MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Sémantický popis UML modelů pomocí OWL ontologií

DIPLOMOVÁ PRÁCE

Alžběta Černá

Brno, jaro 2013

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracovala samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používala nebo z nich čerpala, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: RNDr. Radek Ošlejšek, Ph.D.

Poděkování

Chtěla bych poděkovat RNDr. Radkovi Ošlejškovi, Ph.D. za hodnotné rady a připomínky a za čas, který věnoval konzultacím mé diplomové práce.

Dále bych ráda poděkovala své kamarádce Bc. Zuzaně "Lennonce"Ansorgové, za společné diskuze týkající se tématu mé práce, zkušené rady ohledně používaných technologií a doporučení vhodné odborné literatury.

V neposlední řadě mé poděkování patří trpělivosti mých přátelů a známých, kteří byli ochotní si práci přečíst a provést případné jazykové korektury.

Shrnutí

Cílem práce je vytvořit ontologie v jazyce OWL pro vybrané UML diagramy. Diplomová práce se zabývá popisem podstatných pojmů pro tvorbu OWL ontologií a seznamuje čtenáře s vývojovým prostředím Protégé, ve kterém byly ontologie pro vybrané UML diagramy vytvořeny. Po teoreticky orientovaných kapitolách představí diplomová práce čtenáři jazyk UML, jeho modelovací techniky a podrobněji popisuje vybrané diagramy a pro ně vytvořené ontologie.

Klíčová slova

RDF, OWL, Protégé, UML, UML diagram, Use Case Diagram, Class Diagram, State Diagram

Obsah

1 Úvod

Pochopení sémantiky obrázků nemusí být pouze lidskou doménou. Informatika nabízí širokou škálu nástrojů pro přiblížení lidmi vnímaných vztahů mezi reálnými i neuchopitelnými prvky strojům. Aby šlo prvky rozlišit, musí být vytvořen koncept, který vazby mezi nimi co nejpřesněji zachytí. Cílem mé práce bylo pro vybrané modely UML (Unified Modeling Language) vytvořit ontologie tak, aby bylo možné prostřednictvím těchto ontologií svázat existující diagramy UML s popsanou sémantikou a přiblížit neznalému či nevidomému uživateli každý prvek (element a vazbu), který se v diagramu vyskytuje.

Pro uvedení čtenáře do problematiky a zasazení tématu mé práce do širšího kontextu věnuji několik kapitol specifickým oblastem informatiky, které se zabývají sémantikou, a také projektu GATE, vyvíjenému na Fakultě informatiky Masarykovy univerzity, který nejen prostřednictvím těchto oblastí posouvá hranice použitelnosti grafiky, a jehož motivací je napomoci nevidomým uživatelům "vidět"obsah obrázků a případně i s grafikou s tímto osobním handicapem pracovat a vytvářet vlastní obrázky.

Kapitolu 2 věnuji popisu datového modelu RDF (Resource Description Framework) pro popis metadat a sémantiky informací ve strojově zpracovatelné podobě.

Jazyk pro tvorbu ontologií OWL (Web Ontology Language) je rozšířením RDF a jde o více expresivní jazyk se složitějšími modelovacími primitivy. Tomuto jazyku společně s popisem editoru Protégé, ve kterém lze OWL ontologie vytvářet, věnuji 3. kapitolu práce.

V kapitole 4 čtenáře blíže seznámím s projektem GATE vyvíjeným na Fakultě informatiky Masarykovy univerzity a směrem jeho vývoje spolu se základními moduly tohoto projektu.

V kapitole 5, než přikročím k popisu jednotlivých diagramů UML a pro ně vytvořeným ontologiím, uvedu čtenáře do základů jazyka UML.

Kapitolu 6 pak plně věnuji popisu jednotlivých UML diagramů a pro ně vytvořeným ontologiím. Každému z diagramů věnuji dvě podkapitoly. První z této dvojice vždy věnuji obecnému popisu diagramu a jeho prvků a druhou věnuji ontologii vytvořené pro tento diagram. Specifikace jazyka UML přesně popisuje prvky, které se v diagramech objevují. Vzhledem k návaznosti jednotlivých diagramů v rámci modelu a možnému opakování prvků v různých druzích diagramů jsem se rozhodla vytvářet pro vybrané diagramy jednu společnou ontologii. Sémantiku diagramů jsem do ontologie přidávala ve stejném sledu v jakém popisuji jednotlivé diagramy a pro ně

vytvořené ontologie. Prvním popisovaným diagramem, a také základním stavebním diagramem při objektově orientované analýze a návrhu systémů, je diagram případu užití. Následuje popis diagramu tříd, stavového diagramu a jejich ontologií.

2 RDF

RDF (Resource Description Framework) je datový model vyvinutý konsorciem W3C (World Wide Web Consortium) pro reprezentaci metadat (dat o datech) ve strojově přístupné podobě. [11] RDF byl vyvinut jako průnik světa znalostního managementu se světem knihoven metadat. Tento grafový systém vrstvený nad jazykem XML (Extensible Markup Language) se skládá ze dvou základních konceptů, a to orientovaných grafů a jazyka XML. [9] Hlavním účelem RDF je reprezentace informací o zdrojích na webu a je tedy základním stavebním kamenem sémantického webu. Lze ho využít nejen k vyjádření dat přímo získatelných z webu, ale i věcí, které lze na webu pouze identifikovat. Záměrem použití RDF jsou situace, ve kterých je třeba informace zobrazit lidem a současně tyto informace zpracovat aplikacemi. K naplnění tohoto záměru RDF sdílí strukturu k vyjádření daných informací a jejich výměnu mezi aplikacemi, aniž by došlo ke ztrátě jejich významu. [15]

2.1 Základní koncepty RDF

Pomocí RDF formátu lze vytvářet tvrzení (statements) k reprezentaci a přenosu informací. Tento formát rozhoduje jak budou informace interpretovány a přináší cestu k vyjádření vztahů mezi objekty. [9]

2.1.1 RDF zdroje a URI

Základní myšlenkou RDF je identifikace věcí (zdrojů) prostřednictvím webových identifikátorů, nazývaných jednotné identifikátory zdroje (Uniform Resource Identifiers, URI). Tyto identifikátory popisují zdroje pomocí jednoduchých vlastností a jejich hodnot. [15]

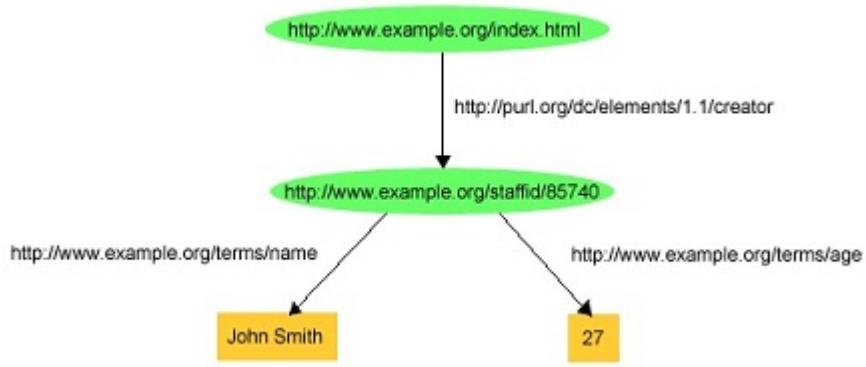
Primárně používaným identifikátorem zdrojů na webu je URL, který je podmožinou URI, ale ne vše v reálném světě lze zachytit jeho prostřednictvím. URI lze vytvořit nejen pro reprezentaci síťově dostupných zdrojů (elektronických dokumentů, obrázků, služeb, skupin zdrojů), ale i pro reprezentaci nesíťově přístupných věcí (lidé, společnosti, knihy v knihovně) a abstraktních konceptů (tvůrce). [15]

2.1.2 RDF tvrzení a trojice

Tvrzení popisují vlastnosti zdrojů. RDF tvrzení sestává z trojice identifikující subjekt, predikát a objekt prostřednictvím mechanismu URI. Subjektem je věc v tvrzení, které se tvrzení týká (například webová stránka). Predikát

značí část tvrzení identifikující vlastnosti či charakteristiky subjektu, který tvrzení popisuje (například tvůrce, datum tvorby, jazyk). Objekt je hodnota vlastnosti stanovené predikátem. Objekty v RDF mohou být znázorněny buď jako URI reference (URIref), nebo jako konstantní hodnoty (literály), které jsou reprezentované řetězcem Unicode znaků. URI reference se skládá z URI spolu s volitelným dílčím identifikátorem napojeným za znak křížku za řetězec znaků popisující URI. Literály nelze v RDF tvrzeních použít pro subjekty nebo predikáty. [15] Příkladem RDF tvrzení může být trojice (Eva, mele, maso), kde Eva je subjekt, mele je predikát a maso objekt.

Obrázek 2.1 ukazuje využití URI referencí v RDF tvrzeních.



Obrázek 2.1: Ukázka RDF tvrzení [15]

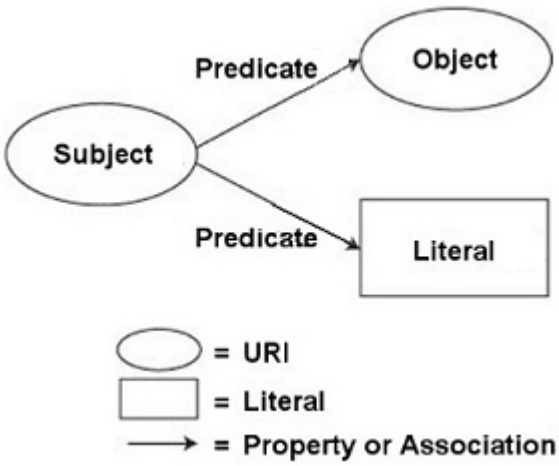
2.2 RDF model

RDF data je možné zobrazit ve třech reprezentacích: jako graf, trojici nebo v XML reprezentaci. Pro uživatele je nejjednodušší a nejpochopitelnější grafové zobrazení. Trojice je nejlépe přístupná aplikačnímu softwaru, který trojice použije jako vstup ke svým operacím. XML verze je vhodná pro přenos RDF dat mezi počítači. Z pohledu logiky jsou tyto tři formy zobrazení ekvivalentní. [9]

Grafový pohled na RDF data je nejjednodušší možnou formou jejich modelování a je často využíván pro jednoduché a pochopitelné zobrazení RDF modelu. Trojici tvořící tvrzení lze chápat jako část orientovaného grafu s označenými uzly, kde hrana reprezentuje pojmenovanou relaci mezi dvěma zdroji. Zdroje jsou v grafu vyznačeny jako uzly. [7] RDF graf sestává z množiny RDF trojic. [8] Orientace hrany je směrována od subjektového uzlu

k objektovému uzlu. RDF přímo znázorňuje pouze binární vztahy. [15] Díky zobrazení pomocí orientovaného grafu lze tvrzení na sebe navazovat a subjekt jednoho tvrzení může být objektem jiného tvrzení. [11]

Následující obrázek 2.2 znázorňuje možné trojice v datovém modelu RDF.



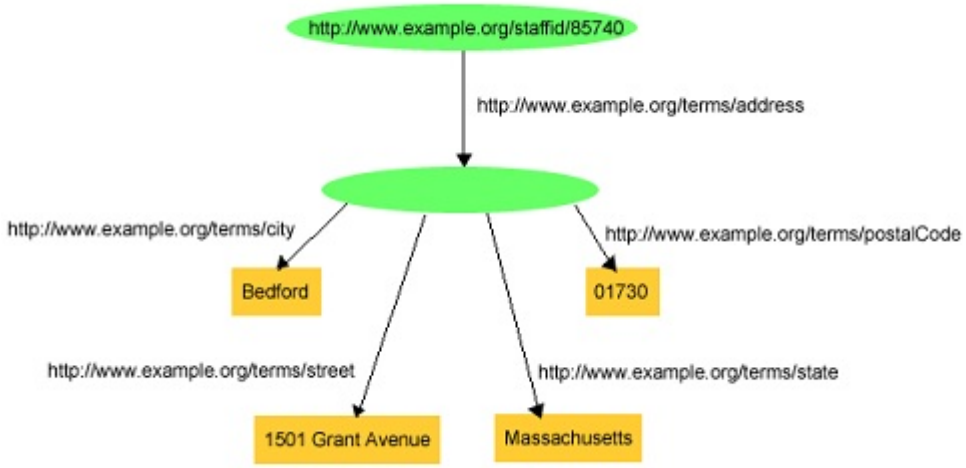
Obrázek 2.2: Trojice v RDF [18]

2.2.1 Prázdné uzly

Některé věci reálného světa jsou strukturově více komplikované (například adresa). Už se nejedná o pouhý literál (například u adresy bychom chtěli separátně zachytit ulici, město a další prvky adresy). Tuto složenou strukturu bereme jako zdroj a tvoříme o tomto zdroji soustavu tvrzení. Pro tvorbu těchto tvrzení je nutné vytvořit prostřednickou URI referenci, která bude reprezentovat společný koncept. Takovéto koncepty nebudou nikdy odkazovány přímo z oblastí mimo RDF grafový model, ve kterém se vyskytují, a z toho důvodu nepotřebují univerzální identifikátory. Proto je vhodné v grafu nakreslit na místě společného konceptu prázdný uzel (bez URI reference), který bude poskytovat spojení mezi částmi grafu. Prostřednictvím prázdného uzlu rozbijeme n-cestnost vztahu do skupiny separátních binárních vztahů. Graf může obsahovat více prázdných uzlů a je třeba mezi nimi nějakým způsobem rozlišovat. Každému prázdnému uzlu v grafu je dán odlišný identifikátor a účelem těchto identifikátorů je určitý způsob

reprezentace prázdných uzlů grafu a způsob vzájemného odlišení mezi prázdnými uzly v případě, že je graf zapsán ve formě trojice. Protože identifikátory prázdných uzlů hrají roli pouze v rámci jednoho RDF grafu, není podstatné, zda dva různé grafy používají stejný název pro některé prázdné uzly. Prázdnými uzly mohou být zobrazeny pouze subjekty či objekty, nikoli predikáty v trojici. [15]

Obrázek 2.3 ukazuje užití prázdného uzlu v tvrzení. Uzel bez URI reference zastupuje koncept adresy.



Obrázek 2.3: Užití prázdného uzlu v RDF tvrzení [15]

2.3 RDF schéma a RDF slovníky

Datový model RDF neposkytuje mechanismus pro deklaraci vlastností, o kterých tvoří tvrzení, ani neposkytuje mechanismus pro definici vztahů mezi vlastnostmi a dalšími zdroji. To je úkolem RDF schématu. [9]

Protože vlastnosti elementů mohou změnit význam RDF grafu, je třeba deklarovat, jaké elementy budou v grafu použity, a jaké vlastnosti tyto elementy ponesou. Tyto vlastnosti změní způsob tvorby tvrzení a vztah, v jakém jsou termíny vzhledem k danému grafu i v jakém vztahu jsou s elementy v jiných grafech. Deklarací elementů, jejich vlastností a struktury, ve které se mohou společně vyskytovat, deklarujeme schéma grafu. Vlatnosti i RDF elementy mohou být objekty nebo i subjekty ve vlastních tvrzeních. [9]

RDF schéma umožňuje definovat termíny a vztahy mezi těmito termíny pomocí slovníku a přidává rozšiřující význam ke konkrétním RDF predikátům a zdrojům. Tento rozšiřující význam, neboli sémantika, definuje jak budou termíny interpretovány. [11] RDF slovník je tedy množina URI referencí ve jmenném prostoru RDF (rdf: namespace). [8]

RDF poskytuje formát pro popis informací o objektech, ale neříká nic o tom, které termíny je vhodné použít k tvrzením pro popis těchto objektů a co tyto termíny znamenají. Přes výše popsaná omezení nabízí RDF formát pro formalizovaný popis termínů. Kvůli strojovému zpracování je třeba pečlivě a jednoznačně definovat všechny termíny. V RDF jsou tyto deklarace termínů, které pak používáme v tvrzeních o objektech, uváděny v odděleném dokumentu nazývaném RDF schéma a to je napsáno ve specifickém jazyce (jazyk RDF schématu). Toto schéma musí být přístupné prostřednictvím elementových URI. [9] Kromě použitých termínů definuje schéma i popisný slovník a zajišťuje zaznamenání jmen vlastností a jejich hodnot obdobně, jako zajišťuje jejich omezení, které lze využít ve výpočtu. [9]

Použití URI referencí pro subjekty, predikáty a objekty v RDF tvrzeních podporuje vývoj a znovupoužití slovníků přístupných na webu. Dostupnost slovníků umožňuje sdílení konceptů se shodným významem. Aby nedocházelo k tolika nejednoznačnostem je třeba hojněji využívat odkazů (URI referencí) namísto literálů. I přesto nebudou všechny identifikační problémy vyřešeny z důvodu občasného používání více různých URI referencí jako odkazů na jeden identický zdroj. Proto je vhodné využívat raději existující slovníky než vytvářet nové termíny, které budou překrývat již existující. [15]

Specifikace RDF schématu popisuje, jak definujeme termíny v tvrzení, zatímco RDF syntax a specifikace modelu popisuje jaký mají nové termíny v tvrzení mezi sebou vztah. Každý autor má svobodu definovat svou vlastní sémantiku a slovník, pokud přidá schéma deklarující jeho sémantiku k URI, které používá k deklaraci jmen elementů. Protože je možné mít více jak jeden jmenný prostor pro dokument, je možné mít více jak jeden slovník popisující objekt, který nese metadata. Autor si může vybrat z existujících slovníků nebo si vytvořit vlastní slovník. [9]

2.4 RDF syntax

RDF graf lze formulovat jako výše popsanou trojici hodnot, kde na první pozici trojice zapisujeme zdroj, na druhé jméno vlastnosti a na poslední, třetí pozici, hodnotu vlastnosti. RDF popisuje, jakým způsobem převést tyto trojice do XML jazyka a současně zachovat integritu grafu. [9]

Pro reprezentaci RDF tvrzení poskytuje RDF na XML (Extensible Markup Language) jazyce založenou syntax, nazývanou RDF/XML. XML bylo navrženo pro umožnění návrhu vlastních dokumentových formátů a tvorbu dokumentů v těchto formátech. RDF/XML je specifický značkovací jazyk XML pro reprezentaci RDF informací a výměnu těchto informací mezi stroji. [15]

Oproti trojicím, jejichž cílem je zkrácení široké notace je RDF/XML normativní syntaxí pro záznam RDF. Tato syntax sestává z URI, vlastností a jejich hodnot. Jako HTML (HyperText Markup Language) je RDF/XML strojově zpracovatelná a díky URI může spojovat informace skrz web. [15]

3 OWL ontologie a Protégé

Nejnovějším trendem ve vývoji standardů jazyků pro tvorbu ontologií je OWL vytvořený World Wide Web konzorciem (W3C)[1]. [10] Vytváření těchto OWL ontologií usnadňuje open source editor Protégé, který je založený na konceptech jazyka OWL. Tato platforma podporuje dva způsoby modelování ontologií, a to prostřednictvím Protégé-Frames a Protégé-OWL editorů. Je postavena na jazyce Java, je rozšiřitelná a poskytuje plug-and-play prostředí, které zajišťuje flexibilní základ pro rychlé prototypování a vývoj aplikací. Ontologie vytvořené v Protégé je možné exportovat do formátů RDF, OWL a XML schémat. [4]

3.1 Rozšíření OWL oproti RDF

RDF nabízí pouze jednoduchá modelovací primitiva pro tvorbu ontologií a není příliš expresivní. RDF schéma umožňuje specifikaci, sdílení díky otevřenému, na webu postavenému standardu, a konvenci danou RDF sémantikou, která přesně specifikuje význam. Stále však u RDF chybí dostatečná síla pro více expresivní modelování ontologií. OWL rozšiřuje RDF o omezení kardinality (například "Stát může mít pouze jedno hlavní město."), dvouargumentové pravdivostní funkce jako konjunkce, disjunkce, negace a ekvivalence (například "Státy a města jsou vzájemně disjunktní. Něco nemůže být zároveň státem i městem.") a lokalizovaná omezení (například "Když je na město aplikována vlastnost populace, její hodnota musí být mezi 20 000 a 10 miliony."). [11]

3.2 Historický vývoj jazyka OWL

Historický počátek vývoje ontologií ve filosofii a na informatickém poli sahá až do devadesátých let minulého století, kdy se zkoumala myšlenka využití reprezentace znalostí a umělé inteligence na webu. Roku 2000 začal ve Spojených Státech vývoj jazyka DAML (DARPA Agent Markup Language) a roku 2001 došlo k prvním pokusům o sjednocení DAML s jazykem OIL (Ontology Inference Layer), který ve stejném roce jako DAML vyvíjela skupina evropských vědců. DAML+OIL je tenká vrstva nad RDFS (RDF Schématem) s formální sémantikou založenou na deskriptivní logice. Po roce 2002 si WebOnt, skupina autorizovaná konsorciem W3C, dala za úkol vyvinout

1. http://w3.org/TR/owl-guide

standard založený na DAML+OIL. Výsledným produktem byl roku 2004 standard nesoucí název OWL (Web Ontology Language). Nejnovější verzí jazyka je OWL 2 z roku 2009. Tuto verzi využívá editor Protégé a sémantické Reasonery jako Pellet, RacerPro, FaCT++ a HermiT. Podrobnějšímu popisu Reasonera se věnuji v poslední podkapitole této kapitoly. [19]
Pro tvorbu OWL ontologií jsem využívala verzi Protégé 4.2, dostupnou na webových stránkách Stanfordské univerzity[2], na jejíž půdě byla aplikace v rámci Centra pro biomedicínský informatický vývoj vytvořena. Aplikaci lze obohatit prostřednictvím široké škály existujících pluginů dostupných taktéž ze stránek Stanfordské univerzity[3]. Při tvorbě následujícího souhrného popisu Protégé v rámci této kapitoly jsem se inspirovala tutoriálem pro tvorbu OWL ontologií pomocí Protégé 4 a CO-ODE nástrojů. [10]
Základními stavebními prvky ontologií v Protégé jsou individua (Individuals), třídy (Classes) a vlastnosti (Properties).

3.3 Individua (Individuals)

Individua reprezentují objekty zájmové domény. OWL narozdíl od Protégé nepředpokládá jedinečnost jména (Unique Name Assumption). Z tohoto důvodu se dvě odlišná jména mohou vztahovat ke stejnému individuu, proto musí být explicitně stanoveno, zda jsou individua shodná nebo odlišná, jinak mohou být shodná nebo mohou být odlišná. Individua lze také použít k popisu tříd, konkrétně v hasValue restrikcích a výčtových třídách.

3.4 Třídy (Classes)

Třídy jsou hlavní stavební složky OWL ontologií. Třídy interpretujeme jako množinu obsahující individua. Specifikujeme je použitím formalních konstrukcí, které stanovují přesné požadavky na členství v dané třídě. Třídy můžeme taxonomicky organizovat do hierarchií podtříd a nadtříd, kde podtřídy specializují své nadtřídy. V OWL podtřída značí nutnou implikaci a tedy všechny instance podtřídy jsou instancemi nadtřídy bez výjimky. Tyto vztahy mohou být automaticky vypočítány prostřednictvím funkce OWL-DL Reasoner. Více se funkci Reasoner věnuji v poslední podkapitole této kapitoly. Třídy jsou sestaveny z popisů specifikujících podmínky, které musí být splněny individuem, aby bylo členem dané třídy. Každá ontologie obsahuje dvě předdefinované třídy owl:Thing a owl:Nothing. Třída Thing re-

2. http://protege.stanford.edu
3. http://protege.stanford.edu/download/plugins.html

prezentuje množinu obsahující všechna individua. Třída Nothing je prázdná třída. Všechny vytvořené třídy jsou podtřídou Thing a každá třída má jako podtřídu Nothing.

3.4.1 Disjunktní třídy

Implicitně se v OWL ontologiích předpokládá překrývání tříd. Z tohoto důvodu nelze vyslovit tvrzení, že individuum není členem konkrétní třídy pouze proto, že není této třídě přiděleno. Pokud chceme, aby dané individuum (objekt) nemohlo být instancí více než jedné třídy ze skupiny, musíme prvky skupiny tříd na stejné úrovni označit jako vzájemně disjunktní. Teprve poté bude platit omezení členství pouze na jednu třídu ze skupiny.

3.4.2 Výčtové třídy (Enumerated Classes)

Třídu v OWL lze popsat pomocí výčtu individuí, která jsou jejími členy. V Protégé 4 tuto třídu vytvoříme vepsáním výčtu v ontologii již existujících individuí v množinových závorkách do pole ekvivalentních tříd.

3.5 Vlastnosti (Properties)

Vlastnosti jsou binární relace na individuích, které spojují dvě individua (přesněji řečeno instance vlastností spojují dvě individua). V deskripční logice jsou vlastnosti známy pod pojmem role a v rámci UML a dalších objektově orientovaných notacích jsou označovány pojmem relace. Vlastnosti reprezentují vztahy. Dva hlavní typy vlastností v Protégé jsou vlastnosti objektů (Object properties) a vlastnosti s datovým typem (Datatype properties). OWL má také třetí typ vlastností, který se nazývá anotační vlastnosti (Annotation properties) a oba výše zmíněné hlavní typy vlastností mohou být označeny jako anotační vlastnosti.

Obdobně jako u tříd je i u vlastností možné tvořit jejich hierarchie. Vlastnosti na nižší úrovni hierarchie specializují vlastnosti, které se nachází na vyšší úrovni nad nimi, a to způsobem obdobným specializaci madtřídy její podtřídou. V rámci vlastnosti na nižší úrovni pod jinou vlastností ale není možné kombinovat objektové vlastnosti a vlastnosti s datovým typem. Není tedy možné vytvořit objektovou vlastnost, která je v hierarchii pod vlastností s datovým typem a obráceně.

3.5.1 Vlastnosti objektů (Object Properties)

Vlastnosti objektů jsou vztahy mezi dvěma individui. Individuum A je ve vztahu s individuem B prostřednictvím vlastnosti P.

Inverzní vlastnosti

Každá vlastnost objektů může mít korespondující inverzní vlastnost. Pokud nějaká vlastnost spojuje individuum A s individuem B, potom inverzní vlastnost bude spojovat individuum B s individuem A.

Domény (Property Domains) a obory hodnot vlastností (Property Ranges)

Vlastnosti spojují individua z dané domény s individui z daného oboru hodnot. V OWL bychom neměli domény a obory hodnot vlastností chápat jako omezující podmínky, které je nutné prověřovat. V OWL jsou domény a obory hodnot použity jako axiomy odvozování. Pokud je u vlastnosti P stanovená doména A a použijeme tuto vlastnost P s doménou B (individua, která jsou členy třídy B), vyvodí se, že B je podtřídou A. K chybě dojde pouze pokud jsou třídy A a B stanoveny jako vzájemně disjunktní. Nastavíme-li jako obor hodnot více tříd, v Protégé 4 se bude takováto specifikace interpretovat jako nastavení oboru hodnot na společný průnik těchto tříd. Obecně tedy není vhodné u všech vlastností zadávat domény a obory hodnot, jelikož stanovení domény a oboru hodnot se nechová jako omezení a může výsledně způsobovat neočekávané klasifikace, které jsou následně obtížně dohledatelné v rozsáhlejších ontologiich.

3.5.2 Vlastnosti s datovým typem (Datatype properties)

Vlastnosti s datovým typem popisují vztahy mezi individuem a datovou hodnotou. Datová hodnota může být stanovena obecně daným typem vymezeným slovníkem XML schématu (například integer, float, string, boolean) nebo uvedena konkrétně. Předdefinovanou množinu datových typů lze blíže specifikovat restrikcemi, které se vepíší do hranatých závorek za typ.

3.5.3 Anotační vlastnosti (Annotation properties)

Anotační vlastnosti se používají k přidání informace (metadat) k třídám, individuím a objektovým vlastnostem nebo vlastnostem s datovým typem. OWL obsahuje pět předdefinovaných anotačních vlastností (owl:versionInfo,

rdfs:label, rdfs:comment, rdfs:seeAlso a rdfs:isDefinedBy). Pro příklad anotační vlastnost rdfs:comment se používá k ukládání komentáře o třídách v Protégé 4 a anotační vlastnost rdfs:label může sloužit k poskytnutí alternativních jmen pro třídy nebo vlastnosti.

3.6 Charakteristiky vlastností objektů

OWL umožňuje obohatit význam vlastností pomocí jejich charakteristik. Vlastnosti objektů mohou mít výše zmíněné inverze, mohou být limitovány pouze na jednu hodnotu neboli být funkcionální. Vlastnosti také mohou být tranzitivní nebo symetrické. U vlastností s datovým typem jsou charakteristiky omezené pouze na charakteristiku pro umožnění více hodnot a inverzně funkcionální charakteristiku. OWL-DL nedovoluje vlastnostem s datovým typem, aby byly tranzitivní, symetrické nebo měli inverzi. Jediná povolená charakteristika u vlastností s datovým typem je funkcionální charakteristika, pomocí které u takovéto vlastnosti stanovíme, že dané individuum se může prostřednictvím této vlastnosti vztahovat nejvýše k jednomu individuu.

Následuje podrobnější výčet a popis charakteristik, které mohou vlastnosti nabývat.

3.6.1 Funkcionální (jednohodnotové) vlastnosti

Pokud je vlastnost funkcionální, pro dané individuum A může existovat nejvýše jedno individuum B, které má vztah prostřednictvím dané vlastnosti k individuu A. V případě, že pro dané individuum A existuje více individuí vztažených prostřednictvím stejné funkcionální vlastnosti, lze z toho odvodit, že všechna takto vztažená individua jsou totožná. Stanovíme-li, že se jedná o rozdílná individua, povede výše uvedené tvrzení k inkonzistenci.

3.6.2 Inverzně funkcionální vlastnosti

V případě, že je vlastnost označena jako inverzně funkcionální, potom musí být inverze k dané vlastnosti funkcionální. Pro dané individuum A tedy může existovat nejvýše jedno individuum B, které je prostřednictvím dané vlastnosti ve vztahu s individuem A.

3.6.3 Tranzitivní vlastnosti

Pokud je vlastnost P tranzitivní a vztahuje individuum A k individuu B a také individuum B k individuu C, můžeme odvodit, že individuum A je

prostřednictvím vlastnosti P vztaženo k individuu C. Pokud je vlastnost tranzitivní, pak i jeho inverze je tranzitivní. O zajištění této podmínky se automaticky postará Reasoner, nebo lze tuto podmínku nastavit manuálně v Protégé editoru. V rámci tranzitivity vlastnosti vzniká řetězení, které vlastnosti znemožňuje, aby byla funkcionální.

3.6.4 Symetrické vlastnosti

Pokud je vlastnost symetrická a dává do vztahu individuum A s individuem B, znamená to, že prostřednictvím stejné vlastnosti je individuum B vztaženo k individuu A.

3.6.5 Asymetrické vlastnosti

Označíme-li vlastnost jako asymetrickou a vztahovala-li individuum A k individuu B, nemůže po asymetrickém označení vztahovat individuum B k individuu A.

3.6.6 Reflexivní vlastnosti

Vlastnost je reflexivní, pokud lze jejím prostřednictvím vztahovat dané individuum samo k sobě.

3.6.7 Ireflexivní vlastnosti

Vlastnost je ireflexivní, pokud může vztahovat individuum A k individuu B jen pokud individuum A a individuum B nejsou totožné.

3.7 Popis a definování tříd

K popisu a definování tříd slouží restrikce (omezení) vlastností. Restrikce popisuje třídu individuí na základě vztahu ve kterém členové této třídy participují. Jinými slovy restrikce popisuje anonymní (nepojmenovanou) třídu, která obsahuje všechna individua splňující danou restrikci. Anonymní třída tedy obsahuje všechna ta individua, která splňují vztahy požadované pro členství v dané třídě. V OWL se restrikce používají k popisu tříd a specifikaci anonymních nadtříd popisované třídy.

Popis třídy (Class Description) v Protégé obsahuje následující pole:

Ekvivalentní třídy (Equivalent classes) - toto pole obsahuje seznam ekvivalentních tříd. Třídy uvedené v tomto poli jsou někdy označovány jako nutná a postačující podmínka.

Nadtřídy (Superclasses) - v tomto poli tvoříme omezení na třídu. Pole nadtříd obsahuje seznam podtříd, které nazýváme nutnou podmínkou.

Zděděné anonymní třídy (Inherited anonymous classes) - toto pole obsahuje seznam podmínek, které jsou zděděné z nadtříd.

Restrikce lze rozdělit do tří hlavních kategorií:

- Kvantifikační restrikce (Quantifier Restrictions)
- Restrikce kardinality (Cardinality Restrictions)
- hasValue restrikce

3.7.1 Kvantifikační restrikce (Quantifier Restrictions)

Kvantifikační restrikce mohou být dále rozděleny na existenciální a univerzální restrikce.

Existencionální restrikce (Some Restrictions)

Existencionální restrikce popisují množinu takových individuí, která participují v alespoň jednom vztahu prostřednictvím specifické vlastnosti P k individuím, která jsou členy specifické třídy A. Tuto specifickou třídu A, v Protégé označovanou jako plnící třída (filler), uvozujeme klíčovým slovem "some". Existencionální restrikce mohou být vyznačeny existencionálním kvantifikátorem ($\exists$) a v jazyce OWL jsou také známy pod pojmem someValuesFrom restrikce. Všechna individua v anonymní třídě, kterou restrikce definuje, mají alespoň jeden vztah prostřednictvím vlastnosti P k individuu, které je členem třídy A. Oproti univerzálním restrikcím existencionální restrikce nepřikazují, že vztah daný vlastností P může pro tuto třídu existovat pouze v návaznosti na individua, která jsou členy specifické plnící třídy A. Existencionální restrikce neomezuje vlastnost pouze na členy určité třídy. Pouze stanovuje, že každé individuum musí mít alespoň jeden vztah prostřednictvím dané vlastnosti s individuem, které je členem dané třídy. Individua mohou mít i jiné vztahy prostřednictvím vlastnosti P s jinými individui, které nejsou členy plnící třídy A.

Univerzální restrikce (AllValuesFrom Restrictions)

Univerzální restrikce popisují množinu takových individuí, která mají prostřednictvím vlastnosti P vztahy pouze skrz tuto vlastnost k individuím, která jsou členy specifikované plnící třídy. Individua díky této restrikci nemohou mít daný vztah P k individuím, která nejsou členy plnící třídy. Univerzální restrikce zároveň popisuje individua, která neparticipují v žádném vztahu daném vlastností P. Univerzální restrikce nezaručují existenci vztahu pro danou vlastnost P. Tento typ restrikce pouze stanovuje, že pokud pro danou vlastnost P vztah existuje, pak musí být k individuím, která jsou členy dané plnící třídy. Tyto restrikce uvozujeme klíčovým slovem only. Univerzální restrikce mohou být vyznačeny univerzálním kvantifikátorem (∀) a v jazyku OWL jsou známy také jako allValuesFrom restrikce.

3.7.2 Nutné a postačující podmínky

Nutná podmínka třídy hovoří o podmínkách, které individuum splňuje, pokud je členem dané třídy. S pouze nutnými podmínkami nelze o individuu vyslovit tvrzení, že díky plnění dané podmínky je členem třídy, protože podmínky nejsou postačující. Abychom určili, zda individuum je členem určité třídy, jak vyplývá z předchozího tvrzení, nestačí nám k tomu vědět plnění nutných podmínek. Pro členství v dané třídě musíme o třídě znát nejen nutné, ale i postačující podmínky.

Primitivní třída

Třída obsahující pouze nutné podmínky je označována jako primitivní třída. V Protégé 4 se nutné podmínky nazývají nadtřídy (Superclasses).

Definovaná třída

Třída obsahující alespoň jednu množinu nutných a postačujících podmínek se označuje termínem definovaná třída. V Protégé 4 se nutné a postačující podmínky nazývají ekvivalentní třídy (Equivalent classes).

3.7.3 Uzávěrový axiom

Uzávěrový axiom vytvořený u specifické vlastnosti se skládá z univerzální restrikce, která stanoví, že vlastnost může být aplikována pouze na specifické plnící třídy. Uzávěrový axiom tedy omezí danou vlastnost v třídě pouze na specifikované plnící třídy. Výsledná plnící třída je sjednocením

jednotlivých plnících tříd, které jsou specifikovány v existencionální restrikci dané vlastnosti. Máme-li tedy vzájemně disjunktní třídy A a B a property P, pak restrikce v nadtřídách dané třídy, zapsaná (P only (A or B)) značí, že třída, ke které se daná restrikce vztahuje, musí mít vztah P pouze s členy tříd A nebo B.

Nechceme-li do popisované třídy zahrnout individua, která ve vztazích prostřednictvím dané vlastnosti vůbec neparticipují, je nutné uvést v nadtřídách třídy nejen univerzální restrikci, ale i jednotlivé existencioníální restrikce. Uvedením pouze univerzální restrikce v popisu třídy zahrneme do popisované třídy i vše, co nemá žádný vztah daný specifikovanou vlastností.

3.7.4 Rozdělení hodnot (Value Partitions)

Rozdělení hodnot není součástí OWL. Jedná se o návrhový vzor. Návrhové vzory v návrhu ontologií jsou analogií návrhových vzorů v objektově orientovaném programování. Návrhové vzory nabízí modelové řešení často se opakujících problémů. Rozdělení hodnot v ontologiích slouží ke zdokonalení popisů tříd a omezuje možné hodnoty na úplný seznam. Postup vytvoření rozdělení hodnot v OWL začíná tvorbou třídy, která bude rozdělení hodnot reprezentovat. Následně vytvoříme podtřídy, které tuto třídu pokrývají a budou tedy reprezentovat všechny možné hodnoty pro dané rozdělení. Tyto podtřídy dávají jako své sjednocení nadtřídu, kterou pokrýváme a jsou vzájemně disjunktní. U třídy, kterou pokrýváme, nastavíme jako ekvivalentní třídy všechny pokrývající podtřídy. Tímto krokem stanovíme pokrývající axiom úplným seznamem pokrývajících tříd. V Protégé 4 se pokrývající axiom projevuje jako třída, která je sjednocením tříd, které pokrývá. V poslední fázi vytvoříme funkcionální objektovou vlastnost pro dané rozdělení hodnot a u této vlastnosti nastavíme obor hodnot na třídu, která rozdělení hodnot reprezentuje.

3.7.5 Restrikce kardinality (Cardinality Restrictions)

Prostřednictvím OWL můžeme popsat třídu individuí, která mají nejméně, nejvíce nebo přesně specifikované množství vztahů s jinými individui nebo hodnotami s daným datovým typem. Restrikce, které popisují tyto třídy označujeme jako restrikce kardinalit.

Minimální restrikce kardinality pro danou vlastnost P je uvozena klíčovým slovem "min"a specifikuje minimální množství P vztahů, ve kterém musí individuum participovat.

Maximální restrikce kardinality, která je uvozena klíčovým slovem "max", specifikuje maximální množství P vztahů ve kterém může individuum participovat.

Klíčovým slovem "exactly"je uvozena restrikce kardinality, která specifikuje přesné množství P vztahů, ve kterých musí individuum participovat.

V případě, že stanovíme třídu objektů v restrikci hovoříme o přesně stanovené kardinalitní restrikci (Qualified Cardinality Restriction). Pokud třída, která je plnící třídou pro restrikci kardinality není stanovena, je jako plnící třída brána třída Thing.

3.7.6 hasValue restrikce

Restrikce hasValue, označená symbolem $\ni$, popisuje anonymní třídu individuí, která mají nejméně jeden vztah prostřednictvím určité vlastnosti k specifickému individuu. V kvantifikační restrikci oproti hasValue restrikci je dána třída, ze které má být vybráno jakékoli individuum, ale samotné individuum není specifikováno. Obdobně jako u existenčních restrikcí hasValue restrikce pro určité individuum neomezují vlastnost v restrikci na vztah pouze se specifikovaným individuem.

3.8 Subjazyky OWL

Jazyk OWL lze rozdělit do tří jazykových vrstev. Každá z vrstev rozšiřuje množinu schopností například jednoduchým, intuitivním modelováním nebo efektivním odvozováním, případně vysokou expresivností. [11]

- OWL Full
- OWL DL
- OWL Lite

V OWL Full je OWL třída definovaná jako ekvivalentní RDFS třídě. Jakákoli třída, která je podtřídou RDFS třídy je také podtřídou OWL třídy. Jakýkoli validní RDF dokument lze tedy vzhledem k důsledkům předchozích tvrzení považovat za validní OWL Full dokument. Narozdíl od OWL Full je v Lite a DL subjazycích OWL třída definovaná jako podtřída RDFS třídy. Tedy ne všechny třídy RDF (podtřídy RDFS třídy) mohou být instance nebo podtřídy OWL třídy. Důsledkem tohoto tvrzení je, že validní RDF dokument nemůže být považován za validní Lite nebo DL dokument. [1]

Druhá zásadní odlišnost mezi subjazyky OWL je vztah mezi OWL objektovou vlastností a RDF vlastností. V OWL Full je objektová vlastnost

považována za ekvivalentní vlastnosti v RDF. Vlastnost s datovým typem v OWL, která je podtřídou RDF vlastnosti je také podtřídou OWL objektové vlastnosti. Tedy jakákoli vlastnost v OWL definovaná s datovým typem může být interpretována jako objektová vlastnost. Díky výše zmíněným rozdílům je OWL Full velmi expresivním jazykem. Narozdíl od OWL full jsou OWL vlastnost s datovým typem a OWL objektová vlastnost definované jako disjunktní podtřídy RDF vlastnosti. Jako důsledek nemohou být vestavěné charakteristiky jako inverze, inverzní funkcionalita, symetrie a tranzitivita specifikovány jako vlastnosti s datovým typem z důvodu, že vlastnost s datovým typem definuje vztah mezi individuem a literálem (číslem, datem). Inverze od vlastnosti s datovým typem by proto nedávala velký význam. [1]

3.9 Reasoner

Protože OWL-DL ontologie je přeložitelná do reprezentace deskripční logiky, je možné provádět automatické odvozování v rámci ontologie pomocí Reasonera deskripční logiky. Reasoner deskripční logiky provádí řadu deduktivních služeb, například spočtení odvozených nadtříd třídy, rozhodnutí zda je třída konzistentní (třída je nekonzistentní, pokud nemůže mít instance), rozhodování zda je jedna třída zahrnuta v jiné a další. Někteří z často používaných Reasonerů deskripční logiky jsou RACER, FaCT, FaCT++ a Pellet. [5]

Protégé 4 umožňuje zapojení rozdílných OWL Reasonerů. Reasoner zahrnutý v Protégé 4 se nazývá Fact++. Na základě podmínek, které musí třída splňovat, Reasoner testuje vztahy nadtříd a podtříd (test zařazení) a kontroluje konzistenci. U kontroly konzistence Reasoner podle podmínek zjistí, zda je možné, aby třída měla instance. Pokud třída nemůže mít instance, je považována za inkonzistentní. Velmi podstatnou prací reasonera je výpočet a správa mnohonásobné dědičnosti nazývaná normalizace ontologií. Reasoner nikdy neumístí jinou třídu jako podtřídu primitivní třídy. Reasoner může automaticky klasifikovat pouze třídy pod definovanými třídami.

Manuálně vybudovaná hierarchie tříd v Protégé 4 se nazývá údajná hierarchie (asserted hierarchy). Hierarchie tříd, která se automaticky vypočítá Reasonerem se nazývá odvozená hierarchie (inferred hierarchy).

Pro kontrolu správného sestavení ontologie je vhodné vytvořit třídu, která bude zároveň podtřídou dvou disjunktních tříd. Takovouto třídu, uměle přidanou do ontologie pro kontrolu testu integrity, nazýváme vyšetřovací třída (Probe class).

4 Dialogový systém GATE

Systém GATE (Graphics Accessible To Everyone)[1] je projekt vyvíjený v rámci Laboratoře vyhledávání a dialogu[2] Fakulty informatiky Masarykovy univerzity v Brně ve spolupráci se střediskem pro pomoc studentům se specifickými nároky Teresiás[3] od roku 2007. Posláním střediska Teresiás je zajišťovat přístupnost studijních oborů akreditovaných na univerzitě studentům se smyslovým nebo jiným handicapem.

V této kapitole popisuji základní myšlenky systému GATE a jeho modulů. Bližší informace týkající se GATE systému a zpřístupnění grafiky nevidomým pomocí anotací obrázků v SGV formátu a OWL ontologií lze najít v článku GATE to Accessibility of Computer Graphics [13].

Prohledávání grafických prvků je v dnešní době pro vizuálně znevýhodněné uživatele komplikované. V případě, že vyhledají obrázek a chtějí mu porozumět, snaží se "přečíst"jeho alternativní popis nebo prozkoumávat obrázek pomocí sonifikace[4], která překládá grafické prvky do neverbální zvukové formy. [16]

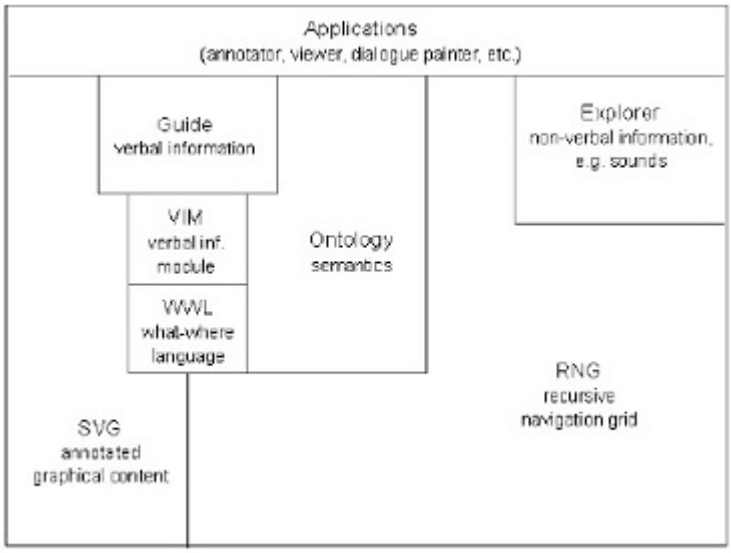
Projekt GATE se zaměřuje na zpřístupnění grafiky nevidomým a slabozrakým studentům pomocí dialogů v přirozeném jazyce. Mezi stěžejní záměry projektu GATE patří vývoj nástrojů pro snadný popis obrázků, poskytnutí podpory nevidomým pro prozkoumávání ("prohlížení") obrázků a v neposlední řadě vývoj systému využitelného pro tvorbu obrázků prostřednictvím dialogů přirozeného jazyka umožňujícího nevidomým vytvářet v omezené formě počítačovou grafiku.

4.1 Základní moduly a principy GATE systému

Obrázek 4.1 [16] znázorňuje hlavní části GATE systému, které popisuji níže v rámci této podkapitoly.

Modul anotátor (Annotator Module) podporuje navigaci obrázkem a je úzce spojen s grafickou ontologií. Základním úkolem modulu anotátor je

1. http://lsd.fi.muni.cz/gate/about
2. http://lsd.fi.muni.cz/tiki-index.php
3. http://www.teiresias.muni.cz/
4. Sonifikace je proces překladu viditelných dat (obrázků) do neverbální zvukové formy. Charakter zvuku jako hlasitost, barva, lokace, rytmus, stupeň, amplituda, trvání či tempo nahrazují viditelné charakteristiky scény jako barva pixelu a oblast, umístění objektu, vzdálenost mezi objekty, a další. Sonifikace poskytuje uživatelům přídavné informace o obrázcích, grafech, mapách a dlaších grafických objektech.



Obrázek 4.1: Struktura GATE

informovat uživatele o grafickém obsahu obrázku nevizuální cestou. K docílení výše zmíněného úkolu poskytuje systém GATE verbální způsob a způsob prostřednictvím zvuku. Demonstraci obou těchto způsobů lze nalézt na stránkách dedikovaných GATE systému[5]. Tato forma komunikace s nevidomým uživatelem je podporována prostřednictvím Co-Kde jazyka (What-Where Language, WWL) a rekurzivně navigační mřížky (Recursive Navigation Grid, RNG). [14]

Co-Kde jazyk (What-Where Language, WWL) je zjednodušený zlomek anglického jazyka. Každá věta tohoto jazyka má formu "CO je KDE", případně formu "KDE je CO". Tento jazyk umožňuje uživateli klást jednoduché otázky o objektech na scéně a jejich pozici (například "Kde je věž?", "Co je uprostřed?"). Rekurzivně navigační mřížka (Recursive Navigation Grid, RNG) je navigační páteř systému, dělící prostor obrázku do devíti identicky velkých obdélníkových sektorů. Každý sektor je rekurzivně podrozdělen obdobným způsobem. Díky tomuto rozdělení může uživatel prozkoumávat body v regionu s vybranou přesností. [14]

Modul slovních informací (Verbal Information Module, VIM) řídí dialogovou komunikaci včetně WWL komunikace. VIM řeší možná nedorozu-

5. http://lsd.fi.muni.cz/gate/demo

mění v komunikaci prostřednictvím strategie pro opravu dialogu. [14]

Modul průvodce (Guide Module) a modul průzkumník (Explorer Module) zajišťují podporu návratu informací.

Průvodce poskytuje verbální informace o obrázku, přičemž využívá informace získané označením obrázku a informace získané přímo z formátu obrázku. Významné informace jsou předány průzkumníkovi, který spolupracuje s VIM a RNG. Komunikace průzkumníka s uživatelem nemusí být primárně verbální, tato komunikace může být i analogová a je kontrolována pomocí myši, tabletu nebo numerické kávesnice. Výstupní zvuková informace také nemusí být primárně verbální. Pro navigaci obrázkem se využívá RNG modulu. Informace vztahující se k místu či objektu na obrázku jsou jak verbálního tak neverbálního charakteru a umožňují uživateli provádět rychlé dynamické prohledávání neanotovaných detailů na obrázku. Informace o barvách jsou poskytovány prostřednictvím procedury založené na zvukové reprezentaci barev. Princip zvukové reprezentace barev funguje způsobem, kde je zvuková informace vytvořena jako kombinace specializovaných zvuků přidělených základním barvám vhodného barevného modelu. [14]

4.2 Detekce a anotace grafických objektů

Aby se nevidomým zadařilo obrázek co nejlépe prozkoumat musí být obrázky anotované. Ruční anotace je pro velké grafické databáze nevhodná, protože jde o zdlouhavý a časově náročný proces. Na druhé straně automatická anotace často vytváří nepřesné a irelevantní výsledky, nejčastěji u více komplexních obrázků. Poloautomatická anotace je v současné době pravděpodobně nejpřijatelnějším řešením pro výše popsaný problém. Základním principem je použití vhodné ontologie, která podpoří anotační proces. Tento princip umožňuje systému pomoci uživateli anotovat obrázek tak jednoduše a efektivně, jak je to jen možné, a současně předejít zmatení v terminologii a udržet sémantickou hierarchii konzistentní. Během anotace uživatel vymezuje hranice konkrétních objektů na obrázku. Toho lze dosáhnout pomocí automatického rozčlenění obrázku a detekcí hran spolu s dodatečným ručním čistěním pomocí fuzzy nástrojů. Dalším krokem v anotačním procesu je definice sémantiky vyznačených objektů. Pomocí ontologií lze značně zlepšit rozpoznávání obrázku a současně i kroky definující sémantiku. Detekce a anotace grafických objektů v rastrovém obrázku je komplexní úkol, který vyžaduje specifický přístup vzhledem ke konkrétní aplikaci. Většinu

běžně používaných metod pro rozdělení obrázku a detekci objektů v něm lze alespoň do jisté míry využít v projektu GATE. Hlavním cílem budoucího vývoje je nalezení a implementace vhodné strategie pro ideální vyvážení pokrytí rozdělujícími metodami a ručním rozdělováním obrázku. [16]

4.2.1 Anotace pomocí ontologií

Ontologie spojuje informace o obrázku a informace o objektech obrázku s jejich popisem v reálném světě a usnadňuje tak práci anotátora díky automatickému poskytování atributů popisovaného objektu. Každé sémantické kategorii v ontologii je dána množina specifických vlastností, které charakterizují přidělenou kategorii. Například sémantická kategorie auta může mít vlastnosti barva a typ. Popis podporovaný ontologií vyžaduje výběr objektu na obrázku, označení jeho sémantické kategorie z ontologie, a poté přidělení hodnot předepsaným vlastnostem. Pokud je tento druh informací přítomný v ontologii, lze jej jednoduše znovupoužít a uživatel není nucen vymýšlet nejlepší charakteristiky pro objekt na obrázku. Pokud tento druh informací v ontologii přítomný není je výsledek rozdělení obrázku prezentován anotátorovi, který je pak schopen popsat vyznačený objekt. Tímto je proces anotace podstatně zrychlen. [16]

Ontologie podporuje automatickou detekci obrázků, ukládá informace o specifických anotovaných objektech a informace ve formě vztahů mezi sémantickými kategoriemi. Na základě těchto znalosti lze automaticky dedukovat významné důsledky a omezení. Například lze dedukovat, že "kočka není nikdy zelená", nebo, že "labuť se často objevuje na obrázku s vodou". Tyto znalosti mohou vylepšit efektivnost automatického rozpoznávání objektů. Popisná data jsou strukturovány do úrovní detailu, kde horní úroveň popisuje sémantiku významných objektů a dolní úroveň obsahuje jejich detaily. Úroveň detailu je podstatná pro navigaci a filtrování informací. Grafická ontologie ukládá tento typ strukturovaných informací a pomáhá urychlit uspořádávání informací. V případě, že uživatel vybere objekt na obrázku, ontologie nabídne validní podobjekty a napomáhá uživateli během tvorby nižších úrovní. Ontologie také podporuje automatickou detekci obrázků. [16]

4.2.2 Spojování na ontologii založených popisů s SVG

GATE využívá přímého zapouzdření bitmapových dat přímo do SVG (Scalable Vector Graphics) formátu, prostřednictvím kterého pak operuje s bitmapovou strukturou. [13]

SVG formát se využívá především k ukládání a organizování základů grafiky. Hlavní myšlenkou popisu rastrových obrázků je vyzvednutí jeho podstatných vlastností, vytvoření jeho popisu a uložení v SVG souboru s původním rastrovým obrázkem. Anotované oblasti se označují pomocí skrytých SVG geometrických obrazců, kde každý má jedinečný identifikátor (například zřetelné mnohoúhelníky, body, polygony, a jiné obrazce). Anotační data, neboli sémantika označených oblastí, se ukládá pomocí metadatových tagů. Primárním účelem těchto tagů je spojit oblasti se specifickými sémantickými kategoriemi s jejich vlastnostmi v ontologii. Kromě výše zmíněného může anotace definovat přídavné informace předepsané ontologií, například vztahy mezi kategoriemi. [16]

5 UML

Cílem jazyka UML (Unified Modeling Language, unifikovaný modelovací jazyk) je poskytnout systémovým architektům, softwarovým inženýrům a vývojářům nástroje pro analýzu, návrh a implementaci softwarově orientovaných systémů a také nástroje pro modelování podnikových a obdobných procesů. [6]

Jazyk UML nenabízí žádný druh metodiky modelování. Poskytuje pouze vizuální syntaxi, kterou můžeme využít při sestavování svých modelů a je použitelný společně se všemi existujícími metodami. Upřednostňovanou metodou užití jazyka UML je Unified Process, protože je pro tento jazyk nejlépe adaptovaná. [2]

5.1 Historický vývoj jazyka UML

Objektově orientované modelovací jazyky se objevily v 80. letech 20. století, když metodologové čelící novému žánru objektově orientovaných programovacích jazyků a narůstajícímu souboru aplikací začali experimentovat s alternativními přístupy analýzy a návrhu. V období mezi lety 1989 a 1994 vzrostl počet objektově orientovaných metod z méně než deseti na více než padesát. Mnoho uživatelů těchto metod mělo problém najít modelovací jazyk, který by odpovídal kompletně jejich potřebám. Do popředí se dostalo několik z metod, a to Boochova metoda, Jackobsonova OOSE (Object-Oriented Software Engineering) metoda a Rumbaughova OMT (Object Modeling Technique) metoda. Tyto tři metody se staly celosvětově uznávanými. Během 90. let Booch, Rumbaugh, Jacobson a další začali vzájemně přijímat myšlenky metod ostatních. Snahou Rumbaugha a Boocha při tvorbě prvních verzí UML počínaje rokem 1994 bylo sjednotit nejlepší z metodik Boocha, OMT a v poslední řadě, při připojení Jackobsona, zahrnout OOSE. Roku 1996 vznikly UML dokumenty a vzhledem k zájmu o využití jazyka UML pro podnikání softwarových organizací bylo založeno UML konsorcium. [3]

První verze UML (UML 1) tedy vznikla ze tří vůdčích objektově orientovaných metod (Booch, OMT, OOSE) a začlenila řadu nejlepších technik z návrhu modelovacího jazyka, objektově orientovaného programování a popisných architektonických jazyků. [6]

Současná přepracovaná verze UML (UML 2) z roku 2005 rozšiřuje původí verzi o přesnější definice pravidel abstraktní syntaxe a sémantiky, více modulárních jazykových struktur a značně vylepšenou schopnost modelování rozsáhlých systémů. Jeden z primárních cílů UML je napomoci stavu

průmyslu umožněním spolupráce vizuálních objektových modelovacích nástrojů. K umožnění smysluplné výměny modelovaných informací mezi nástroji bylo třeba vytvořit dohodu o společné sémantice a notaci. [6]

Od roku 1997 je vývoj jazyka UML pod záštitou sdružení OMG (Object Management Group) a jeho prostřednictvím se UML stalo prvním průmyslovým standardem objektově orientovaného jazyka pro vizuální modelování na světě. [2]

5.2 Konceptuální model UML

Pro pochopení UML je nutné vytvořit konceptuální model jazyka. K tomu je třeba pochopit tři základní prvky [3]:

- Stavební bloky UML. Slovník jazyka UML zahrnuje tři druhy stavebních bloků a to předměty, vztahy a diagramy.
- Společné mechanismy, které popisují, jak lze stavební bloky skládat dohromady.
- Obecné architektonické mechanismy nasazované skrz UML.

5.3 Stavební bloky UML

Jazyk UML je sestaven ze tří stavebních bloků.

Předměty jsou abstrakce a členové prvního řádu v modelu. Vztahy spojují tyto předměty dohromady a diagramy spojují zajímavé soubory předmětů. [3]

5.3.1 Předměty

Předměty v UML mohou být čtyř druhů.

Strukturální abstrakce představují podstatná jména a statické části modelu, reprezentující konceptuální či fyzické elementy. Souhrně se nazývají klasifikátory. Jako strukturální abstrakce můžeme chápat třídy, rozhraní, spolupráci, případ užití, aktivní třídy, komponenty, artefakty a uzly. [3]

Předměty zachycující chování jsou slovesa a dynamické částí UML modelů, reprezentující chování v průběhu času a prostoru. Mezi předměty zachycující chování řadíme interakce, stavové stroje a aktivity. [3]

Sdružující předměty jsou organizační složkou UML. Jsou to krabice, do kterých může být model dekomponován. Jako hlavní typ sdružujícího předmětu jmenujme balíky. [3]

Popisné předměty jsou posledním druhem předmětů v UML a jsou vysvětlující částí UML modelů. Jedná se o komentáře, které lze připsat k popsání, ilustraci a zmínění jakéhokoli elementu v modelu. Hlavním druhem popisného předmětu je poznámka. [3]

5.3.2 Vztahy

Stějně jako předměty, i vztahy v UML mohou být blíže rozděleny do čtyř hlavních druhů, a to vztahů znázorňující závislosti, asociace, generalizace a realizace. [3]

5.3.3 UML diagramy

Diagram je grafická reprezentace množiny elementů, často podaná jako propojený graf vrcholů (předmětů) a cest (vztahů). Různé zakreslení diagramu zobrazuje různé úhly pohledy na systém. Z tohoto důvodu lze diagram chápat jako projekci do systému. [3] Diagramy vytvořené v jazyku UML jsou srozumitelné nejen pro lidi, ale mohou je snadno interpretovat i programy CASE (Computer-Aided Software Engineering). [2]

Teoreticky lze diagram sestavit z jakékoli kombinace předmětů a vztahů, avšak v praxi nastává malé množství častých kombinací, které jsou konzistentní s pěti nejužitečnějšími pohledy zahrnující architekturu software zdůrazňujících systémů. [3]

Bez ohledu na problémovou doménu se při modelování reálných systémů opakují stejné druhy diagramů, protože reprezentují obvyklé pohledy na časté modely. Typicky nahlížíme na statické části systému prostřednictvím jednoho z následujících diagramů:

1. Diagram tříd (Class diagram)
2. Diagram komponent (Component diagram)
3. Diagram složené struktury (Composite structure diagram)
4. Objektový diagram (Object diagram)
5. Diagram nasazení (Deployment diagram)
6. Diagram artefaktů (Artifact diagram)

Často se také používá pět doplňkových diagramů pro pohled na dynamické části systému:

1. Diagram případu užití (Use case diagram)
2. Sekvenční diagram (Sequence diagram)
3. Komunikační diagram (Communication diagram)
4. Stavový diagram (State diagram)
5. Diagram aktivit (Activity diagram)

6 UML diagramy a jejich ontologie

V následujících podkapitolách se věnuji konkrétním modelovacím schématům jazyka UML 2 a pro ně na základě OMG UML specifikace [6] vytvořeným ontologiím. Protože se v práci zaměřuji na sémantiku diagramů, popisuji modelovací schémata převážně z hlediska prvků zobrazovaných v diagramech. Pro tvorbu ukázkových UML diagramů jsem využívala nástroje Visual Paradigm for UML 10.1. Ontologie pro diagramy jsem vytvářela s pomocí nástroje Protégé 4, který spolu s jazykem OWL popisuji v kapitole 4. Přínosným prostředkem pro zobrazení přehledných grafů znázorňujících taxonomie tříd v ontologii mi byl OWLViz plugin, založený na GraphViz softwarovém řešení pro vizualizaci grafů.

Některé elementy a druhy vztahů se mohou v různých UML diagramech opakovat. Z tohoto důvodu jsem nevytvářela jednotlivé ontologie pro každý typ diagramu, ale vytvořila jsem společnou ontologii zahrnující sémantiku všech vybraných UML diagramů. Díky tomuto propojení lze zachytit i souvislosti mezi jednotlivými modelovacími scématy jazyka UML. Jednotlivé UML diagramy jsou definovány podtřídami třídy UMLDiagram v ontologii. Vzhledem k přidávání sémantiky diagramů do ontologie ve stejném sledu, v jakém popisuji jednotlivé diagramy a pro ně vytvořené ontologie, množství tříd a vlastností v ontologii na ukázkových obrázcích roste spolu s postupně přidávanými a v práci popisovanými diagramy.

Z diagramů chování popisuji diagram případu užití, diagram aktivit, sekvenční diagram a stavový diagram. Z diagramů struktury orientujících se na statické aspekty systému se věnuji popisu diagram tříd.

6.1 Diagram případu užití (Use Case Diagram)

Modelování případů užití je jednou z forem inženýrství požadavků. Ke specifikaci požadavků přidává doplňkový způsob získávání a dokumentování požadavků a poskytuje hlavní zdroj objektů a tříd jako prvotní vstup k modelování tříd. [2]

Případy užití zachycují přesně funkčnost, která bude budoucím informačním systémem pokryta a vymezují tak jednoznačně rozsah prací. Každý případ užití popisuje jeden ze způsobů použití systému, popisuje tedy jednu jeho požadovanou funkčnost. Vyvinutý systém nebude obsahovat jinou funkčnost, než jakou popisují případy užití. [12]

Model případů užití obsahuje čtyři komponenty:

- hranice systému (subject, system boundary),
- aktéři (actors),
- případy užití (use cases),
- relace (relationships).

6.1.1 Hranice systému (Subject, System boundary)

Ohraničení zobrazené kolem případů užití je vyznačením území neboli hranic modelovaného systému. [2] Hranice subjektu zachycují rozsah modelovaného systému. Co je součástí systému se nachází uvnitř subjektu a zároveň je díky hranicím patrné co součástí systému není a nachází se za hranicemi subjektu.

6.1.2 Aktéři (Actors)

Aktér je role, ve které vystupuje uživatel v rámci své komunikace se systémem. Jeden fyzický uživatel může vůči systému vystupovat ve více rolích. Jako aktér může vystupovat externí systém, který potřebuje informace ze systému nebo čas, např. v situacích, kdy se v definovanou hodinu spouští v systému bez zásahu obsluhy pravidelné zpracování. V systému může jeden aktér provádět řadu případů užití a obráceně, jeden případ užití může být prováděn více aktéry. Aktéři jsou vůči systému externí entity, které si vyměňují informace se systémem [12]

Zobecnění aktérů (Actor generalization)

Stejné chování lze zachytit pomocí zobecněného aktéra. Zobecněný aktér je obvykle abstrakce vytvořená pro zjednodušené zachycení společného chování aktérů. Konkrétní aktéři dědí od svého abstraktního předka všechny role a relace k případům užití.

6.1.3 Případy užití (Use cases)

Případ užití[1] je sada základních a eventuálně alternativních scénářů (sekvence dialogů uživatele se systémem), které spojuje dohromady společný cíl. Případ užití je vždy iniciován aktérem. Případy užití vyjadřují, co systém nabídne uživateli. [12]

1. Prvním kdo zviditelnil případy užití byl Ivar Jacobson svou publikací Object-Oriented Software Engineering: A Use Case Driven Approach v roce 1992.

6.1.4 Relace (Relationships)

Vztah mezi aktérem a případem užití je znázorněn pomocí plné čáry, což je v jazyku UML symbol přiřazení, a značí komunikaci aktéra a případu užití. [2]

6.1.5 Vztahy (typy závislostí) mezi případy užití

Relace «include»

Relace «include», nastavená mezi případy užití, umožňuje vyčlenění společného chování ze scénářů základních případů užití do rozšiřujícího případu užití. V základním případu užití je nutné určit přesné místo (místo zahrnutí), v němž má být rozšiřující případ užití zahrnut. Základní případ užití není bez rozšiřujícího kompletní. Syntaxi relace «include» lze chápat jako obdobu volání funkce.

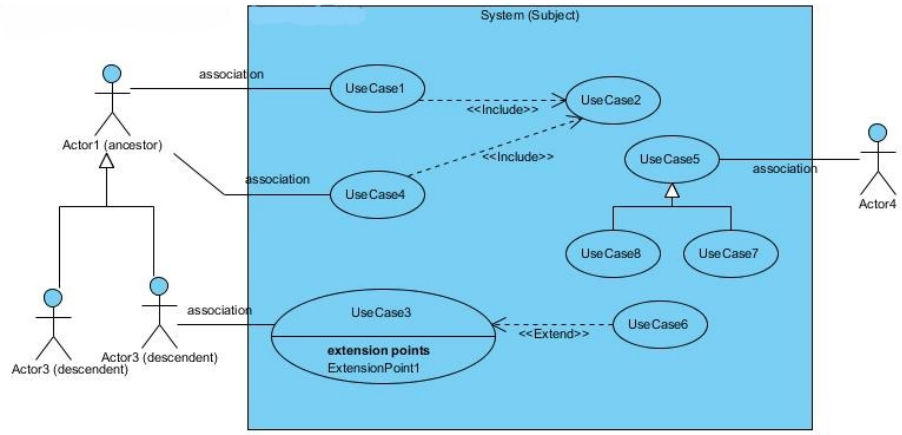
Zobecnění případů užití (Use case generalization)

Obdobně jako zobecnění aktérů umožňuje zobecnění případů užití převést chování společné pro více případů užití do rodičovského případu užití a napomáhá zjednodušení diagramu. Odvozený případ užití (potomek) může dědit funkce a vlastnosti od svých předků, přidávat nové funkce a vlastnosti a překrývat některé prvky zděděných funkcí a vlastností.

Relace «extend»

Relace typu «extend» přidává k základnímu případu užití nové, rozšiřující chování, bez kterého je základní případ užití zcela soběstačný. Případ užití deklaruje tzv. body rozšíření (extension points), které nejsou součástí scénáře. Pouze ukazují místo ve scénáři, kde může být funkčnost rozšířena. Scénář bázového případu užití je zcela nezávislý na bodech rozšíření. Vztah «extend» modeluje volitelné části případů užití. Případ užití může mít více bodů rozšíření a rozšiřující případ užití může rozšiřovat jeden nebo více těchto bodů rozšíření.

Obrázek 6.1 znázorňuje obecný diagram případů užití se zahrnutím všech výše popsaných prvků.



Obrázek 6.1: Diagram případů užití

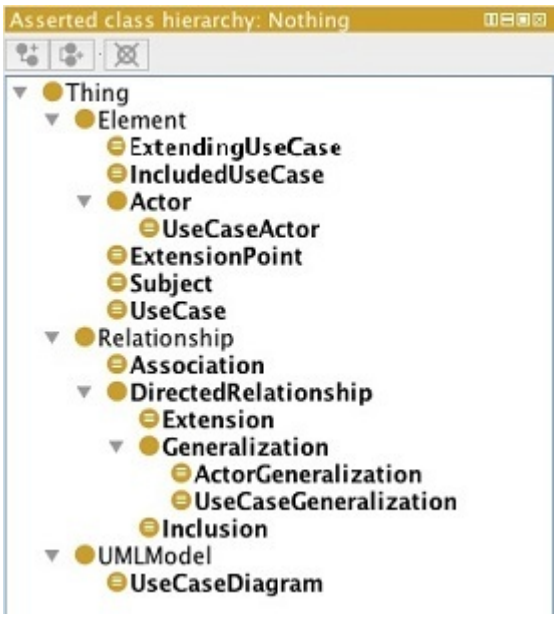
6.2 Ontologie pro diagram případů užití

V této podkapitole popisuji tvorbu ontologie a jednotlivé prvky ontologie pro diagram případů užití.

6.2.1 Třídy ontologie a jejich hierarchie

Hlavním stavebním prvkem OWL ontologií jsou třídy. Prvním krokem při tvorbě ontologie je tedy vytvoření základních tříd a jejich hierarchie. Obrázek 6.2 znázorňuje taxonomii tříd vytvořenou pro diagram případů užití v pohledu hierarchie tříd editoru Protégé. Na dalším obrázku 6.3 je tato hierarchie znázorněna přehledněji prostřednictvím OWLViz pluginu editoru Protégé.

Třídy v ontologii jsem rozdělila do tří hlavních nadtříd. První třída, označená Element 6.4, obsahuje podtřídy zachycující elementy diagramů. V diagramu případů užití tvoří hlavní elementy aktéři (Actor), případy užití (Use Case) a subjekt (Subject). Druhou ze tří hlavních tříd je nadtřída zachycující jednotlivé druhy vztahů (Relationship) 6.5, které mezi elementy uml modelů mohou existovat. Pro diagram případů užití bylo podstatné odlišit vztahy asociační (Association) spojující aktéry s případy užití a směrované (DirectedRelationship) popisující dědičnost (Generalization) aktérů či případů užití, inkluzi (Inclusion) a exkluzi (Exclusion) případů užití. Jako poslední z hlavních nadtříd jsem vytvořila třídu UML diagram 6.6. Každá



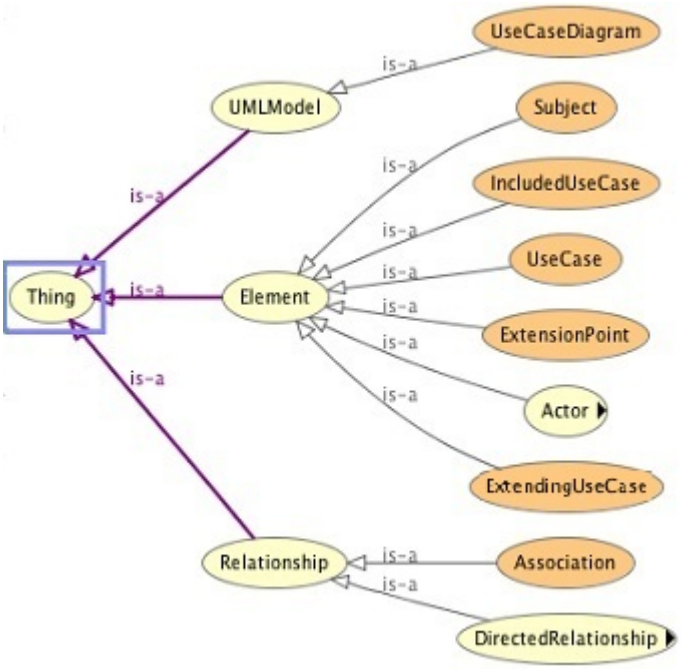
Obrázek 6.2: Třídy v ontologii pro diagram případů užití

podtřída třídy UML diagram zachycuje konkrétní uml model a ve svém popisu zahrnuje pro něj specifické elementy a vztahy.

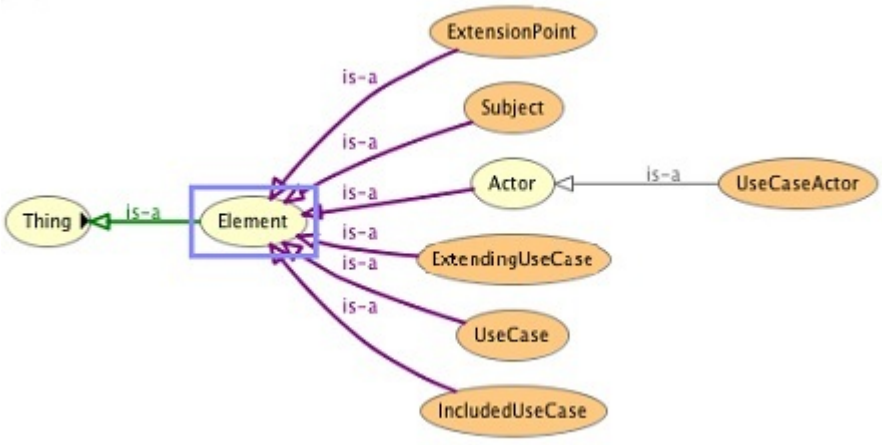
6.2.2 Vlastnosti ontologie a definice tříd

Obrázek 6.7 znázorňuje taxonomii vlastností vytvořenou pro diagram případů užití.

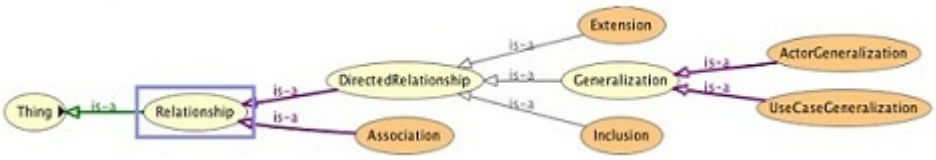
Diagram případů užití využívá dvou sémanticky odlišně zachycených typů vazeb mezi elementy. Kromě jednoduchého nesměrovaného propojení aktérů s případy užití, zachyceného třídou Association a vlastnostmi interactsThrough a mediatesInteractionBetween, se v diagramu často vyskytují vazby zachycující směr zahrnutí touto vazbou spojených elementů. U těchto směrovaných vazeb mezi elementy, zachycených podtřídami třídy DirectedRelationship, je třeba pomocí vlastností v ontologii popsat k jakému elementu je šipka směrována. Z tohoto důvodu jsem vytvořila vlastnosti hasRelatedElement a isRelatedElementOf. Každá z těchto vlastností obsahuje ve své hierarchii níže postavené vlastnosti specifikující jaký element je zdrojem a cílem směrované vazby.



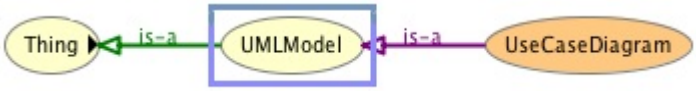
Obrázek 6.3: Celkový pohled na taxonomii tříd diagramu případů užití



Obrázek 6.4: Nadtřída Element a její taxonomie



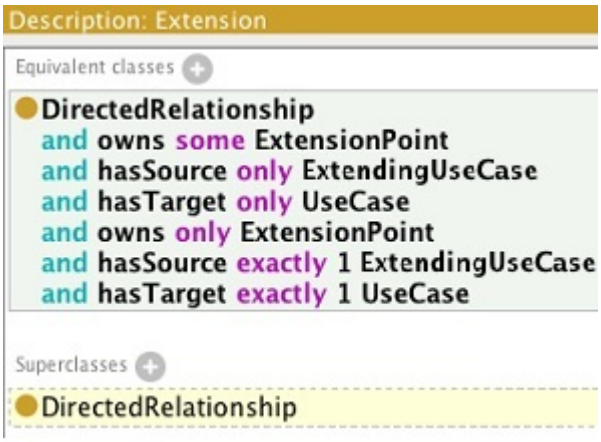
Obrázek 6.5: Nadtřída Relationship a její taxonomie



Obrázek 6.6: Nadtřída UMLModel a její taxonomie

Obrázek 6.7: Vlastnosti v ontologii pro diagram případů užití

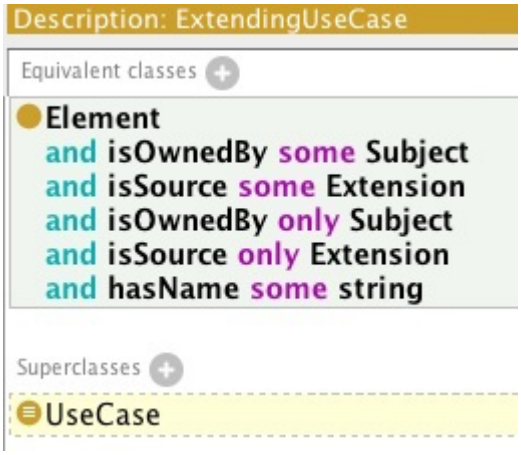
Na obrázku 6.4 je znázorněna definovaná třída Extension, označující směrovanou vazbu mezi rozšiřujícím a rozšiřovaným případem užití. Třídou, ze které šipka zachycující tento specifický druh vazby mezi případy užití vychází (hasSource), je rozšiřující případ užití (ExtendingUseCase) a šipka míří (hasTarget) do rozšiřovaného případu užití, který, protože je bez rozšiřujícího případu užití zcela soběstačný, lze chápat jako obyčejný případ užití (UseCase). Definice třídy ExtendingUseCase zobrazená na obrázku 6.4 vysvětluje vlastnost isSourceOf směrovaných vztahů. Z rozšiřujícího případu užití vychází šipka (isSource) do třídy charakterizující směrovaný vztah mezi elementy. Šipka výsledně míří do případu užití, který je rozšiřován, což zachycuje obrázek definice třídy UseCase 6.10.



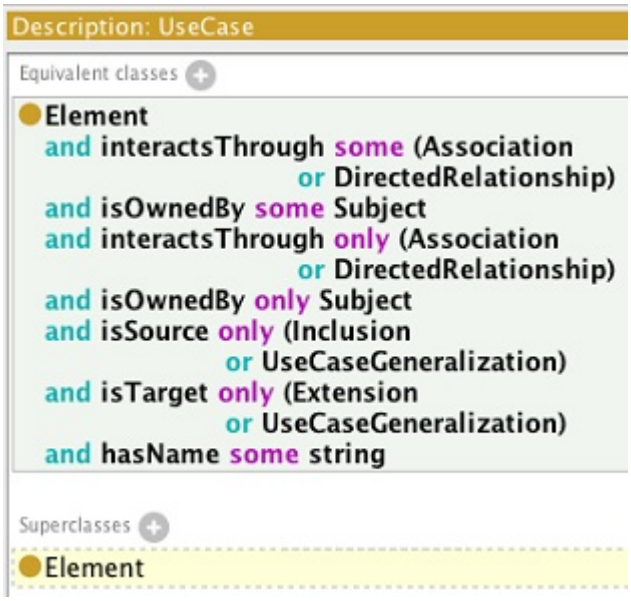
Obrázek 6.8: Definice třídy Extension

Charakteristiky vlastností ontologie pro diagram případů užití

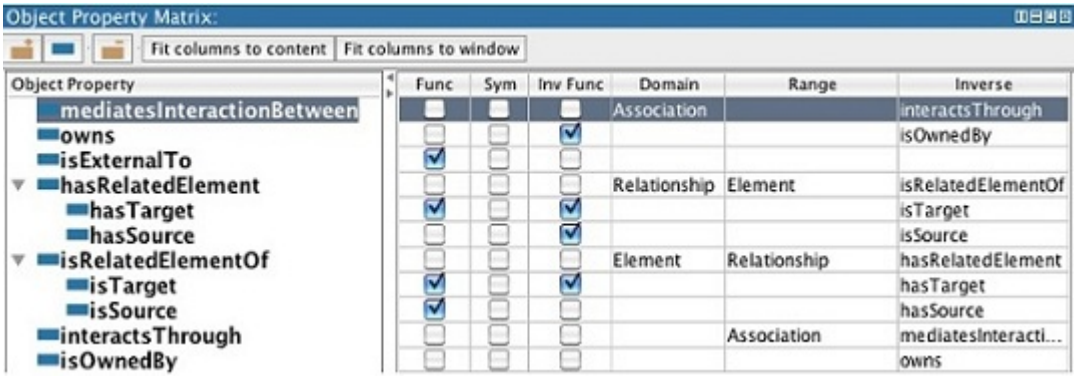
Následující matice vlastností v ontologii pro diagram případů užití 6.11 zachycuje charakteristiky daných vlastností, domény, obory hodnot pokud lze takto vlastnosti omezit a jejich inverzní vlastnosti, pokud nějaké mají. Pro přehlednost a úsporu místa jsem z matice vynechala vlastnostmi nevyužité charakteristiky, jako například reflexivitu, ireflexivitu, tranzitivitu a asymetrii. V původní matici editoru Protégé jsou všechny tyto charakteristiky zahrnuty a lze je u vlastností nastavovat.



Obrázek 6.9: Definice třídy ExtendingUseCase



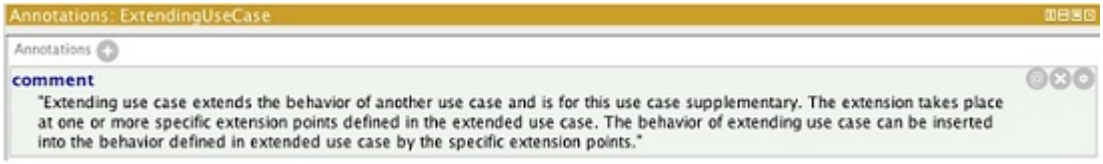
Obrázek 6.10: Definice třídy UseCase



Obrázek 6.11: Matice vlastností ontologie pro diagram případů užití

Anotační vlastnosti

Vzhledem k zaměření práce nejen na rozpoznání uml modelů a jejich sémantiky, ale i k e-learningovým účelům a ku pomoci nevidomým prostřednictvím GATE systému vyvíjeném na fakultě informatiky Masarykovy univerzity, je důležitou součástí ontologie i správné popsání významu jednotlivých prvků ontologie prostřednictvím anotačních vlastností. Pro představu na obrázku 6.12 je ukázka anotační vlastnosti pro rozšiřující případ užití.



Obrázek 6.12: Anotační vlastnost pro třídu ExtendingUseCase

6.3 Diagram tříd (Class Diagram)

Třídy a objekty jsou základními stavebními bloky všech objektově orientovaných systémů. Diagram tříd se využívá k zachycení statické struktury systému, popisuje existující třídy a vztahy mezi nimi. [17] Tento diagram je nejběžnějším diagramem v modelování objektově orientovaných systémů. Diagramy tříd jsou podstatné nejen pro vizualizaci, specifikaci a dokumen-

tování strukturálních modelů, ale i pro konstrukci spustitelných systémů pomocí dopředného a zpětného inženýrství. [3]

6.3.1 Objekt

Objekt můžeme považovat za soudržný balíček dat a funkcí. Jediným způsobem jak se dostat k datům, jež jsou součástí objektu, je užití jedné z funkcí poskytované tímto objektem. [2] Objekt má svou identitu, vlastnosti (atributy), chování (realizováno jeho metodami) a zodpovědnost. [12] Každý objekt je instancí určité třídy, která definuje společnou množinu rysů (atributů a operací či metod), které jsou vlastní všem instancím dané třídy. Objekty spolupracují proto, aby mohly vykonávat funkce poskytované příslušným systémem. Znamená to, že se mezi objekty tvoří spoje, jejichž prostřednictvím si objekty vzájemně předávají zprávy. Kdykoli objekt přijme zprávu, prohledá množinu svých operací (metod), aby zjistil, zda obsahuje operaci, jejíž signatura odpovídá signatuře přijaté zprávy. Pokud ano, zavolá tuto operaci. [2] Vyvolání operace může způsobit změny hodnot atributů objektu. Vyvolání operace může také způsobit změny hodnoty atributů jiných objektů, ke kterým může být navigováno přímo nebo nepřímo, z objektu na kterém byla operace vyvolána. Vyvolání operace může také způsobit tvorbu a zánik objektů. [6]

6.3.2 Třída

Třída je deskriptor množiny objektů, které sdílejí stejné charakteristické vlastnosti (atributy, operace, metody, relace a chování). [2] Cílem třídy je specifikovat klasifikaci objektů a také specifikovat vlastnosti, které charakterizují strukturu a chování těchto objektů. [6]

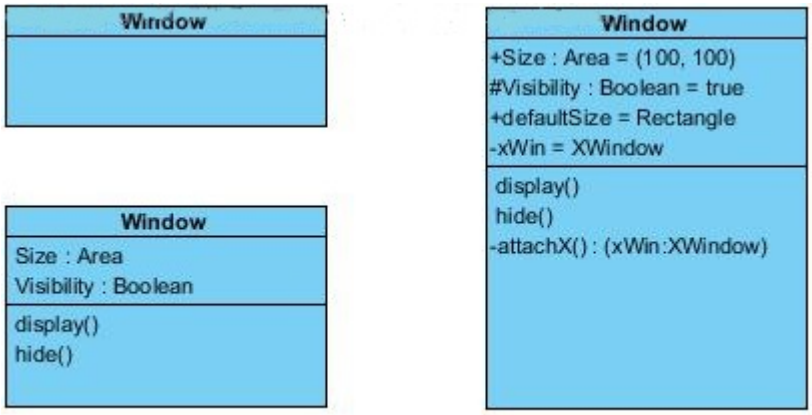
Diagram tříd sestává pouze z jednoho typu komponenty (tříd, classes) a různých vztahů, které mohou mezi třídami v diagramu existovat. Obojí popisuji v následujících podkapitolách.

Notace třídy v jazyce UML

Jedinou povinnou součástí grafické syntaxe třídy v UML je oddíl symbolů s názvem třídy. V případě potřeby zachytit v diagramu bližší informace o dané třídě, je možné grafickou syntaxi rozšířit o další dva oddíly a to oddíl obsahující atributy a oddíl operací třídy. Využití těchto detailnějších oddílů závisí výhradně na účelu diagramu.

Atribut jakožto nositel informací o objektu je definován svým jménem, formátem a viditelností. Chování třídy je definováno operacemi. Některé operace vykonávají operace s daty (všemožné aktualizace) jiné operace mohou být typu "interface", tzn. poskytovat rozhraní ostatním objektům, které požadují služby tohoto objektu. Signatura operace značí charakteristiku operací jako název, seznam parametrů a návratové hodnoty. [12]

Následující obrázek 6.13, blíže popsaný v OMG UML specifikaci [6], znázorňuje možné příklady grafického zobrazení třídy.



Obrázek 6.13: Notace třídy: skryté detaily, analytická úroveň detailů, implementační úroveň detailů

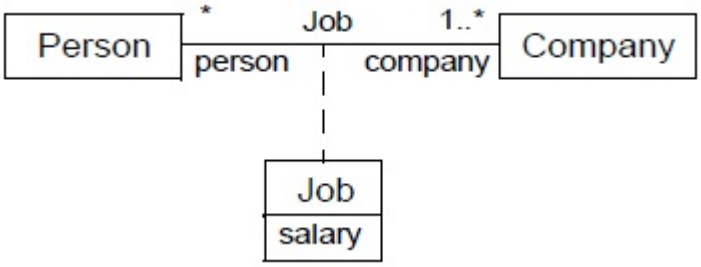
6.3.3 Vztahy mezi třídami

Druhy vztahů, které spojují jednotlivé třídy mezi sebou v diagramu tříd mohou být asociace, agregace, kompozice a generalizace (specializace).

Asociace

Asociace popisují relace mezi třídami. Aby mohlo existovat spojení mezi instancemi, musí existovat rovněž asociace mezi jejich třídami. Asociace mohou obsahovat název asociace, název role, násobnost a průchodnost. Násobnost omezuje počet objektů dané třídy, které se dané relace účastní v libovolném okamžiku. Velmi často jsou třídy asociovány samy se sebou. Tento typ asociace se nazývá reflexivní asociací a znamená, že objekty dané třídy obsahují odkazy na objekty téže třídy. [2]

Asociační třída je druhem asociace, která má vlastnosti třídy. [6] Nejenže spojuje dvě třídy jako asociace, ale navíc definuje množinu vlastností, které patří samotné asociaci. Asociační třídy mohou mít atributy, operace, ale i další asociace. [2] Obrázek 6.13 znázorňuje asociační třídu zobrazenou pomocí asociačního symbolu (přímky) a symbolu třídy (rámečku) spojeného čárkovanou přímkou. Diagram zobrazuje asociační třídu Zaměstnání (Job), která je definovaná mezi dvěma třídami Osoba (Person) a Firma (Company). [6]



Obrázek 6.14: Asociační třída [6]

Generalizace

Dalším důležitým vztahem, který může existovat mezi třídami, je generalizace. Objekt specializující třídy může být zaměněn za objekt více obecné třídy v jakémkoli kontextu, který očekává člena více obecné třídy, ale nikoli obráceně. [17]

Agregace a kompozice

Agregace a kompozice jsou druhy asociace. Kromě poukázání na to, že dvě třídy jsou asociovány se můžeme rozhodnout ukázat více o tomto druhu asociace. [17] Vazba typu agregace říká, že jedna třída je částí druhé třídy. [12] Podřízenost jednoho objektu vůči druhému je v analýze chápána dvojím způsobem, buď jako agregace, nebo jako kompozice. V obou případech se však jedná o vztah dvou objektů, z nichž podřízený objekt má svou objektovou referenci vloženou do prvního objektu. [6]

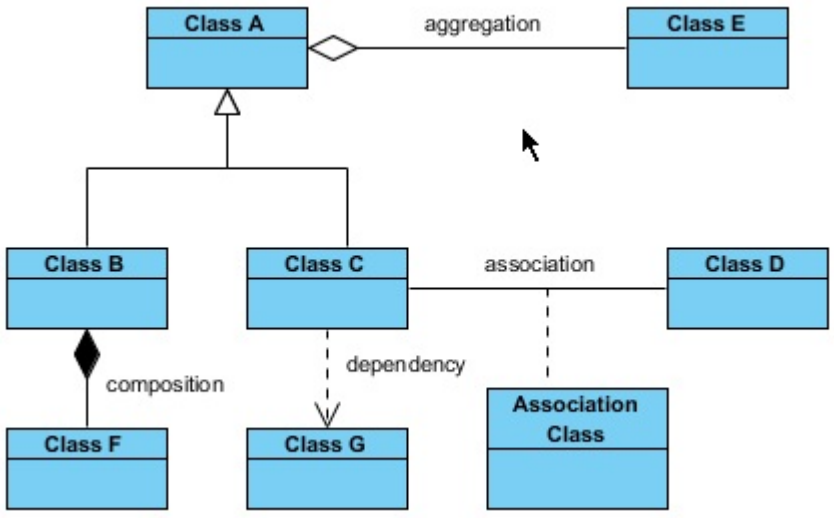
Speciálním typem agregace je kompozice, kdy víme, že podřízený objekt nemůže existovat samostatně bez nadřízeného objektu. Typickým příkladem je doklad a řádky dokladu. [12]

Zatímco pro agregaci platí, že jeden objekt je součástí jiného objektu, u asociace hovoříme o rovnoprávnějším vztahu. U agregace nadřízený objekt využívá dovednosti podřízeného objektu a měl by nést zodpovědnost za jeho vznik a zánik. [12]

Vztah závislosti

Závislost je použití vztahu, který specifikuje, že změna v údajích jednoho prvku může ovlivnit jiný prvek, který ho používá, ale ne obráceně. Graficky se závislost označuje přerušovanou přímkou směrovanou k závislému prvku. [3] Závislosti se používají při modelování relací mezi klasifikátory, kdy jeden klasifikátor je na druhém závislý, ale relace zároveň není fakticky asociací. Relaci závislosti lze specializovat pomocí určitých předdefinovaných stereotypů. K vyjádření závislostí mezi třídami se nejčastěji používá stereotyp «use». [2]

Obrázek 6.15 znázorňuje obecný diagram tříd se zahrnutím všech výše popsaných prvků.



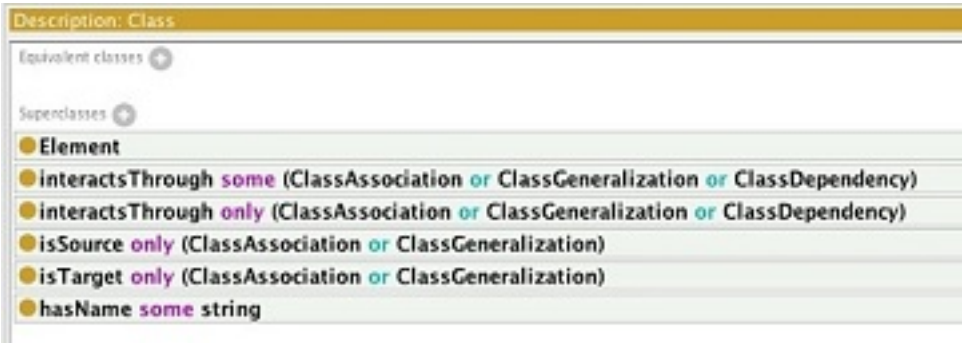
Obrázek 6.15: Diagram tříd

6.4 Ontologie pro diagram tříd

V následující podkapitole popisuji rozšíření původní ontologie o diagram tříd.

6.4.1 Definice tříd v ontologii

Základní elementy ontologie jsem obohatila o entitu Třída (Class). Na obrázku 6.16 je zobrazen její popis. Jako každý element, i Třída musí být řádně pojmenována a mít datovou vlastnost Jméno, komunikace mezi třídami (objekty) probíhá prostřednictvím asociace (ClassAssociation). Třída může mít jiné třídy jako své potomky (ClassGeneralization) a některé třídy mohou být na jiných závislé (ClassDependency).



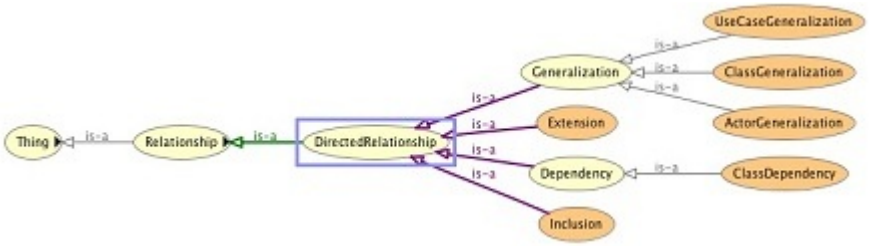
Obrázek 6.16: Element Třída a její popis

6.4.2 Definice vztahů mezi třídami

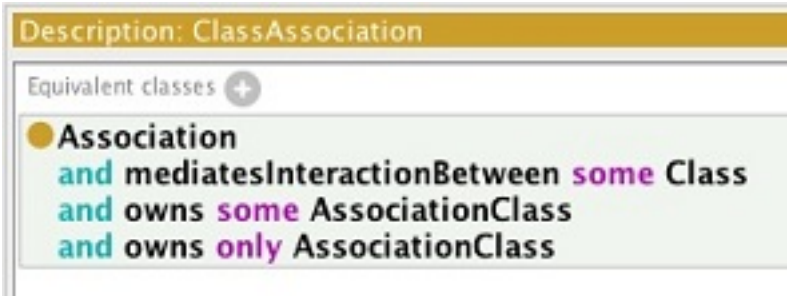
Pro popsání nových vazeb v ontologii jsem rozšířila taxonomii podtříd třídy DirectedRelationship 6.17.

Speciálními druhy tříd v ontologii, které popisují vztahy mezi třídami v diagramu, mohou být asociační třída (AssociationClass) a závislá třída (DependentClass). Obě třídy jsem do ontologie přidala pro zachycení závislostí mezi prvky diagramu tříd. Tyto třídy jsou součástí vazeb popsaných třídami ClassAssociation 6.18 a ClassDependency 6.19 a dědí od svých nadtříd.

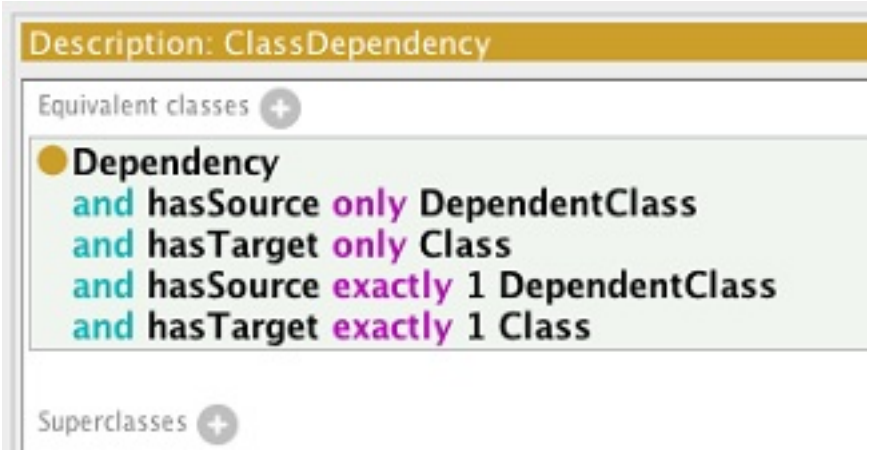
Jako upřesnění asociační vazby mezi třídami diagram tříd poskytuje vztah agregace a silnější vztah nazývaný kompozice. Jejich hierarchii v ontologii popisuje následující obrázek 6.20.



Obrázek 6.17: Podtřídy třídy DirectedRelationship



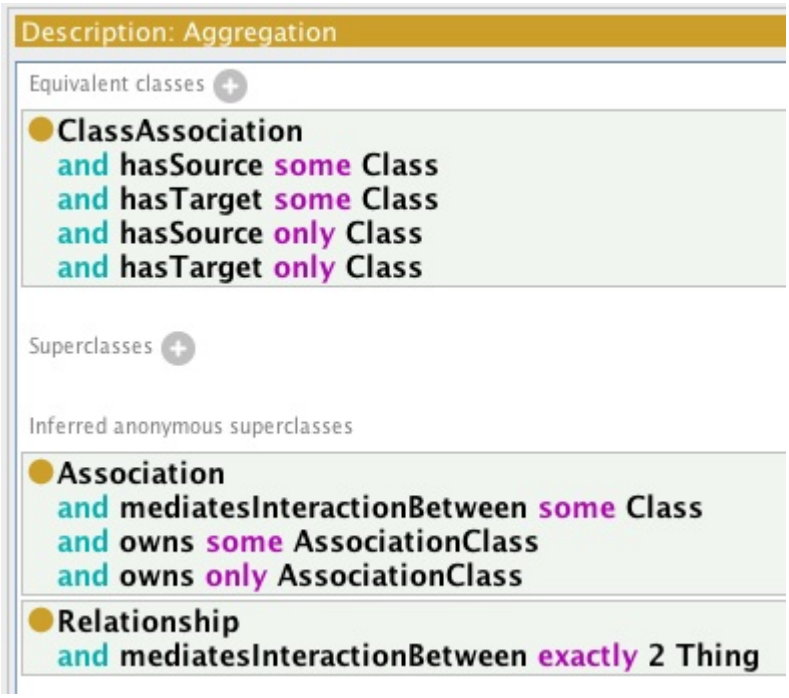
Obrázek 6.18: Asociace tříd



Obrázek 6.19: Závislost tříd

Obrázek 6.20: Hierarchie upřesnění asociace

Popisy tříd zachycujících vazby agregace 6.21 a vazby kompozice 6.22 se nacházejí na následujících obrázcích.



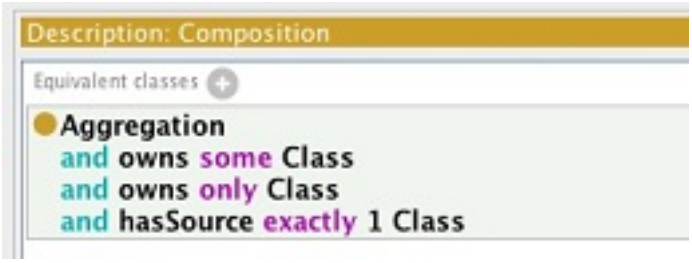
Obrázek 6.21: Agregace

Další z vazeb, které mohou pojit třídy, je generalizace. Popis jejích ekvivalentních tříd je vidět na obrázku 6.23.

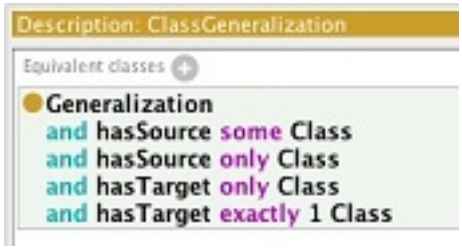
Závěrem jsem rozšířila třídu skládající se z jednotlivých diagramů o popis diagramu tříd 6.24.

6.5 Stavový diagram (State Diagram)

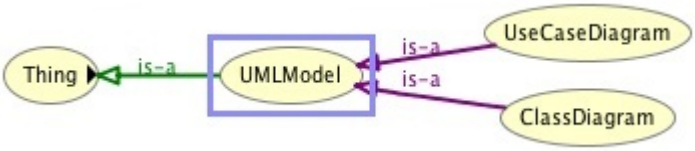
Stavové automaty modelují aspekty dynamického chování systému a používají se k modelování historie životního cyklu jednoho reaktivního objektu



Obrázek 6.22: Kompozice



Obrázek 6.23: Třída definující generalizaci tříd



Obrázek 6.24: Diagram tříd v hierarchii tříd ontologie

jako konečného stavového automatu (automatu, jenž může existovat v konečném počtu stavů). Automat v důsledku událostí přechází mezi těmito stavy přesně definovaným způsobem. [2]

Nejdůležitějšími prvky stavových diagramů jsou [17]:

- stavy
- přechody mezi stavy
- události (vnitřní aktivity stavu)
- počáteční značky
- koncové značky.

Stavy modelují situace během kterých (obvykle implicitně) platí invariantní podmínka. Invariant může reprezentovat statickou situaci jako objekt čekající na nějakou externí událost. Může také modelovat dynamické podmínky jako proces provádění nějakého chování (posuzovaný element vstupuje do stavu, kdy chování zahajuje a opouští tento stav jakmile je chování dokončeno). [6]

6.5.1 Jednoduché stavy

Jednoduchý stav je stav, který nemá podstavy (nemá oblasti a nemá subautomatové stavové automaty). [6]

6.5.2 Souběžné stavové automaty

Každý stav v určité oblasti složeného stavu se nazývá podřízený stav této složené oblasti. Nazýváme ho přímým podstavem, pokud neobsahuje žádné další stavy, jinak je značen jako nepřímý podstav. [6]

6.5.3 Obecná pravidla stavů

V průběhu provádění může být stav aktivní nebo neaktivní. Stav se stává aktivním, když se do něj vstupuje, díky následku nějakého přechodu. Stává se neaktivním, pokud je opuštěn jako následek přechodu. Stav lze opustit nebo do něj vstoupit prostřednictvím stejného přechodu. Při vstupu do stavu se provádí jeho vstupní chování než je provedena jakákoli jiná akce. V případě opuštění stavu provádí stav své výstupní chování jako poslední krok před opuštěním tohoto stavu. [6]

Stav může obsahovat akce a aktivity. Přitom za akci považujeme nepřerušitelný rychle probíhající proces, zatímco aktivita je přerušitelný, jistou dobu trvající proces. [12]

6.5.4 Pseudostavy

Pseudostav je abstrakce, která zahrnuje různé druhy přechodových vrcholů v grafu stavového automatu. [6]

Počáteční a koncový stav

Všechny stavové automaty by měli mít počáteční stav, označovaný jako vyplněný kruh, jenž signalizuje první stav sekvence. Pokud se stavy nepohybují v nekonečném cyklu, měly by stavové automaty mít také koncový stav, značený terčem kolem vyplněného kruhu, jenž ukončuje sekvenci přechodů. [2]

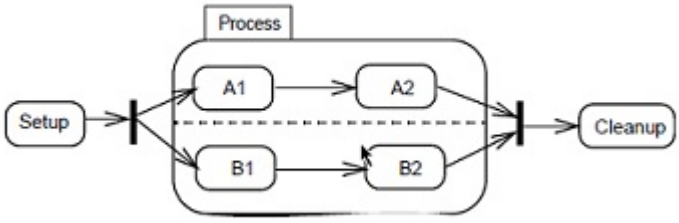
Vstupní a výstupní bod

Každý stav má dvě implicitní akce - vstupní a výstupní, které jsou svázány s událostmi entry a exit. Vstupní bod je zakreslen jako malý kruh na kraji diagramu stavového automatu nebo složeného stavu. Výstupní bod je znázorněn jako malý kruh s křížem na kraji diagramu stavového automatu nebo složeného stavu. Událost entry proběhne automaticky jako první při přechodu do daného stavu objektu a zároveň spustí asociovanou vstupní akci. Událost exit je poslední událostí daného stavu objektu a spouští asociovanou výstupní akci. [12]

Vstupní pseudostavový bod je vstupním bodem stavového automatu nebo složeného stavu. V každé oblasti stavového automatu nebo složeného stavu má nejvýše jedinný přechod k vrcholu se stejnou oblastí. Výstupní pseudostavový bod je výstupním prvkem stavového automatu nebo složeného stavu. [6]

Komplexní přechody

Sjednocující vrcholy, nazývané "join"nebo závory, slouží k spojení více přechodů vycházejících ze zdrojových vrcholů různých ortogonálních oblastí. Větvící vrcholy, nazývané "fork", slouží k rozdělení příchozích přechodů do dvou a více přechodů navazujících na ortogonální cílové vrcholy (vrcholy z různých oblastí složeného stavu). [6] Obrázek 6.25 znázorňuje složený stav ze dvou stavových podautomatů se sjednocujícími a větvícími vrcholy.



Obrázek 6.25: Složený stav se sjednocujícími a větvícími vrcholy [6]

6.5.5 Přechody mezi stavy

Každý přechod obsahuje tři nepovinné prvky a to událost, podmínku a akci. Událost je externí nebo interní výskyt, který zahájí přechod. Podmínka (guard) je booleovský výraz, jehož splnění podmiňuje přechod. Akce je část díla přidruženého k přechodu a dochází k ní při zahájení přechodu. [2]

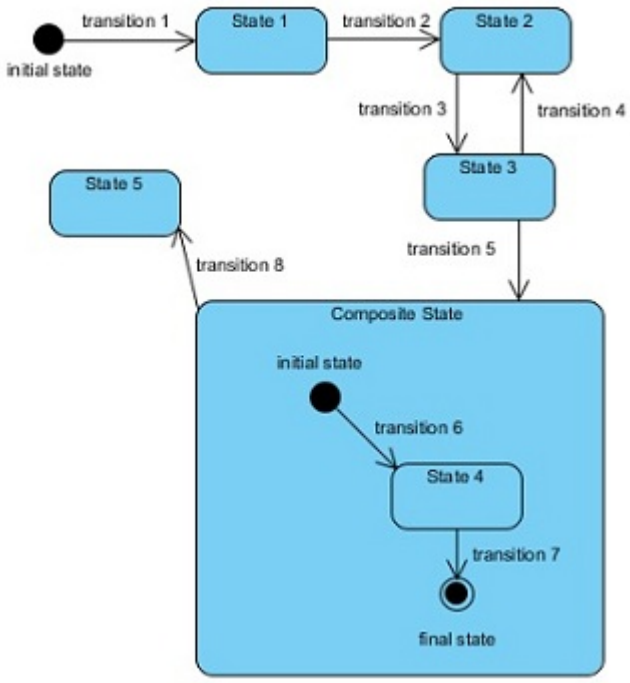
Obrázek 6.26 ukazuje obecný stavový automat s některými výše popsanými prvky.

6.6 Ontologie pro stavový diagram

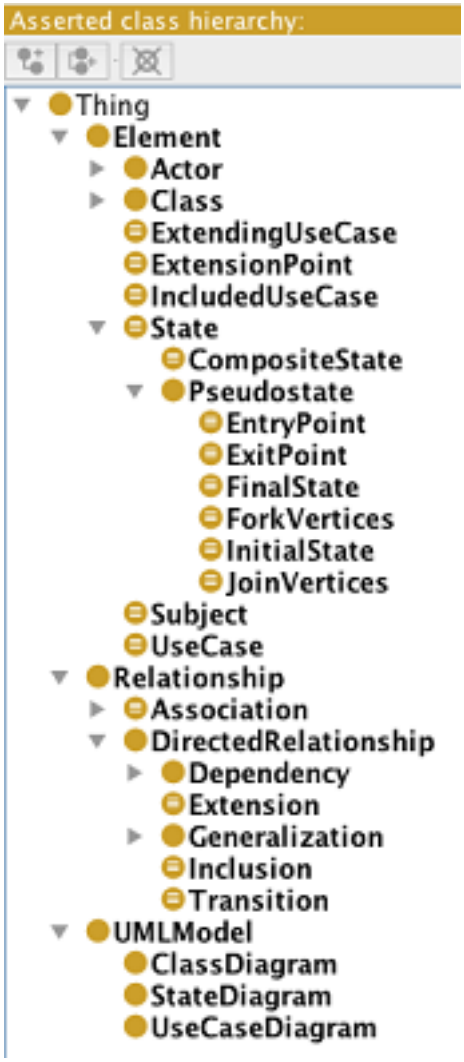
V této podkapitole se věnuji popisu prvků rozšíření ontologie v rámci přidání stavového diagramu.

6.6.1 Třídy v ontologii

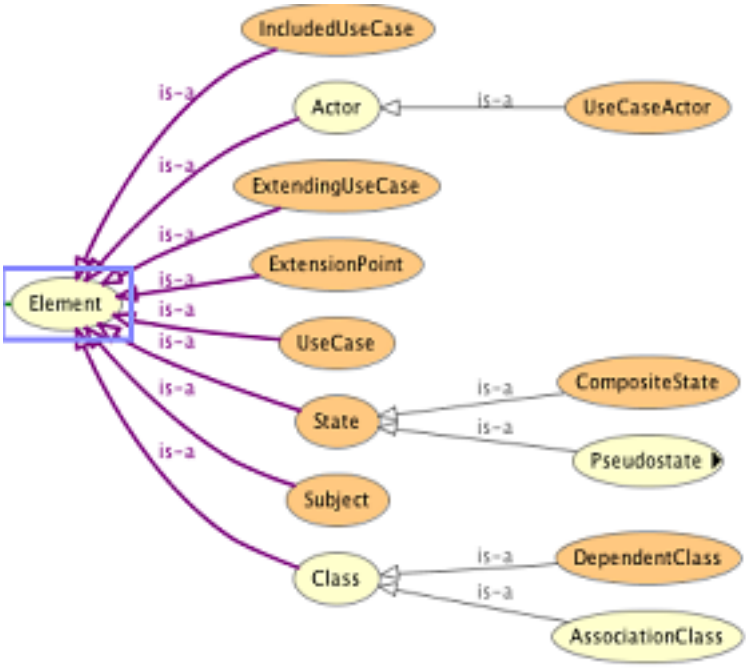
V hierarchii tříd v ontologii došlo k poměrně širokému rozšíření, převážně v podtřídě Element. Třída State znázorňuje stavy ve stavovém diagramu, včetně pseudostavů (Pseudostate) a složených stavů (CompositeState). Mezi pseudostavy specifikace UML zahrnuje vstupní (EntryPoint) a výstupní body (ExitPoint), iniciální (InitialState) a konečné stavy (FinalState) a sjednocující (JoinVertices) a větvící vrcholy (ForkVertices). V třídě Relationship došlo k rozšíření o podtřídu Transition, popisující přechod mezi stavy. Poslední přidanou třídou do ontologie je třída StateDiagram, jako podtřída UMLDiagram třídy. Obrázek 6.27 ukazuje hierarchii tříd po rozšíření o stavový diagram v editoru Protégé. Stav podtříd třídy Element po rozšíření o stavový diagram je graficky zachycen na obrázku 6.28. Obrázek 6.29 znázorňuje hierarchickou strukturu třídy State.



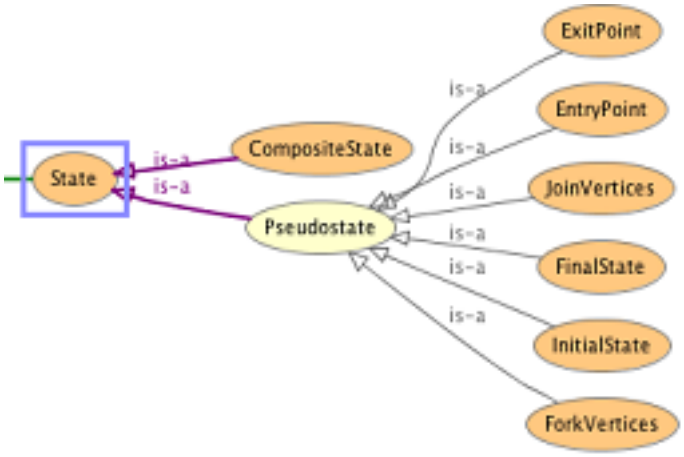
Obrázek 6.26: Obecný stavový diagram



Obrázek 6.27: Třídy v ontologii po rozšíření o stavový diagram



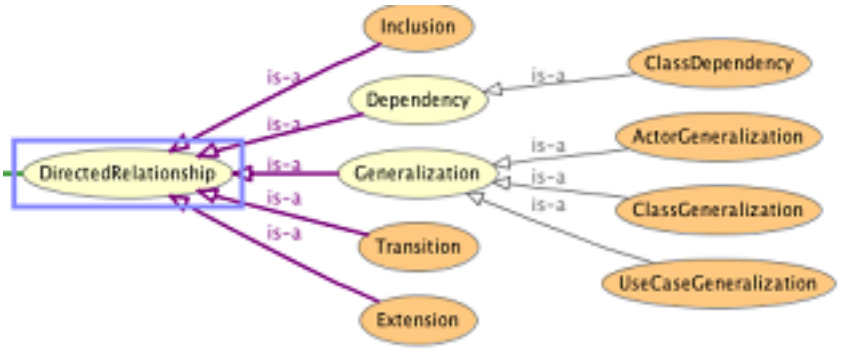
Obrázek 6.28: Podtřídy třídy Element po rozšíření o stavový diagram



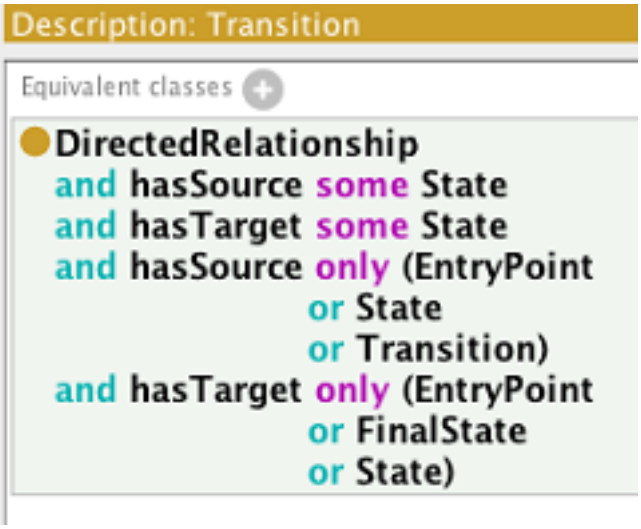
Obrázek 6.29: Hierarchie třídy State

6.6.2 Definice tříd

Přechody mezi stavy jsou zachyceny prostřednictvím třídy Transition 6.30. Její definice je znázorněna na obrázku 6.31. Pomocí objektových vlastností hasSource a hasTarget naznačuji směr orientace přechodu mezi stavy. Zdrojem přechodu je stav. Šipka přechodu ve stavovém diagramu směřuje do následujícího stavu v posloupnosti. Dále definice třídy zachycuje, že přechod může být zdrojem nějakého iniciálního stavu a cílem stavu koncového.

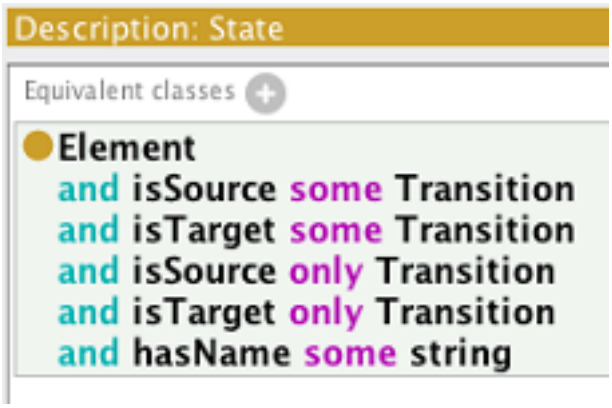


Obrázek 6.30: Třída popisující přechod mezi stavy



Obrázek 6.31: Definice třídy Transition popisující přechod mezi stavy

Prostý popis třídy stavu (State) 6.32 rozšiřují její podtřídy. Například třída FinalState 6.33, která popisuje konečný stav ve stavovém diagramu omezuje definici třídy State pouze na jeden možný vstupující přechod. Obdobně třída popisující vstupní stav může mít pouze jeden výstupní přechod a toto omezení je zachyceno v ontologii.



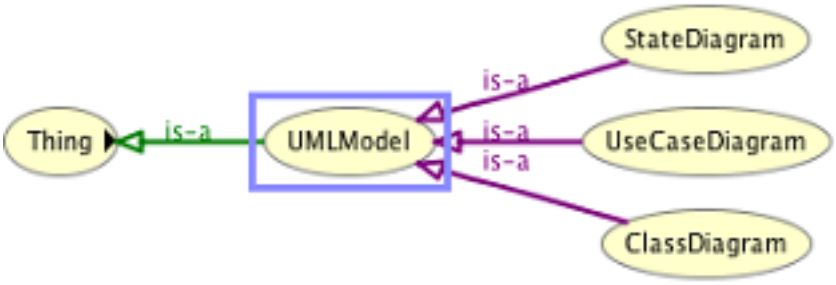
Obrázek 6.32: Třída definující stav ve stavovém diagramu

Třída popisující větvící vrchol (ForkVertex) 6.34 přidává do ekvivalentních tříd podmínku o vycházejících přechodech. Aby došlo k větvení, musejí z větvícího vrcholu vycházet minimálně dva přechody.

Na závěr mezi podtřídy třídy UMLDiagram přibyla třída zachycující stavový diagram. 6.35

Obrázek 6.33: Definice třídy pro konečný stav (FinalState) stavového diagramu

Obrázek 6.34: Definice třídy větvícího vrcholu (fork)



Obrázek 6.35: Grafové znázornění podtříd třídy UMLDiagram

7 Závěr

Ve své diplomové práci jsem se věnovala popisu podstatných konceptů pro tvorbu sémantiky a ontologií. Nejprve jsem popsala základní rysy datového modelu RDF. Následně jsem se věnovala popisu jazyku OWL a editoru Protégé. Pochopení těchto konceptů bylo iniciální pro tvorbu výsledné ontologie. Abych práci uvedla do širšího kontextu, část práce jsem věnovala i projektu GATE, který se věnuje anotaci obrázků mimo jiné prostřednictvím OWL ontologií a se kterým má práce úzce souvisí. Na základě teoretických poznatků z výše popsaných oblasti a specifikaci jazyka UML jsem vytvořila v prostředí Protégé ontologii pro několik vybraných UML diagramů, a to konkrétně pro diagram případu užití (Use Case Diagram), diagram tříd (Class Diagram) a stavový diagram (State Diagram).

Ontologie pro dané diagramy bude tvořit základ dialogového systému pro rozpoznávání informací obsažených v daných UML diagramech. Každý prvek diagramů je popsán pomocí anotačních vlastností. Tyto popisy prvků diagramů jsou nedílnou součástí ontologie a budou v budoucnu sloužit pro e-learningové účely.

Danou ontologii a specifikaci prvků jazyka UML v rámci této ontologie lze využít při možném rozšíření o další UML diagramy. Mezi vhodné diagramy pro rozšíření ontologie lze zařadit sekvenční diagram (Sequence diagram) a diagram aktivit (Activity diagram).

A Obsah přiloženého CD

Součástí práce je přiložené CD, kde je k dispozici ontologie pro popisované UML diagramy a tato práce ve formátu PDF.

Literatura

[1] Ritesh AGRAWAL. Difference between OWL Lite, DL, and Full [online], 2007. Dostupné z: <http://ragrawal.wordpress.com/2007/02/20/difference-between-owl-lite-dl-and-full/>.

[2] Jim ARLOW a Ila NEUSTADT. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. Dotisk 1. vyd. Překlad Bogdan Kiszka. Brno: Computer Press, 2011, 567 s. ISBN 978-80-251-1503-9.

[3] Grady BOOCH a Rob POOLEY. *The unified modeling language user guide: software engineering with objects and components*. 2. vyd. Upper Saddle River: Addison-Wesley, 2005, 475 s. ISBN 03-212-6797-4.

[4] Stanford Center for Biomedical Informatics Research. Protégé [online], 2013. Dostupné z: <http://protege.stanford.edu/>.

[5] Stanford Center for Biomedical Informatics Research. Using the Protégé-OWL Reasoner API [online], 2013. Dostupné z: <http://protege.stanford.edu/plugins/owl/api/ReasonerAPIExamples.html>.

[6] Object Management Group. *OMG Unified Modeling Language™ (OMG UML), Superstructure [online]*. Version 2.4.1. Object Management Group, Inc., 2011, 732 s. Dostupné z: <http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/>.

[7] RDF Working Group. RDF description framework (RDF) [online], 2004. Dostupné z: <http://www.w3.org/RDF/>.

[8] Patrick HAYES. RDF semantics: W3C recommendation [online], 2004. Dostupné z: <http://www.w3.org/TR/2004/PR-rdf-mt-20040210>.

[9] Johan HJELM. *Creating the semantic Web with RDF: Professional developer's guide*. New York: Wiley, 2001, 277 s. ISBN 04-714-0259-1.

[10] Matthew HORRIDGE. *A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools [online]*. Edition 1.3. The University Of Manchester, 2011, 107 s. Dostupné z: <http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf>.

[11] Geert-Jan HOUBEN. Web Information Systems (2II35): Wis Data, Semantic Web, RDF(S) [online], 2008. Dostupné z: <http://wwwis.win.tue.nl/~houben/wis/>.

[12] Hana KANISOVÁ a Miroslav MÜLLER. *UML srozumitelně: objektově orientovaná analýza a návrh prakticky*. Dotisk 2. aktualiz. vyd. Překlad Bogdan Kiszka. Brno: Computer Press, 2007, 176 s. ISBN 80-251-1083-4.

[13] Ivan KOPEČEK a Radek OŠLEJŠEK. Gate to Accessibility of Computer Graphics, Computers Helping People with Special Needs: 11th international conference, Berlin. *Springer-Verlag*, pages 295–302, 2008.

[14] Czech republic LSD FI MUNI, Brno. About GATE Project [online], 2013. Dostupné z: <http://lsd.fi.muni.cz/gate/about>.

[15] Franc MANOLA a Eric MILLER. RDF Primer: W3C Recommendation [online], 2004. Dostupné z: <http://www.w3.org/TR/2004/REC-rdf-primer-20040210>.

[16] Jaromír PLHÁK. *Dialogue Based Processing of Graphics [online]*. PhD thesis, Rigorózní práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce Ivan Kopeček, 2009. Dostupné z: <http://is.muni.cz/th/60773/fi_r/>.

[17] Perdita STEVENS a Rob POOLEY. *Using UML: Software Engineering with Objects and Components*. Updated ed. New York: Addison-Wesley, 2000, 256 s. ISBN 02-016-4860-1.

[18] Jiří ŠTENCEK. Užití sémantických technologií ve značkovacích jazycích. kapitola 3. principy sémantického webu [online]. Bakalářská práce. Vysoká škola ekonomická v Praze. Fakulta informatiky a statistiky. Katedra informačního a znalostního inženýrství, 2009. Dostupné z: <http://vse.stencek.com/semanticky-web/ch03s04.html>.

[19] Wikipedia. Web Ontology Language [online], 2013. Dostupné z: <http://en.wikipedia.org/wiki/Web_Ontology_Language>.