EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

Evidenčné číslo: 103004/I/2021/421000077878

VYUŽITIE ONTOLOGICKÉHO MODELOVANIA NA TVORBU DOMÉNOVEJ ONTOLÓGIE PROGRAMOVACÍCH JAZYKOV

Diplomová práca

2021 **Róberta Červenková**

EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

VYUŽITIE ONTOLOGICKÉHO MODELOVANIA NA TVORBU DOMÉNOVEJ ONTOLÓGIE PROGRAMOVACÍCH JAZYKOV

Diplomová práca

Študijný program: Informačný manažment

Študijný odbor: Ekonómia a manažment

Školiace pracovisko: Katedra aplikovanej informatiky

Vedúci záverečnej práce: RNDr. Eva Rakovská, PhD.

Bratislava 2021 **Róberta Červenková**

Ekonomická univerzita v Bratislave Fakulta hospodárskej informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Róberta Červenková

Študijný program: Informačný manažment (Jednoodborové štúdium, inžiniersky II. st., denná forma)

Študijný odbor: 8. - ekonómia a manažment **Typ záverečnej práce:** Inžinierska záverečná práca **Jazyk záverečnej práce:** slovenský

Sekundárny jazyk: anglický

Názov: Využitie ontologického modelovania na tvorbu doménovej ontológie programovacích jazykov.

Anotácia: Práca sa zaoberá analýzou vlastností a využiteľnosti programovacích jazykov a ich klasifikáciou na základe signifikantných znakov. Na základe analýzy sa z množiny programovacích jazykov vyberú aktuálne a prostredníctvom ontologického modelovania sa vytvorí doménová ontológia, ktorá môže slúžiť ako základ na vyhľadávanie vhodného programovacieho jazyka pre rôzne účely.

Vedúci: RNDr. Eva Rakovská, PhD.

Katedra: KAI FHI - Katedra aplikovanej informatiky FHI
Vedúci katedry: Ing. Mgr. Peter Schmidt, PhD.

Dátum zadania: 28.10.2019

Dátum schválenia: 29.10.2019 Ing. Mgr. Peter Schmidt, PhD.

vedúci katedry

Poďakovanie

Chcela by som sa poďakovať vedúcej mojej diplomovej práce RNDr. Eve Rakovskej, PhD. za cenné rady, odborné vedenie, poskytnuté dokumenty a návrhy pri písaní diplomovej práce.

ABSTRAKT

ČERVENKOVÁ, Róberta: *Využitie ontologického modelovania na tvorbu doménovej ontológie programovacích jazykov.* – Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky; Katedra aplikovanej informatiky. – RNDr. Eva Rakovská, PhD.– Bratislava: FHI EU, 2021, 53 s.

Cieľom práce je vytvorenie ontologického modelu, ktorý poskytuje vzťahy a atribúty programovacích jazykov.

Práca sa zaoberá analýzou vlastností a využiteľnosti programovacích jazykov a ich klasifikáciou na základe signifikantných znakov. Na základe analýzy sa z množiny programovacích jazykov vyberú aktuálne a prostredníctvom ontologického modelovania sa vytvorí doménová ontológia, ktorá môže slúžiť ako základ na vyhľadávanie vhodného programovacieho jazyka pre rôzne účely.

Celá práca sa teda zaoberá ontológiou a programovacími jazykmi. Na záver sme vytvárali model vo vybranom editore.

Prvá kapitola za zaoberá ontológiou ako takou. Priblížili sme čitateľovi pojem ontológia, ktorý je mnohým ľudom stále neznámi, napriek tomu sa v dnešnej dobe veľmi rozširuje. Rovnako sme sa v tejto kapitole zamerali na členenie či atribúty ontológie. Vysvetlili sme si základné pojmy, ktoré sa v ontológii využívajú. Ako každý pojem aj samotná ontológia nesie so sebou určité problémy, ktoré sme si rovnako opísali v prvej kapitole. Nesmieme zabudnúť na konkrétne využitie ontológie v praxi.

Druhú kapitolu sme využili na opísanie našich cieľov a podcielov. Taktiež sme sa zamerali na metodiky práce a metódy skúmania. Opísali sme si tu teda to, aké jazyky či editory sa môžu pri ontológii využívať. Priblížili sme si ich základnú charakteristiku, pozitívne a negatívne vlastnosti. Táto fáza bola pre nás kľúčová, nakoľko sme sa na základe získaných informácií v druhej kapitole rozhodli, ktorý prostriedok na tvorbu ontológie bude pre nás najvhodnejší.

V tretej a zároveň finálnej kapitole sme si vytvorili analýzu doménovej oblasti. Ako vhodné a prehľadnejšie nám prišlo vytvorenie tabuľky, v tejto tabuľke sú opísané základné vlastnosti programovacích jazykov. Zaradili sme sem aj delenie programovacích jazykov, tieto jazyky sme rozdelili do troch kategórií. Vybrali sme si editor a vytvorili sme si ontológiu. Opísali sme priebeh tvorby aj problémy, na ktoré sme narazili pri tvorbe ontológie. V diskusii sme si opísali využitie vytvoreného modelu. Popísali sme aj to ako by sa dal do budúcnosti model rozvíjať.

Kľúčové slová: ontológia, ontologický model, Protégé, programovacie jazyky

Abstract

ČERVENKOVÁ, Róberta: *Using ontological modeling to create domain ontology of programming languages.* – University of Economics in Bratislava. Faculty of the Economic informatics, Department of Applied Computer Science. – RNDr. Eva Rakovská, PhD.– Bratislava: FHI EU, 2021, 53 p.

The work deals with the analysis of the properties and usability of programming languages and their classification on the basis of significant features.

Based on the analysis, current ones are selected from the set of programming languages and a domain ontology is created through ontological modeling, which can serve as a basis for finding a suitable programming language for various purposes.

The first chapter deals with ontology as such. We have approached the reader concept of ontology, which is still unknown to many people, yet it is very widespread today. We also focus in this chapter on the division or attributes of the ontology. We have explained the basic terms used in ontology. Like any concept, the ontology itself carries with it certain problems, which we also identified in the first chapter. We must not forget the specific use of ontology in practice.

We used the second chapter to state our goals and objectives. We also focused on work methodologies and research methods. We have described here what languages or editors may be needed in ontology. We have approached their certain characteristics, effective and efficient properties. This phase was crucial for us, as we decided, based on the information obtained in the second chapter, which tool for creating an ontology would be most suitable for us.

In the third and at the same time final chapter, we performed an analysis of the domain area. We found the creation of a table suitable and clearer, this table describes the basic features of programming languages. We have also included the division of programming languages, we have divided these languages into three categories. We chose an editor and created an ontology. We described the course of creation and the problems we encountered in creating the ontology. In the discussion, we described the use of our model. We also described how the model could be developed in the future.

Keywords: ontology, ontological model, Protégé, programing languages

OBSAH

Zoznam tabuliek, obrázkov a príloh

ÚVOD

Pojem ontológia ako taká sa vyskytuje medzi nami už dlho. Zo začiatku však bola spájaná len s pojmom vo filozofii. Keď sa v 90. rokoch začal tento pojem objavovať aj v informatike, vypadalo to, že nebude mať veľký úspech.

Doba išla a web ako taký obsahoval stále viac a viac informácií, toto bola príležitosť pre ontológiu. Na základe množstva informácií sa v informatike začalo skloňovať slovo ontológia čoraz častejšie. Objavovali sa rôzne prístupy k riešeniu problémov pomocou ontológie. Ontológia sa teda rozvíjala a nebola viac spájaná len s pojmom vo filozofii.

Vznikal web tretej generácie. Jeho zakladateľom bol Tim Barners-Lee, ktorý je považovaný za zakladateľa súčasného internetu. Ontológia ako taká poskytovala základ pre jeho tvorbu, vďaka spôsobu ako informácie spracovávala a aký výstup poskytovala.

Dnešný spôsob vyhľadávania informácií na webe je založený na textovom vyhľadávaní, napríklad podľa kľúčových slov. Tento spôsob vyhľadávania je nedostatočný pre množstvo informácii, ktoré v dnešnej doba web obsahuje. Môžeme to chápať aj tak, že dnešný web je zrozumiteľný pre človeka nie však pre stroj. Moderný web má spočívať v tom, že nebude zrozumiteľný len pre človeka. Mal by obsahovať aplikácie, ktoré vedia spracovať obsah informácií získaných z webu.

Zjednodušene povedané, predstavme si, že kupujeme nejaký produkt. V dnešnej dobe máme veľké možnosti a je nám poskytované veľké množstvo informácií o danom produkte. Každá z informácií sa však nachádza na inom mieste. Tento produkt sa dá deliť do rôznych kategórií a jeho kúpa záleží od preferencií používateľa. Teda musíme poznať vlastnosti produktu, jeho účel či základné znaky. Aby sme sa teda rozhodli pre kúpu správneho produktu, musíme si urobiť prieskum. Je teda nutné dohľadať si informácie o každom produkte z kategórie produktov, o ktoré máme záujem.

Princíp spočíva v tom, že všetky tieto informácie o danom produkte by sme našli na jednom mieste. Tým pádom by sme ušetrili množstvo času a mohli by sme si produkt vybrať rýchlo a jednoducho podľa kritérií, ktoré máme. Na takomto princípe by mal fungovať sémantický web.

Práca poukazuje na význam ontológie aj vo výučbe. Študenti sa stretávajú s informáciami, ktoré vyhľadávajú, spracovávajú a pracujú s nimi. Musia vidieť aj vzťahy medzi jednotlivými subjektmi, o ktorých sa učia. Často ide o doménové ontológie ako napríklad v botanike, zoológii, ale aj geografické väzby. V tejto práci môžeme vidieť aj spôsob, ako si pomocou ontologického modelu vie študent uľahčiť vyhľadávanie informácií o programovacích jazykoch.

Cieľom diplomovej práce je vytvoriť doménový model programovacích jazykov. Je tu možné vidieť, ako sú jazyky rozdelené, vlastnosti samotných skupín, do ktorých sú programovacie jazyky v našom prípade delené. Vytvorený model môže teda študentovi uľahčiť prehľad jazykov, môže lepšie pochopiť vlastnosti programovacích jazykov na základe toho, ako sú dané jazyky rozdelené do skupín. Základom je aby študent jazykom rýchlo a jednoducho porozumel. Nemusí si teda dohľadávať informácie o každom programovacom jazyku samostatne, čo by bolo pri dnešnom množstve informácií na webe časovo náročné a mohli by vzniknúť rôzne nezrovnalosti. Vyriešil by sa teda problém toho, že študenti často programovacím jazykom nerozumejú, či sa jedná už o ich vlastnosti, alebo o zaradenie, poprípade tomu kde by bolo využitie konkrétneho programovacieho jazyku vhodné.

Diplomová práca je zložená z troch kapitol. Každá kapitola má niekoľko podkapitol.

Prvá kapitola nás zoznamuje s problematikou ontológie. Opisujeme v nej jej formy či členenie. Zamerali sme sa v nej aj na problémy, ktoré sú s ontológiou spojené.

Druhá kapitola nám približuje ciele a podciele práce. Určili sme si, čo musíme splniť, aby boli naše hlavné ciele splnené. Opísali sme si v nej metódy práce, ktoré sme využili. Pre splnenie našich cieľov opísaných na začiatku druhej kapitoly sme si museli definovať metodiky ontológie. Oboznámili sme sa teda s jazykmi a spôsobmi, ako môžeme ontológiu spracovať.

Tretia a zároveň finálna časť práce, sa zaoberá samotným modelom ontológie. Pred vytváraním modulu sme si však museli vytvoriť analýzu doménovej oblasti, v ktorej sme si opísali vybrané programovacie jazyky. Zároveň sme si vybrali pre nás najvhodnejší editor, v ktorom sme ontológiu vytvárali.

1 SÚČASNÝ STAV RIEŠENEJ PROBLEMATIKY DOMA A V ZAHRANIČÍ

1.1 Ontológia

V informačných technológiách je pojem ontológia mladý, na rozdiel od filozofie. Začal sa objavovať až začiatkom 90. rokov a postupne sa rozširoval. Aj napriek tomu, že je tento pojem mladý, vzniklo naň mnoho pohľadov.[4]

Pojem ontológia je známa skôr z filozofie, kde sa objavila skôr ako určitá náuka o bytí, kde popisuje znalosti tak ako sú známe. V informatike sa pojem ontológia berie za pomerne mladý. Zjednodušene sa dá povedať, že opisuje presne definované fakty o konkrétnej doméne, čo môže byť využité v informačnom alebo znalostnom systéme.[1]

Jej cieľom je vlastne formalizovať znalosti domény všeobecným spôsobom a poskytovať spoločne dohodnuté chápanie domény, prostredníctvom ktorého sa môžu zdieľať informácie o konkrétnych objektoch.[2]

Russel, Norving hovorí že „Ontológia je formálnym popisom konceptov a vzťahov, ktoré existujú v komunite agentov.“ (Russel, Norving – 1995). Z tohto tvrdenia môžeme povedať, že ontológia umožňuje dohodnúť sa určitej skupine ľudí, na význame pojmov využívajúcich sa v konkrétnej oblasti s tým, že rôzne môže obsahovať synonymá čiže rovnaké názvy pre rozdielny objekt alebo nejednoznačnosť, kedy jeden objekt možno označiť rôznymi pojmami. Niektorí považujú svoje ontológie hlavne za prostriedok na štruktúrovanie znalostnej bázy, iní koncipujú ontológiu, ktorá sa má použiť ako súčasť vedomostnej základne.[2]

Berners-Lee tvrdí, že „Vedci v oblasti umelej inteligencie a webu si zvolili termín pre svoj vlastný žargón a pre nich je ontológia dokument alebo súbor, ktorý formálne definuje vzťahy medzi pojmami. Najtypickejší druh ontológie pre web má taxonómiu a súbor odvodených pravidiel. “ (Berners-Lee, 2001)

Za pomerne krátky čas, ktorý je ontológia súčasťou informačného sveta vzniklo mnoho definícií, ktoré sú do istej miery podobné. Medzi prvými definoval ontológiu T.Gruber, povedal o nej že „Ontológia je explicitná špecifikácia konceptualizácie“.(T.Gruber 1993)

Hovorí teda len o konceptualizácii, ktorá je explicitná, čiže je v hlave autora a vnímaná ako systém pojmov. W.Borst neskôr povedal že ontológia je formálna špecifikácia zdieľanej konceptualizácie, čím vytvoril myšlienku o zdieľanom koncepte, teda informácie nezostávajú len v hlave autora ale zdieľajú sa medzi viac ľudí, ktorí sa zaoberajú danou témou pomocou konkrétne definovanej syntaxe. (W.Borst 1997)

J.Heflin, hovorí že „ontológia definuje pojmy používané na opis a reprezentáciu oblasti poznania. Ontológie zahŕňajú počítačom použiteľné definície základných pojmov v doméne a vzťahov medzi nimi. Ontológie sú zvyčajne vyjadrené v logickom jazyku, takže medzi triedami, vlastnosťami a vzťahmi možno robiť podrobné, presné, konzistentné, spoľahlivé a zmysluplné rozdiely. “(Heflin, 2004)

Tak ako je pre pojem ontológie veľa definícií, je aj mnoho rôznych ponímaní ontológií samotných. Čím viac ontológií bude, tým viac sa budú rozchádzať.

Preto je cieľom vytvoriť konzistentnú ontológiu z rôznych ontológií, ktoré sa už využívajú a pozostávajú z rôznych zdrojov. Tu sa stretávame s pojmom ontologické mapovanie, ktoré predstavuje ekvivalentné vzťahy medzi podobnými konceptmi alebo vzťahy z rôznych ontológií. Zameriava sa na nájdenie sémantických zobrazení dvoma konkrétnymi ontológiami.(Doan a kol., 2003)

Hammed rozoberá problematiku architektúr pre spájanie viacerých ontológií. Jeden z prístupov je zdola nahor, je založený na mapovaní medzi pármi ontológií. Výhodou tohto prístupu je jeho jednoduchosť a flexibilita. Nevýhoda je však vtom, že počet mapovaní musí byť veľký vzhľadom na rozsah ontológií.(Hammed -2004)

V ďalšom z prístupov sa mapujú ontológie smerom k spoločnej ontológii. Tu sa znižuje počet mapovaní, no nevýhodou sú náklady na tento typ architektúry. Tento spôsob spočíva v budovaní klastrov bežných ontológií a v definovaní mapovania medzi týmito klastrami. V tomto prípade sú definované aj jednotlivé mapované ontológie jednou spoločnou ontológiou.

1.2 Členenie ontológie

Ontológia sa môže členiť podľa historického vývoja alebo podľa formalizácie.

1.2.1 Členenie podľa historického vývoja

Nakoľko sa ontológia využíva vo viacerých smeroch môžeme ju rozdeliť do troch základných skupín.

- Lexikálna ontológia podľa V.Svátka:„Ich charakteristickým rysom je ústredná rola termínov, ktoré už nie sú ďalej definované.“(V.Svátka, 2006) Z uvedeného vyplýva, že je založená na textových zdrojoch. Je nutné, aby sa konkrétna skupina ľudí zaoberajúca sa rovnakou problematikou, dohodla na význame jednotlivých pojmov.
- Informačná ontológia je zložitejšia ako lexikálna, nakoľko sa v nej využívajú databázy. Dala by sa považovať za rozvinutie databázových konceptuálnych schém. Dôležité je tu správne spracovanie vstupných údajov.
- Znalostná ontológia je zameraná skôr na teóriu a snaží sa reprezentovať v umelej inteligencii. Na rozdiel od informačnej ontológie nie je až tak fixne viazaná na reálne objekty. [1]

1.2.2 Členenie podľa formalizácie

Toto členenie sa zakladá na jazyku v akom je ontológia písaná. Jedna z možností je písať ju prirodzeným jazykom, čiže ontológia môže byť aj v semi-formálnej alebo dokonca aj neformálnej forme. Ďalšia z možností je písať ju vo formálnom jazyku, ktorý má dané konkrétne vlastnosti a pravidlá.

Pri formalizácii sa tiež vieme zamerať na predmet formalizácie, kde sa ontológia delí na doménovú ontológiu a ontológiu vyššej úrovne.

Doménovú ontológiu budeme využívať aj v práci. Je najpoužívanejšia a najrozšírenejšia. Zameriava sa na konkrétnu rozsiahlu oblasť napríklad zdravotníctvo, doprava a v našom prípade problematiku programovacích jazykov.[1]

V druhom prípade ide o ontológiu vyššej úrovne, ktorá zachytáva obecné zákonitosti, ktoré platia. E. Rakovská napísala „Ontológia na vyššej úrovni je „sémantické spojenie medzi“ (integračnými) doménovými ontológiami, ale keď uvažujeme o modelovaní podnikových procesov, musíme zvážiť doménovú

ontológiu ako základ.“(E. Rakovská, 2016), z čoho vyplýva že tieto dva spôsoby sú prepojené a na vytvorenie jednej musíme využiť druhú.

1.3 Atribúty ontológie

1.3.1 Triedy, sloty

Základným pojmom ontológie sú triedy. Pri jej tvorbe je vhodné, si vopred vypísať pojmy, ktoré budeme potrebovať pre vytvorenie fungujúcej ontológie, teda treba ich vhodne definovať. Po vytvorení pojmov, ktoré budeme využívať, vyberieme tie, ktoré majú nezávislú existenciu. Vybrané pojmy určíme ako triedy, v ktorých platí, že: Ak trieda A je nadtriedou triedy B, potom každá inštancia triedy B je takisto inštanciou triedy A. Čo znamená, že v ontológii existujú hierarchie a nemôžu v nej byť cykly. Trieda A teda nemôže byť nadradenou triedy B a zároveň trieda B nemôže byť nadradená triede A. To, že je trieda podradená druhej, znamená, že podriadená trieda reprezentuje koncept triede nadradenej.[6]

Ostatné pojmy, ktoré sme si vybrali na začiatku a neurčili sme ich ako triedy budú tvoriť vnútornú štruktúru konceptov. Táto štruktúra je vlastne opis vlastností, ktoré jednotlivé triedy obsahujú. Napríklad ak máme programovacie jazyky, každý z týchto jazykov má určité základné vlastnosti. Triedy, ktoré obsahujú určité vlastnosti, označujeme ako definované. Naopak tie, ktoré vlastnosti nemajú, označujeme ako primitívne.[6]

Trieda môže mať tiež svoje podtriedy, ktoré reprezentujú koncepty, ktoré sú špecifickejšie než je nadtrieda. Napríklad triedu SQL môžeme rozdeliť na triedy MySQL a Oracle.

Pri jednoduchých jazykoch, teda pri binárnej relácii, na ktoré sú obmedzené tieto jazyky, sa používa pojem slot.

Ontológia obsahuje aj sloty tieto môžeme rozdeliť na:

- vnútorné (vnútorná vlastnosť triedy, ktorú nevidíme hneď)
- vonkajšie (vlastnosť triedy, ktorá je viditeľná, napríklad názov)
- časti z ktorých sa objekt skladá
- vzťahy [6]

Pri tradičných jazykoch v ontológii rozlišujeme aj vzťahy, teda relácie n-tíc objektov. Relácie sú podstatnou zložkou objektov. Vytvárajú a spájajú sa na základe logických podmienok. Rovnako ako pri triedach, aj medzi slotmi a aj reláciami je určitá hierarchia.[1]

V ontológii je možné priradiť slotom vlastnosti, ktoré môžeme nazvať ako meta-sloty.

Rozlišujeme tranzitívnu binárnu reláciu, pri ktorej sa vlastne jedná o nadriadenosť a podriadenosť jednotlivých slotov. Ako druhú poznáme symetrickú binárnu reláciu, o ktorej V.Svátek tvrdí, že „Symetrickou binárnou (meta-)reláciou je vzťah slotov voči slotu k nemu inverznému; ostatné meta-sloty odpovedajú známym reláciám (vlastnostiam slotu samotného) ako je symetria, tranzitivita, funkčnosť alebo inverzná funkčnosť.“ (V.Svátek, 2006) Vzťahy sa tu teda vytvárajú na zákale inverzie relácií.

Podobne ako v iných jazykoch, aj tu poznáme pojem dedenie. Dedia sa vlastnosti, pri ktorých platí, že všetky podtriedy danej triedy dedia sloty tejto nadradenej triedy.

Počet hodnôt, koľko môže slot obsahovať, mu určuje kardinalita. Každý systém si určuje svoj minimálny a maximálny počet hodnôt.

1.4 Problémy ontológie

Pri tvorbe ontológie sa vyskytujú dva závažné problémy, ktoré je veľmi dôležité riešiť pre funkčnosť a korektnosť vytvoreného modelu.

V prvom prípade ide o problém rozsahu. Tento problém spočíva v tom, že existuje veľa pojmov, ktorými by sa dala vybraná doména vyjadriť. Tu môže nastať chyba aj pri zlom výbere, kde následne vznikajú komplikácie pri tvorbe. V tomto prípade je nutné robiť selekciu, pri ktorej sa zvolí správny výraz relevantný k doméne. Medzi ďalšie opatrenie voči problému rozsahu je, že pojmy by mali mať jasnú väzbu na pojmy zontológie. Pri tomto probléme môže pomôcť aj rozloženie vytváranej ontológie do viacerých nezávislých modelov.

Druhým problémom pri vytváraní ontológie je problém interakcie. Optimálny tvar ontológie je závislý na tom, ako sa bude daná ontológia používať a aká bude jej implementácia. Existuje tu teda interakcia medzi doménovými znalosťami a postupmi, ktoré sme si zvolili. Berie sa do úvahy používateľnosť a znovu použiteľnosť ontológie.

Platí, že čím viac je ontológia nezávislá na konkrétnej aplikácii, tým menej efektívne sa stáva jej využívanie. Samozrejme existujú aj ďalšie problémy, ale tieto dva sú najrozšírenejšie.[1]

1.5 Využitie

1.5.1 Metadáta

Metadáta môžeme definovať ako údaje o údajoch. Dôležité je tu vysvetliť si aj pojem anotácie. Anotácie webových stránok, ktoré sa uložia do databáz alebo na server sa stávajú metadátami. Tu sa vyskytuje pojem XML, ktorý je prvou úrovňou sémantiky, ktorá umožňuje používateľom štruktúrovať dáta vzhľadom na ich obsah. HTML tu vyznačuje spôsob akým by sa mali dáta zobrazovať.[3]

1.5.2 Sémantický web

Pri ontológii je nutné spomenúť aj pojem sémantický web. Postupom času sú zbierané údaje čoraz zložitejšie a ich množstvo narastá. K ich zložitosti prispievajú rôzne faktory, ako napríklad chýbajúca štruktúra, veľký objem, dynamický vývoj alebo viac-rozmernosť dát.

Pri klasickom vyhľadávaní sa údaje hľadajú podľa kľúčových slov, neberie sa však ohľad na relevantnosť týchto nájdených údajov. Tu preto nastáva problém napríklad pri homonymách. Zbytočne sa nám môže zobraziť množstvo údajov, ktoré vôbec nie sú potrebné. Tomuto problému sa snažilo zabrániť v sémantickom webe, v ktorom sa vyhľadáva na základe významu hľadaných údajov.

Pri vyhľadávaní pomocou sémantického webu sa definuje hľadaný výraz. Tento hľadaný výraz sa následne hľadá tak, že sa analyzujú rôzne webové stránky. Ontológie sú zvyčajne zložené zo zoznamu vzájomne súvisiacich výrazov a môžu sa medzi používateľmi a aplikáciami vymieňať. Na základe tohto je možné pochopiť vzťahy medzi jednotlivými výrazmi. Môžu byť definované viac alebo menej formálnym spôsobom, čo zahŕňa aj prirodzený jazyk. V ontológii teda platí, že sa nevyhľadáva len zadaný výraz, ale aj pojmy súvisiace s týmto výrazom. Bežný jazyk však nie je jediným spôsobom vyjadrenia ontológií v sémantickom webe. Často sa používa aj jazyk Web Ontology Language (OWL), ktorý patrí do druhej kategórie, teda do formálnych jazykov. OWL je postavený na RDF a RDFS a rozširuje ich o vyjadrenie vlastností triedy.

Napriek novému spôsobu vyhľadávania sémantický web nie je webom novým, je to len rozšírenie už existujúceho webu tak, aby bol zrozumiteľnejší pre stroje. Hlavný cieľ v sémantickom webe je formálne vyjadrenie vybraných sémantických informácií o údajoch tak, aby tieto vybrané informácie boli spracované a použité počítačom. Sémantické informácie môžu vystupovať ako sémantické anotácie alebo metadáta. Metedáta a ontológie sa navzájom dopĺňajú, pričom tvoria stavebné prvky sémantického webu. Ich cieľom je predísť významovým nejasnostiam a poskytovať presnejšie odpovede, a tiež poskytujú presnejšie výsledky. Jedným z ďalších cieľov sémantického webu je opísať sémantické vzťahy medzi jednotlivými odpoveďami. Počíta sa s tým, že sémantický web bude stále viac a viac využívaný, čo znamená, že techniky súvisiace s ním, sa budú neustále vyvíjať ďalej. Tento fakt sa vzťahuje aj na problematiku ontológie. Bude nutné, aby sa vyvíjali a zdokonaľovali jej formy.

Pre splnenie cieľov, ktoré sú pre sémantický web stanovené, sa navrhlo mnoho formátov, medzi ktoré patrí Resource Description Framework od W3C a Topic Maps od Medzinárodnej normalizačnej organizácie, ktoré vznikli okolo roku 1999. Využívala sa vo veľkom RDF, ktorému sa budeme venovať nižšie. Pomocou RDF sa dá vytvoriť aj schéma RDFS, ktorá slúži ako obohatenie a vyjadruje hierarchie tried a obmedzenia pri písaní, že daný typ relácie môže spájať iba konkrétne triedy.[3]

1.5.3 Ďalšie možnosti využitia

Ďalším zaujímavým využitím ontológie z reálneho prostredia, ktoré ju môže lepšie ozrejmiť je odvetvie žurnalistiky. Vytvárajú sa tu takzvané grafy znalostí, ktoré podporujú novinárske spravodajské uhly.

Thurman povedal že považuje výpočtovú žurnalistiku za pokročilú aplikáciu výpočtovej techniky, algoritmov a automatizácie na zhromaždenie, hodnotenie, prezentáciu, a distribúciu správ . (Thurman 2019) Výpočtová žurnalistika môže slúžiť ako podpora novinárov, alebo automatizácia žurnalistiky.

Systém ontológie v tomto odvetví bol navrhnutý tak, aby zbieral potenciálne spravodajské informácie z rôznych zdrojov, a všetky informácie vhodné na novinárske účely.

Táto ontológia funguje na princípe neustáleho sťahovania nových informácií. Informácie sa následne vložia do databázy, ktorá z nich vytvorí vyššie spomínané znalostné grafy. Z grafu sa následne rozdelia informácie podľa témy, a vytvorí sa finálny graf znalostí.

Ak je nahrané dostatočné množstvo informácií o danej téme, pošle sa jeden veľký finálny graf novinárovi, ktorý s ním pracuje.[16]

2 CIELE PRÁCE , METODIKA PRÁCE A METÓDY SKÚMANIA

Cieľom diplomovej práce je vytvorenie ontologického modelu, ktorý nám poskytuje vzťahy a atribúty programovacích jazykov. Na to, aby sme sa dostali k nášmu stanovenému cieľu, musíme splniť niekoľko podcieľov.

V prvej kapitole práce sme si zhromaždili všetky potrebné informácie o samotnej Ontológii. V tejto časti sme sa zoznámili s problematikou. Vytvorili sme si teda teoretickú časť, ktorú môžeme zaradiť medzi naše podciele. Začali sme opisom ontológie ako celku, pokračovali sme so samotným členením pre lepšiu prehľadnosť. Rovnako bolo nutné, aby sme si definovali základné prvky, ktoré ontológia využíva. V neposlednom rade sme sa oboznámili s problémami, ktoré môže ontológia priniesť, či už pri jej tvorbe alebo samotnom využívaní. Túto kapitolu sme ukončili konkrétnymi príkladmi, kde sa dá ontológia využiť. Posledná časť má pomôcť pri lepšom pochopení využitia ontológii v praxi.

Druhá kapitola obsahuje samotné ciele a podciele práce. Ďalej sme si v nej opísali vedecké metódy, ktoré sme pri práci využili, ako sú analýza, dedukcia či rešerš. Najpodstatnejšou časťou v druhej kapitole sú samotné metodiky ontológie a spôsob jej modelovania. Zaoberali sme sa tu konkrétnymi jazykmi, ktoré ontológia využíva. Opísali sme si aj editory, čo nám slúžilo ako podcieľ, na základe ktorého sme si vybrali v tretej kapitole Protégé ako editor, v ktorom budeme pracovať.

Tretia kapitola práce spočíva vo výbere vhodného editoru na základe splnenia predchádzajúceho podcieľu, ktorým bolo urobiť a spracovať si rešerš o jednotlivých metódach tvorby ontológii. V tomto podcieli sme si predstavili jednotlivé jazyky a editory, ktoré ontológia využíva a teda, ktorými je tvorená. Taktiež bolo nutné si stanoviť ako ďalší podcieľ analyzovanie doménovej oblasti. Pre zjednodušenie a prehľadnosť sme si vytvorili tabuľku, ktorá obsahuje programovacie jazyky, základné informácie a atribúty na základe ktorých sme robili delenie na triedy. Vytvorená ontológia sa spracovávala podľa tejto vytvorenej tabuľky. Tabuľka slúži v práci ako základ na tvorbu doménovej ontológie.

Hlavným cieľom v praktickej časti bolo vytvorenie ontologického modelu, na základe analýzy doménovej oblasti. V práci sme využili voľne dostupný editor Protégé, v ktorom sme pracovali a vytvárali sme v ňom samotný model.

2.1 Metódy práce

V práci sme využili tri základné metódy:

- rešerš: tento typ metódy sme využili najmä v prvej časti práce. Získavali sme potrebné informácie pre splnenie našich cieľov z dostupnej odbornej literatúry, článkov a textov.
- analýzu: v tretej kapitole bolo potrebné analyzovať jednotlivé programovacie jazyky, a práve pre túto potrebu sme si vybrali metódu analýzy. Analyzovali sme či už základné vlastnosti jazykov alebo samotné delenie.
- syntézu: v praktickej časti, pri tvorbe ontológie bolo nutné, aby sme využili a spojili všetky nazbierané informácie. Na základe týchto informácií sme vytvorili model ontológie pomocou metódy syntézy.

2.2 Metodiky ontológie a jej modelovanie

Väčšina ontológií, ktoré zo začiatku vznikali, boli vytvorené ručne. Prvé metodiky boli vytvorené na základe podnikových ontológií. Patrí sme napríklad metodika Grüningera a Foxa, ktorá bola vytvorená vzhľadom na znalostné systémy a s využitím logiky prvého rádu. Pri tejto metodike sa ako prvé vypracuje súbor neformálnych otázok, na ktoré by mala ontológia vedieť zvládnuť odpovedať. Tieto otázky a odpovede sa používajú na extrakciu hlavných pojmov, ich vzťahov a vlastností formalizovaných pomocou logiky prvého rádu. Na koniec je nutné určiť podmienky, pri ktorých je riešenie otázok kompletné. [2]

Táto metodika sa využíva ako základ pri budovaní a validácii ontológii, nie je však dokonalá a chýbajú v nej niektoré podrobné činnosti, napríklad riadiace funkcie alebo integrácia. Tu prichádza metodológia Gomez-Péreza, ktorá vytvára ontológiu od začiatku, a umožňuje využívať ontológiu aj v sfére inžinierstva. Ontologické inžinierstvo je založené na získavaní konceptuálneho modelu a jeho zmeny na prijateľnejšie. Táto metodika sa zameriava na stavbu ontológií na vedomostnej úrovni a spočíva v identifikácii procesu vývoja ontológie, ktorá obsahuje niektoré hlavné činnosti a to sú: hodnotenie, konfigurácia, riadenie, koncepcia, integrácia a implementácia. Životný cyklus je založený na

vyvíjajúcich sa prototypoch. Metodika špecifikuje kroky na vykonávanie každej činnosti, použité techniky a spôsob vyhodnocovania.[2]

Ďalší z prístupov je napríklad prístup On-To-Knowledge, pri ktorom sa využíva nástroj OntoEdit. Zameriava sa na zavedenie a následné udržiavanie ontologických systémov riadenia znalostí. Má konkrétnu techniku riešenia pre každý svoj proces. Definuje tiež vzťahy medzi procesmi. Samotná aplikácia je veľmi dôležitá pre jej správnu funkčnosť.[2]

Pri konštrukcii ontológie si môžeme vybrať z rôznych nástrojov. V zásade sa delia na dve skupiny, a to sú nástroje, pri ktorých je znalostný model formalizovaný v ontologickom jazyku, alebo nástroje, pri ktorých znalostný model nie je závislý od programovacieho jazyka. Oba tieto nástroje si priblížime nižšie.

2.2.1 Jazyky ontológie

V tejto podkapitole sa budeme venovať jazykom, ktorými môže byť ontológia opísaná.

Za krátku dobu, počas ktorej sa vyskytuje pojem ontológia v informatike, sa vytvorilo pomerne veľa jazykov. Nakoľko sa ontológia vyvíjala postupne, vznikali aj nové jazyky postupne ako reakcia na nedostatky jazyka druhého, už existujúceho. Prvý jazyk, ktorý sa v ontológii objavil, bol CYC v roku 1984. Cieľom vzniku tohto jazyka bolo zhromaždiť všetky dostupné znalosti k znalostiam expertným.

Následne sa v 90. rokoch začala vyvíjať, a zároveň aj vznikla Ontolinqua. Cieľom tohto ďalšieho jazyka v ontológii bolo vytvorenie zrozumiteľného jazyka, pomocou ktorého by mohli ontológie zdieľať informácie medzi rôznymi znalostnými systémami. Hlavnou podstatou vzniku teda nebolo, ako v prvom prípade, zhromažďovať informácie. Tento jazyk sa zameriaval na výmenu informácií medzi sebou. Nakoľko boli jazyky nekompatibilné, vznik jazyka s touto funkciou bol nevyhnutný za účelom odľahčenia od prekladania do iného jazyka. [1]

2.2.1.1 CYC

Ako sme spomínali vyššie, jedným z prvých jazykov bol práve spomínaný CYC. Bol to pokus o jedno z prvých zachytení veľkého množstva znalostí vo svete. Názov CYC je odvodený z anglického slova enCYClopedia. Tento jazyk sa začal vyvíjať vďaka

iniciácii D. Lenata v roku 1984. Snažil sa o zhromažďovanie všeobecných znalostí, ktoré by v znalostných systémoch fungovali komplementárne ku znalostiam expertným a zabraňovali absurdnému chovaniu. Napriek tomu, že tvorcovia svoju aplikáciu nespájali príliš s ontológiou, je jasné že o ontológiu sa jedná.[1]

Cyc využíva svoj vlastný jazyk CycL, ktorý má vyjadrujúcu silu predikátovej kalkulácie, kombinuje ho však s prvkami rámcových jazykov.

Ako príklad sme si uviedli kód zapísaný v CycL,

```
#$ist #$NaiveStateChangeMt
(#$implies
(#$and
(#$isa ?FREEZE #$Freezing)
(#$outputsCreated ?FREEZE ?OBJ))
(#$stateOfMatter ?OBJ #$SolidStateOfMatter)))
```

Hovorí, že #$NaiveStateChangeMt je pravda, teda true, že objekt ?OBJ (označenie predmetu) je výsledkom udalosti ?FREEZE (označujúca premenná) typu zamrazenie je v pevnom skupenstve.[1]

2.2.1.2 Ontolingua

Cyc bol založený na uzavretom koncepte a to bola iniciácia pre T.Gruberema na začiatku 90. rokov, aby spolu s kolegamizo Knowledge System Laboratory vytvoril jazyk Ontolingua.

Cieľom tohto jazyka bolo, aby umožnil zdieľať ontológie rôznych odborných komunít, ktoré používajú navzájom nekompatibilné znalostné systémy. Chceli zároveň, aby bol tento jazyk prehľadný a jednoduchý na použitie. Je štylizovaný ako nadstavba jazyka KIF (Knowledge Interchange Format), ktorý využíva syntax LISP.

Medzi základný konštruktor jazyka Ontolingua sú definície tried, relácií a funkcií. Vymedzujúce podmienky pre inštancie sú vyjadrené v KIF. Keďže cieľom jazyka

Ontolingua, ako sme už spomínali vyššie, bolo, aby jednotlivé ontológie mohli medzi sebou komunikovať, respektíve sa spájať. Aby bola teda zaistená ich jednotná sémantika, vzniklo Frame-Ontology. Nachádza sa v jazyku Ontolingua a obsahuje príslušné definície, je teda možné na tieto vlastnosti odkazovať.

Ontolingua bola teda od začiatku koncipovaná ako medzi jazyk, ktorý je primárne určený k výmene informácií v ontológii medzi sebou v rámci systému. Na základe tohto môžeme vidieť mnoho obmedzení pri odvodzovaní priamo v tomto jazyku. Nakoľko bol však jazyk Ontolingua veľmi rozšírený, mnoho rokov sa využíval aj ako základný jazyk pre tvorbu ontológií nezávisle na konkrétnom znalostnom systéme. Nebol teda využívaný len na spájanie ontológií a komunikáciu medzi nimi.[1]

2.2.1.3 OCML

Ako sme si spomenuli vyššie pri definícii jazyka Ontolingua, tento jazyk má veľa obmedzení pri odvodzovaní. Preto sa rozhodol E. Mottu z Open University vo Veľkej Británii navrhnúť ďalší jazyk.

Týmto jazykom je OCML (Operational Conceptual Modelling Language). OCML mal vlastne slúžiť ako jazyk, ktorý umožňuje na základe zachovania ontologickej podstaty podporiť vývoj programovacích aplikácií vo väčšom, bez nutnosti prekladať model do iného jazyka. Jednalo sa teda tiež o snahu spojenie ontológie do jednej, bez nutnosti využívania viacerých jazykov.[1]

2.2.1.4 OKBC

V roku 1993 začal vznikať návrh aplikačného rozhrania. Cieľom bolo, aby umožnil otvorenú komunikáciu medzi znalostnými systémami OKBC (Open Knowledge Base Connectivity). Jedná sa o udávanie spôsobu, akým sa budú predávať konštruktory znalostných báz, ktorými sú triedy alebo sloty, alebo volanie jednotlivých operácií nad týmito konštruktormi. Vo veľkej miere OKBC sa zhoduje z konštruktormi, ktoré obsahujú jazyk Ontolingua.[1]

2.2.1.5 XOL

OKBC bol inšpiráciou pre vznik ďalšieho ontologického jazyka, a to jazyka XOL (eXtensible Ontology Language). Tento jazyk mal podstatu v syntaxe jazyka XML, ktorá mala mnoho nástrojov, ktoré sa dali využiť pre XOL. Jazyk sa venoval časti využitia ontológie, ktorej sa v predchádzajúcich jazykoch nedávala taká pozornosť. Slúžiť mal priamo na konkrétny účel a to bolo zdieľanie štruktúry znalostí o génovom výskume.

Rozdiel medzi XML a XOL je v tom, že v jazyku XOL sa nachádza len jedna generická definícia typu dokumentu, ktorá pomáha definovať štruktúru tried a slotov. [1]

Ako príklad si môžeme uviesť kód, ktorý nám hovorí o tom, že trieda strom má vlastnosť výška do 200 :

```
<class>
<name>strom</name>
</class>
<slot>
<name>vyska</name>
<domain>strom</domain>
<value-type>integer</value-type>
<numeric-max>200</numeric-max>
</slot>
```

2.2.1.6 SHOE

Môžeme povedať že jazyky ontológie rozdeľujeme na staršie a novšie. Práve jazyk SHOE by sme zaradili medzi prvý historický jazyk. Vytvoril ho J. Hendler na University of Maryland v roku 1996.

Je založený na koncepte jazyka HTML, jazyk SHOE umožňuje priamo do jeho štruktúry vkladať ontológiu rovnako ako aj metadáta, kde sa pomocou ontológie definuje

sémantika týchto metadát. K základným črtám SHOE patrí jeho jednoduchosť v porovnaní s inými jazykmi ontológie. Tento jazyk zachytáva iba triedy a relácie (sloty). [1]

Vznikom SHOE sa chcú jeho tvorcovia vyhnúť tomu, aby sa stala ontológia zastaranou ešte skôr, by sa mohla stať populárnou. Pre riešenie týchto problémov má jazyk SHOE dve riešenia.

Prvé z nich je, že umožňuje vytváranie nových verzií ontológie, pričom sa zachová integrita modelu. Druhým zo spôsobov je rozšírenie ontológie za účelom reflexie viac špecializovaných informácií. Čo v skratke znamená, že môže byť vytvorená hierarchia ontológií, rovnako ako sa to umožňuje aj v objektovo orientovanom prístupe.

Deklarácia sa dávala na začiatok kódu HTML stránky:

<ONTOLOGY ID="identifikátor"

VERSION="*verzia*">

</ONTOLOGY>

Do značiek sa vkladali jednotlivé triedy a ich definície:

<DEF-CATEGORY NAME="*názov triedy*">

<DEF-RELATION NAME="*názov vzťahu*">

Anotácia sa označovala pomocou URL stránky:

<INSTANCE KEY="*URL*">

Vnorené elementy vzhľadom na ontológiu:

<USE-ONTOLOGY ID="*identifikátor ontológie*"

URL="*URL ontológie*"

VERSION="*verzia ontológie*">

Tiež sa vkladajú metadáta:

<CATEGORY NAME="kategória">[9]

2.2.1.7 Ontobroker

V prípade jazyka Ontobroker sa jedná o ďalší historický jazyk. Tento jazyk vznikol skoro v rovnakej dobe ako SHOE. Koncept jazyka Ontobroker sa začal vyvíjať na univerzite v Karlsruhe.

V tomto jazyku sa vytvorila iná architektúra, ktorá mala byť presne centralizovaná. Hlavnou myšlienkou tvorcov bola možnosť pridať k webovým stránkam sémantiku. Táto myšlienka mala byť založená na existencii ontologického serveru, ktorý by spracovával všetky ontológie. K tomuto serveru by mali prístup iba kvalifikované osoby.

Pri vývoji sa tiež nepočítalo s tým, že údaje, ktoré sú pre ontológiu potrebné, by sa získavali z náhodných webových stránok. Zbieranie údajov malo byť založené skôr na cielenom navštevovaní registrovaných stránok a tieto informácie by museli patriť k nejakej odbornej komunite.

Ontobroker používa rozdielny jazyk pre ontológie a pre anotácie. Jazyk venovaný anotáciám je jednoduchým obohatením HTML o dodatočné atribúty. [10]

2.2.1.8 RDF

Jazyk RDF začal vznikať koncom 90. rokov. Cieľom jeho vzniku bolo vytvoriť jazyk na reprezentovanie informácií o webových zdrojoch. Teda pre popis, výmenu a znovu použitie metadát.

RDF na rozdiel od iných jazykov nemá presne a jasne definovanú syntax. Jeho syntax je abstraktná a funguje na predpoklade, že každý webový zdroj má svoje vlastnosti. Všetky tieto vlastnosti sa dajú popísať a na základe nich sme schopní o zdrojoch vytvárať jednoduché tvrdenia. Tieto tvrdenia označujeme ako takzvané trojice.

- Subjekt vystupuje ako vec, ktorú chceme pomocou RDF opísať. Jedinou podmienkou pri výbere subjektu je, že mu musí byť možné priradiť URI. To znamená, že ho vieme identifikovať pomocou URI referencie.
- Vlastnosť tvrdenia popisujú vzťahy medzi subjektom a objektom.
- Objekt predstavuje hodnotu danej vlastnosti tvrdenia.

Tu sa dostávame k pojmu RDG grafy. Môžeme povedať, že množina trojíc tvorí RDF graf. Tento graf sa zobrazuje pomocou prepojených uzlov orientovaného grafu.
Platí tu, že:

- Orientácia hrany je vždy smerom od subjektu k objektu. Vyjadruje vzťah medzi subjektom a objektom.
- Každá z trojíc je reprezentovaná dvojicou uzlov a orientovanou spojnicou.
- Každú trojicu RDF môžeme spojiť s inou trojicou. Pri tomto kroku sa však nemení význam ani jednej z trojíc. Tiež nezáleží na zložitosti grafu.
- Objekt tvrdenia môže vystupovať ako subjekt iného tvrdenia.[10]

2.2.1.9 DAML+OIL

Tento jazyk vznikol spojením dvoch rôznych ontologických jazykov.

Prvým z nich bol DAML. Cieľom jeho vzniku bolo vytvoriť sémantický jazyk pre formát RDF, ktorý by bol vhodnejší ako vyššie spomínaný RDFS. Jeden jazyk však nestačil. Nakoľko bolo nutné vytvoriť ontologický jazyk, ktorý by mal prepracovanejšiu logiku a umožňoval by konštruovať zložitejšie podmienky vznikol jazyk OIL (Ontology Inference Layer). Tieto dva jazyky sa spojili do jedného ako DAML+OIL.

Základom nového jazyka DAML+OIL sú triedy. Tieto triedy môžu byť definované buď svojím menom, teda URI alebo určitým logickým významom. Výhodou je, že konštruktory v tomto jazyku je možné svojvoľne skladať a to umožňuje vytvárať zložité výrazy. V tomto prípade sa ontológia napĺňa axiómami (základná logická veta v určitom odbore), vybudované nad výrazmi reprezentujúcimi triedy.[10]

2.2.1.10 OWL

Pri nových jazykoch ontológie je určite nutné spomenúť jazyk OWL. Prvá pracovná verzia jazyka sa objavila v roku 2002. Vznikol na základe predchádzajúcich skúsenosti s jazykom DAML+OIL.

OWL bol navrhnutý tak, aby vyhovoval potrebe jazyka webovej ontológie. Je súčasťou rastúceho množstva W3C týkajúcich sa sémantického webu.

XML poskytuje len povrchovú syntax pre štruktúrované dokumenty, nedefinuje im však žiadne sémantické obmedzenia ani význam. Schéma XML obmedzuje štruktúry dokumentov XML, ale zároveň ju rozširuje o dátové typy.

RDF je dátový model objektov a vzťahov medzi objektmi, poskytuje jednoduchú sémantiku pre rozšírený dátový model. Pričom schéma RDF je slovník na popis vlastností a tried zdrojov RDF so sémantikou týchto vlastností a tried.

Tu prichádza na rad jazyk OWL, ktorý pridáva rozšírenejšiu slovnú zásobu na opis vlastností a tried. Opisovať môže vzťahy medzi triedami, mohutnosť, rovnosť, bohatší opis vlastností, charakteristiky vlastností a mnoho ďalších.[10]OWL poskytuje tri čoraz príťažlivejšie podjazyky.[10]

- Prvým z nich je OWL Lite, ktorý podporuje používateľov vyhľadávajúcich predovšetkým hierarchiu klasifikácie a jednoduché obmedzenia. Tento jazyk by mal uľahčiť implementáciu programových nástrojov, ktorá bola pre plnú verziu DAML+OIL veľmi komplikovaná. OWL Lite podporuje len niektoré typy lokálnych obmedzení na sloty. Slabšie sú tu možnosti využívania anonymných tried a obmedzení na kardinalitu. [1] Napríklad, hoci podporuje obmedzenia mohutnosti, povoľuje iba hodnoty mohutnosti 0 alebo 1. Poskytovanie podpory nástrojov pre OWL Lite by malo byť jednoduchšie ako v ostatných prípadoch jazyka OWL. OWL Lite zároveň poskytuje cestu rýchleho prechodu pre taxonómie. Owl Lite má tiež nižšiu formálnu zložitosť ako OWL DL.[10]

- OWL DL slúži ako podpora pre používateľov, ktorí požadujú maximálnu expresivitu pri zachovaní výpočtovej úplnosti a zároveň žiada, aby všetky výpočty skončili v určitom čase. OWL DL obsahuje všetky jazykové konštrukcie OWL, ale je možné ich použiť iba za určitých obmedzení. Napríklad zatiaľ čo trieda môže byť podtriedou mnohých tried, trieda nemôže byť inštanciou inej triedy. Názov OWL DL vznikol na základe korešpondencii s logikou popisu, oblasťou výskumu, ktorá študovala logiku a tvorí formálny základ OWL.

- OWL Full je určený pre používateľov, ktorí požadujú maximálnu expresivitu a syntaktickú slobodu RDF bez výpočtových záruk. Ako príklad môžeme uviesť, že v OWL Full môže byť s triedou zaobchádzané ako s kolekciou jednotlivcov a súčasne ako s jednotlivcom samostatne. OWL Full umožňuje ontológii rozšíriť význam preddefinovanej slovnej zásoby.

Každý z týchto čiastkových jazykov je rozšírením svojho jednoduchšieho predchodcu.

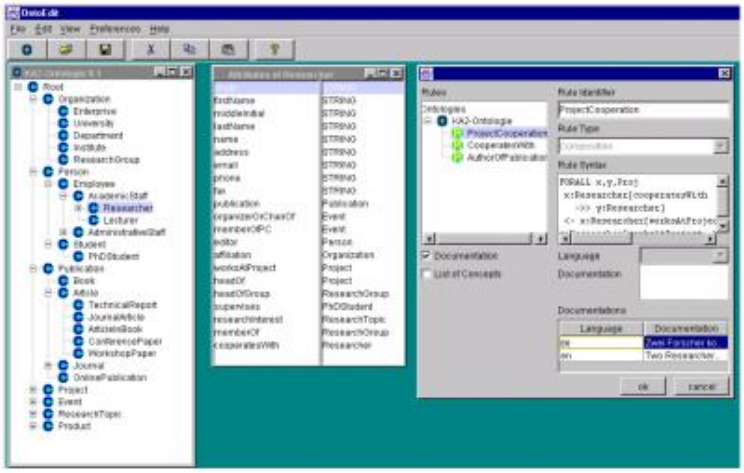
Rozdiel medi OWL Lite a OWL DL spočíva teda v tom, že OWL Lite používa iba niektoré funkcie OWL a má viac obmedzení pri používaní týchto funkcií. Ako príklad môžeme povedať, že v triedach OWL Lite je možné definovať iba pojmy pomenovaných nadtried. To znamená, že nadtriedy nemôžu mať ľubovoľné názvy. Rovnocennosť medzi

triedami a vzťahy podtried medzi triedami sú tiež povolené iba medzi pomenovanými triedami, a nie medzi ľubovoľnými výrazmi tried. Obmedzenia v OWL Lite používajú iba pomenované triedy. OWL Lite má tiež obmedzenú predstavu mohutnosti - jediné kardinality, ktoré je možné uviesť, sú 0 alebo 1.[10]

2.2.2 Editory ontológie

2.2.2.1 OntoEdit

Tento editor slúži ako vývojové prostredie a využíva grafické nástroje na správu a vývoj ontológií. Editor OntoEdit využíva jazyk XML, ktorý mu pomáha pri práci so súbormi. Uľahčuje jazyku modelovanie konceptov, axióm a vzťahov. Na obrázku môžeme vidieť ukážku prostredia.



Obrázok 1 Prostredie

Vďaka grafickému pohľadu štruktúry ontológie sa dá modelovať v akejkoľvek časti jej vývojového cyklu. Výstup je možné dostať v E-R diagrame alebo DTD. Tento editor je založený na komerčnej báze. Obsahuje aj funkcie ako sú pomenovanie objektov vo viacerých jazykoch alebo spájanie ontológií. [11]

2.2.2.2 OilEd

Ako už môže vyplývať z názvu tento editor slúži pre ontológie, ktoré sú vytvárané v jazyku DAML+OIL. Jeho tvorcovia ho definovali ako poznámkový blok pre ontológie. Nie je teda určený pre tvorbu veľkých ontológií. Je to jednoduchý nástroj, ktorý obsahuje len základné funkcie potrebné pre tvorbu modelov. Pretože obsahuje hlavne základné

funkcie a jeho používanie je teda nimi obmedzené, využíva sa najmä na výučbové účely. [1]

2.2.2.3 Protégé

Ako môžeme vidieť vyššie, je množstvo jazykov v ontológii, my sme si opísali tie základné. Napriek tomu, že môžeme na tvorbu využiť tieto jazyky v bežnom textovom editore alebo editore XML, je vytvorených množstvo programov, ktoré nám túto prácu zjednodušujú a nekladú veľké nároky na používateľa.

Medzi takéto editory parí aj Protégé. Výhodou je, že ontológiu môžeme po jej vytvorení exportovať do veľa formátov ako napríklad OWL, XML, RDF. Tento editor je založený na základe open-source, teda je voľne dostupný. Jeho tvorcovia využili pri jeho tvorbe programovací jazyk Java. Ponúka možnosť jednoduchého rozšírenia vďaka zásuvným modulom. Tiež obsahuje nástroje pre tvorbu vizualizácie, manipulácie ontológie vo viacerých formátoch, a v neposlednom rade umožňuje tvorbu doménových modelov. Protégé je preto komplexný nástroj na prácu s ontológiami, pričom práca s ním je veľmi intuitívna, čo môžeme považovať za veľkú výhodu pre používateľa.

Poznáme dva typy editoru Protégé:

- Protégé-Frames editor- tento editor uľahčuje modelovanie znalostí pre používateľov, a rovnako tak aj vkladanie dát, nakoľko obsahuje rozsiahlu množinu prvkov užívateľského rozhrania. Editor je založený na tom, že sa v ňom vytvárajú frame-based doménové ontológie, ktoré umožňujú upravovanie formulárov na zadávanie dát a ich vkladanie do inštancie dát. Model ontológie je v tomto prípade zložený z množiny tried, ktoré sú zobrazované v hierarchickej štruktúre na reprezentáciu konceptov domény, slotov združených do tried obsahujúcich ich vlastnosti. Obsahuje takzvanú plug-in architektúru, ktorú môžeme rozširovať pomocou vlastných prvkov a tie môžu byť napríklad rôzne médiá, grafické komponenty, ukladacie formáty. API tu vytvára prístup, zobrazovanie a používanie pre aplikácie a plug-ink k vytvoreným ontológiám.
- Protégé-OWL editor - chápeme ako rozšírenie pôvodného Protégé. Podporuje jazyk OWL, ktorý je ako posledný schválený konzorciom W3C ako podpora pre sémantický web. Táto ontológia obsahuje zväčša popis tried, vlastností a ich inštancie. Na základe sémantiky v OWL sa odvodzujú fakty, ktoré v ontológii nie sú uvedené. Pomocou Protégé-OWL môže používateľ definovať vlastnosti tried,

využívať reasonery, editovať triedy, upravovať OWL, ukladať ontológiu vo formáte RDF a OWL.[12]

2.2.2.4 Reasoner HermiT

Tento editor využíva jazyk OWL. Je to open-source reasoner, ktorého komponenty a všetky zdrojové kódy sú voľne dostupné ku stiahnutiu. HermiT vznikol na University of Oxford. Môže sa používať priamo v Jave ale podporuje aj OWL API.

HermiT je vytvorený tak, aby sa všetky úlohy, ktoré má plniť, mohli zadávať cez príkazový riadok. Pre túto vlastnosť je práca s ním veľmi jednoduchá. Ako príklad na jej funkcie si môžeme uviesť to, že vie identifikovať relácie medzi jednotlivými triedami, alebo môže určiť, či je daná ontológia konzistentná alebo nie. HermiT používa pri plnení úloh hypertableau kalkul. Tento algoritmus je účinnejší ako tie, ktoré sa používali skôr. Ďalšou jeho veľkou výhodou je rýchlosť. Vie klasifikovať ontológiu behom pár sekúnd na rozdiel od jeho predchodcov, ktorým to trvalo niekoľko minút, niekedy aj hodín. Tento editor nemá problém ani s rozsiahlymi ontológiami, ktoré sa považovali za príliš náročné. [13]

3 VÝSLEDKY PRÁCE

3.1 Analýza doménovej oblasti

V tejto kapitole sa zameriame priamo na jednotlivé programovacie jazyky, ktoré budeme používať pri vytváraní ontológie. Budeme sa tu zameriavať na analýzu vlastností a využiteľnosť programovacích jazykov, zároveň ich klasifikáciou na základe signifikantných znakov.

Na začiatku program ako taký fungoval na princípe núl a jednotiek. Počítač bral tieto využívané nuly a jednotky a príkazy, na základe ktorých pracoval. Bola to prvá generácia programovacích jazykov. Tento spôsob programovania bol však veľmi nepraktický, nakoľko ak sme chceli, aby počítač splnil danú úlohu, musel programátor napísať veľké množstvo kombinácie núl a jednotiek. Z týchto dôvodu, začali vznikať nové programovacie jazyky. Jazyky mali kratší kód a boli pre používateľa zrozumiteľnejšie. Dnes už máme preto programovanie uľahčené, pretože môžeme použiť predpripravené jazyky. Tieto jazyky vedie úplne samé prepísať program do jazyku strojového.

Samotný program sú postupe zapísané inštrukcie pre počítač tak, aby mu rozumel a využíva sa na vykonanie úlohy.

Počas programovania sa vytvára algoritmus a následne aj samotný program. Programovaním môžeme rozumieť aj akýkoľvek zásah do kódu ako je testovanie alebo jeho úprava či údržba. Na všetky tieto zásahy do programu či samotnú tvorbu programu sa využíva programovací jazyk. V dnešnej dobe už existuje množstvo programovacích jazykov, takže používateľ si môže vybrať ten, ktorý mu najviac vyhovuje a zároveň sa hodí k práci, ktorú vykonáva. Nižšie sme si opísali delenie, na základe ktorého sa používateľ môže pre konkrétny programovací jazyk rozhodnúť.

Programovacie jazyky využívajú typy a premenné, pričom premenné patria medzi najdôležitejšie prvky programu. Na základe typov sa označuje, aké dáta sú uložené v premenných. Ďalej poznáme operátory, ktoré vykonávajú operácie medzi sebou. Operácie môžu byť bitové, matematické alebo logické. V neposlednom rade nesmieme zabudnúť na príkazy, ktoré obsahujú programovací jazyk. V zásade platí, že jeden riadok príkazu je jeden riadok programu. Pod pojmom príkaz si môžeme predstaviť napríklad toky, cykly alebo vetvenie. [15]

Za účelom splnenia cieľov opísaných v druhej kapitole, sme si vybrali niekoľko programovacích jazykov. Či už ide o jazyky staršie alebo nové. Každý z týchto jazykov má prostredie, v ktorom sa vytvára, syntax aj svoje využitie.

Programovacie jazyky sa môžu deliť podľa rôznych kritérií. My sme si v diplomovej práci vybrali tri základné z nich.

1. Podľa formy spracovania

- Kompilované programovacie jazyky- jazyk spočíva v tom, že pred samotným spustením programu, je nutné preložiť zdrojový kód pomocou kompilátoru do strojového kódu. Až po tomto kroku je možné program spustiť. Tento postup je rýchlejší, ale kladie sa pri ňom dôraz na presnosť písaného kódu.
- Interpretované programovacie jazyky- sú to jazyky, ktoré sa prekladajú už za behu programu. Musia obsahovať špeciálny program, ktorý sa nazýva interpret a samotný zdrojový kód. Tieto jazyky bývajú síce pomalšie, nedbá sa tu však taký veľký dôraz na presnosť.

2. Podľa prístupu k dátam

- Imperatívne programovacie jazyky- rieši danú úlohu pomocou algoritmov

- Deklaratívne programovacie jazyky- na programovanie sa využívajú definície

3. Podľa koncepcie

- Multi-paradigmatické programovacie jazyky- spôsob programovania, ktorý nepodporuje len jednu programovaciu paradigmu ale viac. Používateľ teda môže zlučovať konštruktory podľa paradigmy, ktoré mu vyhovujú a samozrejme, ktoré jazyk podporuje
- Objektovo orientované programovacie jazyky- tieto programovacie jazyky obsahujú objekty. Jednotlivé objekty majú svoje vlastnosti, udalosti a metódy, na základe ktorých vykonávajú činnosť, pre ktorú boli naprogramované
- Štruktúrované programovacie jazyky- algoritmus sa tu rozdeľuje na niekoľko častí, ktoré sú na konci spojené do jednej
- Funkcionálne programovacie jazyky- nepopisuje spôsob riešenia ale len to, čo sa bude riešiť. Využíva funkcie.
- Logické programovacie jazyky- druh programovacieho jazyka, ktorý využíva logické funkcie.

Pre lepšiu vizualizáciu sme si vytvorili tabuľku s prehľadom vybraných programovacích jazykov. Z tejto tabuľky sme neskôr vytvárali ontológiu.

Tabuľka 1 Analýza programovacích jazykov (Zdroje 17 až 21)

Názov	Vlastnosti	Podľa formy spracovania	Podľa prístupu k dátam	Podľa koncepcie
ADA	• považuje sa za nasledovníka jazyka Pascal • podporuje štruktúrované aj paralelné programovanie	kompilovaný programovací jazyk	imperatívny programovací jazyk	multi-paradigmatický programovací jazyk
ALGOL	• využíva sa na riešenie vedeckých a technických úloh • považuje sa za	kompilovaný programovací jazyk	imperatívny programovací jazyk	objektovo orientovaný programovací jazyk

	základ moderných programovacích jazykov			
AWK	• syntax podobná C • podporuje prácu s regulárnymi výrazmi, asociatívnymi poliam • vhodná voľba pre písanie programov na spracovanie textových súborov	interpretovaný programovací jazyk	imperatívny programovací jazyk	štruktúrovaný programovací jazyk
BASIC	• malá náročnosť na používateľa • nie je vhodný pre zložité úlohy	kompilovaný programovací jazyk	imperatívny programovací jazyk	štruktúrovaný programovací jazyk
C	• jazyk si používateľ jednoducho osvojí • programy sa jednoducho píšu • základný jazyk pre začiatočníkov • obsahuje: kompilátory, sieťové ovládače, textové procesory, databázové systémy	kompilovaný programovací jazyk	imperatívny programovací jazyk	štruktúrovaný programovací jazyk
C++	• rýchle spracovanie informácií • kompilačný mechanizmus	kompilovaný programovací jazyk	imperatívny programovací jazyk	multi-paradigmatický programovací

	• rozsiahla štandardná knižnica • veľká ponuka online podpory pre používateľov • skvelý pre začiatočníkov			jazyk
C#	• založený na jazykoch C, C ++ a Java • ideálny pre začiatočníkov • deklaratívne, funkčné objektovo orientované komponenty	kompilovaný programovací jazyk	deklaratívny programovací jazyk	funkcionálny programovací jazyk
ColdFusion	• intuitívna práca pri písaní • rôzne funkcie a možnosti zo strany serveru • výkon a flexibilita	kompilovaný programovací jazyk	deklaratívny programovací jazyk	logický programovací jazyk
Delphi	• vytvorené na základe jazyku Pascal • využíva Visual Component Library a tiež Component Library for Cross Platform • poskytuje možnosť	kompilovaný programovací jazyk	imperatívny programovací jazyk	objektovo orientovaný programovací jazyk

	prepojenia s databázou • tvorba a použitie komponent • možnosť použitia vlastných správ pre vyvolávanie udalostí jednotlivých tried • objektový model nie je závislý na počte implementácií jednotlivých tried			
Eiffel	• umožňuje viacnásobné a opakované dedenie • podporuje dynamické určovanie typov • ponúka možnosť rozširovať kód o moduly napísané v iných jazykoch	kompilovaný programovací jazyk	imperatívny programovací jazyk	objektovo orientovaný programovací jazyk
GML	• určený pre použitie v programe vytvorenom s cieľom vytvárať hry	interpretovaný programovací jazyk	imperatívny programovací jazyk	objektovo orientovaný programovací jazyk
HTML	• jednoduché použitie • nevyžaduje zložité programy na tvorbu html	kompilovaný programovací jazyk	imperatívny programovací jazyk	štruktúrovaný programovací jazyk

	súborov, stačí Poznámkový blok alebo Notepad++ • poskytuje viacero verzií			
Java	• nie je závislí od platformy • rýchli a jednoduchý prenos aplikácie • rozsiahla sieťová knižnica • vhodný na vývoj hier alebo aplikácií	kompilovaný programovací jazyk	imperatívny programovací jazyk	objektovo orientovaný programovací jazyk
JavaScript	• univerzálne využitie • jednoduché osvojovanie si základných finkcií • viacnásobné rámce • komplexná Javascript knižnica • s jazykom HTML a CSS vytvorili piliere webového dizajnu	interpretovaný programovací jazyk	deklaratívny programovací jazyk	funkcionálny programovací jazyk
Objective-C	• zvýšená flexibilita pri dynamickom písaní • skvelý prvý jazyk pre začínajúcich programátorov	kompilovaný programovací jazyk	imperatívny programovací jazyk	objektovo orientovaný programovací jazyk
Pascal	• pôvodne bol učený k výučbe	kompilovaný programovací	imperatívny programovací	štruktúrovaný programovací

	programovania • neskôr sa začal používať pre programovanie reálnych aplikácií • jeho základné príkazy sú for, while, if, then, else	jazyk	jazyk	jazyk
Pawn	• open sourceprogramovací jazyk	kompilovaný programovací jazyk	imperatívny programovací jazyk	štruktúrovaný programovací jazyk
Perl	• ponúka možnosť napísať prakticky čokoľvek • v praxi sapoužíva k písaní CGI skript	interpretovaný programovací jazyk	imperatívny programovací jazyk	multi-paradigmatický programovací jazyk
PHP	• open-source aplikácia • jednoduchý na naučenie • môže byť použitý na všetkých operačných systémoch a webových serveroch • je možné v ňom využívať moduly a knižnice, a tie zaisťujú dynamický vývoj softvéru • obsahuje funkcie,	interpretovaný programovací jazyk	imperatívny programovací jazyk	multi-paradigmatický programovací jazyk

	ktoré z neho robia efektívnejší jazyk na vývoj webu bez nutnosti písania dlhšieho kódu			
PL/SQL	• možnosť zaviesť dáta z externých webových služieb • dáta je možné získať a použiť z ďalších cloudových služieb • jedná sa o robustný vývojársky nástroj • výhodou je možnosť vytvorenia zrozumiteľných klientských aplikácií • vykonávanie komplexných databázových operácií	kompilovaný programovací jazyk	imperatívny programovací jazyk	štruktúrovaný programovací jazyk
Prolog	• využíva logické výroky • pri problémoch, ktoré sa dajú popísať vo forme objektov a vzťahov • program pripomína výpis vzťahov medzi objektmi	interpretovaný programovací jazyk	deklaratívny programovací jazyk	logický programovací jazyk

Python	• najžiadanejší programovací jazyk na trhu práce • ľahko pochopiteľná syntax • jednoduchá integrácia s webovými službami • nenáročné čítanie kódu • štandardná knižnica	interpretovaný programovací jazyk	imperatívny programovací jazyk	multi-paradigmatický programovací jazyk
Ruby	• univerzálny programovací jazyk, ktorý sa používa na vývoj webových aplikácií • jednoduchý a dynamický jazyk • flexibilný jazyk	interpretovaný programovací jazyk	imperatívny programovací jazyk	multi-paradigmatický programovací jazyk
Rust	• syntax je podobná ako syntaxC a C++	kompilovaný programovací jazyk	imperatívny programovací jazyk	multi-paradigmatický programovací jazyk
SQL	• jazykom pre prístup k relačným databázam • široké možnosti dátových štruktúr	kompilovaný programovací jazyk	deklaratívny programovací jazyk	štruktúrovaný programovací jazyk

	• rýchlejšie načítanie veľkého počtu databázových záznamov • zachovanie integrity databáz • adaptabilný pre akékoľvek prostredie • jednoduchá syntax • voľný a ľahko dostupný			
Visual Basic	• udalosťami riadený • zodpovedajúce vývojové prostredie • autorom je Microsoft	kompilovaný programovací jazyk	imperatívny programovací jazyk	objektovo orientovaný programovací jazyk

3.2 Výber editora

Pre účel diplomovej práce sme si vybrali nástroj na tvorbu ontológie Protégé. Tento nástroj je komplexný pre prácu s ontológiami. Výhodou je, že tento editor je open-source, teda je voľne dostupný pre širokú verejnosť. Jedinou podmienkou je prihlásenie sa na stránke tvorcu tohto editora a označenie účelu, za ktorým si editor Protégé sťahujeme.

Práca v ňom je jednoduchá a intuitívna, čo používateľovi ušetrí veľa času. Taktiež sú k Protégé dostupné rôzne učebné materiály či návody. Do tejto kategórie patrí aj prehľadnosť pri práci s týmto editorom.

Model sa v ňom môže jednoducho rozširovať a v budúcnosti sa môže ďalej využívať.

3.3. Ontológia programovacích jazykov

Ako program pre vytvorenie ontológie sme si vybrali Protégé. Práca v ňom je intuitívna a ontológia jednoducho pochopiteľná pre používateľa.

Samotná inštalácia editora Protégé bola veľmi jednoduchá. Celý systém je postavený na tom, že je freeware. To v praxi znamená, že autor editoru Protégé umožňuje jeho používanie zadarmo, podporuje aj jeho šírenie. Nemáme však prístup k zdrojovému kódu, a teda ho nemôžeme meniť. Licencia spadá pod open-source Mozilla Public Licenc, čo nám dáva možnosť voľného stiahnutia z internetu.

Editor je vytvorený pomocou programovacieho jazyka Java, je ním implementovaný. Pre jeho správne fungovanie je nutné aby bolo na danom zariadený nainštalované aj Java Virtual Machine. Ani pri tomto kroku by však nemal nastať problém, nakoľko je možné ho nainštalovať spolu s editorom Protégé priamo, ako balíček.

Ako bolo spomínané vyššie Protégé je veľmi intuitívny na prácu. Je teda jednoduchý aj pre bežného používateľa. Jediný problém, ktorý pri tvorbe vznikol bol, že sa graf v OWLViz nezobrazoval správne. Tento problém však vznikol len prvotnou nedostatočnou informovanosťou o editore.

Editor Protégé totižto obsahuje rôzne balíčky. Medzi tieto balíčky patrí aj OWLVIz. OWLViz je grafický balíček, ktorý zabezpečuje zobrazovanie grafu tried a podtried. Pre správne zobrazovanie tohto grafu je nutné si nainštalovať GraphViz, inak uvidíme iba prázdne okienko, ktoré zobrazuje vždy len jednu triedu.

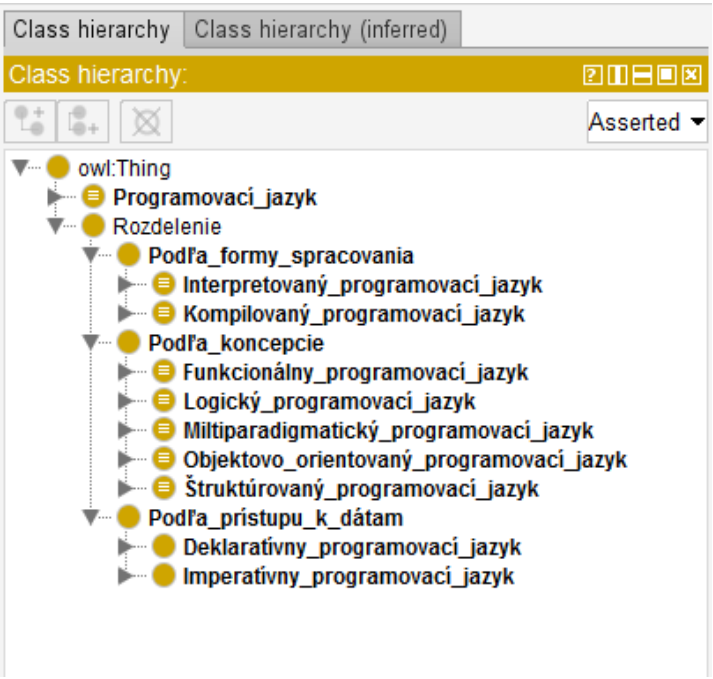
Po prevedený všetkých týchto krokov môžeme začať používať editor Protégé. Pri jeho otvorení sa nás opýta či chceme otvoriť nový projekt alebo načítať ten, ktorý sme už vytvorený mali. Po otvorení sa nám zobrazí základné menu editora, ako môžeme vedieť na obrázku 3. Hore na lište si môžeme prepínať medzi oknami, ktoré potrebujeme. Nás budú zajímať najme tri z nich.

Prvé okno, ktoré je nutné spomenúť je okno OWLclasses. v tomto okne máme na ľavej strane triedy a podtriedy. Hlavná nadtrieda je vždy určená pod názvom owl:Thing. Tento názov nevieme meniť a triedu ani nemôžeme vymazať. Zapisujú sa tu teda triedy a nadtriedy, ktoré sa neskôr zobrazujú v grafe. Rovnako sa tu zapisujú vzťahy medzi jednotlivými triedami, určuje sa ich nadradenosť, podradenosť či rovnosť tried. Môžeme sem napísať aj vlastnosti alebo poznámky k danej triede ako komentár. Druhým oknom je

OWLproperties. V tomto okne sa zapisujú dátové typy, vlastnosti a anotácie k daným triedam a podtriedam. V neposlednom rade pre nás je tu okno OWLviz, ktoré sme si už vyššie opisovali. Slúži ako grafický editor, v ktorom môžeme vidieť zobrazený graf vytvorený z tried a podtried.

Ako sme už spomínali vyššie, prvým krokom v ontológii bolo vytvoriť si prehľad programovacích jazykov, v našom prípade sme použili tabuľku.

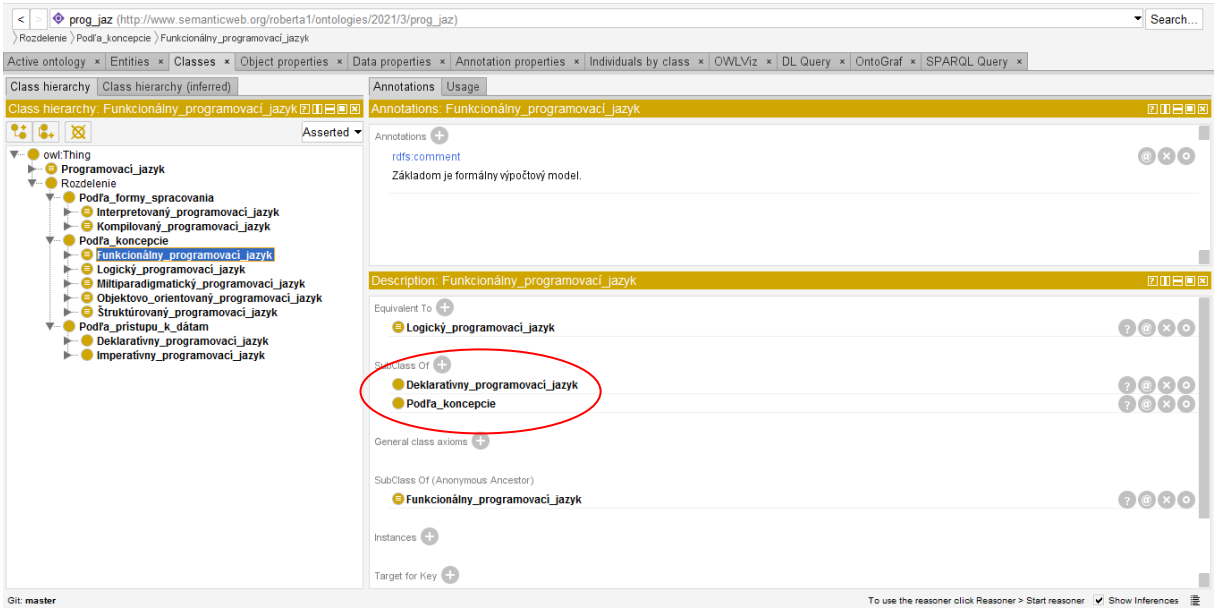
V programe Protégé sme si vytvorili triedy. Tieto triedy sme vytvárali podľa rozdelenia programovacích jazykov.



Držali sme sa troch základných rozdelení a to podľa formy spracovania, podľa koncepcie a podľa prístupu k dátam. Ako môžeme vidieť na obrázku č.2. Podľa formy spracovania sa programovacie jazyky delia na interpretované a kompilované programovacie jazyky. Ďalšie delenie podľa prístupu k dátam je na deklaratívne a imperatívne programovacie jazyky.

Obrázok 2 Triedy v Protégé

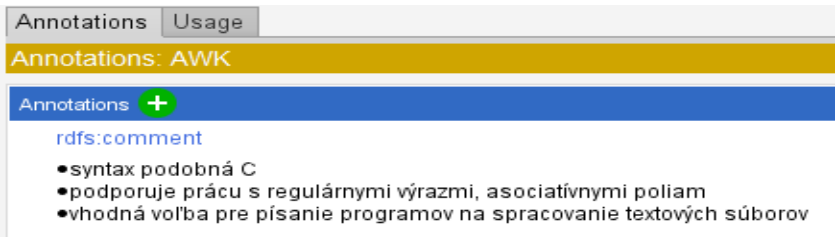
Posledné delenie, ktoré sme opisovali, bolo delenie na základe koncepcie. Pri tomto delení sme museli brať ohľad aj na to, či konkrétny programovací jazyk patrí do deklaratívnych alebo imperatívnych programovacích jazykov. Deklaratívne programovacie jazyky sa delia na funkcionálne a logické programovacie jazyky. Imperatívne programovacie jazyky sa delia na multi-paradigmické, objektovo orientované a štruktúrované programovacie jazyky. Ako môžeme vidieť na obrázkoch uvedených nižšie.



Obrázok 3 Trieda funkcionálneho programovacieho jazyka v Protégé

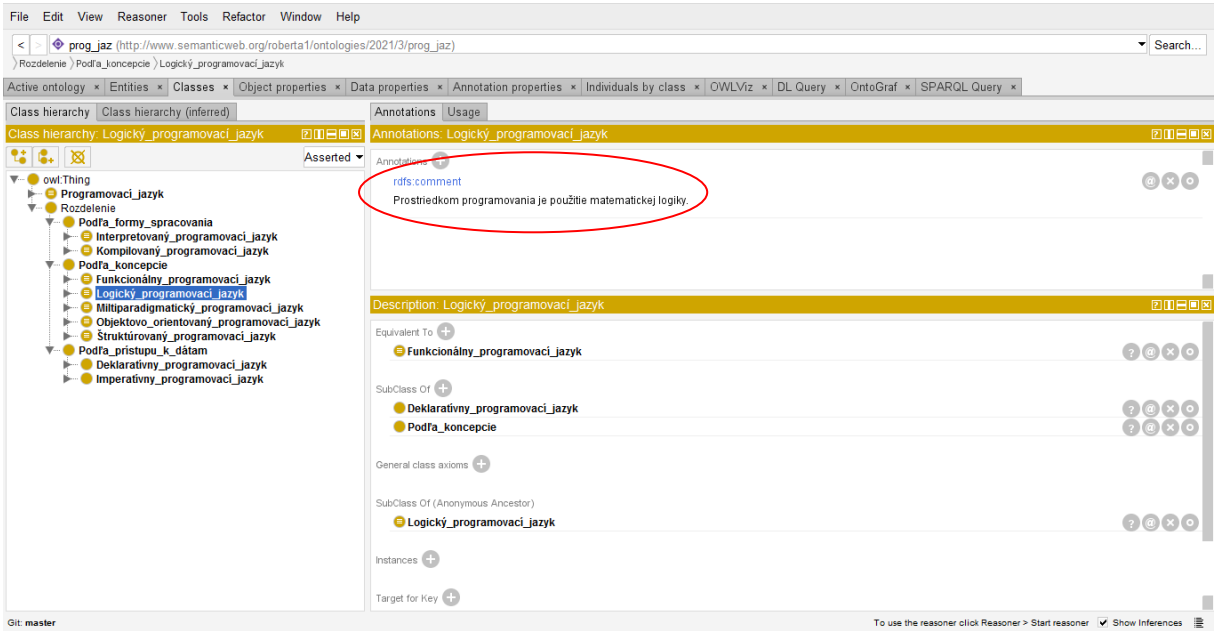
Pre správne vytvorenie vizualizácie ontológie v programe Protégé sme určili všetkým druhom programovacích jazykov v rozdelení podľa koncepcie takzvanú SubClass, čím sme vlastne dosiahli, aby sa nám konkrétny programovací jazyk viazal na typ rozdelenia podľa koncepcie a zároveň aj na typ rozdelenia podľa prístupu k dátam. Príklad SubClass môžeme vidieť na obrázku č.3 zvýraznené červeným.

Ďalšou možnosťou Protégé je pridať komentár k jednotlivým triedam. Túto funkciu sme využili na to, aby sme popísali jednotlivé delenia. Vďaka tejto možnosti nie je nutné, aby používateľ musel dohľadávať význam delenia. Príkladom je komentár označený červeným na obrázku č.5.

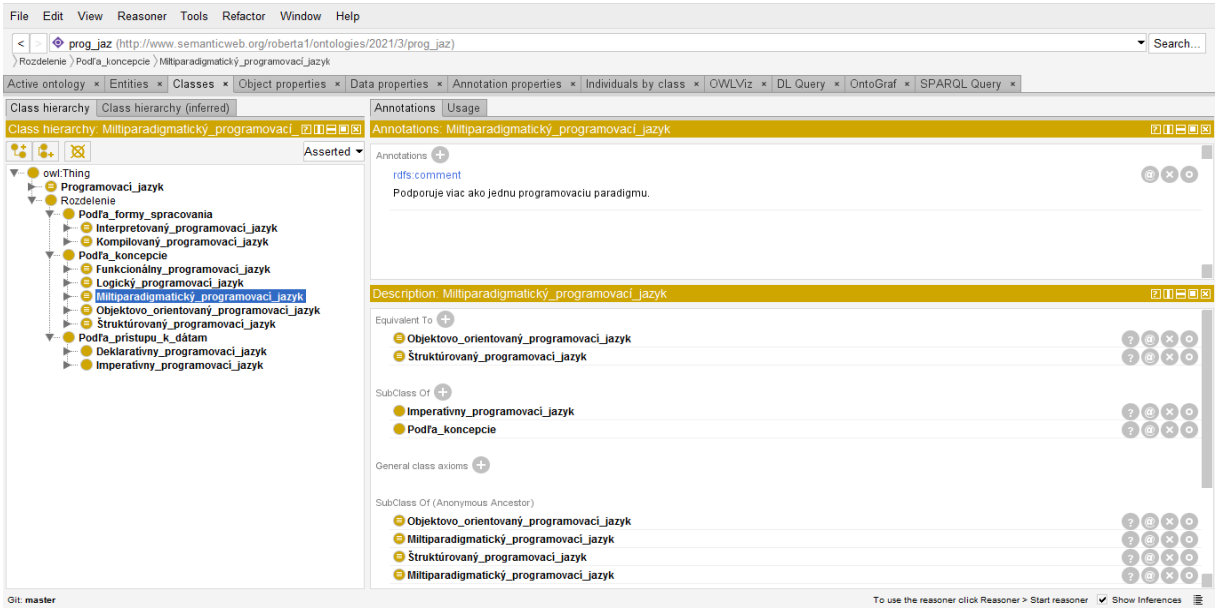


Obrázok 4 Komentár programovacieho jazyka AWK v Protégé

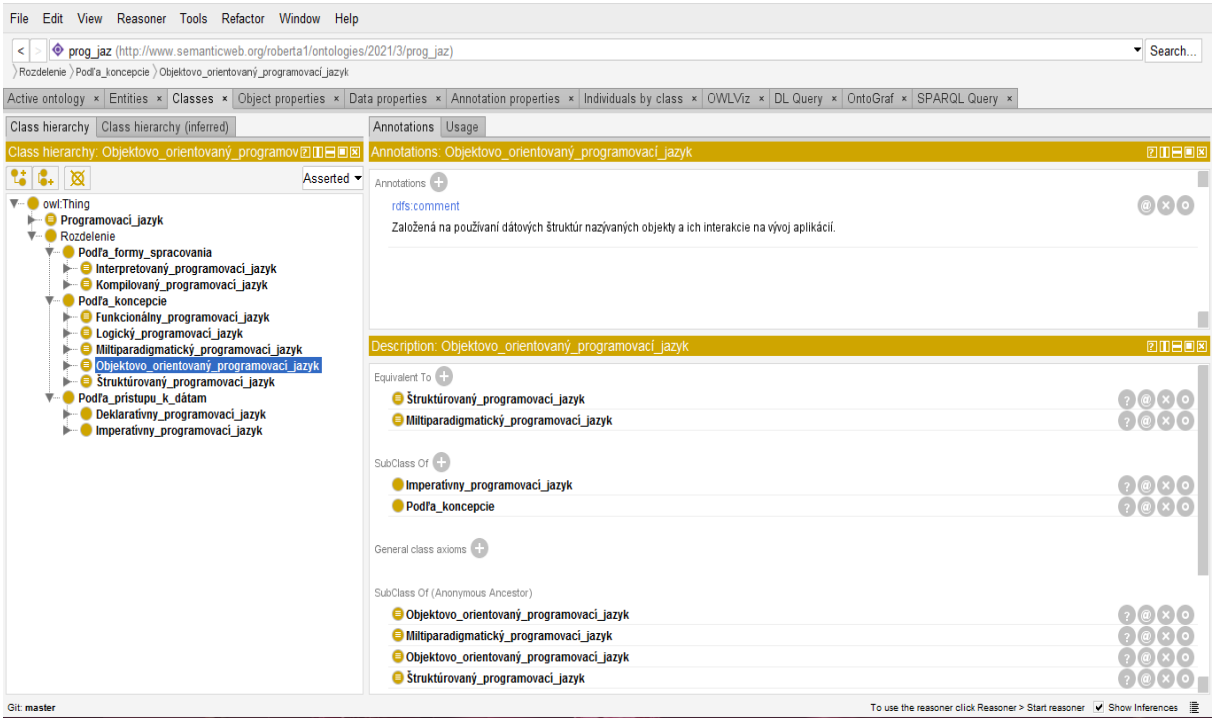
Túto možnosť sme využili aj pri popisovaní programovacích jazykov. Pri každom jazyku sme si vytvorili komentár. Tento komentár obsahoval základné znaky jednotlivých programovacích jazykov. Vlastnosti sme čerpali z už vopred vytvorenej tabuľky. Ako príklad si sme si uviedli programovací jazyk AWK na obrázku č.4.



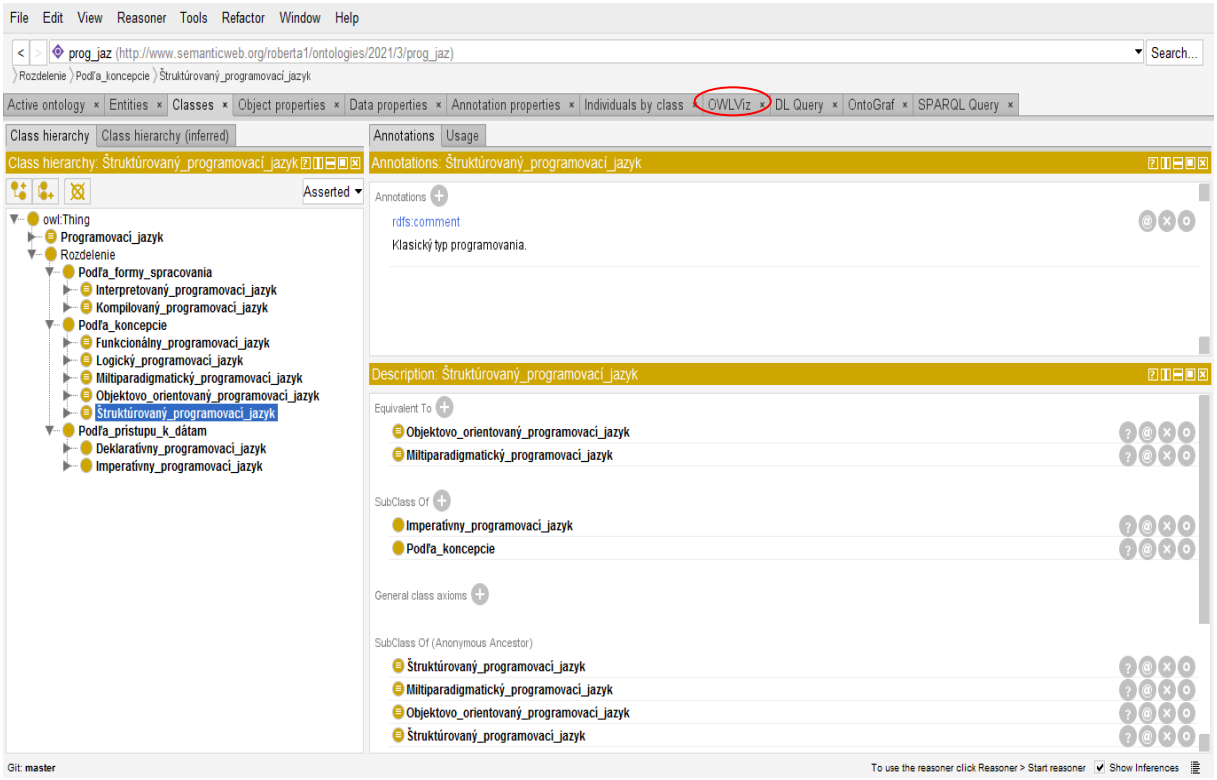
Obrázok 5 Trieda logického programovacieho jazyka v Protégé



Obrázok 6 Trieda multiparadigmatického programovacieho jazyka v Protégé



Obrázok 7 Trieda objektovo orientovaného programovacieho jazyka v Protégé



Obrázok 8 Trieda štrukrúrovaného programovacieho jazyka v Protégé

Na základe tried a ich vzťahov, ktoré sme si určili, sme si vytvorili vizualizáciu. Túto vizualizáciu nájdeme v OWLViz, ktoré môžeme vidieť na obrázku č.8.

Základné rozdelenie a zároveň aj vizualizáciu môžeme vidieť na obrázku č.9. Ako môžeme vidieť sú tu nami vybrané programovacie jazyky v samostatnej časti. Druhú časť sme nazvali rozdelenie. V tejto časti môžeme vidieť základné nerozvetvené rozdelenie

programovacích jazykov, ako sme si ho popísali vyššie, a zároveň ho môžeme vidieť aj v tabuľke programovacích jazykov.

Po rozkliknutí jednotlivých kategórií sa nám zobrazia ich podrobnejšie rozdelenia.

Obrázok č.10 obsahuje rozdelenie podľa formy spracovania. Ako môžeme vidieť nachádza sa tu rozdelenie na kompilovaný a interpretovaný programovací jazyk. V tomto stave môžeme vidieť už aj konkrétne programovacie jazyky patriace do jednotlivých tried. Je tu jasne určené pomocou šípok, ktorý programovací jazyk je kam zaradený. Vďaka tomu si môžeme vybrať jazyk podľa formy a nie je teda nutné rozbaľovať jednotlivé jazyky samostatne.

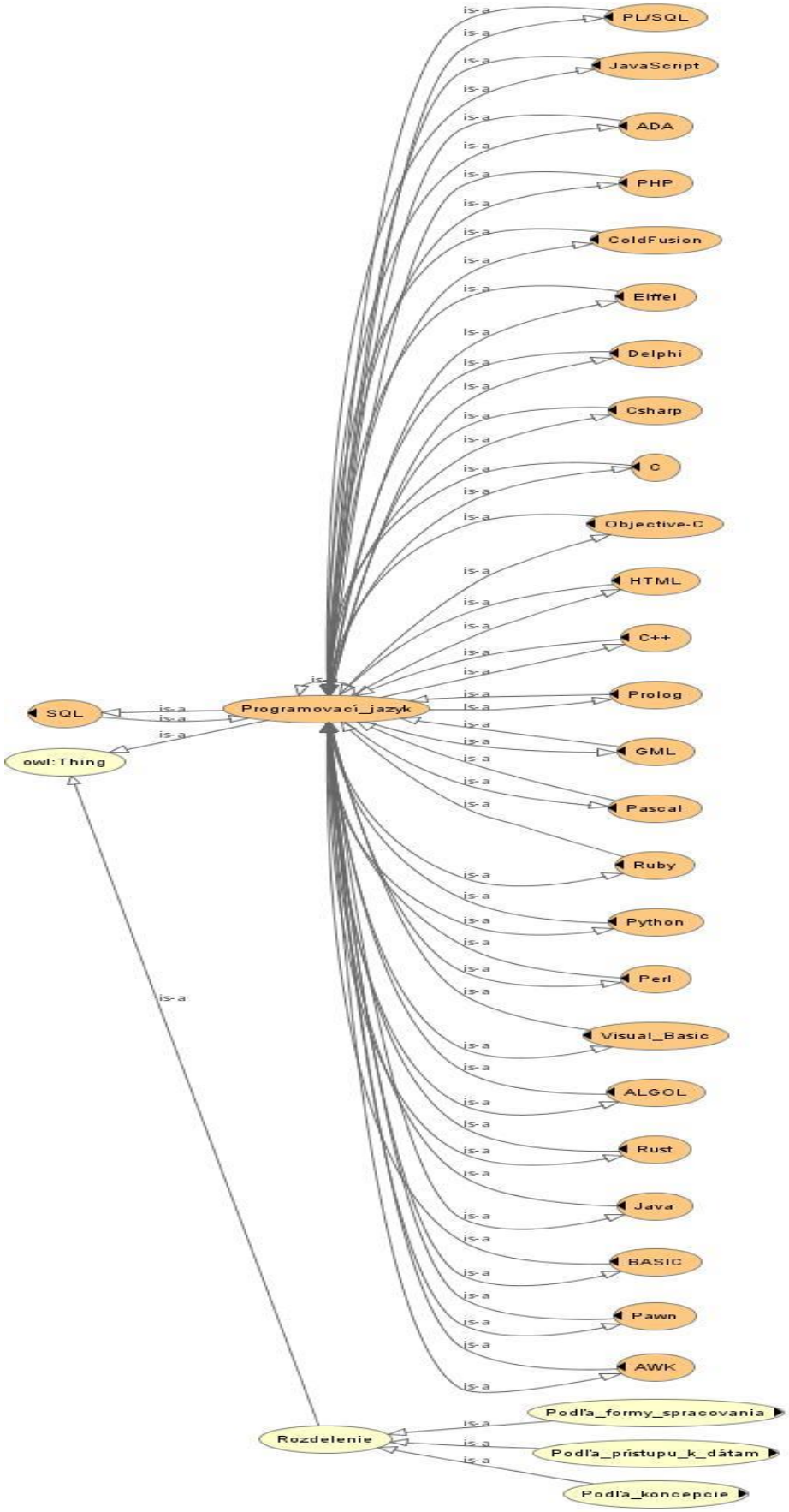
Ďalší obrázok č.13 v prílohách zobrazuje rozdelenie podľa prístupu k dátam. Tu môžeme vidieť komplikovanejšiu štruktúru, kde je zobrazené nielen rozdelenie na deklaratívne a imperatívne programovacie jazyky, je tu zaradené aj rozdelenie podľa koncepcie. Tu sa nám zobrazuje rozdelenie podtried, ktoré sme si určili pri vytváraní tried. Môžeme ho vidieť zobrazené na obrázkoch č.3 a 5 až 8.

Posledné zvolené rozdelenie je rozdelenie podľa koncepcie zobrazené na obrázku č.11. Môžeme ho vidieť aj na obrázku č.13 v prílohách ako sme spomínali vyššie. Tu je však zobrazené samostatne a jeho prehľadnosť je teda väčšia.

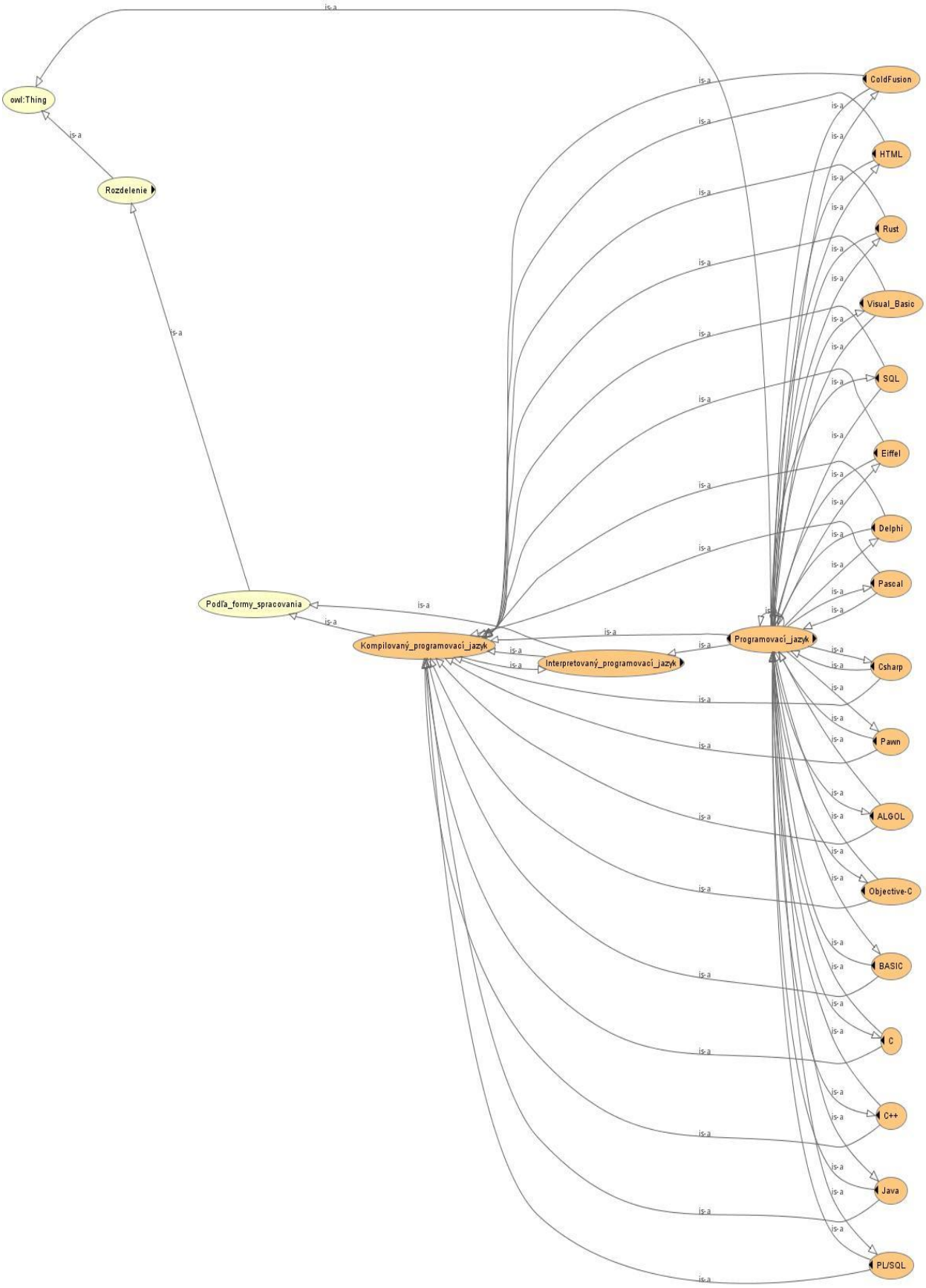
Všetky programovacie jazyky máme zobrazené v týchto deleniach. Je teda jednoduché sa k ním dostať a vidíme aké programovacie jazyky obsahujú. Toto delenie môžeme použiť, ak vieme, aké kritériá pri programovaní potrebujeme a vieme sa teda rozhodnúť, z ktorej kategórie by nám programovací jazyk najviac vyhovoval pre naše účely.

Môže sa však vyskytnúť prípad, že vieme, o aký programovací jazyk máme záujem, ale nevieme či je vhodný pre náš účel. V takejto situácii si vieme otvoriť konkrétny programovací jazyk v triedach Programovacie jazyky. Tu sa nám zobrazí presné zaradenie programovacieho jazyka ako môžeme vidieť na uvedenom príklade na obrázku č.12. V tomto prípade sme si zvolili jazyk ALGOL. Ako môžeme vidieť, jasne sa nám zobrazujú zelené šípky a teda vieme, že programovací jazyk ALGOL je kompilovaný, imperatívny a objektovo orientovaný.

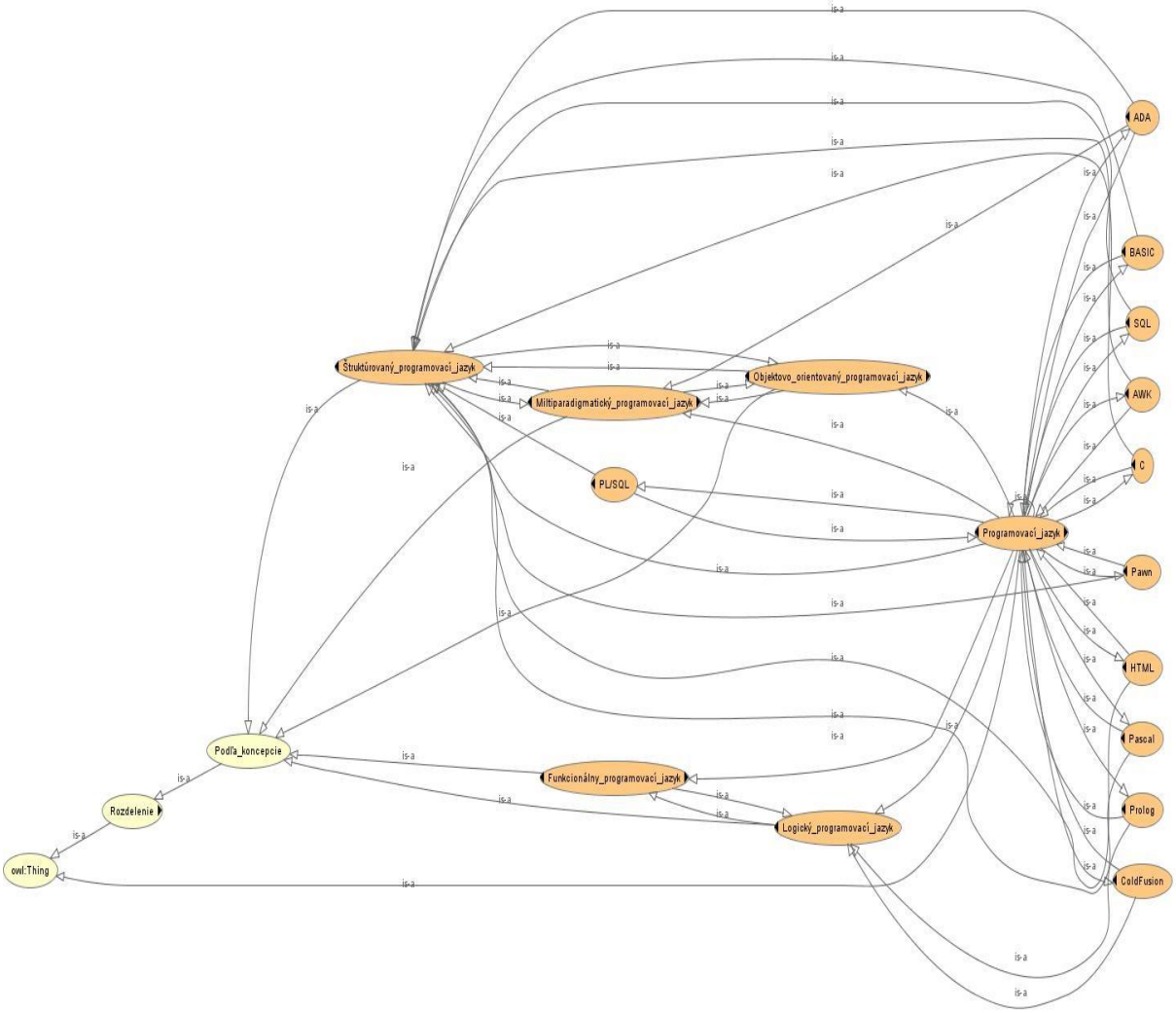
Ontológia v Protégé nám teda môže pomôcť pri určovaní potrebného programovacieho jazyka v oboch týchto prípadoch .



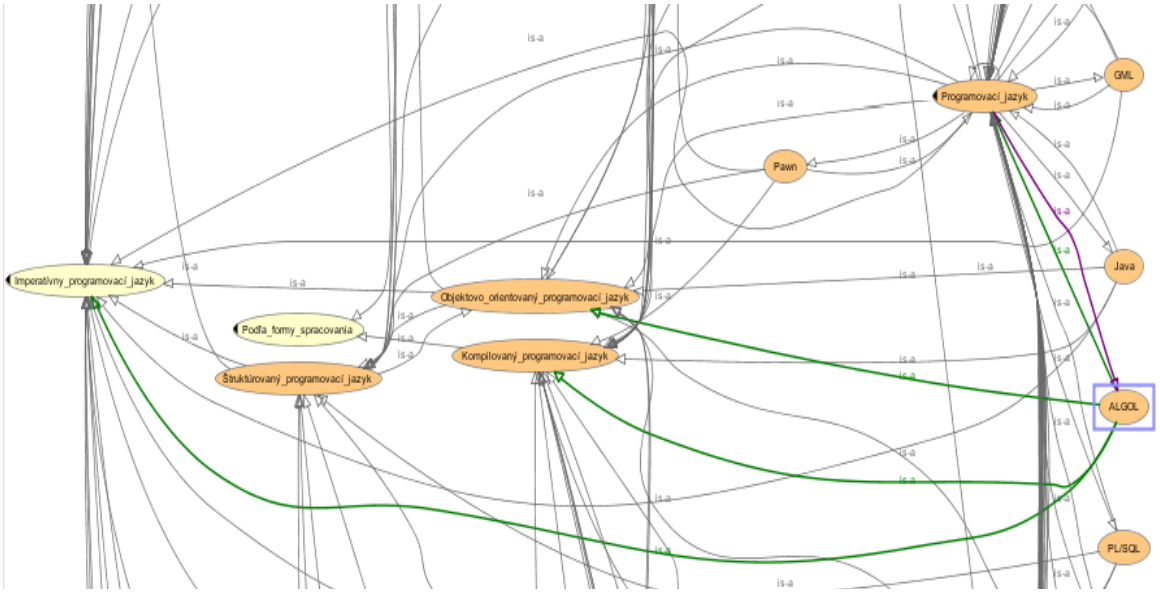
Obrázok 9 Základné zobrazenie schémy v OWLViz



Obrázok 10 Rozdelenie podľa formy spracovania v OWLViz



Obrázok 11 Rozdelenie podľa koncepcie v OWLViz



Obrázok 12 Programovací jazyk ALGOL v OWLViz

3.4 Diskusia

Ako sme už spomínali, cieľom diplomovej práce bolo vytvorenie modelu ontológie programovacích jazykov na základe ich signifikantných znakov. Ontológia a jej model teda obsahujú základné znaky programovacích jazykov, ako sú vlastnosti, niektoré triedy, výhody či nevýhody, kde je vhodné ich využiť a v neposlednom rade ich delenie do skupín.

Pre používateľa môže slúžiť ako jednoduchá a prehľadná príručka, ktorá obsahuje základné informácie a vlastnosti o jednotlivých programovacích jazykoch. Stačí si nájsť programovací jazyk, ktorý potrebujeme a pri jednoduchom rozkliknutí zistíme jeho vlastnosti.

Keďže ontológia, ktorú sme vytvorili je stavaná na základe delenia programovacích jazykov podľa troch kategórií, vieme sa jednoducho zorientovať aj v tom, kam by sme daný programovací jazyk zaradili. Rovnako ako pri programovacích jazykoch, môže používateľ jednoduchým kliknutím na konkrétne rozdelenie vidieť jeho vlastnosti. Napríklad pre jazyk PL/SQL môžeme zistiť nasledujúce vlastnosti:

- možnosť zaviesť dáta z externých webových služieb
- dáta je možné získať a použiť z ďalších cloudových služieb
- jedná sa o robustný vývojársky nástroj
- výhodou je možnosť vytvorenia zrozumiteľných klientskych aplikácií
- vykonávanie komplexných databázových operácií

Môžeme teda povedať, že vytvorený ontologický model popisujúci programovacie jazyky a ich delenie používateľovi slúži na výber správneho programovacieho jazyka pre konkrétne úlohy alebo aj v prípade ak by nevedel, kam sa programovací jazyk zaraďuje. Nakoľko pri rozbalení každého delenia vidíme aj ostatné programovacie jazyky, ktoré patria do danej skupiny, používateľ si môže vybrať iný vhodný jazyk zo skupiny, ak by potreboval jazyky porovnať alebo by bol vhodnejší na riešenie úlohy. Takýmto istým spôsobom si môže vybrať jazyk aj používateľ, ktorý vie len to, akým smerom by sa chcel v programovaní uberať.

Vytvorený ontologický model môže mať aj informatívnu úroveň. Model je pomerne prehľadný a obsahuje základné informácie, môžeme zistiť výhody a nevýhody, ktoré sprevádzajú jednotlivé programovacie jazyky či už na základe delenia, alebo na

základe jazyku samotného. Používateľ si teda môže urobiť celkový prehľad o danej problematike bez rozsiahleho hľadania informácií, čo skráti čas pri práci.

Pri otázke, ako by sme mohli vytvorený ontologický model rozšíriť, sa môžeme uberať rôznymi smermi. Jedným z nich môže byť delenie. Tri delenia, ktoré sme si v práci vybrali ,nie sú jediné. Poznáme mnoho ďalších delení, na základe rôznych znakov, črtou či vlastností, ako napríklad to či sa rieši daná úloha pomocou algoritmu alebo nie, množstvo podporovaných programovacích paradigiem, či je program založený na objektoch a podobne. V tomto prípade by sa dal model rozšíriť vo veľkej miere, zhoršila by sa však jeho prehľadnosť.

Doba sa stále vyvíja a programovacie jazyky sú v nej naplno využívané. Vyvíjajú sa aj nové technologické možnosti, ktoré vyžadujú od programu či jazyku vyššie nároky, či už časové alebo používateľské. Na základe toho by sa mohol model rozširovať o ďalšie nové programovacie jazyky, a zároveň by sa z neho mohli vylučovať tie, čo zanikli, poprípade také, ktoré sa nahradili inými jazykmi. Samozrejme vo vytvorenom aktuálnom ontologickom modeli nie sú zahrnuté všetky programovacie jazyky, vybrali sme si len tie, ktoré sú v súčasnosti reprezentatívne pre jednotlivé triedy programovacích jazykov.

Ďalšou cestou, ktorou by sa mohol model uberať by bolo jeho rozšírenie o konkrétne príklady z praxe. Napríklad ktorý programovací jazyk by bol vhodný na tvorbu hier, alebo naopak ktorý programovací jazyk by bol vhodný na riešenie matematických úloh a podobne.

ZÁVER

Diplomová práca bola zameraná na vytvorenie ontologického modelu, ktorý bude poskytovať vzťahy a atribúty programovacích jazykov, čo bolo hlavným cieľom diplomovej práce.

Na záver môžeme vyhodnotiť, že rozvíjanie ontológie bude ďalej pokračovať. Môže sa využiť v rôznych odvetviach, v tých čo sme si spomínali ako je napríklad sémantický web, ale aj v mnoho ďalších. Vytvorením ontologického modelu sme si ukázali že ontológia ako taká sa dá užitočne využiť aj v školách, pre jednoduchšie pochopenie a spracovanie veľkého množstva informácií, s ktorými sa študent stretne. Mohla by uľahčiť študentom chápania vzťahov medzi jednotlivými predmetmi, alebo vzťahov zahŕňajúcich konkrétny predmet.

V prvej časti bolo pre nás nutné aby sme sa zoznámili so samotnými pojmami ako je ontológia a v čom spočíva. Neskôr sme si opísali samotné jazyky a editory, ktoré ontológia využíva.

Pri plnení hlavného cieľu diplomovej práce bolo nutné opísať vlastnosti programovacích jazykov a ich klasifikácia. Vytvorili sme prehľadnú tabuľku, ktorá obsahuje potrebné informácie o programovacích jazykoch. Na základe analýzy, ktorú sme vytvorili, vznikal ontologický model a samotná ontológia.

Pre naše účely sme sa rozhodli využiť editor ontológie Protégé, ktorý bol pre nás najvhodnejší. Zoznámili sme sa s prácou a postupmi v editore.

Vytvorili sme prehľadný aj pre používateľa jednoducho použiteľný model na rôzne účely. Využili sme pri tom editor Protégé. Analyzovali sme všetky potrebné dáta.

Zoznam použitej literatúry

[1] SVÁTEK, Vojtěch, Ontologie a WWW, Katedra informačního a znalostního inženýrství, VŠE
Praha, 2002.

[2] AUFAURE, M., Metadata- and Ontology- Based Semantic Web Mining, 2006. Dostupné na: https://www.researchgate.net/publication/228934706_Metadata-_and_Ontology-_Based_Semantic_Web_Mining

[3] USCHOLD, Michael, Ontologies: Principles, methods and applications, článok v The Knowledge Engineering Review, 1996

[4] RAKOVSKÁ, E., *Importance of domain ontologies in business informatics*, In Trendy a inovácie v internetovej podpore podnikania a vzdelávania, Bratislava, 2013, ISBN 978-80-225-3769-8

[5] RAKOVSKÁ, E., Modelovanie znalostí v manažérskych úlohách prostredníctvom ontológií a webových služieb, 2016, ISSN 1339-987X. Dostupné na: https://www.researchgate.net/publication/315810681_MODELOVANIE_ZNALOSTI_V_MANAZERSKYCH_ULOHACH_PROSTREDNICTVOM_ONTOLOGII_A_WEBOVYCH_SLUZIEB

[6] KAPUSTÍK, I., Ontológie v prostredí sémantického webu, STU Bratislava, 2011. Dostupné na: http://www2.fiit.stuba.sk/~kapustik/ZS/Clanky0506/bellus/index.html

[7] BECKETT, D., W3C, 2014 Dostupné na: https://www.w3.org/TR/rdf-syntax-grammar/#section-Syntax

[8] Decker, S., The Semantic Web - On the respective roles of XML and RDF. IEEE Internet Computing, 2000. Dostupné na: https://www.researchgate.net/publication/2802703_The_semantic_web_-_On_the_respective_roles_of_XML_and_RDF

[9] Luke, S., Spector L., Rager D., Hendler J. : Ontology-based Web Agents, First International Conference on Autonomous Agents, 1997

[10] MCGUINNESS D.,OWL Web Ontology Language Overview, 2009

[11]SURE-VETTER, Y.,OntoEdit: Collaborative Ontology Development for the Semantic Web, Karlsruhe Institute of Technology, 2009

[12]TUDORACHE, T., NOY, N., TU, S., MUSEN, M., Supporting Collaborative Ontology Development in Protégé, International Semantic Web Conference, 2008

[13] GLIMM, B., HORROCKS, I.,MOTIK, B., STOILOS, G., WANG, Z., HermiT: An OWL 2 Reasoner, Journal of Automated Reasoning manuscript, 2014

[14] Marek Laurenčík, SQL, Grada, 2018, ISBN 978-80-271-0774-2

[15] Jindřich Kaluža, Ludmila Kalužová, Informatika, Ekopress, 2012, ISBN 978-80-869-2983-5

[16] Paul Rissen, Journalism Ontology, 2014, Dostupné na: https://www.bbc.com/ontologies/journalism

[17] Stephen Prata, Mistrovství C++, Computer Press, 2013, ISBN 978-80-251-3828-1

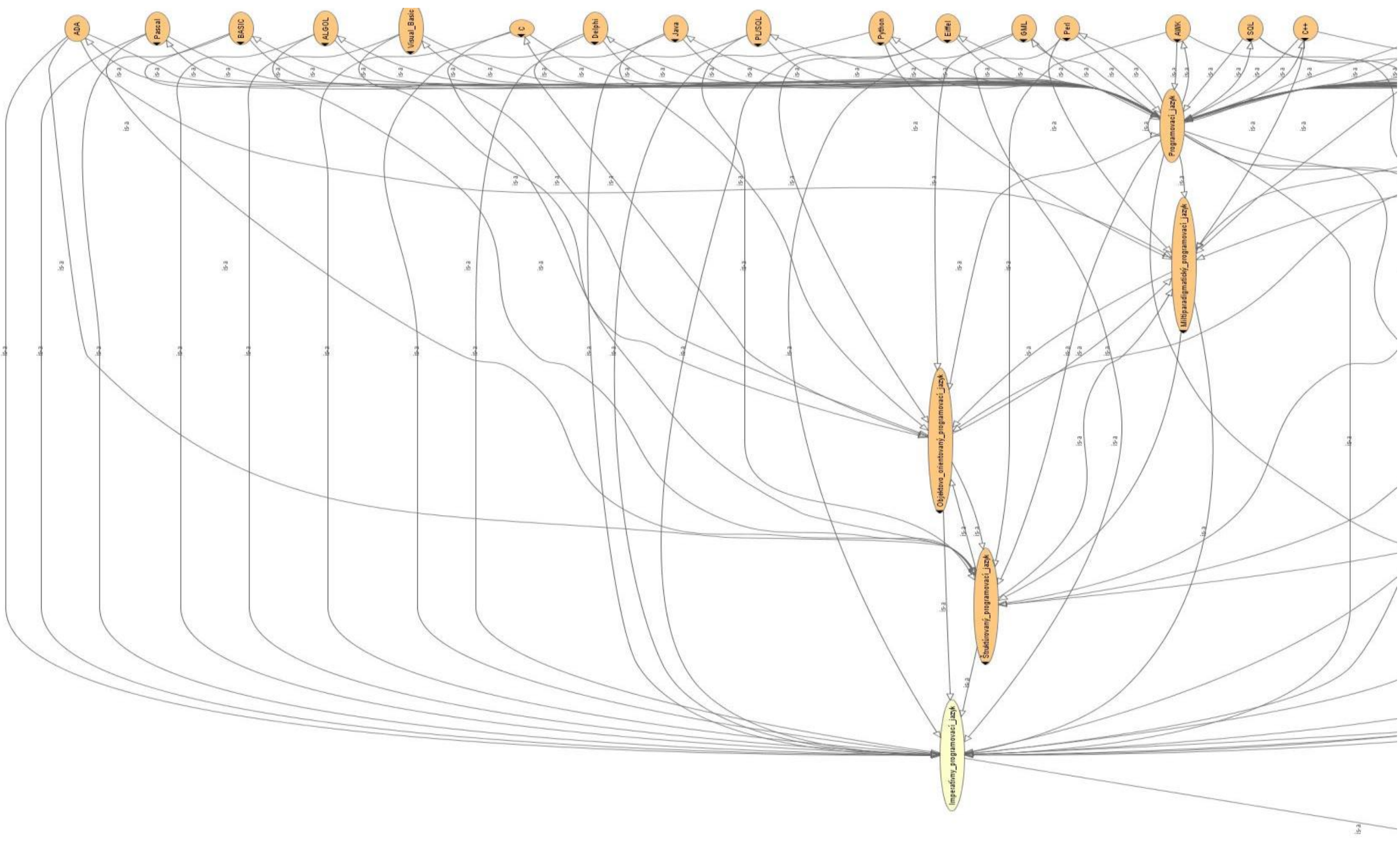
[18] Pavel Herout, Java a XML, Kopp, 2007, ISBN 807-23-2307-4

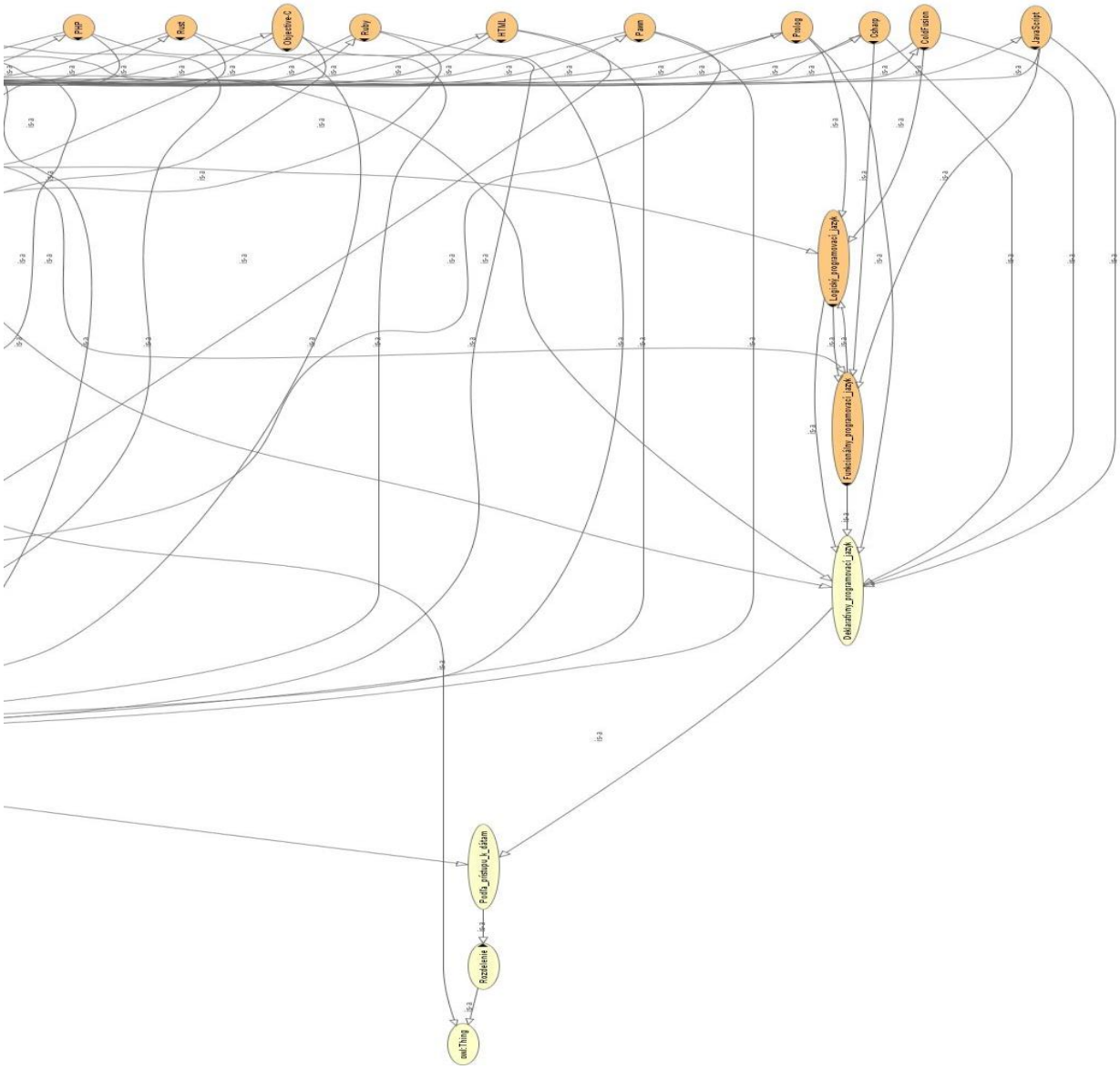
[19] Hal Fulton, Ruby, Zoner Press, 2009, ISBN 978-80-741-3018-2

[20] Benjamin C. Pierce, Types and Programming Languages, The MIT Press, 2002, ISBM 0-262-16209-1 Dostupné na: https://books.google.sk/

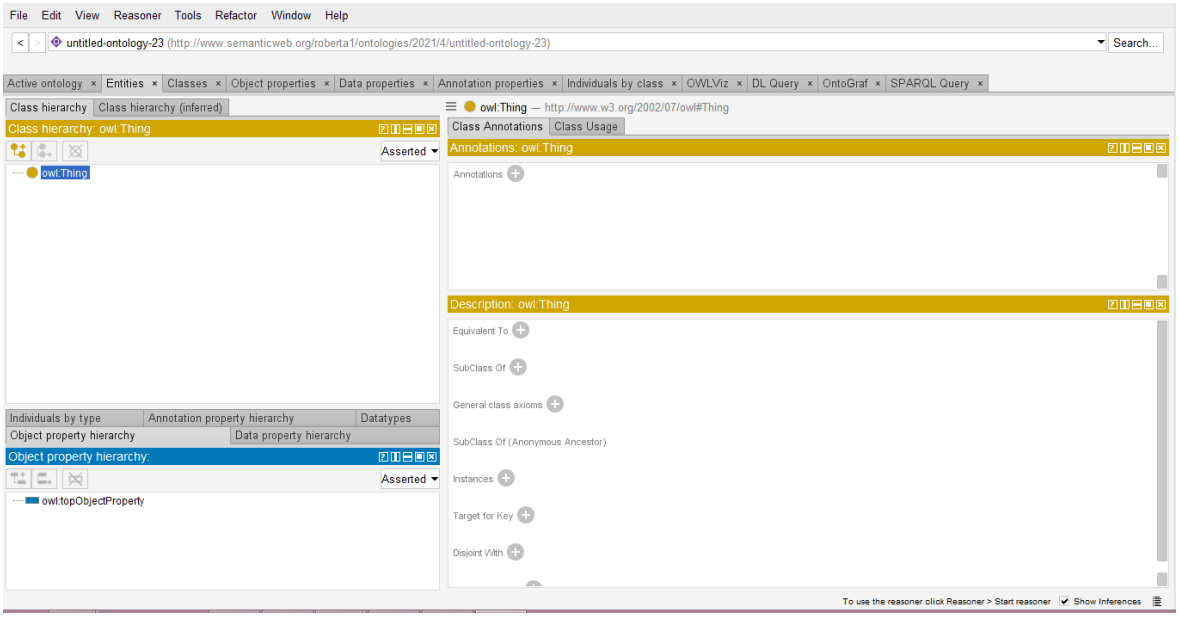
[21] Pavel Satrapa, Perl pro zelenáče, CZ.NIC, 2009, ISBN 978-80-881-4835-5

Prílohy

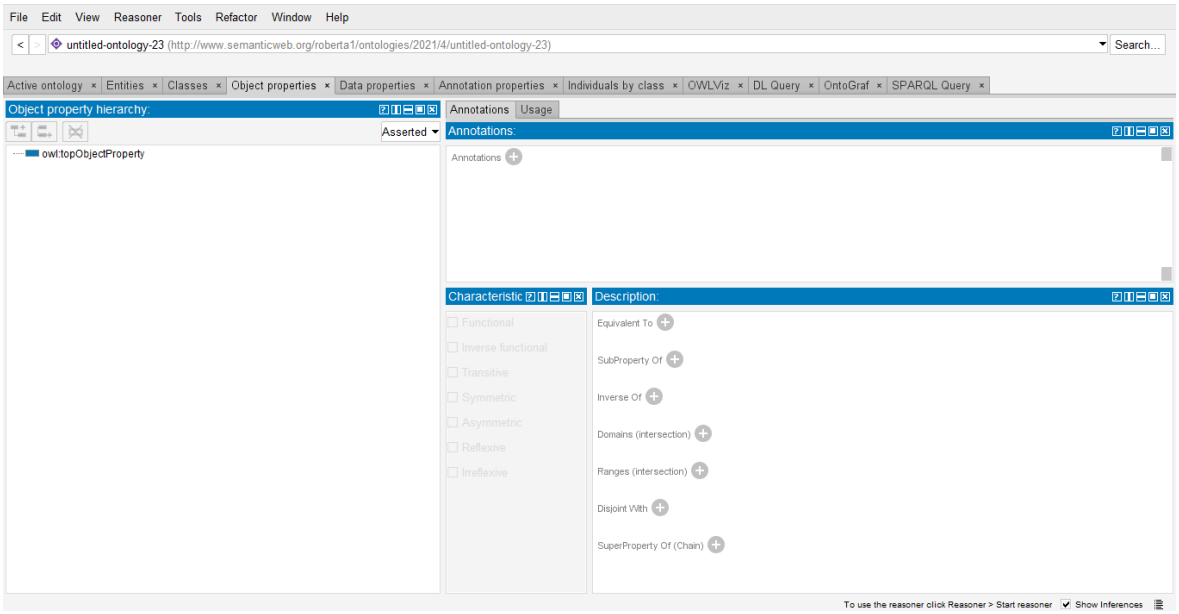

ADA
Pascal
BASIC
ALGOL
C
Delphi
Java
Python
Eiffel
GML
Perl
AWK
SQL
C++



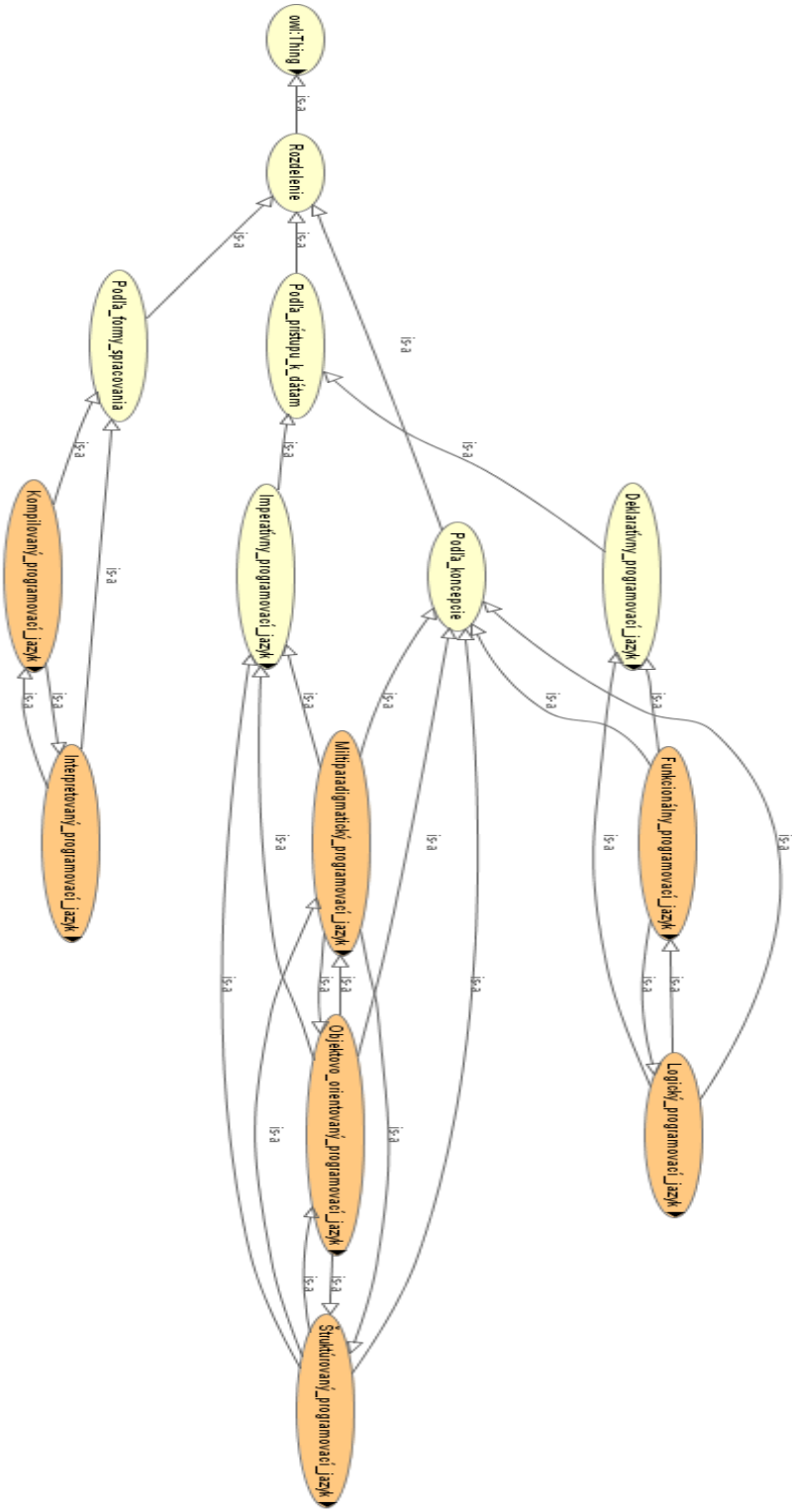
Obrázok 13 Rozdelenie podľa prístupu k dátam



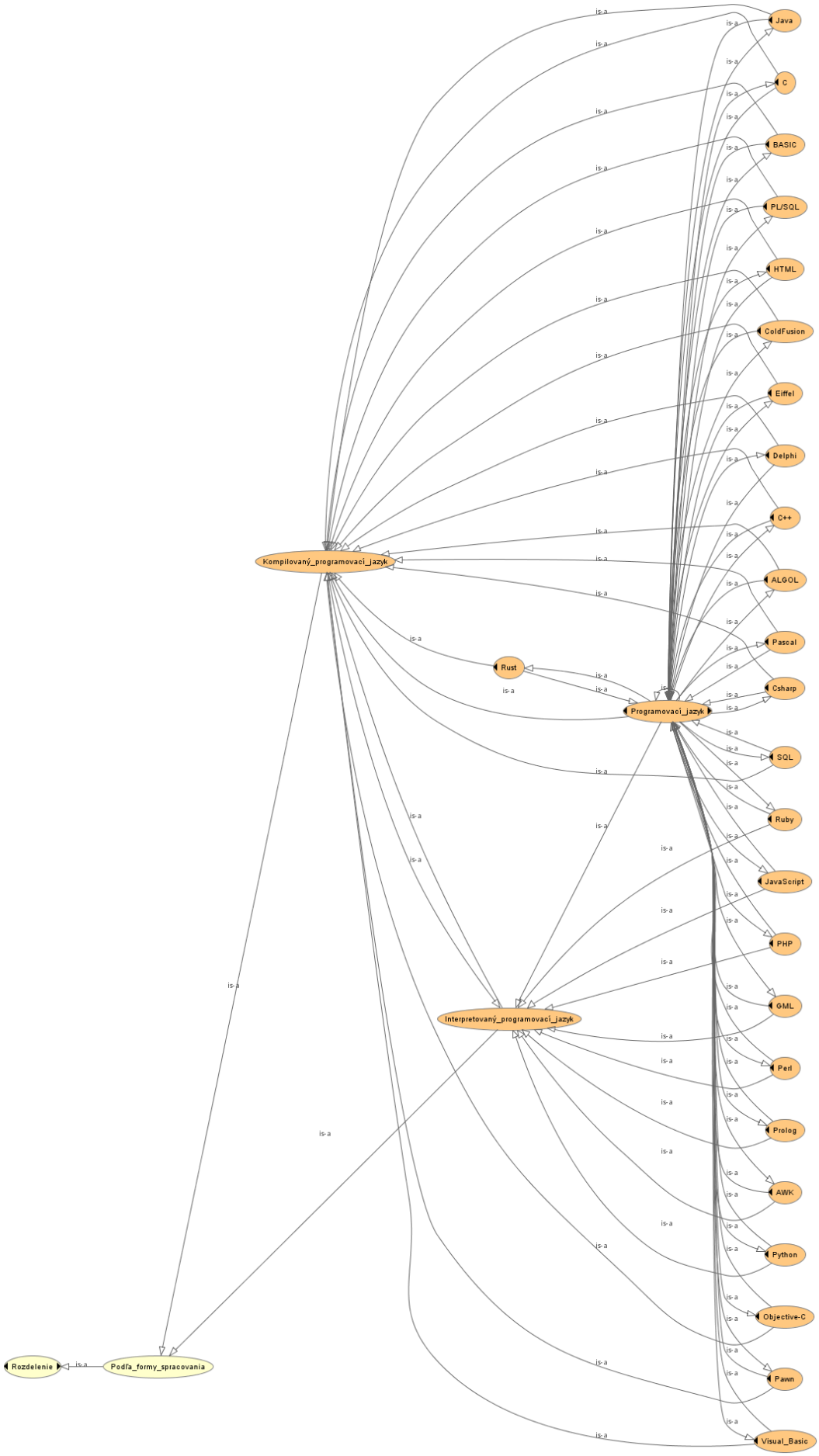
Obrázok 14 OWLentities



Obrázok 15 OWLproperties



Obrázok 16 Rozdelenie



Obrázok 17 Kompilované programovacie jazyky