Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Študijný program: Softvérové inžinierstvo

Bc. Michal Tury

Identifikácia a manažment zmien ontológií

Diplomová práca

Vedúci diplomovej práce: prof. Ing. Mária Bieliková, PhD.
december 2005

Anotácia

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Študijný program: Softvérové inžinierstvo

Autor: Bc. Michal Tury

Diplomová práca: Identifikácia a manažment zmien ontológií

Vedúci projektu: prof. Ing. Mária Bieliková, PhD.

december, 2005

Práca spadá do oblasti webu so sémantikou, ktorý sa vyvíja s cieľom, aby sa stal nástupcom súčasného webu. Hlavnou myšlienkou webu so sémantikou je rozšírenie súčasného webu o metadáta, ktoré opisujú význam obsahu webu. Tieto metadáta sú pritom zapísané tak, aby boli zrozumiteľné stroju a tak aj strojovo spracovateľné. V tomto kontexte sa práca zaoberá problematikou identifikácie zmien a ich manažmentu medzi verziami ontológií. Navrhuje riešenie identifikácie zmien medzi verziami metadát opísaných ontológiami, ktoré sa používajú v technológii webu so sémantikou. Používa pri tom heuristicky na sémantickej aj štrukturálnej úrovni a taktiež sa zaoberá problémom zisťovania sémanticky významných zmien textu prostredníctvom relatívneho porovnávania. Návrh bol overený vytvorením softvérového systému OntoDiff, ktorý je určený na automatickú identifikáciu zmien medzi verziami ontológií. Identifikované zmeny umožňuje analyzovať a exportovať ich do iných systémov.

Annotation

Slovak University of Technology in Bratislava

Faculty of Informatics and Information Technologies

Degree Course: Software Engineering

Author: Bc. Michal Tury

Thesis: Identification and management of ontology changes

Supervisor: prof. Ing. Mária Bieliková, PhD.

2005, December

This thesis deals with the theme, which is included in the thematic field of semantic web. Semantic web is developed with the aim to become the successor of current web technologies. The main idea of semantic web is to extend the current web by metadata, which describe the meaning of information. These metadata are written down in such a way to be understandable to the machine language and simply processable. In this context is the thesis structured and deals with topic of identification of changes and their managing among the ontology versions. The project suggests a solution to changes identification among versions of metadata described by ontologies used in semantic web. There are used the heuristic methods on the semantic and structural level and deals with the topic of searching the significantly important changes of text through the relative compartment as well. The proposal was verified by OntoDiff software system designed for automatic identification of changes among different versions of ontologies. Identified changes can be analyzed and exported to other systems.

Čestne prehlasujem, že som diplomovú prácu vypracoval samostatne,
s použitím uvedenej literatúry.

Bratislava, december 2005

Touto cestou by som sa rád poďakoval pani prof. Márii Bielikovej za jej ochotu, obetavosť a ústretovosť. Kiež by bolo viacej pedagógov ako je ona. Veľká vďaka patrí môjmu otcovi a bratovi, ktorí toho obetovali a spravili nesmierne veľa aby som sa dostal tam, kde som. Taktiež sa musím poďakovať svojim priateľom za pomoc, podporu, ochotu a povzbudenie. Ďakujem, ste skvelí.

Venované tej najlepšej mamine na svete.
Snáď by si bola hrdá.

Obsah

1 Úvod

Množstvo informácií, ktoré sa v súčasnej dobe nachádzajú v hyperpriestore je také obrovské, že si ho vie len málokto prestaviť. Objem týchto informácií sa s príchodom informačnej spoločnosti a ekonomiky založenej na vedomostiach, bude ešte viacej zväčšovať a je len otázkou času, kedy napr. súčasné problémy vyhľadávania informácií na webe a v hyperpriestore všeobecne, budú ešte viac prehlbovať. V blízkej budúcnosti sa preto očakáva príchod webu druhej generácie, ktorá má potenciál tieto problémy vyriešiť – webu so sémantikou.

Základným princípom webu so sémantikou je myšlienka rozšírenia súčasného webu o metadáta, ktoré opisujú sémantiku informácií na webe. Príchod webu novej generácie so sebou prináša aj množstvo problémov, ktoré je potrebné skúmať a riešiť. Udržiavanie aktuálnosti metadát je na webe so sémantikou o to dôležitejšie, že v prípade nekonzistencie metadát a informácií, ktoré opisujú, môže dôjsť k neplatným uzáverom. Taktiež je dôležité vedieť kedy a ako sa údaje zmenili, čo napomôže rýchlejšej orientácii a úspore času ich konečným konzumentom.

Táto práca stavia na koncepte webu so sémantikou, uvádzame jeho základné myšlienky a princípy ako sú ontológie a jazyky, ktorých prostredníctvom sa dajú metadáta zapísať a v tomto kontexte analyzuje technológie vyvinuté konzorciom W3C. Cieľom práce je navrhnúť podporný prostriedok pre automatizovanú identifikáciu zmien medzi verziami ontológií a ich manažment. Návrh sa zameriava ako na sémantickú úroveň, tak aj na štrukturálnu úroveň a bol overený prototypom. Ďalej sa zameriava aj na riešenie problémov súvisiacich s ukladaním verzií jednotlivých metadát a pokúša sa upozorniť a hľadať riešenia problémov, vzniknutých pri používaní rôznych verzií metadát a aj problémom spájania ontológií.

Práca je zložená zo šiestich kapitol. V kapitole 2 uvedieme základné princípy webu so sémantikou, objasníme pojem ontológie, predstavíme jazyky a problémy, ktoré s týmito technológiami súvisia. Taktiež uvedieme niektoré nástroje, ktoré táto technológia využíva. V kapitole 3 podrobne analyzujeme a navrhujeme riešenia základného problému, ktorému sa práca venuje – identifikácii a manažmentu zmien ontológií. Z uvedených faktov, úvah a návrhov sa v kapitole 4 venujeme návrhu nástroja na automatizovanú identifikáciu zmien medzi verziami ontológie. Kapitola 5 opisuje spôsob overenia kritických častí navrhovaného systému prostredníctvom

implementovaného prototypu systému. V kapitole 6 zhrnieme a zhodnotíme dosiahnuté výsledky, ale taktiež načrtneme ďalšie možnosti výskumu, zlepšenia a obmedzenia vytvoreného systému. V prílohe A sa nachádza používateľská dokumentácia k vytvorenému systému a príloha B zase obsahuje jeho technickú dokumentáciu. Príloha C obsahuje podrobnú špecifikáciu a dokumentáciu k navrhnutému exportnému formátu identifikovaných zmien na báze XML. Príloha D opisuje súvis tejto diplomovej práce s predmetom Diplomový projekt a v prílohe E sa nachádza opis obsahu priloženého elektronického nosiča. V prílohe F sa nachádza kópia príspevku na konferenciu ITAT 2005, ktorý bol jedným z výsledkov diplomového projektu.

2 Web so sémantikou

Dnešný web, ktorý sa od svojho vzniku rozrástol do vopred nepredvídaného rozsahu, poskytuje množstvo nielen textových informácií, ale podáva svojim konzumentom taktiež multimediálny obsah. Práve vďaka pridaniu tohto multimediálneho obsahu, sa z pôvodného hypertextu stalo hypermédium. Množina textových, obrazových a zvukových informácií spolu s ich vzájomnými vzťahmi vytvára hyperpriestor. Súčasný web má jedno zásadné obmedzenie, ktoré spočíva v tom, že informácie sú prostredníctvom neho prezentované tak, aby boli dobre zrozumiteľné a čitateľné pre človeka. Avšak to, čo je zrozumiteľné človeku, nemusí byť tak dobre zrozumiteľné stroju. Koncepcia webu so sémantikou sa toto obmedzenie snaží zmeniť.

Web so sémantikou, sa stal aktívnou oblasťou výskumu posledných rokov. Sémantika sa zaoberá významom jazykových jednotiek a web so sémantikou umožňuje počítačové spracovávanie písaného textu, jednoduché vyhľadávanie na webe a kategorizáciu. Podľa (Berners-Lee, 2001) "web so sémantikou je rozšírením súčasného webu, v rámci ktorého má informácia presne definovaný význam, čím umožňuje lepšiu spoluprácu počítačov a ľudí".

V zmysle vyššie uvedenej definície treba súčasný web rozšíriť o definície významu informácií. V rámci týchto definícií významu je možné sa odkazovať podobne ako v prípade hypertextových dokumentov. Vytvorí sa tak sieť výrokov, ktoré je možné na rozdiel od samotných webových stránok automatizovane spracovávať (Kosek, 2003). Tieto výroky tvoria údaje o význame informácií – metadáta. Veľmi dôležitou vlastnosťou dát a metadát je to, že môžu byť jednoducho zdieľané a znovu používané v rôznych aplikáciách.

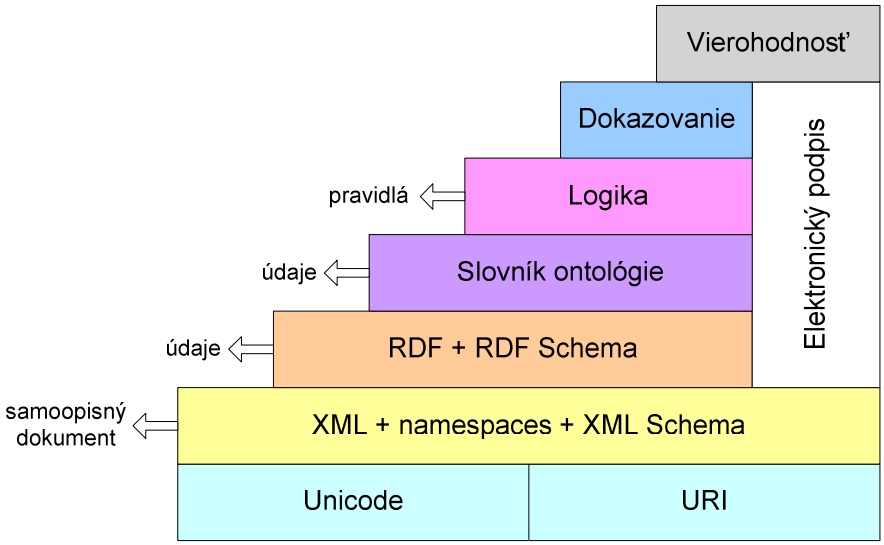
Myšlienka webu so sémantikou počíta s existenciou inteligentných sofvérových agentov na vyhľadávanie, ktoré dokážu efektívne prechádzať hyperpriestorom a vyberať len relevantné informácie. Sú schopné odpovedať na zložité používateľove požiadavky v krátkych časových intervaloch. Poslaním týchto agentov nebude len porozumieť významu prehľadávaných údajov, ale taktiež hľadať súvislosti medzi už známymi skutočnosťami. Ak sa takýto spôsob prehľadávania spojí s existenciou tak veľkého zdroja informácií, akým bezpochyby internet je, získame tak nepredstaviteľne obrovský potenciál a priestor pre získavanie poznatkov.

Dôležitým predpokladom webu so sémantikou je taktiež štandardizovaný opis webových zdrojov. Zdrojom sa v tejto súvislosti rozumie čokoľvek, teda napr. textové dokumenty, obrázky, video, zvukové súbory, atď. Každý zdroj by bol vybavený charakteristikami (autor, typ zdroja, kľúčové slová, atď.), čo by umožnilo používateľom pracovať s webom ako s databázou a dostávať odpovede na ich požiadavky napr. prostredníctvom jazyka podobného SQL. Významným dôsledkom by napríklad bola veľmi vysoká presnosť a relevancia odpovedí, čo znamená, že by sa používateľovi pri vyhľadávaní určitej informácie vrátil zoznam všetkých zdrojov, ktoré sa tejto informácie týkajú a žiadny zdroj naviac, resp. by sa vynechali odkazy na irelevantné alebo neplatné zdroje.

2.1 Koncept webu so sémantikou

Základnú predstavu o koncepte webu so sémantikou predstavil Tim Berners-Lee na konferencii XML-2000 a je znázornená na obr. 1 (Berners-Lee, 2000). Ide o vrstvovú štruktúru, kde vyššia vrstva závisí od nižších, využíva jej služby a zároveň dodáva ďalšiu silu a význam do modelu.

Na spodných vrstvách modelu sa nachádzajú technológie zabezpečujúce všeobecnú identifikáciu zdroja (URI – uniformný identifikátor zdroja) a štandardizáciu použitých jazykov vrátane lokalizácie (Unicode). Druhá vrstva (XML + namespaces + XML Schema) zabezpečuje syntaktickú jednotnosť dokumentu, čo je základným predpokladom pre správnu prácu vyšších vrstiev. Tvorí ju jazyk XML, ktorý slúži na zápis štruktúrovaných údajov v textovej podobe, priestory mien a XML Schema, ktorý je nástupcom DTD a popisuje štruktúru dokumentu. Uvedené spodné dve vrstvy tvoria technológie, ktoré si už našli svoje široké spektrum aplikácií a sú praxou overené.



obr. 1 Vrstvová štruktúra webu so sémantikou

Vrstva RDF poskytuje možnosť použiť metadáta, ktoré opisujú dáta na webe. Základom RDF je jednoduchá konštrukcia vyjadrujúca vzťah medzi subjektom a objektom. Už

takéto metadáta umožňujú strojové odvodzovanie vzťahov medzi zdrojmi. RDF Schema tvorí nadstavbu nad RDF a umožňuje vytvárať triedy, sloty, definovať definičný obor, obor hodnôt, vytvárať hierarchie tried, atď. Tieto technológie tvoria základ pre web so sémantikou, pretože všetky jeho ďalšie vrstvy stavajú práve na týchto dvoch technológiách.

Slovník ontológie poskytuje ešte väčšiu paletu metadát, ako sú napr. tranzitívne vlastnosti, určenie unikátnosti, obmedzenia hodnôt, atď. Túto vrstvu tvorí jazyk OWL, ktorý bol v roku 2004 konzorciom W3C vydaný ako štandard pre zápis a publikovanie ontológií na webe.

Vrstva logiky predstavuje systém pravidiel, na základe ktorých sa môžu dokazovať ďalšie fakty. Aby sa dala dosiahnuť vierohodnosť informácií, ktorá je taktiež naznačená na obr. 1, je možné vrstvy zabezpečiť naviac napr. elektronickým podpisom, čo je v súčasnosti taktiež praxou overená technológia.

2.2 Ontológie

V predchádzajúcej časti sme sa oboznámili so skutočnosťou, že údaje o význame informácií, tzv. metadáta sú základným kameňom webu so sémantikou. Tieto metadáta je možné reprezentovať okrem iného aj prostredníctvom ontológií. Ontológia je vo filozofickom zmysle náuka o bytí, poprípade ako univerzálna sústava znalostí popisujúcich objekty, javy a zákonitosti sveta „tak ako je“, t.j. nezávisle od ľudského usudzovania o ňom. V „informatickom“ chápaní opisuje ontológia to, čo už existuje a môže byť reprezentované v informačnom, resp. znalostnom systéme. Pre naše potreby použijeme definíciu ontológie podľa (Gruber, 1993), podľa ktorej „ontológia je explicitná špecifikácia konceptualizácie“. Táto definícia požaduje, že konceptualizácia, t.j. systém pojmov modelujúci určitú časť sveta, musí byť špecifikovaná explicitne, t.j. nie je len „skrytá“ v hlave svojho autora. Z hľadiska znalostného inžinierstva je možné ontológie, používané v procese vývoja znalostnej aplikácie, taktiež chápať ako znalostné modely, teda abstraktné opisy určitej časti znalostného systému, ktoré sú relatívne nezávislé od finálnej reprezentácie a implementácie znalostí.

Táto definícia bola neskôr doplnená o pojem „formálna“ na „ontológia je explicitná formálna špecifikácia zdieľanej konceptualizácie“. Podstatným doplnením definície je, že ide o modely časti sveta, ktoré sú zdielateľné viacerými procesmi v rámci jednej aplikácie a opakovateľne použiteľné pre rôzne ďalšie aplikácie. Aplikácie pritom môžu byť oddelené časovo, priestorovo i personálne (Svátek, 2002). Tieto možnosti už boli naznačené vyššie. Podľa tejto definície sa naviac kladie požiadavka na formalizmus ontológie. Táto požiadavka je v našom ponímaní logická a je dobré, že je takto explicitne vyjadrená, pretože v prípade webu so sémantikou je snaha zameraná na spracovanie dát a metadát strojmi, takže táto požiadavka je na mieste. Ťažko by sa strojovo spracovávali akékoľvek dáta bez ich formálneho vyjadrenia.

Základnými spôsobmi využitia ontológií sú (Svátek, 2002):

- podpora porozumenia medzi ľuďmi (napr. medzi expertmi a znalostnými inžiniermi),
- podpora komunikácie medzi počítačovými systémami,
- uľahčenie návrhu znalostne orientovaných aplikácií.

Ontológie je možné rozdeliť prinajmenšom na tri typy, ktoré sa dajú chápať aj ako súčasť tradičnejších odborov:

- *terminologické* – pokročilejšie synonymické slovníky, ktoré sa používajú v knihovníctve a oboroch zameraných prevažne na textové informácie,
- *informačné* – rozvinutie databázových konceptuálnych schém, ktoré zaisťujú abstrakciu a vyššiu kontrolu integrity,
- *znalostné* – reprezentácie znalostí v rámci umelej inteligencie; objekty a relácie medzi objektmi sú dôsledne definované prostredníctvom formálneho jazyka.

Jedným zo základných prvkov ontológií sú triedy. Triedy predstavujú spravidla hierarchicky usporiadané množiny prvkov s rovnakými vlastnosťami, pričom samotnú triedu jednoznačne opisuje jej meno. Táto úroveň ontológie sa nazýva *sémantickou*.

Inštancie tried sa nazývajú *indivíduá*. Jedná sa o konkrétne objekty záujmu v určitej doméne. Inštancia je vždy identifikovaná pomocou URI. Na rozdiel od tried v objektovo-orientovaných jazykoch, ontologické triedy nemajú procedurálne metódy. Túto úroveň ontológie nazývame *štrukturálnou*.

Podobne ako v prípade relačných databáz, je možné aj v prípade ontológií definovať relácie n-tíc objektov. Pre binárne operácie sa používa pojem *slot*, prípadne vlastnosť. Na rozdiel od objektovo-orientovaného prístupu nie sú sloty napevno spojené so žiadnou triedou ako časť ich definície a väzba na svoj definičný obor (resp. obor hodnôt) je sprostredkovaná len obmedzeniami. Podobne ako triedy, môžu mať aj sloty nad sebou definovanú hierarchiu. Triedy tak môžu mať definované niekoľko slotov, pričom indivíduá definujú samotnú informačnú hodnotu slotov, ktoré predstavujú konkrétne údaje.

Využitie ontológií v kontexte webu so sémantikou je možné v súčasnej dobe zhrnúť do týchto oblastí (Svátek, 2002):

- *znalostný manažment vo firmách* – ontológiami je možné zachytiť vecnú podstatu vedomostí, z čoho vyplýva možnosť strojovej kontroly ich konzistencie, aktuálnosti, zaistenia jednoduchého a presného vyhľadávania, ako aj ich zjednodušeného vkladania,

- *elektronický obchod typu B2B a B2C* – jednoduché vyhľadávania produktov, obchodných partnerov, ale napr. aj zjednodušené dohadovanie obchodných podmienok,
- *spracovanie prirodzeného jazyka* – terminologické ontológie môžu napomáhať pri prekladoch alebo automatickej tvorbe rešerší,
- *inteligentná integrácia informácií* – zastrešenie schém údajov distribuovaných heterogénnych zdrojov,
- *webové portály so sémantikou* – umožnia poloautomatické vytváranie obsahu na základe metadát od poskytovateľa obsahu,
- *vyhľadávanie informácií*,
- *inteligentné výučbové systémy*.

Reprezentácii a zápisu ontológií sa venuje množstvo autorov a postupom času sa vyvinulo množstvo jazykov určených na ich zápis. V ďalších častiach sa zameriame na jazyky, ktorými sa zapisujú „webové“ ontológie a vznikli na pôde konzorcia W3C.

2.2.1 RDF

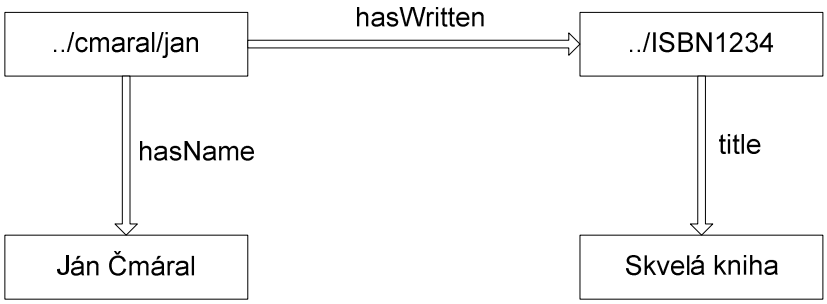
Resource Description Framework (RDF) je doporučením W3C z r. 1999 pre reprezentáciu štruktúry metadát popisujúcich údaje na webe (W3C, 2004b). Model údajov RDF je založený na lingvisticky inšpirovanej konštrukcii výrokov, ktoré sú zložené z troch prvkov: *subjekt*, *predikát* a *objekt*. Znamená to, že ľubovoľný subjekt je možné opísať predikátom, ktorému prislúcha objekt. Tento vzťah znázorňuje obr. 2.

obr. 2 Vzťah medzi subjektom, predikátom a objektom

Subjekt môže tvoriť URI alebo prázdny uzol, predikát je tvorený URI a objekt môže byť reprezentovaný URI, prázdnym uzlom alebo literálom. Vďaka použitiu URI ako identifikátora zdrojov, môže byť týmto zdrojom ako webová stránka, tak aj napr. celé webové sídlo alebo aj dokument umiestnený mimo webu. Literálom sa rozumie elementárna údajová hodnota, t.j. číslo, znak, ale aj reťazec znakov, atď. Takýmto spôsobom je možné zapisovať jednoduché pravidlá a znalosti o zdrojoch prostredníctvom vlastností a ich hodnôt.

Vyjadrenie zložitejších štruktúr pomocou RDF vyžaduje ich dekompozíciu do trojíc previazaných premennými, pričom sú takto previazané trojice následne serializované napr. pomocou XML. Reprezentácií takýchto výrokov je viacero, napr. RDF/XML, N3, N-Triples, TriX, ale je možné tieto výroky vizualizovať aj pomocou grafov.

Na obr. 3 je príklad RDF vo forme grafu, ktorý reprezentuje fakt, že zdroju *.../cmaral/jan* je priradené meno „*Ján Čmáral*" a napísal knihu s názvom „*Skvelá kniha*", ktorá je reprezentovaná zdrojom *.../ISBN1234*. Takto zapísaný graf je možné serializovať do XML, ako je znázornené na obr. 4. Na tomto mieste treba podotknúť, že uvedený zápis RDF/XML nie je jediný a existuje viacero ďalších možností, ako je možné uvedený kus kódu v tomto jazyku zapísať (vrátane zmeny poradia niektorých riadkov). Pri všetkých týchto variantoch sa však samotný význam zachováva.



obr. 3 Príklad RDF grafu

```
<rdf:Description rdf:about="http://www.writers.org/cmaral/jan">
  <s:hasName>Ján Čmáral</s:hasName>
  <s:hasWritten rdf:resource="http://www.books.org/ISBN1234"/>
</rdf:Description>

<rdf:Description rdf:about="http://www.books.org/ISBN1234">
  <rdf:type rdf:resource="http://www.description.org/schema#Book"/>
  <s:title>Skvelá kniha</s:title>
</rdf:Description>
```

obr. 4 Príklad zapísaný v RDF/XML

Zásadný príklon k RDF ako univerzálnej vyjadrovacej štruktúre sa časovo zhruba zhoduje so vznikom webu so sémantikou, a aj preto sa niekedy web so sémantikou definuje dosť nepresne ako sieť sémantických metadát zapísaných pomocou RDF a aj problematika ontologických jazykov je od tej doby z veľkej časti podriaďovaná práve webu so sémantikou (Svátek, 2002).

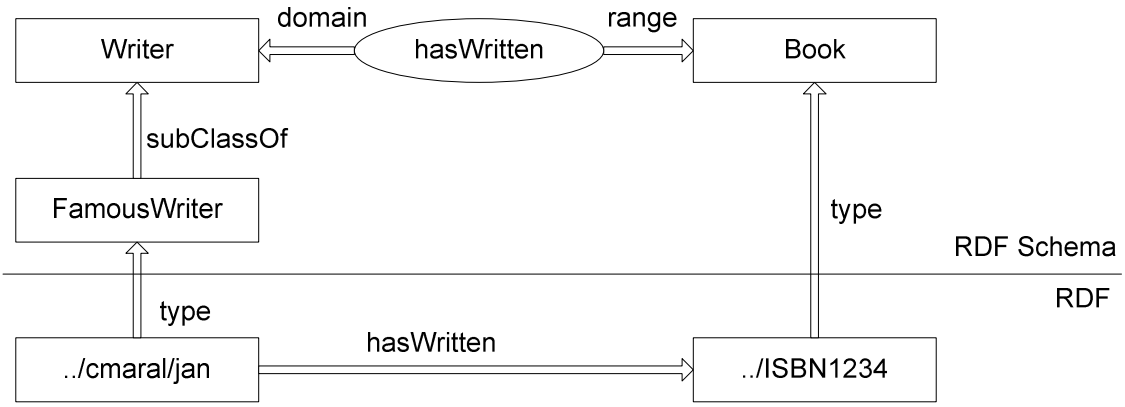
2.2.2 RDF Schema

RDF Schema (inak aj RDFS) predstavuje jazyk pre opis slovníka a dopĺňa do štruktúry RDF hlavné konštrukcie z rámcových či objektových systémov, t.j. hierarchie tried a binárne sloty s možnosťou stanoviť definičný obor a obor hodnôt. Nad triedami i slotmi môže byť definovaná hierarchia. Zdroje z RDF je potom možné jednoducho priraďovať triedam z RDF Schema ako ich inštancie pomocou atribútu *type*. RDF Schema však ponúka aj ďalšie vlastnosti, ako sú kontajnerové triedy a vlastnosti, kolekcie, reifikáciu, údajové typy, atď. Dá sa teda povedať, že v istom zmysle tvorí zápis v RDF Schema metadáta o metadátach (W3C, 2004a).

RDF Schema spĺňa intuitívne požiadavky webových návrhárov na možnosť zachytenia sémantiky obsahu stránok, najme pokiaľ ide o definovanie hierarchických štruktúr. Oproti tradičným ontologickým jazykom mu však chýba možnosť precíznejšie špecifikovať podmienky príslušnosti k triedam a úplne mu chýbajú údajové typy (Svátek, 2002).

Všetky výrazy zapísané v RDF Schema sú valídne s RDF a rozdiel medzi týmito dvoma jazykmi je v tom, že RDF Schema pridáva niektorým atribútom význam, ktorý má definovanú interpretáciu. Napr. atribútom *type*, ako už bolo spomenuté, sa dá priradiť trieda k nejakému RDF tvrdeniu, alebo prostredníctvom atribútu *subClassOf* je možné vytvárať hierarchiu tried (Broekstra, 2002).

Graf na obr. 5 znázorňuje vzťah medzi RDF a RDF Schema, pričom naznačuje možnosť definovania hierarchie tried, definovania vlastností v rámci triedy, definovania ich definičného oboru a oboru hodnôt. Spodnú časť obrázku sme si opísali už v časti 2.2.1. Táto časť je vďaka RDF Schema rozšírená cez atribút *type* tak, že zdroju *http://www.writers.org/cmaral/jan* je priradená trieda *FamousWriter* a zdroju *http://www.books.org/ISBN1234* je priradená trieda *Book*. Vďaka RDF Schema vieme ďalej určiť, že *FamousWriter* je podtriedou triedy *Writer*, ktorá má definovanú vlastnosť *hasWritten* s oborom hodnôt, tvorený práve triedou *Book*.



obr. 5 Príklad RDFS a vzťahu s RDF (Broekstra, 2002)

Pohľad na RDF Schema je možné analogicky zapísať aj v relačnej databáze. Rozdielom oproti relačným databázam však je to, že RDF Schema nevyžaduje striktný zápis inštancií podľa ontologického zápisu. Aj napriek podobnosti názvov, RDF Schema plní odlišnú úlohu ako XML Schema. Zatiaľ čo XML Schema určuje poradie a možné kombinovanie značiek v XML dokumente, RDF Schema nesie informáciu o interpretácii výrazov v RDF modeli a nič nevraví o syntaktickom vzhľade RDF opisu. RDF Schema je postavená nad RDF a poskytuje konceptuálny pohľad na RDF trojice, umožňuje teda definovať triedy, vzťahy medzi nimi a ich vlastnosti.

2.2.3 OWL

OWL (Ontology Web Language) je jazyk navrhnutý pre tvorbu komplexných ontológií (W3C, 2004c). Rozširuje RDF a RDF Schema o ďalšie elementy vzťahujúce sa k triedam a vlastnostiam. Predchodcom jazyka OWL bol jazyk DAML+OIL, ktorý vznikol doplnením jazyka DAML (DARPA Agent Markup Language), vyvinutého americkou armádnou organizáciou DARPA, o niektoré konštrukcie európskeho projektu OIL (Ontology Inference Layer).

OWL existuje vo verziách Lite, DL a Full. Dôvodom vzniku týchto verzií boli dve navzájom protichodné požiadavky – náročnosť na odvodzovacie prostriedky a výrazová sila. OWL Full obsahuje všetky konštrukcie jazyka, ale nie je úplne vhodný pre odvodzovanie. OWL DL (Description Logic) prináša určité obmedzenia ako je nemožnosť využívať všetky RDF konštrukcie, trieda ani vlastnosť nesmie byť zároveň inštanciou a taktiež platia určité obmedzenia pre použitie OWL výrazov. OWL Lite predstavuje najnižšiu úroveň zložitosti. S jeho pomocou je umožnené definovať základnú hierarchiu a jednoduché obmedzenia, ale nesmú sa používať niektoré elementy na definíciu tried.

Ontológia v OWL sa skladá z hlavičky sa samotného dokumentu. V hlavičke, ktorej príklad je znázornený na obr. 6 sa definuje komentár, verzia, prípadne ďalšia vnorená ontológia. Telo dokumentu sa podobá na RDF a RDF Schema.

```
<owl:Ontology rdf:about="http://www.vzorova.ontologia.sk/muni">
  <owl:versionInfo>v 1.0, 13.01.2003, 14:21</owl:versionInfo>
  <owl:priorVersion rdf:resource=
      "http://www.vzorova.ontologia.sk/muni030101.owl"/>
  <rdfs:comment> Vzorová OWL </rdfs:comment>
  <owl:imports rdf:resource="http://vzorova.ontologia/msmt"/>
</owl:Ontology>
```

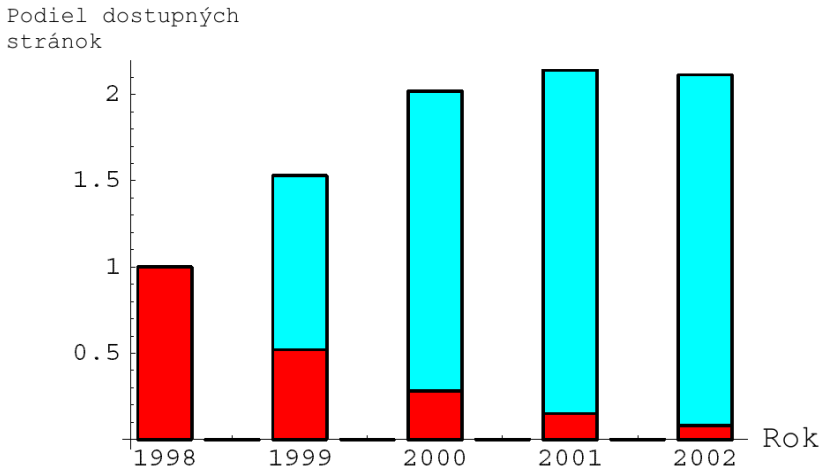
obr. 6 Hlavička vzorovej OWL ontológie

Samotné triedy je možné v OWL definovať identifikátorom, vymenovaním prvkov, zjednotením iných tried, prienikom, prípadne doplnkom. Príklady definície takýchto tried je možné nájsť v (Hradský, 2003). Celkovo pri hodnotení jazyka OWL je dobré vyzdvihnúť, že umožňuje jednoduchý zápis ontológie a taktiež umožňuje rozširovať už existujúce ontológie. Poskytuje prostriedky pre vývoj ontológií v čase a v prípade rôznej interpretácie nejakej oblasti poskytuje nástroje na mapovanie rôznych ontológií navzájom.

2.3 Verzie ontológií

Existuje množstvo štúdií, ktoré sa zaoberajú skúmaním zastarávania a evolúcie webu (Fatterly, 2004), (Klein, 2002a), (Ntoulas, 2004). Podľa výsledkov Online Computer Library Center (OCLC, 2002) sa približne 50% zo sledovaných webových stránok v rozmedzí rokov 1998 až 2002 stane každoročne nedostupnými. Výsledky tejto štúdie

sú zobrazené na obr. 7. Na x-ovej osi je znázornený uvažovaný rok, v ktorom prieskum prebiehal a na y-ovej osi je znázornený celkový počet dostupných webových stránok. Tmavý stĺpec predstavuje množstvo dostupných sledovaných stránok oproti roku 1998 pričom svetlý stĺpec predstavuje celkové množstvo dostupných sledovaných webových stránok v danom roku.



obr. 7 Percento dostupných webových sídiel v danom roku (Ntoulas, 2004)

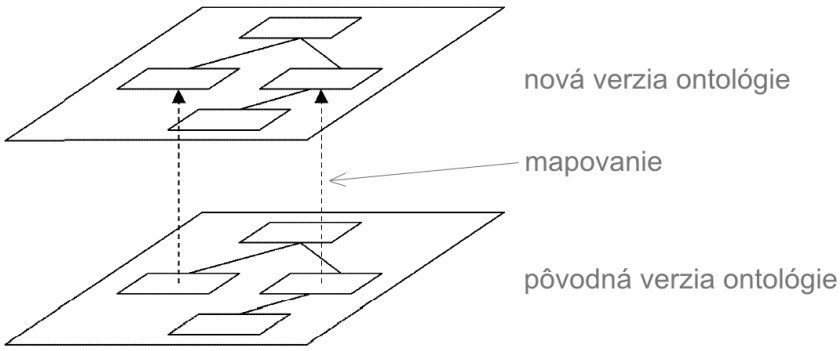
Je možné predpokladať, že v prípade rozšírenia webu so sémantikou, by jeho evolúcia mohla prebiehať podobne rýchlo ako v prípade súčasného webu (ak nie rýchlejšie). To je aj dôvodom, prečo je vhodné premýšľať o spôsobe, ako zabezpečiť, aby boli metadáta v každom okamihu konzistentné s obsahom, ktorý popisujú. Ontológie pozostávajú z tried, vlastností, indivíduí a vzťahmi medzi nimi, ktoré modelujú nejakú časť sveta. Samotnú ontológiu môžu vytvárať viacerí ľudia, pričom každý z nich môže rovnakú časť sveta modelovať iným spôsobom. Dokonca aj v prípade, že ontológie vytvára jeden človek, môže vytvoriť v čase viacero verzií jednej ontológie.

Uvažujeme dve verzie ontológie, pričom jediným rozdielom medzi nimi by bol len názov triedy, s ktorými by pracoval agent, majúci za úlohu vyhľadávanie informácií v takejto ontológii. Agent je vytvorený pre jednu konkrétnu verziu ontológie, takže má problém pracovať s inou verziou ontológie, pretože nepozná identifikátor takejto triedy a nevie identifikovať jej indivíduá, t.j. tie, ktoré prehľadáva. Keby však agent poznal rozdiely alebo mapovanie medzi triedami verzií ontológie, vedel by si príslušnú triedu a aj jej indivíduá sprístupniť. Takýmto spôsobom by mohol pracovať aj s inými verziami ontológie, než pre ktoré bol pôvodne vytvorený.

Rozdiely medzi dvoma verziami ontológie môžu byť rôzne. Môžu ich predstavovať len jednoduché premenovania tried, či slotov, zmeny typov dát, či obmedzení, ale môžu predstavovať aj komplexné zmeny v hierarchii tried a konečnom dôsledku aj inú sémantiku dát pri zmene domény. Vhodnou aplikáciou identifikácie takýchto zmien je kontrola aktuálnosti informácií, ktoré sa v konečnom dôsledku ponúkajú používateľovi. Predstavme si burzové správy, správy na trhu pracovných ponúk, či nejaké pravidelne

vyhodnocované štatistiky. Ontológia, popisujúca uvedené časti sveta, by po určitú dobu mohla byť na sémantickej úrovni rovnaká, t.j. plat zamestnanca by mal vždy rovnaký význam, podobne aj význam kurzu akcie, atď. Na štrukturálnej úrovni by sa však v čase menila omnoho častejšie ako po sémantickej, t.j. pridávanie nových pracovných ponúk, zmena údajov o ponúkanom plate zamestnanca, zmena kurzu akcií spoločnosti, atď.

Na základe uvedenej úvahy môžeme povedať, že nová verzia ontológie definuje vzťah medzi pôvodnou definíciou ontológie a definíciou v jej novej verzii. Tento vzťah ilustruje obr. 8. V spodnej časti sa nachádza pôvodná verzia ontológie a v hornej časti sa nachádza jej nová verzia. Vzťahy medzi nimi sú znázornené vertikálnymi šípkami.



obr. 8 Vzťah medzi dvoma verziami ontológie (Klein, 2002b)

Pri identifikácii rozdielov medzi verziami ontológie musíme vziať do úvahy niekoľko aspektov. Jedným je fakt, čo vlastne v danej verzii definície ontológie bolo zmenené. Aby sme mohli ľahko sledovať tieto zmeny, treba zadefinovať množinu operácií týkajúcich sa zmien, ako je napr. pridanie triedy/slotu, odstránenie triedy/slotu, atď. Iným pohľadom môže byť mapovanie medzi dvoma verziami ontológie, t.j. určenie ktorý prvok v jednej verzii zodpovedá prvku v druhej verzii. Oba tieto prístupy je možné vhodne skombinovať za účelom čo najvyššej efektivity.

Na tomto mieste je vhodné ešte spomenúť, že mapovanie ontológií sa nemusí využívať len pri porovnávaní ich verzií, ale existujú aj projekty, ktoré sa snažia realizovať mapovanie z ontológií napr. do programovacích jazykov. Takéto mapovanie je dôležité napr. v prípade rámca, ktorý by dokázal pre danú ontológiu vytvoriť napr. hierarchiu tried a indivíduí priamo z OWL súboru. Príkladom je projekt OntoJava (Kalyanpur, 2004), ktorý umožňuje zo súboru ontológie vygenerovať sadu tried v jazyku Java.

Dôležité je zohľadňovať aj granularitu zmien, t.j. či uvažujeme zmeny na úrovni slov, riadkov, definícii, celých súborov, atď. V prípade zmien na úrovni sémantiky je vhodné túto granularitu stanoviť v rozsahu definície, pretože je len otázkou syntaxe, ako sa sémantika zapíše. Na syntaktickej úrovni je v prípade indivíduí vhodné granularitu voliť na úrovni definície, ale v prípade hodnôt literálov je úroveň granularity otázna, pretože literál môže obsahovať ako znak, či číslo, tak aj rozsiahle texty.

Vhodné je taktiež uchovávať informácie o verziách ontológie, na základe ktorých sa dá identifikovať, kto, kedy a prečo vykonal danú úpravu, či zmenu v ontológii. Pri zmenách ontológie je dôležité taktiež vymedziť platnosť zamýšľaného kontextu, pričom jednoduchým príkladom vymedzenia takejto platnosti môže byť určenie časovej platnosti danej verzie ontológie.

Existujú aj také zmeny ontológie, ktoré nemôžu byť automaticky identifikované, pretože závisia od rozhodnutí samotného tvorcu ontológie. Môžu existovať zmeny, ktoré sú len zapísané, ale ich logický význam ostáva zachovaný (napr. definovanie zápisu dátumu vo formáte „rrrr-mm-dd“ má rovnaký význam ako zápis vo formáte „dd.mm.rrrr“) a na druhej strane môže byť reprezentácia rovnaká, pričom logický význam bude iný (napr. udanie hodnoty vzdialenosti bez jednotky má rovnaký spôsob reprezentácie číselnou hodnotou; keď je implicitne dané, že v jednej verzii sa tou hodnotou myslí meter a v druhej to je stopa, prichádza k rozdielom na logickej úrovni).

Podpora verziovania ontológií je dôležitá, pretože zmeny v ontológiách môžu spôsobiť už spomínanú nekonzistenciu (napr. vymazaním indivídua zo zreťazeného zoznamu), prípadne nekompatibilitu (napr. zmenou modelu údajov softvérových systémov). Existuje niekoľko dôsledkov, ktoré zmenou údajov alebo metadát môžu nastať. Prvým dôsledkom môže byť nekonzistencia, ktorá vznikne inou interpretáciou údajov po zmene ontológie. Druhý dôsledok môže nastať v prípade, že jedna ontológia používa inú ontológiu, čiže sú dve alebo viacej ontológií od seba závislých a v prípade zmeny jednej majú vplyv aj na ostatné ontológie. V takomto prípade dochádza taktiež k rozdielnej interpretácii modelu, čo môže spôsobovať medzi modelom a svetom, ktorý sa ontológiou modeluje. Tretím dôsledkom je, že po zmene ontológie môže prestať fungovať aplikácia, ktorá takúto ontológiu používa (Klein, 2001). Pokiaľ uvažujeme nižšiu úroveň, tak negatívny vplyv na ontológiu môžu mať zmeny DNS záznamov, či IP adries, čo by malo za následok úplnú nedostupnosť zdrojov, ktoré ontológie popisujú, prípadne sa na ne odkazujú. Tento problém je však dobre známy už z dnešnej podoby webu.

Zmeny vonkajšieho sveta majú na ontológie taktiež silný dopad, pretože práve ontológie tento svet opisujú. Pokiaľ sa zmení doména, t.j. časť sveta, ktorú ontológia opisuje, môže dôjsť k nekonzistencii medzi týmto svetom, údajmi a metadátami, t.j. ontológia už neopisuje časť sveta, ktorú pôvodne opisovala. Jednoduchým príkladom môže byť zmena organizačnej štruktúry úradu či firmy. Po nástupe nového vedenia sa organizačná štruktúra zmení a teda sa musí meniť aj model údajov, ktorý túto štruktúru opisuje. Iný pohľad na tento problém môže byť aj taký, kedy sa snažíme jednu organizačnú štruktúru aplikovať na viacero organizácií.

Nie každá zmena môže znamenať, že verzia ontológie bude kompatibilná s inou verziou, preto medzi ontológiami môžeme uvažovať štyri druhy kompatibility:

- *spätná kompatibilita* – sémantika ontológie je zmenená tak, že nová verzia ontológie vie interpretovať údaje rovnako ako predchádzajúce verzie, napr. pridaním nezávislej triedy,
- *doprednáá kompatibilita* – sémantika ontológie je zmenená tak, že verzia ontológie môže byť použitá na správnu interpretáciu novších údajov, napr. odstránením nezávislej triedy,
- *plná kompatibilita* – ide o spätnú aj doprednú kompatibilitu súčasne, t.j. sémantika ontológie nie je zmenená,
- *nekompatibilita* – sémantika ontológie je zmenená tak, že interpretácia starších údajov je nesprávna, napr. zmenou pozície v hierarchii tried.

Spätná aj doprednáá kompatibilita je tranzitívna, takže ak sú napr. zmeny z verzie V_1 do V_2 spätne kompatibilné a aj zmeny z verzie V_2 do V_3 sú spätne kompatibilné, potom aj zmeny z verzie V_1 do verzie V_3 sú spätne kompatibilné.

2.4 Vybrané nástroje pre prácu s ontológiami

V tejto časti sa zameriame na analýzu existujúcich systémov pre prácu s ontológiami. Ukážeme si editor ontológií, prostriedky na kontrolu zmien a správu verzií a na záver sa zameriame na rámce pre prácu s ontológiami, ktoré sa dajú využiť pri tvorbe vlastných systémov pracujúcich s ontológiami.

2.4.1 Protégé

Protégé (Protégé, 2005) je voľne dostupný editor, ktorý umožňuje vytvárať a upravovať celé ontológie, vytvárať formuláre pre vkladanie údajov ako aj samotné vkladanie údajov prostredníctvom týchto formulárov do novovytvorených inštancií tried. Jednoduchým spôsobom je možné vytvárať triedy a sloty, vytvárať z nich hierarchiu.

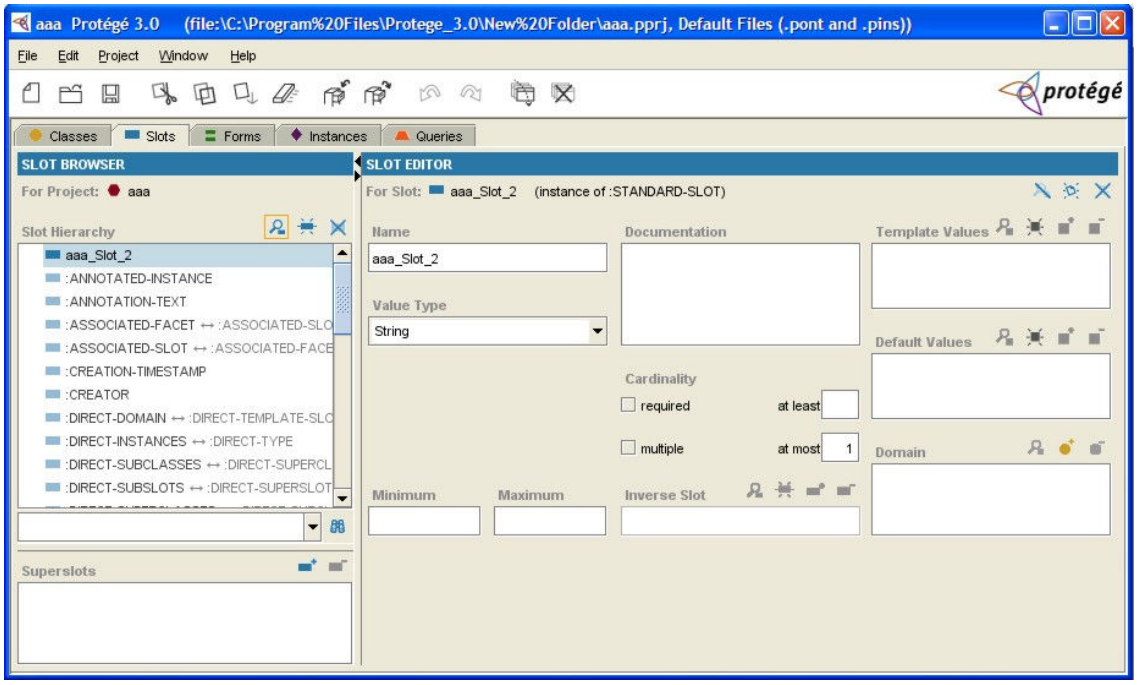
Editor poskytuje rozhranie pre zásuvné moduly, ktorými sa dá rozšíriť o širokú škálu možností, ktoré zahŕňajú grafické komponenty, ale aj zvuk či video. Výsledky je možné exportovať do najrôznejších formátov ako je napr. OWL, RDF, RDF Schema, XML, atď. Podporovaný je aj import z týchto formátov. Príklad používateľského prostredia je znázornený na obr. 9.

2.4.2 Prompt

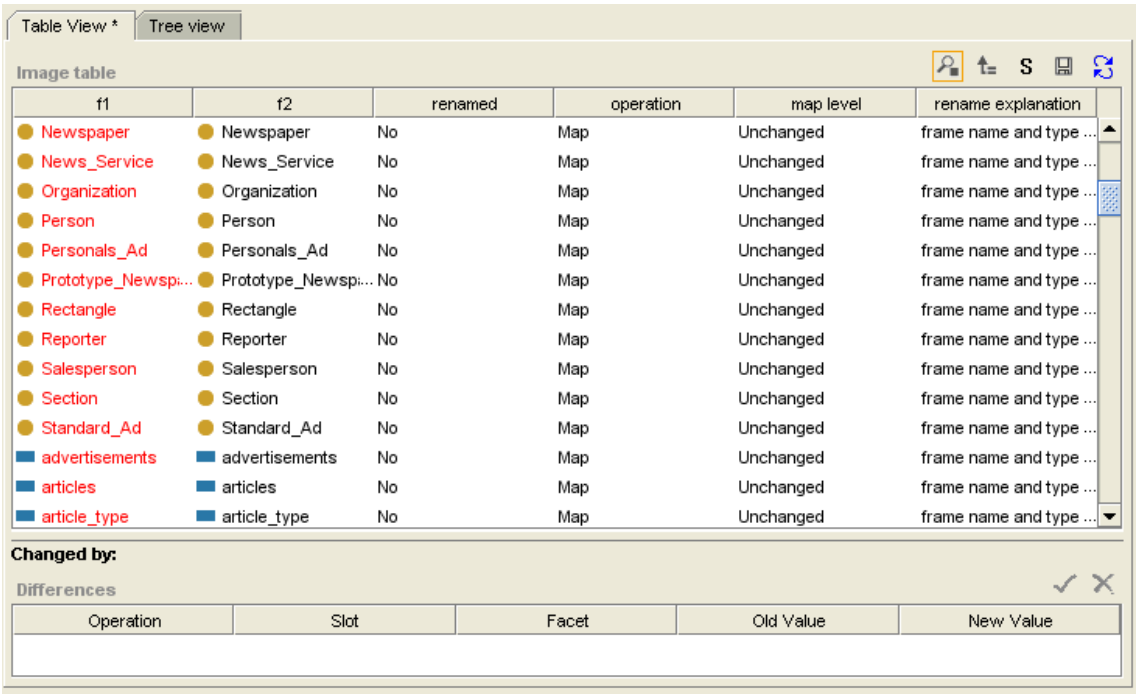
Prompt (Prompt, 2005) je zásuvný modul do systému Protégé umožňujúci interaktívne porovnávanie verzií ontológie, presúvanie časti ontológie do iných ontológií, extrahovanie častí ontológie a zlučovanie ontológií do jednej ontológie. Príklad jeho používateľského prostredia je znázornený na obr. 10.

Pre porovnávanie medzi verziami ontológie používa komponentu nazvanú PromptDiff (Noy, 2003), ktorá slúži na verziovanie ontológií. Na nájdené rozdiely ponúka dva typy pohľadov – tabuľku a strom, pričom sú v oboch pohľadoch prehľadne zobrazené

pridané, odobraté, zmenené a premiestnené časti ontológie. Pri porovnaní má používateľ možnosť prijať alebo zamietnuť identifikované zmeny. Na identifikáciu zmien využíva heuristické metódy, z ktorých niektoré sú využité aj v návrhu identifikácie sémantických zmien (pozri časť 3.1).



obr. 9 Ukážka používateľského prostredia editora ontológií Protégé



obr. 10 Tabuľkový pohľad na výsledok porovnania v zásuvnom module Prompt

Presúvaním častí ontológií sa dá znovu použiť časť už vyvinutej ontológie v inom projekte. Pri presúvaní má používateľ možnosť presne zvoliť, ktoré časti ontológie budú

presunuté, aby sa zabezpečila konzistentnosť. Zásuvný modul taktiež umožňuje aj extrakciu časti ontológie napr. pre účely použitia v inom projekte, distribúcie časti ontológie či jej ladenia. Používateľ si teda môže vybrať časť ontológie, ktorá bude extrahovaná, pričom pôvodná ontológia ostane nedotknutá. Posledným režimom, v ktorom môže Prompt pracovať je režim zlučovania ontológií do jednej ontológie. Ide o interaktívny postup, pri ktorom nástroj poskytuje isté odporúčania, identifikuje konflikty a poskytuje stratégie, ako sa konfliktom vyhnúť.

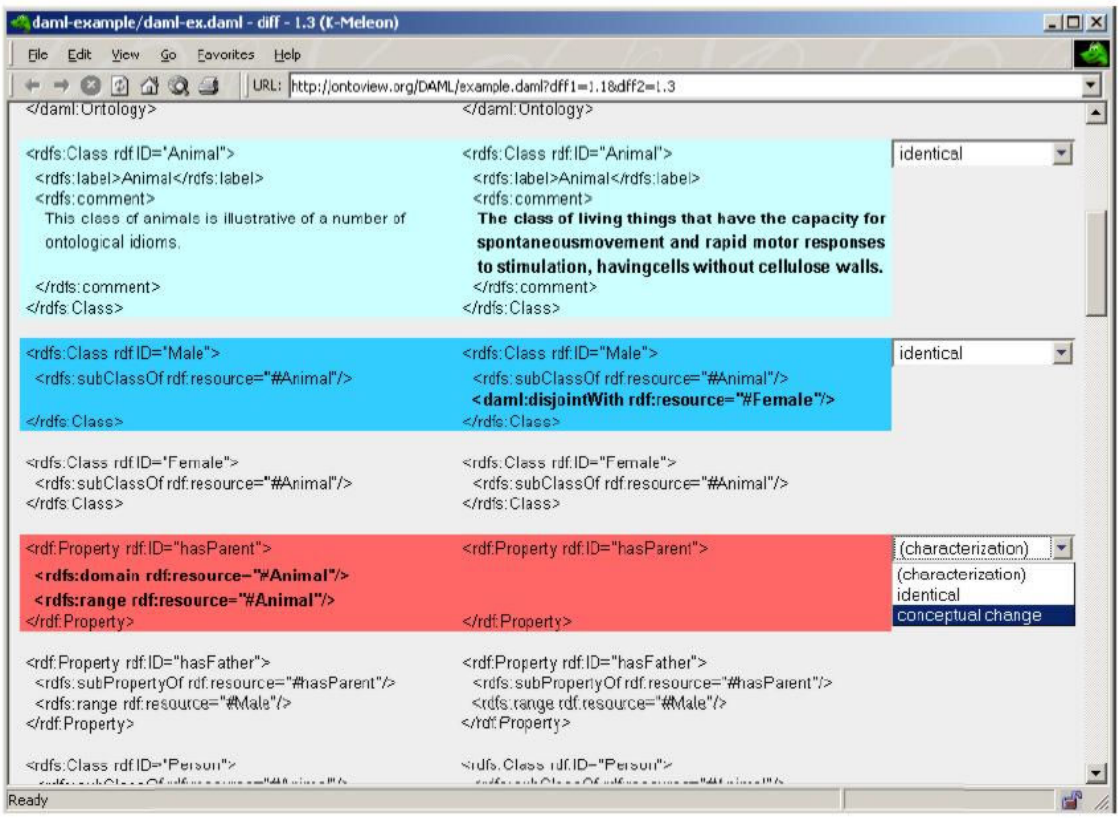
2.4.3 OntoView

Systém OntoView (OntoView, 2005) je inšpirovaný systémom CVS, ktorý sa používa pri spolupráci na tímovom vývoji softvéru. Jeho prvá verzia bola taktiež založená na CVS a jeho webovom rozhraní CVSWeb. Postupom času sa však vývoj ubral smerom vytvorenia novej implementácie, postavenej na vlastných základoch. Prvá verzia síce podporuje len jazyky DAML+OIL (z ktorého neskôr vznikol OWL) a RDF Schema, ale autori predpokladajú, že budúce verzie budú môcť byť prostredníctvom systému zásuvných modulov rozšírené o podporu ďalších jazykov na reprezentáciu ontológií.

V súčasnej dobe môžu byť ontológie do systému vkladané buď to zadaním URL alebo priamym vložením súboru do systému. Používateľ musí určiť, či zadávaná ontológia je nová alebo je to iná verzia už existujúcej ontológie a následne sa systém používateľa spýta na charakterizáciu typu zmien.

Jednou z najdôležitejších vlastností systému OntoView je schopnosť porovnávať ontológie na sémantickej úrovni, k čomu autorov inšpiroval systém CVSWeb. Na rozdiel od systému CVSWeb, ktorý vykonával porovnávanie verzií na úrovni riadkov, OntoView porovnáva ontológie na konceptuálnej úrovni, t.j. zobrazuje, ktoré definície ontológie, prípadne jej vlastnosti sú zmenené. Príklad porovnania dvoch ontológií v systéme OntoView je znázornený na obr. 11.

Systém je schopný identifikovať pridanú alebo zmenenú definíciu, identifikovať zmenu, ktorá nesúvisí s logikou triedy, ako je napr. zmenený komentár, identifikovať logické zmeny triedy (napr. zmena oboru hodnôt, prípadne určenie rodičovskej triedy). Systém vie tieto operácie vykonávať automaticky a vyznačí danú zmenu určitou farbou. Automaticky nedokáže určiť typ zmeny a tu je potrebný zásah používateľa, ktorý určí, či je význam zmeny sémantický, prípadne môže určiť vzťah medzi dvoma verziami konceptu. OntoView taktiež poskytuje základné nástroje na analýzu dôsledkov zmeny verzie ontológie a aj nástroje na export zmien ontológií (Klein, 2002a).



obr. 11 Porovnávanie dvoch ontológií v systéme OntoView

2.4.4 Jena

Jena (Jena, 2005) je voľne dostupný rámec vytvorený v jazyku Java, ktorý umožňuje tvorbu aplikácií založených na webe so sémantikou. Poskytuje programové prostriedky pre prácu s RDF, RDF Schema a OWL vrátane odvodzovacieho stroja a možnosti spolupráce s SQL servermi. Na obr. 12 je úsek programového kódu v jazyku Java, ktorý prostredníctvom rámca Jena vytvorí jednoduchý model, kde je k zdroju *http://j.hrasko.org* priradené plné meno osoby „*Janko Hraško*“.

```
// vytvorenie prázdneho modelu
Model model = ModelFactory.createDefaultModel();

// vytvorenie zdroja
Resource jankoHrasko = model.createResource("http://j.hrasko.org");

// pridanie atribútu a hodnoty ku zdroju
jankoHrasko.addProperty(VCARD.FN, "Janko Hraško");
```

obr. 12 Príklad tvorby jednoduchého modelu pomocou rámca Jena

Rámec uchováva interne všetky RDF tvrdenia v trojiciach. Pre príklad z obr. 12 je takáto trojica znázornená na obr. 13. Tu si môžeme všimnúť, že ide o klasický zápis trojice subjekt – predikát – objekt, tak ako bolo spomenuté vyššie. Pokiaľ by v zápise bolo použité nejaké tvrdenie, ktoré obsahuje prázdny uzol, rámec Jena ho reprezentuje

svojim vlastným interným identifikátorom, s ktorým môže pracovať taktiež aj aplikácia, ktorú rámec využíva.

```
http://j.hrasko.org http://www.w3.org/2001/vcard-rdf/3.0#FN "Janko Hraško"
```

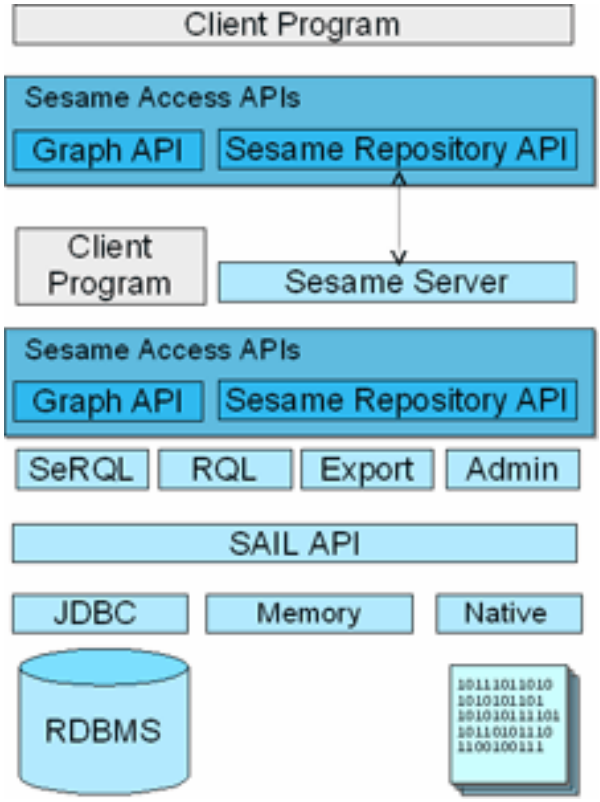
obr. 13 Príklad RDF trojice

Jena umožňuje jednoduchým spôsobom celé ontológie serializovať do XML súborov, či už vo formáte RDF Schema, alebo OWL, ale taktiež aj z týchto formátov ontológie načítať. Pri načítaní sa do objektu triedy *OntModel* vytvorí celá hierarchia ontológie so všetkými triedami, indivíduami, atď., čo následne umožňuje jednoduchú prácu s modelom, ako je jeho prechádzanie, upravovanie či rozširovanie. Uvedené triedy naviac poskytujú metódy na jednoduché dopytovanie. Rámec poskytuje okrem tohto jednoduchého dopytovania aj možnosť dopytovania jazykom RDQL, ktorý umožňuje vytvárať komplexné dopyty.

2.4.5 Sesame

Sesame (Sesame, 2005) je podobne ako Jena voľne dostupný rámec vytvorený v programovacom jazyku Java pre uchovávanie, dopytovanie a odvodzovanie. V prípade tohto rámca je možné využívať RDF a RDF Schema formáty. Môže byť použitý ako databáza alebo ako knižnica pre ďalšie aplikácie, ktoré potrebujú pracovať interne s RDF alebo RDF Schema. Na obr. 14 je znázornená architektúra rámca. Spodnú vrstvu rámca tvorí rozhranie *SAIL API*, ktoré je nezávislé od spôsobu uloženia údajov. Takto je možné údaje ukladať v relačnej databáze, pamäti, prípadne v natívnom súborovom formáte.

Nad týmto rozhraním sa nachádzajú moduly pre dopytovanie, export a administrátorský modul. Rámec takto poskytuje dopytovací jazyk na úrovni ako sémantickej (úroveň RDF Schema), tak aj na úrovni štrukturálnej (RDF). Prístup do uvedených modulov je zabezpečovaný prostredníctvom vrstvy *Sesame Access API*. Toto rozhranie využívajú samotné klientské programy v prípade, že Sesame používajú ako knižnicu, alebo ako *Sesame Server*, ktorý poskytuje funkcionalitu rámca prostredníctvom protokolu HTTP.



obr. 14 Architektúra rámca Sesame (Sesame, 2005)

2.5 Zhrnutie analýzy

V analýze sme sa venovali základom webu so sémantikou, jeho štruktúrou, ontológiám a jazykom na zápis ontológií. Načrtli sme problémy, ktoré spôsobujú zmeny v ontológiách, ako aj dôležitosť identifikácie týchto zmien. Nakoniec boli predstavené vybrané nástroje pre prácu s ontológiami. Z analýzy vyplýva, že problém identifikácie zmien medzi verziami ontológií je stále otvorený. Existuje niekoľko nevyriešených problémov ako je napr. identifikácia na štrukturálnej úrovni, porovnávanie hodnôt slotov s ohľadom na ich sémantiku. Analyzovali sme dva systémy na identifikáciu zmien, pričom jeden z nich (Prompt) využíva heuristiky, ale pri druhom (OntoView) sa nám nepodarilo zistiť algoritmy, s ktorými pracuje. Bolo by teda vhodné navrhnúť metódu na identifikáciu zmien medzi verziami ontológií ako na sémantickej, tak aj na štrukturálnej úrovni. Metóda musí zahŕňať identifikáciu rozdielov medzi triedami a vzťahmi medzi nimi a následne tieto využiť pri identifikácii rozdielov medzi indivíduami. Na úrovni indivíduí je nutné porovnávať hodnoty slotov a pokúsiť sa odhaliť sémanticky významné zmeny ich hodnôt. Taktiež bude potrebné navrhnúť spôsob, ako identifikované zmeny reprezentovať pre prípadnú ďalšiu distribúciu týchto zmien. Treba pritom vziať do úvahy aj možnosti analyzovaných jazykov pre zápis ontológií, ktoré istými prostriedkami disponujú.

3 Identifikácia zmien

Pri kontrole aktuálnosti informácií sa porovnávajú dve verzie. Medzi týmito dvoma verziami je potrebné identifikovať zmeny a identifikovať časti verzií, ktoré sú identické, t.j. určiť mapovanie týchto častí. Identifikáciu zmien medzi verziami je možné charakterizovať aj tak, že treba nájsť časti, ktoré boli upravené, časti, ktoré boli pridané a časti, ktoré boli odstránené.

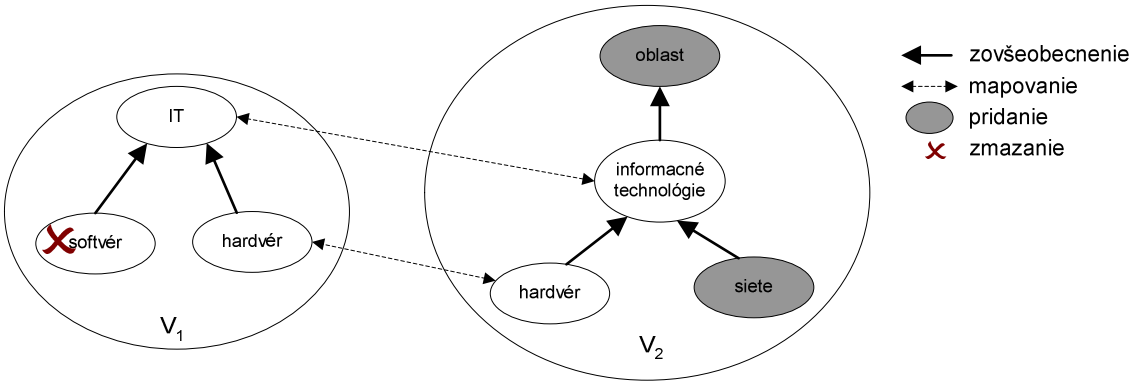
V prípade ontológií sa dá úloha identifikácie zmien rozdeliť na sémantickú a štrukturálnu úroveň. V prípade sémantickej úrovne sa kontroluje tá časť ontológie, ktorá definuje triedy, ich sloty, definičné obory, obory hodnôt, vzťahy medzi triedami, obmedzenia, atď. Štrukturálnu úroveň predstavujú inštancie/indivíduá tried definovaných na sémantickej úrovni. Kontrola na štrukturálnej úrovni zahŕňa nielen kontrolu konzistencie indivíduí s triedami, ktorých sú inštanciami, ale zahŕňa taktiež hodnoty indivíduí v kontexte tried a hodnoty ich slotov, ale takisto porovnanie indivíduí medzi dvoma verziami a identifikovanie rovnakých, upravených, pridaných a odstránených indivíduí.

Pri identifikácií zmien je možné využiť niekoľko metód, ktorými sa táto identifikácia dá realizovať:

- *hľadanie identických prvkov ontológií* – najjednoduchší spôsob založený na porovnávaní štruktúr, objektov, reťazcov a vyhľadávaní zhôd,
- *heuristické metódy* – dopĺňa predchádzajúcu metódu o prvky, pravidlá a algoritmy, ktoré na základe nedokonalej či neúplnej doplnkovej informácie vyvodzujú isté závery,
- *metódy výpočtovej inteligencie* – napr. neurónové siete.

Uvedené metódy umožňujú identifikovať rozdiely medzi dvoma verziami, pričom tieto rozdiely môžu mať rôznu povahu. Príklad identifikovaných zmien s využitím uvedených operácií je znázornený na obr. 15. Obrázok znázorňuje dve verzie ontológie na sémantickej úrovni. Plnými šípkami je znázornený vzťah medzi triedou a podtriedou, napr. vo verzii V_1 je trieda *hardvér* podtriedou triedy *IT*. Čiarkované šípky znázorňujú triedy, ktoré sú navzájom mapované, t.j. ich sémantika vo verziách je ekvivalentná, napr. trieda *hardvér* vo verzii V_1 je mapovaná na triedu *hardvér* vo verzii V_2. Môžeme si taktiež všimnúť, že mapovanie nemusí existovať len medzi triedami, ktoré sú identické, pričom príkladom takéhoto mapovanie je mapovanie medzi triedami *IT* vo

verzii V_1 a *informačné technológie* vo verzii V_2. Tieto triedy majú rovnakú sémantiku, ale ich názov je rozdielny. Trieda *softvér* vo verzii V_1 je označená krížikom, pretože je vzhľadom na verziu V_2 odstránená. Analogicky sú triedy *oblasť* a *siete* vo verzii V_2 vyznačené šedou farbou, pretože sú pridané vo verzii V_2 vzhľadom na verziu V_1.



obr. 15 Príklad identifikovaných zmien na sémantickej úrovni medzi dvoma verziami ontológie

Okrem už známych pojmov ako trieda, slot, indivíduum, apod. sme spomenuli aj niektoré ďalšie pojmy, ktorých význam si teraz zhrnieme, pretože ich budeme potrebovať pre ľahšie porozumenie:

- *element* – akákoľvek časť ontológie, ktorej zvyčajne hľadáme ekvivalent,
- *typ elementu* – bližšie určuje, o aký element sa jedná, t.j. určuje, či je element triedou, slotom, indivíduom, atď.,
- *ekvivalent* – element E_2 vo verzii V_2, ktorý má rovnakú sémantiku ako element E_1 vo verzii V_1.

Uvedený príklad nám umožňuje povedať, že zmeny, ktoré boli identifikované v predchádzajúcej úvahe, je možné definovať troma operáciami:

- *pridanie* – element sa nachádza vo verzii V_2 a súčasne sa nenachádza vo verzii V_1,
- *mapovanie* – určuje, ktorý element z verzie V_1 je sémanticky ekvivalentný elementu vo verzii V_2,
- *odobratie* – element sa nachádza vo verzii V_1 a súčasne sa nenachádza vo verzii V_2.

Tieto operácie môžu slúžiť na uchovávanie rozdielov medzi dvoma verziami ontológie. Samozrejmosťou je, že tieto operácie musia obsahovať údaje o tom, medzi ktorými dvoma verziami je operácia definovaná a taktiež aj údaje, na ktorú časť, resp. elementy verzie sa vzťahuje. V prípade operácií mapovania a stačí identifikovať konkrétnu triedu, slot, indivíduum, atď. V prípade operácie pridania, musí táto obsahovať aj všetky potrebné informácie, ktorými sa daný pridaný element dá definovať. Naviac môžu byť

opisy operácií doplnené aj o informácie o dôvodoch, autoroch, časovej platnosti a ďalšími prípadnými doplnkovými informáciami, ktorých hodnoty však bude musieť zadefinovať samotný používateľ.

Samotná reprezentácia zmien môže byť realizovaná na viacerých úrovniach:

- úrovni ontológie,
- úrovni jazyka, v ktorom je ontológia zapísaná,
- úroveni jazyka zmien, t.j. vytvorenie vlastného nezávislého formátu a teda aj úrovne, ktorá bude slúžiť len na účely reprezentácie zmien.

Reprezentácia zmien na úrovni ontológie predstavuje definovanie tried a slotov tak, aby sme priamo v jednotlivých indivíduách povedali, ktoré elementy ontológie sú v akom vzťahu s elementmi v inej verzii. Takéto definície sú súčasťou ontológie a nepriamo sa tak stávajú aj súčasťou modelovanej domény. Nevýhodou tohto prístupu je, že aj samotná definícia reprezentácie zmien môže podliehať zmenám a v podstate by sme potrebovali ďalší prostriedok na zisťovanie a reprezentáciu týchto zmien, prípadne by mohli vzniknúť neželateľné nekonečné cykly v rámci ontológie. Pre reprezentáciu zmien na úrovni jazyka je možné využiť napr. podporu mapovania v jazyku OWL. Toto mapovanie sa dá realizovať na sémantickej úrovni prostredníctvom atribútov *equivalentClass*, *equivalentProperty* a na štrukturálnej úrovni atribútom *sameAs*. Uvedené atribúty sú podporované od verzie OWL Lite. Tretia možnosť môže predstavovať nový formát opisujúci zmeny. V tomto prípade máme najväčšie možnosti, pretože nie sme prakticky nijak obmedzení. Môžeme tak vytvoriť formát založený na XML, prípadne nejaký binárny formát, aby sa dosiahla vyššia efektivita alebo využiť hierarchiu tried programovacieho jazyka a následne tieto triedy serializovať do inej podoby.

Proces identifikácie zmien môže prebiehať vo forme, že všetky elementy verzie V_1 (staršej verzie) prehlásime za odobraté a všetky elementy verzie V_2 (novšej verzie) za pridané. Následne sa snažíme nájsť mapovania medzi elementmi verzií, či už automatizovane, alebo ručne. Po procese identifikácie týchto mapovaní dospejeme do stavu, kedy v starej verzii budeme mať elementy odobraté a mapované a v novej verzii elementy mapované a pridané. Operácia mapovania medzi verziami ontológie v podstate vytvárajú graf a teda pri interpretácii tohto grafu treba byť opatrný, pretože nie je vylúčené, že v takomto grafe vznikne cyklus, čo môže spôsobiť problémy pri jeho interpretácii.

V nasledujúcich častiach opíšeme techniky a základné algoritmy, ktoré sme navrhli na identifikáciu zmien medzi dvoma verziami ontológie ako na sémantickej, tak aj na štrukturálnej úrovni. Pri tomto návrhu sme sa zamerali na hľadanie identických prvkov ontológií a heuristické metódy.

3.1 Sémantická úroveň

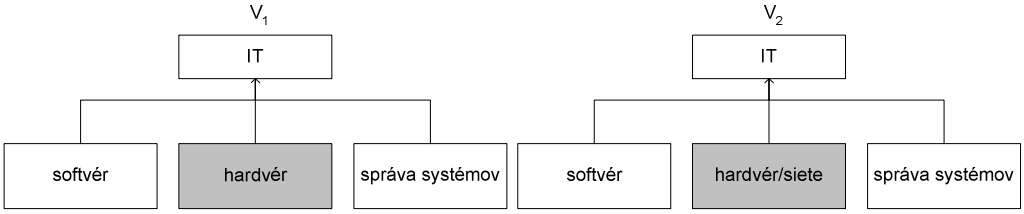
Úlohou kontroly na sémantickej úrovni je porovnať dve verzie ontológie, identifikovať jej rovnaké časti, t.j. rovnaké triedy, rovnaké vlastnosti tried, atď. a nájsť rozdiely, resp. zmeny. Tieto zmeny musia byť podrobené ďalšej analýze, pri ktorej sa musia nájsť prípadné vzťahy medzi oboma verziami, t.j. či ide o pridanie, odobratie, zmenu nejakej časti. Podobnosť medzi dvoma verziami sa dá vyjadriť ekvivalenciou elementov medzi nimi, t.j. vytvorí sa mapovanie medzi elementmi dvoch verzií ontológie a určí sa vzťah medzi týmito elementmi. V prípade pridaného alebo odstráneného elementu sa vytvorí opis tejto transformácie.

Za účelom identifikácie zmien sme navrhli 6 heuristík inšpirovaných podľa (Noy, 2003). Ako už vyplýva z povahy heuristík, môže byť identifikácia chybná. Môže sa napr. stať, že heuristika označí nejaké dve triedy ako ekvivalentné, ale ich vnútorná štruktúra nie je rovnaká. To by nebol ani tak veľký problém, pretože sa musia identifikovať aj zmeny vnútri týchto tried a tak zistíme tieto vnútorné rozdiely. Aj napriek tomu môže byť celková sémantika tried rozdielna. Ak sa už stane takýto prípad, bude musieť samotný používateľ rozhodnúť o tom, či dané triedy sú alebo nie sú ekvivalentné.

V nasledujúcich opisoch heuristík sú používané pre elementy názov E, pre verzie názov V a pre triedy názov C.

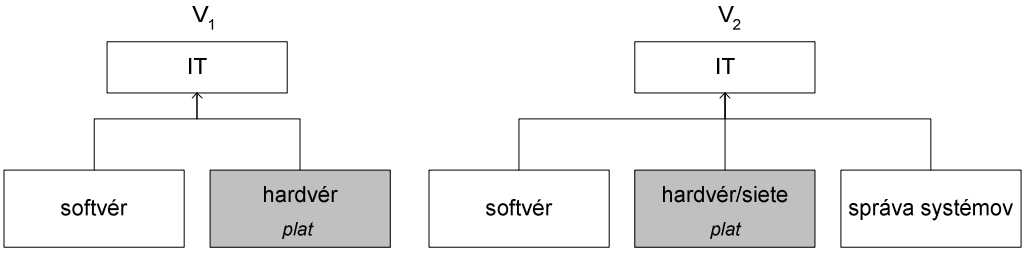
1. *Elementy s rovnakým názvom* – dva elementy $E_1 \in V_1$ a $E_2 \in V_2$ sú ekvivalentné, ak majú rovnaké meno a typ. Ide o štandardnú situáciu, kedy hľadáme základné mapovanie medzi dvoma verziami. Príklad takéhoto mapovania bol uvedený na obr. 15 medzi triedami *hardvér*.

2. *Jedna neekvivalentná podtrieda* – ak dve triedy $C_1 \in V_1$ a $C_2 \in V_2$, ktoré sú ekvivalentné, majú práve jednu podtriedu, ktorá nemá ekvivalent, potom tieto dve podtriedy označíme ako ekvivalentné. Táto situácia nastáva napr. v prípadoch, kedy sa zmení názov podtriedy, čo znázorňuje obr. 16, kde je vo verzii V_2 zmenený názov triedy *hardvér/siete* z pôvodného názvu *hardvér* vo verzii V_1. Ďalšia zjemnená verzia tejto heuristiky by mohla naviac porovnať, či sa názvy slotov týchto dvoch podtried rovnajú a až po splnení tejto podmienky by triedy prehlásila za ekvivalentné.



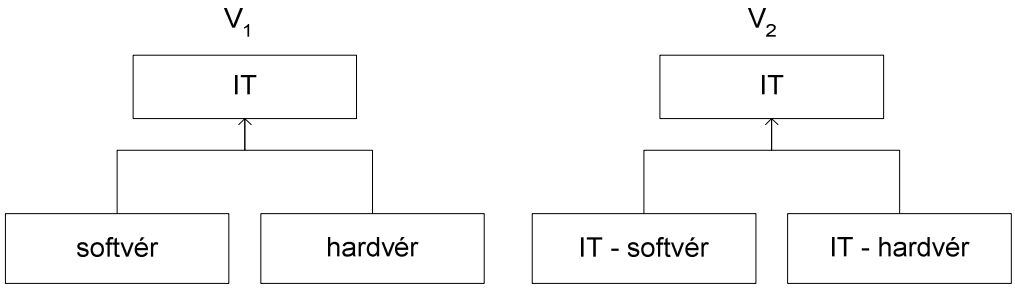
obr. 16 Príklad jednej neekvivalentnej podtriedy

3. *Viacero neekvivalentných podtried* – nech $C_1 \in V_1$ a $C_2 \in V_2$, C_1 a C_2 sú ekvivalentné, $subC_1$ je podtriedou C_1 a $subC_2$ je podtriedou C_2, $subC_1$ má všetky sloty ekvivalentné so $subC_2$, pričom tieto podtriedy majú sloty iné ako v ostatných podtried ich rodičovskej triedy, potom $subC_1$ a $subC_2$ sú ekvivalentné. Príklad tejto situácie znázorňuje obr. 17, kde bolo vo verzii V_2 pridaná triede *IT* nová podtrieda *správa systémov* a zároveň bol zmenený názov podtriedy z *hardvér* vo verzii V_1 na *hardvér/siete* vo verzii V_2. V takomto prípade má trieda *IT* dve neekvivalentné podtriedy. Keďže triedy *hardvér* vo verzii V_1 a *hardvér/siete* vo verzii V_2 majú sloty s rovnakými názvami, majú vlastne len iné meno a teda sú ekvivalentné. Z tejto úvahy vyplýva, že trieda *správa systémov* je novou triedou a teda pridaným elementom.



obr. 17 Príklad viacerých neekvivalentných elementov

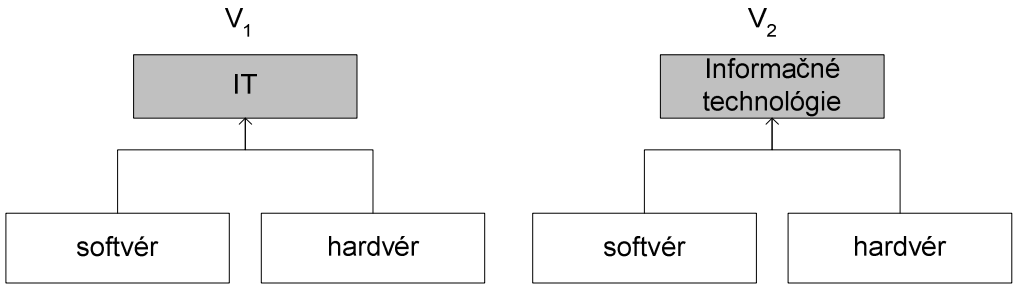
4. *Zmena názvov s rovnakým prefixom alebo sufixom* – ak $C_1 \in V_1$ a $C_2 \in V_2$ sú ekvivalentné a všetky podtriedy C_1 majú rovnaké názvy s podtriedami C_2 okrem konštantného prefixu alebo sufixu, potom podtriedy C_1 sú ekvivalentné podtriedam C_2. Na obr. 18 je znázornený prípad použitia uvedenej heuristiky, kde je obom podtriedam triedy *IT* vo verzii V_2 v porovnaní s verziou V_1 pridaný prefix „*IT* - ". Túto heuristiku je možné upraviť do množstva ďalších podôb, kedy sa posudzuje zmenený názov napr. na základe relatívneho porovnávania textov (pozri časť 3.3) alebo vyhľadaním názvov elementov v synonymickom slovníku.



obr. 18 Príklad zmeny názvu pridaním rovnakého sufixu

5. *Neekvivalentná rodičovská trieda* – ak $C_1 \in V_1$ a $C_2 \in V_2$ majú ekvivalentné všetky podtriedy, potom C_1 a C_2 sú ekvivalentné. Na obr. 19 je znázornený príklad, v ktorom sa názov rodičovskej triedy zmenil zo skratky (*IT*) na celý názov (*Informačné technológie*). Všetky jej podtriedy však zostali nezmenené

a je možné medzi nimi nájsť ekvivalenciu, takže aj ich rodičovské triedy možno prehlásiť za ekvivalentné.



obr. 19 Príklad neekvivalentnej rodičovskej triedy

6. *Jeden nezhodný slot* – ak $C_1 \in V_1$ a $C_2 \in V_2$ sú ekvivalentné a obe triedy majú práve jeden neekvivalentný slot, tieto sloty sú ekvivalentné. Takáto situácia nastáva v prípadoch, kedy sa zmení názov slotu v triede.

Podľa (Noy, 2003) sa uplatňuje najjednoduchšia heuristika „elementy s rovnakým názvom“ v 97,9 % prípadov. Na základe tohto čísla sa dá usúdiť, že ontológie sa v jednotlivých po sebe nasledujúcich verziách veľmi nemenia. Je to aj pozitívny fakt z pohľadu výkonnosti, pretože sa bude pri vykonávaní ostatných uvedených heuristík prehľadávať omnoho menšia časť verzie ontológie. Neznamená to však, že do ostatných heuristík bude vstupovať len zvyšná 2,1% časť ontológie. Dôvodom je fakt, že v niektorých heuristikách (napr. pri neekvivalentnej rodičovskej triede) je potrebné prehľadávať už mapované elementy, napr. za účelom zistenia všetkých mapovaných podtried. Dá sa však domnievať, že sa aj napriek tomuto faktu ostatné heuristiky radikálne zrýchlia po vykonaní vyhľadania elementov s rovnakým názvom, pretože budú stále pracovať s relatívne malou množinou elementov.

Veľký vplyv na úspešnosť správnej identifikácie zmien má nielen spôsob akým heuristiky určujú, či indivíduá sú ekvivalentné, ale aj to, v akom poradí sa tieto heuristiky na ontológiu aplikujú. Ako bolo spomenuté vyššie, heuristika „elementy s rovnakým názvom“ sa uplatňuje vo veľkej väčšine prípadov, takže ak ju aplikujeme ako prvú, množina tried, ktoré vstupujú do ďalších heuristík sa môže radikálne zúžiť. Ide teda o prístup, kedy sa čo najrýchlejšie snažíme obmedziť množinu elementov vstupujúcich do nasledovnej heuristiky. Tento prístup je efektívnejší ako opačný postup, kedy sa aplikujú najprv heuristiky, ktoré majú menšiu pravdepodobnosť, že sa uplatnia. Na druhej strane ale môže byť skoršie aplikovanie heuristík, ktoré berú do úvahy viacero atribútov spoľahlivejšie, pretože jemnejšie posudzujú prípadnú ekvivalenciu elementov. Tento prístup je nanešťastie omnoho neefektívnejší, pretože musíme porovnávať väčšiu množinu elementov.

Bolo by taktiež možné vytvoriť heuristiku, ktorá by nejakým spôsobom určovala ekvivalenciu tried na základe podobnosti indivíduí tried. Keďže môžeme predpokladať,

že indivíduí bude rádovo viacej ako tried, tento prístup bude opäť narážať na problém, že takáto heuristika by musela porovnávať podstatne väčšiu množinu elementov a pravdepodobne udržiavať aj veľké množstvo údajov o už analyzovaných elementoch.

Takisto je rozdiel, či sa v reťazi verzií V_1 až V_n porovnávajú postupne za sebou nasledujúce verzie a tak získame inkrementálne rozdiely medzi vzdialenými verziami napr. medzi V_1 a V_n ako v prípade, že priamo tieto dve verzie porovnáme. Je totiž predpoklad, že zmeny od prvej k poslednej verzii nastávajú postupne a medzi susednými verziami budú menšie rozdiely ako medzi verziami, ktoré priamo nenasledujú po sebe. Toto však vo všeobecnosti nemusí platiť a tak môžu byť algoritmy použité na zisťovanie rozdielov takisto ako v predchádzajúcom prípade, rôzne účinné.

Po identifikácii zmien na sémantickej úrovni sa dá pre potreby identifikácie zmien na úrovni štrukturálnej, určiť aj úroveň identifikovaných zmien. V prípade, že je element pridaný alebo odobratý, vieme presne určiť, čo na štrukturálnej úrovni má byť a čo nie. Ak napr. pridáme slot do triedy, vieme, že v novej verzii ontológie môžu mať indivíduá tejto triedy definované hodnoty nového slotu. Aby sme však vedeli ľahšie a efektívnejšie zmeny identifikovať, môžeme si určiť aj úroveň mapovania elementov. Táto úroveň predstavuje atribút operácie mapovania a určuje, či má daný element nejaké zmeny vnútornej štruktúry, alebo nie. Rozpoznávať sa tak dajú tri úrovne mapovania:

- *bez zmeny* – štruktúra elementu aj jeho meno je rovnaké v oboch verziách,
- *izomorfná zmena* – štruktúra elementu je v oboch verziách rovnaká, ale meno je v oboch verziách rôzne,
- *sémantická zmena* – je zmenená vnútorná definícia elementu.

Mapovanie slotov má významný dosah na hodnoty, ktoré obsahujú v jednotlivých verziách ontológie. Hodnota slotu totiž záleží nielen od významu, ale aj od spôsobu akým sa daná hodnota vyjadrí. Pokiaľ by sme teda mali slot definujúci napr. dĺžku, je rozdiel, či je dĺžka udávaná v míľach, kilometroch, stopách alebo palcoch. Preto je vhodné definovať ku každému mapovaniu slotov aj vzťahy medzi ich hodnotami napr. *transformačnou funkciou* z hodnoty v starej verzii na hodnotu v novej verzii.

Ďalšou otázkou zostáva, aké parametre transformačnej funkcii predať? V najjednoduchšom prípade by to mohli byť len hodnoty zo starej a novej verzie, ale tieto môžu závisieť od hodnôt ďalších slotov či už v indivíduu alebo v iných indivíduách (napr. určenie mernej jednotky v inom slote), pripadne od úplne inej časti ontológie (napr. či sa v nej nachádza nejaká trieda). Môže sa teda používateľovi ponúknuť možnosť určiť tri transformačné funkcie, prvú s predanými hodnotami, druhú s predanými hodnotami aj indivíduami a tretiu s predanými hodnotami, indivíduami a celou ontológiou, resp. odkazom na ňu.

V prípade, že prvá funkcia je definovaná a vie vykonať transformáciu na základe predaných hodnôt, využije sa táto transformácia a ostatné transformačné funkcie sa ignorujú, v opačnom prípade sa využije druhá. Ak je táto definovaná definovaná a vie transformovať hodnoty, využije sa táto transformácia a tretia sa ignoruje. V opačnom prípade sa využije tretia transformačná funkcia, opäť za podmienky, že je definovaná a vie transformovať hodnoty. Ak nie je definovaná žiadna z funkcií alebo nevedia transformovať hodnoty, hodnota sa jednoducho prekopíruje.

3.2 Štrukturálna úroveň

Úlohou kontroly na štrukturálnej úrovni je porovnať dve verzie ontológie, identifikovať jej rovnaké časti, t.j. rovnaké indivíduá, rovnaké hodnoty slotov v indivíduách, atď. a nájsť rozdiely, resp. zmeny medzi nimi. Analogicky ako na sémantickej úrovni, treba tieto zmeny podrobiť ďalšej analýze, pri ktorej sa musia nájsť prípadné vzťahy medzi oboma verziami, t.j. či ide o pridanie, odobratie, zmenu nejakej časti. Identifikácia zmien na štrukturálnej úrovni sa odráža od identifikovaných zmien na sémantickej úrovni. Základným elementom ontológie, ktorý vstupuje na tejto úrovni do procesu identifikácie zmien, je indivíduum triedy. Keďže vnútorná štruktúra indivídua vychádza z vnútornej štruktúry triedy, ktorej inštanciou je, musí táto identifikácia zmien odrážať identifikované zmeny na úrovni sémantickej. Takto sa dá dosiahnuť omnoho vyššia úspešnosť a aj efektivita identifikácie, pretože pri porovnávaní dvoch verzií ontológie vieme, ktoré triedy sú nové, ktoré sú odobraté a aj to, ktoré sú ekvivalentné.

Identifikácia zmien na štrukturálnej úrovni využíva niektoré heuristiky použité na sémantickej úrovni, aj keď mierne modifikované. Keďže indivíduá definujú hodnoty slotov tried, môžu tieto hodnoty obsahovať ako literál, tak aj URI na nejaký zdroj. Tu sa vynára jedna zo základných otázok pri porovnávaní takýchto hodnôt a vyhodnocovaní týchto porovnávaní. Je známe, že v prípade aj najmenšej chyby (preklepu) v zápise URI, sa zdroj stáva úplne nedostupný, prípadne je to odkaz na iný zdroj, ktorý nebol vôbec zamýšľaný (napr. pri preklepoch v parametroch predávaných metódou GET). V prípade literálov a najmä textových literálov je situácia opačná. V takomto prípade je jednoduchý preklep pre používateľa spravidla nezaujímavý a nevníma jeho opravu ako zmenu. Všimnúť si takúto chybu síce môže, ale v drvivej väčšine danému textu pochopí a porozumie. Z uvedeného dôvodu je dobré rozlišovať typ a význam porovnávaných údajov.

Vytvoriť mechanizmus na jednoduché porovnanie dvoch literálov je triviálna záležitosť a takéto prostriedky obsahuje prakticky každý dnes používaný programovací jazyk. Pri takomto porovnaní jednoducho vezmeme dva reťazce a zistíme, či sú zhodné. Výsledkom je informácia, či sú reťazce *zhodné*.

Iným mechanizmom môže byť spôsob porovnania, pri ktorom berieme do úvahy určitú prahovú hodnotu množstva rozdielov dvoch reťazcov. Pokiaľ množstvo rozdielov nestúpne nad túto definovanú hodnotu, môžeme oba reťazce označiť za *relatívne*

zhodné. Prahovú hodnotu je vhodné, aby mohol určiť používateľ a taktiež je vhodné presne vyznačiť pre potreby používateľa, na základe čoho systém usúdil, že dva reťazce sú relatívne zhodné alebo nie. Každopádne treba pre lepšiu efektivitu a spoľahlivosť stanoviť prahovú hodnotu experimentálne.

Otázkou ďalej zostáva, aká má byť granularita rozdielov, na základe ktorých sa bude rozhodovať o relatívnej zhodnosti. Sú to znaky, slová, odstavce? Je značný rozdiel, keď budeme porovnávať slová s prahovou hodnotou dva znaky a celé odstavce s rovnakou prahovou hodnotou. Najjednoduchším riešením by mohlo byť určenie počtu znakov. Ďalším aspektom je, ako „husto“ sa zmeny v texte nachádzajú, podľa čoho sa dá usúdiť aj ich závažnosť, ale napr. aj pridaná či odobratá diakritika v texte, prípadne citlivosť na malé a veľké písmená. Problém relatívneho porovnávania môže byť široký a závisí od typu textu a jazyka, v ktorom je text napísaný. Táto problematika spolu s načrtnutými riešeniami je bližšie opísaná v časti 3.3.

Aj na základe uvedených úvah môžeme dospieť k záveru, že úlohou identifikácie zmien na štrukturálnej úrovni verzie ontológie je nájsť indivíduá, ktoré boli pridané, odstránené a zmenené. Pritom treba zohľadniť aj zmeny sémantickej úrovne ako sú napr. zmeny názvov slotov, tried, pridanie slotu. Jednou z najdôležitejších úloh je identifikácia správneho mapovania ekvivalentných indivíduí medzi dvoma verziami. Pokiaľ dosiahneme takéto mapovanie, vieme určiť, ktoré sloty boli do akej miery zmenené, ktoré údaje sú aktuálne a vieme taktiež túto aktuálnosť aj overiť.

V predchádzajúcej časti bolo spomenuté, že identifikované zmeny je možné definovať troma operáciami – pridanie, mapovanie a odobratie. Proces identifikácie prebieha podobne ako na sémantickej úrovni, t.j. všetky indivíduá verzie V_1 (staršej verzie) prehlásime za odobraté a všetky indivíduá verzie V_2 (novšej verzie) za pridané. Následne sa snažíme nájsť mapovania medzi indivíduami verzií, či už automatizovane, alebo ručne. Po procese identifikácie týchto mapovaní dospejeme do stavu, kedy v starej verzii budeme mať indivíduá odobraté a mapované a v novej verzii indivíduá mapované a pridané. Pri tomto procese je pre lepšiu efektivitu užitočné prehľadávať len indivíduá tých tried, pre ktoré existuje mapovanie medzi verziami.

Pre identifikáciu mapovania indivíduí medzi dvoma ontológiami sú navrhnuté tri heuristiky. V nasledujúcich opisoch heuristík sú používané pre indivíduum názov I, pre verzie názov V a pre triedy názov C.

1. *Indivíduá s rovnakým názvom* – nech dve indivíduá I_1 je triedy C_1 a $C_1 \in V_1$ a I_2 je triedy C_2 a $C_2 \in V_2$ a C_1 s C_2 sú ekvivalentné. Potom I_1 a I_2 sú ekvivalentné, ak majú zhodné meno. Ide o štandardnú situáciu, kedy hľadáme mapovanie indivíduí medzi dvoma verziami. Táto heuristika sa nedá aplikovať v prípade anonymných indivíduí, pretože tie neobsahujú meno.

2. *Relatívne zhodné hodnoty slotov* – nech dve indivíduá I_1 je triedy C_1 a $C_1 \in V_1$ a I_2 je triedy C_2 a $C_2 \in V_2$ a C_1 s C_2 sú ekvivalentné. Potom I_1 a I_2 sú ekvivalentné, ak majú relatívne zhodné hodnoty slotov, ktoré majú mapovanie medzi triedami C_1 a C_2. Táto heuristika je vhodná na identifikovanie ekvivalentných anonymných indivíduí. Môže sa však stať, že triedy obsahujú príliš málo ekvivalentných slotov na to, aby sa dali indivíduá pomocou tejto heuristiky s dostatočnou mierou istoty určiť. Tento problém sa dá čiastočne riešiť tým, že sa určí minimálny počet slotov, ktoré triedy C_1 a C_2 musia mať namapované alebo minimálny počet definovaných hodnôt v indivíduách I_1 a I_2, aby sa táto heuristika aplikovala.

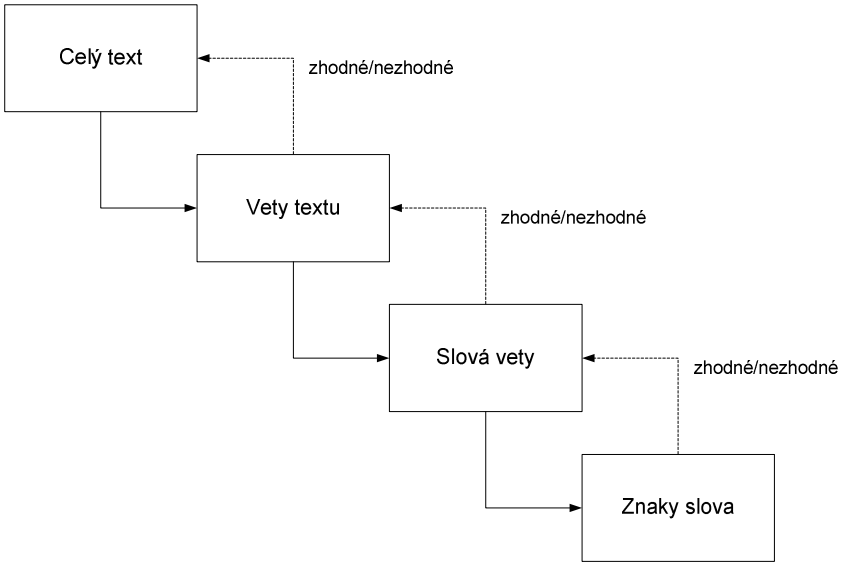
3. *Jedna nezhodná hodnota slotu* – nech dve indivíduá I_1 je triedy C_1 a $C_1 \in V_1$ a I_2 je triedy C_2 a $C_2 \in V_2$ a C_1 s C_2 sú ekvivalentné. Potom I_1 a I_2 sú ekvivalentné, ak majú absolútne zhodné hodnoty slotov, ktoré majú mapovanie medzi triedami C_1 a C_2 okrem jedného. Táto heuristika je vhodná na identifikovanie ekvivalentných anonymných indivíduí, ktoré majú zmenenú hodnotu práve jedného slotu. Podobne ako pri predchádzajúcej heuristike je vhodné definovať minimálny počet slotov, ktoré sa budú analyzovať.

Uvedené heuristiky slúžia na identifikáciu ekvivalencie indivíduí. Všetky mapované indivíduá môžu byť podrobené ďalšej analýze, kedy sa identifikujú zmeny na úrovní slotov, resp. ich hodnôt. Opäť sa analyzujú len sloty, ktoré majú identifikované mapovanie na sémantickej úrovni, lebo len pre takéto sloty vieme pri indivíduách získať hodnoty, ktoré môžeme porovnávať. Toto porovnávanie už neslúži na to, aby sme zistili rozdiely medzi ontológiami pre účely systému, ale pre účely používateľa, t.j. aby samotný používateľ vedel a prípadne aj videl, ktoré konkrétne údaje sú zmenené a čo je na nich zmenené.

3.3 Relatívne porovnávanie textov

Za účelom zisťovania relatívnej zhodnosti dvoch literálov je potrebné navrhnúť metódu, ako túto zhodnosť zistiť. Jej účelom je zistiť, ktoré zmeny textu sú pravdepodobne sémanticky významné. To by malo pre koncového používateľa znamenať, že pri zmenách textu, ktoré nie sú sémanticky významné ako napr. preklepy, nebude zaťažovaný tým, že bude takýto text v novej verzii prehlásený za sémanticky rozdielny oproti tomu v starej verzii. Metóda môže pracovať na úrovni znakov, slov, viet a celého textu, resp. ich kombináciou. Prvky, nad ktorými sa na jednotlivých úrovniach pracuje, t.j. znak, slovo, veta, text, označíme pre ďalšie použitie ako *element textu*. Základným princípom metódy je určenie, či dané literálne jednotky sú relatívne zhodné na základe definovaných prahových hodnôt, ktoré sa dotýkajú množstva syntaktických zmien, prípadne ich hustoty. V prípade, že texty nie sú zhodné, ale sú relatívne zhodné má význam taktiež zistiť rozdiely medzi týmito dvoma textami.

Postup pri určovaní relatívnej zhodnosti spočíva v počiatočnom porovnaní elementov textu algoritmom diff (Hunt, 1976). Aplikovaním tohto algoritmu medzi dvoma verziami vieme zistiť vektor rovnakých elementov, vektor odobratých elementov zo starej verzie a vektor pridaných elementov v novej verzii. Keďže algoritmus diff nevie v niektorých prípadoch identifikovať presunutý element, je pre niektoré typy elementov vhodné zisťovať medzi vektorom pridaných a vektorom odobratých elementov, či nejde len o prípad presunutého elementu. Toto je možné realizovať relatívnym porovnávaním elementov aj na nižšej úrovni ako to znázorňuje obr. 20, t.j. v prípade presunutej vety sa snažíme nájsť vetu tak, že zisťujeme, či neobsahuje zhodné slová medzi verziami, v prípade presunutého slova sa snažíme nájsť slovo s rovnakými znakmi, atď.



obr. 20 Porovnávanie na viacerých úrovniach metódou relatívneho porovnávania textov

V prípade úrovne celého textu je vhodné tento najprv dekomponovať na jednotlivé odstavce. Tento krok nie je síce bezprostredne nutný, ale má vplyv na zisťovanie presunutých viet. V prípade, že sa text dekomponuje na odstavce, presunuté vety sa budú vyhľadávať len v rámci tohto odstavca, v opačnom prípade sa presunuté vety vyhľadávajú v celom texte, čo má dopad na celkovú efektivitu, časovú a pamäťovú náročnosť. Z uvedeného dôvodu by mohla byť dostupná možnosť voľby, či sa text má dekomponovať na odstavce a každý odstavec analyzovať zvlášť, alebo analyzovať celý text ako celok. Problém dekompozície na odstavce sa dá riešiť vyhľadávaním znaku konca odstavca, ktorý predstavuje vlastne znak konca riadku. Pre dekomponovaný alebo celý text sa pokračuje následne analýzou na úrovni viet.

Pri analýze na úrovni viet potrebujeme dekomponovať text na jednotlivé vety. Táto operácia sa dá jednoducho vykonať hľadaním znamienok ukončujúcich vetu, t.j. bodka, otáznik a výkričník. Tu môže nastať problém, pretože napr. skratky obsahujú bodku, pričom sa nachádzajú v tele vety. Tento problém sa dá obísť zoznamom slov, ktoré obsahujú bodku, pričom sa pri výskyte bodky v texte skontroluje, či nejde o slovo

z uvedeného zoznamu, prípadne sa môže bodka v takýchto prípadoch nahradiť špeciálnym znakom alebo reťazcom. Ďalšou možnosťou je úplne ignorovať takýto prípad, pričom by sa veta obsahujúca podobné slovo považovala za viac viet. To však môže mať podobné následky ako v prípade dekompozície textu na odstavce, pretože presunuté slová budú vyhľadávané len v rámci vety. Ak sa teda nachádza slovo s bodkou uprostred vety a nastane presun slova z jednej polovice vety do druhej, nebude takýto presun identifikovaný.

Ako už bolo spomenuté vyššie, po dekomponovaní textu, prípadne odstavca, porovnáme vety algoritmom diff. Výsledkom porovnania budú vektor rovnakých viet, vektor odobratých viet zo starej verzie a vektor viet pridaných viet v novej verzii. Následne sa snažíme nájsť vetám z vektora odobratých viet relatívne zhodnú vetu z vektora pridaných viet. Najjednoduchšou cestou je porovnávať vety každú s každou dovtedy dokým sa nenájde relatívne zhodná veta alebo kým neprejdeme všetky vety. Do tohto porovnávania už však nezahrnieme vety, pre ktoré už relatívne zhodná bola nájdená. Výsledkom tohto vyhľadávania je množina mapovaní medzi odobratými a pridanými vetami.

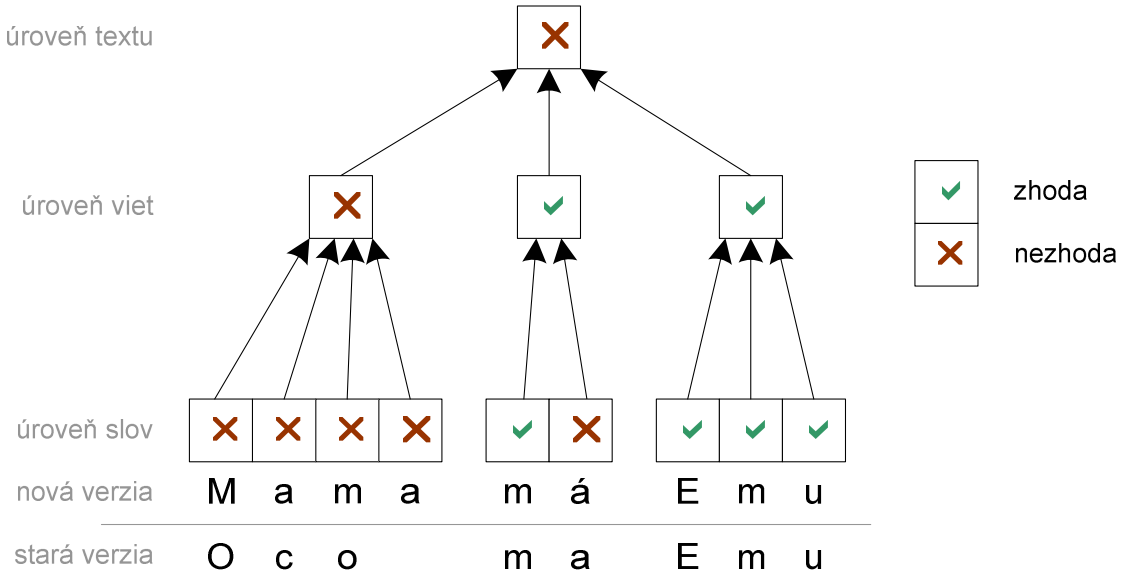
Relatívne porovnávanie viet prebieha podobne, ale s tým že sa analyzujú slová vety. Slová sú vo vetách oddelené medzerami, takže sa dajú jednoducho vo vete nájsť. Dekomponovaním vety na slová vzniknú dva vektory – slová starej vety a slová novej vety. Tieto dva vektory sa porovnajú algoritmom diff a podobne ako pri porovnávaní viet, sú výsledkom vektor rovnakých slov, vektor odobratých slov v starej vete a vektor pridaných slov v novej vete. Následne sa snažíme nájsť slovám z vektora odobratých slov relatívne zhodné slovo z vektora pridaných slov. Toto hľadanie sa taktiež realizuje relatívnym porovnávaním jednotlivých slov každého s každým, pričom slovo s už nájdeným relatívne zhodným slovom sa do ďalšieho hľadania nezahŕňa. Výsledkom tohto vyhľadávania je množina mapovaní medzi odobratými a pridanými vetami.

Jednotlivé slová sa medzi dvoma verziami relatívne porovnajú na úrovni znakov 1:1, čiže prvý znak s prvým znakom, druhý znak s druhým znakom, atď. Pri porovnávaní sa zaznamenáva početnosť zmien, t.j. pri nerovnosti znakov sa táto početnosť zvýši o jedna. V prípade, že sú slová rôznych dĺžok, sú znaky, ktoré sú navyše v jednom slove nerovné prislúchajúcim znakom v druhom slove (pretože neexistujú) a početnosť zmien sa tým pádom zväčší o tento rozdiel.

Na základe početnosti zmien znakov sa na úrovni slov odhadne, či dané slovo obsahuje sémantickú zmenu. Túto hodnotu nazvime *hodnotou rozhodnutia* na úrovni slova a keďže je binárna, môže nadobúdať hodnoty pravda alebo nepravda. Rozhodnutie sa realizuje na základe už spomínaných prahových hodnôt. Prahové hodnoty je vhodné nedefinovať všeobecne pre akúkoľvek dĺžku slova, ale v pásmach. Pásmo predstavuje rozmedzie dĺžok slov, pre ktoré platí daná prahová hodnota, napr. pre dĺžky slov menej ako 2 znaky vrátane bude prahová hodnota 100 % zmenených znakov, pre dĺžky slov

menej ako 5 znakov vrátane bude prahová hodnota 30 % a pre slová dlhšie ako 5 znakov 10 %. Je vhodné, aby existovala možnosť určiť tieto prahové hodnoty nielen ako relatívnu veličinu, ale aj ako absolútnu veličinu, t.j. v počte zmenených znakov. Vhodným doplnením určovania prahových hodnôt by mohlo byť vytváranie profilov pre jednotlivé druhy textu. Dosiahlo by sa tým to, že porovnávanie napr. technických textov alebo zákonov by mohlo byť omnoho citlivejšie nastavené ako napr. poézia či beletria, kde nemusia mať zmeny tak veľký dosah na význam. Uvedený spôsob rozhodovania počíta s istou nepresnosťou či odchýlkami, preto by bolo dobré experimentálne zistiť vhodné definície pásiem pre jednotlivé typy textov.

Po získaní hodnôt rozhodnutí, či jednotlivé slová obsahujú sémantickú zmenu alebo nie, posúdime relatívnu zhodnosť na úrovni viet. Táto sa realizuje podobne ako na úrovni slov. Z hodnôt na úrovni slov vieme zistiť nielen početnosť relatívnych zmien, ale vieme taktiež vytvoriť pole, ktoré nám popisuje hustotu týchto zmien. Najjednoduchšou možnosťou rozhodnutia o relatívnej zhodnosti dvoch viet je použiť pásma, podobne ako na úrovni slov. Analogicky môžeme urobiť rozhodnutie o sémantickej zhode aj na úrovni textu.



obr. 21 Príklad relatívneho porovnávania textov

Príklad porovnávania na jednotlivých úrovniach ilustruje obr. 21, kde je znázornené porovnávanie medzi dvoma verziami textu s jednou vetou. Rozdiel medzi verziami je v nahradení slova „oco“ zo starej verzie slovom „mama“ v novej verzii. V starej verzii sa taktiež nachádza preklep v slove „má“. Na úrovni slov sú v slove „mama“ zmeny vo všetkých znakoch a tak sa táto zmena premietne aj do úrovne viet. Naopak v prípade slova „má“ je nastavená prahová hladina pre pásmo, do ktorého toto slovo patrí na 100 % (pre definície pásiem uvedené vyššie) a tak je označené za relatívne zhodné. Slovo „Emu“ je úplne zhodné a teda je zhodné aj relatívne. Ak vezmeme do úvahy definíciu prahovej hodnoty jedného slova pre pásmo do 3 slov vrátane na úrovni viet, tak sa na

tejto úrovni dá prehlásiť, že daná veta nie je relatívne zhodná so starou verziou. Analogicky pri príslušnej prahovej hodnote sa aj pre celý text dá prehlásiť, že nie je relatívne zhodný so starou verziou.

Pri posudzovaní zhodnosti na úrovni znakov, resp. slov je vhodné rozlišovať, či slovo nie je číslom, pretože zmena znaku v čísle má spravidla väčší dosah na sémantiku ako zmena znaku v slove. Ak teda nájdeme slovo, ktoré pozostáva len z číslic, je vhodné pri zmene jedného znaku prehlásiť celé slovo za zmenené. To však platí len do desiatkovej číselnej sústavy a aj to len za podmienky, že budú pre zápis čísiel využité arabské číslice. Každopádne je to možné rozšírenie možností vytvárania spomínaných profilov textu.

Podobne by bolo užitočné uvažovať aj o rozlišovaní veľkých a malých písmen v slovách. Existujú slová, kde význam zmeny jedného veľkého písmena nemá podstatný význam na sémantiku, pretože takáto zmena môže byť považovaná len za gramatickú chybu (napr. bratislava namiesto Bratislava). Tento problém sa dá riešiť ignorovaním rozdielov medzi veľkými a malými znakmi, napr. prevedením znakov na malé a až po tomto prevode ich porovnávať. Na druhej strane existujú slová, kde podstatný dosah na sémantiku zmena veľkosti znaku má na sémantiku podstatný vplyv (napr. rozdiel medzi skratkami kilobajt – KB a kilobit – Kb). Tento prípad by sa dal riešiť rozšírením metódy o slovníky podobných slov. Jednoduchým kompromisom medzi týmito dvoma prípadmi môže byť použitie nižšej hodnoty pri vyčísľovaní početnosti zmien. Pri porovnávaní znakov by sa teda početnosť zmien nezvýšila o 1, ale napr. len 0,3 bodu. Presná hodnota by opäť mohla byť voliteľná v rámci profilu textu.

Veľký dopad môže mať v niektorých jazykoch fakt či sa porovnávajú verzie obsahujúce/neobsahujúce diakritiku. Problém s diakritikou sa dá riešiť podobne ako v prípade veľkých a malých znakov, t.j. znížením hodnoty pri výpočte početnosti zmien. Uvedená metóda pracuje na sémantickej úrovni a jej prípadné rozšírenie, pri ktorom by sa dala povýšiť o úroveň vyššie, by mohlo byť zapojenie synonymického slovníka do procesu porovnávania, kedy by sa mohla posudzovať skutočná podobnosť významu slov medzi verziami.

3.4 Reprezentácia zmien

Identifikované zmeny je nutné pre ďalšie použitie vhodne reprezentovať či už pre interné alebo externé použitie (napr. pre účely uchovávanie a distribuovania). V navrhnutých metódach identifikácie zmien ide vždy o vzťah medzi dvoma verziami ontológie, ale v závislosti od typu zmeny je potrebné uviesť aj daný element/elementy, ktorého sa opis zmeny týka/týkajú. Je teda potrebné pri popise zmien vedieť konkrétne dve verzie ontológie, ktorých sa týkajú zmeny a identifikácia elementu je následne možná prostredníctvom jeho názvu v ontológii.

V prípade, že sa snažíme reprezentovať zmeny v anonymných elementoch, vynára sa otázka, ako ich budeme pre tento účel identifikovať. V takomto prípade je možnosť identifikované elementy dodatočne pomenovať, pričom je potrebné, aby boli názvy unikátne. Tak by sa dosiahlo, že by sa dali elementy jednoznačne identifikovať. Spomeňme aspoň niekoľko spôsobov, ako je možné názvy jednoducho generovať:

- *sekvencia čísiel* – unikátnosť mena sa zabezpečí tým, že meno bude obsahovať číslo z radu čísiel, kde sa pre každý element inkrementuje aktuálna hodnota sekvencie. Tento prístup je analogický sekvenciám známym z relačných databázových systémov.
- *pamäťová adresa* – využije sa pamäťová adresa, od ktorej je reprezentácia elementu uložená v pamäti. Obmedzením tohto prístupu je, že unikátnosť adresy sa dá zabezpečiť unikátnosťou adresy a teda musí byť v pamäti v čase generovania názvov všetky elementy, ktorým chceme generovať názvy.
- *hashovacia funkcia* – využije sa hodnota hashovacej funkcie, ktorá sa generuje zo serializovanej podoby elementu spolu s napr. časom vytvorenia hashu.
- *pomenovanie z Jeny* – využijú sa mená, ktoré rámec Jena využíva pre interné účely. Tento prístup je možný len pri použití uvedeného rámca, ale na druhej strane je najmenej implementačne náročný.
- *elektronický podpis* – predstavuje možnosť podpísania napr. serializovanej podoby elementu čo naznačuje už architektúra webu so sémantikou na obr. 1. Tento prístup je pre naše potreby možno až príliš výpočtovo aj priestorovo náročný v porovnaní s predchádzajúcimi.

V závislosti od účelu, za ktorým je potrebné reprezentovať zmeny, je nutné pre niektoré typy zmien doplniť ďalšiu informáciu. Je totiž rozdiel, keď máme obidve verzie ontológie k dispozícii a následne potrebujeme len vedieť, aké sú vzťahy medzi nimi a je rozdiel, keď máme jednu z verzií a snažíme sa prostredníctvom opisu zmien získať druhú verziu. Reprezentáciu zmien, kedy máme k dispozícii starú verziu ontológie a chceme z nej získať novú verziu nazveme *doprednú reprezentáciu zmien*, reprezentáciu zmien, kedy máme k dispozícii novú verziu ontológie a chceme z nej získať starú verziu nazveme *spätná reprezentácia zmien* a reprezentáciu zmien, keď sú k dispozícii obe verzie ontológie nazveme *obojsmerná reprezentácia zmien*.

Všetky uvedené typy reprezentácií zmien sú vhodné pre distribučné účely v závislosti od potrieb danej aplikácie. Klientská aplikácia, ktorá používa starú verziu ontológie bude napr. požadovať, aby dostala všetky informácie pre vygenerovanie novej verzie ontológie zo starej. Vďaka doprednej reprezentácii zmien je možné vygenerovať napr. XML súbor obsahujúci všetky potrebné informácie, aby si klientská aplikácia vedela novú verziu „rekonštruovať“ zo starej verzie. Posledne uvedený typ je vhodný pre interné spracovanie a uchovávanie v rámci nástrojov na správu verzií, pretože

v takýchto systémoch spravidla máme k dispozícii všetky porovnávané verzie, naviac sa s ním ľahko pracuje a zo všetkých dostupných informácií je možné vygenerovať aj ostatné spomenuté typy. Ďalej si opíšeme jednotlivé typy zmien spolu s tým, ako sme navrhli reprezentáciu v uvedených prípadoch.

3.4.1 Dopredná reprezentácia zmien

Údaje potrebné pre doprednú reprezentácii zmien na sémantickej úrovni:

- *pridanie triedy* – úplná definícia pridanej triedy,
- *mapovanie tried* – názov triedy v starej verzii a definícia triedy v novej verzii[1],
- *odobratie triedy* – názov triedy v starej verzii,
- *pridanie slotu* – názov slotu v novej verzii (definícia slotu je obsiahnutá v mapovaní triedy),
- *mapovanie slotu* – názov slotu v starej verzii a názov slotu v novej verzii, vzťah medzi nimi definovaný transformačnou funkciou, ktorej budú vstupy predávané zo starej verzie (definícia slotu je obsiahnutá v mapovaní triedy),
- *odobratie slotu* – názov slotu v starej verzii.

Údaje potrebné pre doprednú reprezentácii zmien na štrukturálnej úrovni:

- *pridanie indivídua* – úplná definícia pridaného indivídua,
- *mapovanie indivídua* – názov indivídua v starej verzii a názov indivídua v novej verzii spolu s nezhodnými hodnotami slotov (sem patria aj tie, ktoré sú relatívne zhodné) medzi starou a novou verziou,
- *odobratie indivídua* – názov indivídua v starej verzii.

3.4.2 Spätná reprezentácia zmien

Údaje potrebné pre spätnú reprezentácii zmien na sémantickej úrovni:

- *pridanie triedy* – názov triedy v novej verzii,
- *mapovanie tried* – názov triedy v novej verzii a definícia triedy v starej verzii,
- *odobratie triedy* – úplná definícia odobranej triedy,
- *pridanie slotu* – názov slotu v novej verzii,

[1] V prípade, že ide o sémantickú zmenu (pozri úrovne mapovania v časti 3.1), je možné pridať len definície sémantických zmien v definícii triedy v novej verzii a v prípade izomorfnej zmeny stačí ak sa uvedie, že išlo len o zmenu názvu, pretože vnútorná štruktúra triedy je rovnaká. V našom prípade to však nemôžeme urobiť, pretože sme sa v práci nezaoberali všetkými zmenami v rámci tried t.j. zmenami dátových typov, ohraničení, atď, t.j. vlastností tried, ktoré sú závislé od jazyka, v ktorom je ontológia zapísaná. Z tohto dôvodu sú pridávané k mapovaniu aj celé definície mapovaných elementov.

- *mapovanie slotu* – názov slotu v starej verzii a názov slotu v novej verzii, vzťah medzi nimi definovaný transformačnou funkciou, ktorej budú vstupy predávané z novej verzie (definícia slotu je obsiahnutá v mapovaní triedy),
- *odobratie slotu* – názov slotu v starej verzii (definícia slotu je obsiahnutá v mapovaní triedy).

Údaje potrebné pre spätnú reprezentácii zmien na štrukturálnej úrovni:

- *pridanie indivídua* – názov indivídua v novej verzii,
- *mapovanie indivídua* – názov indivídua v starej verzii a názov indivídua v novej verzii spolu s nezhodnými hodnotami slotov (sem patria aj tie, ktoré sú relatívne zhodné) medzi starou a novou verziou,
- *odobratie indivídua* – úplná definícia odobraného indivídua.

3.4.3 Obojsmerná reprezentácia zmien

Údaje potrebné pre obojsmernú reprezentácii zmien na sémantickej úrovni:

- *pridanie triedy* – meno triedy v novej verzii,
- *mapovanie tried* – názov triedy v starej verzii a názov triedy v novej verzii,
- *odobratie triedy* – názov triedy v starej verzii,
- *pridanie slotu* – názov slotu v novej verzii
- *mapovanie slotu* – názov slotu v starej verzii a názov slotu v novej verzii a vzťah medzi nimi definovaný transformačnou funkciou (pre doprednú aj spätnú reprezentáciu),
- *odobratie slotu* – názov slotu v starej verzii.

Údaje potrebné pre obojsmernú reprezentácii zmien na štrukturálnej úrovni:

- *pridanie indivídua* – meno indivídua v novej verzii,
- *mapovanie indivídua* – názov indivídua v starej verzii a názov indivídua v novej verzii spolu voliteľne aj s nezhodnými hodnotami slotov (sem patria aj tie, ktoré sú relatívne zhodné) medzi starou a novou verziou,
- *odobratie indivídua* – názov indivídua v starej verzii.

Jednou z možností obojsmernej reprezentácie zmien je aj export do súboru OWL s danou verziou ontológie. Keďže jazyk OWL poskytuje len prostriedky pre mapovanie medzi elementami ontológie, sú pri použití tohto formátu zapísané len informácie o mapovaní a informácie o pridaní a odobratí elementov sa získajú implicitne, t.j. čo nie je v starej verzii mapované, je odstránené a čo nie je mapované v novej verzii, je pridané. Na uchovanie informácie o mapovaní triedy sa využíva atribút *equivalentClass*, na uchovanie informácie o mapovaní vlastnosti triedy je to atribút *equivalentProperty*

a na štrukturálnej úrovni na mapovanie indivíduí atribút *sameAs*. Ako hodnoty týchto atribútov sa uvádzajú URI na mapovaný element ontológie.

3.4.4 Hodnoty slotov

Jednou z vlastností relatívneho porovnávania textov je, že je ho možné použiť aj na vyznačovanie zmien v porovnávaných textoch. Mapované indivíduá môžu obsahovať zmenené hodnoty slotov aj v prípade, že sú relatívne zhodné, takže reprezentácia mapovania indivídua môže obsahovať práve aj hodnoty s vyznačenými zmenami oproti druhej verzii. V porovnávaných textoch je tak možné vyznačiť, ktoré vety a slová boli pridané v novej verzii, ktoré vety a slová boli odobraté v starej verzii, ktoré znaky v slovách boli zmenené a ktoré vety a slová sú na seba mapované.

Samotné vyznačovanie zmien je možné realizovať jednoducho vkladaním značiek priamo do textu, pričom sa naviac použije identifikátor mapovania, aby sa dali presne identifikovať mapované dvojice. Ako značkovací jazyk je ideálne použiť dobre známy XML. Technológie XML naviac poskytujú prostredníctvom transformácií jednoducho použiteľný a silný nástroj, ktorým je možné takto zapísané zmeny jednoducho prezentovať používateľovi.

Vyššie uvedená myšlienka relatívneho porovnávania textov umožňuje dokonca jednoduchým spôsobom spojiť oba porovnávané texty do jedného s tým, že rovnaké časti sa v ňom budú nachádzať iba raz, pričom následnou transformáciou je z neho možné vygenerovať ako starú, tak aj novú verziu hodnoty slotu. Takto zlúčené texty obohatené o značky je vhodné použiť pre reprezentáciu zmien hodnôt slotov uvedených v predchádzajúcich častiach.

3.5 Spájanie ontológií

Spojenie dvoch ontológií môže spočívať v princípe len jednoduchým prekopírovaním elementov z jednej ontológie do druhej. Pri takomto zlúčení prvkov však môže prísť k vzájomným konfliktom medzi elementmi, t.j. triedy s rovnakými názvami môžu mať inú sémantiku, triedy s rôznymi názvami môžu mať sémantiku rovnakú, dve indivíduá môžu reprezentovať ten istý objekt, atď. Jedným z využití navrhovanej metódy identifikácie zmien môže byť práve riešenie uvedených konfliktov, kedy medzi dvoma ontológiami[2] identifikujeme rovnaké a rozdielne časti.

Navrhovaná metóda identifikácie zmien v podstate hľadá ekvivalentné elementy medzi verziami ontológie a výstupom teda sú identifikované ekvivalentné a neekvivalentné časti ontológií. Neekvivalentné elementy môžu byť súčasťou novej ontológie tak ako sú, pričom ekvivalentné časti môžu byť do tejto novej ontológie vložené tak, ako sa nachádzajú v jednej z verzií prípadne v upravenej podobe, aby vyhovovali zámerom spojenia ontológií. Samotné úpravy elementu by už mala byť zodpovednosť

[2] V našom ponímaní to môžu byť dve rôzne verzie tej istej ontológie, pretože opisujú istú časť výslednej ontológie, ktorá vznikne po ich spojení.

používateľa, pretože pri týchto úpravách sa môžu zmeniť názvy elementov a ich vlastnosti, čo je silne závislý fakt od výsledného použitia ontológie. Takýmto prístupom by sa dve verzie elementu nahradili elementom jedným.

Ďalším prístupom by mohlo byť, že by výsledná ontológia pozostávala z oboch verzií elementu, pričom by sa využil jazyk na mapovanie rozdielov (pozri Ehrig, 2004), ktorý by opisoval vzťahy medzi elementmi tej istej verzie ontológie. Na mapovanie je možné s istými obmedzeniami využiť aj prostriedky jazyka OWL, pričom by sa tieto informácie stali priamo súčasťou ontológie. Problémom pri oboch uvedených prístupoch je zlučovanie a rozdeľovanie elementov, čo sa ale dá vyriešiť na úrovni mapovania, resp. jazyka na mapovanie rozdielov, alebo na úrovni samotnej ontológie, čo je ale menej systémový prístup. V prípade, že treba spájať viacej ako dve ontológie, je toto možné realizovať postupným spájaním, t.j. na začiatku sa spoja dve, k výsledku sa pridá tretia, atď.

4 Nástroj OntoDiff

Jedným z cieľov projektu je vytvorenie nástroja, ktorý dokáže automatizovane identifikovať zmeny, pričom taktiež poskytne prostriedky na manažment identifikovaných zmien medzi verziami ontológií. Táto časť sa zaoberá návrhom tohto nástroja. Hlavným cieľom návrhu bolo overiť metódy a princípy identifikácie zmien uvedené v predchádzajúcej kapitole, ale brali sme ohľad aj na tieto požadované vlastnosti aplikácie:

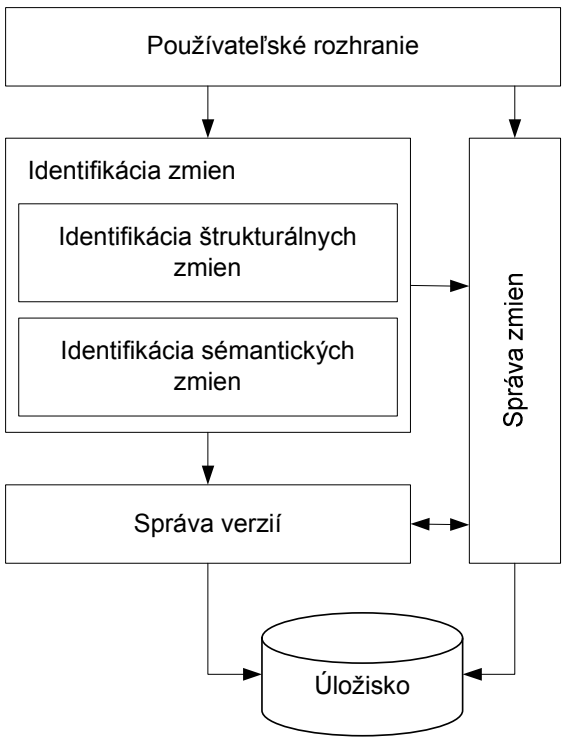
- vytvorenie úložiska pre rôzne ontológie a ich verzie,
- identifikácia a klasifikácia zmien medzi dvoma ľubovoľnými verziami ontológie – používateľovi sa umožňuje, aby mohol potvrdiť správnosť identifikácie zmien a v prípade, že identifikácia zmien bude nesprávna, má možnosť ručnej úpravy už identifikovanej zmeny,
- využitie relatívneho porovnávania textov pri porovnávaní hodnôt slotov indivíduí,
- možnosť nastavenia pásiem a prahových hodnôt relatívneho porovnávania textu – používateľ si bude môcť nastaviť prahovú úroveň citlivosti na zmeny napr. v prípade preklepov, pričom sa budú dať nastaviť rôzne pásma a prahové hodnoty pre úroveň slov, viet, celého textu,
- vizuálne vyznačovanie zmien medzi dvoma rôznymi verziami ontológie,
- uchovávanie metadát o verzii (číslo verzie, autor zmeny, popis zmeny, atď.),
- návrh reprezentácie identifikovaných zmien a vytvorenie úložiska pre tieto zmeny,
- možnosť exportu zmien s obojsmernou reprezentáciou (pozri časť 3.4.3) za účelom ich ďalšieho šírenia.

4.1 Architektúra systému

Požiadavky na softvérový systém, uvedené vyššie, sa odrážajú v hrubom návrhu z pohľadu modulov, ktorý je znázornený na obr. 22, kde šípky znázorňujú volanie funkcií modulov. Najnižšiu vrstvu systému tvorí *úložisko*. V tomto nie sú uložené len samotné verzie ontológií, ale aj informácie o ontológiách a verziách ontológií. Súčasne uchováva informácie o identifikovaných rozdieloch medzi verziami ontológií spolu s opisom zmien, autoroch zmien a časovej platnosti zmeny.

Modul *správy verzií* zabezpečuje prístup a prácu s úložiskom. Poskytuje funkcionalitu, ktorá umožňuje prechádzať a vytvoriť novú ontológiu a verziu ontológie. Verzia ontológie môže byť načítaná či už zo súboru, alebo zadaním URI, pričom táto vrstva zabezpečí, aby boli údaje správne načítané a uložené do úložiska. Ide o jednoduchého manažéra, ktorý má zjednodušiť prácu s ontológiami nielen v rámci systému, ale aj koncovému používateľovi.

Funkcionálne jadro systému predstavuje modul na *identifikáciu zmien*. Zabezpečuje identifikáciu a klasifikáciu zmien a mapovania medzi dvoma verziami ontológie. Kontroluje taktiež, či práve porovnávané verzie sú verziami jednej ontológie, aby sa tak zamedzilo porovnaniu verzií rôznych ontológií, ktoré spolu nesúvisia. Modul pozostáva z dvoch podmodulov – *identifikácia sémantických zmien* a *identifikácia štrukturálnych zmien*. Tieto zodpovedajú už spomínaným dvom úrovniam ontológie. Význam spolupráce oboch modulov sa zvyšuje faktom, že identifikácia sémantických zmien má veľký dopad na efektivitu identifikácie zmien na štrukturálnej úrovni, pretože identifikovať zmeny na štrukturálnej úrovni je omnoho zložitejšie bez poznania zmien sémantických. Modul umožňuje nielen automatizovanú identifikáciu, ale sprístupňuje taktiež možnosť ručnej identifikácie zmien alebo určenie mapovania medzi verziami ontológie. Tento modul úzko spolupracuje s modulom na správu verzií, pretože potrebuje pristupovať a porovnávať dve zvolené verzie ontológie. Modul identifikácie štrukturálnych zmien implementuje taktiež relatívne porovnávanie textov, ktoré využíva algoritmus diff.



obr. 22 Štruktúra systému z pohľadu modulov

Modul *správy zmien* zabezpečuje prístup a prácu so zmenami uloženými v úložisku. Pre správnu prácu so zmenami potrebuje spolupracovať s modulom na správu verzií, aby mohol presne určiť, medzi ktorými verziami je tá ktorá zmena identifikovaná. S týmto modulom spolupracuje taktiež modul na identifikáciu zmien, pretože ten potrebuje nielen ukladať identifikované zmeny a mapovania, ale potrebuje pristupovať k už identifikovaným zmenám, aby mohol identifikovať zmeny ďalšie.

Používateľské rozhranie umožňuje používateľovi spravovať uložené ontológie a jej verzie. Poskytuje používateľovi ako pohľad na dve verzie ontológie, tak aj jednoduchým spôsobom automatizovane a ručne identifikovať medzi nimi zmeny a mapovanie. Nájdené zmeny jednoducho umožňuje zobraziť vo forme stromu tried a zoznamu indivíduí. Taktiež umožňuje identifikované zmeny exportovať do XML súboru.

V nasledujúcich častiach opíšeme najdôležitejšie časti systému, algoritmy, ktoré sú použité, ako aj spôsoby, ako sú reprezentované údaje, ktoré systém používa.

4.2 Úložisko

Úložisko údajov aplikácie je možné realizovať viacerými spôsobmi. Okrem možnosti využitia súborového systému operačného systému je možné použiť aj relačnú databázu. Obe tieto riešenia je možné veľmi jednoducho použiť, ale ich efektivita a rýchlosť spracovania uložených dát je rôzna. Úložisko neuchováva len samotné verzie ontológií, ale aj informácie o zmenách medzi týmito verziami. Ako prvú vec treba navrhnúť spôsob ukladania verzií ontológií. Samotné vloženie verzie do systému je realizované prostredníctvom súboru vo formáte OWL, pričom je jedno, či je tento súbor vložený cez webový formulár alebo zadaním jeho URI.

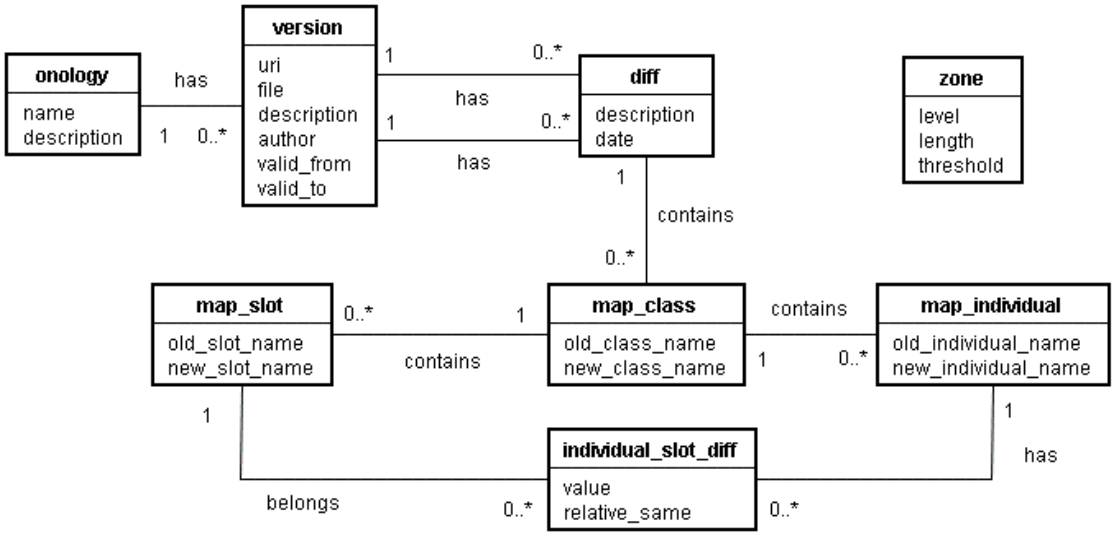
Súbor vložený do systému sa následne musí spracovať, t.j. analyzujú sa jednotlivé elementy ontológie. Toto môže zabezpečovať napr. rámec Jena, ktorý bol spomenutý v časti 2.4.4. Uvedený rámec poskytuje triedy a metódy na prácu s ontológiou, takže proces spracovania súboru je pre zvyšok aplikácie úplne transparentný. Systém tak pracuje už len s rozhraním, ktoré poskytuje Jena. Jena poskytuje uloženie údajov priamo do databázy, lenže táto funkcionalita nie je v aktuálnej verzii rámca Jena na požadovanej úrovni, a preto sme ju v našom systéme nevyužili. Verzie ontológie sa tak ukladajú do súborov vo formáte OWL a pri načítaní sa musia opäť spracovať. Spôsobí to síce spomalenie systému, ale toto riešenie je jednoduché a pre použitie prototypu úplne postačujúce. Naviac je ho možné jednoducho a kedykoľvek nahradiť riešením iným.

Systém ukladá verzie ontológie do spoločného adresára, ktorý má východiskový názov „*C:/OntoDiff Files*". V tomto adresári sa nachádzajú podadresáre ontológií, ktoré obsahujú súbory vo formáte OWL, reprezentujúce verzie ontológie. Tieto podadresáre majú názov *ontology-ID* kde *ID* je vnútorné identifikačné číslo ontológie. Toto číslo,

ako uvidíme neskôr, zodpovedá ID záznamu, ktorý popisuje ontológiu. V adresároch ontológie majú súbory verzií ontológie názvy *version-ID*, kde predstavuje *ID*, podobne ako v prípade ontológie, vnútorné identifikačné číslo verzie ontológie, ktoré zodpovedá ID záznamu, ktorý popisuje verziu ontológie.

Vnútorné údaje systému sú uchovávané v databáze. Navrhnutý model údajov je znázornený na obr. 23. Entita *ontology* reprezentuje ontológiu a obsahuje jej názov a popis. Tieto údaje by bolo možné zapísať aj do súboru ontológie, lenže vzhľadom na fakt, že ide o interné poznámky a aj to, že by sa mohli vo verziách meniť, sú uchovávané práve v tejto entite. Entita *version* reprezentuje verziu ontológie. Obsahuje popis, URI, prípadne aj názov súboru, kde je súbor s danou verziou ontológie v rámci úložiska uložený, autora verzie a jej časovú platnosť.

Entita *diff* je pomocnou entitou, reprezentujúcou zmeny medzi dvoma verziami jednej ontológie, preto obsahuje popis a dátum vytvorenia. Entita *map_class* reprezentuje mapovanie medzi dvoma triedami ontológie. Obsahuje meno triedy v starej verzii (*old_class_name*) a meno triedy v novej verzii (*new_class_name*). Keďže mapovanie jednej triedy môže byť v našej aplikácii najviac jedno, netreba názvy tried reprezentovať ako zvláštnu entitu. Entita *map_slot* reprezentuje mapovanie medzi dvoma slotmi, ktoré sa nachádzajú v dvoch mapovaných triedach. Analogicky ako v entite *map_class*, aj táto entita obsahuje názov starého slotu (*old_slot_name*) a názov nového slotu (*new_slot_name*). Keďže aj názov slotu je v rámci triedy unikátny a môže existovať maximálne práve jedno jeho mapovanie, nie je potrebné pre sloty vytvárať zvlášť entitu.



obr. 23 Logický model údajov systému

Entita *map_individual* reprezentuje mapovanie medzi dvoma indivíduami triedy. Obsahuje názov indivídua zo starej verzie (*old_individual_name*) a názov indivídua z novej verzie (*new_individual_name*). Výsledky porovnania hodnôt slotov indivídua sú

reprezentované entitou *individual_slot_diff*, ktorého atribút *value* obsahuje hodnoty oboch slotov v jednom reťazci s vyznačenými zmenami prostredníctvom XML značiek ako to bolo naznačené v časti 3.4.4 a bližšie popísané v časti 4.5.3. Hodnota atribútu *relative_same* obsahuje výstupnú hodnotu relatívneho porovnávania hodnôt slotov.

Entita *zone* reprezentuje jedno pásmo, ktoré sa používa pri relatívnom porovnávaní textov. Úroveň, na ktorej je pásmo definované je dané hodnotu atribútu *level* (T – text, S – veta, W – slovo), atribút *length* udáva šírku pásma, t.j. počet posudzovaných elementov textu (viet, slov, znakov) na danej úrovni, pre ktoré platí prahová hodnota daná atribútom *threshold*. Prahová hodnota vyjadruje relatívny počet zmenených elementov, t.j. pomer zmenené elementy/počet posudzovaných elementov, pre ktorý je text na danej úrovni ešte relatívne zhodný, t.j. ak je prahová hodnota menšia ako uvedený pomer, je text na danej úrovni relatívne zhodný.

4.3 Správa verzií

Modul správy verzií umožňuje pristupovať k verziám ontológie. Poskytuje rozhranie na prístup k jednej verzii, prípadne všetkým verziám danej ontológie, pričom sa verzie dajú určiť identifikačným číslom alebo URI. Modul ďalej realizuje vytvorenie novej verzie ontológie, pričom zabezpečuje jeho správne prenesenie od používateľa do úložiska. Pri spracovaní všetkých údajov, ktoré modul ukladá do úložiska, tento zabezpečuje aj integritu a platnosť údajov, t.j. overuje, či pri práci s dvoma verziami ontológie sa pracuje s verziami jednej ontológie v prípade, že je neprípustné, aby sa pracovalo s verziami dvoch odlišných ontológií, podobne na úrovni tried, slotov, atď.

4.4 Správa zmien

Modul na správu zmien slúži na jednoduchú manipuláciu s vytvorenými a uloženými zmenami a na tvorbu nových záznamov o zmenách, ktoré sú vytvorené modulom na identifikáciu zmien. Tento modul je v značnej miere využívaný pri identifikácii zmien, pretože je potrebné nielen vytvárať a rušiť záznamy o identifikovaných zmenách, ale zisťovať, ktoré zmeny už identifikované sú, aby bolo možné použiť navrhované heurisitky. Pri správe zmien je potrebné využívať aj správu verzií, pretože každá zmena sa viaže, na konkrétne dve verzie. Modul poskytuje správu na úrovni ako sémantickej, t.j. zmeny tried, slotov, tak aj na úrovni štrukturálnej, t.j. indivíduí a aj hodnôt slotov indivíduí.

4.5 Identifikácia zmien

Identifikácia zmien, ako je už načrtnuté na obr. 22, pozostáva z dvoch podmodulov – identifikácia sémantických zmien a identifikácia štrukturálnych zmien. Oba tieto moduly vychádzajú z postupov a heuristík, ktoré boli opísané v kapitole 3. V nasledujúcich častiach si opíšeme, ako tieto dva podmoduly implementujú navrhnuté heuristiky (pozri časti 3.1 a 3.2) a ako spolupracujú s ostatnými modulmi systému.

4.5.1 Identifikácia sémantických zmien

Identifikácia zmien prebieha medzi dvoma verziami tej istej ontológie, t.j. *medzi starou* a *novou verziou*. Na sémantickej úrovni je možné identifikovať zmeny buď automaticky za pomoci heuristík alebo ručne používateľom. Na začiatku sa v starej verzii prehlásia všetky elementy ontológie za odobraté a v novej verzii za pridané. Úlohou teda zostáva, nájsť mapovanie medzi elementmi oboch verzií. Vstupom pre túto identifikáciu sú dve verzie ontológie a výstupom sú identifikované mapovania medzi triedami a slotmi týchto dvoch verzií. Hľadanie mapovania prebieha v dvoch krokoch, t.j. najprv sa hľadá na úrovni tried a následne sa medzi namapovanými triedami hľadajú mapovania medzi slotmi týchto tried. Mapovania sa prostredníctvom modulu na správu zmien uložia do úložiska.

Pred samotnou aplikáciou heuristík sa najprv vytvoria interné zoznamy tried oboch verzií ontológie (*zoznam starých tried* a *zoznam nových tried*), pre ktoré sa má hľadať mapovanie. Do týchto zoznamov sa môžu vložiť ako len triedy, ktoré nemajú mapovanie, t.j. sú označené ako odstránené, resp. pridané, tak aj všetky triedy vo verzii ontológie. Nad takto pripravenými zoznamami už môžu byť aplikované heuristiky. V nasledujúcich častiach uvedieme algoritmy, ktorými sú realizované niektoré z vyššie uvedených heuristík. Treba poznamenať, že uvedené algoritmy nebrali do úvahy viacnásobné rodičovské triedy.

1. *Triedy s rovnakým názvom*
    1. pre každú triedu v zozname starých tried opakuj:
        2. zisti názov triedy zo zoznamu starých tried (*stará trieda*)
        3. pre každú triedu v zozname nových tried opakuj:
            4. zisti názov triedy zo zoznamu nových tried (*nová trieda*)
            5. ak sa názov starej triedy rovná názvu novej triedy, potom
                6. vytvor mapovanie medzi starou a novou triedou
                7. odstráň starú triedu zo zoznamu starých tried
                8. odstráň novú triedu zo zoznamu nových tried

2. *Jedna neekvivalentná podtrieda*
    1. pre každú triedu v zozname starých tried opakuj:
        2. zisti triedu zo zoznamu starých tried (*stará trieda*)
        3. zisti rodičovskú triedu starej triedy (*stará nadtrieda*)
        4. ak existuje nejaké mapovanie so starou nadtriedou, potom:
            5. zisti zoznam podtried starej nadtriedy (*zoznam starých podtried*)
            6. prehľadaj zoznam starých podtried a ak obsahuje práve jednu nenamapovanú triedu (*nenamapovaná stará trieda*), potom
                7. zisti triedu, na ktorú je mapovaná stará trieda (*nová nadtrieda*)

8. zisti zoznam podtried novej nadtriedy (*zoznam nových podtried*)
    9. prehľadaj zoznam nových podtried a ak obsahuje práve jednu nenamapovanú triedu (*nenamapovaná nová trieda*), potom
        10. vytvor mapovanie medzi nenamapovanou starou a nenamapovanou novou triedou
        11. odstráň nenamapovanú starú triedu zo zoznamu starých tried
        12. odstráň nenamapovanú novú triedu zo zoznamu nových tried

3. *Viacero neekvivalentných podtried*

    1. pre každú triedu v zozname starých tried opakuj:
        2. zisti názov triedy zo zoznamu starých tried (*stará trieda*)
        3. zisti rodičovskú triedu starej triedy (*stará nadtrieda*)
        4. ak existuje nejaké mapovanie so starou nadtriedou, potom:
            5. pre každú triedu v zozname nových tried opakuj:
                6. zisti názov triedy zo zoznamu nových tried (*nová trieda*)
                7. zisti rodičovskú triedy novej triedy (*nová nadtrieda*)
                8. ak sú stará nadtrieda a nová nadtrieda mapované, potom:
                    9. ak majú stará trieda a nová trieda rovnako nazvané sloty, potom:
                        10. vytvor mapovanie medzi starou a novou triedou
                        11. odstráň starú triedu zo zoznamu starých tried
                        12. odstráň novú triedu zo zoznamu nových tried

Po identifikácii zmien medzi triedami prebieha identifikácia zmien medzi slotmi. Tá prebieha veľmi podobne ako v prípade tried, t.j. na začiatku sa medzi dvoma namapovanými triedami prehlásia sloty v starej verzii za odstránené a sloty v novej verzii za pridané. Následne sa medzi týmito namapovanými triedami hľadá za pomoci heuristík mapovanie medzi slotmi. Pri mapovaní sa môžu uvažovať buď len sloty, ktoré nemajú mapovanie, t.j. sú označené ako odstránené, resp. pridané, tak aj všetky sloty konkrétnej triedy. V nasledujúcich častiach si ukážeme konkrétne algoritmy, ktorými sú realizované niektoré z heuristík.

1. *Sloty s rovnakým názvom*

    1. zisti zoznam mapovaní tried (*zoznam mapovaní*)
    2. pre každé mapovanie v zozname mapovaní opakuj:
        3. zisti mapovanie triedy zo zoznamu mapovaní (*mapovanie tried*)
        4. zisti starú triedu z mapovania tried (*stará trieda*)
        5. zisti novú triedu z mapovania tried (*nová trieda*)

6. zisti zoznam slotov určených na mapovanie zo starej triedy (*staré sloty*)
7. zisti zoznam slotov určených na mapovanie z novej triedy (*nové sloty*)
8. pre každý slot v zozname starých slotov opakuj:
    9. zisti názov slotu zo zoznamu starých slotov (*starý slot*)
    10. pre každý slot v zozname nových slotov opakuj:
        11. zisti názov slotu zo zoznamu nových slotov (*nový slot*)
        12. ak sa názov starého slotu rovná názvu nového slotu, potom
            11. vytvor mapovanie medzi starým a novým slotom
            12. odstráň starý slot zo zoznamu starých slotov
            13. odstráň nový slot zo zoznamu nových slotov

2. *Jeden nezhodný slot*

    1. zisti zoznam mapovaní tried (*zoznam mapovaní*)
    2. pre každé mapovanie v zozname mapovaní opakuj:
        3. zisti mapovanie triedy zo zoznamu mapovaní (*mapovanie tried*)
        4. zisti starú triedu z mapovania tried (*stará trieda*)
        5. zisti novú triedu z mapovania tried (*nová trieda*)
        6. zisti zoznam slotov určených na mapovanie zo starej triedy (*staré sloty*)
        7. zisti zoznam slotov určených na mapovanie z novej triedy (*nové sloty*)
            8. je v zozname starých slotov práve jeden slot a v zozname nových slotov práve jeden slot, potom
                9. vytvor mapovanie medzi starým a novým slotom
                10. odstráň starý slot zo zoznamu starých slotov
                11. odstráň nový slot zo zoznamu nových slotov

Vzhľadom na fakt, že tieto dva algoritmy používajú viaceré časti, ktoré majú spoločné, je možné ich implementovať do jedného tak, že kroky 8 až 11 z algoritmu „jeden nezhodný slot“ sa vykonajú hneď po algoritme „sloty s rovnakým názvom“.

4.5.2 Identifikácia štrukturálnych zmien

Identifikácia na štrukturálnej úrovni pokrýva v prvom rade správne identifikovať ekvivalentné indivíduá tried. Ostatné indivíduá môžeme potom pre konzistentnosť prehlásiť v starej verzii za odobraté a v novej verzii za pridané. Vo všeobecnosti môžeme proces identifikácie ekvivalentných indivíduí medzi verziami ontológie rozdeliť do dvoch častí – *identifikáciu neanonymných indivíduí* a *identifikáciu anonymných indivíduí*. Keďže pri mapovaní indivíduí je potrebné na uloženie potrebné identifikovať, t.j. pomenovať indivíduá, je potrebné vygenerovať pre anonymné indivíduá jednoznačný názov. Tento názov je vytvorený tak, že sa použije presne to

pomenovanie, ktoré pre svoje účely využíva Jena, tak ako to bolo uvedené v časti 3.4. Vstupom pre identifikáciu na tejto úrovni je zoznam mapovaní tried (*zoznam mapovaní*), medzi ktorými existuje mapovanie na sémantickej úrovni a výstupom sú identifikované mapovania medzi indivíduami týchto tried.

1. *Indivíduá s rovnakým názvom*
    1. pre každé mapovanie v zozname mapovaní opakuj:
        2. zisti mapovanie zo zoznamu mapovaní (*mapovanie*)
        3. zisti názov starej triedy z mapovania (*stará trieda*)
        4. zisti názov novej triedy z mapovania (*nová trieda*)
        5. zisti zoznam indivíduí starej triedy (*staré indivíduá*)
        6. zisti zoznam indivíduí novej triedy (*nové indivíduá*)
        7. pre každé indivíduum v zozname starých indivíduí opakuj:
            8. zisti názov indivídua zo zoznamu starých indivíduí (*staré indivíduum*)
            9. pre každé indivíduum v zozname nových indivíduí opakuj:
                10. zisti názov indivídua zo zoznamu nových indivíduí (*nové indivíduum*)
                11. ak sa názov starého indivídua rovná názvu nového indivídua, potom
                    12. vytvor mapovanie medzi starým a novým indivíduom
                    13. relatívne porovnaj mapované sloty medzi novou a starou triedou v starom a novom indivíduu
                    14. odstráň staré indivíduum zo zoznamu starých indivíduí
                    15. odstráň nové indivíduum zo zoznamu nových indivíduí

2. *Relatívne zhodné hodnoty slotov*
    1. pre každé mapovanie v zozname mapovaní opakuj:
        2. zisti mapovanie zo zoznamu mapovaní (*mapovanie tried*)
        3. zisti starú triedu z mapovania tried (*stará trieda*)
        4. zisti novú triedu z mapovania tried (*nová trieda*)
        5. zisti zoznam mapovaní slotov medzi slotmi starej a novej triedy (*mapovanie slotov*)
        6. ak je počet mapovaných slotov menší než minimálny počet nutných mapovaní slotov, pokračuj bodom 1
        7. zisti zoznam indivíduí starej triedy (*staré indivíduá*)
        8. zisti zoznam indivíduí novej triedy (*nové indivíduá*)
        9. pre každé indivíduum v zozname starých indivíduí opakuj:
            10. zisti indivíduum zo zoznamu starých indivíduí (*staré indivíduum*)
            11. pre každé indivíduum v zozname nových indivíduí opakuj:

12. zisti indivíduum zo zoznamu nových indivíduí (*nové indivíduum*)
13. pre každé mapovanie z mapovania slotov opakuj:
    14. zisti mapovanie slotu z mapovania slotov (*mapovanie slotu*)
    15. zisti starý slot z mapovania slotu (*starý slot*)
    16. zisti nový slot z mapovania slotu (*nový slot*)
    17. ak hodnota starého slotu v starom indivíduu nie je relatívne zhodná s hodnotou nového slotu v novom indivíduu pokračuj bodom 10
18. vytvor mapovanie medzi starým a novým indivíduom
19. relatívne porovnaj mapované sloty medzi novou a starou triedou v starom a novom indivíduu
20. odstráň staré indivíduum zo zoznamu starých indivíduí
21. odstráň nové indivíduum zo zoznamu nových indivíduí

3. Jedna nezhodná hodnota slotu

1. pre každé mapovanie v zozname mapovaní opakuj:
    2. zisti mapovanie zo zoznamu mapovaní (*mapovanie tried*)
    3. zisti starú triedu z mapovania tried (*stará trieda*)
    4. zisti novú triedu z mapovania tried (*nová trieda*)
    5. zisti zoznam mapovaní slotov medzi slotmi starej a novej triedy (*mapovanie slotov*)
    6. ak je počet mapovaných slotov menší než minimálny počet nutných mapovaní slotov, pokračuj bodom 1
    7. zisti zoznam indivíduí starej triedy (*staré indivíduá*)
    8. zisti zoznam indivíduí novej triedy (*nové indivíduá*)
    9. pre každé indivíduum v zozname starých indivíduí opakuj:
        10. zisti indivíduum zo zoznamu starých indivíduí (*staré indivíduum*)
        11. pre každé indivíduum v zozname nových indivíduí opakuj:
            12. zisti indivíduum zo zoznamu nových indivíduí (*nové indivíduum*)
            13. nastav príznak nájdenia nezhodného slotu na 0 (*príznak*)
            14. pre každé mapovanie z mapovania slotov opakuj:
                15. zisti mapovanie slotu z mapovania slotov (*mapovanie slotu*)
                16. zisti starý slot z mapovania slotu (*starý slot*)
                17. zisti nový slot z mapovania slotu (*nový slot*)
                18. ak hodnota starého slotu v starom indivíduu nie je absolútne zhodná s hodnotou nového slotu v novom indivíduu potom:

19. ak príznak je rovný 0 potom
    20. nastav príznak na 1
    21. inak pokračuj bodom 11
22. ak príznak sa rovná 1 potom:
    23. vytvor mapovanie medzi starým a novým indivíduom
    24. relatívne porovnaj mapované sloty medzi novou a starou triedou v starom a novom indivíduu
    25. odstráň staré indivíduum zo zoznamu starých indivíduí
    26. odstráň nové indivíduum zo zoznamu nových indivíduí

4.5.3 Relatívne porovnávanie textov

V algoritmoch identifikácie na štrukturálnej úrovni sa používa už spomínané relatívne porovnávanie. Účelom návrhu implementácie relatívneho porovnávania textov je overiť realizovateľnosť myšlienky relatívneho porovnávania textu uvedenú v časti 3.3 a zistiť jej prípadné nedostatky. Pre jednoduchosť návrh vychádza z definovania prahových hodnôt pre rôzne pásma (počtu elementov textu) na jednotlivých úrovniach textu, na základe ktorých sa rozhoduje o relatívnej zhodnosti. Toto sa odrazilo aj v modeli údajov, kde je definovaná entita *zone*, predstavujúca pásmo úrovne dané hodnotou atribútu *level*. Pre úroveň slov budú definované prahové hodnoty (atribút *threshold* entity *zone*) pásma pre počet nezhodných znakov, pre úroveň viet to bude počet nezhodných slov a pre úroveň celého textu to bude počet nezhodných viet. Možnú definíciu prahových hodnôt pásiem na úrovni slov znázorňuje obr. 24, pričom podobné definície je možné vytvoriť aj na úrovni viet a textu. Návrh poskytuje otvorenú možnosť doplnenia o posudzovanie relatívnej zhodnosti na základe váh typov zmien v jednotlivých literálnych jednotkách.

pásmo	prahová hodnota
2 znaky	*0,51*
5 znakov	*0,33*
10 znakov	*0,21*
< 10 znakov	*0,20*

obr. 24 Príklad definovania prahových hodnôt pásiem na úrovni slov

Jednou z požiadaviek na návrh je, aby systém nielen porovnával dva texty na relatívnu zhodnosť, ale taktiež vyznačil jednotlivé zmeny v texte pre používateľa s tým, že budú odlíšené typy zmien, t.j. inak bude zvýraznená zmena, ktorá ešte spadá do tolerancie danou prahovou hodnotou pri relatívnom porovnávaní a inak zvýraznená zmena, ktorá túto prahovú hodnotu prekračuje. Samozrejmosťou pri tom je, že by sa mali vyznačovať zmeny na všetkých úrovniach. V prípade implementácie v prostredí webu je toto vyznačovanie možné jednoducho implementovať použitím kaskádových štýlov.

Vstupom algoritmu je stará verzia textu (*starý text*), nová verzia textu (*nový text*) a definície pásiem s prislúchajúcimi prahovými hodnotami na jednotlivých úrovniach

(tieto definície sa skladajú z troch častí – pásiem a prahových hodnôt na úrovni textu, pásiem a prahových hodnôt na úrovni viet, pásiem a prahových hodnôt na úrovni slov). Výstupom je pozitívny výsledok v prípade relatívnej zhodnosti starého textu a nového textu, inak je výsledok negatívny. Výstupom algoritmu je taktiež vyznačenie zmien medzi starým a novým textom. Algoritmus pozostáva zo štyroch častí – porovnaj text, porovnaj vety, porovnaj slová a rozhodnutie.

1. *Porovnaj text*
    1. dekomponuj starý text na jednotlivé vety (*staré vety*)
    2. dekomponuj nový text na jednotlivé vety (*nové vety*)
    3. porovnaj staré a nové vety algoritmom diff[3]
    4. ak sú všetky staré a nové vety zhodné, ukonči s pozitívnym výsledkom bez vyznačenia zmien, t.j. nenastali žiadne zmeny
    5. zisti *odobraté vety* zo starých viet a *pridané vety* z nových viet
    6. pre každú vetu z odobratých viet opakuj (*i* je poradové číslo v zozname odobratých viet):
        7. pre každú vetu z pridaných viet opakuj (*j* je poradové číslo v zozname pridaných viet):
            8. relatívne porovnaj i-tu vetu z odobratých viet s j-tou vetou z pridaných viet – algoritmus *porovnaj vety*
            9. ak je výsledok porovnania v kroku 8 pozitívny:
                10. označ vety ako mapované
                11. z odobratých viet odstráň i-tu vetu
                12. z pridaných viet odstráň j-tu vetu
    13. zisti zostávajúci počet viet v odobratých vetách (*počet odobratých viet*)
    14. zisti zostávajúci počet viet v pridaných vetách (*počet pridaných viet*)
    15. definuj *počet zmien* ako väčšiu hodnotu z počtu odobratých viet a počtu pridaných viet
    16. zisti výsledok rozhodnutia pre počet starých viet, počet nových viet, počet zmien a pásiem a prahových hodnôt na úrovni textu – algoritmus *rozhodnutie*
    17. vráť výsledok rozhodnutia a vyznačené zmeny

Vstupom algoritmu porovnaj vety, ktorý je využívaný algoritmom porovnaj text, je stará verzia vety (*stará veta*), nová verzia vety (*nová veta*) a definície pásiem a prahových hodnôt na úrovni viet. Výstupom je pozitívny výsledok v prípade relatívnej zhodnosti starej a novej vety, inak je výsledok negatívny. Výstupom algoritmu je taktiež vyznačenie zmien medzi starou a novou vetou.

[3] Vstupom použitej implementácie algoritmu diff sú dve polia a výstupom je zreťazený zoznam štruktúr. Element tohto zreťazeného zoznamu pozostáva z hodnoty čísla elementu poľa novej verzie, počtu pridaných elementov v novej verzii od tohto čísla a čísla elementu poľa starej verzie a počtu odobratých elementov zo starej verzii od tohto čísla.

2. *Porovnaj vety*

    1. dekomponuj starú vetu na jednotlivé slová (*staré slová*)
    2. dekomponuj novú vetu na jednotlivé slová (*nové slová*)
    3. porovnaj staré a nové slová algoritmom diff
    4. ak sú všetky staré a nové slová zhodné, ukonči s pozitívnym výsledkom bez vyznačenia zmien
    5. zisti *odobraté slová* zo starých slov a *pridané slová* z nových slov
    6. pre každé slovo z odobratých slov opakuj (*i* je poradové číslo v zozname odobratých slov):
        7. pre každé slovo z pridaných slov opakuj (*j* je poradové číslo v zozname pridaných slov):
            8. relatívne porovnaj i-te slovo z odobratých slov s j-tym slovom z pridaných slov – algoritmus *porovnaj slová*
            9. ak je výsledok porovnania v kroku 8 pozitívny:
                10. označ slová ako mapované
                11. z odobratých slov odstráň i-te slovo
                12. z pridaných slov odstráň j-te slovo
    13. zisti zostávajúci počet slov v odobratých slovách (*počet odobratých slov*)
    14. zisti zostávajúci počet slov v pridaných slovách (*počet pridaných slov*)
    15. definuj *počet zmien* ako väčšiu hodnotu z počtu odobratých slov a počtu pridaných slov
    16. zisti výsledok rozhodnutia pre počet starých slov, počet nových slov, počet zmien a pásiem a prahových hodnôt na úrovni viet – algoritmus *rozhodnutie*
    17. vráť výsledok rozhodnutia a vyznačené zmeny

Vstupom algoritmu porovnaj slová, ktorý je využívaný algoritmom porovnaj vety, je stará verzia slova (*staré slovo*), nová verzia slova (*nové slovo*) a definície pásiem a prahových hodnôt na úrovni slov. Výstupom je pozitívny výsledok v prípade relatívnej zhodnosti starého a nového slova, inak je výsledok negatívny. Výstupom algoritmu je taktiež vyznačenie zmien medzi starým a novým slovom.

3. *Porovnaj slová*

    1. inicializuj *počet zmien* na 0
    2. definuj *počet iterácií* ako menšiu hodnotu z dĺžky starého slova a z dĺžky nového slova
    3. opakuj počet iterácií krát (*i* je číslo iterácie):
        4. ak i-ty znak starého slova nie je rovnaký ako i-ty znak nového slova zväčši počet zmien, inak vyznač zmenu v starom slove a novom slove

5. zväčši premennú počet zmien o hodnotu abs(dĺžka nového slova – dĺžka starého slova)
6. zisti výsledok rozhodnutia pre dĺžku starého slova, dĺžku nového slova, počet zmien a pásiem a prahových hodnôt na úrovni slov – algoritmus *rozhodnutie*
7. vráť výsledok rozhodnutia a vyznačené zmeny

V algoritmoch porovnaj text, porovnaj vety a porovnaj slová sa používa algoritmus *rozhodnutie* na určenie, či daný počet zmenených elementov spadá do relatívnej zhodnosti alebo nie. Ako jeho vstup sú *počet starých elementov*, *počet nových elementov*, *počet zmien* a definície *pásiem a prahových hodnôt*. Výstupom je negatívny výsledok v prípade, že je prahová hodnota menšia ako relatívny počet zmien v prislúchajúcom pásme, inak je výsledok pozitívny.

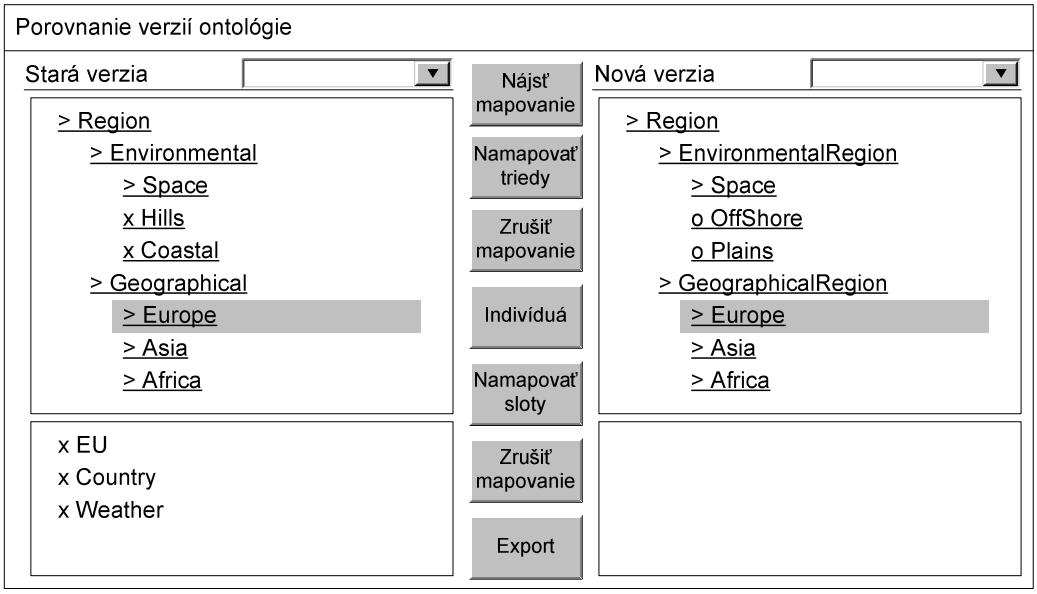
4. Rozhodnutie

1. definuj *dĺžku* ako väčšiu hodnotu z počtu starých elementov a počtu nových elementov
2. zisti *pásmo*, do ktorého spadá dĺžka z pásiem a prahových hodnôt
3. definuj *prahovú hodnotu* pre pásmo
4. definuj *určovacie číslo* ako pomer počtu zmien/dĺžka
5. ak je prahová hodnota menšia ako určovacie číslo, vráť negatívny výsledok, inak vráť pozitívny výsledok

4.6 Používateľské rozhranie

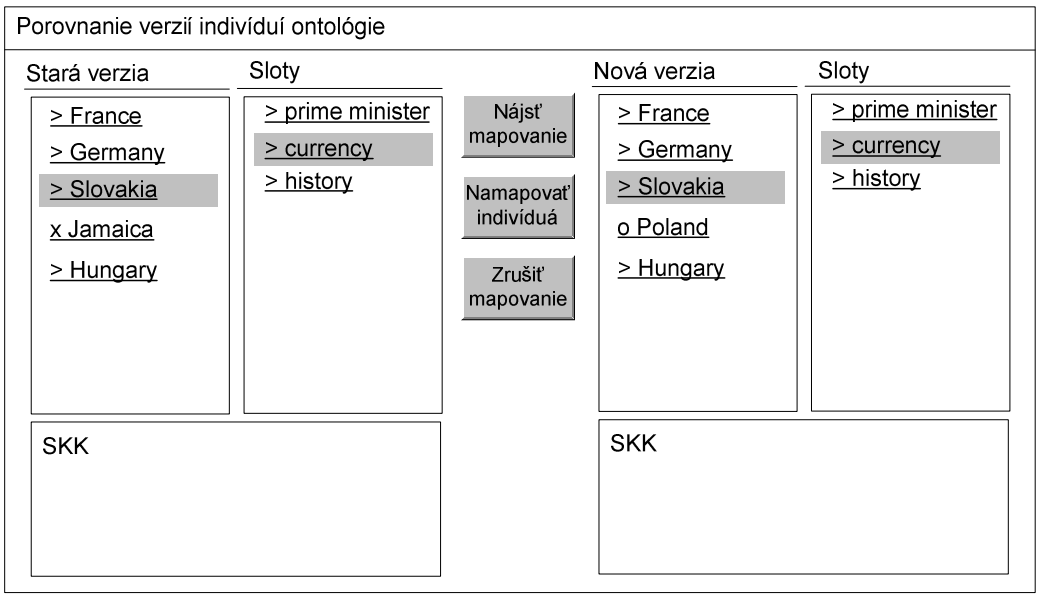
Používateľské rozhranie nástroja OntoDiff odráža navrhnuté metódy pre zisťovanie rozdielov a porovnávanie ontológií. Obrazovka porovnávania verzií ontológie na sémantickej úrovni je znázornená na obr. 25. Nachádzajú sa na nej dve verzie ontológie, pričom si používateľ môže vybrať iné verzie, ktoré chce porovnávať prostredníctvom vysúvacích zoznamov v hornej časti obrazovky. Pod nimi sa nachádza stromová hierarchia tried verzií ontológie a v spodnej časti sa nachádzajú zoznamy slotov zvolenej triedy.

Pred názvom triedy alebo slotu sa nachádza značka, ktorá identifikuje, či bol element pridaný (krúžok), odobratý (krížik) alebo je mapovaný (šípka). Ak používateľ klikne na mapovaný element, tento sa zvýrazní spolu s elementov v druhej verzii, na ktorý je mapovaný. V strednej časti obrazovky sa nachádza sada tlačidiel, ktorými môže používateľ určovať mapovania elementov. Tlačidlo *Export* umožňuje používateľovi exportovať identifikované zmeny do formátu systému OntoDiff, ktorý predstavuje obojsmernú reprezentáciu zmien spomínanú v časti 3.4.3. Tento formát je založený na XML, takže by nemal byť problém s jeho ďalším spracovaním. Špecifikácia tohto formátu je uvedená v prílohe C.



obr. 25 Návrh používateľského rozhrania – porovnávanie verzií na sémantickej úrovni

Po kliknutí na tlačidlo *Indivíduá* sa používateľovi zobrazí obrazovka porovnávania verzií ontológie na štrukturálnej úrovni znázornená na obr. 26, kde má možnosť porovnávať indivíduá a hodnoty ich slotov vybraných mapovaných tried. V hornej časti obrazovky sa nachádza zoznam indivíduí starej a novej verzie spolu so zoznamom slotov. Podobne ako na predchádzajúcej obrazovke aj tu sa po kliknutí na mapovaný element tento zvýrazní spolu s elementom v druhej verzii. Po kliknutí na nejaký názov slotu sa v spodnej časti zobrazí jeho hodnota a v prípade, že je to slot mapovaný, zobrazia sa aj rozdiely medzi dvoma verziami ako je to opísané v nasledujúcej časti. V strednej časti obrazovky sa nachádza sada tlačidiel, ktorými môže používateľ určovať mapovania indivíduí.



obr. 26 Návrh používateľského rozhrania – porovnávanie verzií na štrukturálnej úrovni

4.6.1 Prezentácia zmien hodnôt slotov

Algoritmy uvedené v časti 4.5.3 umožňujú „zlúčenie“ dvoch verzií textov do jedného, kde je vyznačené XML značkami ktoré časti sú pridané, zmenené, mapované a ktoré zostali bez zmeny a sú rovnaké v oboch verziách. Takto vyznačené texty sú ukladané aj do úložiska, aby sa mohli jednoduchšie prezentovať a exportovať. Konkrétna definícia exportného formátu, ktorého časti sa používajú aj na ukladanie hodnôt slotov je ďalej opísaná v prílohe C. Ako už bolo spomenuté, je možné zmeny prezentovať spôsobom, že budú rôzne farebne odlíšené, t.j. odstránený text môže byť vyznačený červenou a prečiarknutý, pridaný text môže byť modrou a podčiarknutý, mapovaný text môže byť na farebnom pozadí, pričom iným odtieňom bude znázornený na úrovni viet a inou na úrovni slov, atď. Tieto vyznačené zmeny je možné jednoduchou XML transformáciou pretransformovať do HTML formátu, kde za pomoci kaskádových štýlov môžu byť spomenuté zvýraznenia textu realizované.

Na obr. 27 je znázornený výstup relatívneho porovnania textov vo formáte XML. Po pretransformovaní dvoma šablónami pre starú a novú verziu dostaneme výslednú starú verziu textu s vyznačenými zmenami znázornenú na obr. 28 vľavo a novú verziu textu s vyznačenými zmenami znázornenú na obr. 28 vpravo. Na poslednom spomenutom obrázku predstavuje text na šedom pozadí mapované vety v rámci ktorých sú na tmavom pozadí mapované slová s vyznačenými zmenenými znakmi. Prečiarknutý text znamená odstránený text a podčiarknutý text je pridaný vzhľadom na novú verziu. Takýmto spôsobom je jednoduchou úpravou šablóny možné znázorniť akúkoľvek časť textu v podstate akýmkoľvek spôsobom, či už vo formáte HTML alebo inom.

```
<relativeDiff>
<mapSentOld id="25"><mapWordOld id="2">Dne<change id="0">s</change>ný</mapWordOld>
web, <deleteWord>ktorý sa od svojho vzniku rozrástol do nepoznateľnej podoby,</deleteWord>
poskytuje <mapWordOld id="24">mno<change id="22">z</change>stvo</mapWordOld> nielen
textových informácií, ale podáva svojim konzumentom taktiež multimediálny
obsah.</mapSentOld><mapSentNew id="25"> <mapWordNew id="2">Dne<change
id="0">š</change>ný</mapWordNew> web, poskytuje <mapWordNew id="24">mno<change
id="22">ž</change>stvo</mapWordNew> nielen textových informácií, ale podáva svojim
konzumentom taktiež multimediálny obsah.</mapSentNew> Práve vďaka pridaniu tohto
multimediálneho obsahu, sa z pôvodného hypertextu stalo hypermédium.<mapSentOld id="180">
Množina textových, obrazových a zvukových informácií spolu s ich vzájomnými vzťahmi vytvára
hyperpriestor.</mapSentOld><addSent> Súčasný web má jedno zásadné
obmedzenie.</addSent><mapSentNew id="180">  Množina <addWord>všetkých</addWord>
textových, obrazových a zvukových informácií spolu s ich vzájomnými vzťahmi vytvára
hyperpriestor.</mapSentNew>
</relativeDiff>
```

obr. 27 Výstup relatívneho porovnávania starej a novej verzie

Uvedený spôsob prezentácie hodnôt slotov indivíduí môže byť použitý napr. v obrazovke porovnávania verzií na štrukturálnej úrovni znázornenej na obr. 26. Takto má používateľ prehľadne vedľa seba obe verzie textu a môže jednoducho vidieť a porovnať, čo sa v texte zmenilo a čo nie. Pre lepšiu prehľadnosť môže byť tento

pohľad doplnený aj o ďalšiu vlastnosť, kedy po prejdení kurzorom myši cez nejaký mapovaný element, sa tento dodatočne zvýrazní v oboch verziách textu. Pri implementovaní tejto vlastnosti sa dá použiť identifikátor v každej značke mapovania elementov.

Dnesný web, ~~ktorý sa od svojho vzniku rozrástol do nepoznateľnej podoby,~~ poskytuje mnozstvo nielen textových informácií, ale podáva svojim konzumentom taktiež multimediálny obsah. Práve vďaka pridaniu tohto multimediálneho obsahu, sa z pôvodného hypertextu stalo hypermédium. Množina textových, obrazových a zvukových informácií spolu s ich vzájomnými vzťahmi vytvára hyperpriestor.	Dnešný web, poskytuje množstvo nielen textových informácií, ale podáva svojim konzumentom taktiež multimediálny obsah. Práve vďaka pridaniu tohto multimediálneho obsahu, sa z pôvodného hypertextu stalo hypermédium. Súčasný web má jedno zásadné obmedzenie.Množina všetkých textových, obrazových a zvukových informácií spolu s ich vzájomnými vzťahmi vytvára hyperpriestor.

obr. 28 Stará a nová verzia textu s vyznačenými zmenami

5 Overenie návrhu

Za účelom overenia návrhu systému OntoDiff bol vytvorený prototyp, ktorého cieľom bolo implementovať kritické časti systému, overiť ich funkčnosť a úspešnosť na niekoľkých vzorkách ontológií. Výsledný prototyp teda musí spĺňať predovšetkým tieto ciele:

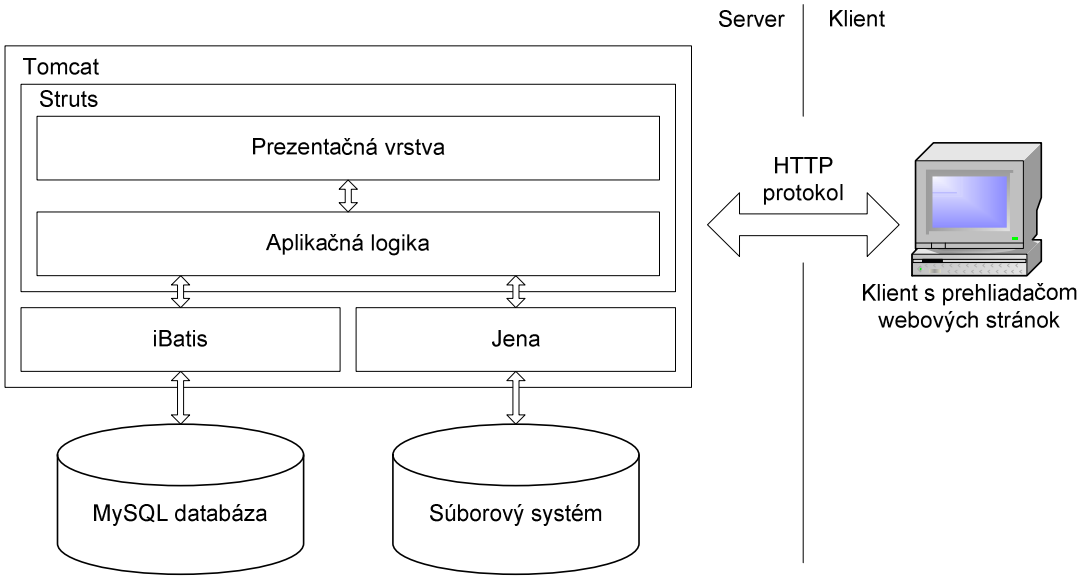
- implementácia modulu úložiska a rozhraní pre prácu s ním,
- implementácia modulu pre správu verzií,
- implementácia modulu pre správu zmien,
- implementácia modulu identifikácie sémantických zmien medzi dvoma verziami ontológie s využitím algoritmov z časti 4.5.1,
- implementácia modulu identifikácie štrukturálnych zmien medzi dvoma verziami ontológie s využitím algoritmov z časti 4.5.2,
- implementácia relatívneho porovnávania textov,
- implementácia vhodnej prezentácie identifikovaných zmien,
- vytvorenie jednoduchého a intuitívneho grafického používateľského prostredia ako webovej aplikácie,
- implementácia exportu identifikovaných zmien v obojsmernej reprezentácii.

5.1 Implementácia

Prototyp bol implementovaný ako webová aplikácia na platforme J2EE (J2EE, 2005), pričom bolo použité vývojové prostredie IntelliJ IDEA 4.5 (Idea, 2005). Pri implementácii bolo využitých niekoľko rámcov a komponentov pre jazyk Java, ktoré zásadným spôsobom uľahčili prácu. Architektúra prototypu, kde je možné taktiež vidieť použitie týchto komponentov, je znázornená na obr. 29, pričom podrobná programátorská dokumentácia je vygenerovaná nástrojom JavaDoc, dostupná na priloženom médiu v adresári */OntoDiff/javadoc* a v prílohe B.

Systém vychádza z architektúry klient/server. Ako aplikačný server využíva Apache Tomcat 5.0.28 (Tomcat, 2005), ale vďaka použitým technológiám je nasaditeľný aj na inom aplikačnom serveri. Bol použitý rámec Apache Struts (Struts, 2005), čo je open source rámec pre tvorbu webových aplikácií na platforme Java, pričom využíva

architektonický vzor Model-View-Controller. Vo vnútri rámca je ako model vytvorená samotná aplikačná logika opísaná v predchádzajúcej kapitole a aj prezentačná vrstva (ako view), ktorá zabezpečuje používateľské prostredie, t.j. vstup údajov od používateľa, ako aj výstup údajov. Ako úložisko je využitý súborový systém operačného systému na ukladanie súborov s verziami ontológií. Tieto súbory sú ukladané podľa pravidiel uvedených v časti 4.2. Fyzický model údajov sa nachádza v prílohe B a je údajovým základom pre moduly na správu verzií a zmien, pričom sa využíva databázový server MySQL 4.0 (MySQL, 2005). Pre jednoduchšiu prácu s databázou je využitý rámec iBatis (iBatis, 2005), ktorý predstavuje nástroj na mapovanie medzi záznamami v tabuľkách databázy a JavaBean objektmi. Na prácu s ontológiami sa využíva rámec Jena a pri relatívnom porovnávaní textov bola použitá GNU implemetácia algoritmu diff pre Javu (Diff, 2005). Na XML transformácie bol použitý XSLT procesor Apache Xalan (Xalan, 2005).



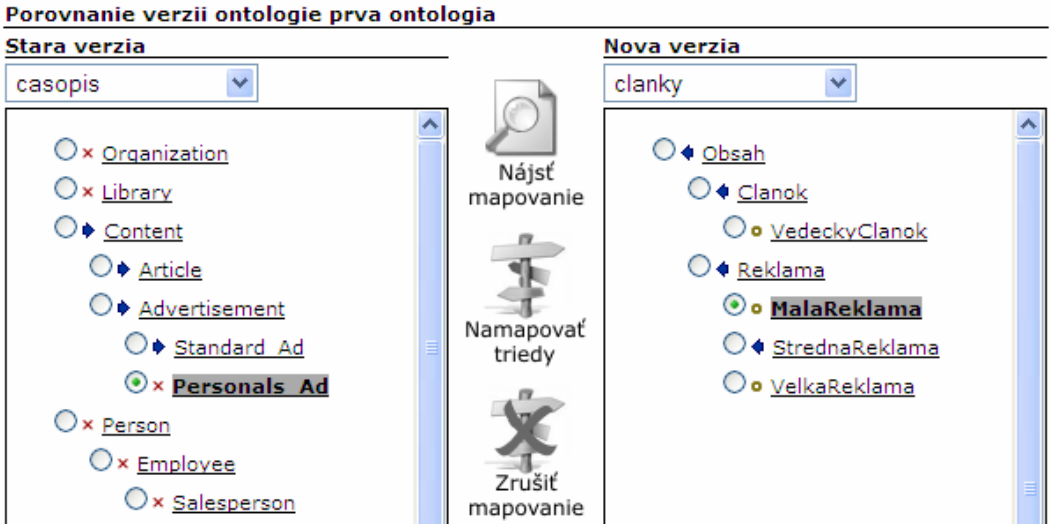
obr. 29 Architektúra prototypu

5.2 Zhodnotenie prototypu

Ako bolo spomenuté, prototyp bol vytvorený pre overenie kritických častí systému, ktoré predstavujú aj jeho jadro, t.j. cieľom bolo overenie realizovateľnosti navrhnutých algoritmov identifikácie zmien. Týmito algoritmami sa overila úspešnosť heuristík na identifikáciu sémantických zmien ontológií, spomenutých v časti 3.1. Pri testovaní prototypu boli použité tri ontológie, pričom každá z nich obsahovala 5 až 15 verzií s rôznymi typmi zmien a v rôznom rozsahu. Išlo o ontológiu regiónov sveta, o ontológiu pracovných ponúk a o ontológiu časopisu (túto je možné nájsť aj na priloženom elektronickom nosiči).

Vo všeobecnosti sa dajú potvrdiť očakávané výsledky, pretože systém s mimoriadne veľkou spoľahlivosťou identifikoval menšie a stredne veľké zmeny ontológie, kedy nenastali nejaké zásadné presuny a súčasne aj premenovania celých častí či vetiev

v štruktúre tried ontológie. To bol prípad prvých dvoch spomenutých ontológií, pretože tieto mali medzi verziami len malé rozdiely, takže boli ľahko identifikovateľné. V prípadoch, kedy bola systému poskytnutá verzia s množstvom koncepčných zmien (prípad spomínanej ontológie časopisu), úspešnosť identifikácie mapovaní medzi triedami sa znížila a systém bol prakticky schopný identifikovať len triedy a sloty s rovnakými názvami. V takýchto prípadoch však identifikácia zmien môže napomôcť vďaka možnosti ručného mapovania. Systému totižto stačí, aby našiel „záchytné body“, t.j. aby boli splnené podmienky pre tú ktorú heuristiku. Po splnení takýchto podmienok bol systém opäť schopný identifikovať ďalšie zmeny. Ukážka obrazovky z tohto porovnania je na obr. 30



obr. 30 Porovnanie verzií dvoch rôznych ontológií

Relatívne porovnávanie bolo testované na troch testovacích textoch. Prvou testovacou vzorkou bol text e-mailu bez diakritiky, ktorý bol následne o diakritiku doplnený a tieto dve verzie porovnávané. Systém spoľahlivo identifikoval a vyznačil príslušné zmeny v texte a prehlásil verzie o relatívne zhodné. Druhou vzorkou bol dokument, ktorý bol výstupom z Diplomového projektu a jeho upravená verzia s opravenými chybami a zapracovanými pripomienkami. Tieto dva dokumenty boli prehlásené za relatívne zhodné. Treťou vzorkou bol dokument Diplomového projektu v porovnaní s takmer konečnou verziou tejto práce a tieto dva dokumenty boli prehlásené za relatívne nezhodné, čo je na značný nárast textu a aj nových informácií pochopiteľné. Pri porovnaní boli použité východzie definície pásiem, takže by bolo vhodné vykonať viacej testov a experimentálne určiť vhodné definície pásiem a prahových hodnôt pre jednotlivé typy textov.

6 Zhodnotenie

Predložená práca sa zaoberá riešením problémov identifikácie a manažmentu zmien ontológií v kontexte technológií webu so sémantikou. Poskytuje nielen prehľad problematiky webu so sémantikou, ontológií, ktoré sú jedným z jeho základných kameňov, jazykov na zápis ontológií, ale najmä prehľad v oblasti problematiky identifikácie zmien medzi verziami ontológií, čo je základným predpokladom pre správnu kontrolu aktuálnosti informácií. Jedným z výstupov práce je návrh prostriedku pre automatizovanú kontrolu zmien verzií ontológií a ich manažment, pričom podstatné časti tohto návrhu boli overené aj funkčným prototypom.

Základ riešenia spočíva vo využívaní heuristických metód na vyhľadávanie ekvivalentných elementov verzií ontológií, čo znamená, že sa počíta s istou mierou neistoty pri identifikácií zmien. Niektoré z metód, ktoré sa používajú na sémantickej úrovni boli prevzaté, čo bolo predpokladom pre to, aby sme mohli navrhnúť prostriedky na identifikáciu zmien štrukturálnej úrovne. Práca teda prináša návrh metód identifikácie rozdielov na spomenutej štrukturálnej úrovni spolu s identifikáciou ekvivalencie anonymných indivíduí. Všetky takto identifikované zmeny je v konečnom dôsledku možné použiť na kontrolu platnosti samotných hodnôt slotov indivíduí, t.j. tých informácií, ktoré sa v konečnom dôsledku dostanú až k používateľovi. V tomto kontexte je prínosom práce metóda relatívneho porovnávania textov spolu so spôsobom reprezentácie identifikovaných zmien a návrhu ich prezentovania používateľovi, ktorý umožňuje jednoduchým spôsobom vyznačiť používateľovi sémanticky významné zmeny. Celkovo možno prehlásiť, že pri poznaní mapovania a rozdielov medzi verziami ontológie môžu napr. agenty, ktoré sú navrhnuté pre jednu verziu ontológie, pracovať či spolupracovať s inou verziou ontológie, pre ktorú pôvodne navrhnutí neboli. V podstate sa tým dosiahne cieľ, že existujúci systém s nejakým modelom údajov by bol schopný pracovať aj s iným modelom údajov.

Prototyp, ktorý bol v projekte vytvorený, poskytuje dobrý základ pre ďalšie rozšírenie ako na sémantickej, tak aj na štrukturálnej úrovni. Základným cieľom riešenia bolo vytvoriť akýsi všeobecný rámec pre identifikáciu zmien s tým, že sme zanedbali niektoré prípady, ako je rozdeľovanie a delenie elementov ontológií. Ďalej treba upozorniť aj na niektoré z nevýhod či obmedzení navrhnutého riešenia, ako je zanedbanie identifikácie vlastností elementov, ako sú napr. údajové typy, obory hodnôt, ich kardinalita či obmedzenia hodnôt. Tieto vlastnosti elementov sú silne závislé od konkrétneho použitého jazyka na zápis ontológie, čo však dáva priestor pre ďalšiu prácu

v zdokonaľovaní navrhnutých metód. Zanedbaním identifikácie všetkých detailov ontológie má dosah na reprezentáciu zmien a má za následok napr. zväčšenie súboru s exportovanými údajmi o zmenách.

Uvedené metódy a riešenia je možné využiť pri vybudovaní systému na distribúciu rozdielov medzi dvoma ontológiami alebo na online identifikáciu, ktorý môže byť jednoducho a automatizovane použitý inými systémami. To všetko na báze napr. dnes veľmi populárnej technológie webových služieb alebo Enterprise JavaBeans. Prototyp síce nebol vytvorený pre čo najefektívnejšie spracovanie, takže využíva uchovávanie ontológií v súboroch, čo so sebou pri neustálom parsovaní prináša zníženie výkonu. Tento problém je však možné vyriešiť použitím rámca (napr. existujúci rámec Sesame), ktorý vie ukladať celé ontológie do databázy, prípadne navrhnúť vlastné riešenie.

Azda najzaujímavejším rozšírením systému by mohlo byť dotvorenie systému, ktorý by bol viacej orientovaný na konečného konzumenta informácií. Takýto systém by si mohol uchovávať v definovaných časoch verzie ontológie, či už ručným zadaním súboru alebo automatickým získaním z internetu. Následne môže analyzovať sémantické a štrukturálne zmeny. Na základe týchto zmien by vedel efektívnym spôsobom podať používateľovi zmeny v sledovaných informáciách z istej domény a tieto zmeny aj efektne vyznačiť. V spojení s adaptívnymi hypermediálnymi systémami môžu byť takéto informácie „servírované“ používateľovi presne podľa jeho záujmu a potrieb, čo môže byť mimoriadne zaujímavé pre komerčné využitie

Použitá literatúra

BERNERS-LEE, Tim: *Semantic Web – XML2000*. 2000. http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide1-0.html (08.04.2005)

BERNERS-LEE, Tim – HENDLER, James – LASSILA, Ora: *The Semantic Web*. Scientific American. 2001, vol. 284, no. May. s. 35–43.

BROEKSTRA, JEEN – KAMPMAN, ARJOHN – VAN HARMELEN, FRANK: *The Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema*. International Semantic Web Conference 2002, Sardinia, Italy. http://www.openrdf.org/doc/papers/Sesame-ISWC2002.pdf (08.04.2005).

EHRIG, M. – DE BRUIJN, J. – MANOV, D. – MARTÍN-RECUERDA, F.: *State-of-the-art survey on Ontology Merging and Aligning*. TechReport. 2004. http://www.aifb.uni-karlsruhe.de/WBS/meh/publications/debruijn04state.pdf (30.11.2005)

FETTERLY, Dennis – MANASSE, Mark – NAJORK, Marc – WIENER, Janet: *A Large-Scale Study of the Evolution of Web Pages*. In Software: Practice and Experience, 34(2):213-237, February 2004.

GRUBER, T.R.: A *Translation Approach to Portable Ontology Specifications*. Knowledge Acquisition. 1993.

HRADSKÝ, Jaroslav: *Jazyk OWL a sémantický web*. Brno: Fakulta informatiky MU, 2003. 30 s. Bakalárska práca.

HUNT, J. W. – MCILROY, M. D.: *An algorithm for differential file comparison*. Bell Telephone Laboratories CSTR #41. 1976. http://www.cs.dartmouth.edu/~doug/diff.ps (19.11.2005)

KALYANPUR, Aditya – PASTOR, Daniel Jiménez – BATTLE, Steve – PADGET, Julian A.: *Automatic Mapping of OWL Ontologies into Java*. SEKE 2004. s. 98-103.

KLEIN, Michael – FENSEL, Dieter – KIRYAKOV, Atanas – OGNYANOV, Damyan: *Ontology versioning and change detection on the Web*. EKAW02, LNCS 2473, Sigüenza, Spain. 2002a.

KLEIN, Michael – FENSEL, Dieter: *Ontology versioning on the Semantic Web*. Semantic Web Working Symposium, Stanford. 2001.
http://www.semanticweb.org/SWWS/program/full/paper56.pdf (07.05.2005)

KLEIN, Michael – FENSEL, Dieter: *Ontoview: Web-based Ontology Versioning*. 2002b.
http://www.cs.vu.nl/~mcaklein/papers/ontoview.pdf (19.11.2004)

KOSEK, Jiří: *Sémantický web*. 2003.
http://badame.vse.cz/izi228/prednasky/xml/foil18.html (11.12.2004)

NOY, Natalya F. – MUSEN, Mark A.: *Ontology Versioning as an Element of an Ontology-Management Framework*. Stanford Medical Informatics, March 2003.

NTOULAS, Alexandros – CHO, Junghoo – OLSTON, Christopher: *What's New on the Web? The Evolution of the Web from a Search Engine Perspective*. In Proc. of the Thirteenth Int. World Wide Web Conf., May 2004, New York.

OCLC – Online Computer Library Center. 2002.
http://wcp.oclc.org (12.12.2004)

SVÁTEK, Vojtěch: *Ontologie a WWW*. 2002.
http://nb.vse.cz/~svatek/onto-www.pdf (11.12.2004)
(skrátená verzia publikovaná v zborníku konferencie DATAKON 2002)

TURY, Michal – BIELIKOVÁ, Mária: *Identifikácia zmien v ontológiách*. In Proc. of ITAT 2005 – Workshop on Theory and Practice of Information Technologies, P. Vojtáš (Ed.), september 2005, Račkova dolina, pp.381-390.

W3C: *RDF Vocabulary Description Language 1.0: RDF Schema.* W3C Recommendation 10 February 2004a.
http://www.w3.org/TR/rdf-schema/ (15.05.2005)

W3C: *RDF/XML Syntax Specification*. W3C Recommendation 10 February 2004b.
http://www.w3.org/TR/rdf-syntax-grammar/ (15.05.2005)

W3C: *OWL Web Ontology Language Reference*. W3C Recommendation 10 February 2004c.
http://www.w3.org/TR/owl-ref/ (15.05.2005)

Použitý softvér

DIFF: *GNU Diff for Java.*
http://www.bmsi.com/java/#diff (27.11.2005)

IBATIS: *Apache iBatis – Data Mapper framework.*
http://ibatis.apache.org/ (13.11.2005)

IDEA: *IntelliJ IDEA – intelligent Java IDE.*
http://www.jetbrains.com/idea/ (13.11.2005)

J2EE: *Java 2 Platform, Enterprise Edition.*
http://java.sun.com/j2ee/ (13.11.2005)

JENA: *Jena – A Semantic Web Framework for Java.*
http://jena.sourceforge.net/ (13.11.2005)

MYSQL: *MySQL – database server.*
http://www.mysql.com/products/database/ (13.11.2005)

ONTOVIEW: *OntoView.*
http://ontoview.org/ (13.11.2005)

PROMPT: *Prompt Plugin.*
http://protege.stanford.edu/plugins/prompt/prompt.html (20.11.2005)

PROTÉGÉ: *Protégé – ontology editor.*
http://protege.stanford.edu/ (13.11.2005)

SESAME: *Sesame – RDF database with support for RDF Schema inferencing and querying.*
http://www.openrdf.org/ (13.11.2005)

STRUTS: *Apache Struts – MVC framework.*
http://struts.apache.org/ (13.11.2005)

TOMCAT: *Apache Tomcat – servlet container.*
http://tomcat.apache.org/ (13.11.2005)

XALAN: *Apache Xalan-Java – XSLT processor for transforming XML documents* http://xml.apache.org/xalan-j/ (21.11.2005)

Príloha A Používateľská príručka

Systém OntoView je webová aplikácia, ktorá umožňuje identifikovať a manažovať zmeny vo verziách ontológií či už automatizovane, alebo ručne a taktiež umožňuje ukladať verzie ontológií v úložisku. Je napísaná v jazyku Java, ktorá vyžaduje databázový server MySQL a bola testovaná so servlet kontajnerom Apache Tomcat.

A.1 Požiadavky na systém

Na prevádzku systému je potrebné mať nainštalovaný JDK 1.5 alebo vyšší, servlet kontajner (napr. už spomenutý Apache Tomcat) s prístupom na databázový server MySQL 4.0. Systém OntoDiff nemá žiadne ďalšie požiadavky na svoju prevádzku. Všetky uvedené serverové aplikácie spolu s ich dokumentáciou nájdete na elektronickom médiu v adresári *Install*. Na strane klienta sa vyžaduje internetový prehliadač ako je napr. Mozilla, Opera či Internet Explorer.

A.2 Inštalácia systému OntoDiff

Predpokladom pre úspešnú inštaláciu je správne konfigurovaný databázový server MySQL, servlet kontajner a JDK s možnosťou prístupu na databázový server MySQL.

Samotná inštalácia zahŕňa tieto kroky:

1. Do koreňového adresára servlet kontajneru vytvorte nový adresár napr. s názvom OntoDiff. Tento adresár predstavuje koreňový adresár systému.

2. Do adresára vytvoreného v bode 1 skopírujte celý obsah adresára */OntoDiff/bin* z inštalačného média.

3. Vytvorte novú databázu, napr. s menom *ontodiff* v databázovom systéme MySQL.

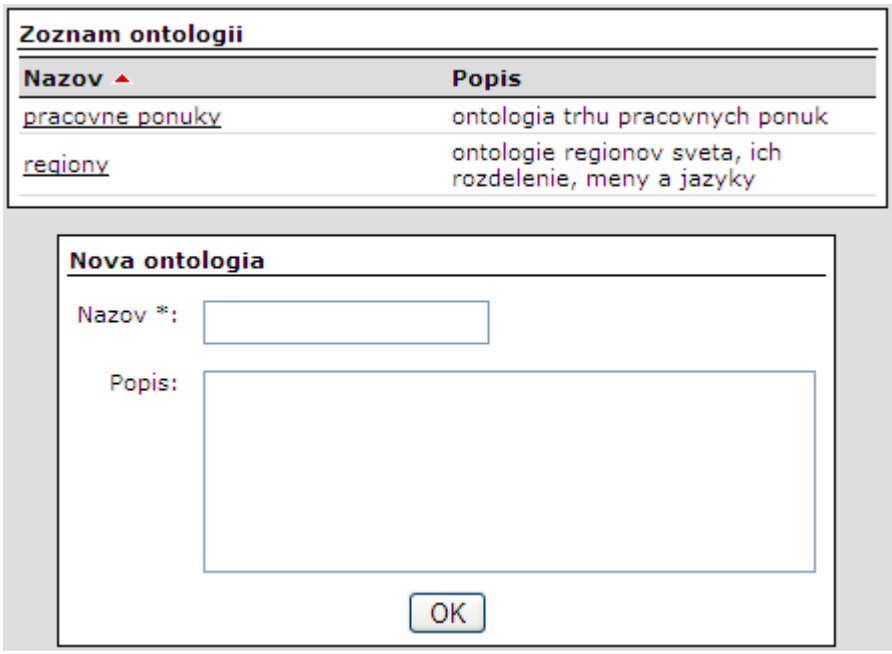
4. Nad novo vytvorenou databázou *ontodiff* spustite skript *db_dump.sql*, ktorý sa nachádza v adresári */OntoDiff/SQL* inštalačného média. Týmto krokom vytvoríte štruktúru databázových tabuliek, potrebných na prevádzku systému.

5. Vytvorte nového používateľa v databázovom systéme, ktorý bude mať plné práva na databázu *ontodiff*. Tento používateľský účet bude využívať systém OntoDiff na prístup do databázy.

6. Otvorte v textovom editore súbor *WEB-INF/sqlmap/SqlMapConfig.properties*, nachádzajúci sa v koreňovom adresári systému. Parametru *url* priraďte adresu počítača, na ktorom je nainštalovaný databázový server MySQL, názov databázy a prípadne aj port, na ktorom MySQL načúva. Parametru *username* priraďte meno používateľa, ktorého ste vytvorili v kroku 4 a parametru *password* jeho používateľské heslo. Takto nainštalovaný systém je možné overiť zadaním adresy do prehliadača (napr.: http://localhost:8080/OntoDiff).

A.3 Správca ontológií

Správca ontológií slúži na katalogizovanie ontológií, ku ktorým sa následne môžu priraďovať ich verzie. Na obr. 31 je znázornený správca ontológií. V hornej časti sa nachádza zoznam ontológií a v spodnej časti formulár na pridanie novej ontológie. Zoznam ontológií je možné zoradiť vzostupne alebo zostupne kliknutím na názov stĺpca v titulku tabuľky. Kliknutím na názov konkrétnej ontológie sa zobrazia verzie danej ontológie v správcovi verzií.



obr. 31 Správca ontológií

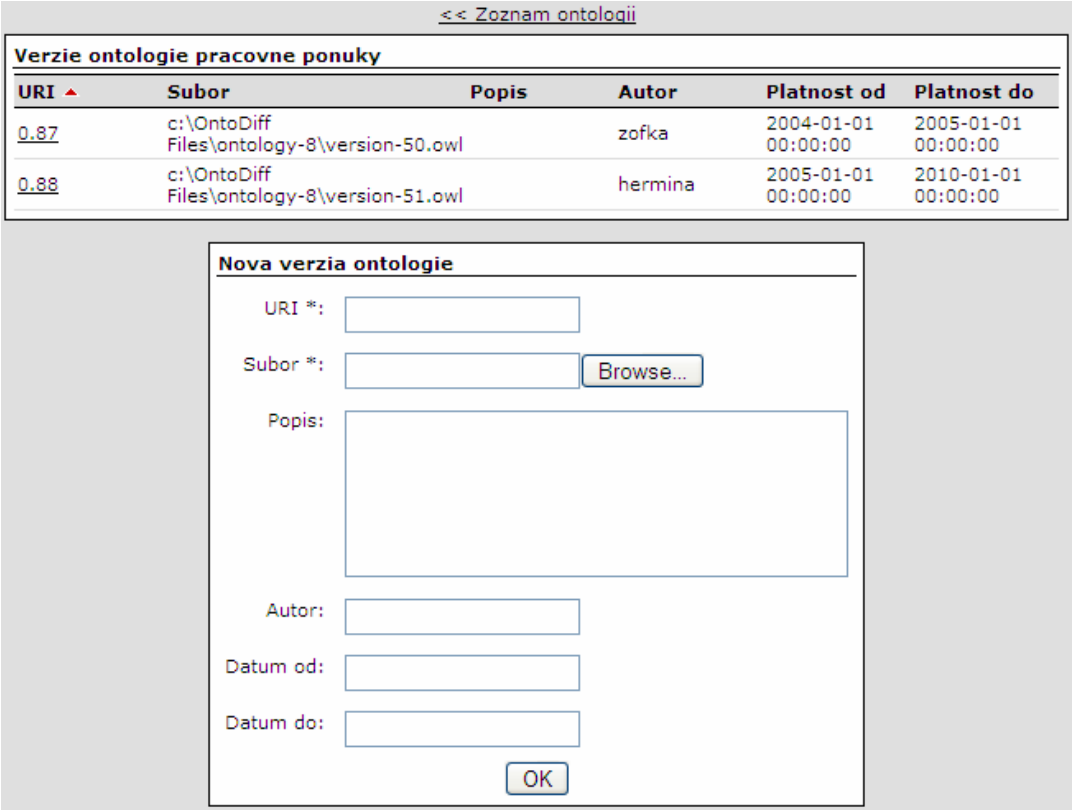
Na pridanie novej ontológie vyplňte formulár v spodnej časti. Do položky *Názov*, ktorý je zároveň aj povinnou položkou, vyplňte názov vytváranej ontológie. Do nepovinnej položky *Popis* môžete uviesť stručnú charakteristiku práve vytváranej ontológie. Po kliknutí na tlačidlo *OK* sa vytvorí nová ontológia.

A.4 Správca verzií

Správca verzií slúži na katalogizovanie verzií ontológie. Na obr. 32 je znázornený správca verzií. V hornej časti obrazovky sa nachádza zoznam verzií ontológie a v spodnej časti formulár na pridanie novej verzie ontológie. Zoznam verzií je možné

zoradiť vzostupne alebo zostupne kliknutím na názov stĺpca v titulku tabuľky. Po kliknutí na URI verzie, je možné túto verziu začať porovnávať s verziou inou.

Na pridanie novej verzie ontológie vyplňte formulár v spodnej časti. Do položky *URI*, vyplňte URI novej verzie ontológie. Do položky *Súbor* zadajte názov súboru vo formáte OWL, reprezentujúci danú verziu ontológie. Zadaný súbor systém uloží do svojho úložiska. Do nepovinnej položky *Popis* môžete uviesť stručnú charakteristiku práve pridávanej verzie ontológie. V položke *Autor* je možné uviesť autora verzie ontológie a hodnotami položiek *Dátum od* a *Dátum do* sa dá určiť časová platnosť verzie. Po kliknutí na tlačidlo *OK* sa pridá nová verzia ontológie.



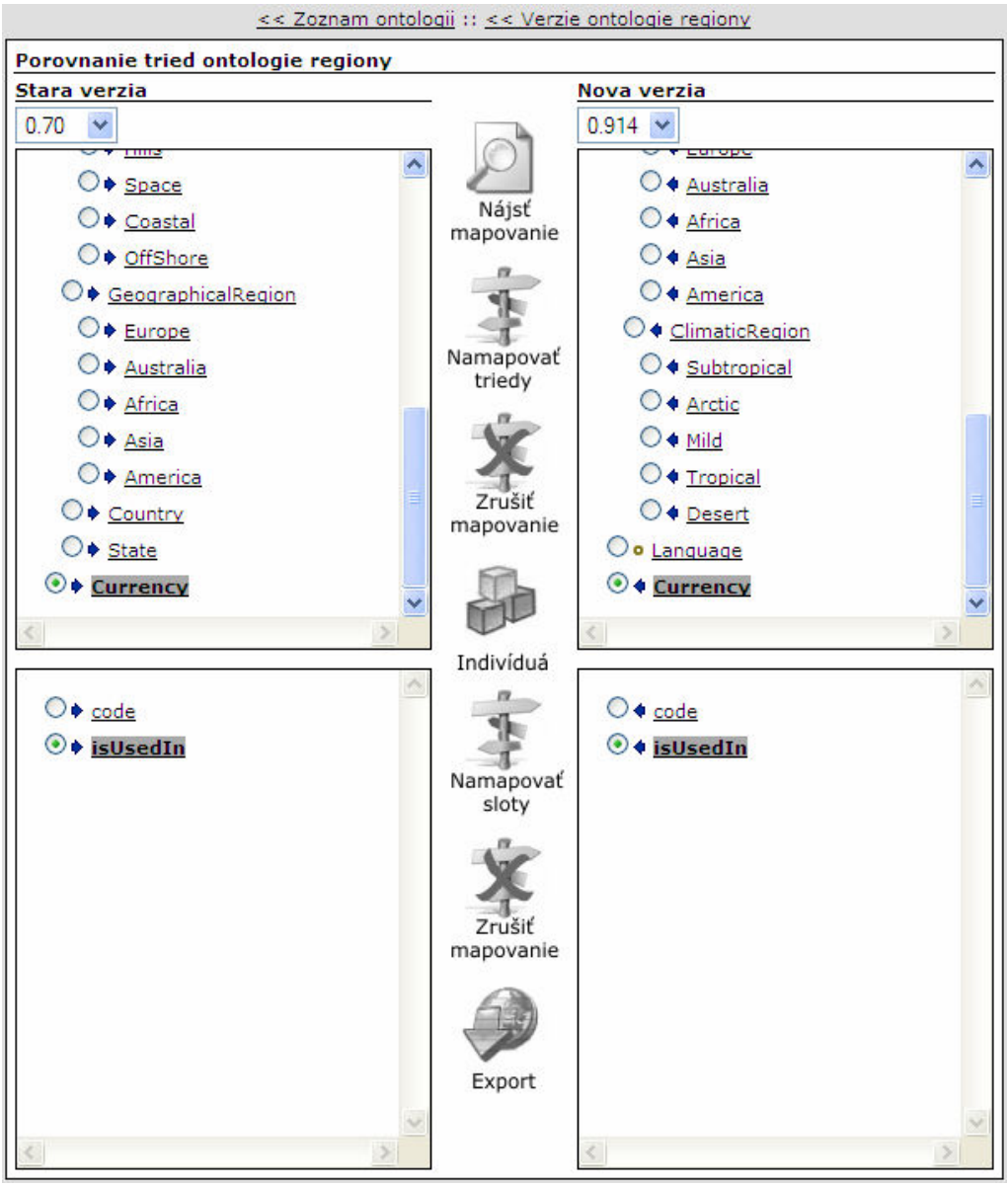
obr. 32 Správca verzií

A.5 Porovnávanie verzií

Na obr. 33 je znázornené porovnanie dvoch verzií ontológie. Na ľavej strane sa nachádza staršia verzia a na pravej strane novšia verzia. V hornej časti sa pritom nachádza hierarchia tried a v spodnej časti sloty, práve označenej triedy. Výber porovnávaných verzií sa realizuje prostredníctvom vysúvacích zoznamov v hornej časti obrazovky.

Pri každej triede či slote sa nachádza ikona znázorňujúca, či element je mapovaný (modrá šípka), pridaný (žltý krúžok) alebo odobratý (červený krížik). V prípade, že element je mapovaný, tak kliknutím na jeho názov, sa tento označí a označí sa aj element, na ktorý je tento mapovaný.

Kliknutím na ikonu *Nájsť mapovanie* sa systém pokúsi nájsť mapovanie elementov medzi práve zvolenými verziami ontológie. Vyznačením dvoch tried prostredníctvom tlačidiel výberu, ktoré sa nachádzajú pri každej z nich a kliknutím na ikonu *Namapovať triedu* sa tieto dve triedy namapujú. Podobne, vybratím dvoch už namapovaných tried prostredníctvom tlačidiel výberu a kliknutím na ikonu *Zrušiť mapovanie*, sa existujúce mapovanie medzi triedami zruší.



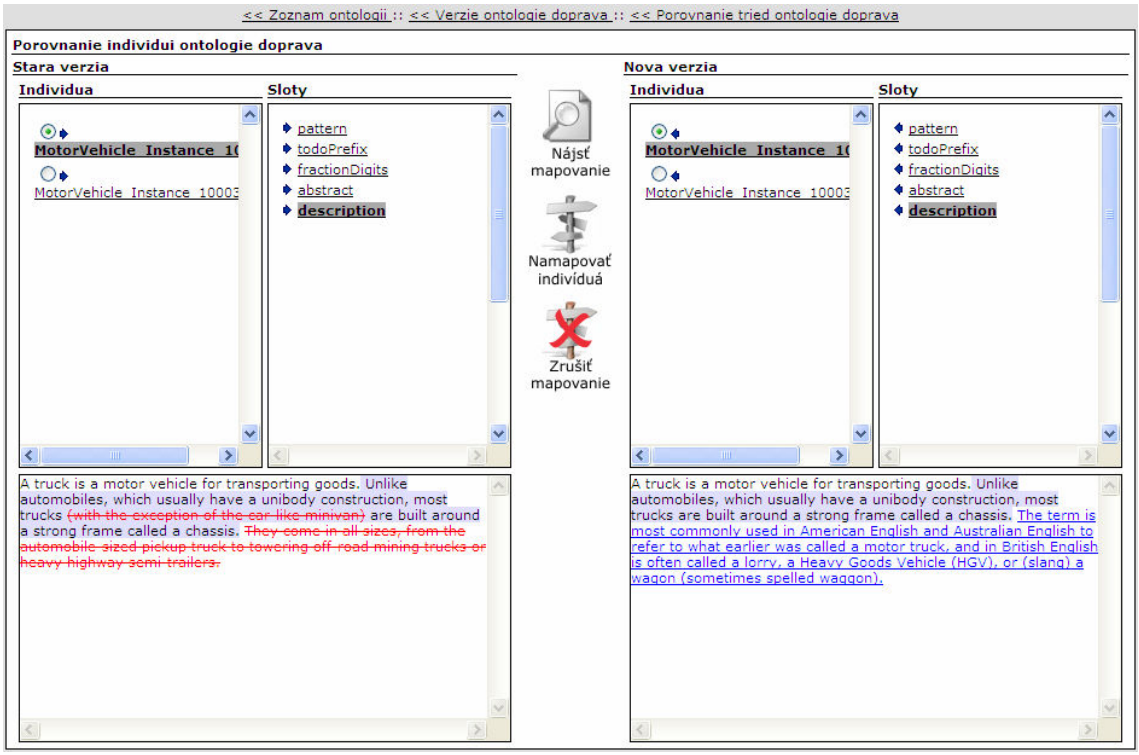
obr. 33 Porovnávanie verzií

Po kliknutí na niektorú z tried sa v spodnej časti zobrazia jej sloty. V prípade, že pre triedu existuje mapovanie, zobrazia sa aj mapovania medzi slotmi týchto mapovaných tried. Kliknutím na ikonu *Indivíduá* sa zobrazia indivíduá mapovaných tried (pozri A.6). Vyznačením dvoch slotov prostredníctvom tlačidiel výberu, ktoré sa nachádzajú pri každom z nich a kliknutím na ikonu *Namapovať sloty* sa tieto dva sloty namapujú. Podobne, vybratím dvoch už namapovaných slotov prostredníctvom tlačidiel výberu

a kliknutím na ikonu *Zrušiť mapovanie*, sa existujúce mapovanie medzi slotmi zruší. Kliknutím na ikonu *Export* sa vytvorí XML súbor s identifikovanými zmenami v obojsmernej reprezentácii. Tento súbor sa otvorí v novom okne, takže si ho používateľ môže následne uložiť.

A.6 Porovnávanie indivíduí

Na obr. 34 je znázornené porovnanie indivíduí medzi mapovanými triedami dvoch verzií ontológie. Na ľavej strane sa nachádza staršia verzia a na pravej strane novšia verzia. V hornej časti sa nachádza zoznam indivíduí tried spolu so zoznamom slotov definovaných v triede, do ktorej dané indivíduá patria. Podobne ako pri porovnávaní tried, pri každom elemente sa nachádza ikona znázorňujúca, či je mapovaný (modrá šípka), pridaný (žltý krúžok) alebo odobratý (červený krížik). Kliknutím na názov elementu sa tieto označia. Po označení indivídua a slotu sa v spodnej časti zobrazia ich hodnoty.



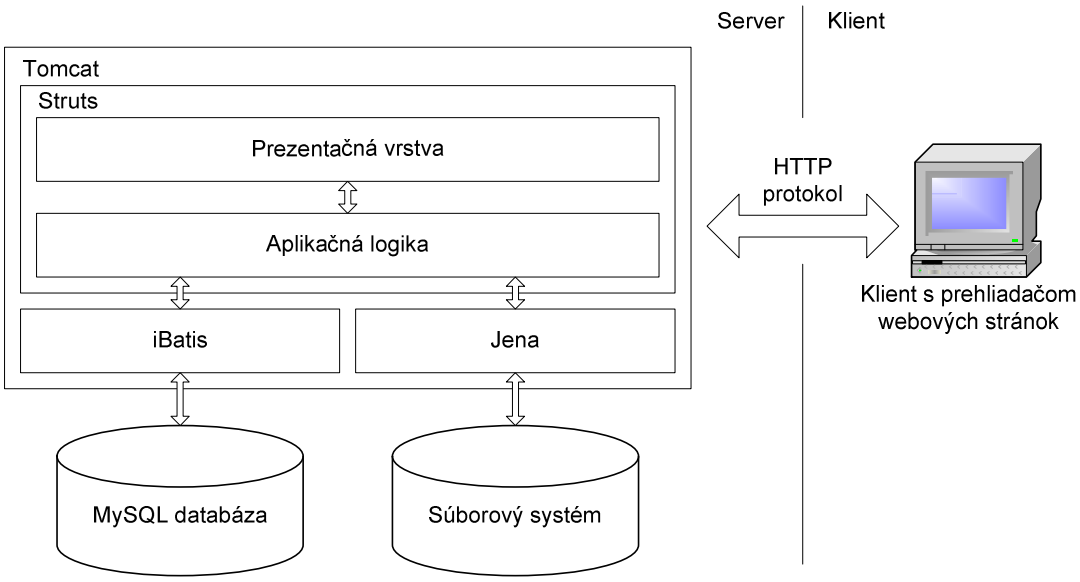
obr. 34 Porovnávanie indivíduí

V prípade, že sú označené mapované sloty v mapovaných indivíduách, majú vyznačené zmeny medzi sebou tak, že rovnaké časti sú na bielom pozadí, mapované vety na bledomodrom pozadí, mapované slová v rámci nich na tmavomodrom pozadí a zmenené znaky sú vyznačené na červenom pozadí. Časti, ktoré boli odobraté sú vypísané červenou farbou a prečiarknuté a časti pridané sú podčiarknuté a znázornené modrou farbou.

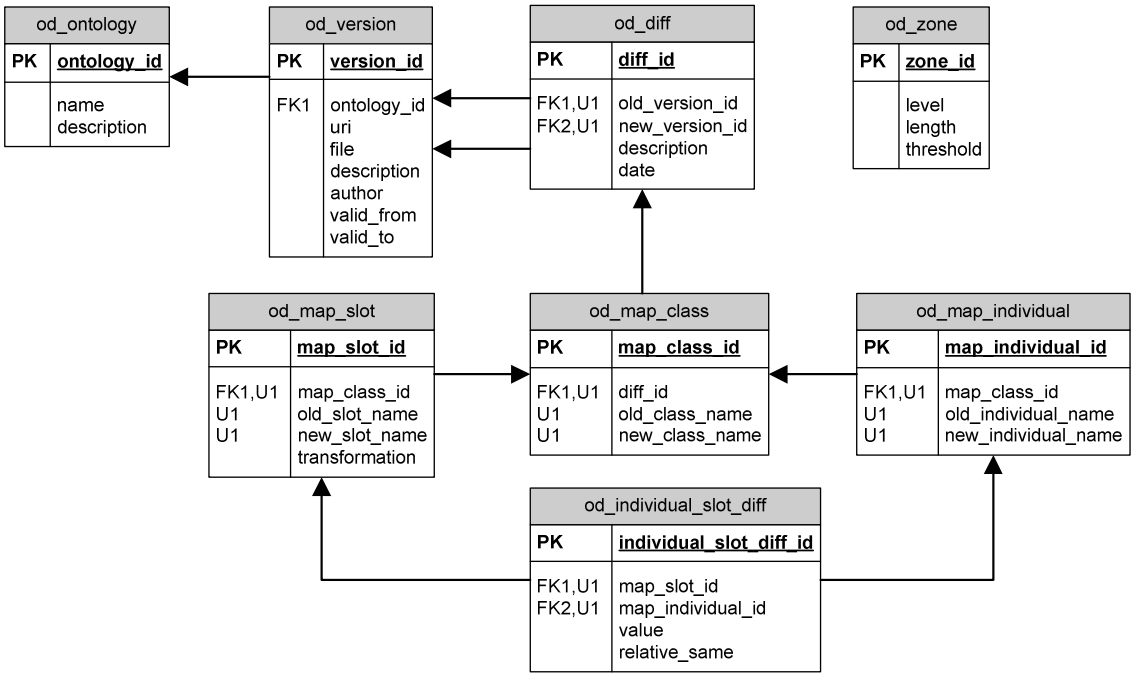
Kliknutím na ikonu *Nájsť mapovanie* sa systém pokúsi nájsť mapovanie indivíduí medzi starou a novou triedou. Vyznačením dvoch indivíduí prostredníctvom tlačidiel výberu, ktoré sa nachádzajú pri každom z nich a kliknutím na ikonu *Namapovať indivíduá* sa tieto dve indivíduá namapujú. Podobne, vybratím dvoch už namapovaných indivíduí prostredníctvom tlačidiel výberu a kliknutím na ikonu *Zrušiť mapovanie*, sa existujúce mapovanie medzi indivíduami zruší.

Príloha B Technická dokumentácia

Systém OntoDiff je implementovaný ako webová aplikácia na platforme J2EE (J2EE, 2005), ktorý vychádza z architektúry klient/server znázornenej na obr. 35. Ako aplikačný server využíva Apache Tomcat 5.0.28 (Tomcat, 2005), ale vďaka použitým technológiám je nasaditeľný aj na inom aplikačnom serveri. Bol použitý rámec Apache Struts (Struts, 2005), čo je open source rámec pre tvorbu webových aplikácií na platforme Java, pričom využíva architektonický vzor Model-View-Controller. Vo vnútri rámca je ako model vytvorená samotná aplikačná a aj prezentačná vrstva, ktorá zabezpečuje používateľské prostredie, t.j. vstup údajov od používateľa, ako aj výstup údajov. Ako úložisko je využitý súborový systém operačného systému na ukladanie súborov s verziami ontológií. Informácie o ontológiách a ich zmenách sa ukladajú do databázy, pričom ako databázový server sa využíva MySQL 4.0 (MySQL, 2005). Fyzický model údajov je znázornený na obr. 36. Pre jednoduchšiu prácu s databázou je využitý rámec iBatis (iBatis, 2005), ktorý predstavuje nástroj na mapovanie medzi záznamami v tabuľkách databázy a JavaBean objektmi. Na prácu s ontológiami sa využíva rámec Jena a pri relatívnom porovnávaní textov bola použitá GNU implemetácia algoritmu diff pre Javu (Diff, 2005). Na XML transformácie bol použitý XSLT procesor Apache Xalan (Xalan, 2005). Podrobná dokumentácia vrátane opisu všetkých metód sa nachádza na priloženom elektronickom nosiči v adresári */OntoDiff/javadoc*.



obr. 35 Architektúra systému OntoDiff



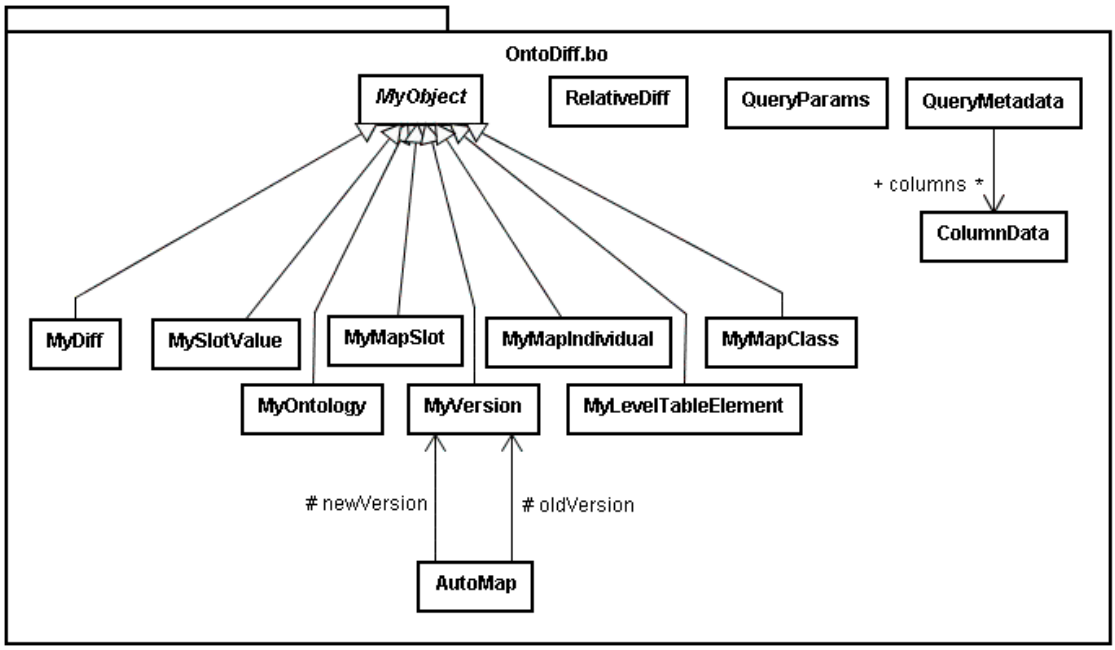
obr. 36 Fyzický model údajov systému

B.1 Aplikačná logika

Aplikačná logika a objekty použité pre mapovanie s databázovými tabuľkami sa nachádzajú v balíku *OntoDiff.bo* ako to znázorňuje obr. 37. Tento balík obsahuje nasledujúce triedy:

- *MyObject* – abstraktná trieda, od ktorej sú odvodené všetky ostatné JavaBean triedy určené pre mapovanie s databázovými tabuľkami,
- *MyOntology* – trieda reprezentujúca ontológiu, ktorá zároveň reprezentuje jeden riadok v databázovej tabuľke *od_ontology* (pozri fyzický model údajov na obr. 36),
- *MyVersion* – trieda reprezentujúca verziu ontológie, ktorá zároveň reprezentuje jeden riadok v databázovej tabuľke *od_version*,
- *MyDiff* – trieda reprezentujúca rozdiely medzi verziami ontológie, ktorá zároveň reprezentuje jeden riadok v databázovej tabuľke *od_diff*,
- *MyMapClass* – trieda reprezentujúca mapovanie medzi triedami, ktorá zároveň reprezentuje jeden riadok v databázovej tabuľke *od_map_class*,
- *MyMapSlot* – trieda reprezentujúca mapovanie medzi slotmi tried, ktorá zároveň reprezentuje jeden riadok v databázovej tabuľke *od_map_slot*,
- *MyMapIndividual* – trieda reprezentujúca mapovanie medzi indivíduami, ktorá zároveň reprezentuje jeden riadok v databázovej tabuľke *od_map_individual*,

- *MySlotValue* – trieda reprezentujúca hodnotu porovnávaných slotov, ktorá zároveň reprezentuje jeden riadok v databázovej tabuľke *od_individual_slot_diff*,
- *MyLevelTableElement* – trieda reprezentujúca pásmo s prahovou hodnotou v definícii pásiem pri relatívnom porovnávaní textov, ktorá zároveň reprezentuje jeden riadok v databázovej tabuľke *od_zone*,
- *AutoMap* – trieda implementujúca automatickú identifikáciu zmien medzi verziami ontológie,
- *RelativeDiff* – trieda implementujúca relatívne porovnávanie textov,
- *QueryMetadata* – trieda implementujúca kontajner prezentovaných údajov,
- *ColumnData* – údaje o stĺpci tabuľky využívajúca kontajnerom prezentovaných údajov QueryMetadata,
- *QueryParams* – pomocná trieda na predávanie parametrov do rámca iBatis.



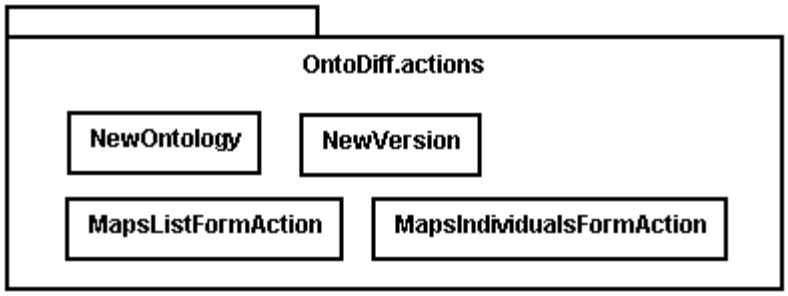
obr. 37 Diagram tried aplikačnej logiky

B.2 Akcie

Akcie pre rámec Struts sa nachádzajú v balíku *OntoDiff.actions* ako je to znázornené na obr. 38. Tento balík obsahuje nasledujúce triedy:

- *NewOntology* – trieda implementujúca akciu na vytvorenie novej ontológie v úložisku,
- *NewVersion* – trieda implementujúca akciu na vytvorenie novej verzie ontológie v úložisku,

- *MapsListFormAction* – trieda implementujúca akciu na vytvorenie, rušenie a prezeranie mapovania na sémantickej úrovni,
- *MapsIndividualsFormAction* – trieda implementujúca akciu na vytvorenie, rušenie a prezeranie mapovania na štrukturálnej úrovni.

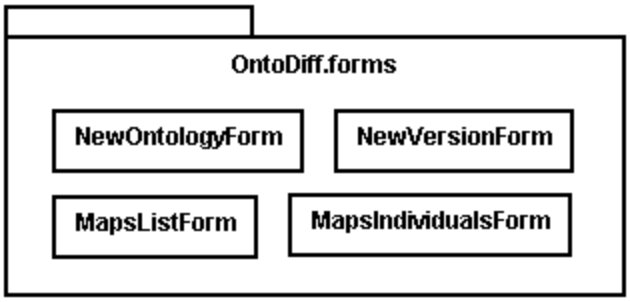


obr. 38 Diagram tried akcií pre rámec Struts

B.3 Formuláre

Akcie pre rámec Struts sa nachádzajú v balíku *OntoDiff.actions* ako je to znázornené na obr. 39. Tento balík obsahuje nasledujúce triedy:

- *NewOntologyForm* – trieda implementujúca formulár využívaný pri vytváraní novej ontológie v úložisku,
- *NewVersionForm* – trieda implementujúca formulár využívaný pri vytváraní novej verzie ontológie v úložisku,
- *MapsListFormAction* – trieda implementujúca formulár využívaný pri vytváraní, rušení a prezeraní mapovania na sémantickej úrovni,
- *MapsIndividualsFormAction* – trieda implementujúca formulár využívaný pri vytváraní, rušení a prezeraní mapovania na štrukturálnej úrovni.

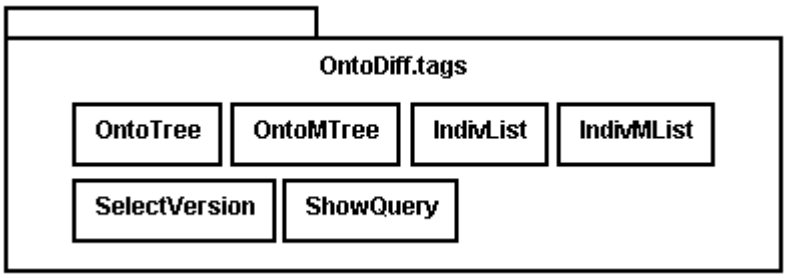


obr. 39 Diagram tried formulárov pre rámec Struts

B.4 Značky

Značky, ktoré sa využívajú v prezentačných JSP súboroch sa nachádzajú v balíku *OntoDiff.tags* ako to znázorňuje obr. 39. Tento balík obsahuje nasledujúce triedy:

- *OntoTree* – trieda implementujúca značku na zobrazenie stromu tried starej ontológie,
- *OntoMTree* – trieda implementujúca značku na zobrazenie stromu tried novej ontológie,
- *IndivList* – trieda implementujúca značku na zobrazenie zoznamu indivíduí starej ontológie,
- *IndivMList* – trieda implementujúca značku na zobrazenie zoznamu indivíduí novej ontológie,
- *SelectVersion* – trieda implementujúca značku na zobrazenie vysúvacieho zoznamu s verziami ontológie,
- *ShowQuery* – trieda implementujúca značku na zobrazenie tabuľky so záznamami uložených v kontajneri prezentovaných údajov QueryMetadata.

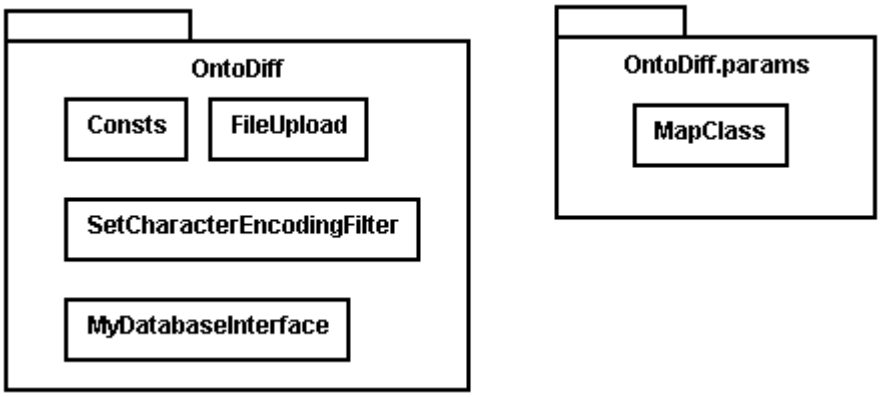


obr. 40 Diagram tried značiek

B.5 Pomocné triedy

Pomocné triedy, ktoré využíva systém OntoDiff sú znázornené na obr. 41. Zahŕňa nasledujúce triedy:

- *Consts* – trieda obsahujúca všetky potrebné konštanty používané v systéme,
- *FileUpload* – trieda zabezpečujúca „upload“ súborov do systému,
- *SetCharacterEncodingFilter* – filter, ktorý zabezpečuje konverziu znakových sád,
- *MyDatabaseInterface* – databázové rozhranie určené pre MySQL,
- *MapClass* – pomocná trieda na predávanie parametrov do rámca iBatis,



obr. 41 Diagram pomocných tried

B.6 Ukážka implementácie

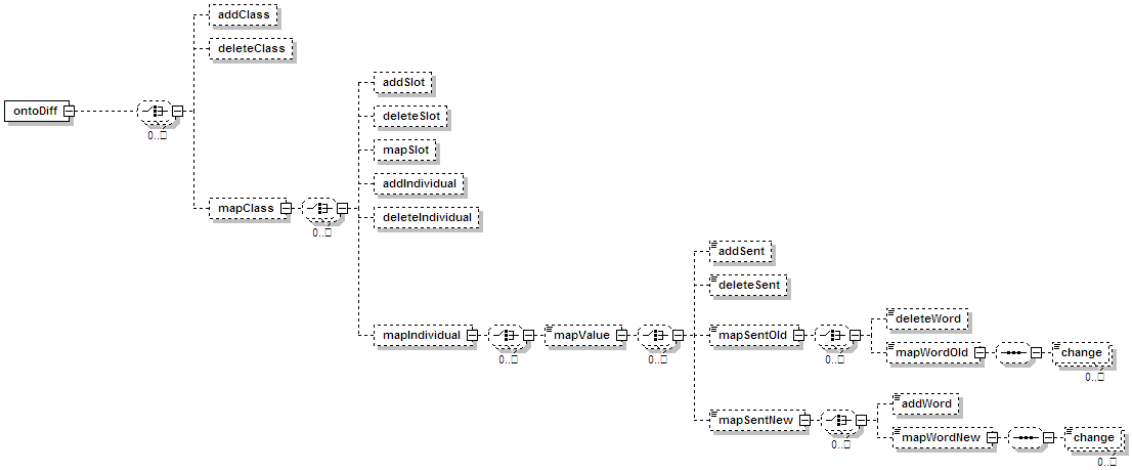
Na obr. 42 sa nachádza ukážka implementácie algoritmu *rozhodnutie*, ktorý bol navrhnutý v časti 4.5.3.

```
/**
 * urobi rozhodnutie, ci pre dane dlzky su elementy relativne
 * zhodne alebo nie
 * @param oldTextLength  dlzka stareho elementu
 * @param newTextLength  dlzka noveho elementu
 * @param changedElements  zmenenych elementov
 * @param levelTable  tabulka s definovanymi pasmami a prahovymi hodnotami
 * @param  defaultThreshold  vychodzia prahova hodnota
 * @return vracia true, ak su texty relativne zhodne, inak vracia false
 */
protected boolean makeResult(int oldTextLength, int newTextLength,
            double changedElements, ArrayList levelTable,
            double defaultThreshold) {

    int length = (oldTextLength > newTextLength
                        ?oldTextLength:newTextLength);
    double number = changedElements / (double)length;
    double threshold = defaultThreshold;
    MyLevelTableElement myLevelTableElement;

    if(levelTable != null) {
        for(int i = levelTable.size() - 1; i >= 0; i--) {
            myLevelTableElement = (MyLevelTableElement)levelTable.get(i);
            if(length > myLevelTableElement.getLegth()) {
                break;
            }
            threshold = myLevelTableElement.getThreshold();
        }
    }

    if(threshold < number) return false;
    return true;
}
```

obr. 42 Ukážka implementácie

Príloha C Špecifikácia exportného formátu

Exportný formát systému OntoDiff slúži na ukladanie a šírenie identifikovaných zmien medzi dvoma verziami ontológie a predstavuje štandardný XML dokument. Tento dokument obsahuje hlavičku, ktorá nesie informácie o tom, ktorých dvoch verzií sa zmeny týkajú a tela, kde sú popísané jednotlivé zmeny. V tejto prílohe sú opísane elementy tohto XML dokumentu. Na obr. 43 je znázornená XML Schema exportného formátu. Súbor s XML Schemou sa nachádza na priloženom elektronickom nosiči v adresári */OntoDiff/xsd*.



obr. 43 XML Schema exportného formátu

C.1 Koreňový uzol

Koreňový uzol *ontoDiff* je základným elementom dokumentu. Obsahuje dva atribúty *oldVersion* udávajúci URI starej verzie a *newVersion* udávajúci verziu novej verzie. Tento element môže obsahovať niekoľko poduzlov *addClass*, *deleteClass* a *mapClass*.

C.2 Triedy

Uzlom *addClass* je možné určiť pridanú triedu v novej verzii. Jeho jediným atribútom je *new*, ktorý špecifikuje meno triedy, ktorá bola v novej verzii pridaná. Uzol *deleteClass* určuje odobratú triedu v starej verzii. Jeho jediným atribútom je *old*, ktorý špecifikuje meno triedy, ktorá bola v starej verzii odobratá. Uzol *mapClass* definuje mapovanie medzi triedami v starej a novej verzii. Názvy tried, medzi ktorými toto mapovanie existuje je definované atribútmi tohto uzla. Atribút *old* určuje meno triedy v starej verzii

a atribút *new* určuje meno triedy v novej verzii. Uzol môže obsahovať niekoľko poduzlov *addSlot*, *deleteSlot*, *mapSlot*, *addIndividual*, *deleteIndividual* a *mapIndividual*.

C.3 Sloty

Uzlom *addSlot* je možné určiť pridaný slot v rámci triedy v novej verzii. Jeho atribútom je *new*, ktorý špecifikuje meno slotu, ktorý bol v novej verzii pridaný. Uzol *deleteSlot* určuje odobratý slot v rámci triedy v starej verzii. Jeho atribútom je *old*, ktorý špecifikuje meno slotu, ktorú bol v starej verzii odobratý. Uzol *mapSlot* definuje mapovanie medzi slotmi v rámci mapovaných tried. Názvy slotov, medzi ktorými toto mapovanie existuje je definované atribútmi tohto uzla. Atribút *old* určuje meno slotu v starej verzii a atribút *new* určuje meno slotu v novej verzii.

C.4 Indivíduá

Uzlom *addIndividual* je možné určiť pridané indivíduum triedy v novej verzii. Jeho atribútom *new* špecifikuje meno indivídua, ktoré bolo v novej verzii pridané. Uzol *deleteIndividual* určuje odobraté indivíduum v starej verzii. Obsahuje atribút *old*, ktorý špecifikuje meno indivídua, ktoré bolo v starej verzii odobraté. Uzol *mapIndividual* definuje mapovanie medzi indivíduami triedy. Názvy indivíduí, medzi ktorými toto mapovanie existuje je definované atribútmi tohto uzla. Atribút *old* určuje meno indivídua v starej verzii a atribút *new* určuje meno indivídua v novej verzii. Uzol môže obsahovať niekoľko poduzlov *mapValue*.

C.5 Hodnoty slotov

Uzlom *mapValue* je možné mapovať hodnoty slotov indivíduí. Názvy slotov, medzi ktorými je hodnota mapovaná sú definované atribútmi tohto uzla. Atribút *old* určuje meno slotu triedy v starej verzii a atribút *new* určuje meno slotu triedy v novej verzii. Atribút *equal* definuje, či sú dané hodnoty slotov relatívne zhodné alebo nie. Uzol mapValue obsahuje hodnoty slotov s vyznačenými zmenami, pričom môže obsahovať niekoľko poduzlov *addSent, deleteSent, mapSentOld, mapSentNew*.

Uzol *addSent* vyznačuje pridanú vetu v novej verzii v rámci mapovanej hodnoty slotu. Neobsahuje žiadne atribúty. Uzol *deleteSent* vyznačuje odobratú vetu zo starej verzie v rámci mapovanej hodnoty slotu. Neobsahuje žiadne atribúty. Uzol *mapSentOld* definuje mapovanú vetu v starej verzii v rámci mapovanej hodnoty slotu. Atribút *id* určuje identifikátor mapovania v rámci mapovania hodnoty slotu, t.j. musí existovať uzol mapSentNew s rovnakou hodnotou tohto identifikátora. Uzol mapSentOld môže obsahovať niekoľko poduzlov *deleteWord* a *mapWordOld*. Uzol *mapSentNew* definuje mapovanú vetu v novej verzii v rámci mapovanej hodnoty slotu. Atribút *id* určuje identifikátor mapovania v rámci mapovania hodnoty slotu t.j. musí existovať uzol mapSentOld s rovnakou hodnotou tohto identifikátora. Uzol mapSentNew môže obsahovať niekoľko poduzlov *addWord* a *mapWordNew*.

Uzol *deleteWord* vyznačuje odobraté slovo zo starej verzie v rámci mapovanej hodnoty slotu. Neobsahuje žiadne atribúty. Uzol *mapWordOld* definuje mapované slovo v starej verzii v rámci mapovanej hodnoty slotu. Atribút *id* určuje identifikátor mapovania v rámci mapovania hodnoty slotu t.j. musí existovať uzol mapWordNew s rovnakou hodnotou tohto identifikátora. Uzol mapWordOld môže obsahovať niekoľko poduzlov *change*. Uzol *addWord* vyznačuje pridané slovo do novej verzie v rámci mapovanej hodnoty slotu. Neobsahuje žiadne atribúty. Uzol *mapWordNew* definuje mapované slovo v novej verzii v rámci mapovanej hodnoty slotu. Atribút *id* určuje identifikátor mapovania v rámci mapovania hodnoty slotu t.j. musí existovať uzol mapWordOld s rovnakou hodnotou tohto identifikátora. Uzol mapWordNew môže obsahovať niekoľko poduzlov *change*. Uzol *change* vyznačuje zmenu znakov v rámci slova mapovaného slova. Atribút *id* určuje identifikátor mapovania v rámci mapovania hodnoty slotu.

Príloha D Súvis s diplomovým projektom

V diplomovom projekte sme sa zaoberali problematikou identifikácie zmien ontológií na sémantickej úrovni. Keďže šlo o problematiku široký a autorovi neznámu, projekt zahŕňal analýzu súčasného stavu ako technológií webu so sémantikou, tak aj problematiku identifikácie zmien v ontológiách. Súčasťou projektu bol návrh heuristík pre identifikáciu zmien na sémantickej a štrukturálnej úrovni, pričom sme predpokladali existenciu nástroja na porovnávanie textov, ktorý by zohľadňoval len sémanticky významné zmeny textu. Návrh bol overený prototypom, ktorý dokázal uchovávať verzie ontológií a identifikovať zmeny na sémantickej úrovni.

Diplomová práca nadväzuje na tento projekt tým, že sme sa v nej zaoberali hlbšie danou problematikou. Študovali sme ďalšie možnosti a problémy pri identifikácií zmien, ktoré boli navrhnuté v diplomovom projekte, pričom sme kládli dôraz na štrukturálnu úroveň. Bola navrhnutá metóda relatívneho porovnávania textov, určená na identifikáciu sémanticky významných zmien v texte. Pozornosť sme venovali aj spôsobom prezentácie identifikovaných zmien a aj ich reprezentácie za účelom uchovávania a distribúcie. Pôvodný prototyp z diplomového projektu bol za účelom overenia návrhu rozšírený o identifikáciu zmien na štrukturálnej úrovni, kde sa na porovnávanie hodnôt slotov využíva relatívne porovnávanie textov. Prototyp taktiež zahŕňa možnosť automatického či ručného mapovania medzi elementmi verzií ontológie či exportu identifikovaných zmien.

Príloha E Obsah elektronického nosiča

Ako prílohu tejto práce tvorí aj elektronický nosič, na ktorom sa nachádzajú všetky dokumenty, ktoré súvisia s touto prácou ako aj jej elektronickú formu. Nachádza sa tu aj implementácia funkčného prototypu nástroja OntoDiff.

Elektronický nosič pozostáva z troch adresárov:

```
/Diplomova praca
/Install
/OntoDiff
```

Adresár *Diplomova praca* obsahuje elektronickú formu dokumentu diplomovej práce spolu s kópiami zdrojov dostupných na internete, ktoré boli použité pri jej vypracovaní. V adresári *Install* sa nachádzajú softvérové prostriedky, ktoré boli použité pri vytváraní systému OntoDiff. Adresár *OntoDiff* obsahuje zdrojové a binárne súbory nástroja OntoDiff.

/Diplomova praca

```
/Diplomova praca
/Diplomova praca/Zdroje
```

V tomto adresári sa nachádza text dokument diplomovej práce v elektronickej forme. Súbor *Diplomova praca.doc* je určený pre textový procesor Microsoft Word XP a súbor *Diplomova praca.pdf* pre prehliadač Adobe Acrobat Reader. V podadresári *Zdroje* sú niektoré dostupné internetové zdroje, na ktoré sa práca odkazuje a boli pri jej písaní použité.

/Install

```
/Install/Java
/Install/Linux
/Install/Linux/IntelliJ IDEA
/Install/Linux/JDK
/Install/Linux/MySQL
/Install/Linux/Tomcat
/Install/Windows
/Install/Windows/IntelliJ IDEA
/Install/Windows/JDK
/Install/Windows/MySQL
/Install/Windows/Tomcat
```

V tomto adresári sa nachádza použité vývojové prostredie a softvérové systémy potrebné pre prevádzku systému OntoDiff na platforme x86 a operačných systémoch Linux a Windows. Jedná sa o vývojové prostredie IntelliJ IDEA 5.0 (časovo obmedzená skúšobná verzia), vývojový balík JDK 1.5, databázový server MySQL 4.0 a servlet kontajner Apache Tomcat 5.0. Pokiaľ by ste mali záujem prevádzkovať tento systém na inej platforme, sú k dispozícii inštalačné balíky a aj zdrojové kódy na týchto adresách:

- pre IntelliJ IDEA: http://www.jetbrains.com/idea/download/,
- pre JDK: http://java.sun.com/j2se/1.5.0/download.jsp,
- pre MySQL: http://dev.mysql.com/downloads/mysql/4.0.html,
- pre Jakarta Tomcat: http://jakarta.apache.org/site/downloads/downloads_tomcat-5.cgi.

Podadresár *Java* obsahuje pre prípadných záujemcov aj všetky použité rámce ako je iBatis, Jakarta Struts, Xalan, Jena a MySQL Connector for Java.

/OntoDiff

```
/OntoDiff
/OntoDiff/bin
/OntoDiff/javadoc
/OntoDiff/ontology
/OntoDiff/src
/OntoDiff/sql
/OntoDiff/xsd
/OntoDiff/xsl
```

V tomto adresári sa nachádza implementácia funkčného prototypu systému OntoDiff. V podadresári *bin* sa nachádza binárna verzia systému, ktorá je nasaditeľná priamo na aplikačnom serveri či servlet kontajneri. Podadresár *javadoc* obsahuje programátorskú dokumentáciu systému OntoDiff. Zverejniteľné testovacie verzie ontológií sa nachádzajú v podadresári *ontology*. V podadresári *src* sa nachádzajú zdrojové kódy systému vo forme projektu pre vývojové prostredie IntelliJ IDEA, ale je ich možné migrovať do akéhokoľvek iného vývojového prostredia. V podadresári *sql* sa nachádza dump databázy pre databázový server MySQL, ktorú systém využíva. Podadresár *xsd* obsahuje XML Schemu exportného formátu a podadresár *xsl* šablóny na generovanie HTML formátu z výstupu relatívneho porovnávania.

Príloha F Príspevok na ITAT 2005

TURY, Michal – BIELIKOVÁ, Mária: *Identifikácia zmien v ontológiách*. In Proc. of ITAT 2005 – Workshop on Theory and Practice of Information Technologies, P. Vojtáš (Ed.), september 2005, Račkova dolina, pp.381-390.