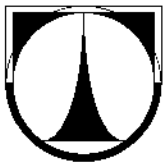
TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií



DIPLOMOVÁ PRÁCE

Liberec 2009 **Bc. Jan Loufek**

Fakulta mechatroniky, informatiky a mezioborových studií

Zadání bakalářské / diplomové práce

Příjmení a jméno studenta (osobní číslo - nepovinné)	Bc. Jan Loufek (M07000178)
Zkratka pracoviště	MTI
Datum zadání DP	31. 10. 2008
Plánované datum odevzdání	29. 5. 2009
Rozsah grafických prací	Dle potřeby dokumentace
Rozsah průvodní zprávy	cca 40 – 50 stran
Název DP (česky)	Zásuvný modul pro matching ontologií v systému Protégé
Název DP (anglicky)	Ontology matching plugin for Protégé system

Zásady pro vypracování DP:

1) Student se v první části seznámí s jazykem Java, se systémem Protégé a možnostmi tvorby jeho zásuvných modulů.
2) Student se dále seznámí s metodami matchingu ontologií a existujícími volně dostupnými nástroji pro matching a vytvoří zásuvný modul do systému Protégé, který bude podporovat základní metody matchingu OWL ontologií a práci s vybranými nástroji.
3) Zásuvný modul student prověří na vhodných testovacích OWL ontologiích.
4) Zásuvný modul bude naprogramován v jazyce Java s důrazem na přehledné ovládání a zobrazení výsledků.

Seznam odborné literatury:

EUZENAT, Jérôme – SHVAIKO, Pavel. Ontology Matching. Springer-Verlag, Berlin/Heidelberg, 2007. ISBN 3-540-49611-4.

DARVIN, Ian F. JAVA – Kuchařka programátora. Computer Press, Praha, 2006. ISBN 80-251-0944-5.

HOEBER, Mark – HOMMEL, Scott – RABINOVITCH, Isaac – RISSER, Tom – ROYAL, Jacob – ZAKHOUR, Sharon. Java 6 – Výukový kurz. Computer Press, Praha, 2007. ISBN 978-80-251-1575-6.

Ontologymatching.org [online]. 2006. Dostupný z WWW: <http://www.ontologymatching.org>.

Protégé – Ontology Editor and Knowledge Acquisition System [online]. 1997. Dostupný z WWW: <http://protege.stanford.edu>.

Vedoucí BP/DP	Ing. Pavel Tyl, MTI
Konzultant BP/DP (u externích pracovníků uveďte plný název pracoviště – firmy)	Ing. Jiří Týř, NTI

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Zásuvný modul pro matching ontologií v systému Protégé

Ontology matching plugin for Protégé system

Diplomová práce

Autor: **Bc. Jan Loufek**
Vedoucí práce: Ing. Pavel Tyl
Konzultant: Ing. Jiří Týř

V Liberci 5. 5. 2009

Na této stránce bude originál zadaní DP

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé diplomové práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své diplomové práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce a konzultantem.

Datum 5. 5. 2009 Jan Loufek

. . . . . .

Poděkování

V této části bych chtěl poděkovat především komunitám vytvářejícím open-source software a to nejen v akademickém prostředí. Především tedy těm, díky kterým jsem měl možnost vytvořit tuto diplomovou práci. Open-source je dle mého názoru hnacím mechanismem pro různá odvětví informačních technologií a měl by být především v akademickém prostředí silně podporován.

Mé poděkování také patří Ing. Pavlu Tylovi, který tuto diplomovou práci vedl, umožnil mi částečné nahlédnutí do problematiky tvorby a zpracování ontologií a také mi pomocí již existujících aplikací nastínil cestu, podle které se vývoj tohoto modulu ubíral.

V neposlední řadě bych rád poděkoval rodičům, kteří mě celá léta podporují ve studiu a dali mi vhodné podmínky k mému působení na univerzitě.

Abstrakt

Cílem tohoto projektu je vytvoření zásuvného modulu pro systém Protégé. Zásuvný modul slouží k porovnávání dvou ontologií a vyhledávání souvislostí mezi jednotlivými třídami těchto ontologií.

Projekt se především bude zabývat sledováním závislostí na slovní bázi (např. slovní vzdálenosti, n-gramy nebo využívání synonym).

Výsledný kód bude pro přehlednost důkladně okomentován a bude navržen takovým způsobem, že základní funkční část zásuvného modulu bude oddělena tak, aby bylo snadné dodělat jiné metody, aniž by se muselo zasahovat do jádra vlastního modulu.

Tento zásuvný modul by také měl být snadno rozšiřitelný o případné další metody.

Klíčová slova: matching, ontologie, Protégé, zásuvný modul

Abstract

The aim of this project is to create plug-in for the Protégé system. This plug-in is used to compare two ontologies and to seek relations between several classes of those ontologies.

Project will mainly consider relations within the word base (e.g. word distances, n-grams or synonyms usage).

Due to the lucidity the final code will be properly commented. The fundamental functional part of the plug-in will be separated so that we can easily add anytime other methods without a need of an intervention to the core of this module.

This plug-in should be easily extended by another methods.

Keywords: matching, ontology, Protégé, plug-in

Obsah

Seznam obrázků

Seznam použitých zkratek

API	Application Programming Interface
C-OWL	Contextualized Web Ontology Language
DAML	DARPA Agent Markup Language
DARPA	Defense Advanced Research Projects Agency
DBMS	Database Management System
GUI	Graphical User Interface
JWI	Java Wordnet Interface
LGPL	Lesser General Public License
MIT	Massachusetts Institute of Technology
OIL	Ontology Interchange Language
OKBC	Open Knowledge Base Connectivity
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
UML	Unified Modeling Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language

Úvod

Současný trend ve vývoji informačních technologií je ve shromažďování velkého objemu dat, která nelze, nebo jen velmi obtížně, automaticky zpracovávat pomocí agentových služeb. Ve světě internetu je uchováváno velké množství znalostí, mezi kterými nelze strojově nalézt vztahy, jež by vedly k nalezení relevantních výsledků. Jedním východiskem z tohoto problému je použití ontologií, které nám do jisté míry umožní označit data a tím jim dát význam.

Cílem diplomové práce je seznámit se s možnostmi vytváření zásuvných modulů pro systém Protégé, především pak s možnostmi porovnávání více ontologií v tomto systému. Proces nazývaný „matching ontologií" nám umožňuje nalézat vazby mezi dvěma a více doménovými ontologiemi, zaměřujícími se vždy na jistý výsek světa. K vytvoření tohoto modulu bude využit programovací jazyk Java, ve kterém je napsán i samotný systém Protégé.

Samozřejmou součástí této práce bude podrobné prostudování problematiky ontologií jakožto základního kamene vývoje Sémantického Webu a také problematiky matchingu ontologií, jež popisuje různé metody, díky nimž lze odhalit vztah mezi objekty ontologií.

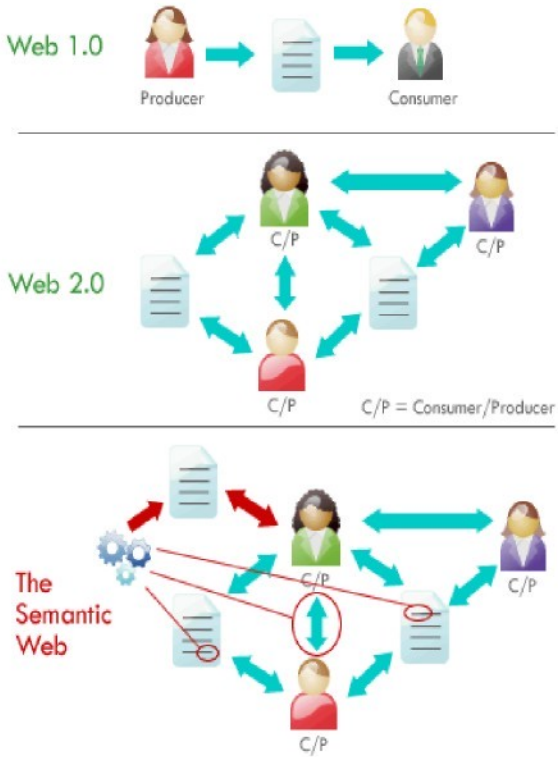
Tato práce bude vytvářena ve vývojovém nástroji Eclipse, pro který jsou popsány výukové texty, umožňující snadnější náhled do problematiky práce s ontologiemi pomocí API, které nám tento systém nabízí. Pozornost bude věnována především ontologickému jazyku OWL, standardizovanému pracovní skupinou W3C Web Ontology a vytvořenému tak, aby pokryl většinu požadavků na ontologie.

Jako inspirace pro vytváření tohoto modulu poslouží aplikace COMA++ a dostupný modul Prompt pro systém Protégé. V poslední části tohoto projektu bude provedeno otestování funkcionality zásuvného modulu na vhodných testovacích ontologiích.

1 Vymezení pojmů

1.1 Sémantický Web

Idea sémantického webu byla světu poprvé prezentována v květnu roku 2001. Tim Berners-Lee, tvůrce současného webu a ředitel Konsorcia W3C spolu s dalšími spoluautory čtivě, s nadhledem, ale důrazně upozornili v článku časopisu Scientific American na skutečnost, že současná síť WWW je v podstatě jen haldou webových stránek, která neustále roste a ve které je stále složitější nalézt relevantní informace. Východisko z tohoto chaosu spatřují v postupném přerodu stávajícího webu v tzv. sémantický web, jehož uživatelská představa je vyjádřena hned v úvodu jejich článku. Hovoří se tam o světě, kde jsou inteligentní zařízení schopná navzájem automaticky komunikovat, jednat a řešit za člověka nejrůznější praktické úlohy, jejichž řešení se opírá o informace, znalosti a jejich důvěryhodné sdílení.



Obrázek 1-1: Vývoj webových služeb

Zdroj: http://blogs.nesta.org.uk/innovation/2007/07/the-future-is-s.html

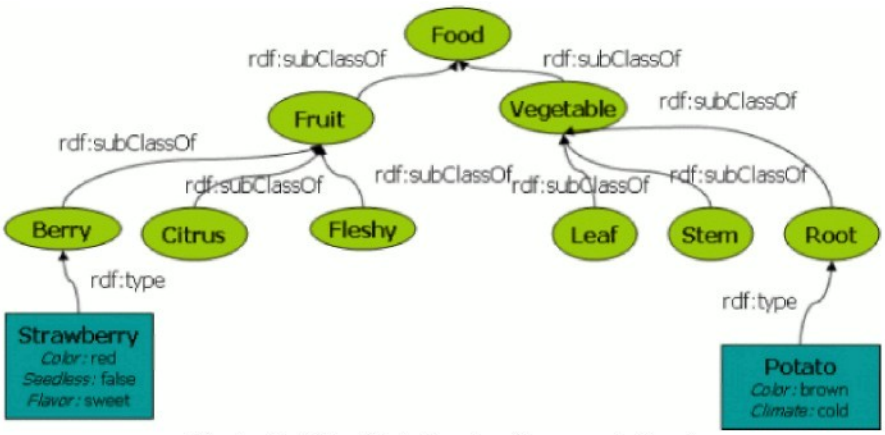
Poněkud suchopárněji lze sémantický web charakterizovat takto: Sémantický web je rozšířením současného webu, v němž informace mají přidělen jasně definovaný význam lépe umožňující počítačům a lidem spolupracovat. Sémantický web představuje reprezentaci dat na WWW. Je založen na technologii Resource Description Framework (RDF), která integruje širokou škálu aplikací využívajících syntaktický zápis v XML a identifikátory URI pro pojmenovávání.

Jde tedy o to, aby data prezentovaná na internetu měla přesně definovaný význam a dovolovala do značné míry automatizované zpracování. [3]

1.2 Ontologie

Vize sémantického webu předpokládá organizaci a provázanost informací umožňující jejich zpracování a „pochopení" nejen člověkem, ale i počítači. Prostředkem k dosažení tohoto dlouhodobého cíle mají být mj. ontologie.

Ontologie v tradičním filosofickém pojetí označuje nauku o bytí (jsoucnu). Vzhledem k tomu, že dnešní stroje nemají dostatek inteligence, aby mohly být rovnocenným partnerem člověka, je jejich „jsoucno" modelováno obvykle pomocí sady pojmů, které zadává člověk. V oblasti informatiky jsou dnes ontologie chápány jako explicitní, formální specifikace pojmů a vztahů mezi nimi.



Obrázek 1-2: Ukázka struktury ontologie

Zdroj: http://www.sei.cmu.edu/isis/guide/technologies/owl-s.htm

Cílem ontologie je definovat společné, jednotné chápání určité třídy pojmů. Je zřejmé, že „jednotnosti" se snáze dosáhne v určité uzavřené oblasti – doméně. Proto

dnes existuje poměrně velké množství tzv. doménových ontologií. Aktuální otázkou výzkumu je, zda pro úspěšnou realizaci informačních systémů budoucnosti postačí rozšiřování počtu těchto „kamínků mozaiky“ sémantického webu, či zda je nutné zapojit i nějakou obecnou, široce pojatou, „všeobjímající“ ontologii. [4]

1.3 Matching ontologií

Ontologie jsou užitečné pro mnoho aplikací (integrace informací, P2P systémy, e-komerce, služby sémantického webu, sociální sítě atd.). Používání a vývoj ontologií různými skupinami uživatelů nebo vývojářů vede (obecně) ke vzniku problému heterogenity. Integrace heterogenních ontologií (či jejich zdrojů) může pomoci tento problém řešit, umožnit správu ontologií a kooperaci v různých skupinách ontologického inženýrství.

Matching ontologií je nadějné řešení pro problém sémantické různorodosti. Umožňuje nalézt shody mezi sémanticky souvisejícími prvky ontologií. Tyto shody mohou být využity pro různé činnosti jako je slučování ontologií, zodpovídání dotazů, překlad dat nebo pro zdroje v sémantickém webu. Tudíž matching ontologií dává možnost sdružení znalostí a vyjádření údajů s využitím složených ontologií. [5]

1.4 Jazyk JAVA

Java je programovací jazyk pocházející od firmy Sun Microsystems. Jedná se o objektově orientovaný jazyk vycházející z jazyka C++, ke kterému má také syntakticky nejblíž.

Oproti svému předchůdci Java neobsahuje některé konstrukce, které způsobovaly při programování největší potíže a navíc přidává mnoho užitečných vlastností – například:

- Přidělování a uvolňování paměti je zde obstaráno automaticky (pomocí garbage collectoru)
- Klasický problém z C/C++, ukazatele, je zde zcela odstraněn, neboť ty zde prostě nejsou (resp. jsou nahrazeny referencemi)
- Je implementován mechanismus vláken (threads) a lze tudíž spouštět více úloh v rámci jednoho programu

- Vytvořené objekty lze automaticky serializovat, tj. ukládat do souboru, zasílat po síti apod.
- Lze provádět introspekci, neboli zjišťování informací o objektu (jaké má proměnné, metody atd.)
- Je implementován mechanismus výjimek, takže veškeré runtime chyby je možné odchytit a zpracovat. Výjimky jsou samozřejmě objektové, takže lze zachytit i celou hierarchii výjimek v jednom hlídaném bloku
- Značným ulehčením pro programátory je obsáhlost standardně dodávaných knihoven, se kterou se nemůže srovnávat asi žádný běžně používaný jazyk
- Java podporuje tvorbu dynamicky rozšiřitelných aplikací (plug-inů apod.)
- Java klade značný důraz na bezpečnost

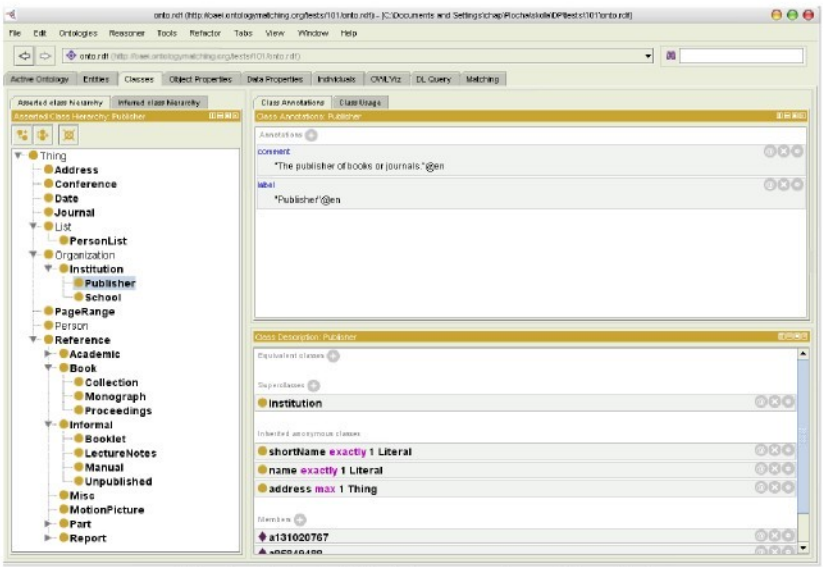
Velkou výhodou Javy je také její hardwarová nezávislost, neboť je překládaná do speciálního mezikódu (bytecode), který je na konkrétním počítači nebo zařízení (PC, handheld, mobilní telefon apod.) interpretován, příp. za běhu překládán do nativního kódu (tzv. JIT – Just-In-Time compilerem). Programátor tedy může napsat javovský program například na PC pod Windows a spustit jej na PC s Linuxem, na Macu, SGI, DEC – zkrátka všude, kde je k dispozici prostředí Java runtime. [9]

2 Matchovací nástroje

2.1 Systém Protégé

Protégé je open-source platforma, která nabízí vzrůstající uživatelskou komunitu s řadou nástrojů k tvorbě doménových modelů a vědomostních aplikací založených na ontologiích. V jádře systému Protégé je implementována řada struktur a funkcí pro modelování znalostí, které umožňují vytvářet, zobrazovat a zpracovávat ontologie v různých formátech. Protégé může být přizpůsobeno k poskytnutí přátelské podpory pro vytváření znalostních modelů a zadávání dat. Protégé dále může být rozšířeno několika zásuvnými moduly založenými na Java API (Application Programming Interface) pro budování znalostních nástrojů a aplikací.

Ontologie popisuje koncept a vztahy, které jsou důležité pro jednotlivé obory, poskytující slovník pro dané obory stejně jako automatizovaný předpis významu výrazů použitých ve slovníku. V posledních letech jsou ontologie přijaty mnohými obchodními a vědeckými komunitami jako cesta ke sdílení, opětovnému použití a zpracování oboru znalostí. Ontologie jsou nyní důležitou součástí mnoha aplikací jako jsou vědecké vědomostní portály, systémy pro správu informací, elektronické obchodování a sémantické webové služby.



Obrázek 2-1: Ukázka systému Protégé

Zdroj: vlastní

Platforma Protégé podporuje dva základní přístupy pro modelování ontologií:

- **Protégé-Frames** umožňuje uživateli vytvářet ontologie, které jsou založeny na rámcích (frame-based) v souladu s OKBC (Open Knowledge Base Connectivity protocol). V tomto modelu se ontologie skládá ze série tříd organizovaných do hierarchického zařazení k vyjádření oboru hlavního konceptu, sadou bloků přidružených ke třídám k popisu jejich vlastností a vztahů, sadou instancí těchto tříd – jednotlivé příklady konceptu, který obsahuje přesné hodnoty jejich vlastností.
- **Protégé-OWL** umožňuje uživateli vytvářet ontologie pro sémantický web pomocí OWL (Web Ontology Language) vydaným konsorciem W3C. OWL ontologie mohou zahrnovat popisy tříd, vlastností a jejich instancí. V dané ontologii OWL formálně specifikuje sémantiku jak odvozovat její logické výsledky, tj. fakta nereprezentovaná doslovně v ontologii, ale dokázaná sémantikou. Tyto důkazy mohou být založeny na jednom dokumentu nebo na složených dokumentech, které jsou kombinovány pomocí stanoveného OWL mechanismu. [6]

S aplikací Protégé nelze v základu pracovat jako s nástrojem pro matching ontologií, ale Protégé je navržena jako modulární systém a obsahuje různé nástroje pracující s ontologiemi. Z oficiálních zásuvných modulů pro tento systém lze například uvést modul Prompt.

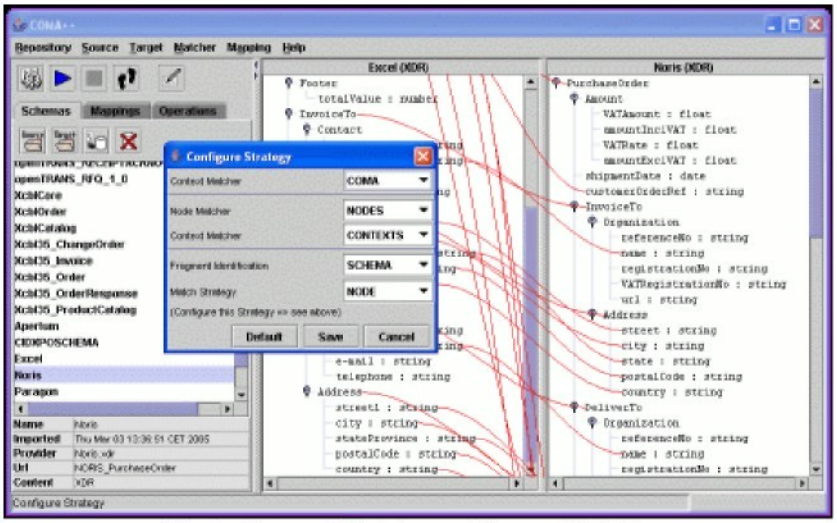
Prompt – zásuvný modul

Modul Promt je kompatibilní se starší verzí systému Protégé a to s verzí 3.1. Tento modul nabízí čtyři funkcionality:

- Porovnání verzí ontologií
- Přesouvání rámců mezi otevřenými ontologiemi
- Propojování ontologií
- Vybírat části ontologií

2.2 COMA++

Matching ontologií a schémat směřuje k rozpoznání shod mezi metadaty systémů nebo modely jako jsou databázová schémata, XML zprávy a ontologie. Řešení jako je „matching problém“ má klíčovou roli při spolupráci služeb a v integraci dat v mnoha aplikacích. Proto je cílem snížit manuální přístup k ontologiím.



Obrázek 2-2: Ukázka aplikace COMA++

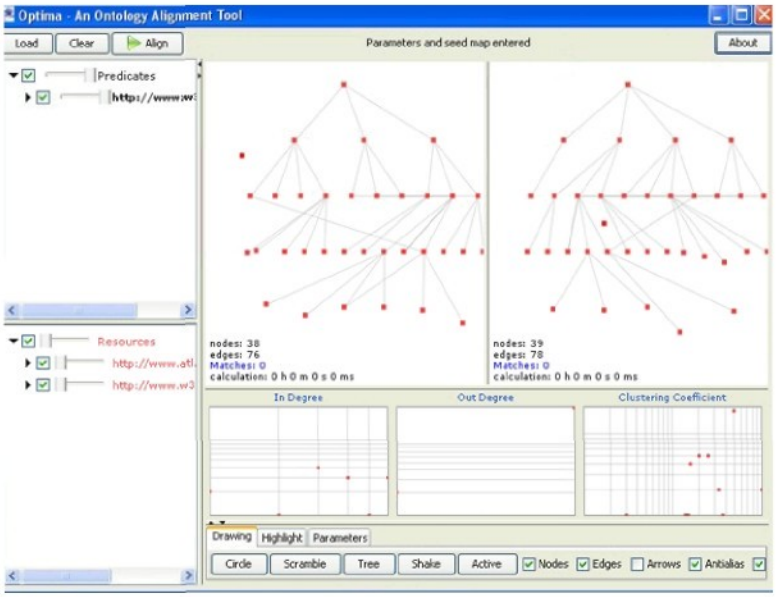
Zdroj: http://dbs.uni-leipzig.de/Research/coma.html

COMA++ (COmbination of Matching Algorithms) je nástroj pro porovnávání ontologií založený na souběžném skládání různých metod. Grafické rozhraní nabízí rozšiřitelné knihovny pro algoritmy porovnávání, lze upravovat nastavení a parametry jednotlivých ontologií k dosažení lepšího výsledku, parametry a nastavení v tomto případě nejsou omezeny pouze hranicí (threshold) nebo jednou metodou porovnání slov, ale také jinými (např.: úprava výsledků jednotlivých metod). [5]

2.3 Optima

Optima, je univerzální nástroj, jehož cílem je porovnávání ontologií, které automaticky identifikuje a porovná příslušné pojmy ontologie. Tento nástroj obsahuje intuitivní uživatelské rozhraní, které usnadňuje vizualizaci a analýzu ontologií v RDF a OWL a alignment výsledcích.

Alignment ontologií je proces určování korespondence mezi pojmy dvou zadaných ontologiích.

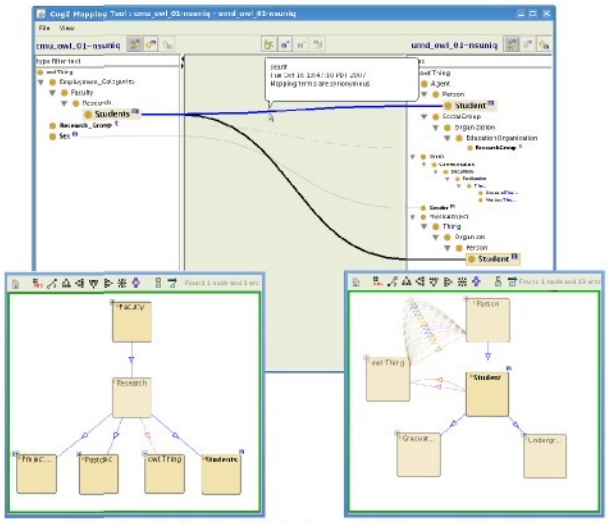


Obrázek 2-3: Ukázka aplikace Optima

Zdroj: http://lsdis.cs.uga.edu/projects/sensormap/index.php?page=5

2.4 CogZ

CogZ (Cognitive Support and Visualization for Human-Guided Mapping Systems) je projekt zkoumající uživatelskou podporu při mapování ontologie. CogZ je jednou ze softwarových komponent pro modul Prompt. Tato komponenta řeší konkrétní problémy znalostní podpory, které byly pozorovány pomocí uživatelských studií. CogZ poskytuje lepší podporu pro tyto výsledky, které tvoří různé znalostní pomůcky a vizualizační techniky.



Obrázek 2-4: Ukázka aplikace CogZ

Zdroj: http://www.thechiselgroup.com/cogz

3 Ontologický jazyk OWL

V rámci evropské výzkumné skupiny byl navržen modelovací ontologický jazyk OIL, zatímco americká výzkumná skupina zpracovala jazyk DAML-ONT. Jako sjednocení této snahy vznikl jazyk DAML+OIL, který sloužil jako výchozí bod pro pracovní skupinu W3C Web Ontology pro přípravu ontologického jazyka OWL na rozšíření vyjadřovací síly jazyka RDF a RDFS. Jazyk OWL má být základním ontologickým jazykem sémantického webu.

3.1 Kompatibilita OWL s RDF a RDFS

Jazyk OWL měl být čistým rozšířením RDFS. Tento jazyk měl využívat vyjadřovací prostředky RDF a RDFS (třídy a vlastnosti) a rozšířit je vlastními výrazy. Při vývoji jazyka bylo ovšem nutné hledat kompromis mezi vyjadřovací silou a efektivním odvozováním. RDFS obsahuje výrazy s velmi vysokou vyjadřovací silou, které by při rozšíření o možnosti jazyka OWL přinášely nepřijatelné výpočetní vlastnosti při odvozování.

3.2 Druhy jazyka OWL

Aby jazyk OWL vyhovoval všem kladeným požadavků a zároveň překonal výše uvedená výpočetní omezení, byl tento jazyk navržen ve třech variantách. Nejvyšší varianta nabízí nejvyšší vyjadřovací sílu a následující dvě varianty jsou vždy podmnožinou vyšší varianty.

- **OWL Full**

OWL Full je plnou variantou jazyka OWL, která používá všechny výrazy a konstrukty jazyka OWL a umožňuje je libovolně kombinovat s výrazy RDF a RDFS. Umožňuje také změnit význam výrazů OWL a RDF aplikováním výrazů jazyka navzájem. Výhodou této varianty je plná zpětná sémantická i syntaktická kompatibilita s RDF. Každý RDF dokument je také dokumentem jazyka OWL Full a každý závěr na základě jazyka RDF je také platným závěrem v jazyce OWL. Na druhé straně složitost jazyka vede k nemožnosti úplné výpočetní podpory pro odvozování a vysoké složitosti zpracování jazyka.

- **OWL DL**

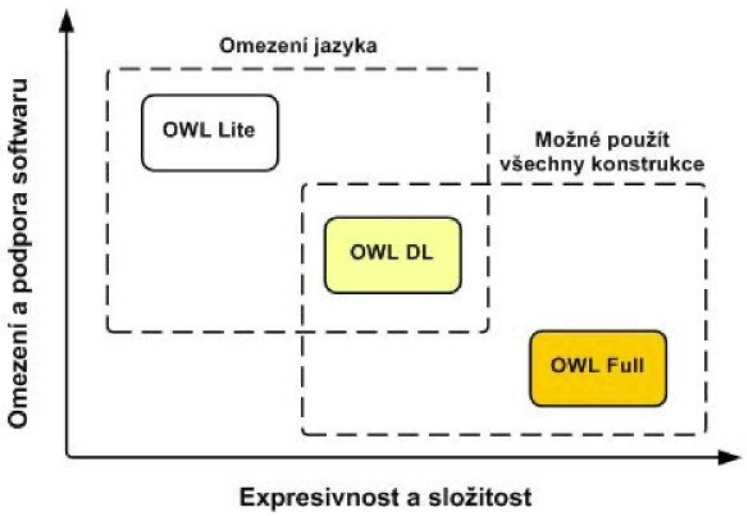
Tato verze je kompromisem mezi výpočetní výkonností a vyjadřovací silou. V této variantě není možné aplikovat výrazy jazyka navzájem, což zajišťuje, že jazyk odpovídá standardům deskripční logiky (odtud zkratka DL). Výhoda efektivního zpracovávání jazyka a dobré výpočetní podpory je vyvážena ztrátou plné kompatibility s RDF a RDFS. Zatímco každý platný OWL DL dokument je také platným dokumentem RDF, opačně toto neplatí.

- **OWL Lite**

OWL Lite je podmnožinou jazyka OWL DL. Na jazyk byla aplikována další omezení, která snižují vyjadřovací sílu. Jazyk ovšem zjednodušují a přináší snazší a efektivnější zpracovávání. Tento jazyk by měl být snadno přijatelný pro uživatele i výrobce softwarových nástrojů.

Jakou verzi vybrat

Záleží na tom, jak složitou ontologii chceme vytvořit. Jestliže budeme chtít větší expresivnost (tzn. vyjadřovací sílu, bohatost jazyka), musíme brát ohled i na dostupný software. Větší vyjadřovací síla vyžaduje schopnější, resp. komplexnější programové vybavení. Když chceme použít všechny OWL konstrukce – OWL Full a OWL DL je dobrou volbou – pro odvozování spíše OWL DL. Při požadavku využít všech konstrukcí, ale s tolerancí určitých omezení je vhodné OWL DL. [13]



Obrázek 3-1: Dialekty jazyka OWL

Zdroj: http://lide.uhk.cz/fim/ucitel/fshusam2/lekarnicky/zt1/zt1_kap04.html

3.3 Charakteristika jazyka OWL

OWL dokumenty se nazývají OWL ontologie. Vzhledem k tomu, že OWL vychází z jazyka RDF jsou OWL ontologie platnými RDF dokumenty. Základním elementem je element *rdf:RDF*, který zapouzdřuje ostatní RDF a OWL obsahy. Vlastní OWL ontologii představuje element *owl:Ontology*, který obsahuje popis OWL ontologie, popis verze a seznam vkládaných ontologií.

Základní třída *owl:class* je podtřídou *rdfs:class* a třídy vlastností *owl:ObjectProperty* a *owl:DatatypePropery* jsou podtřídou *rdf:property*. Mezi třídami je možné definovat disjunktnost pomocí elementu *owl:disjointWith* a rovnost pomocí elementu *owl:equivalentClass*. Kromě toho jazyk OWL definuje nejobecnější třídu *owl:Thing* a prázdnou třídu *owl:Nothing*. Je možné definovat tranzitivní vlastnosti pomocí *owl:TransitivityProperty* (vlastnosti jako např. *větší než*), symetrické vlastnosti (např. vlastnost *sourozenec*) pomocí elementu *owl:SymetricProperty* či inverzní vlastnosti pomocí elementu *owl:InverseFunctionalProperty*. Dále OWL nabízí podporu dalších vyjadřovacích možností. Je možné definovat třídu jako logickou kombinaci (sjednocení, nebo doplněk) jiných tříd či deklarovat výčtové třídy atd. [10]

4 Metody matchingu ontologií

Cílem matchingu ontologií je najít vztahy mezi objekty vyjádřenými v různých ontologiích. Prostřednictvím odhadu podobnosti mezi objekty ontologií lze nalézt různé vztahy jakou jsou zobecnění, upřesnění, ekvivalence, slovní podobnost a jiné.

4.1 Matching problém

V systémech, které jsou distribuované a volně rozšiřitelné, jako je sémantický web, se nelze vyhnout heterogenitě dat. Různí lidé pracující s ontologiemi mají různé zájmy a zvyky, používají jiné nástroje, mají jiné znalosti, různé formy heterogenity, a proto je zapotřebí k těmto faktům přihlédnout.

4.2 Typy heterogenity

Cílem matchingu ontologií je omezit heterogenitu mezi ontologiemi. Existuje mnoho typů, hlavní jsou:

- **Syntaktická heterogenita** nastane, když dvě ontologie nejsou vyjádřeny ve stejném jazyce. Toto se stává, když jsou dvě ontologie modelovány pomocí různých znalostních reprezentačních formalismů (např. OWL a F-Logic). Tento typ neshody je obecně řešen na teoretické úrovni, zavedením shod mezi vztahy různých jazyků. Takto je někdy možné převádět ontologie mezi různými ontologickými jazyky při zachování významu.

- **Terminologická heterogenita** nastane kvůli odchylkám v názvech, které odkazují na stejný objekt v různých ontologiích. To může být způsobeno například použitím různých přirozených jazyků (např. *Papír* - *Paper*), jiným technickým jazykem (např. *Popis – Poznámka*) nebo použitím synonym (např. *Článek – Stať*).

- **Heterogenita pojmů**, také nazývaná sémantická heterogenita nebo logická záměna, je založena na rozdílech v modelování na stejné oblasti zájmu. K tomu může dojít v důsledku použití zcela odlišných pojmů, např. geometrie popsaná bodem jako základním objektem, nebo geometrie popsaná koulí jako základním objektem. V tom je rozdíl mezi záměnou konceptu, která se opírá o rozdíl mezi

pojmy vzoru a zřetelnou neshodou, která se spoléhá na to, jak jsou tyto pojmy vyjádřeny. Existují tři důvody pro neshodu pojmů:

1. *Rozdíl v rozsahu* nastane, když dvě ontologie popisují dvě různé, případně překrývající se oblasti světa stejnou úrovní detailu a jedinečným pohledem. To může nastat například pro dvě částečně se překrývající zeměpisné mapy.

2. *Rozdíl ve spojitosti* nastane, když dvě ontologie popisují stejnou oblast světa ze stejného pohledu, ale v jiné úrovni detailu. Toto může být aplikované na zeměpisné mapy, kdy jedna zobrazuje budovy a druhá města jako body.

3. *Rozdíl v pohledu*, nastane, když dvě ontologie popisují stejnou oblast světa, ve stejném detailu ale z jiného pohledu. Pro mapy může být příkladem účel, kdy jedna mapa zobrazuje krajinu a druhá politické rozložení.

- **Příznaková heterogenita**, také nazývaná účelná heterogenita, se týká toho, jak jsou objekty nazývány lidmi. Ve skutečnosti jsou objekty, které mají stejný sémantický smysl, často interpretovány lidmi s ohledem na kontext například tak, jak jsou nakonec použity. Tento typ heterogenity je pro počítač obtížně zjistitelný a také obtížně řešitelný, protože je mimo jeho úsilí. Účelné použití objektů má velký vliv na jejich interpretaci, tedy přizpůsobování objektů, které nejsou určeny k použití ve stejné souvislosti je často náchylné k chybě.

4.3 Proces matchingu ontologií

Operace matchingu určuje seskupení A' pro dvojici ontologií o a o'. Dále zde mohou být další parametry, které rozšiřují popis matching procesu a to:

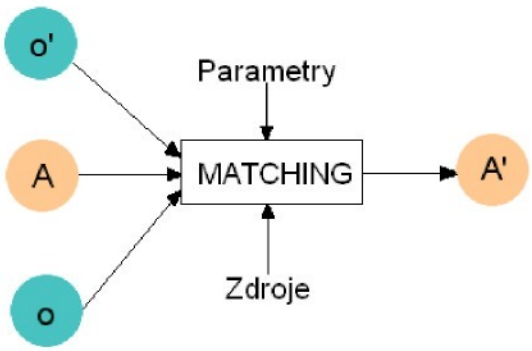
1. použití vstupního seskupení A, které bude dokončeno tímto procesem
2. parametry matchingu p, např. maximální přípustné hodnoty jednotlivých metod
3. externími zdroji r, použitými procesem matchingu, např. běžné znalosti a doménově specifické slovníky

Technicky lze tento proces popsat takto:

Def.: Proces matchingu může být zobrazen jako funkce *f*, která z dvojice ontologií *o* a *o'*, vstupního seskupení *A*, sady parametrů a sadě odhadů a jiných zdrojů vrátí propojení *A'* mezi těmito dvěma ontologiemi:

$$A' = f(o, o', A, p, r)$$

To lze schématicky znázornit takto:



Obrázek 4-1: Proces matchingu

Zdroj: převzato z Euzenat, Jérôme – Shvaiko, Pavel. Ontology Matching.

Tento proces je také vhodný pro posouzení více než dvou ontologií stejnou metodou. Tomuto se říká složený matching.

4.4 Obecná klasifikace metod matchingu ontologií

Základní metodika testování ontologií se dělí na osm typů přístupů, jejichž vztahy jsou naznačeny na obrázku 4-2. Stručný popis těchto metod:

- **Metody založené na řetězci** – Pracují s předponami (resp. příponami) slov, kdy jsou vstupem dva řetězce a kontroluje se, zda první řetězec začíná (resp. končí) druhým řetězcem. Dále je možné určovat počet stejných N-gramů (počet N-tic písmen, které mají dva řetězce společné) či vzdálenost dvou řetězců.

- **Metody založené na zpracování přirozeného jazyka** – Využívají analýzy přirozeného jazyka. *Tokenizace* je rozdělení textu na jednotlivé slovní tvary (tokeny). *Lemmatizace* je analýza tokenů pro zjištění všech základních forem slov. *Eliminací* odstraníme „bezvýznamná“ slova.

- **Lingvistické prostředky** – Zabývají se významem slov, na tomto principu funguje třeba Wordnet.

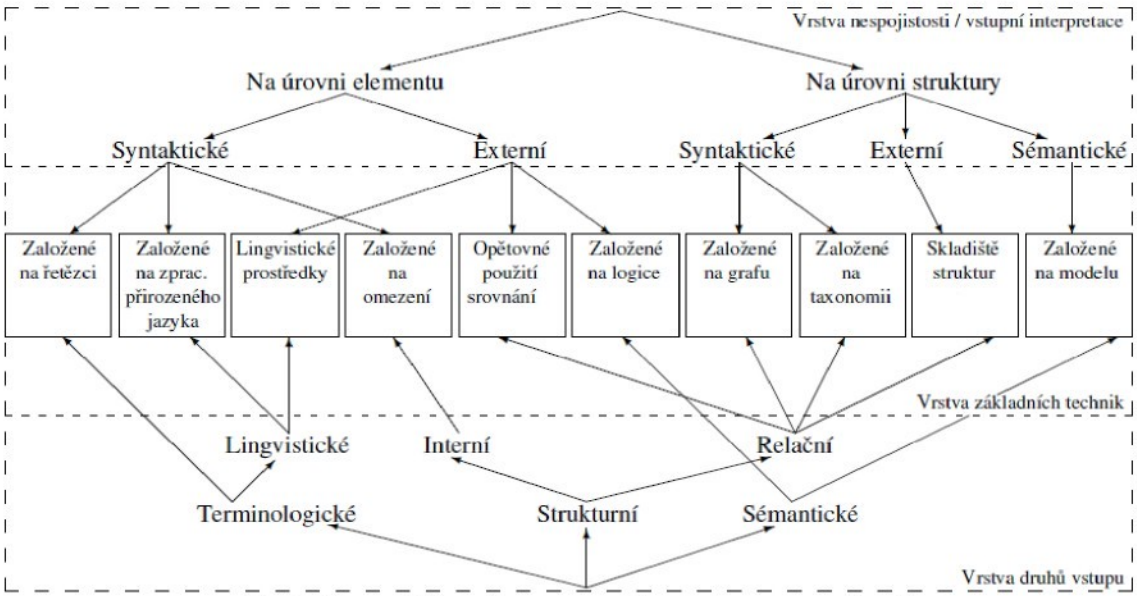
- **Metody založené na omezeních** – Metoda srovnání datových typ°u.

- **Znovupoužití srovnání** – Potřebujeme-li provést srovnání schématu/ontologie O' a O'' a již máme srovnání mezi O a O' a zároveň O a O'', využijeme ho.

- **Metody založené na taxonomii** – Na schémata/ontologie se díváme jako na grafy obsahující termy a vztahy mezi nimi. Například pokud se shodují koncepty vyšší úrovně, aktuální koncepty se podobají.

- **Metody založené na grafu** – Elementy dvou nelistových schémat jsou strukturou podobné, pokud jsou množiny přímých potomků podobné nebo pokud jsou podobné jejich listové množiny, i když množiny jejich přímých potomků nejsou. Jestliže dva uzly dvou schémat/ontologií jsou podobné, jejich sourozenci mohou být rovněž podobní.

- **Metody založené na modelu** – Převedeme srovnání grafu (stromu) na srovnání množiny jeho uzlů. Vytvoříme páry uzlů, které spolu mohou souviset a vztahy mezi nimi zapíšeme výrokovými formulemi. Poté kontrolujeme platnost jednotlivých formulí.



Obrázek 4-2: Klasifikace metod matchingu ontologií

Zdroj: převzato z Euzenat, Jérôme – Shvaiko, Pavel. Ontology Matching.

4.5 Metody založené názvech

Některé metody využívají porovnávání textových řetězců. Mohou být použity pro název, popis nebo komentáře objektů s cílem nalézt podobnost. Tyto metody lze použít např. k porovnání názvů tříd nebo identifikátorů zdrojů.

Hlavním problémem při porovnávání ontologií na základě jejich názvů je výskyt synonym a homonym

- **Synonyma** jsou různá slova použitá k pojmenování stejného objektu. Např. článek a příspěvek v jistém kontextu.
- **Homonyma** jsou slova použitá pro různé objekty. Například slovo „kolej", které ve významech „ubytovna" a „dopravní cesta".

Následkem toho není možné s jistotou odvodit, že dva objekty jsou stejné jméno, nebo že jsou různé, když mají různá jména. Existují však i jiné důvody, kdy toto může nastat:

- Slova jsou v různých jazycích jako jsou angličtina, italština a čeština použitá k pojmenování objektů. Např.: book, libro a kniha.
- Syntaktické varianty stejného slova často nastanou podle různě vhodných způsobů psaní, zkratek, použití volitelných předpon a přípon atd. Například Compact disk, CD, C.D. a CD-ROM mohou být považovány v určitém kontextu za stejné. Nicméně někdy CD může také znamenat Corps Diplomatique nebo také Change Directory.

Tyto druhy odchylek se mohou vyskytovat v rámci jedné ontologie ale také mnohem častěji napříč různými ontologiemi.

4.5.1 Porovnání textových řetězců

Metody pro porovnávání textových řetězců jsou vhodné při porovnávání struktur slov (sekvence znaků). Tyto metody nám z pravidla vyhodnotí podobnost u slov jako „Book" a Textbook, však nikoliv u slov jako „Book" a „Volume".

Normalizace

Před porovnáním řetězců, které mají význam v přirozeném jazyce, je vhodné použít normalizační postupy, které nám pomohou zlepšit výsledek následného porovnání.

- **Normalizace velikosti písmen** se skládá z přeměny každého písmene na malé. Např. z CD se stane cd nebo z SciFi se stane scifi.
- **Potlačení diakritiky** se skládá ze záměny znaků s diakritickým znaménkem za jeho náhradu bez znaménka. Např. nahrazení Montreál za Montreal.
- **Normalizace mezer** se skládá z přeměny všech prázdných znaků jako je mezera, tabelátor, nový řádek atp. do jedné mezery.
- **Odstraňování spojek** spočívá v normalizaci některých znaků mezi slovy, jako jsou odstranění apostrofů a podtržení případné nahrazení za mezeru. Např.: ze spojení peer-reviewed se převede na peer reviewed.
- **Potlačení číslovek** spočívá v nahrazení nebo odstranění číslovek. Např.: slovo kniha23-6 se upraví na kniha.
- **Odstranění interpunkce** eliminuje interpunkční znaménka. Např.: ze slova C.D. Se stane CD.

Tyto normalizační operace se však musí používat opatrně z několika důvodů:

- jsou často jazykově závislé,
- jsou závislé na pořadí,
- mohou mít výsledek v nějaké potlačené informaci,
- mohou změnit význam slov.

Ekvivalence slov

Rovnost slov vrací *0*, pokud jsou textové řetězce odlišné v opačném případě vrátí *1*. Toto lze považovat za hlavní měřítko podobnosti. Ekvivalence slov lze provádět po syntaktické normalizaci.

Hammingova vzdálenost

Ekvivalence slov lze rozšířit pomocí Hammingovy vzdálenosti, která udává počet znaků, ve kterých se slova neshodují.

Test podřetězců

Tento test vyjadřuje podobnost, porovnávají se zde ekvivalence podřetězců, které měří podíl na společných částí slov mezi dvěma řetězci.

N-gramová podobnost

Vyhledávání podobnosti využitím n-gramů se velmi často využívá k porovnávání řetězců. Tato metoda vypočítá počet n-gramů to jest souslednost *n* znaků, vyskytujících se v obou řetězcích. Např.: všechny trigramy pro slovo matching jsou: mat, atc, tch, chi, hin a ing.

Editovací vzdálenost

Intuitivně je editovací vzdálenost mezi dvěma řetězci nejmenší počet operací na jednom řetězci potřebných k tomu, abychom získali druhý řetězec. Editovací vzdálenost byla navržena pro odhalení podobnosti mezi dvěma řetězci, které mohou obsahovat pravopisné chyby.

- *Levenshteinova vzdálenost* určuje minimální počet vložení, smazání a změn znaků.
- *Needleman-Wunchova vzdálenost* vrací editovací vzdálenost s vyšším ohodnocením pro operace vložení a smazání znaku.

Porovnání cest

Rozdíly cest se skládají nejen z porovnávání názvů jednotlivých objektů, ale i ze sledu názvů souvisejících s tímto objektem.

4.5.2 Jazykově založené metody

Doposud jsme považovali textové řetězce jako sekvence znaků. Při posuzování jazykových vlastností zkoumáme text tvořený těmito řetězci. Texty mohou být rozděleny do slov (snadno rozeznatelná posloupnost znaků), které jsou odvozeny od záznamu ve slovníku. Tato slova se neobjevují po hromadě, ale v posloupnosti, která má

gramatickou strukturu.

Lingvistická normalizace

Účelem lingvistické normalizace je redukovat tvary všech slov do nějakého stejného tvaru, který je jednoduše rozpoznatelný. Rozlišujeme tři hlavní typy:

1) morfologický – odlišnost ve tvaru a účelu slova se stejným kořenem
2) syntaktický – odlišnost v gramatickém složení slova
3) sémantický – odlišnost v pohledu na slovo, použití hyperonym a hyponym

Při lingvistické normalizaci se provádí tyto činnosti:

Značkování – spočívá v rozdělování řetězců do sekvence znaků, které rozeznají interpunkci, velikost písmen, prázdné znaky, číslice, atd.

Strojové sestavování slovníků – řetězce vychází ze znaků, které jsou morfologicky analyzovány, za účelem omezení do jejich základních tvarů. Morfologická analýza umožňuje nalézt odvozené a ohnuté tvary kořene slova.

Extrakce slov – všeobecně souvisí s tím, co je nazýváno lingvistický korpus a obsahuje relativně velké množství textu. Nástroje pro extrakci slov rozpoznají slova z opakujících se morfologicky podobných frází v textu a z použití vzorů.

Eliminace zakázaných slov – rozpoznaná slova jako jsou předložky, spojky, atd. jsou označeny k vyškrtnutí, vzhledem k tomu, že nemají žádný slovní význam pro porovnání.

Vnější metody

Vnější lingvistické metody využívají vnějších zdrojů, jako jsou slovníky a lexikony. Mohou být použity následující:

Lexikon – nebo také slovník je sada slov společně s popisem v přirozeném jazyce.

Mnohojazyčné lexikony – jsou lexikony, ve kterých je popis zaměněn za výraz v jiném jazyce.

Sémanticko-syntaktické slovníky – jsou prostředky používané k analýze přirozeného jazyka. Často obsahují nejen záznam názvů, ale také kategorie a seznam parametrů přijatých slovesy nebo přídavnými jmény. Toto je složité na vytvoření a při matchingu ontologií se příliš nevyužívá.

Tezaurus – jedná se o druh lexikonu, ke kterému jsou přidány některé příbuzné informace. Obvykle obsahuje vazby mezi slovy, hyperonyma, hyponyma, synonyma atp.

Terminologie – v tomto případě jde o tezaurus, který obsahuje spíše fráze než samostatná slova. Typicky se používají oborově specifikované a snaží se být méně nepřesné jako slovníky.

Významový slovník WordNet

Při porovnávání názvů elementů je vhodné využít výkladového slovníku. Nejrozšířenějším volně dostupným řešením je slovník WordNet. Jak název napovídá, jedná se o síť slov, spojených sémantickými vztahy. Základními stavebními kameny Wordnetu jsou synonymické řady (*synsety)* tvořené slovy, která mají v určitém kontextu totožný význam. Jednotlivé prvky synsetů se nazývají *literály*. Wordnet dále obsahuje celou řadu sémantických vazeb mezi literály a zejména mezi synsety. Nejvýznamnějším typem jsou hypero-hyponymické vztahy spojující synsety s obecnějším resp. konkrétnějším významem, např. *flanel* je druhem *textilie*. Důležité jsou dále vztahy meronymie, zachycující vztahy mezi částí a celkem, např. *nos* je částí *obličeje*.

Předchozí odstavec byl věnován samotným slovům. Synsety však obsahují i *slovní spojení* (sousloví), např. „vysoká škola". Obecně tedy mluvíme o *jazykových výrazech*. Literály dále obsahují *číselný identifikátor významu*, např. anglické podstatné jméno *bank:1* označuje finanční instituci, *bank:2* - břeh.

První a současně největší (americký) wordnet vznikl a je dále vyvíjen a rozšiřován na Princentonské univerzitě pod vedením George Millera a Christinne Felbaum. V aktuální verzi 2.0 obsahuje více než 100 000 synsetů složených přibližně z dvojnásobku literálů. V roce 1996 odstartoval evropský projekt *EuroWordNet* s cílem vytvořit wordnety pro další jazyky (holandštinu, španělštinu, italštinu, francouzštinu,

němčinu, estonštinu a češtinu) a provázat je do mnohojazyčné databáze.

Pokud přijmeme jisté zjednodušení, můžeme literály v rámci jednoho synsetu považovat za výrazy označující jeden pojem. To je důležité pro spojení Wordnetu s ontologiemi. Wordnet je dnes nejen výchozím zdrojem pro budování nových ontologií, ale také pro „zakotvení" existujících i nově vznikajících ontologií k výrazům přirozeného jazyka (angličtiny a dalších jazyků, pomocí vícejazykových vazeb). [4]

4.5.3 Metody založené na struktuře

Porovnávání názvů objektů a identifikátorů může být nahrazeno nebo rozšířeno o testování struktury objektů, které lze nalézt v ontologiích.

Takové porovnávání může být rozděleno na porovnání vnitřní struktury objektu, jejích vlastností. V případě OWL ontologií vlastnosti, které nabývají hodnot datového typu nebo porovnání objektů, se kterými daný objekt souvisí.

Interní struktura

Tyto metody jsou založeny na vnitřní struktuře objektů a používají taková kritéria jako oblasti jejich vlastností (atributů a vztahů), jejich kardinality nebo násobnosti, tranzitivity nebo symetrie jejich vlastností k výpočtu podobnosti mezi nimi.

Porovnávání vlastností a klíčů

V databázových systémech, na rozdíl od formálních ontologií, jsou tabulky opatřeny klíči (kombinace vlastností, jejichž vlastnosti jednoznačně identifikují objekt). Klíče však mohou být také použity k identifikaci tříd. Navíc schémata mohou používat různé klíče pro stejnou třídu (např. pokud *Produkt* má jako klíč *id* a *Kniha* má jako klíč *isbn*, může být považováno, že se tyto vlastnosti shodují v případě že jsou třídy stejné).

Porovnávání datových typů

Porovnávání vlastností zahrnuje také porovnávání jejich datových typů (v jazyce OWL se může jednat o rozsah vztahů nebo omezení použitých na vlastnosti tříd). Na rozdíl od objektů, které vyžadují interpretaci, datové typy lze považovat za objektivní a je možné určit, jak blízko je jeden datový typ jinému.

Porovnání oboru

Záleží na objektech, které porovnáváme, co lze z vlastností získat (ve třídách to jsou domény, zatímco u instancí to jsou hodnoty). Kromě toho může být rozdělen do řad nebo sekvencí. Proto je důležité při porovnávání brát na tuto skutečnost ohled.

Porovnávání násobností vlastností

Vlastnosti mohou být omezeny násobnostmi (tak jak jsou nazývány v UML). Obdobně k podobnosti mezi datovými typy, podobnost mezi mohutnostmi může být stanovena na základě prohledání tabulky.

V jazyce OWL jsou mohutnosti nebo násobnosti vyjádřeny pomocí minCardinality, maxCardinality a omezením mohutnosti. Násobnosti mohou být vyjádřeny jako interval celých čísel.

4.5.4 Relační struktura

Ontologie může být považována za graf, jehož hrany jsou popsány názvy vztahů (matematicky, mluvou, jde o graf četných vztahů v ontologii). Vyhledávání shod mezi prvky těchto grafů odpovídá tvaru grafu vyjadřujícímu problém stejnotvárnosti.

Porovnání podobnosti mezi dvěma prvky dvou ontologií mohou být založeny na vztahu těchto prvků s jinými prvky těchto ontologií (přímý vztah, tranzitivní omezení, tranzitivní uzávěr, atp.).

Struktura taxonomie

Struktura taxonomie (tzn. graf vytvořený vztahy typu *subClassOf*) je páteří ontologií. Z tohoto důvodu je tato struktura detailně studována a velmi často používána pro porovnávání zdrojů pro matching tříd. Lze prohledávat struktury taxonomií (nadtřídy a podtřídy nebo případně vazby mezi cestami v ontologiích).

Mereologická struktura

Druhá velmi známá struktura po struktuře taxonomie je mereologická struktura (tj. struktura odpovídající vztahu *part-of*). Složitost práce s tímto typem struktury je, že

není snadné zjistit vlastnosti, které s sebou mereologická struktura přináší.

Relace

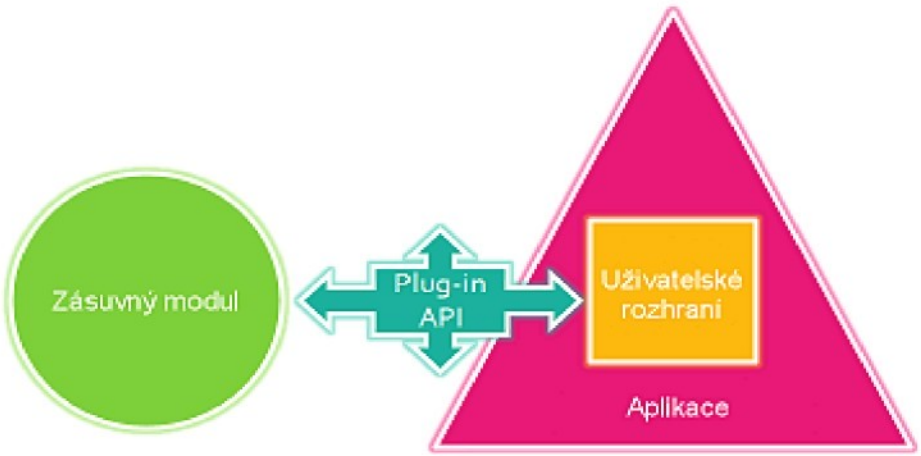
Ve srovnání s předchozími dvěma typy struktur, lze považovat obecný problém matchingu objektů založený na vztazích mezi nimi. Třídy jsou propojeny rovněž přes definice jejich vlastností. Tyto vlastnosti jsou také hrany grafu a lze mezi nimi nalézt podobnost, mohou být použity k hledání takových tříd, které jsou stejné. Nicméně grafy těchto relací mohou obsahovat cykly, tento problém však řeší až pokročilé metody matchingu ontologií. [2]

5 Vytvoření modulu do systému Protégé

V praktické části diplomové práce je cílem vytvořit zásuvný modul pro systém Protégé. Vlastní navržený zásuvný modul je vytvořen pro Protégé verzi 4. Návrh rozmístění komponent byl proveden s využitím grafického návrhového prostředí NetBeans. Ostatní zdrojové kódy jsou tvořeny ve vývojovém prostředí Eclipse. Při tvorbě jsem vycházel z tutoriálu tvorby modulu pro Protégé 4 vytvořeným Nickem Drummondem.

5.1 Tvorba zásuvných modulů v jazyce Java

Hostitelská aplikace, která obsahuje aplikační rozhraní (API), pomocí kterého lze komunikovat se zásuvným modulem, aniž by bylo zapotřebí zasahovat do vlastní hostitelské aplikace. Zásuvné moduly obvykle nejsou schopny pracovat samostatně bez aplikace, pro kterou jsou vytvořeny, ale aplikace není na těchto modulech závislá. Moduly nám jednoduchým způsobem umožňují rozšířit funkčnost aplikace dle aktuálních potřeb a přitom nemusí být zahlcena nepotřebnými funkcemi.



Obrázek 5-1: Schematické znázornění funkce zásuvného modulu

Zdroj: vlastní

5.2 Export do balíčků JAR

V programovacím jazyce Java jsou zásuvné moduly distribuovány pomocí Java archívů (soubory JAR).

Pro sestavování archívů v Javě se používá archivátor *jar*. Archívy plní stejný účel jako knihovny programů, které používají některé programovací jazyky. Java běžně načítá své standardní třídy z archívů. Některé aplikace souborů JAR vyžadují zvláštní soubor v archívu, který se označuje jako *manifest file* a je uložen v adresáři *meta-inf*. V tomto souboru je uveden obsah archívu JAR a jeho atributy. [1]

5.3 Ukázka jednoduchého modulu pro systém Protégé

V tomto příkladu bude názorně popsán jednoduchý modul, jehož cílem je vypsat strukturu vybrané třídy a jejich potomků.

5.3.1 Zdrojový kód

Nejprve je potřeba do projektu přidat knihovny aplikace Protégé, díky kterým lze pomocí API přistupovat k celé aplikaci. V našem vytvářeném zásuvném modulu podědíme třídu od třídy *AbstractOWLClassViewComponent*, která implementuje následující metody:

- *initialiseClassView()* volaná, když je vytvářen vzhled – nejčastěji při vytvoření grafického rozhraní.
- *updateView()* volaná, když je změněn globální výběr třídy – použito k zobrazení nového výběru.
- *disposeView()* volaná, když je rozhraní odebíráno – odebrání listenerů a vyčistění.

Nejužitečnější metodu v pluginu je *getOWLModelManager()*, která nám umožní přistupovat k jádru Protégé frameworku. *OWLModelManager* poskytuje přístup k ontologiím, vyjadřovacím parserům, rendererům, odvozovačům, řízení změn, událostí a serializaci. Uživatelské rozhraní pro přístup k příslušnému frameworku, *getOWLWorkspace()*, vrací GUI zajišťující přístup k výběru modelu, záložkám, panelům s výsledky atd.

Vlastní kód ukázkového modulu lze vidět v příloze A.

5.3.2 Specifikace zásuvného modulu

V druhé části tvorby je třeba popsat zásuvný modul tak, aby systém Protégé tento modul nalezl. Je potřeba vytvořit soubor *plugins.xml* podobný jako je níže. Tento soubor popisuje identifikátor, popis, implementaci Java třídy a nastavení specifické pro zobrazení zásuvného modulu (např. zda je v nabídce nebo jakou barvu má jeho hlavička).

```
<?xml version="1.0" ?>
 <plugin>
   <extension id="org.coode.taxonomy.TabbedHierarchyView"
      point="org.protege.editor.core.application.ViewComponent">
     <label value="Tabbed Subclasses"/>
     <class value="org.coode.taxonomy.TabbedHierarchyView"/>
     <headerColor value="@org.protege.classcolor"/>
     <category value="@org.protege.classcategory"/>
   </extension>
 </plugin>
```

5.3.3 Vytvoření manifestu

Soubor manifest nastaví závislosti mezi moduly (OWL editační nástroje jsou také zásuvný modul a musíme je zahrnout), určuje další cesty. Budeme muset vytvořit soubor MANIFEST.MF v adresáři s názvem META-INF v zásuvném modulu.

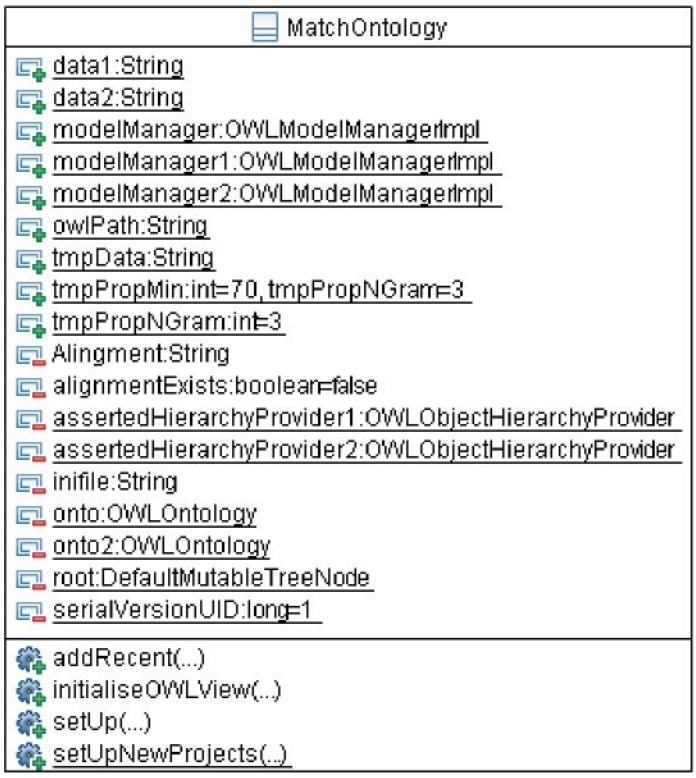
```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Taxonomy Example View
Bundle-SymbolicName: org.coode.taxonomy;singleton:=true
Bundle-Category: protege
Bundle-Description: An example view to demonstrate the plugin
   mechanism for developers
Bundle-Vendor: The CO-ODE Group
Bundle-DocURL: http://www.co-ode.org
Bundle-ClassPath: .
Import-Package: org.osgi.framework,
                org.apache.log4j
Bundle-Version: 1.0.0
Bundle-Activator:
   org.protege.editor.core.plugin.DefaultPluginActivator
Require-Bundle: org.eclipse.equinox.registry,
                org.eclipse.equinox.common,
                org.protege.editor.core.application,
                org.protege.editor.owl,
                org.semanticweb.owl.owlapi
```

5.4 Struktura balíčku

Balíček je strukturován do tří částí. První část je kořenový adresář, ve kterém jsou třídy, které obstarávají komunikaci s vlastní aplikací Protégé popřípadě obstarávají vytvoření výstupního formátu, který určuje propojení dvou ontologií (tzv. *alignment*). Druhou částí balíčku je adresář obsahující třídy obstarávající grafickou reprezentaci tohoto zásuvného modulu. Poslední částí je adresář, ve kterém jsou uloženy třídy, které obsahují metody, porovnávající jednotlivé třídy v obou ontologiích.

5.4.1 Hlavní třída MatchOntology.java

Třída MatchOntology je hlavní třídou celého zásuvného modulu. Třída je děděna od třídy *AbstractOWLViewComponent*, která na rozdíl od té, jenž byla uvedena v příkladu nereaguje na změnu vybrané třídy ontologie. V této třídě bylo potřeba implementovat pouze metody *initialiseOWLView* a *disposeOWLView* (funkčnost viz. Příloha A).



Obrázek 5-2: Class diagram hlavní třídy modulu

Zdroj: vlastní

Tato třída obstarává zapojení modulu do systému Protégé, vytvoří při inicializaci grafické prostředí, nastaví seznam již použitých ontologií, použitých v tomto modulu případně obstará ukládání nově otevřených ontologií do seznamu otevřených ontologií.

V této třídě byl také seznam porovnávacích algoritmů, který se z důvodu jednoduššího přidání nové třídy přidal do třídy Choice.java, ve které jsou přehledně zapsané rutiny, potřebné k porovnání dvou ontologií s využitím dané metody matchingu ontologií. Díky této třídě není třeba po připsání nových metod zasahovat do hlavní třídy, kde by bylo možné udělat chybu, jenž by se hůře hledala.

5.4.2 Výstupní třída Output.java

Třída Output obstarává výstupní propojení, které je odvozeno z uskutečněného testu dvou ontologií. V tomto zásuvném modulu je využit C-OWL Alignment, který je rozšířením formátu OWL (Contextualized OWL), který vyjadřuje zobrazení mezi heterogenními ontologiemi. Tento formát vyjadřuje zobrazení vztahů mezi třídami resp. individui. Tento formát zavádí tzv. překlenovací pravidla mezi ontologiemi, která určují nalezenou vazbu mezi prvky zdrojové ontologie *o* a cílové ontologie *o'*. Formát C-OWL zavádí čtyři propojovací pravidla pro objekty porovnávaných ontologií:

- Zobecnění – More general
- Upřesnění – More specific
- Ekvivalence – Equivalent
- Disjunkce – Disjoint
- Překrývání – Overlapping

5.4.3 Třídy obstarávající porovnávání ontologií

V části balíčku *tests* jsou uloženy třídy, které provádí testování jednotlivých prvků předaných dvou ontologií. Pro zjednodušení je zde vytvořena třída *DefaultTest*, která obsahuje metody a inicializované parametry, které musí obsahovat každý test nad těmito ontologiemi. Jedná se především o metody, obstarávající komunikaci s grafickým rozhraním zásuvného modulu.

Vytvoření nového testu není nyní složité a provede se ve dvou krocích:

1) vytvoření třídy pro testování

```
public class SameNames extends DefaultTest{
     private int numberOfSame,FirstCount,SecondCount;
     private String listOfSame;
     public SameNames (OWLOntology onto,OWLOntology onto1) {
        super();
        numberOfSame = 0;
        listOfSame = "";
        FirstCount = onto.getReferencedClasses().size();
        SecondCount = onto1.getReferencedClasses().size();
        ontoURI=onto.getURI().toString();
        ontoURI1=onto1.getURI().toString();
        for (OWLClass a :onto.getReferencedClasses()){
             for (OWLClass b :onto1.getReferencedClasses()){
                 if(a.toString().equals(b.toString())){
                      numberOfSame++;
                      listOfSame += a.toString()+"\n";
                      set1.add( a.toString()) ;
                      set2.add( b.toString());
                      type.add( "Equivalent");
                      Odds.add("100%");
                      break;
                 }
             }
        }
     }
     public String returnString(){
           return "First Ontology count of classes: "
                  +String.valueOf(FirstCount)+"\n"+
                  "Second Ontology count of classes: "
                  +String.valueOf(SecondCount)+"\n"+
                  "---------------------\n"+
                  "Count of ident names of classes: "+
                   String.valueOf(numberOfSame)+
                  "\n---------------------\n"+
                  listOfSame;
     }
}
```

Jak je vidět v tomto jednoduchém příkladu, veškerá režie potřebná k porovnání se odehrává v konstruktoru dané třídy. Jsou zde vloženy dva cykly, které testují každý prvek první třídy s každým prvkem druhé třídy. Konkrétně tato metoda testuje shodu názvů tříd ontologií. Při nalezení shody se uloží názvy tříd do polí *set1* a *set2*, do pole *type* je uložena textová reprezentace typu vztahu pro C-OWL alignment a jako poslední je nutné přiřadit do pole *Odds* pravděpodobnost shody (v tomto případě je to vždy 100 %). Dále je zde potřeba implementovat metodu *returnString* obstarávající pouze

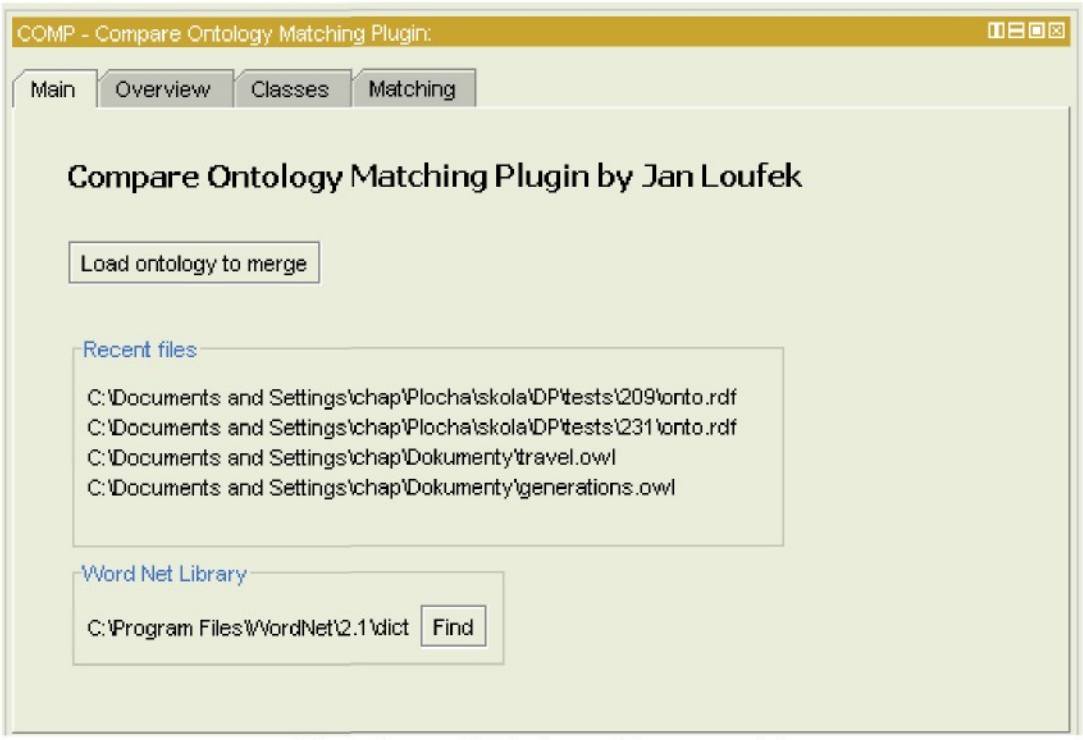
textový výpis protokolu o provedení metody (tato metoda nemá funkční význam, tedy může vracet i prázdný řetězec).

2) zavedení této testovací metody do grafického rozhraní

Zavedení nové testovací metody se provede přidáním parametrů do třídy *Choice* v kořenové části balíčku. V první řadě je třeba přidat název, jenž se bude zobrazovat v rolovací nabídce grafického rozhraní, čehož docílíme přidáním prvku do pole *choices* a zapsat rutinní příkazy (viz. následující příklad):

```
case 0: SameNames a = new SameNames(onto, onto2);
    writeResult = a.returnString();  // text, zobrazeny v paticce
    set1 = a.getSet1(); // seznam prvni ontologie
    set2 = a.getSet2(); // seznam druhé ontologie
    Odds = a.getOdds(); // seznam hodnoty spoje
    Alingment = a.getAlingment(); //xml výstup
    alignmentExists = true;
         //informace o existenci Alingment souboru
    break;
```

5.4.4 Grafické rozhraní zásuvného modulu

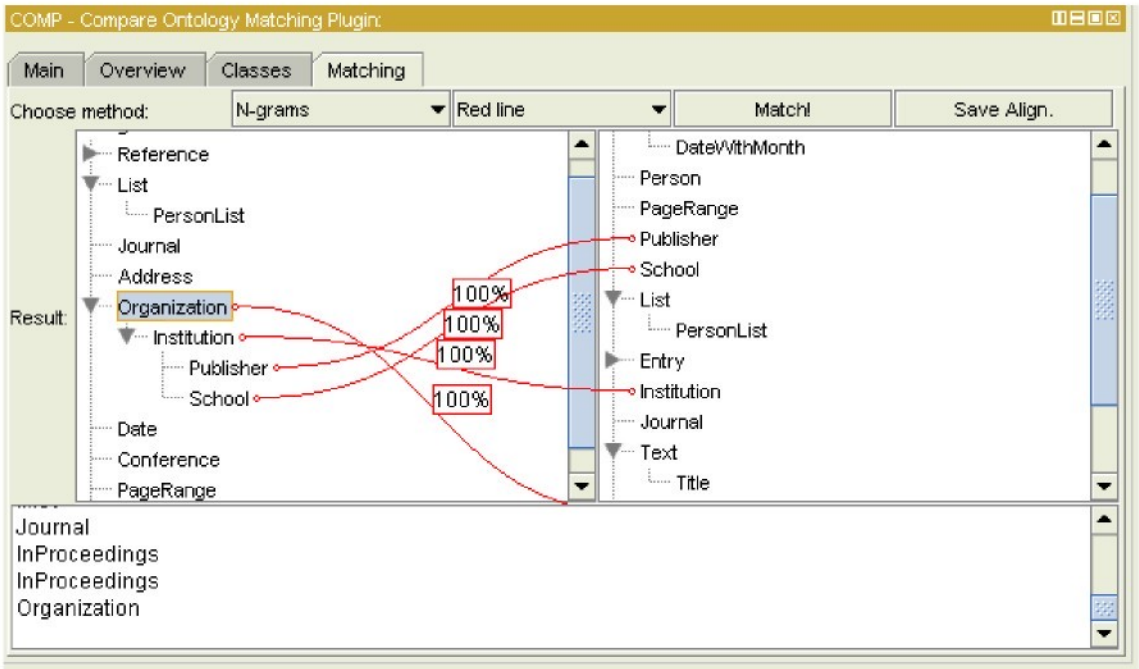


Obrázek 5-3: Vzhled první karty modulu

Zdroj: vlastní

Základ grafického rozhraní je tvořen panelem se čtyřmi kartami. První karta nazvaná „Main“ (viz. Obrázek 5-3) je důležitá pro výběr ontologie, se kterou bude hlavní ontologie (aktivní v rámci spuštěné relace systému Protégé), dále je zde seznam ontologií, s nimiž se v tomto modulu pracovalo a poslední částí je pole, ve kterém nastavíme umístění slovníku Wordnet, sloužící pro porovnávání názvů ze sémantického hlediska.

Na druhé kartě nazvané „Overview“ je obsahem pouze tabulka, ve které jsou vypisovány obecné informace o ontologiích, které se budou pomocí tohoto modulu porovnávat. V této verzi modulu jsou zde k dispozici informace: Název, počet tříd a URI.



Obrázek 5-4: Vzhled čtvrté karty modulu

Zdroj: vlastní

Třetí karta pouze textově vypíše seznam všech tříd. Jejich struktura je naznačena odsazením pomocí mezer.

Čtvrtá a nejdůležitější karta nese název „Matching“ a obsahuje výběrové menu s metodami, které lze aplikovat nad ontologiemi, dále výběrové menu, kterým se určí zobrazení výsledku. Dále se zde nachází tlačítko pomocí kterého lze uložit výsledek posledního testu (C-OWL Alignment). Pod těmito ovládacími prvky se nachází komponenta, na které je vizuálně reprezentován výsledek. Tato komponenta je složena

ze dvou komponent *JTree* a v metodě pro vykreslování jsou zde doprogramovány propojující křivky, znázorňující propojení prvků dvou porovnávaných ontologií. Této komponentě byla věnována vysoká pozornost a je inspirována podobným řešením v aplikaci COMA++ (viz. Obrázek 2-2 a Obrázek 5-4). Pod touto komponentou se nachází textové pole, které vypisuje záznam o výsledku proběhlého testu.

5.5 Práce s třídami ontologií v grafické reprezentaci modulu

Při vytváření především grafické reprezentace výstupu tohoto modulu bylo zapotřebí několikrát částečně předělat komponentu zobrazující hierarchii tříd obou ontologií. Při tvorbě modulu bylo vycházeno z ukázkového modulu, proto nejprve bylo zapotřebí, aby celý strom tříd byl rozvinut, jelikož se při počítání řádků používalo rekurzivní procházení stromu tříd. Z důvodu nízké přehlednosti bylo potřeba tento přístup předělat. Za předpokladu unikátnosti názvů tříd v rámci dané ontologie se po výběru ontologie k porovnání naplní dané stromy hierarchií objektů a od té chvíle se s nimi pracuje jako s objekty na stejné úrovni. Po otestování podobnosti těchto ontologií jsou navráceny tři seznamy (seznamy shodných tříd v obou ontologiích a hodnota shody pro danou metodu), které jsou vloženy spolu s vybranou barvou do této komponenty. Zde jsou při založení, rozbalování a sbalování objektů přepočítávány hodnoty řádků, na kterých se vyskytují třídy, jenž se spojí. Při ostatních událostech (nezasahujících to zobrazené hierarchie), je spouštěno pouze překreslení těchto tříd, pro které je vypočítána zobrazená pozice.

Pro vykreslení křivky je použit objekt *CubicCurve2D* , dále do středu této křivky je vepsána hodnota shody mezi spojenými třídami. Při vybrání konkrétní třídy v jedné z vybraných ontologií jsou zobrazeny pouze takové shody, které souvisí s vybranou třídou popřípadě jejími podtřídami.

5.6 Testování pomocí vhodných ontologií

Pro testování zásuvného modulu byla použita testovací knihovna ontologií. [http://oaei.ontologymatching.org] Knihovna se skládá z referenční ontologie, vůči které se provádí ostatní testy dle různých kritérií. Referenční ontologie vychází ze subjektivního pohledu na to jak má vypadat bibliografická ontologie. Existuje samozřejmě mnoho různých klasifikací publikací, založených například na obsahu

a kvalitě. Tato zde zvolená je založena na kategoriích publikací. Tato referenční ontologie se porovnává s různými variantami této ontologie.

Systematická zkušební testovací sada je postavena na jedné referenční ontologii a mnoha jejích variantách. Tyto ontologie jsou popsány jazykem OWL-DL a jsou serializovány do RDF/XML formátu. Referenční ontologie obsahuje 33 pojmenovaných tříd, 24 vlastností objektů, 40 datotypových vlastností, 56 pojmenovaných instancí a 20 anonymních instancí. Varianty jsou zaměřeny spíše na charakterizaci chování různých nástrojů, než na problémy reálného života. Jsou uspořádány do tří skupin:

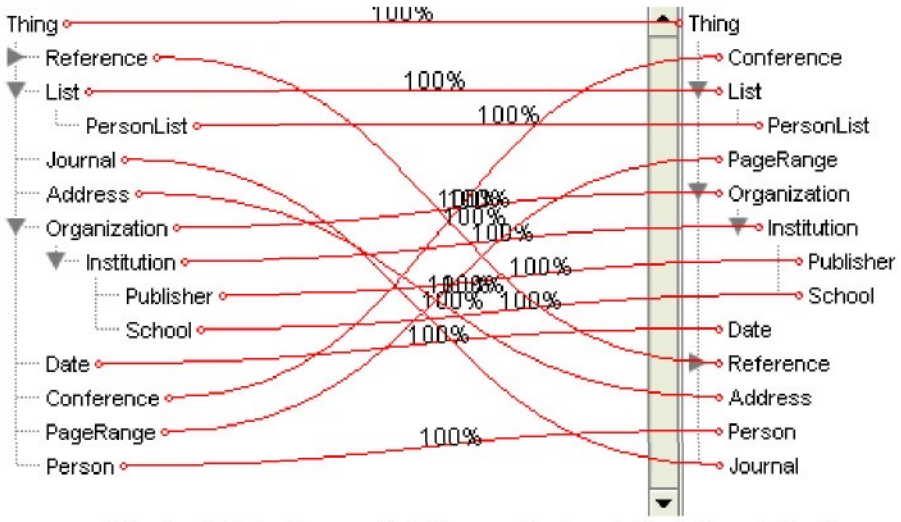
Jednoduché testy (1xx), jako jsou porovnávání referenční ontologie se sebou samotnou , s jinou irelevantní ontologií (ontologie *wine* používaná jako OWL ukázka) nebo stejnou ontologií s jinými omezeními OWL-Lite.

Systematické testy (2xx), získané vyjmutím některých prvků z referenční ontologie. Tyto ontologie testují, jak se algoritmus chová, pokud určitá část informací chybí. Ontologie v této sekci jsou připraveny k testování v šesti kategoriích:

- Názvy – názvy objektů mohou být nahrazeny náhodnými slovy, synonymy, jinými pravidly nebo slovy v jiném jazyce než v angličtině.
- Komentáře – komentáře mohou být odstraněné nebo přeložené do jiného jazyka.
- Hierarchie – může být odstraněna, rozšířena nebo omezena.
- Individua – mohou být odstraněna.
- Vlastnosti – mohou být odstraněny nebo omezeny nad smazanými třídami.
- Třídy – mohou být rozšířeny, tj. rozšířeny několika třídami nebo omezeny.

Čtyři ontologie vycházející z reálného života obsahující bibliografické odkazy (3xx) nalezené na webu a ponechané téměř nedotčené (byly přidány pouze atributy *xmlns* a *xml:base*).

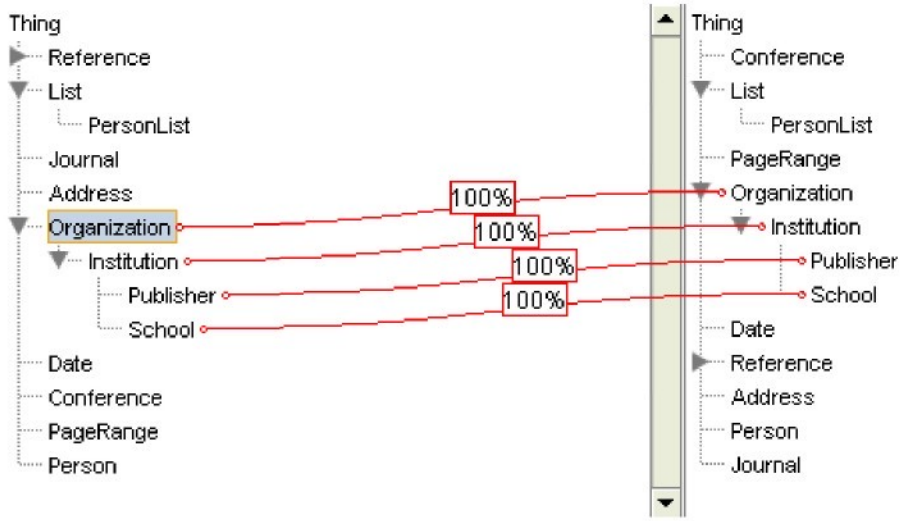
Vzhledem k tomu, že cílem těchto zkoušek je nabídnout nějaké trvalé měřítko, které má být používáno v mnoha aplikacích, byla testovací sada rozšířena v roce 2004 organizací EON Ontology Alignment Contest, jež (téměř) plně zachovává číslování těchto testů.



Obrázek 5-5: Porovnání dvou velmi podobných ontologií

Zdroj: vlastní

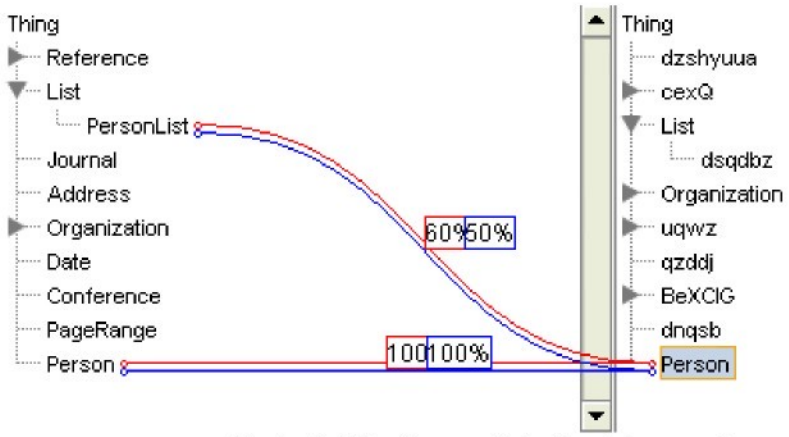
V rámci vytvořeného zásuvného modulu je implementováno několik základních metod porovnávajících především porovnávání textových řetězců z hlediska slovní podobnosti názvů tříd v ontologii. Z tohoto důvodu bylo provedeno testování nad takovými ontologiemi z testovací knihovny, které jsou pro tento problém připraveny. V níže zobrazených výsledcích lze názorně zobrazit výsledek jedné z matchovacích úloh, kde se shodují jménem všechny ontologické třídy (viz. Obrázek 5-5). Pro zvýšení přehlednosti lze vybrat nějakou třídu a zobrazit nalezené vztahy pouze pro ní a její potomky. Také je možné části stromové hierarchie různě skrýt, pokud nás zajímá výsledek konkrétní části (viz. Obrázek 5-6).



Obrázek 5-6: Zobrazení shod pro určitou třídu a její podtřídy

Zdroj: vlastní

V dalším příkladu (viz. Obrázek 5-7) je porovnávána referenční ontologie s ontologií obsahující mimo jiné náhodné názvy tříd a je zde názorně zobrazeno porovnání výsledku různých metod. Konkrétně v tomto příkladu zobrazuje modrá křivka podobnost podle trigramového porovnání a červená křivka Levenshteinovu vzdálenost.



Obrázek 5-7: Porovnání různých metod

Zdroj: vlastní

5.7 Použité standardní Java knihovny

Pro grafickou reprezentaci zásuvného modulu byl použit balíček komponent Swing. Swing je knihovna uživatelských prvků na platformě Java pro ovládání počítače pomocí grafického rozhraní. Knihovna Swing poskytuje aplikační rozhraní pro tvorbu a obsluhu klasického grafického uživatelského rozhraní. Pomocí Swingu je možno vytvářet okna, dialogy, tlačítka, rámečky, rozbalovací seznamy, atd.

K obsluze akcí vyvolaných uživatelským vstupem byla použita knihovna AWT, která je základní knihovnou grafického rozhraní v Javě. Knihovna Swing využívá metod tříd knihovny AWT, která obsahuje komponenty a konstrukce k jejich obsluze nativně obsažené v platformě Java.

Jelikož bylo vhodné ušetřit uživateli práci od často vykonávaných činností (především pro otevírání ontologií k porovnání vytvořený seznam naposledy otevřených souborů, který zjednoduší výběr ontologie k testování a také uložení cesty ke knihovně Wordnet), byla použita třída *Propeties*, která slouží k reprezentaci perzistentní sady vlastností. Vlastnosti pomocí této třídy mohou být uloženy respektive otevřeny pomocí

datového toku (např. souboru) a jednotlivé klíče seznamu a jejich vlastnosti jsou reprezentovány jako datový typ *string*. Tyto informace jsou použitím této knihovny uloženy do samostatného konfiguračního *ini* souboru umístěného v adresáři konfigurací systému Protégé.

5.8 Rozhraní pro práci s ontologiemi OWL-API

OWL-API je Java rozhraní a implementace pro ontologický jazyk OWL specifikovaný konsorciem W3C. Poslední verze tohoto API je zaměřena na OWL-2, které zahrnuje OWL-Lite, OWL-DL a některé prvky OWL-Full. OWL-API má volně dostupný kód šířený pod licencí LGPL. OWL-API obsahuje následující komponenty:

- Aplikační rozhraní pro implementaci funkčních OWL-2 vztahů v paměti.
- RDF/XML parser a popisovač
- OWL/XML parser a popisovač
- OWL parser a popisovač funkcionální syntaxe
- Podpora pro integraci s odvozovači, jako jsou Pellet a FACT++
- Podpora pro ladění

OWL-API je předně vyvíjeno na University of Manchester, ale má také významné přispěvatele z různých skupin a společností. Toto rozhraní je také základním stavebním kamenem systému Protégé.

5.8.1 Použití částí API v zásuvném modulu

Základ OWL API poskytuje datové struktury reprezentující OWL ontologie, třídy pro obsluhu (vytvoření, manipulaci, syntaktickou analýzu, vyjádření a odvozování). Základní datová struktura reprezentuje objekty v ontologii a přibližně odpovídá abstraktní syntaxi OWL.

OWLOntology

Třída *OWLOntology* reprezentuje ontologii. Poskytuje nám jednoznačné souvislosti, dle kterých získáme informace o třídách a vlastnostech.

OWLClass

Třída *OWLClass slouží k* reprezentaci ontologické třídy. Samotná třída je relativně odlehčený objekt (třída neobsahuje informace o definicích, které se na ní mají použít). Axiomy týkající se třídy jsou obsaženy v objektu *OWLOntology*, protože se netýkají jen dané třídy, ale třídy v rámci celé ontologie.

OWLProperty

OWL rozlišuje dva typy vlastností, prvním jsou vlastnosti objektu (týkají se instancí) a druhým jsou vlastnosti dat (týkají se konkrétních hodnot instancí). Tyto třídy jsou zde striktně rozděleny na *OWLObjectProperty* a *OWLDataProperty*.

Správa ontologií

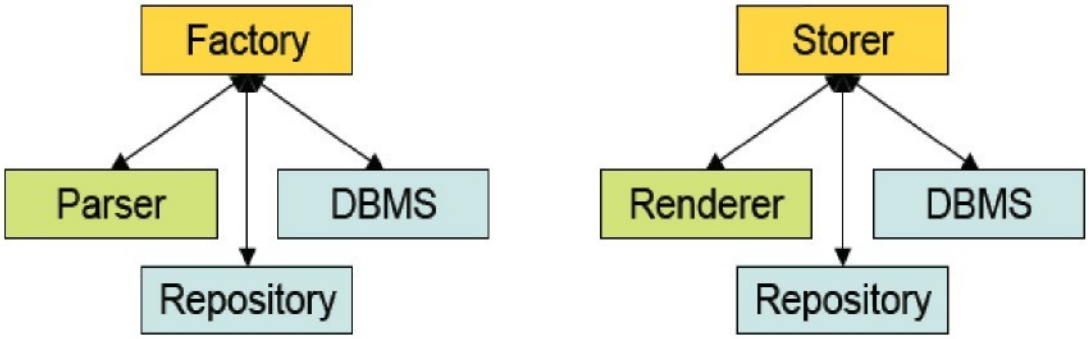
Struktury datového modelu poskytují reprezentaci základních stavebních prvků. Správa a tvorba ontologií je ovládána využitím třídy *OWLOntologyManager* (nahrazující *OWLConnection* a *OWLManager* z původní implementace). Správce je zodpovědný za sledování ontologie, práci s konkrétními formáty ontologií a pracuje se změnami v ontologii.

Mapováni URI

Třída *URIMapper* dává podporu pro oddělení logických a fyzických URI (slouží k oddělení při práci offline, dočasnému ukládání ontologií a řeší problém při identifikaci jména/umístění při importu).

Factory a Storer

Třídy OWLOntologyFactory a OWLOntologyStorer slouží k načtení resp. uložení ontologie. Představují další úroveň abstrakce nad parsery a interpretery. Umožňují také přístup k ontologiím z různých datových zdrojů kromě souborů a streamů, ale také z databází, repositářů atp.



Obrázek 5-8: Schematické znázornění funkce tříd Factory a Storer

Zdroj: http://owlapi.sourceforge.net/SKB-SemTech-OWLAPI-2up.pdf

Formáty ontologií

Třída *OWLOntologyFormat* reprezentuje formát použitý pro konkrétní datový vstup (OWL, RDF/XML). Formát také může obsahovat informace o přesné serializaci (např.: definice jmenného prostoru, řazení, informace o struktuře). Po rozparsování si správce udržuje informace o původním formátu ontologie.

5.9 Použité externí knihovny

Pro jednoduché rozšíření funkčnosti celého zásuvného modulu byly použity také specializované knihovny, které zjednodušili přístup k testování shod v ontologiích. Všechny přídavné balíčky bylo nutné také namapovat do souboru *manifest* .

SecondString

Tento projekt je open-source balíček založený na platformě Java a obsahuje techniky porovnávající textové řetězce z různých pohledů. SecondString je určen především pro výzkumné pracovníky v oblasti informačních integrace a jiné vědecké pracovníky. Obsahuje řadu metod pro vyhledávání shod v řetězcích z různých pohledů včetně statistiky, umělé inteligence, vyhledávání informací a databází. Dále obsahuje nástroje pro systematické hodnocení výkonu na testovacích datech. Není určen pro použití na velké datové soubory. [11]

Z tohoto balíčku lze použít porovnávací metody jako jsou například: Jaro, JaroWinkler, Levenshtein, Jaccard, N-Gram a jiné, popřípadě různé jejich kombinace.

JWI (Java Wordnet Interface)

JWI je balíček vyvíjený organizací MIT (Massachusetts Institute of Technology). Jedná se o knihovnu jednoduchou k použití a lehce rozšířitelnou, která slouží jako rozhraní pro Wordnet. JWI podporuje přístup k databázi Wordnet od verze 1.6 po verzi 3.0. Wordnet je volně a veřejně dostupný sémantický slovník angličtiny.

JWI je psán pro Javu ve verzi 1.5.0 a jmenný prostor balíčku je *edu.mit.jwi*. Balíček neobsahuje soubory knihovny Wordnet. Balíček JWI je šířen pod licencí pro nekomerční použití. [12]

```
public void testDictionary() throws IOException {
// Vytvoření URL ukazující na adresář se slovníkem Wordnet
 String wnhome = System.getenv("WNHOME");
 String path = wnhome + File.separator + "dict";
 URL url = new URL("file", null, path);

 // Vytvoření a otevření objektu se slovníkem
 IDictionary dict = new Dictionary(url);
 dict.open();
 IIndexWord idxWord = dict.getIndexWord("computer", POS.NOUN);
 if (idxWord != null){
   IWordID wordID = idxWord.getWordIDs().get(0); //první nalezený
   IWord word = dict.getWord(wordID);
   ISynset synset = word.getSynset();
   List<ISynsetID> setOfWords = null;
   setOfWords = synset.getRelatedSynsets(Pointer.HYPONYM);
   List<IWord> words;
   for (ISynsetID sid : setOfWords) {
      words = dict.getSynset(sid).getWords();
      System.out.print("{");
      for (Iterator<IWord> i = words.iterator(); i.hasNext();) {
         System.out.print(i.next().getLemma());
         if (i.hasNext()) System.out.print(", ");
      }
      System.out.print("}");
   }
 }
}
```

Výše zobrazený příklad nalezne hyperonyma od slova „computer“ a vypíše je v závorkách podle tzv. synsetů (sady synonym). Konkrétně vypíše:

{analog_computer, analogue_computer} {digital_computer}{home_computer} {node, client, guest} {number_cruncher} {pari-mutuel_machine, totalizer, totaliser, totalizator, totalisator} {predictor} {server, host} {Turing_machine} {web_site, website, internet_site, site}.

Cesta k tomuto slovníku je vybírána na úvodní stránce zásuvného modulu, kde je v dolní části rozhraní tlačítko, které otevře dialog, pro vyhledání souboru a akceptuje pouze soubory s příponou *dict*, obsahující sémantickou databázi slovníku Wordnet. V případě správného nalezení slovníku je tato cesta uložena do konfiguračního souboru pro použití při dalším spuštění aplikace Protégé.

5.10 Možnosti dalšího vývoje

Zásuvný modul byl cíleně navrhován tak, aby se logicky co nejvíce oddělily objekty, které tvoří celý tento zásuvný modul, aby bylo dbáno na propracované rozhraní jednotlivých objektů z důvodu snadnější implementace dalších tříd (především testovacích).

V první řadě je potřeba rozšířit zásuvný modul o širší nabídku testů, popřípadě nalézt kombinace různých metod, které jsou výhodné právě pro matching problém. Dále zohlednit veškeré objekty nalézající se v ontologii a pokusit se vyhledávat veškeré vazby z těchto testů plynoucí.

Takto vytvořený modul také neumí využít všech výhod vycházejících z použití sémantického slovníku Wordnet. V tomto případě je zde vytvořen pro úlohy, ve kterých se přímo pracuje s kombinacemi slov, které slovník popisuje, a nevyužívá kombinací jiných metod (spojování slov, vyhledávání překlepů atp.).

Mezi další cíle, které by mohl takto navržený zásuvný modul umět, je možnost výběru výstupního formátu, který definuje nalezené vztahy mezi objekty v ontologii.

Závěr

Cílem mé diplomové práce bylo seznámit se s možnostmi vytváření zásuvných modulů pro systém Protégé a s metodami matchingu ontologií. Následně navrhnout vlastní zásuvný modul pro tento systém.

Systém Protégé je rozšířený modelovací nástroj pro práci s ontologiemi. Protégé je napsán v jazyce Java a má volně dostupný kód. Protégé v aktuální verzi 4 nabízí modelování pouze s užitím ontologického jazyka OWL vyvíjeným organizací W3C. Systém Protégé je tvořen jako modulární a je tedy možné vytvářet nové doplňky aniž by bylo zapotřebí zasahovat do kódu samotné aplikace, čímž se vyvarujeme fatálních chyb, způsobujících problémy, které lze obtížně nalézt.

Při vytváření grafického rozhraní mi byla velkou inspirací aplikace COMA++, podle které byla vytvořena komponenta, která zobrazuje ve stromové hierarchii dvě ontologie a pomocí křivek propojuje jednotlivé elementy v závislosti na výsledku provedeného testu. Do této komponenty byla přidána možnost zobrazit různě barevné propojovací křivky, pro porovnání výsledků různých metod. V hierarchii těchto ontologií lze různě rozvinout resp. sbalit prvky, dle potřeby zkoumání nalezených shod. V případě více nalezených shod je také možné po vybrání prvku z celé hierarchie zobrazit pouze nalezené shody pro daný prvek a pro prvky, jenž jsou mu podřízeny.

Jako výstupní negrafický formát byl zvolen formát C-OWL, který přímo vychází z návrhu OWL, je však rozšířen o vyjádření vztahů v rámci dvou různých ontologií.

V této práci se podařilo vytvořit modul, který podporuje základní metody pro matching ontologií založené na porovnávání elementů ontologií především z hlediska testování slovních řetězců (vyhledává mezi nimi podobnost).

Takto vytvořený zásuvný modul byl testován na několika ontologiích, určených právě pro testování aplikací pro matching ontologií.

Zdroje

[1] Darvin, Ian F. *JAVA – Kuchařka programátora.* Computer Press, Praha, 2006. ISBN 80-251-0944-5.

[2] Euzenat, Jérôme – Shvaiko, Pavel. *Ontology Matching.* Springer-Verlag, Berlin/Heidelberg, 2007. ISBN 3-540-49611-4.

[3] Petr Matulík, Tomáš Pitner. *Sémantický web* [online]. 2004 [2009-2-20]. Dostupný z WWW: <http://www.ics.muni.cz/zpravodaj/articles/296.html>

[4] Pavel Smrž, Tomáš Pitner. *Ontologie* [online]. 2004 [2009-2-20]. Dostupný z WWW: <http://www.ics.muni.cz/zpravodaj/articles/307.html>

[5] *Ontology matching* [online]. 2009 [2009-2-20]. Dostupný z WWW: <http://www.ontologymatching.org/>

[6] *Protégé* [online]. 2008 [2009-2-20]. Dostupný z WWW: <http://protege.stanford.edu>

[7] *COMA++* [online]. 2008 [2009-3-21]. Dostupný z WWW: <http://dbs.uni-leipzig.de/Research/coma.html>

[8] Pavel Tyl. *Combination of Methods for Ontology Matching* [online]. 2008 [2009-3-21]. Dostupný z WWW: <http://www.cs.cas.cz/hakl/doktorandsky-den/files/2008/dk08proc.pdf>

[9] *Programovací jazyk Java* [online]. 2009 [2009-4-4]. Dostupný z WWW: <http://v1.dione.zcu.cz/java/>

[10] Petr Bartoš. *Ontologické jazyky* [online]. 2009 [2009-4-20]. Dostupný z WWW: <http://sites.google.com/a/globalsemantic.net/gsn/swp/>

[11] *SecondString* [online]. [2009-4-20]. Dostupný z WWW: <http://secondstring.sourceforge.net>

[12] *Java Wordnet Interface* [online]. [2009-4-20]. Dostupný z WWW: <http://projects.csail.mit.edu/jwi>

[13] Martina Husáková. *Znalostní technologie.* [online]. [2009-4-22] Dostupný z WWW: <http://lide.uhk.cz/fim/ucitel/fshusam2/lekarnicky/zt1/zt1_kap04.html>

[14] *Optima* [online]. [2009-5-6]. Dostupný z WWW: <http://lsdis.cs.uga.edu/projects/sensormap/>

Příloha A – Ukázka jednoduchého modulu

```
package org.coode.taxonomy;
import org.protege.editor.owl.ui.view.AbstractOWLClassViewComponent;
import org.protege.editor.owl.ui.renderer.OWLModelManagerEntityRenderer;
import org.protege.editor.owl.model.hierarchy.OWLObjectHierarchyProvider;
import org.semanticweb.owl.model.OWLClass;

import javax.swing.*;
import java.awt.*;

/**
 * Author: Nick Drummond
 * http://www.cs.man.ac.uk/~drummond
 *
 * The University Of Manchester
 * Bio Health Informatics Group
 * Date: Jul 10, 2007
 *
 * Zobrazí strom potomků pro otevřenou třídu
 */


public class TabbedHierarchyView extends AbstractOWLClassViewComponent {

    private JTextArea namesComponent;

    // vhodná třída pro přímé dotazování na hierarchii
    private OWLObjectHierarchyProvider<OWLClass>
assertedHierarchyProvider;

    // poskytuje názvy modelu Tříd/Vlastností/Individuí, zobrazující
    // aktuální stav
    private OWLModelManagerEntityRenderer ren;

    // Vytvoření grafického rozhraní.
    public void initialiseClassView() throws Exception {
        setLayout(new BorderLayout(6, 6));
        // vložíme pouze komponentu JtextArea se ScrollPane
        namesComponent = new JTextArea();
        namesComponent.setTabSize(2);
        add(new JScrollPane(namesComponent), BorderLayout.CENTER);
    }

    // metoda volána automaticky při změně tříd
    protected OWLClass updateView(OWLClass selectedClass) {
        namesComponent.setText("");
        if (selectedClass != null){
            assertedHierarchyProvider =
                 getOWLModelManager().getOWLClassHierarchyProvider();
            ren = getOWLModelManager().getOWLEntityRenderer();
            render(selectedClass, 0);
        }
        return selectedClass;
    }
```

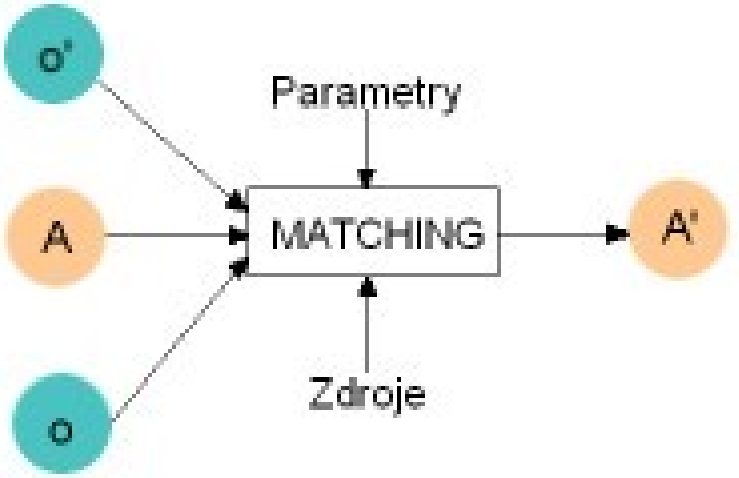
```
    // Vypíše všechny třídy a rekurzivně i její podtřídy
    private void render(OWLClass selectedClass, int indent) {
        for (int i=0; i<indent; i++){
            namesComponent.append("\t");
        }
        namesComponent.append(ren.render(selectedClass));
        namesComponent.append("\n");
        // hierarchy provider nám předá podtřídy
        for (OWLClass sub:
                assertedHierarchyProvider.getChildren(selectedClass)){
            render(sub, indent+1);
        }
    }

    // odebere listenery a provede úklid (zde není potřeba)
    public void disposeView() {
    }
}
```

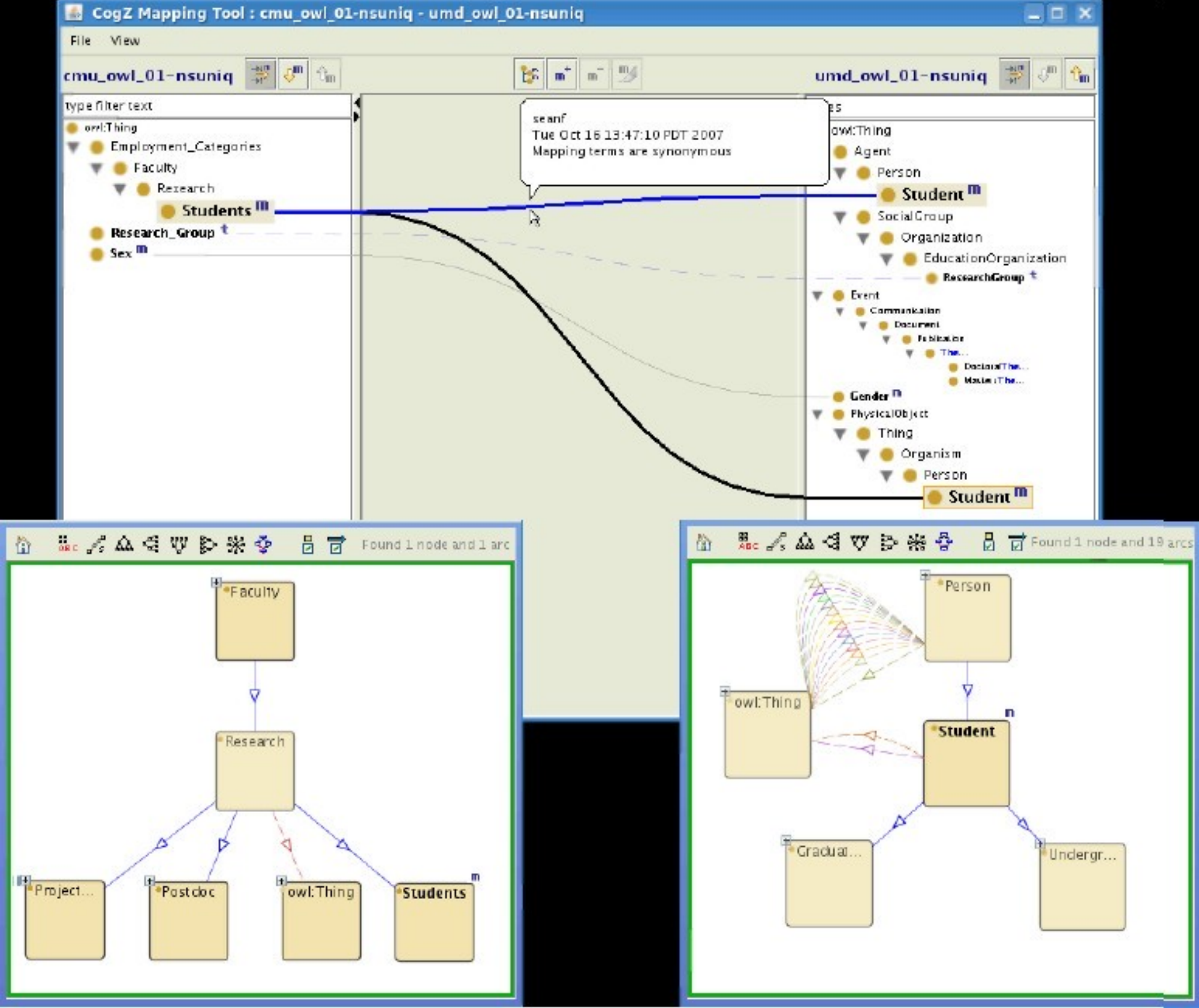
Příloha B – Výstupní C-OWL alignment pro testovací ontologie

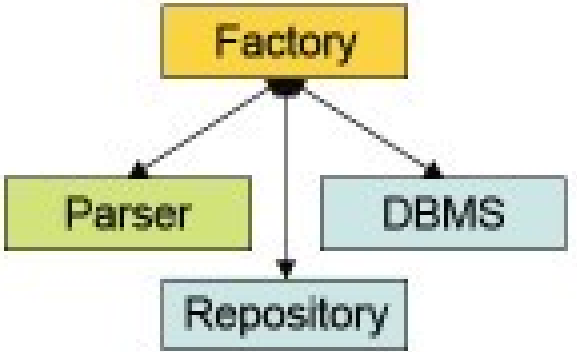
```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
<!ENTITY rdf    "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
<!ENTITY owl    "http://www.w3.org/2002/07/owl#" >
<!ENTITY cowl   "http://www.itc.it/cowl#" >
<!ENTITY onto1
"http://oaei.ontologymatching.org/tests/202/onto.rdf" >
<!ENTITY onto2  "http://oaei.ontologymatching.org/tests/101/onto.rdf" >
 ]>
<rdf:RDF
xmlns           ="&cowl;"
 xmlns:cowl     ="&cowl;"
 xmlns:owl      ="&owl;"
 xmlns:rdf      ="&rdf;"
>
<cowl:Mapping>
<cowl:sourceOntology>
<owl:Class rdf:about="&onto1;"/>
</cowl:sourceOntology>
<cowl:targetOntology>
<owl:Class rdf:about="&onto2;"/>
</cowl:targetOntology>
<cowl:Equivalent>
<cowl:source>
<owl:Ontology rdf:about="&onto1;#Organization"/>
</cowl:source>
<cowl:target>
<owl:Ontology rdf:about="&onto2;#Organization"/>
</cowl:target>
</cowl:Equivalent>
<cowl:Equivalent>
<cowl:source>
<owl:Ontology rdf:about="&onto1;#Person"/>
</cowl:source>
<cowl:target>
<owl:Ontology rdf:about="&onto2;#Person"/>
</cowl:target>
</cowl:Equivalent>
<cowl:Equivalent>
<cowl:source>
<owl:Ontology rdf:about="&onto1;#List"/>
</cowl:source>
<cowl:target>
<owl:Ontology rdf:about="&onto2;#List"/>
</cowl:target>
</cowl:Equivalent>
<cowl:Equivalent>
<cowl:source>
<owl:Ontology rdf:about="&onto1;#Thing"/>
</cowl:source>
<cowl:target>
<owl:Ontology rdf:about="&onto2;#Thing"/>
</cowl:target>
</cowl:Equivalent>
</cowl:Mapping>
</rdf:RDF>
```

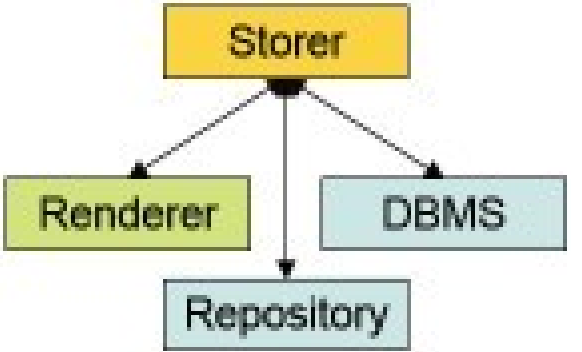
Příloha C – Obsah doprovodného CD

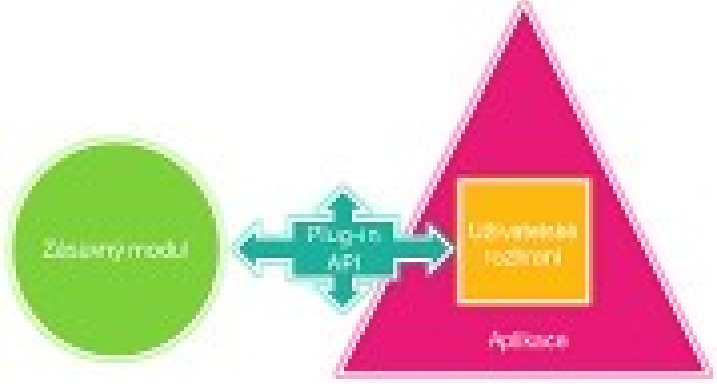
Adresář	Soubor
Doprovodný text DP	Desky.pdf
	Zadání.pdf
	Hlavní text.pdf
Plugin	edu.tul.comp.jar
	(Eclipse projekt se zdrojovými kódy)
Obrázky	(Obrázky použité v DP)
Testovací ontologie	(Testovací ontologie)

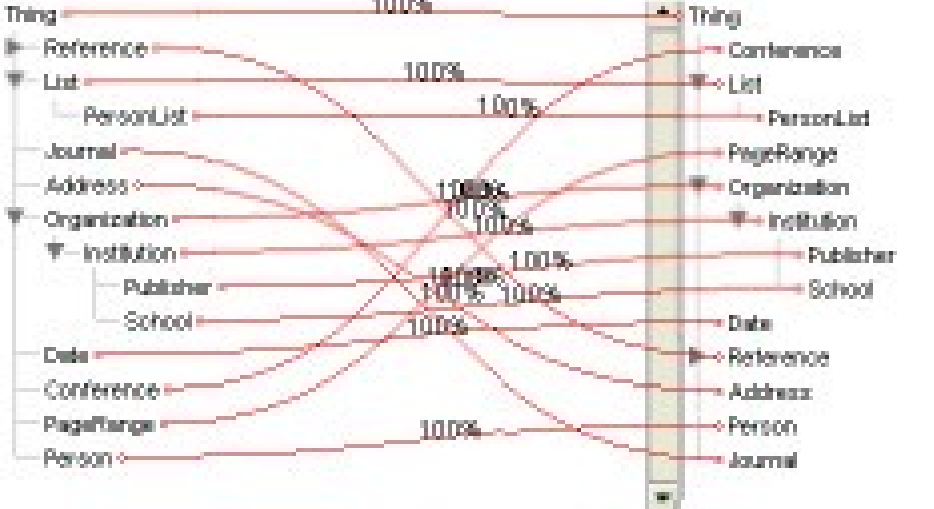

o'
Parametry
A
MATCHING
A'
Zdroje
o

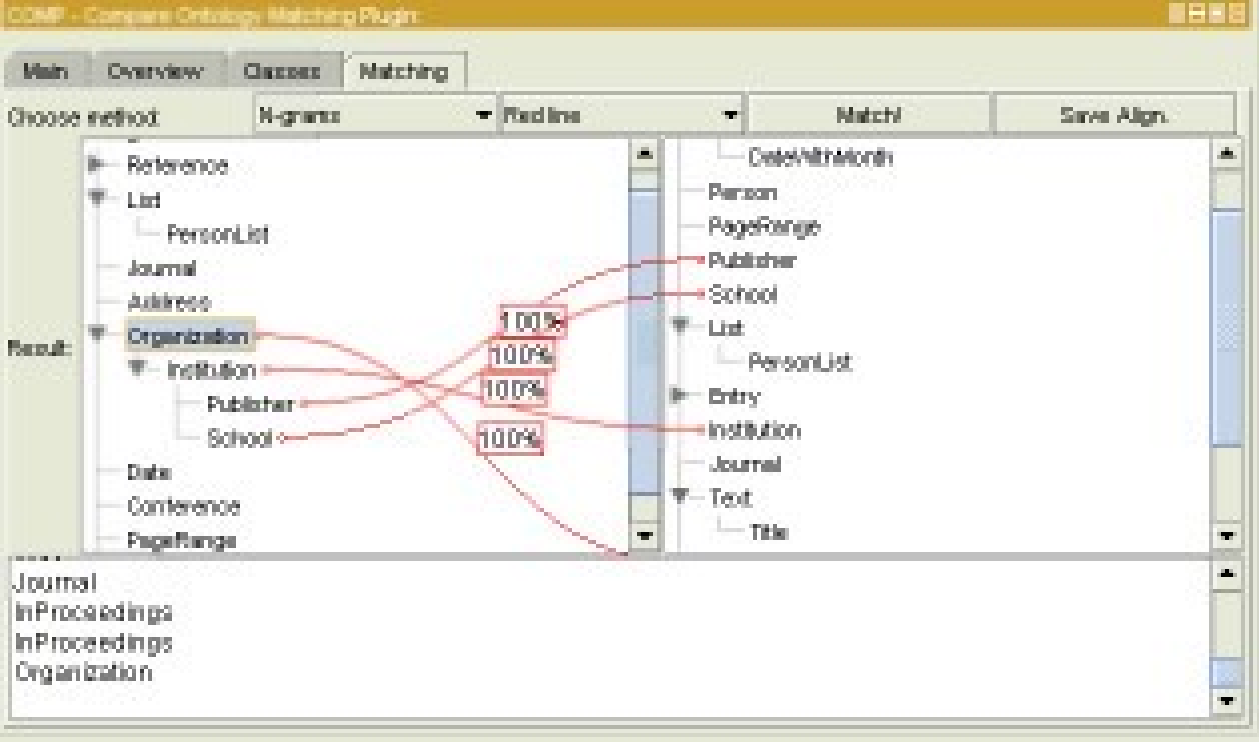

CogZ Mapping Tool : cmu_owl_01-nsuniq - umd_owl_01-nsuniq
File
View
cmu_owl_01-nsuniq
umd_owl_01-nsuniq
type filter text
owl:Thing
Employment_Categories
Faculty
Research
Students
Research_Group
Sex
seanf
Tue Oct 16 13:47:10 PDT 2007
Mapping terms are synonymous
owl:Thing
Agent
Person
Student
SocialGroup
Organization
EducationOrganization
ResearchGroup
Event
Gender
PhysicalObject
Thing
Organism
Person
Student
Found 1 node and 1 arc
Faculty
Research
Project...
Postdoc
owl:Thing
Students
Found 1 node and 19 arcs
Person
owl:Thing
Student
Graduat...
Undergr...

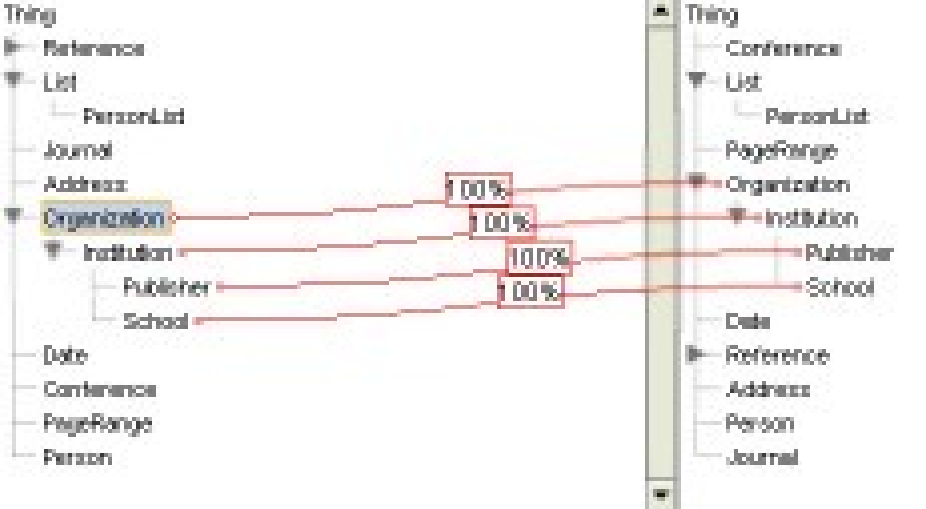

Factory
Parser
DBMS
Repository

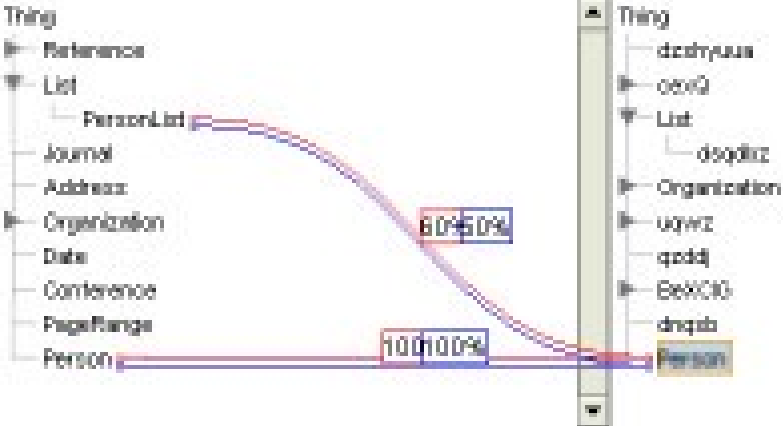

Storer
Renderer
DBMS
Repository


Zásuvný modul
Plug-in API
Uživatelské rozhraní
Aplikace


Thing
Reference
List
PersonList
Journal
Address
Organization
Institution
Publisher
School
Date
Conference
PageRange
Person
100%
Thing
Conference
List
PersonList
PageRange
Organization
Institution
Publisher
School
Date
Reference
Address
Person
Journal


COMP - Compare Ontology Matching Plugin
Main
Overview
Classes
Matching
Choose method:
N-grams
Radius
Match!
Save Align.
Result:
Reference
List
PersonList
Journal
Address
Organization
Institution
Publisher
School
Date
Conference
PageRange
100%
100%
100%
100%
DateWithMonth
Person
PageRange
Publisher
School
List
PersonList
Entry
Institution
Journal
Text
Title
Journal
InProceedings
InProceedings
Organization


Thing
Reference
List
PersonList
Journal
Address
Organization
Institution
Publisher
School
Date
Conference
PageRange
Person
100%
100%
100%
100%
Thing
Conference
List
PersonList
PageRange
Organization
Institution
Publisher
School
Date
Reference
Address
Person
Journal


Thing
Reference
List
PersonList
Journal
Address
Organization
Date
Conference
PageRange
Person
60%
50%
100%
100%
Thing
dzshyuua
ozvQ
List
dsqdkz
Organization
uqvvz
qzdd
Be
dnqsb
Person


COMP - Compare Ontology Matching Plugin
Main
Overview
Classes
Matching
Compare Ontology Matching Plugin by Jan Loufek
Load ontology to merge
Recent files
C:\Documents and Settings\ichap\Plocha\skola\DP\testy\209\onto.rdf
C:\Documents and Settings\ichap\Plocha\skola\DP\testy\231\onto.rdf
C:\Documents and Settings\ichap\Dokumenty\travel.owl
C:\Documents and Settings\ichap\Dokumenty\generations.owl
WordNet Library
C:\Program Files\WordNet\2.1\dict
Find