme definovať, či to musí byť celé číslo, desatinné číslo, reťazec znakov a podobne. Takisto vieme definovať rozsah hodnôt, aké môže nadobúdať.



Obrázok 12: Charakteristika objektových vlastností [vlastné spracovanie]

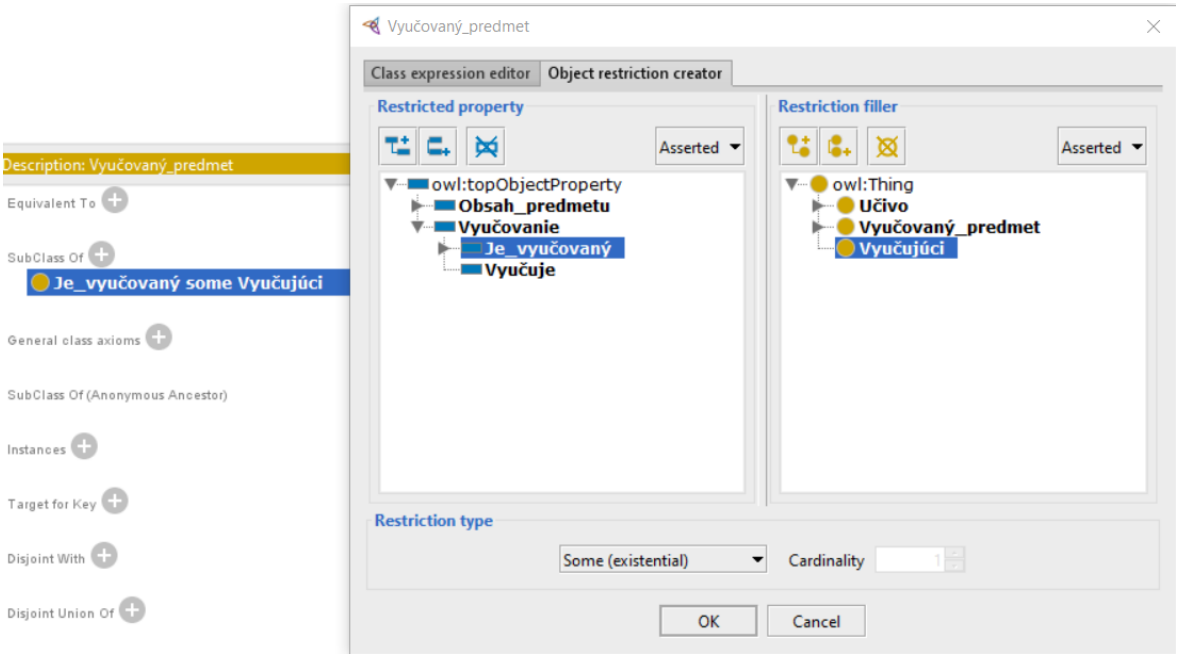
3.2.4 Charakteristika a definícia tried

V okamihu, kedy sú máme vytvorené triedy, vzťahy medzi triedami a inštancie tried, sa dostávame do momentu, kedy začíname definovať skutočnú sémantiku. Náš vizuálny model môžeme nazvať ľahkou ontológiou. Tým, že ju implementujeme v OWL vznikajú matematické obmedzenia, čim sa ontológia stáva lepšia pre výmenu poznatkov medzi počítačovými systémami. Teraz začíname vytvárať reštrikcie.

V OWL rozlišujeme tri druhy reštrikcií. Je to obmedzenie kvantifikátora, obmedzenie hodnoty a obmedzenie kardinality.

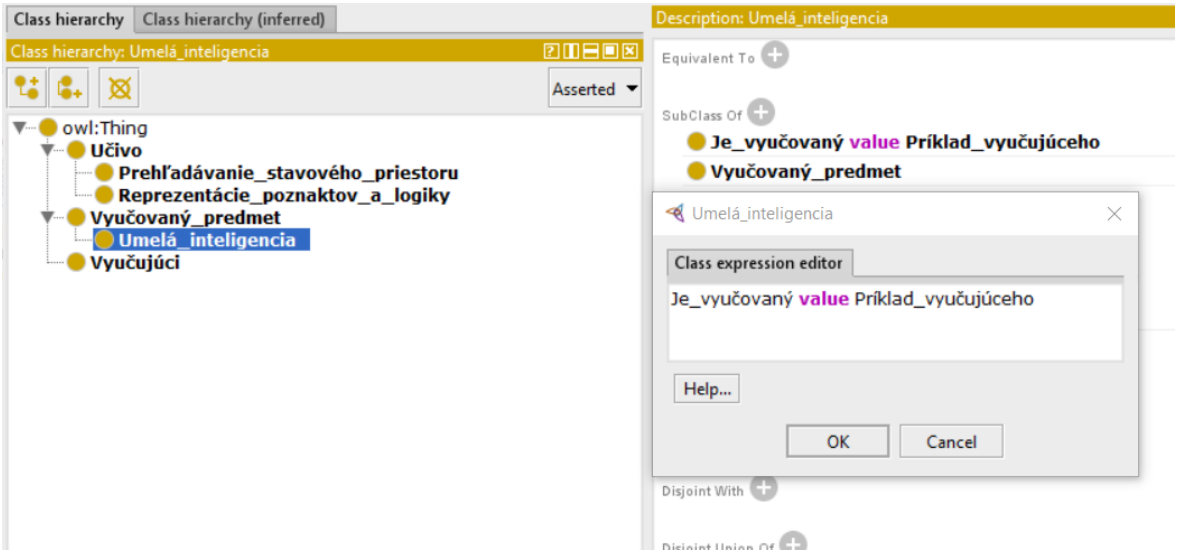
Existenčné obmedzenie je obmedzením kvantifikátora a je najbežnejším obmedzením používaným v OWL. Napríklad jedinec triedy Vyučovaný predmet potrebuje asociáciu s jedincom triedy Vyučujúci, čim vyjadríme podmienku, že vyučovaný predmet

musí mať aspoň jedného vyučujúceho. V Protégé je existenčné obmedzenie vyjadrené prostredníctvom výrazu some. V hierarchii vyberieme triedu a vyberieme pridanie podtriedy. V novootvorenom okne prejdeme na kartu Object restriction creator, kde editujeme obmedzenie. Vyberieme objektovú vlastnosť Je vyučovaný, na pravej strane zvolíme triedu Vyučujúci a vyberieme typ obmedzenia Some (existential).



Obrázok 13: Existenciálne obmedzenie [vlastné spracovanie]

Definovali sme fakt, že vyučovaný predmet je vyučovaný aspoň jedným vyučujúcim, ale nevieme ktorým. Sú prípady, kedy chceme obmedziť fakt, že inštancia triedy je viazaná s konkrétnou inštanciou inej triedy. V takomto prípade nám už nestačí existenčné obmedzenie, ale použijeme obmedzenie, ktoré sa obmedzenie hodnoty. Napríklad, ak máme bod varu vody, tak hodnota teploty vody musí byť 100 stupňov Celzia a nie ľubovoľná hodnota teploty. Ak by sme v našom modeli chceli vyjadriť, že každá hodina predmetu Umelá inteligencia musí byť vyučovaná vyučujúcim Príklad vyučujúceho, tak zvolíme triedu Umelá inteligencia, zvolíme možnosť definovania podtriedy a zvolíme kartu Class expression editor. V editore najprv napíšeme vzťah medzi triedami, ďalej vyjadrenie obmedzenia hodnoty, teda value a nakoniec inštanciu triedy, ktorá reprezentuje hodnotu, na ktorú chceme obmedziť jedincov triedy.



Obrázok 14: Obmedzenie hodnoty [vlastné spracovanie]

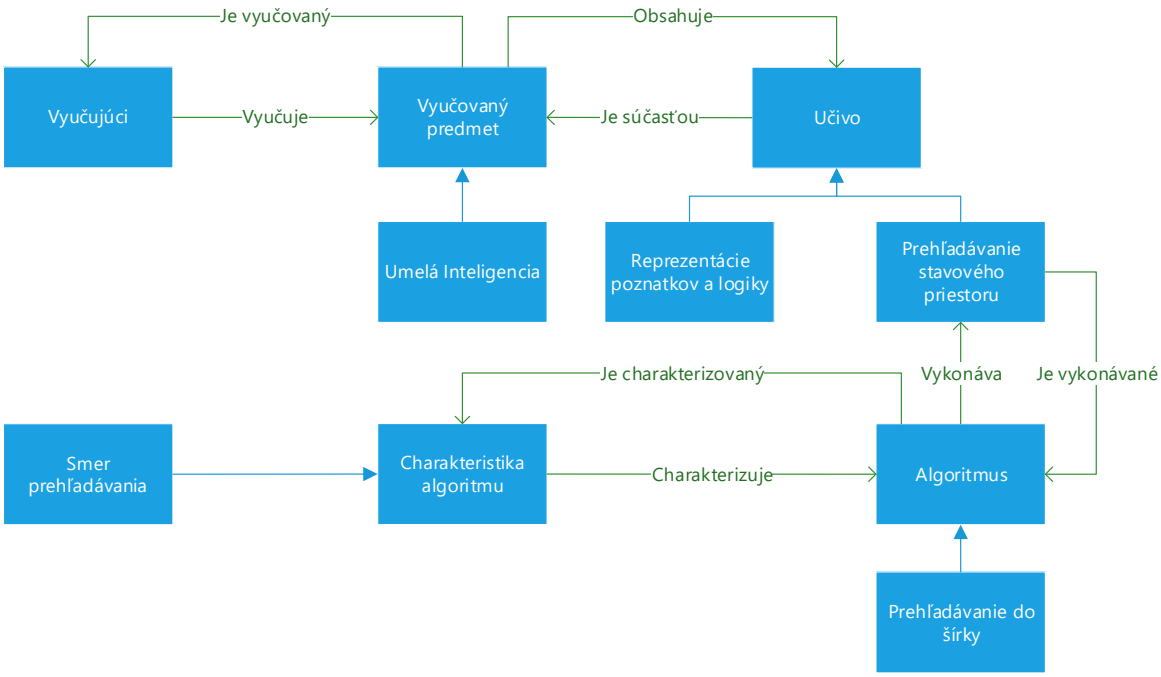
Okrem existenciálneho obmedzenia kvantifikátora poznáme aj univerzálne obmedzenie. Existenčným kvantifikátorom sme existenciu minimálne jednej väzby s použitím konkrétneho vzťahu, čo ale nezabraňuje vytvoreniu väzby s inštanciou inej triedy s použitím toho istého vzťahu. Na takýto účel používame univerzálne obmedzenie. V Protégé ho vytvárame v časti editovania podtried na karte Object restriction creator. Na ľavej strane vyberieme vzťah a na pravej vyberieme triedy, s ktorými je trieda spojená týmto vzťahom a s spodnej časti nastavíme Only (universal). Týmto definujeme, že týmto vzťahom trieda nemôže byť viazaná na žiadnu inú triedu mimo nami definovaného rozsahu.

V prípade, kedy chceme vyjadriť, je inštancia triedy je viazaná s konkrétnym počtom inštancií inej triedy, použijeme obmedzenie kardinality. Vieme vyjadriť presný počet, maximum a minimum. Obmedzenie pridávame na karte Object restriction creator výberom vzťahu, triedy a zvolením jeden z možností: Min (min cardinality), Max (max cardinality), Exactly (exact cardinality).

3.2.5 Primitívne a definované triedy

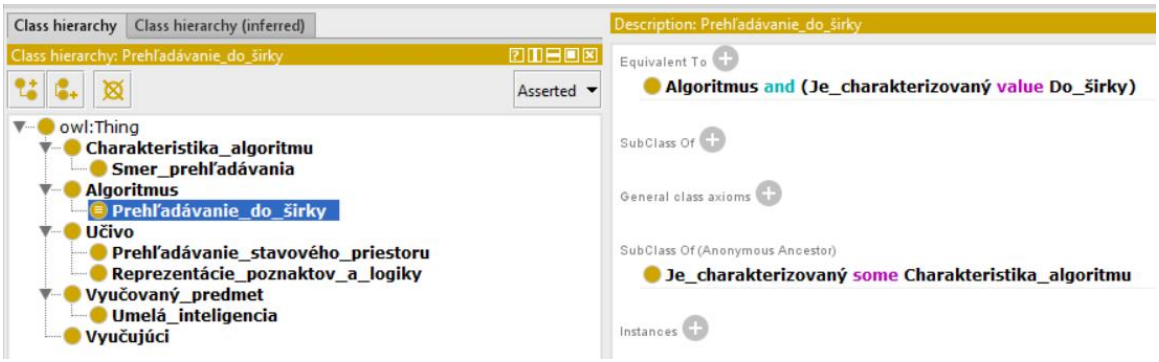
Triedy ktoré sme doteraz vytvorili a definovali pomocou reštrikcií sú primitívnymi triedami. Aby bol jedinec inštanciou triedy musí spĺňať súbor podmienok definovaný konkrétnou triedou.. Toto je súbor nevyhnutných podmienok. V OWL definujeme súbor nevyhnutných a dostatočných podmienok. S týmito podmienkami vieme vyjadriť dve skutočnosti. Prvou je, že inštancie triedy musia spĺňať súbor reštrikcií triedy a druhou je, že ak jedinec spĺňa reštrikcie triedy, tak musí byť jej inštanciou V Protégé definovanú triedu

vytvoríme tak, že vyberieme triedu, a cez edit vyberieme možnosť convert to defined class. Náš vizuálny náčrt ontológie sme rozšírili a teraz vyzerá nasledovne.



Obrázok 15: Triedy a vzťahy [vlastné spracovanie]

Z modelu je zrejmé, že Prehľadávanie stavového priestoru je vykonávané algoritmom a ten obsahuje podtriedu algoritmov prehľadávania do šírky. Algoritmy majú charakteristiky. My zatiaľ uvažujeme že podtrieda Smer prehľadávania obsahuje inštanciu Do šírky. Predpokladáme, že aby algoritmus patril do triedy Algoritmov prehľadávania do šírky, tak smer prehľadávania musí byť do šírky a zároveň, ak algoritmus má smer prehľadávania do šírky, tak patrí do triedy prehľadávania do šírky. Preto triedu Prehľadávanie do šírky konvertujeme z primitívnej triedy na definovanú. Všimnime si na obrázku ako sa zmenil opis triedy. Nie je vyjadrené, že je podtriedou triedy Algoritmus, ale vidíme, že v sekcii Equivalent To pribudol záznam Algoritmus and (Je_charakterizovaný value Do-šírky).



Obrázok 16: Definovaná trieda [vlastné spracovanie]

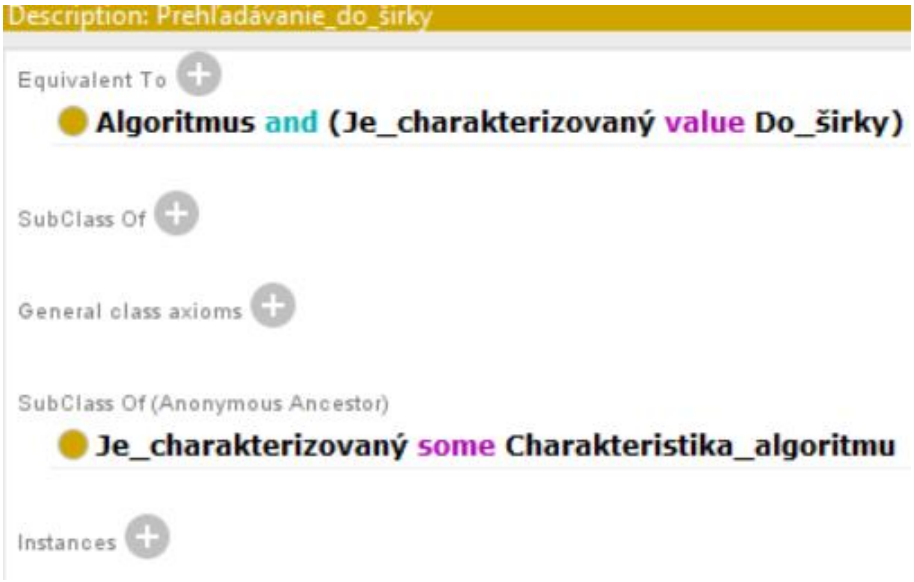
3.3 Analýza algoritmov prehľadávania stavového priestoru

Teraz, keď už vieme, ako pracovať v systéme Protégé, tak sa dostávame k samotnému jadru práce a tou je analýza jednotlivých algoritmov prehľadávania stavového priestoru, Získané informácie zaznamenávame v Protégé, čím vytvárame ontologický model časti vyučovaného predmetu Umelá inteligencia.

Zozbierané poznatky o algoritmoch v Protégé zaznamenávame spôsobom, že pre triedu Charakteristika algoritmu vytvárame podtriedy s jednotlivými charakteristikami, ktoré obsahujú jedincov reprezentujúcich konkrétne údaje a v triede Algoritmus vytvárame podtriedy pre jednotlivé algoritmy. Následne v anotácii pridáme krátky popis algoritmu a v pomocou vzťahov spojíme triedu algoritmu s vlastnosťami, ktoré ho reprezentujú.

3.3.1 Prehľadávanie do šírky

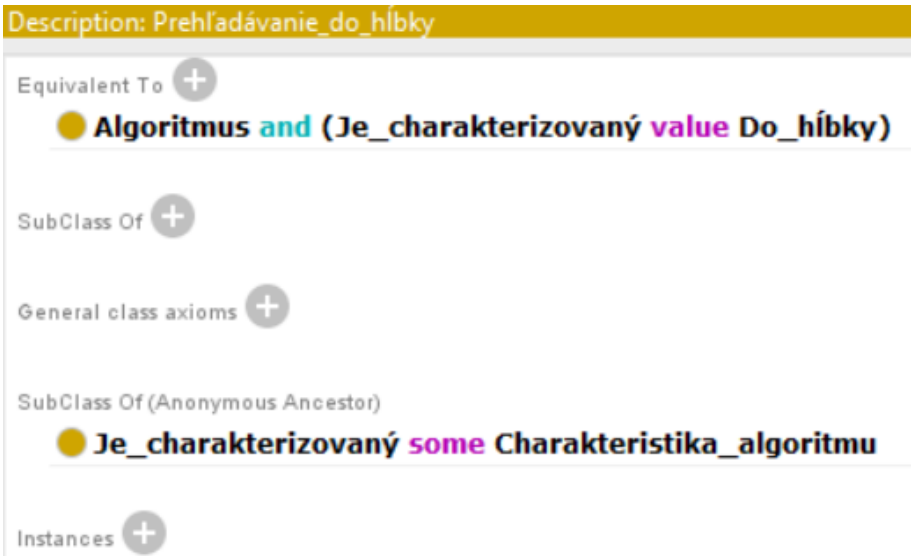
Využitie algoritmu nachádzame v prehľadávaní stromových alebo dátových štruktúr z počiatočného uzla preskúmava všetky susedné uzly na nasledujúcej úrovni. [32] Smer prehľadávania je do šírky. Túto triedu konvertujeme z primitívnej na definovanú triedu. Pamäťová náročnosť s počtom uzlov rastie exponenciálne.



Obrázok 17: Prehľadávanie do šírky [vlastné spracovanie]

3.3.2 Prehľadávanie do hĺbky

Algoritmus Prehľadáva stromové alebo dátové štruktúry, pričom z počiatočného uzla postupuje čo najhlbšie po jednej vetve. Výhodou je, že jeho pamäťová náročnosť s počtom uzlov rastie lineárne. [32]



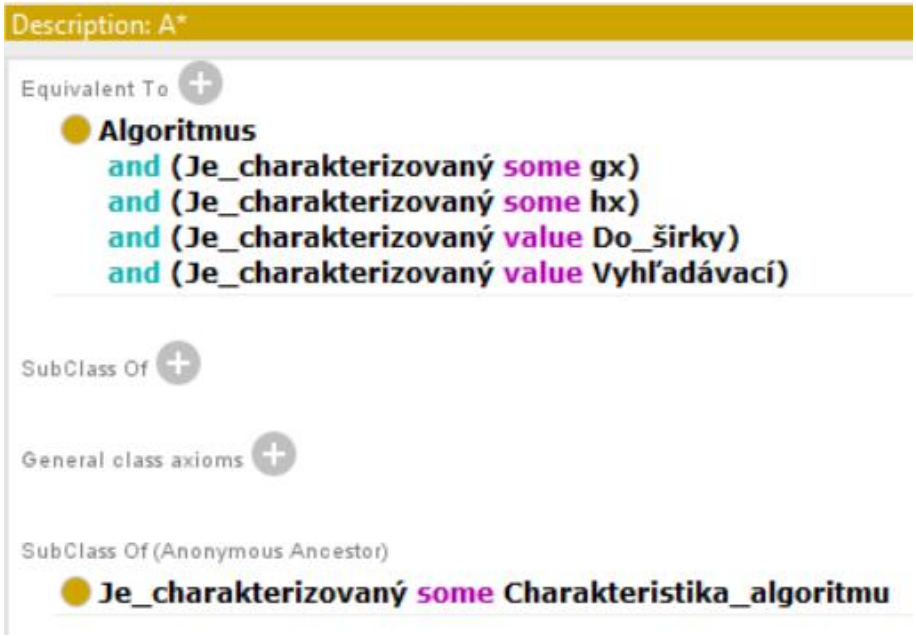
Obrázok 18: Prehľadávanie do hĺbky

3.3.3 A*

A* je algoritmus, ktorý je vyhľadávacím algoritmom a prehľadáva smerom do šírky. Jeho cieľom je nájsť najlacnejšiu cestu v počiatočného do cieľového uzla grafu. Udržuje si strom ciest a tieto cesty rozširuje o jednu hranu, kým sa nedostane do cieľového uzla. [33]

V každom kroku je potrebné určiť, z ktorého bodu sa bude rozširovať. Toto rozširovanie ja založené na heuristickej funkcii, f(x) = g(x) + h(x), kde x reprezentuje uzol v grafe, g(x) je cena cesty do tohto uzla a h(x) je odhad ceny cesty z uzla x do cieľového uzla. [33]

V systéme Protégé zaznamenáme skutočnosti, že A* algoritmus je podtriedou algoritmu. Tiež vieme, že má nejakú hodnotu g(x) a h(x), primárny smer prehľadávania je do šírky a patrí k vyhľadávacím algoritmom. Prijmime predpoklad, že toto sú nevyhnutné podmienky, aby konkrétny algoritmus bol algoritmom A*. Tak isto tieto podmienky vnímame ako dostačujúce a preto triedu algoritmov konvertujeme na definovanú triedu.



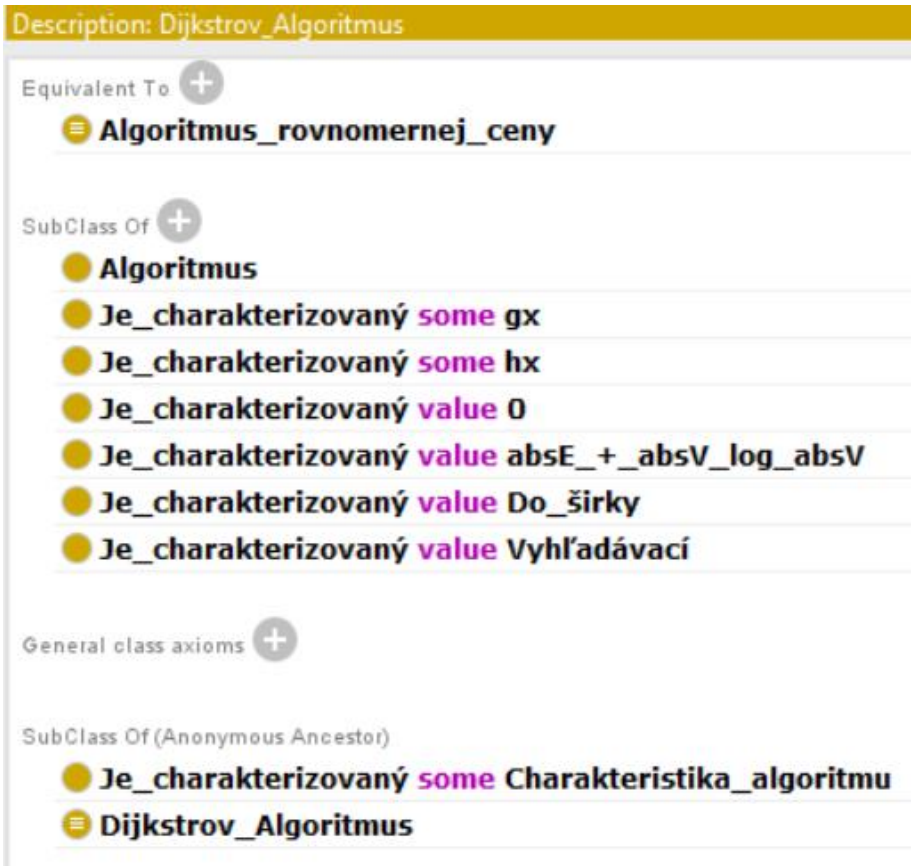
Obrázok 19: Algoritmus A*

3.3.4 Dijkstrov Algoritmus

Dijkstrov algoritmus je špeciálnym typom algoritmu A*, kedy v heuristickej funkcii odhad h(x) postavíme rovný nule. Používa sa pre nájdenie najkratšej cesty medzi dvoma bodmi grafu, pričom prehľadávanie grafu je vykonávané primárne smerom do šírky. Jeho zložitosť je |E|+|V|*log|V|, kde E reprezentuje počet hrán grafu a V počet uzlov grafu. [34] V niektorých oblastiach je Dijsktrov algoritmus známy aj ako algoritmus rovnomernej ceny.

V Protégé zaznamenáme smer prehľadávania, definujeme, že má heuristickú funkciu zloženú z hx a gx, kde ale zadefinujeme aj to, že hx sa rovná 0. ďalšou charakteristikou, ktorú definujeme je najhorší výkon dosiahnutý algoritmom.

Okrem triedy Dijsktrov algoritmus vytvárame aj triedu Algoritmus rovnomernej ceny. Jediný rozdiel medzi nimi je iba v názve. Toto v Protégé zaznamenáme tak, že v sekcii Equivalent to vyberieme triedu, s ktorou je algoritmus eqivalentný. Definícia Disjkstrovho algoritmu v Protégé potom vyzerá nasledovne.

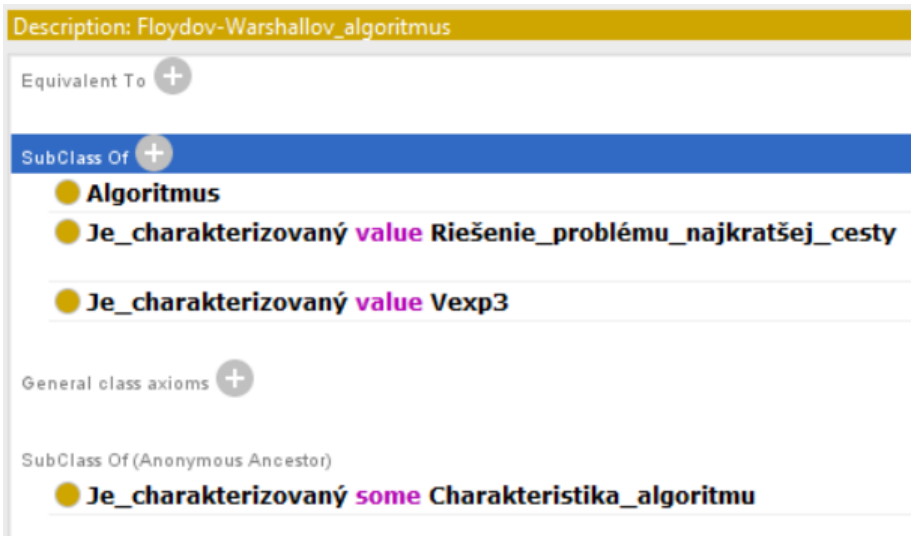


Obrázok 20: Dijkstrov algoritmus [vlastné spracovanie]

3.3.5 Floydov-Warshallov algoritmus

Floydov-Warshallov algoritmus slúži na nájdenie najkratšej vzdialenosti v ohodnotenom grafe. Vykonaním algoritmu získame najkratšie vzdialenosti medzi všetkými vrcholmi grafu, ale informácie o ceste nie sú dostupné. Informáciu o ceste medzi vrcholmi získavame úpravou algoritmu. Je príkladom dynamického programovania. Rozdeľuje úlohu na menšie úlohy a potom kombinuje riešenia pre vyriešenie pôvodnej úlohy. Jeho komplexita je $O(|V|^3)$, kde V je počet vrcholov v grafe. [35]

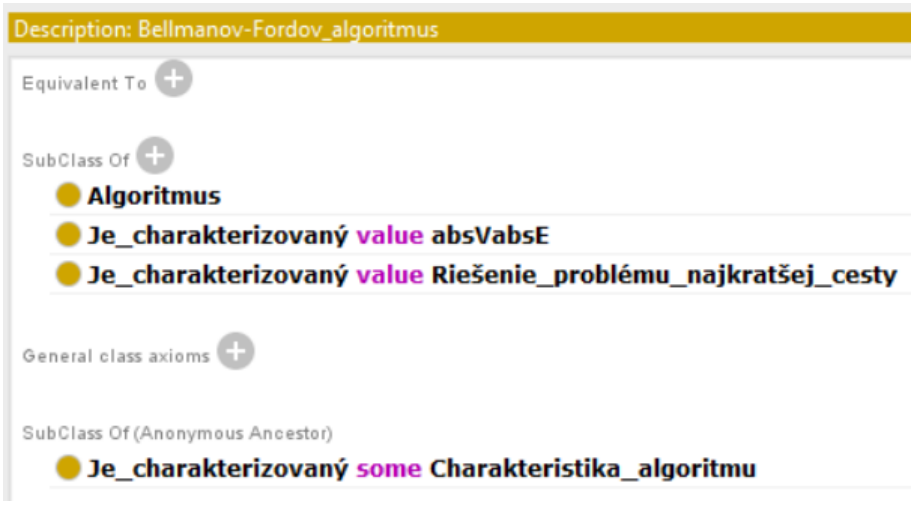
V protégé zaznamenáme jeho komplexitu a to, že sa používa pri riešení problémov najkratšej cesty. Zároveň sme vytvorili novú triedu algoritmov Riešenie problémov najkratšej cesty, ktorá je definovanou triedou. Opis triedy Floydov-Warshallov algoritmus je zobrazený nižšie.



Obrázok 21: Floydov-Warshallov algoritmus [vlastné spracovanie]

3.3.6 Bellmanov-Fordov algoritmus

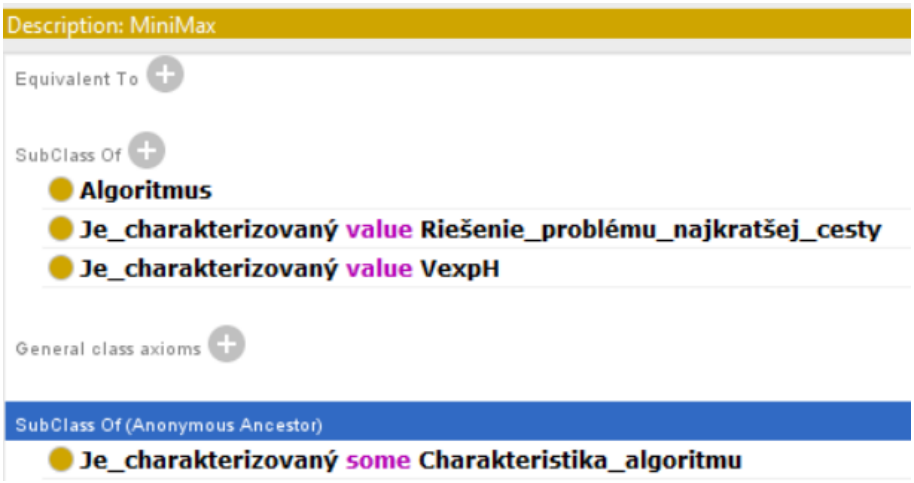
Bellmanov-Fordov algoritmus je algoritmus, ktorý dokáže vypočítať najkratšie cesty z jedného vrcholu k všetkým ostatným vrcholom v ohodnotenom grafe. Jeho trvanie je dlhšie, ako Dijkstrov algoritmus, pri použití na ten istý problém, ale je to cenou za to, že dokáže pracovať aj s grafmi, v ktorom sú niektoré ohodnotenia hrám záporné. Jeho náročnosť je O(|V|*|E|), kde V je počet vrcholov v grafe a E je počet hrán v grafe. [36] Nižšie je zobrazená jeho implementácia v Protégé.



Obrázok 22: Bellmanov-Fordov Algoritmus {vlastné spracovanie]

3.3.7 *Algoritmus Minimax*

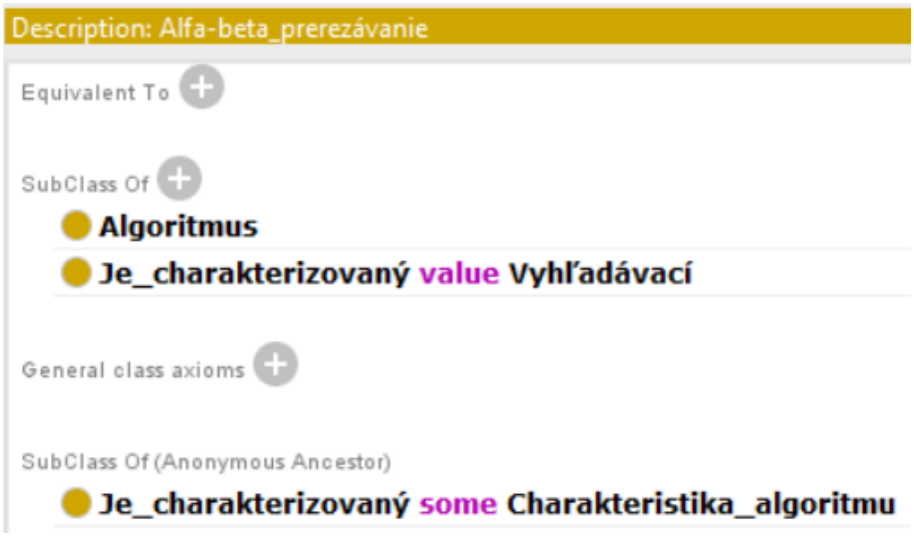
Minimax je algoritmus pre prechádzanie stromu a minimalizáciu maximálnych možných strát. Tvorí základ množstva počítačových hier. Typickou úlohou je nájsť najlepší ťah z danej pozície. Jeho náročnosť je ovplyvnená vetviacim faktorom, ktorý predstavuje priemerný počet vetiev vychádzajúcich z jedného uzla, označeným v a hĺbkou vetvenia, označenou h. Hodnotu potom vypočítame zo vzťahu $O(v^h)$. [37]



Obrázok 23: Algoritmus Minimax [vlastné spracovanie]

3.3.8 *Alfa-beta prerezávanie*

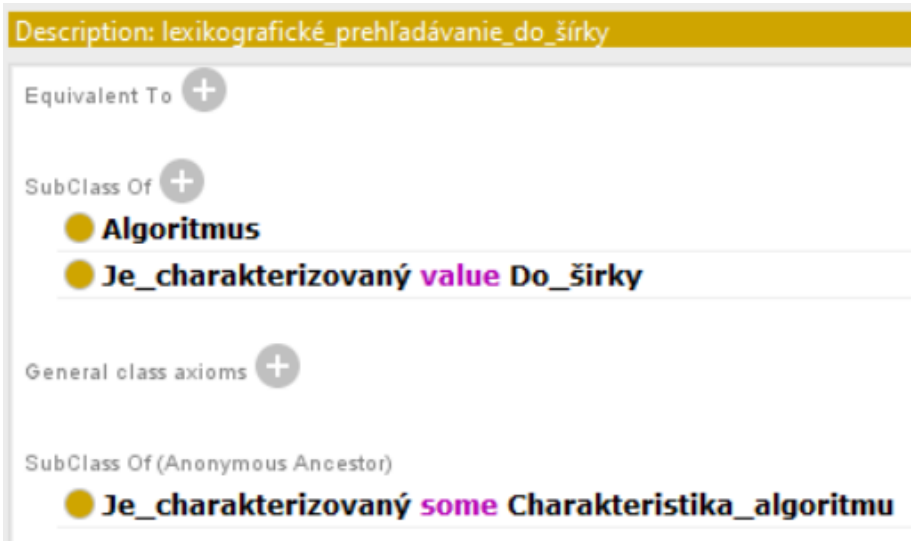
Alfa-beta prerezávanie je algoritmus pre vyhľadávanie a nadväzuje na minimax algoritmus tým, že sa snaží znížiť počet uzlov, ktoré sú ohodnotené minimax algoritmom v strome. Používa sa v počítačovom hraní hier pre dvoch hráčov, ako sú napríklad šachy. Prehľadávanie končí, ak práve vyhodnocovaný uzol je horší, ako uzol ohodnotený pred ním. Zložitosť je závislá, ad koeficientu vetvenia a hĺbky. Hodnota ja rovnaká, ako pri minimax algoritme, $O(v^h)$. [38]



Obrázok 24: Alfa-Beta prerezávanie [vlastné spracovanie]

3.3.9 *Lexikografické prehľadávanie do šírky*

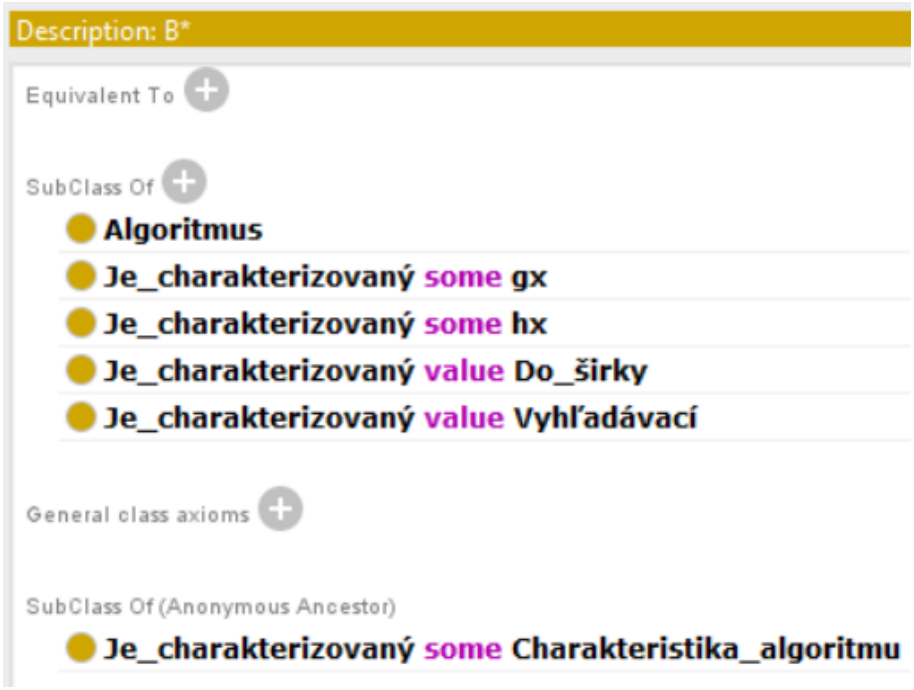
Činnosť algoritmu je odlišná ako pri algoritme prehľadávania do šírky. Slúži pre usporiadanie vrcholov grafu. Jeho náročnosť raste lineárne s počtom vrcholov v grafe. Lexikografické prehľadávanie do šírky nahradí frontu štandardného prehľadávania do šírky usporiadanou sekvenciou množín vrcholov. Sety v sekvenciách tvoria časť zostávajúcich vrcholov. V každom kroku sa odstráni vrchol z prvej skupiny v sekvencii a ak to spôsobilo, že množina je prázdna, tak sa zo sekvencie odstráni celý súbor a každý súbor v sekvencii je nahradený podmnožinami susediacich a nesusediacich vrcholov, pričom množina susediacich v sekvencii predbieha množinu nesusediacich uzlov. [39]



Obrázok 25: Lexikografické prehľadávanie do šírky [vlastné spracovanie]

3.3.10 B*

Algoritmus B* vychádza z algoritmu A*. Používa pri hľadaní optimálnej cesty medzi východiskovým a cieľovým uzlom. Každý uzol, ktorý je rozšírený má 2 hranice a to optimistickú a pesimistickú. V priebehu prehľadávania majú tieto hranice tendenciu konvergovať, čo vedie k ukončeniu prehľadávania. [40]



Obrázok 26: Algoritmus B* [vlastné spracovanie]

3.3.11 D*

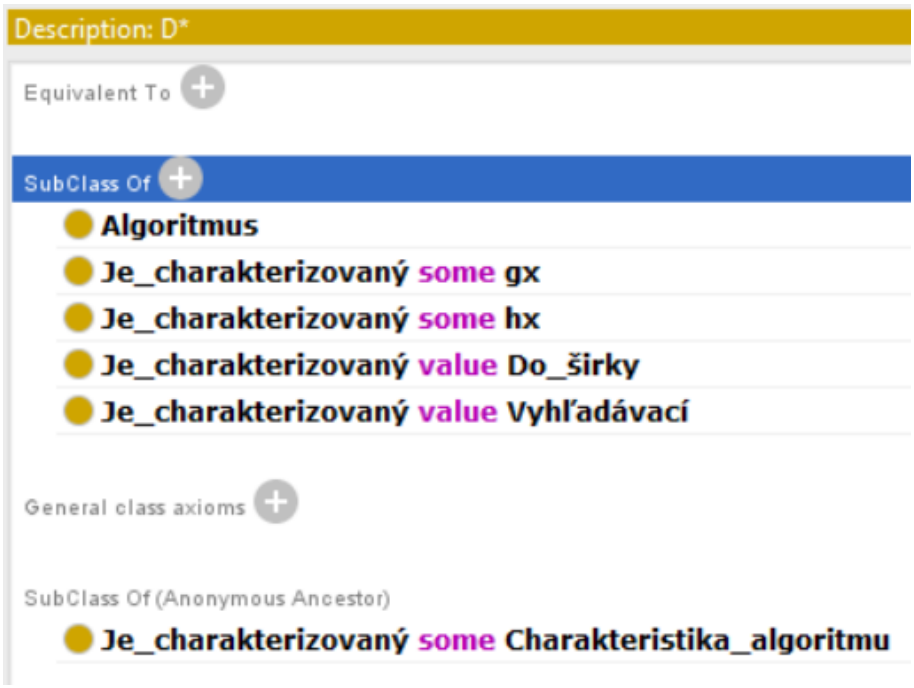
D* je vyhľadávací algoritmus a rozlišujeme tri varianty: [41]

- Pôvodný D*
- Sústredený D* - Informovaný inkrementačný vyhľadávací mechanizmus, ktorý vychádza z myšlienky A* a pôvodného D* a vznikol ďalším vývojom D*.
- D* lite je inkrementačné heuristické vyhľadávanie založené na inkrementačnej verzii algoritmu A*.

Tieto algoritmy riešia úlohy plánovania cesty, kde je potrebné sa dostať do cieľového uzla cez neznáme uzly. Vytvára predpoklady o neznámej časti grafu a hľadá najkratšiu cestu k cieľovému vrcholu prostredníctvom týchto predpokladov. Ak rozvíjaním nájde ďalšie uzly, pridá ich do svojej mapy a v prípade potreby nahradí novú najkratšiu cestu z aktuálneho uzla do cieľového uzla.. Proces sa opakuje, až kým sa nedosiahne cieľ, alebo

už nie je možné ďalej pokračovať. Pri dodržaní predpokladu, že cieľový uzol ostáva nemenný, sú tieto algoritmy efektívnejší, ako opakované prehľadávanie algoritmom A*. Inkrementačné algoritmy urýchľujú prehľadávanie použitím vzorov z predchádzajúcich hľadaní na aktuálny problém. [41]

Použitie týchto algoritmov je v navigácii robotov a autonómnych vozidiel. Väčšina súčasných systémov je založená na D* lite. [41]



Obrázok 27: Algoritmus D* [vlastné spracovanie]

3.3.12 Bronov-Keborchov algoritmus

Bronov-Keborschov algoritmus sa používa pre nájdenie maximálneho počtu slučiek v neorientovanom grafe. Nájde všetky podmnožiny vrcholov, z ktorých je každá dvojica v rámci podmnožiny prepojená hranou a spolu tvoria uzavretú slučku a zároveň žiadna podmnožina nemôže obsahovať ďalší vrchol tak, aby spolu tvorili uzavretú slučku. [42]

Je to algoritmus, ktorého zložitosť je závislá od počtu výstupov. Každý uzol grafu je súčasťou najviac $3^{n/3}$ slučiek, kde n je počet vrcholov v grafe. So použitím pivotovej stratégie, s pomocou ktorej sa minimalizuje počet rekurzívnych volaní uskutočňovanom v každom kroku je zložitosť $O(3^{n/3})$. [42]

V popise algoritmu je vidieť odlišné použitie algoritmu, ako sme mali doteraz a to pre nájdenie maximálneho počtu slučiek v grafe. Protégé pridávame preto inštanciu do triedy Typ grafu a pomenujeme ju Nájdenie maximálneho počtu slučiek v grafe. Taktiež môžeme vytvoriť Podtriedu triedy Algoritmus, ktorá sa bude nazývať Algoritmy hľadajúce

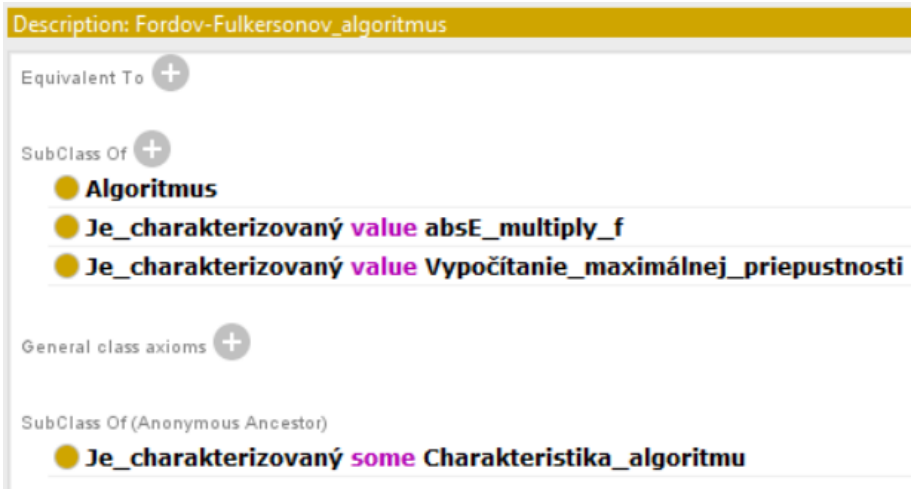
maximálny počet slučiek, ktorá bude mať obmedzenie, že algoritmus musí mať práve definovaný typ algoritmu a konvertujeme ju na definovanú triedu a tým pádom bude združovať algoritmy, ktoré slúžia pre výpočet maximálneho počtu slučiek v grafe.

Obrázok 28: Bronov-Keborchov algoritmus [vlastné spracovanie]

3.3.13 Fordov-Fulkersonov algoritmus

Je to algoritmus, ktorý počíta maximálny tok tokovou sieťou. Toková sieť je orientovaný graf, kde každá hrana je popísaná svojou kapacitou, ktorá nesmie byť prekročená. Platí pravidlo, že vstupný tok sa rovná výstupnému toku uzla, pokiaľ sa nejedná a počiatočný alebo cieľový uzol Zložitosť tohto algoritmu je O(|E|*f), kde E je počet hrán grafu a f je maximálna kapacita grafu. [43]

Opäť sa jedná a nový typ algoritmu a to algoritmus pre nájdenie maximálnej priepustnosti. V Protégé vytvoríme takúto triedu a pridáme jej obmedzenie, že jedinec tejto triedy musí mať atribút typ algoritmu rovný vypočítanie maximálnej priepustnosti. Táto trieda je zároveň definovanou triedou.

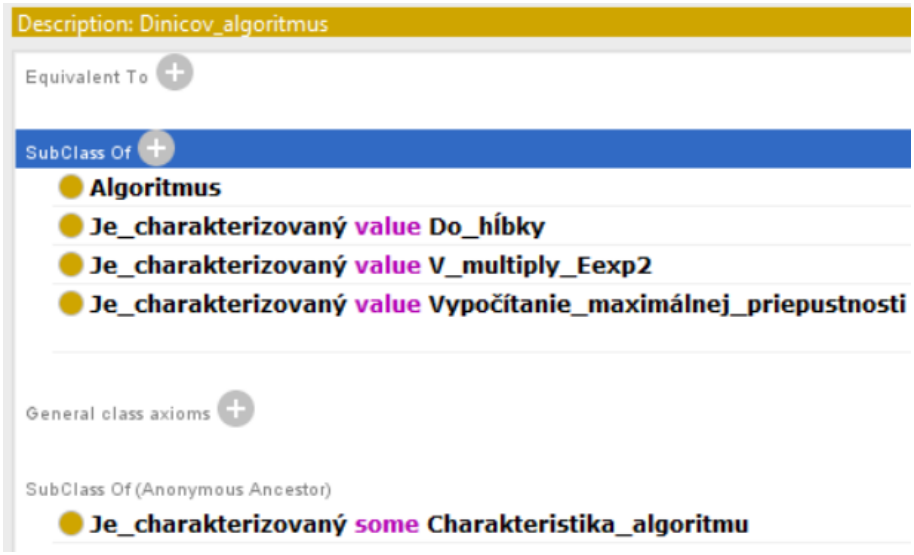


Obrázok 29: Fordov-Fulkerosonov Algoritmus [vlastné spracovanie]

3.3.14 *Dinicov algorimus*

Tento algoritmus sa používa pri hľadaní maximálneho možného toku v grafe. Jeho primárny smer prehľadávania grafu je do hĺbky. Vždy keď nájde väčšiu priepustnosť, tak sa posunie týmto smerom. Ak narazí na slepú uličku, tak vymaže poslednú hranu, vráti sa späť a pokračuje iným smerom. Jeho zložitosť je $O(V*E^2)$. [44]

V Protégé o ňom zaznamenáme, že jeho smer prehľadávania je do hĺbky, je určený pre výpočet maximálnej priepustnosti grafu a nakoniec zaznamenáme jeho zložitosť.

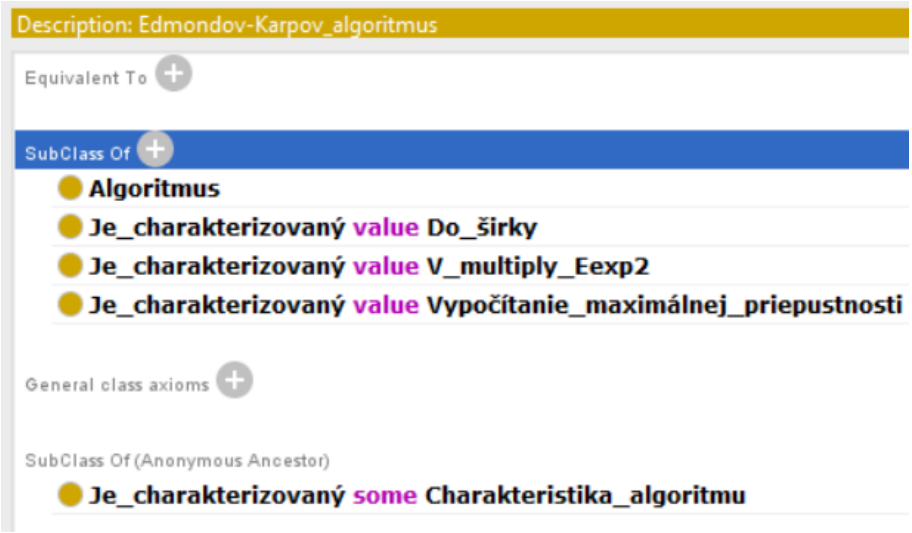


Obrázok 30: Dinicov algoritmus [vlastné spracovanie]

3.3.15 *Edmondov-karpov algoritmus*

Edmondov-Karpov algoritmus je špecifickou implementáciou Fordovho-Fulkersonovho algoritmu. Rozdielom je to, že pri Edmondovo-Karpovom algoritme je poradie prehľadávania dobre definované. Rozdielom je to, že ďalšiu cestu si volí spôsobom prehľadávania do šírky. Ak existuje niekoľko rozširujúcich ciest, tak si vyberá najkratšiu smerom k cieľovému uzlu. Jeho zložitosť je $O(V*E^2)$. [45]

V Protégé zaznamenáme, že sa jedná o algoritmus, ktorý hľadá maximálnu, priepustnosť v grafe, že jeho prehľadávanie primárne postupuje smerom do šírky a tiež definujeme jeho zložitosť.



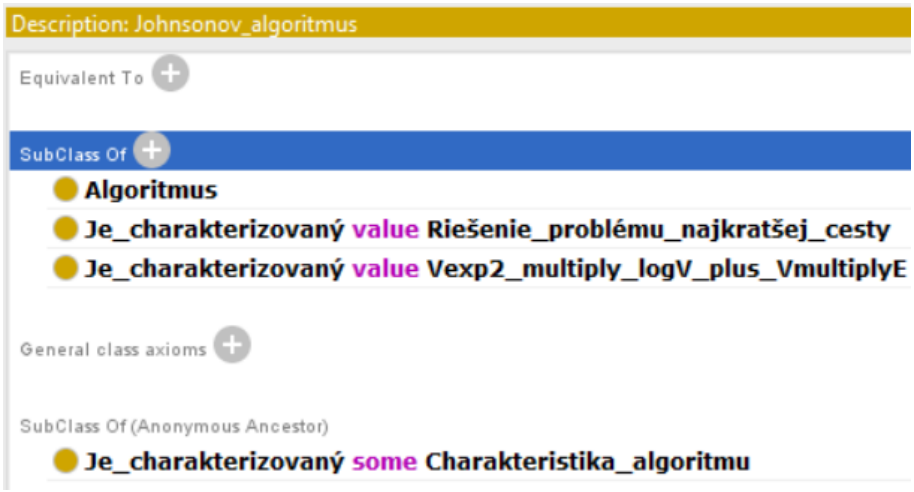
Obrázok 31:Edmondov-Karpov algoritmus [vlastné spracovanie]

3.3.16 Johnsonov algoritmus

Je to algoritmus, ktorý rieši problém najkratšej cesty, teda hľadá najkratšiu cestu medzi všetkými vrcholmi grafu. Je veľmi podobný Floydovmu-Warshallovmu algoritmu. Keď tieto dva algoritmy porovnáme, tak zistíme, že Floydov-Warshallov algoritmus je účinnejší pri grafoch, ktoré obsahujú veľké množstvo hrán. Naopak Johnsonov algoritmus je účinnejší pri riedkych grafoch, teda takých, ktoré obsahujú malá množstvo hrán. Jeho zložitosť je $O(V^2*\log(V) + V*E)$. [46]

Johnsonov algoritmus je zaujímavý, pretože používa ďalšie dva algoritmy hľadania najkratšej cesty, ako svoje podprogramy. Sú to Bellmanov-Fordov algoritmus, aby sa prehodnotil vstupný graf, kvôli eliminovaniu záporných hrán a detekcii záporných slučiek. Tento nový graf potom vstupuje do Dijkstrovho algoritmu a výstupom je súbor najkratších ciest medzi všetkými vrcholmi pôvodného grafu. [46]

V protége vytvárame triedu Johnsonov algrotimus, ktorý je pomocou vlastnosti je charakterizovaný a obmedzením hodnoty špecifikovaný ako algoritmus hľadania najkratšej cesty a jeho zložitosť je $O(V^2*\log(V) + V*E)$.

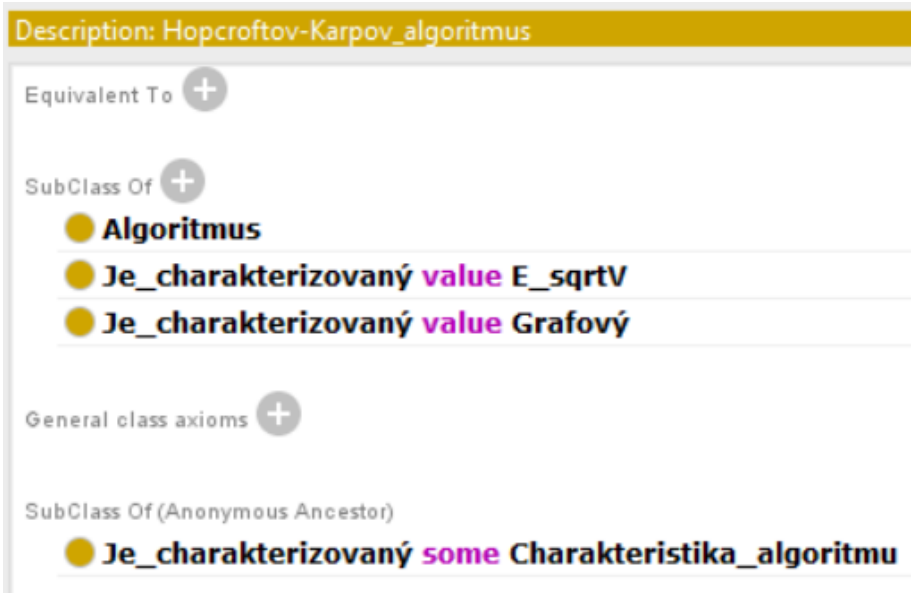


Obrázok 32: Johnsonov algoritmus [vlastné spracovanie]

3.3.17 Hopcroftov-Karpov algoritmus

Hopcroftov-Karpov algoritmus na výstupe poskytuje maximálnu zhodu kardinality grafu, teda množinu čo najväčšieho počtu hrán, ktoré sú charakteristické tým, že žiadne z nich nezdieľajú spoločný vrchol. Algoritmus opakovane zvyšuje veľkosť čiastočných zhôd stanovením rozširujúcich ciest. Hopcroftov-Karpov algoritmus nachádza maximálnu množinu najkratších rozširujúcich ciest v každej iterácii na navyšuje rozširujúcu cestu maximálnym tokom, namiesto toho, aby postupoval postupne rad radom. Jeho zložitosť je O(E√V). [47]

V praxi Hopcroftov-Karpov algoritmus nie je až tak efektívny. Je často prekonaný použitím prístupov prehľadávania do šírky, alebo prehľadávania do hĺbky. [47]

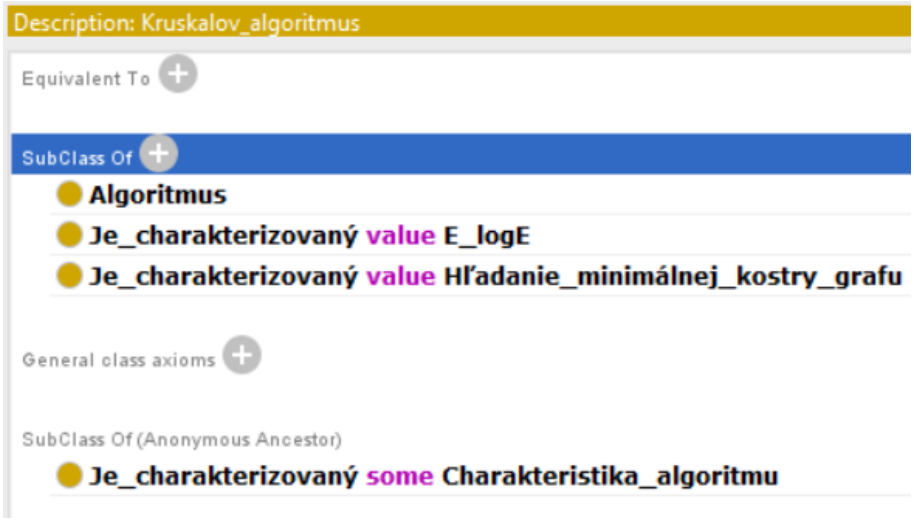


Obrázok 33: Hopcroftov-Karpov algoritmus [vlastné spracovanie]

3.3.18 *Kruskalov algoritmus*

Použitie Kruskalovho algoritmu nachádzame pri nájdení minimálnej kostry súvislého neorientovaného grafu, ktorý má ohodnotené hrany. Výstupom je kostra grafu, ktorá ma minimálne ohodnotenie hrán, spájajúca všetky vrcholy grafu. Algoritmus funguje takým princípom, že si usporiada všetky hrany a to do stromu ich pridáva postupne od hrany s najnižším ohodnotením a overí, či sa pridaním vrcholu nevytvorí slučka. Ak áno, tak sa táto hrana vynechá a tento krok sa opakuje, až kým strom neobsahuje V-1 hrán, kde V je počet vrcholov grafu. Zložitosť tohto algoritmu je O(E*log(E)). [48]

V Protégé evidujeme nový typ algoritmu a to algoritmy pre nájdenie minimálnej kostry grafu. Preto vytvoríme novú definovanú triedu a tiež do triedy typ algoritmu pridáme novú inštanciu.

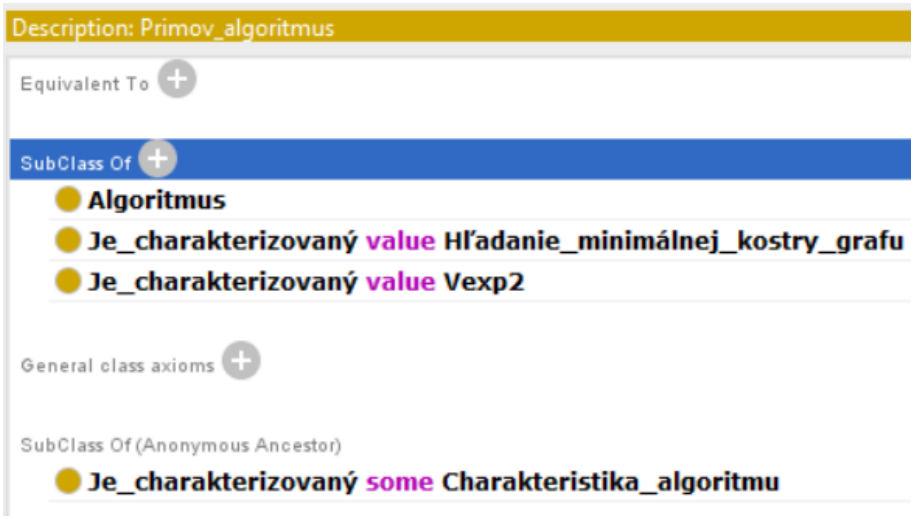


Obrázok 34: Kruskalov algoritmus [vlastné spracovanie]

3.3.19 *Primov algoritmus*

Primov algoritmus je taktiež algoritmom, ktorý nájde minimálnu kostru grafu. Funguje tak, že si udržuje 2 množiny vrcholov. Jedna obsahuje vrcholy, ktoré už sú súčasťou minimálnej kostry grafu a druhá obsahuje vrcholy, ktoré ešte nie sú zahrnuté. V každom kroku berie hranu s najmenším ohodnotením, ktorá spája tieto dve množiny a presunie vrchol do minimálnej kostry grafu, ktorý je spojený touto hranou. Jeho zložitosť je $O(V^2)$. [49]

V Protégé vytvárame triedu Primov algoritmus, ktorá je definovaná tým, že tieto algoritmy sa používajú pre nájdenie minimálnej kostry grafu a ich zložitosť je $O(V^2)$. Ako jedinca triedy chápeme implementáciu algoritmu.



Obrázok 35: Primov algoritmus [vlastné spracovanie]

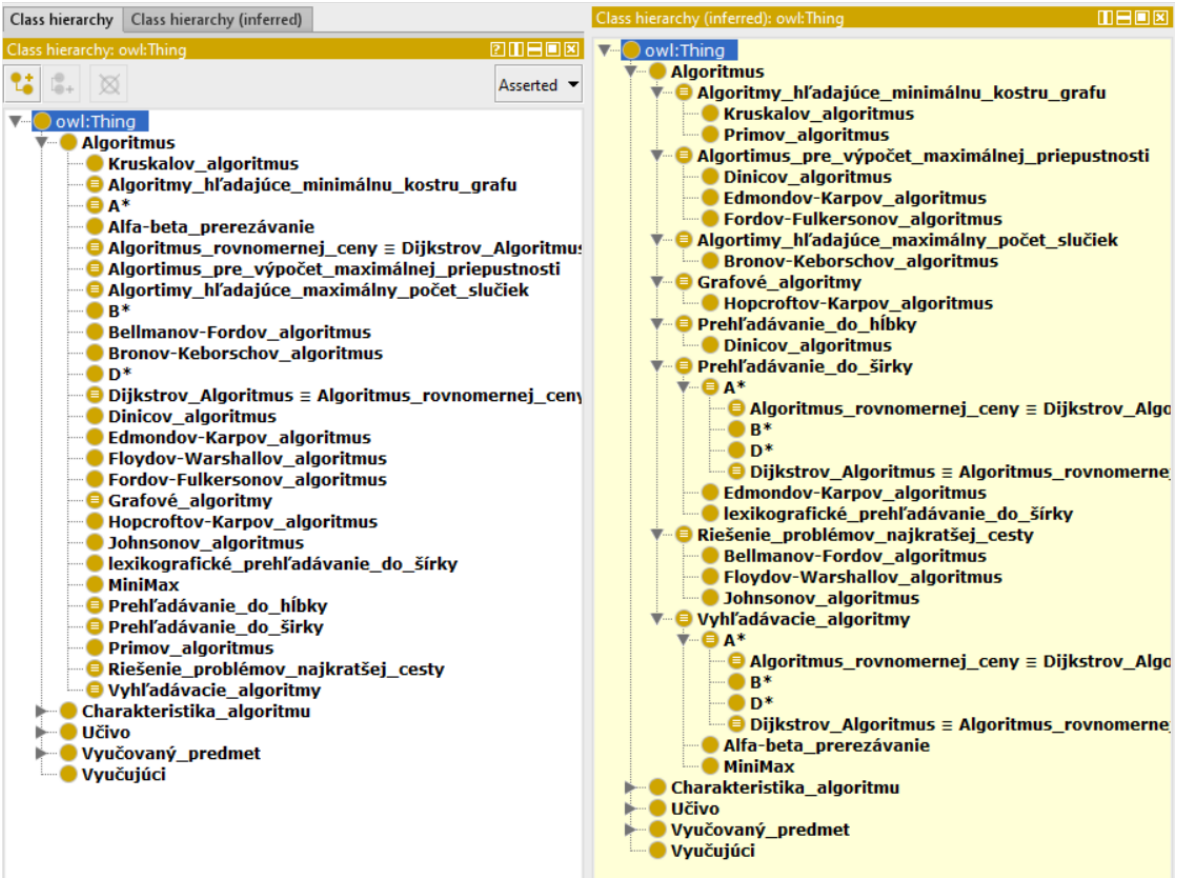
3.4 Odvodzovanie v systéme Protégé

Doteraz sme zbierali a analyzovali informácie o vyučovanom predmete, kde sme sa zamerali na časť prehľadávania stavového priestoru a to konkrétne na jednotlivé algoritmy. Všimnime si, že všetky triedy algoritmov sme vytvárali ako podtriedy triedy Algoritmus. Už z opisu niektorých algoritmov je zrejmé, že majú niektoré spoločné črty s inými algoritmami. Táto hierarchia nie je zlá, ale môžeme uvažovať o tom, či by mohla byť ešte lepšia. V praxi sa často stáva, že ontológiu nevytvára jeden používateľ, ale pracuje na nej viacero ľudí a ako používateľ si nemusím všimnúť, alebo ani nepotrebujem vedieť, čo vykonali ostatní používatelia.

Systém Protégé obsahuje odvodzovací mechanizmus, ktorý z nami vytvorených poznatkov dokáže odvodiť nové poznatky. My používame odvodzovací mechanizmus HermiT vo verzii 1.3.8.413. Je to odvodzovací mechanizmus pre ontológie napísané v jazyku OWL. Jeho základnými prínosmi sú, že dokáže zistiť, či je naša ontológia konzistentná a identifikovať vzťahy medzi jednotlivými triedami.

HermiT je verejne dostupný odvodzovací mechanizmus dostupný na stránkach http://www.hermit-reasoner.com a ako prvý je založený na „hyperbleau“ kalkulácii, a preto je oveľa rýchlejším odvodzovaním, ako pred ním používané mechanizmy. Veľké ontológie často potrebovali hodiny pre klasifikáciu a s použitím HermiT odvodzovania sa táto doba skrátila rádovo na sekundy. [50] HermiT používa priamu sémantiku a kontroluje všetky OWL2 testy pre priame sémantické odvodzovania.

V Protégé v záložke Reasoner zvolíme HermiT a spustíme odvodzovanie. Tým sa spustila automatická klasifikácia a vytvoril sa nový odvodený strom tried. Na obrázku nižšie vidíme nami vytvorenú hierarchiu tried a odvodenú hierarchiu tried HermiTom.



Obrázok 36: Pôvodné a odvodené triedy [vlastné spracovanie]

Na obrázku vidíme, že algoritmy už nie sú všetky na jednej úrovni v hierarchii, ale trieda Algoritmus už obsahuje aj hlbšie úrovne v hierarchii. Všimnime si napríklad triedu Vyhľadávacie algoritmy. My sme o nej uviedli, že je to typ triedy, pre vyhľadávanie, čo chápeme ako algoritmy, ktoré hľadajú najkratšiu cestu medzi počiatočným a cieľovým vrcholom. Túto triedu sme konvertovali na definovanú triedu, čím sme definovali, že táto podmienka je nielen nevyhnutnou, ale aj postačujúcou podmienkou. Preto ako podtriedy tejto triedy vidíme všetky triedy algoritmov, ktoré majú definovaný typ algoritmu vyhľadávací.

Ďalším zaujímavým výsledkom je, že jedna trieda nemusí byť podtriedou iba jednej triedy. Všimnime si triedu A*, ktorá je podtriedou vyhľadávacích algoritmov a aj algoritmov prehľadávania do šírky, pretože spĺňa postačujúce podmienky týchto obidvoch tried.

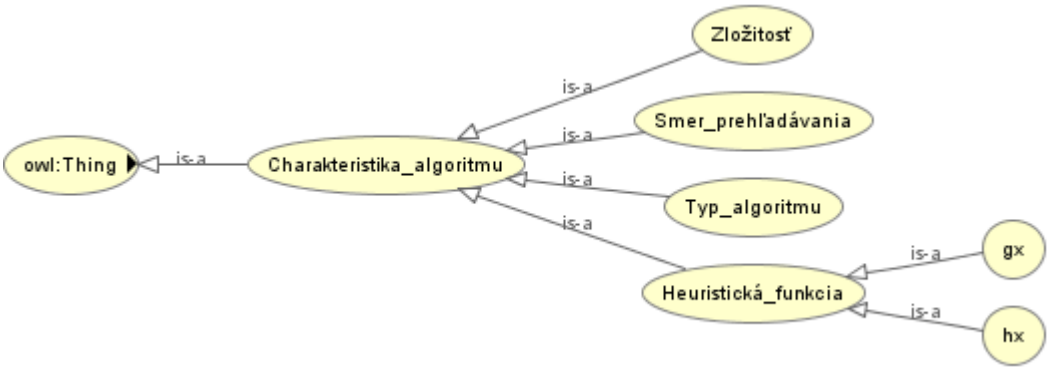
Uvažujem o prípade, keď sme spravili chybu pri definovaní tried. V takomto prípade aj odvodzovaním môžeme získať chybné výsledky, V praxi však vytváranie ontológií funguje v tíme, kde prebieha viacúrovňová kontrola, kde každá úprava ontológie je skontrolovaná ďalšou osobou.

Tu nachádzame prínos odvodenia nových poznatkov o algoritmoch. Za bežných okolností by mohol vyučujúci vyučovať jednotlivé algoritmy postupne, ako sú uvedené v nami vytvorenej ontológii, ale po odvodení poznatkov vidíme, že niektoré algoritmy spolu súvisia a preto je vhodné ich zaradiť vo výučbe spolu ako tematickú časť.

3.5 Vizualizácia ontológie

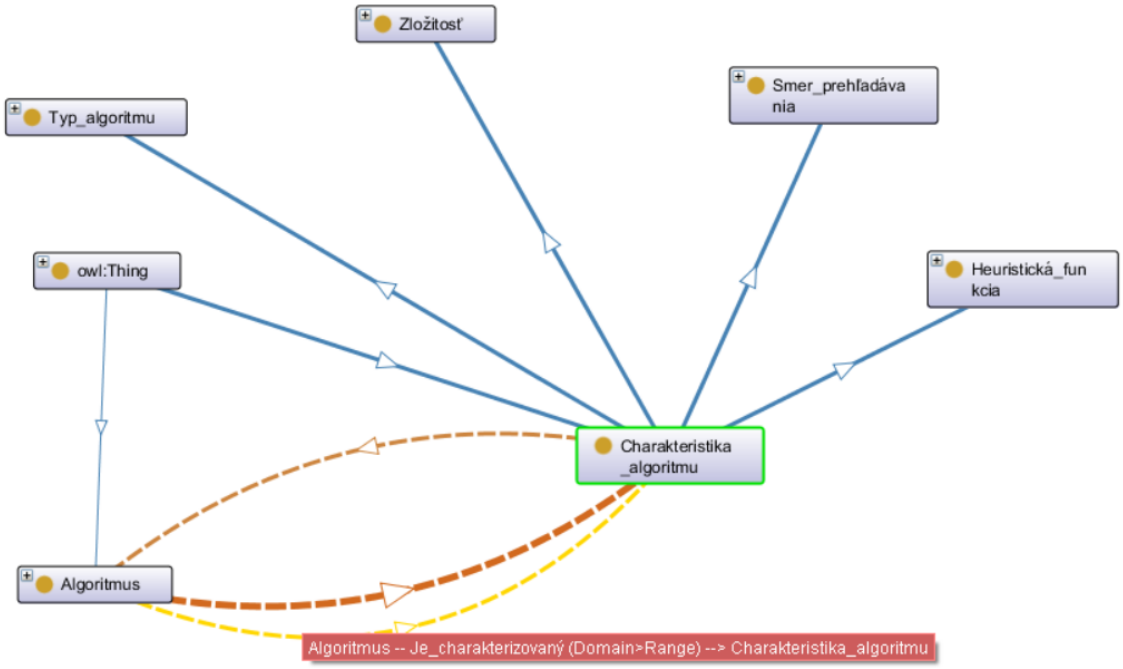
Nami vytvorená hierarchia tried je zobrazená ako viacúrovňový zoznam. Niekedy je prehľadnejšie zobrazenie v grafickej podobe, ako strom tried. Priamo v Protégé sú na toto určené dve rozšírenia OWLViz a OntoGraf.

Po zobrazení OWLViz ako samostatnej karty je na ľavej strane zobrazená hierarchická štruktúra tried. Po zvolení konkrétnej triedy sa nám táto trieda zobrazí spolu s jej nadradenou triedou a v opačnom smere sa zobrazia všetky jej dcérske triedy. Užitočnou funkcionalitou je export vo forme obrázka. Z dôvodu prehľadnosti toto zobrazenie demonštrujeme na triede Charakteristika algoritmu, pretože v triede algoritmus je veľa dcérskych tried a zobrazenie by bolo nečitateľné. V aplikácii je možné použiť lupu pre priblíženie a oddialenie grafu.



Obrázok 37: OWLViz [vlastné spracovanie]

Všimnime si, že OWLViz zobrazuje iba vzťahy nadradenosti a podradenosti triedy. V ontológii, sú ale dôležité vzťahy, ktorými sú poprepájané jednotlivé inštancie tried a toto už OWLViz zobraziť nedokáže. Ak chceme zobraziť aj nami definované vzťahy medzi triedami, tak je vhodné použiť rozšírenie Ontograf. Na obrázku nižšie je zobrazená trieda Charakteristika algoritmu s vyjadrením hierarchie tried, ale aj vzťahov medzi triedami.



Obrázok 38: OntoGraf [vlastn0 spracovanie]

Záver

Výsledkom práce je vytvorená ontológia predmetu Umelá inteligencia, ktorá je zameraná na algoritmy prehľadávania stavového priestoru. Takéto ontológia môže prispieť k skvalitneniu tvorby študijných predmetov.

V prvej časti je opísaný stav problematiky. Venujeme sa téme poznatkov a práce s nimi. Najprv sme popísali životný cyklus znalostí. Ďalej sme sa venovali znalostnému manažmentu. V súvislosti so znalosťami sa venujeme aj formám reprezentácie poznatkov a ontológiám. V rámci ontológií sme popísali aj metodiky vývoja ontológii a možnosť odvodzovania nových poznatkov v ontológiách.

Druhej časti sme popísali cieľ práce a použitú metodiku. Určili sme, že vytvárame ontológiu v jazyku OWL v prostredí Protégé, definovali obsah ontológie a použitú metodológiu pri vývoji ontológie.

V Praktickej časti sa venujeme popisu prostredia Protégé, ktorého použitie nachádzame vo vývoji ontológií a popísali jeho jednotlivé funkcionality a zároveň sme vytvárali základnú štruktúru nami vytváranej ontológie.

Ďalej sme sa zaoberali získavaním a analýzou informácií o algoritmoch používaných na prehľadávanie stavového priestoru, čo je jedným vyučovaným okruhom v predmete Umelá inteligencia. Evidovali sme rôzne vlastnosti algoritmov, čím postupne niektoré algoritmy zdieľali v ontológii rovnaké vlastnosti.

Na tento ontológii sme demonštrovali odvodzovanie a to prostredníctvom automatizovanej klasifikácie, kde sme poukázali na novo odvodenú hierarchickú štruktúru tried, ktorá môže byť akýmsi vodítkom, kde do výučby zahrnúť jednotlivé algoritmy.

V poslednou témou praktickej časti, ktorej sme sa venovali bola vizualizácia ontológie, kde sme predviedli vizualizáciu prostredníctvom OWLViz a Ontograf.

Prínosom práce je najme poukázanie na použitie ontológií v rámci tvorby vyučovaných predmetov na vysokých školách. V praktickej časti bolo odvodzovaním ukázané, ako sa môže zmeniť štruktúra predmetu s použitím pôvodných poznatkov. Nami vytvorenú ontológiu je možné rozšíriť o ďalšiu tematickú časť predmetu Umelá inteligencia, ktorou sú reprezentácie poznatkov a logiky. Obdobným spôsobom je možné vytvoriť ontológie iných vyučovaných predmetov a následne tieto ontológie použiť pri vytváraní ontológie študijného programu a namodelovať závislosti nie len v rámci jedného predmetu,

ale aj medzi predmetmi navzájom, čim je možné vylepšiť nie len výučbu jednotlivých predmetov, ale aj štruktúru samotného študijného programu.

Zoznam použitej literatúry

[1] BELLINGER Gene, CASTRO Durval, MILLS Anthony. Data, Information, knowledge and Wisdom [elektronický zdroj]. 2014. [cit. 2018-10-10]. Dostupné na: https://s3.amazonaws.com/academia.edu.documents/

[2] BIRKINSHAN Julian. Managing the knowledge lifecycle [elektronický zdroj]. London Business School, 2002-04-16. [cit: 2018-10-11]. Dostupné na: http://facultyresearch.london.edu/docs/SIM14.pdf

[3] GOURLAY, Stephen. The SECI model of knowledge creation: some empirical shortcomings[elektronický zdroj]. Kingston Business School, 2003. [cit: 2018-10-13]. Dostupné na: https://eprints.kingston.ac.uk/2291/1/Gourlay%202004%20SECI.pdf

[4] DUBBERLY, Hugh; EVENSON, Shelley. Design as learning---or" knowledge creation"---the SECI model [elektronický zdroj]. *Interactions.acm.org*, 2010. [cit: 2018-10-15]. Dostupné na: http://interactions.acm.org/archive/view/january-february-2011/design-as-learning-or-knowledge-creation-the-seci-model1

[5] NICKOLS, Fred. The knowledge in knowledge management[elektronický dokument]. *The Knowledge Management Yearbook, 2000–2001*, 2000. [cit: 2018-10-18]. Dostupné na: https://books.google.sk/books?hl=sk&lr=&id=rCGH3EvRwnUC&oi=fnd&pg=PA12&dq=explicit+tacit+implicit+knowledge&ots=JHFZXbtkrf&sig=55df4GCCkevdE69aRYFzMiqW6pc&redir_esc=y#v=onepage&q=explicit%20tacit%20implicit%20knowledge&f=false

[6] NOY Natalya, MCGUINESS Deborah, Ontology Development 101: A Guide to Creating Your First Ontology [elektronický zdroj]. *Standford university.* Dostupné na: https://protege.stanford.edu/publications/ontology_development/ontology101.pdf

[7] Berners-Lee T, Hendler J, Lassila O. The semantic web. Scientific american [elektronický zdroj]. 2001. [cit: 2018-10-25]. Dostupné na: https://kask.eti.pg.gda.pl/redmine/projects/sova/repository/revisions/master/entry/doc/Master%20Thesis%20(In%20Polish)/materials/10.1.1.115.9584.pdf

[8] BOOCH, Grady, et al. The unified modeling language. *Unix Review*, 1996. [cit: 2018-11-02]. Dostupné na: https://www2.ccs.neu.edu/research/demeter/course/f96/readings/uml9.pdf

[9] BLÁZQUEZ, Juan, et al. Building ontologies at the knowledge level using the ontology design environment [elektronický zdroj]. 1998. [cit: 2018-11-05]. Dostupné na: http://oa.upm.es/6457/1/Building_Ontologies_at_the_K.pdf

[10] Integrated definition for ontology description capture method [elektronický zdroj. [cit 2018-12-10]. Dostupné na: http://www.idef.com/

[11] SMITH, Barry. Beyond concepts: ontology as reality representation [elektronický zdroj]. *Proceedings of the third international* conference *on formal ontology in information systems (FOIS 2004)*. IOS Press Amsterdam, 2004. str. 73-84. Dostupné na: https://philpapers.org/archive/SMIBCO

[12] METTREY, William. An assessment of tools for building large knowledge-based systems [elektronický zdroj]. *AI Magazine*, 1987. Dustupné na: https://www.aaai.org/ojs/index.php/aimagazine/article/view/625/558

[13] CHUNGOORA Tisch, Ontology from a logic based perspective. *Practical knowledge modeling.*

[14] What is Data Modelling? Conceptual, Logical, & Physical Data Models [elektronický zdroj]. Dostupné na: https://www.guru99.com/data-modelling-conceptual-logical.html#4

[15] MA, Zongmin. Fuzzy database modeling of imprecise and uncertain engineering information. Heidelberg*: Springer,* 2006. str. 134.

[16] Data Modelling Using EXPRESS-G for IFC Development [elektronický zdroj]. Dostupné na: https://pdfs.semanticscholar.org/0c32/aea4d2ba2c26b693536634967d11923da623.pdf

[17] MINSKY, Marvin. A framework for representing knowledge [elektronický zdroj]. 1974. [cit: 2018-12-20]. Dostupné na: https://dspace.mit.edu/bitstream/handle/1721.1/6089/AIM-306.pdf?%20sequence=2

[18] SCHANK, Roger C.; ABELSON, Robert P. Scripts, plans, goals, and understanding: An inquiry into human knowledge structures [elektronický zdroj]. *Psychology Press*, 2013. [cit:2018-12-25]. Dostupné na: https://dspace.mit.edu/bitstream/handle/1721.1/6089/AIM-306.pdf?%20sequence=2

[19] BARR, Avron; FEIGENBAUM, Edward A. (ed.). The handbook of artificial intelligence [elektronický zdroj]. Butterworth-Heinemann, 2014. [cit: 2019-01-02]. Dostupné na: https://books.google.sk/books?hl=sk&lr=&id=xP7iBQAAQBAJ&oi=fnd&pg=PP1&dq=the+handbook+of+artificial+intelligence&ots=KamX-X1rUd&sig=1vZEt3EQ-aHeD44UT5XfC3r9Nkk&redir_esc=y#v=onepage&q=the%20handbook%20of%20artificial%20intelligence&f=false

[20] BRACHMAN Roland, SCHMOLZE James. An overview of the KL-ONE Knowledge Represetnation System [elektronický zdroj]. *Cognitive Science, Volume 9, Issue 2.* Apríl-Jún 195. str. 171-216. Dostupné na: https://www.sciencedirect.com/science/article/abs/pii/S0364021385800148

[21] MACGREGOR, Robert M. Using a description classifier to enhance deductive inference [elektronický zdroj]. *The Seventh IEEE Conference on Artificial Intelligence Application*. 1991. Dostrupné na: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.37.5042&rep=rep1&type=pdf

[22] What is UML [elektronický zdroj]. [cit: 2019-01-10]. Dostupné na: https://www.uml.org/what-is-uml.htm

[23] CLARK, Anthony; EVANS, Andy. Foundations of the Unified Modeling Language [elektronický zdroj]. *Proceedigs of the 2nd Northern Formal Methods Workshop.* 1997. [cit. 2019-01-11] Dostupné na: http://shura.shu.ac.uk/11889/1/nfmw97.pdf

[24] CETA Noel. All You Need to Know About UML Diagrams: Types and 5+ Examples [elektronický zdroj]. [cit. 2019-01-11]. Dostupné na: https://tallyfy.com/uml-diagram/

[25] BIRCKLEY Dan, GUHA R.V. RDF Schema 1.1 [elektronický zdroj]. *W3C Recommendation February 2014.* 2014. [cit. 2019-01-15]. Dostupné na: https://www.w3.org/TR/2014/REC-rdf-schema-20140225/

[26] MCGUINESS Deborah L., HARMELEN Frank Van. OWL Web Ontology Language Overview [elektronický zdroj]. *W3*C Recommendation. 10.2.2004. Dostupné na: https://www.w3.org/TR/owl-features/

[27] PAN, Wen-lin; LIU, Da-xin. Mapping object role modeling into common logic interchange format [elektronický zdroj]. *3rd International Conference on Advanced Computer Theory and Engineering.* 2010. [cit. 2019-01-16]. Dostupné na: https://ieeexplore.ieee.org/abstract/document/5579141

[28] GENESERETH, Michael R. Knowledge interchange format-version 3.0: reference manual [elektronický zdroj]. 1992. [cit. 2019-01-17 Dostupné na: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.47.1022&rep=rep1&type=pdf

[29] SOWA, John F. Conceptual graphs summary [elektronický zdroj]. *Conceptual Structures: current research and practice*. 1992. [cit. 2019-01-20]. Dostupné na: http://www.jfsowa.com/cg/cgif.htm

[30] Proposition for the Common Logic Interchange Format [elektronický zroj]. 2005. [cit. 2019-01-20]. Dostupné na: https://www.ihmc.us/users/phayes/CLIF.html

[31] RAZNIEWSKI, Simon; SAVKOVIC, NUTT. Turning The Partial-closed World Assumption Upside Down [elektronický zdroj]. 2016. [cit. 2019-01-22]. Dostupné na: http://ceur-ws.org/Vol-1644/paper3.pdf

[32] ATALLAH, Mikhail J. Algorithms and theory of computation handbook. *CRC press*, 1998. [cit. 2019-02-01].

[33] AGRAWAL Siddhardth. Artificial Intelligence – A* search algorithm [elektronický zdroj]. 2013. [cit. 2019-02-03]. Dostupné na: https://algorithmicthoughts.wordpress.com/2013/01/04/artificial-intelligence-a-search-algorithm/

[34] ABIZ Thaddeus, PANG Hannah, WILLIANS Christopher. Dijksttra's Algorithm [elektronický zdroj]. [cit. 2019-02-10]. Dostupné na: https://brilliant.org/wiki/dijkstras-short-path-finder/

[35] CHUMBLEY Alex, MOORE Karleigh, YANG Jack. Floyd-Warshall Algrotthm [elektronický zdroj]. [cit. 2019-02-10]. Dostupné na: https://brilliant.org/wiki/floyd-warshall-algorithm/

[36] CHUMBLEY Alex, MOORE Karleygh, ROSS Eli. Bellman-Ford Algorithm [elektronický zdroj]. [cit. 2019-02-11]. Dostupné na: https://brilliant.org/wiki/bellman-ford-algorithm/

[37] Algorithms – Minimax [elektronický zdroj]. *Strategies and tactics for intelligent search.* [cit. 2019-02-15]. Dostupné na: https://cs.stanford.edu/people/eroberts/courses/soco/projects/2003-04/intelligent-search/minimax.html

[38] Alpha-Beta [elektronický zdroj]. *Chess Programming WIKI.* [cit. 2019-02-20]. Dostupné na*:* https://www.chessprogramming.org/Alpha-Beta

[39] CORNEIL Derek G. Lexicographical Breadth First Search – A Survey[elektronický zdroj]. 2004. [cit. 2019-02-22]. Dostupné na: https://www.semanticscholar.org/paper/Lexicographic-Breadth-First-Search-A-Survey-Corneil/d4b5a492f781f23a30773841ec79c46d2ec2eb9c

[40] BERLINER Hans. The B* tree search algorithm: A best=first proof procedure [elektronický zdroj]. *Artificial Intelligence Volume 12, Issue 1.* 1979. [cit. 2019-02-22]. Dostupné na: https://www.sciencedirect.com/science/article/abs/pii/0004370279900031

[41] FERGUSON Dave, STENTZ Anthony. The field D* Algorithm for Improved Path Planning and Replanning un Uniform and Non-Uniform Cost Enviroments [elektronický zdroj]. *Carnegie Mellon University*. [cit. 2019-02-28]. Dostupné na: https://pdfs.semanticscholar.org/58f3/bc8c12ee8df30b3e9564fdd071e729408653.pdf

[42] CAZALS F., KARANDE C. A note on the problem of reporting maximal cliques [elektronický zdroj]. 2008. [cit. 2019-03-10]. Dostupné na: https://www.sciencedirect.com/science/article/pii/S0304397508003903

[43] HOR Tim, MOORE Karleigh, CHUMBLEY Alex. Ford-Fulkerson Algorithm[elektronický zdroj]. [cit. 2019-03-10]. Dostupné na: https://brilliant.org/wiki/ford-fulkerson-algorithm/

[44] Dinic's Algorithm [elektronický zdroj]. [cit. 2019-03-15]. Dostupné na: https://algods.fandom.com/wiki/Dinic%27s_Algorithm

[45] TOMS Pats, CHUMBLEY Alex, CHATS Shaurya. Edmonds-Karp Algorithm [elektronický zdroj]. [cit. 2019-03-17]. Dostupné na: https://brilliant.org/wiki/edmonds-karp-algorithm/

[46] KHIM Jimin, MOORE Karleigh, CHUMBLEY Alex. Johnson's Algorithm [elektronický zdroj]. [cit. 2019-03-20]. Dostupné na: https://brilliant.org/wiki/johnsons-algorithm/

[47] KHIM Jimin, LANDMAN Nathan, MOORE Karleigh. Hopcroft-Karp Algorithm [elektronický zroj]. [cit. 2019-03-28]. Dostupné na: https://brilliant.org/wiki/hopcroft-karp/

[48] Kruskal's Minimum Spanning Three Algorithm | Greedy Algo-2 [elektronický zdroj]. [cit. 2019-04-01]. Dostupné na: https://www.geeksforgeeks.org/?p=26604/

[49] Prim's Algorithm [elektronický zdroj]. [cit. 2019-04-01]. Dostupné na: https://www.programiz.com/dsa/prim-algorithm

[50] HermiT OWL Reasoner [elektronický zdroj]. [cit. 2019-04-01]. Dostupné na: http://www.hermit-reasoner.com/index.html