TECHNICKÁ UNIVERZITA LIBEREC

Fakulta Mechatroniky a mezioborových inženýrských studií

DIPLOMOVÁ PRÁCE

Ukládání a přenos metadat a implementace jejich prohledávání v prostředí J2EE

Studijní program: **N 2612 – Elektrotechnika a informatika**

Studijní obor: **1802T007 – Informační technologie**

Zpracoval: **Bc. Václav Dlouhý**

Vedoucí práce: **Ing. Igor Kopetschke**

Datum odevzdání: květen 2007

UNIVERZITNÍ KNIHOVNA
TECHNICKÉ UNIVERZITY V LIBERCI

3146089419

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky a mezioborových inženýrských studií

atedra aplikované informatiky Akademický rok: 2006/2007

ZADÁNÍ DIPLOMOVÉ PRÁCE

ěno a příjmení: Václav Dlouhý

udijní program: N 2612 – Elektrotechnika a informatika

or: 1802T007 – Informační technologie

Vedoucí katedry Vám ve smyslu zákona o vysokých školách č.111/1998 Sb. určuje tuto plomovou práci:

ázev tématu: **Ukládání a přenos metadat a implementace jejich prohledávání v prostředí J2EE**

Zásady pro vypracování:

Rozbor problematiky sémantického webu, ontologie a metadat
Použití webových služeb jako nástroje získávání informací z metadat ve formátu XML/RDF
Implementace a použití webových služeb v prostředí J2EE + Tomcat + Axis
Praktická tvorba repository s metadaty za použití LOM a jejich prohledávání za použití výše uvedených technologií

TECHNICKÁ UNIVERZITA V LIBERCI
Univerzitní knihovna
Voronežská 1329, Liberec 1
PSČ 461 17

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo). Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé DP a prohlašuji, že **s o u h l a s í m** s případným užitím mé diplomové práce (prodej, zapůjčení apod.).

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

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím diplomové práce Ing. Igorem Kopetschke, kterému bych tímto rád poděkoval za řadu cenných rad a pomoc při odstraňování chyb.

V Liberci dne 18.5. 2007 Bc. Václav Dlouhý

Abstrakt

Cílem práce je vytvoření aplikace na bázi webové služby, která dokáže procházet repository s metadaty, vyhledávat relevantní informace a vracet požadované výsledky ve formátu RDF/XML. Text je koncipován do třech hlavních kapitol, přičemž první část je věnována oblasti sémantického webu, metadat a jejich aplikací. Ve druhé části jsou představeny standardy a technologie webových služeb a jejich implementace. V souvislosti s předchozí částí jsou uvedeny také možnosti integrace sémantiky do webových služeb. Třetí část je věnována tvorbě webové služby v prostředí J2EE, Apache Tomcat a Apache Axis.

Klíčová slova

Metadata, sémantický web, ontologie, RDF, RDFS, OWL, Dublin Core, LOM, Edutella, QEL, webové služby, SOAP, WSDL, UDDI, sémantické webové služby, J2EE, Apache Axis.

Abstract

The object of the thesis is to create an application on the web service basis, which could browse repositors with metadata, find the relevant information and return the required outputs in RDF/XML format. Text is conceived in three main chapters, the first one is dedicated to semantic web area, metadata and its application. In the second part of thesis there are presented the standards and technologies of the web service and its implementation. In the connection there are stated the possibilities of semantics integration into the web services. The last part is devoted to the web service creation in J2EE, Apache Tomcat and Axis environment.

Keywords

Metadata, Semantic Web, Ontology, RDF, RDFS, OWL, Dublin Core, LOM, Edutella, QEL, Web Services, SOAP, WSDL, UDDI, Semantic Web Services, J2EE, Apache Axis.

Obsah

Kapitola 1

Úvod

Současný trend rozvoje Internetu sebou přináší řadu problémů. Původní návrh technologie webu jako sítě hypertextových odkazů začíná být vlivem takřka exponenciálního růstu internetových zdrojů nedostačující. Internet se stal zdrojem obrovského množství informací, kdy však na úkor kvantity často trpí kvalita poskytovaných dat. Důsledkem je pak stále obtížnější nalezení relevantních informací.

Tento stav dal vzniknout několika desítkám nových technologií a myšlenek, které by lépe odrážely dnešní význam Internetu. Jednou z hlavních iniciativ pod záštitou W3C je vize tzv. sémantického webu, kde by bylo možné publikovaná data nejen číst, ale také jim přiřadit strojově čitelný význam. Sémantické informace by přitom byly vyjádřeny prostřednictvím metadat, neboli zjednodušeně řečeno „dat o datech". Význam těchto dat je dán tzv. ontologií, což je v podstatě systém pro strojové zachycení určitého výseku reálného světa. Základem pro formální vyjádření pak zprostředkovává rámec pro reprezentaci webových metadat RDF.

Dalším významným trendem současného Internetu jsou webové služby, které lze považovat za obdobu distribuovaných systémů RPC, CORBA nebo RMI. Z pohledu toku informací jde prakticky o opačný přístup toho současného, kdy uživatel „míří" za informacemi, kdežto webové služby poskytují data přímo uživateli, tedy informace „jdou" za uživatelem. Dalším specifickým rozdílem je oddělení datové složky od prezentační, což mimo jiné výborně zapadá do koncepce sémantického webu. Primárním cílem webových služeb je standardizace komunikačních prostředků, které umožňují bezproblémovou komunikaci mezi různými systémy na různých platformách. K dosažní tohoto požadavku je tato technologie tvořena třemi součástmi: komunikačním protokolem SOAP, jazykem WSDL pro popis rozhraní služeb a mechanizmu UDDI, který vznikl za účelem publikování služeb.

Oba nové přístupy tvorby webu se mohou velmi dobře doplňovat, což bude názorně demonstrováno v této práci jako její hlavní přínos. Společným pojítkem je především XML. V čtvrté kapitole bude popsána aplikace na bázi webové služby, která dokáže procházet repository obsahující metadata uložená dle standardu LOM, vyhledávat relevantní informace a vracet požadované výsledky ve formě RDF/XML. Aplikace je však navržena tak, aby nepředjímala žádné konkrétní formáty dotazů, implementaci repository ani formáty výsledků.

Jinými slovy jde o službu realizující obecné dotazovacího rozhraní nad obecným úložištěm dat, kdy konkrétní formáty záleží pouze na konkrétní implementaci.

Kapitola 2

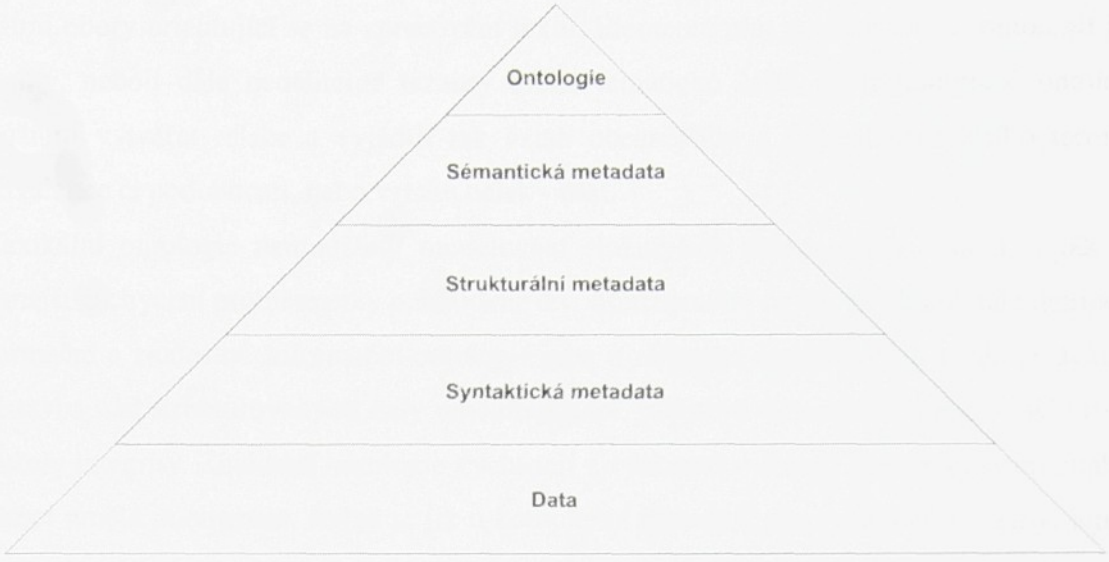
Metadata

2.1 Definice a klasifikace metadat

Metadata lze chápat jako prostředek pro popis libovolného dokumentu. Problematika metadat vychází z pojmů jako katalogizační popis či záznam, resp. bibliografický popis či záznam, s nimiž pracují katalogizátoři v knihovnách již více než 150 let. S digitalizací informačních zdrojů, rozvojem informačních technologií, zejména pak Internetu a jeho služeb, nabraly metadata zcela nový rozměr. V síťové komunikaci jsou metadata již běžně používány např. v hlavičkách, kdy každý přenášený objekt nese sebou určité informace o sobě samém.

Často se setkáváme s nepřesnou definicí, která říká, že metadata jsou data o datech. Taková definice je pravdivá, avšak příliš zjednodušující. Je zřejmé, že největší význam mají metadata v prostředí WWW. Pokud omezíme metadata pro popis webových zdrojů, lze definici formulovat takto: jde o stroji srozumitelné informace o webových zdrojích nebo dalších věcech. Tuto definici formuloval jeden z tvůrců současné architektury WWW T. Bernes-Lee. Pro pochopení definice může být důležité uvědomit si rozdíl mezi výrazy stroji srozumitelné a stroji čitelné, který byl užíván v minulosti. Stroj by tedy měl být schopen zpracovávaným datům přiřadit i jejich význam. Připravuje se budování tzv. sémantického webu (viz dále) - webu strojům srozumitelných informací o čemkoliv: lidech, věcech, pojmech, faktech, myšlenkách atd. To však předpokládá sjednocení sémantiky dokumentů, jejího formálního vyjádření a to nezávisle na platformě. Právě nezávislost informačního obsahu na formě dat je jednou ze zásadních výhod použití metadat. Druhá důležitá vlastnost spočívá ve faktu, že i metadata jsou data a nesou tedy informaci. Vzniká popis určité informační oblasti, který může být použit k nejrůznějším úsudkům a odvozování o daných datech, aniž bychom k vlastním datům vůbec přistupovali.

Dle úrovně abstrakce, se kterou metadata popisují obsah, můžeme klasifikovat metadata jako ontologie, sémantická, strukturální a syntaktická metadata (obrázek č.1).



Obrázek č.1 – Typy metadat

2.2 Ontologie

Ontologie je pojem pocházející z filozofie 18. století, kde označuje nauku o bytí, o tom „co je". Tento termín však nabyl v kontextu informačních technologií do značné míry odlišné role. Ontologie představují nejvyšší formu metadat a současně jsou klíčovým principem sémantického webu.

Definici pojmu ontologie vyslovil T. Gruber a následně modifikoval W. Borst takto: „Ontologie je formální, explicitní specifikace sdílené konceptualizace." Pojem konceptualizace říká, že pracujeme s určitým výsekem reálného světa, který musí být definován explicitně. Formalizace poukazuje na použití jazyka s přesně definovanou syntaxí. Požadavek na sdílení vyjadřuje, že ontologie není individuální záležitostí, nýbrž je výsledkem určitého konsensu. Jinými slovy ontologie je určitým systémem zachycení reality, který je přenosný a je možné ho sdílet.

Nutným předpokladem aplikace ontologií je analýza základních jmenných prvků určujících entity zkoumané oblasti, vztahů mezi nimi, případně jejich popisu. Právě takové aparáty v informační technologii nazýváme ontologiemi.

Historicky se ontologie rozpadají na tři hlavní oblasti poukazující na vývoj tradičnějších oborů. Terminologické (lexikální) ontologie jsou svázány především s knihovnictvím a dalšími obory orientující se na zpracování textu. Elementárním prvkem těchto ontologií jsou termíny, neboli dále nedělitelné tezaury určité tematické části. Terminologické ontologie umožňují vytvářet relace a vyjádřit tak vztah obecnějšího a specializovanějšího termínu, ekvivalence či podobnosti, nebo vztahu celek - část.

Lexikální ontologie neumožňují modelování složitějších doménových vztahů, a tak pro přesnější zachycení problematiky používáme tzv. konceptuální ontologie, které dále dělíme na informační a znalostní. Informační ontologie jsou rozšířením databázových schémat. Jakožto nadstavba nad strukturovanými daty umožňují např. pojmové dotazování, nebo vyšší úroveň kontroly integrity. Znalostní ontologie vycházejí z výzkumu v oblasti zaznamenávání znalostí v rámci umělé inteligence. Jedná se již o komplexní formálně propracované systémy, jejichž vazba na reálný objekt je oproti informačním ontologiím volnější. Možnost vytváření složitějších vztahů klade vyšší nároky také na formální jazyky, které definují relace většinou prostřednictvím hierarchie tříd.

Tento historický podtext v klasifikaci ontologií se v současnosti vzájemným prolínáním oborů částečně stírá, avšak stále usnadňuje orientaci v použitých přístupech.

Návrh homogenní ontologie pro popis všech a všeho je samozřejmě neřešitelný úkol. V praxi nejčastěji vytváříme popis určitého výseku reality (domény), kdy hovoříme o doménové ontologii. Předmět zkoumané domény může být vymezen jak šířeji (např. problematika celé firmy) tak úžeji (např. problematika určitého produktu). Tzv. generické ontologie se snaží o zachycení nejobecnějších pojmů a vztahů (problematika času, uspořádání a skládání objektů). Je zřejmé, že generická ontologie poslouží jako taxonomický základ či prostředek pro propojení ontologií nižší úrovně (např. doménových). Další kategorií jsou ontologie úhlové, které se, narozdíl od doménových a generických ontologií zaznamenávající věci „tak jak jsou", zaměřují na procesy odvozování (diagnostika, plánování). Poslední možností klasifikace dle předmětu konceptualizace jsou ontologie aplikační, jako sjednocení doménových a úhlových ontologií adaptovaných na konkrétní aplikaci.

Jedním z problémů při vytváření ontologií je problém rozsahu, což vyjadřuje skutečnost, že relevantních pojmů k dané doméně může být nezvládnutelné množství. V takovém případě provádíme selekci pojmů, které jsou těsněji spjaty s podstatou popisovaného jevu, nebo rozdělení ontologie do více nezávislých modulů. Další problém interakce souvisí se situací, kdy mezi doménovými pojmy existuje možnost usuzovat další znalosti. Měl by se tedy zvážit

kompromis mezi použitím statických pojmů a jejich dynamickou interakcí, která má výrazný vliv na efektivitu aplikace.

2.3 Sémantická metadata

Sémantická metadata popisují doménově závislé informace, které jsou odvozené přímo od dané ontologie. Jednotlivým elementům přiřazují smysluplnou interpretaci. Pokud např. pracujeme nad obchodní doménou, relevantní sémantická metadata mohou být jméno firmy, e-mail, produkt, cena atd.

Interoperabilita, neboli schopnost spolupracovat (např. na úrovni aplikací), z pohledu sémantiky může být řešena mapováním metadat tabulkou nebo schématem, který představuje sémantické mapování polí nebo datových prvků z jednoho ontologického standardu do druhého. Mapování má samozřejmě smysl pouze mezi prvky stejného nebo podobného významu.

Klasifikace sémantických metadat kopíruje zaměření popisované problematiky, rozlišujeme např. administrativní metadata, metadat pro právní nároky, technická metadata aj.

Historicky první aplikací sémantických metadat na webu byl standard PICS, jehož primární zaměření spočívalo v hodnocení nevhodného obsahu internetových zdrojů pro děti a mládež. Na podporu hodnocení vznikl specielní software, hodnotící služby a webové servery. Nasazení sémantických metadat však teoreticky přináší další široké možnosti využití: katalogizaci webových serverů a knihoven, s čímž souvisí zdokonalení vyhledávacích služeb a indexace (vyhledáváme pouze konkrétní význam hesla), filtraci informací, sdílení a výměnu informací mezi softwarovými agenty, ochranu autorských práv, vznik sítě vzájemné důvěry (web of trust), usnadňující elektronické obchodování nebo týmovou práci na dálku atd.

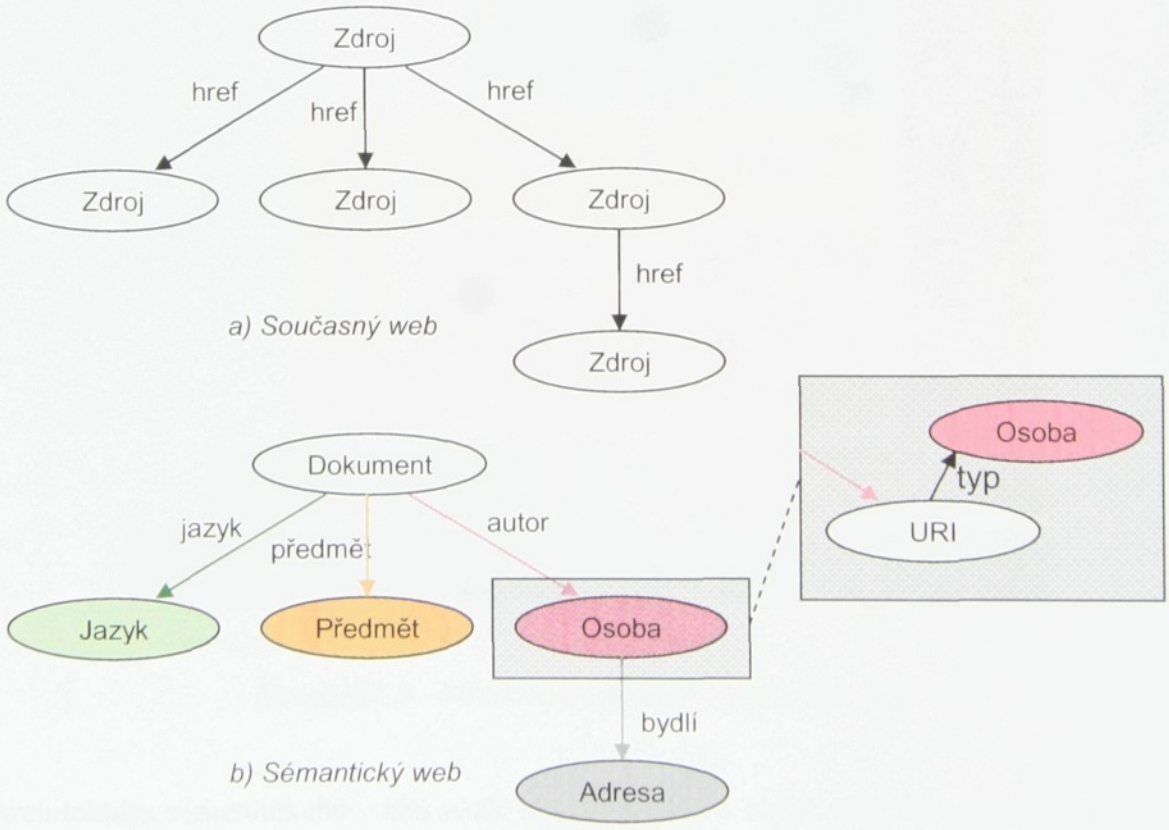
2.4 Strukturální a syntaktická metadata

Strukturální metadata se zaměřují na strukturu dokumentu, čehož lze využít při ukládání, zpracování, prezentaci (např. typ dokumentu, předmět). Tyto informace zjednodušují vyhledávání informací, korektní zobrazení či kontrolu správnosti. Typickými příklady jsou XML Schema a DTD. Syntaktická metadata popisují nekontextuální informace o obsahu

dokumentu. Častými elementy jsou velikost, umístění, bitrate nebo datum vzniku. Tyto data nám poskytují lepší pohled na data a dobře se uplatňují při katalogizaci či kategorizaci.

2.5 Sémantický web

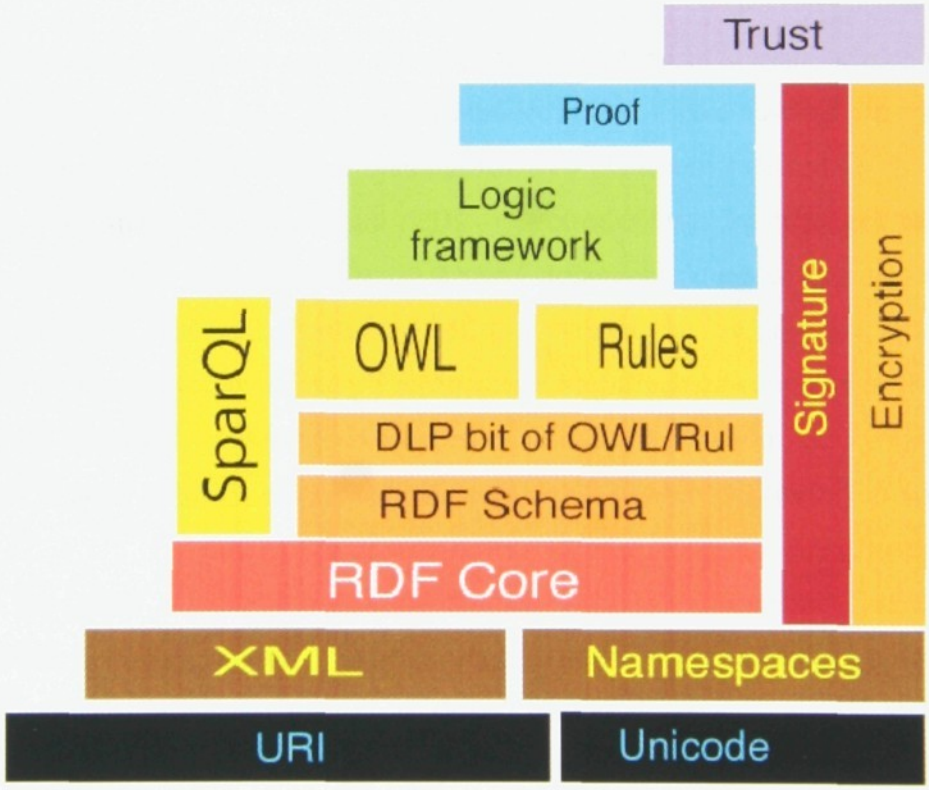
S vývojem internetu se ukazuje, že informace poskytované v rámci sítě WWW jsou neuspořádané a nelze zaručit jejich spolehlivost. Příčina současného stavu souvisí se samotným vznikem webu, který byl navržen pro publikování výsledků vědeckých prací početně omezené skupiny akademiků. Web se vyznačuje velmi snadným způsobem publikování informací, vysokou rozšiřitelností (v současnosti prakticky exponenciální rychlostí) a odolností vůči globálnímu zhroucení. Naproti těmto pozitivům utrpěla zejména integrita, úplnost a kvalita poskytovaných dat.



Obrázek č.2 - Zdroje a jejich relace na webu

Tím vznikla iniciativa pro vznik tzv. sémantického webu, který se snaží vnést do chaotického prostředí webu využitelnou sémantiku (ontologii). Vznik sémantického webu předpokládá zavedení formální struktury pro vyjadřování sémantických schémat, což by umožnilo strojům (tzv. agentům), aby data nejen četly, ale dokázaly jim také přiřadit význam.

Současný web je tvořen decentralizovanou strukturou hypertextových odkazů. Právě vhodnou volbou URL lze vyjádřit význam uložených dat, což přes značný nárůst webových zdrojů napomáhá udržet současný web jako použitelný zdroj informací. K určení obsahu odkazovaného dokumentu uživatel často požívá také kontext, ve kterém je odkaz uveden. Získat takové informace strojově prakticky nelze. Sémantický web zachovává strukturu zdroj – odkaz, avšak oběma volitelně přiřazuje význam prostřednictvím URI, která definuje konkrétní typ zdroje a relace (obrázek č.2).



Obrázek č.3 – Vrstvy sémantického webu (cit. [3])

Architekturu sémantického webu tvoří vrstvy jazyků, o nichž platí, že jazyk z vyšší vrstvy využívá a rozšiřuje jazyk z vrstvy nižší (Obrázek č. 3). První vrstva specifikuje používání mezinárodního kódování Unicode a adresaci pomocí URI (Universal Resource Identifier). Další vrstvu představuje XML s definicí namespaces a schémat, což umožňuje vyšším vrstvám implementovat další standardy založené na XML. Primární datová struktura pro

uchování faktů o objektech je zajištěna RDF. Teprve nad touto vrstvou leží vrstva ontologická, formalizovaná RDFS (RDF Schema), ontologickým jazykem OWL (Ontology Web Language). Dotazování do RDF databází implementuje jazyk SparQL. Vrstva digitálního podpisu a šifrování zajišťuje platnost dokumentu. Další vrstvy konceptu zatím nejsou standardizovány, týkají se odvozování na základě pravidel vycházejících ze zkušeností a ověřování věrohodnosti zdrojů.

2.6 Formální jazyky a standardy sémantického webu a ontologií

2.6.1 Extensible Markup Language (XML)

Rozvoj metadat na webu je do značné míry závislý na sjednocení komunikačních standardů. Nástup značkovacího jazyka XML (a jeho předchůdce HTML), jako nástroje pro výměnu dat, lze označit jako klíčový. Komunikace systémů různých platforem prostřednictvím binárních protokolů se přes snahu o standardizaci (EDI – Electronic Data Interchange) ukázala býti slepou vývojovou větví. XML dokument je textový dokument, který kombinuje čitelná data se strojově čitelnými značkami. Použitím XML pro výměnu dat se na jednu stranu o něco zvýší množství přenášených bytů, což při možnostech dnešního hardware není příliš podstatné (navíc HTTP protokol podporuje kompresi), na stranu druhou poskytuje snadnou rozšiřitelnost o nové položky, díky čitelnosti např. lepší ošetření výjimečných stavů, chyb, apod.

Standard pro zápis semistrukturovaných dat XML vznikl zobecněním jazyka HTML. Zatímco HTML definuje gramatiku i slovní zásobu jazyka, XML specifikuje v podstatě pouze interpunkci, způsob definice gramatiky a slovní zásobu ponechává na uživateli. Vysoká flexibilita XML dovoluje jak zápis málo strukturovaných dokumentů podobných HTML, tak vysoce strukturovaných dat, např. výstupu z relační databáze.

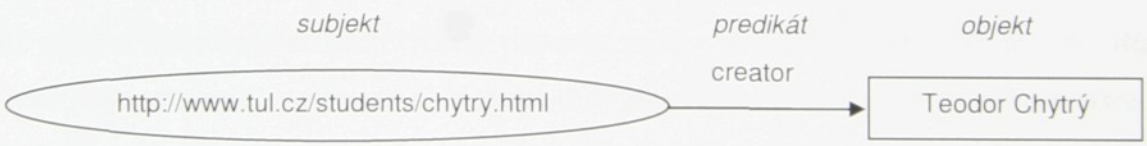
XML tedy prakticky představuje mechanismus pro reprezentaci jiných jazyků. Jako spodní vrstva sémantického webu tak nabízí dobrý základ pro implementaci RDF a ontologických jazyků.

2.6.2 Resource Description Framework (RDF)

RDF představuje obecný, volně šiřitelný rámec pro reprezentaci webových metadat, který vyvíjí konsorcium W3C. Základní syntaxe doporučovaná W3C je založena na XML, obecně však nepředjímá žádnou konkrétní reprezentaci. U vazby RDF/XML zaznamenáváme jednotlivá tvrzení tzv. serializací, kde jednotlivé prvky jsou specifickým způsobem řazeny za sebe do elementů a atributů XML.

Datový model RDF je založen na trojici subjekt - predikát - objekt, která se dohromady označuje jako tvrzení. Univerzálním prvkem RDF je zdroj, identifikovaný svým URI. Tento způsob identifikace zdrojů může navodit domněnku, že zdroj je síťově dostupný objekt (např. webová stránka, obrázek, či služba). Není to však podmínkou a jako zdroj lze použít i věc mimo web (např. osobu, výrobek z katalogu) nebo abstraktní pojem, jejichž URI plní pouze funkci jednoznačného identifikátoru. Zdroje vystupují jak v roli subjektu, tak v roli objektu. Predikáty, jež rovněž identifikujeme pomocí URI, odpovídají sledovaným vlastnostem subjektů a objekty hodnotám těchto vlastností. Objektem může být kromě zdroje také literál, tj. hodnota primitivního datového typu.

Tvrzení v RDF tvoří orientovaný graf. Na obrázku č.4 je uveden příklad tvrzení, že tvůrcem zdroje identifikovaného adresou `http://www.tul.cz/studenti/chytry.html` je `Teodor Chytrý`.



Obrázek č.4 – Tvrzení v RDF

Odpovídající tvrzení zapíšeme v RDF/XML následovně:

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:dc="http://purl.org/dc/elements/1.1/">
    <rdf:Description rdf:about=" http://www.tul.cz/studenti/chytry.html">
 <dc:Creator>Teodor Chytrý</dc:Creator>
   </rdf:Description>
</rdf:RDF>
```

V příkladu jsou definovány dva jmenné prostory: `rdf` pro konstrukty samotného RDF a `dc` pro metadatový standard Dublin Core, z něhož je převzata sledovaná vlastnost „být tvůrcem". Samotné tvrzení zaznamenáváme v elementu `rdf:Description`, kde subjekt určuje atribut `rdf:about`, predikát vnořený element `dc:Creator` a objekt odpovídá jeho obsahu.

Potřebujeme-li zapsat více tvrzení o jednom zdroji, můžeme to učinit zahrnutím několika subelementů v rámci elementu `rdf:Description`. V RDF lze také definovat prázdné uzly, tedy věci, které nemají URI referenci. Tyto uzly se používají jako agregátory dalších tvrzení a umožňují tak vyjádření i vícenásobných relací. Nejčastější implementace agregace spočívá v tom, že pomocný prázdný uzel (např. adresa) odkazuje atributem `rdf:nodeID` na jiný element `rdf:Description`, jehož atribut `rdf:nodeID` definuje relaci a obsahuje subelementy agregovaných vlastností (např. ulice, město). Na běžné (neprázdné) elementy definujeme identifikátor atributem `rdf:ID`. Pak se lze na daný element odkazovat pomocí URI, použitím konstrukce `URI_Dokumentu#ID_Elementu`.

Pro upřesnění hodnoty literálů se typicky používá datových typů XML Schema. Za tímto účelem většinou definujeme entitu či jmenný prostor XML Schema s konvenčně používaným prefixem `xsd`. Datový typ elementu specifikuje atribut `rdf:datatype`.

Častým požadavkem bývá také možnost věci různě klasifikovat či třídit do kategorií. Tato myšlenka odpovídá v objektově orientovaných programovacích jazycích příslušnosti ke třídám. V RDF lze podobnou hierarchii implementovat pomocí předdefinované vlastnosti `rdf:type`. Jakmile je součástí tvrzení vlastnost `rdf:type`, subjekt se stává instancí třídy určené atributem `rdf:resource`. V rámci třídy jsou definovány další třídy nebo konkrétní vlastnosti, přičemž jednomu zdroji je možné přiřadit i více tříd. Samotné RDF neposkytuje prostředky pro definici tříd. Pro tyto účely použijeme např. RDF Schema nebo ontologické jazyky DAML+OIL či OWL.

Za účelem seskupování tvrzení či zdrojů RDF implementuje kontejnery a kolekce. Kontejner je zdroj, jež obsahuje členy ve formě literálů nebo zdrojů (včetně prázdných uzlů). RDF nabízí celkem tři předdefinované typy kontejnerů: `rdf:Bag` pro neuspořádaný seznam, `rdf:Seq` pro uspořádaný seznam (např. abecedně) a `rdf:Alt` pro seznam alternativ. Jednotlivé položky seznamu uvozuje element `rdf:li`. První dva kontejnery reprezentují např. tvrzení ve tvaru „Studentem na TUL je Teodor Chytrý, Bruno Koubek a Zbyšek Mazák". Kontejner alternativ umožňuje zapsat tvrzení ve tvaru: „Dokument je dostupný na *URI_1* nebo *URI_2*". Abychom dobře pochopili význam kontejnerů, je dobré zdůraznit, že

na rozdíl od seznamů v tradičních programovacích jazycích, kontejner v RDF většinou zastupuje něco, co skutečně existuje.

Omezením kontejnerů je neexistence konstrukce, která by vyjadřovala skutečnost, že právě zapsané členy seznamu jsou kompletním popisem zdroje, neboli že žádní další členové již neexistují. Kontejner je tedy vždy otevřený seznam. Uzavřené strukturované seznamy se implementují pomocí kolekcí. Kolekce je zdroj předdefinovaného typu `rdf:List`, vlastností `rdf:first`, `rdf:rest` a zdroje `rdf:nil`. Význam jednotlivých komponent je zřejmý z jejich názvu a neliší se např. od seznamů v programovacím jazyce Lisp. Kolekce můžeme implementovat přiřadíme-li zdroji, který obaluje prvky kolekce, atribut `rdf:parseType="Collection"`, či použitím sítě identifikátorů společně s vlastnostmi `rdf:first` (ukazující na zdroj typu položka seznamu), `rdf:rest` (ukazující na zdroj typu `rdf:List`) a prázdného seznamu `rdf:nil` (v případě posledního prvku).

Někdy potřebujeme učinit tvrzení o tvrzení, tedy tvrzení, jehož subjektem je nějaké dříve zapsané tvrzení (např. kdo dané tvrzení vyslovil). Tento postup se nazývá reifikace. Za účelem reifikace použijeme předdefinovaný typ `rdf:Statement` a trojici vlastností `rdf:subject`, `rdf:predicate`, `rdf:object`, jejichž hodnoty identifikují původní tvrzení. Takto definovaný zdroj `rdf:Statement` pak bude subjektem nového tvrzení. Jestliže chceme např. zapsat informaci o autorovi tvrzení, zdroj `rdf:Statement` bude mít definovány vlastnosti `rdf:subject`, `rdf:predicate`, `rdf:object` a např. `dc:Creator`. Je dobré si uvědomit, že proces reifikace pracuje s konkrétní instancí tvrzení, nikoliv s nějakým tvrzením, které odpovídá uvedeným reifikačním vlastnostem. V situaci, kdy dokument obsahuje v různém kontextu dvě tvrzení se stejným subjektem, predikátem i objektem, nastávají problémy, jejichž řešením bývá identifikace tvrzení aplikačně závislými prostředky.

Na závěr kapitoly uvedeme příklad v RDF/XML, který popisuje školní kurzy - název, počet studentů, seznam doporučených zdrojů a jejich popis. Příklad je poněkud vyumělkovaný, avšak demonstruje většinu uvedených konstrukcí – agregaci, identifikaci, kontejner, reifikaci, typování literálů a zdrojů.

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema/"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:lom-technical="http://ltsc.ieee.org/2002/09/lom-technical#"
  xmlns:moje="http://www.tul.cz/schemata/kurzy#"
  xml:base="http://www.tul.cz/kurzy"
>

<!--Popis kurzu WWW1 se seznamem literatury jako neuspořádaným seznamem
    a typovaným literálem pro počet přihlášených studentů-->
<rdf:Description rdf:about="#WWW1">
  <rdf:type rdf:resource="http://www.tul.cz/schemata/kurzy#kurz" />
  <dc:Title>Technologie WWW</dc:Title>
  <moje:pramen>
    <rdf:Bag>
      <rdf:li rdf:resource="#src13"/>
      <rdf:li rdf:resource="#src21"/>
      <rdf:li rdf:resource="#src55"/>
    </rdf:Bag>
  </moje:pramen>
  <moje:pocetStudentu rdf:datatype="xsd:decimal">15</moje:pocetStudentu>
</rdf:Description>

<!--Detailnější popis zdroje, typ může zastoupit rdf:Description-->
<moje:elektronicky_zdroj rdf:ID="src13">
  <dc:Title>Protokol HTTP</dc:Title>
  <moje:format rdf:nodeID="agregace_prav_src13"/>
</moje:elektronicky_zdroj>

<!--Agregace vlastností pomocí prázdného uzlu-->
<rdf:Description rdf:nodeID="agregace_prav_src13">
  <lom-technical:Format>text/html</lom-technical:Format>
  <lom-technical:Size>1226</lom-technical:Size>
  <!--...atd....-->
</rdf:Description>

<!--Reifikace určující datum výroku že na kurzu WWW1 je 15 studentů-->
<rdf:Statement rdf:about="#datum_ps">
  <rdf:subject rdf:resource="http://www.tul.cz/kurzy#WWW1" />
  <rdf:predicate
        rdf:resource="http://www.tul.cz/schemata/kurzy#pocetStudentu" />
  <rdf:object rdf:datatype="xsd:decimal">15</rdf:object>
  <dc:date>2007-01-30</dc:date>
</rdf:Statement>
</rdf:RDF>
```

2.6.3 RDF Schema (RDFS)

Jazyk RDF nám umožňuje zapisovat jednoduchá tvrzení o zdrojích prostřednictvím vlastností a hodnot. RDFS představuje nadstavbu, která doplňuje RDF o možnosti specifikace tříd zdrojů a vlastností, jež tyto třídy popisují. V RDFS lze vytvořit hierarchii tříd v podobném významu jako např. v jazyku Java, čímž dostáváme nástroj pro zachycení určité domény (ontologie). Používání předdefinovaných tříd a vlastností RDFS zprostředkujeme zavedením jmenného prostoru typicky s předponou `rdfs` a URI

http://www.w3.org/2000/01/rdf-schema# (seznam všech předdefinovaných tříd a vlastností RDF a RDFS viz Příloha 1).

Chceme-li definovat novou třídu použijeme třídu rdfs:Class s identifikátorem rdf:ID, který využijeme při vytváření instance třídy uvedením vlastnosti rdf:type. Podtřídu lze vytvořit prostřednictvím vlastnosti rdfs:subClassOf a jejího atributu rdf:resource. Vytváření podtříd je tranzitivitní, tzn. definujeme-li třídu *b*, jako podtřídu *a*, třídu *c* jako podtřídu *b*, pak platí, že *c* je podtřídou třídy *a*. Jinými slovy všechny instance třídy *c* jsou zároveň instancemi třídy *b* i *a*.

Vyjádření vlastnosti se provádí vně definice třídy tvrzením, jehož subjektem je třída rdf:Property s atributem vymezující název vlastnosti (typicky rdf:ID). RDFS dále definuje tři základní vlastnosti jako predikáty subjektu rdf:Property pro konkrétnější vymezení vlastností: rdfs:range, rdfs:domain a rdfs:subPropertyOf. Hodnotou vlastnosti rdfs:range vyjadřujeme obor hodnot vlastnosti, kterým může být jiná třída nebo literál. Jestliže uvedeme více vlastností rdfs:range, pak hodnota vlastnosti musí být instancí všech těchto tříd. Element rdfs:domain určuje definiční obor vlastnosti, tedy u kterých tříd se daná vlastnost vyskytuje. Znovu jich může být více, pak musí být třída, jež má tuto vlastnost implementovanou, instancí všech tříd uvedených jako rdfs:domain. Není-li uvedena žádná definice rdfs:domain, můžeme ji začlenit do kterékoliv třídy. Analogicky ke třídám je možné definovat konstrukcí rdfs:subPropertyOf také podvlastnosti pro upřesnění významu. Použijeme-li pak v nějakém tvrzení hodnotu podvlastnosti, bude tvrzení platné i pro všechny její nadvlastnosti. Je zřejmé, že u podvlastnosti se již rdfs:range a rdfs:domain neuvádí, protože ty jsou dány původní vlastností (hierarchicky nejvyšší nadvlastností).

Jako příklad poslouží schéma vztahující se k předchozímu příkladu o kurzech. Vytvoříme zde tři třídy – kurz, zdroj a podtřídu třídy zdroj elektronicky_zdroj, pro nějž navíc definujeme vlastnost format. Za povšimnutí také stojí explicitní definice datových typů a kontejnerů.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://www.tul.cz/schemata/kurzy"
>

<rdfs:Class rdf:ID="kurz" />

<rdfs:Class rdf:ID="zdroj" />

<rdfs:Class rdf:ID="elektronicky_zdroj">
  <rdfs:subClassOf rdf:resource="#zdroj" />
</rdfs:Class>


<rdf:Property rdf:ID="pramen">
  <rdfs:range rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag" />
  <rdfs:domain rdf:resource="#kurz" />
</rdf:Property>

<rdfs:Container rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag" />

<rdf:Property rdf:ID="pocetStudentu">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#decimal" />
  <rdfs:domain rdf:resource="#kurz" />
</rdf:Property>

<rdfs:Datatype rdf:about="http://www.w3.org/2001/XMLSchema#decimal" />

<rdf:Property rdf:ID="format">
  <rdfs:domain rdf:resource="#elektronicky_zdroj" />
</rdf:Property>

</rdf:RDF>
```

Tento přístup při vytváření tříd a vlastností podobný objektově orientovaným jazykům, přináší však i několik rozdílů. V první řadě jsou definice třídy a vlastnosti oddělené, tedy vlastnost není přímo součástí třídy. Výše uvedeným způsobem sice lze specifikovat, u jaké třídy se určitá vlastnost může vyskytnout, pokud však chceme, aby každá instance třídy měla vždy definovanou určitou vlastnost, jediným řešením je ošetření na úrovni aplikace. Dalším nepříjemným důsledkem filosofie RDFS je, že není možné definovat různé obory hodnot vlastností v závislosti na konkrétní instanci třídy. Nevýhodou také může být absence přímé implementace datových typů. Tyto nedostatky odstraníme použitím některého vyššího ontologického jazyka.

2.6.4 OWL (Onthology Web Language)

Základním principem sémantického webu jsou ontologie. Výrazové prostředky, kterými RDFS popisuje ontologie, nejsou pro strojového zpracování sémantiky dostačující. Jazyk OWL vychází z hlediska popisu ontologie ze specifikace DAML+OIL, formální vyjádření

pak přebírá z konceptu RDFS. Tím vznikla technologie vyvíjená pod záštitou W3C, která má velmi dobré výrazové možnosti a může se snadno kombinovat s ostatními technologiemi sémantického webu (XML, XML Schema, RDF, RDFS). V rámci specifikace OWL rozlišujeme tři podjazyky OWL Lite, OWL DL a OWL Full, přičemž platí, že dokument vyjádřený jazykem nižší třídy vyhovuje specifikaci třídy vyšší.

OWL Lite je omezen na „minimální množinu" jazyka za účelem snadnějšího vyjádření ontologie, což má v prostředí Internetu obzvláště logické opodstatnění. Oslabeny jsou především možnosti využívání anonymních tříd a omezení na kardinalitu, u kterých byla v případě DAML+OIL empiricky zjištěna malá míra využívání. Specifikace OWL DL (Description Logic) nabízí maximální možnosti expresivity při garanci dosažitelnosti a rozhodnutelnosti všech operací (např. je-li třída podtřídou jistých tříd, nemůže být instancí třídy jiné). Značné schopnosti formalizace ontologie jsou taktéž doménou specifikace OWL Full, která však není, analogicky k RDF, tak syntakticky restriktivní.

Následující odstavce budou věnovány základnímu popisu konstrukcí OWL Lite. Definice tříd je převzata z RDFS, což zaručí zpětnou kompatibilitu. Zopakujme tedy, že jde o třídy – `rdfs:Class`, `rdfs:subClassOf`, `rdf:Property`, `rdfs:subPropertyOf`, `rdfs:range` a `rdfs:domain`. Třída `owl:Individual` označuje třídu všech instancí tříd (individuí), přičemž každá instance je podtřídou kořenového prvku označeného `owl:Thing`. OWL také definuje prázdnou třídu `owl:Nothing`. Novou třídu jako průnik dvou stávajících tříd můžeme vytvořit pomocí `owl:intersectionOf`.

Ekvivalenci dvou tříd vyjádříme prostřednictvím konstrukce `owl:equivalentClass` (např. tvrzením `ns1:Člověk – owl:equivalentClass – ns2:Osoba`). Množina individuí ekvivalentních tříd je stejná. Analogicky lze také vytvořit ekvivalentní vlastnosti pomocí `owl:equivalentProperty`. Pokud potřebujeme vytvořit vícenásobné pojmenování jedné instance (individua), poslouží element `owl:sameAs`. Naopak pro vyjádření odlišnosti subjektu a objektu použijeme predikát `owl:differentFrom`. Jestliže je třeba tuto odlišnost specifikovat pro více zdrojů, je výhodné použít seznam `owl:DiferentAll`, jehož všechny prvky jsou vždy po dvou rozdílné. Prvky seznamu obaluje element `owl:distinctMembers`. Těmito konstrukty způsobem můžeme vytvářet distribuované ontologie, které propojíme právě pomocí zmíněných ekvivalencí.

V jazyku OWL je implementována také několik tříd za účelem detailnější charakteristiky vlastností. Třídy `owl:ObjectProperty` a `owl:DatatypeProperty` jsou třídami všech vlastností, jejichž objektem je individuum, respektive literál. Inverzní vlastnosti

označuje direktiva `owl:inverseOf`. Máme-li např. definovány vlastnosti *P1* a *P2* jako inverzní a dále platí, že *X* je svázáno s *Y* vlastností *P1*, pak bude také *Y* svázáno s *X* vlastností *P2*. Dále je možné vytvářet tranzitivní vlastnosti pomocí `owl:TransitiveProperty`, anebo vlastnosti symetrické `owl:SymetricProperty`. Pokud některou vlastnost označíme typicky pomocí `rdf:type` jako `owl:FunctionalProperty`, znamená to, že daná třída může mít vlastnost definovánu maximálně jednou. Jinými slovy hodnota vlastnosti je funkcí, která nemusí být vždy definována. Můžeme také označit inverzní vlastnost funkční vlastnosti elementem `owl:inverseFunctionalProperty`.

Omezení na vlastnosti klademe jednak z hlediska oboru hodnot, nebo omezujeme kardinalitu. Všechna tyto omezení jsou individui třídy `owl:Restriction`. Omezovanou vlastnost určuje element `owl:onProperty`. Restrikci oboru hodnot definují elementy `owl:allValuesFrom` *či* `owl:someValuesFrom`, které umožňují, aby objekty vlastnosti náležely do definované množiny všechny, nebo ve druhém případě alespoň jeden z nich. Kardinalitu, neboli minimální a maximální počet individuí připojených k dané vlastnosti, omezíme konstrukcemi `owl:cardinality` (přesná kardinalita), `owl:minCardinality` a `owl:MaxCardinality`. V OWL Lite mohou tyto omezení nabývat pouze hodnot 0 a 1.

Specifikace OWL DL a OWL Full implementují navíc `owl:oneOf` pro definici třídy výčtem prvků, `owl:disjointWith` označuje dvě třídy jako různé, `owl:hasValue` omezuje hodnoty vlastnosti na jedno individuum a umožňuje definici tříd množinovými operacemi - `owl:unionOf` (sjednocení tříd), `owl:complementOf` (doplněk tříd) a `owl:intersectionOf` (průnik tříd). Podporována je také libovolná hodnota kardinality.

2.6.5 Přehled dalších ontologických jazyků

Pro potřebu formální reprezentace ontologie bylo vyvinuto několik desítek jazyků. Mezi nejvýznamnější patří projekty Cyc, Ontolingua, OCML (Operational Conceptual Modelling Language), které vycházejí z jazyka LISP, nebo OKBC (Open Knowledge Base Connectivity) a XOL (eXtensible Ontology Language) jako znalostní analogie k ODBC a XML.

Z hlediska sémantického webu se však počítá s jazyky, které jsou specielně navržené k implementaci ontologií v prostředí WWW. V roce 1995 vznikl historicky první takový jazyk SHOE (Simple HTML Ontology Extension). Pomocí vlastní sady elementů začleňoval

do zdrojového kódu webových stránek jednak metadata, která popisovala obsah stránky (sémantická metadata), tak i vlastní ontologie tyto metadata popisující. V přibližně stejné době vzniká také projekt Ontobroker, který na rozdíl od SHOE nepoužívá jednotný jazyk pro ontologie a anotace. Zatímco jazyk anotací je velmi jednoduchým obohacením HTML o dodatečné atributy (nikoliv elementy), jazykem ontologií je značně propracovaný formalismus tzv. *F-Logic*. V dnešní době již značně zastaralé SHOE a Ontobroker nejsou kompatibilní s RDF, což předurčuje jejich využití maximálně v prostředí intranetu.

V polovině roku 2000 byl oficiálně zahájen projekt DAML (DARPA Agent Mark-Up Language), jehož cílem bylo vytvořit ontologický jazyk pro RDF s větší vyjadřovací silou než má RDFS. Vznikl jazyk nazvaný DAML-ONT, který zahrnoval již celou řadu konstrukcí pro vytvoření hierarchie tříd a vlastností. Souběžně s rostoucím významem RDF se dospělo k potřebě postavit nové jazyky na některém propracovaném logickém kalkulu, který by umožňoval konstrukci složitějších podmínek při zachování výhodných vlastností pro výpočty. Volba padla na deskripční logiku, která se stala základem jazyka nazvaného OIL (Ontology Inference Layer). Jeho sloučením s DAML-ONT vznikl jazyk nazvaný DAML+OIL. Jak již bylo uvedeno výše, jedná se o předchůdce jazyka OWL, tedy jazyka s podobnou formalizací jako OWL, jehož konstrukty již byly popsány.

Existují ještě další jazyky pro konceptuální modelování, aniž by se v souvislosti s nimi mluvilo o ontologiích. Asi nejpoužívanější současný standard pro datové modelování UML může být např. v podnikovém prostředí velmi dobrým řešením pro vznik ontologie. Rozdílů oproti specializovaným jazykům je hned několik. V prvé řadě procedury nejsou součástí ontologie, kde je kladen důraz především na vytváření vztahů mezi třídami, jež mohou mít i složitější sémantiku, než poskytuje UML. Toto souvisí převážně se skutečností, že UML je primárně určen k tvorbě softwarových systémů, zatímco ontologie by měla sloužit k zachycení vztahů relativně nezávisle na systému. Podobný charakter má také rozdíl v chápání instancí. Zatímco v UML je instance napevno spojena s třídou, v ontologiích instance (individuum) odpovídají objektům reálného světa, jejichž třída je pouze jakousi vlastností a může se časem případně i měnit.

2.7 Aplikace metadat

2.7.1 Dublin Core (DC)

Dublin Core je metadatový standard pro popis zdrojů. Pojem zdroj zde chápeme analogicky k RDF, proto se nemusí jednat pouze o síťově dostupný dokument, nýbrž o libovolný objekt, u nějž mají pochopitelně definované vlastnosti smysl. Velmi flexibilní sadu elementů tvoří 15 volitelných a libovolně opakovatelných základních prvků, tvořících tzv. jednoduchý Dublin Core (viz Tabulka č. 1). Tyto elementy je také možné dále rozšiřovat, kdy obvykle disjunktní podmnožiny tvoří rozklad původního elementu. Standard DC byl přeložen do několika desítek jazyků včetně češtiny.

Element	Popis
Název (title)	Jméno dané zdroji
Tvůrce (creator)	Entita primárně odpovědná za vytvoření obsahu zdroje
Datum (date)	Datum spojené s určitou událostí během existence zdroje
Předmět a klíčová slova (subject)	Téma obsahu zdroje
Vydavatel (publisher)	Entita odpovědná za zpřístupnění zdroje
Formát (format)	Fyzická nebo digitální reprezentace zdroje
Popis (description)	Vysvětlení obsahu zdroje
Přispěvatel (contributor)	Entita, která přispěla k vytvoření obsahu zdroje
Identifikátor zdroje (identifier)	Jednoznačný odkaz na zdroj v rámci daného kontextu
Typ zdroje (type)	Povaha nebo druh obsahu zdroje
Správa autorských práv (rights)	Informace o právech k popisovanému zdroji
Jazyk (language)	Jazyk intelektuálního obsahu zdroje
Zdroj (source)	Odkaz na zdroj, z něhož je popisovaný zdroj odvozen
Vztah (relation)	Odkaz na příbuzný zdroj
Pokrytí (coverage)	Rozsah nebo záběr obsahu zdroje

Tabulka č. 1 – Elementy Dublin Core

2.7.2 Learning Object Metadata (LOM)

Základní myšlenka elektronické výuky spočívá ve znovupoužitelnosti výukových materiálů. Tomu odpovídá koncept tzv. výukových objektů (*learning objects*). Definic výukového objektu existuje mnoho, jejich společnou podstatu však můžeme shrnout takto: Výukové objekty jsou úzce specializované balíčky, které je možné použít k osvětlení dané

problematiky nezávisle na prostředí a kontextu. Popíšeme-li tyto objekty vhodnými metadaty, bude možné např. skládat tyto objekty do komplexnějších entit, zefektivní se služby poskytování či vyhledávání apod. K těmto účelům je pochopitelně nutné standardizovat sadu elementů, která nabídne dostatečné výrazové prostředky. Typicky vycházíme z obecné sady Dublin Core jako základu pro implementaci metadatového standardu výukových objektů, jímž je např. standard LOM (Learning Object Metadata) nebo některá jeho podmnožina např. specifikace SCO (Shareable Content Objects).

Learning Object Metadata je příkladem standardu, který dospěl ze stádia specifikace IMS Learning Resource Metadata do stádia oficiálního standardu. Standard specifikuje syntaxi a sémantiku metadat pro výukové objekty, které jsou zde definovány jako znovupoužitelné, distribuované, digitální i nedigitální entity pro podporu výuky, vzdělávání nebo tréninku. Tím je dán velmi široký záběr standardu, protože popisované objekty mohou být digitální soubory, knihy, osoby apod.

LOM definuje celkem 77 elementů v devíti hlavních kategoriích (viz Tabulka č. 2). Vznikne tak stromová hierarchie, v níž pouze listy mají definovanou hodnotu. Stejně jako u Dublin Core jsou všechny elementy volitelné, což má za následek, že i když nemá výukový objekt definovány žádná metadata, odpovídá standardu LOM. V praxi se proto často používají aplikační profily, které zohledňují specifické požadavky prostředí, ve kterém vznikly (např. ARIDANE pro Evropu, UK LOM Core pro Velkou Británii, CanCore pro Kanadu atd.). K zápisu metadat se většinou používá RDF.

Pro strojové zpracování metadat je důležitá nejenom standardizovaná sada elementů, ale také specifikace hodnot, které jednotlivé elementy mohou nabývat. To je však část poněkud komplikovanější a v současné době stále rozpracovaná.

Kategorie	Charakteristika elementů
General	Základní popis výukového objektu jako celku *(Title, Language, ...)*
Life Cycle	Historie, současný stav a vývoj objektu *(Version, Status, ...)*
Meta-metadata	Popis vlastní metadatové instance (ne objektu) *(Metadata Schema, ...)*
Technical	Technické požadavky a charakteristika objektu *(Size, Location, ...)*
Educational	Výuková a pedagogická charakteristika *(Typical Age Range, ...)*
Rights	Intelektuální vlastnictví a podmínky použití *(Cost, ...)*
Relation	Vztahy s ostatními objekty *(Resource, Identifier, Catalog, ...)*
Annotation	Komentář k použití objektu *(Description, ...)*
Classification	Vztah vůči určitému klasifikačnímu systému *(Purpose, Taxon Path, ...)*

Tabulka č.2 – Kategorie LOM

2.7.3 Edutella

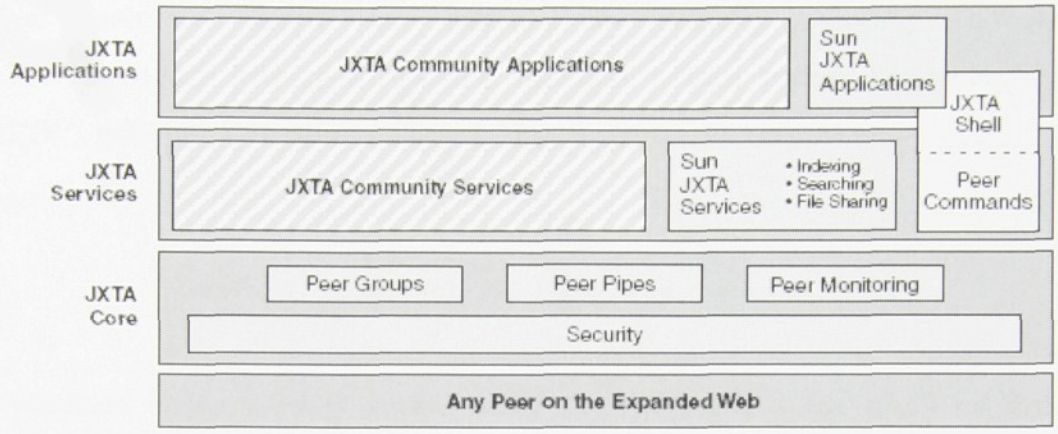
Edutella je Peer-to-Peer (P2P) síťová infrastruktura založená na RDF. Ačkoli jsou metadata ve webovém prostředí velmi užitečná, v P2P sítích můžeme jejich použití označit za klíčové. Informační zdroje v P2P netvoří strukturu jako v případě hypertextu. Data jsou poskytována prostřednictvím dotazu na určitého klienta (pozn. v dalším textu označení klient odpovídá anglickému peer), kdy je vhodné předem znát jaký klient jaké služby nabízí. Proto by každý zdroj, který je nabízen v rámci P2P sítě, měl mít vlastní metadatový popis. Toho lze snadno dosáhnout u specializovaných sítí (např. pro výměnu hudebních souborů), složitější situace pak nastává u komplexnějších aplikací (jako např. výměna výukových objektů).

Projekt Edutella řeší nedostatky způsobené velkým počtem metadatových elementů obecnějších objektů implementací dotazování do RDF metadat pro P2P sítě založených na technologii JXTA (platformově nezávislá open-source implementace P2P firmy Sun). Vize projektu spočívá v implementaci služeb poskytující metadata, jako nutného předpokladu vytvoření interoperabilní sítě mezi různými JXTA aplikacemi, přičemž první fáze projektu se zaměřuje na výukové objekty (standardy LOM, IMS, SCROM).

Architektura klientů sítě Edutella staví na vrstveném modelu rámce JXTA (viz Obrázek č.5), který tvoří sada protokolů založených na XML. Vrstva jádra (JXTA Core) zabezpečuje vytvoření a základní běh P2P sítě, vrstva služeb (JXTA Services) rozšiřuje možnosti jádra o indexaci, vyhledávání nebo sdílení obsahu (např. pevného disku klienta) a vrstva aplikační (JXTA Applications) pak využívá a rozšiřuje předchozí vrstvy o další služby konkrétní aplikace. Do této architektury dobře zapadá aplikační schéma Edutelly. Na vrstvě služeb (Edutella Services), jež je popsaná jazyky jako WSDL či DAML-S, definujeme formáty a protokoly pro výměnu dat (zasílání dotazů, jejich výsledků a dalších metadat). Repository, anotační nástroje nebo grafické rozhraní náleží do vrstvy aplikační (Edutella Peers).

Každý klient v síti distribuuje metadata jako množinu RDF tvrzení. Charakteristika klienta je dána sadou určitých služeb, které nabízí. Základem každého klienta je implementace dotazovací služby (Query Service), kdy jednotliví klienti registrují druh dotazu (specifikací standardu „tento klient poskytuje metadata standardu LOM 1.0" nebo specifikací vlastnosti „tento klient poskytuje metadata dc_title(X,Y)"). Tyto specifikace jsou odeslány klientům, kteří projevili zájem o daný druh dotazu. Edutella Query Service je zamýšlen jako univerzální dotazovací mechanismus pro výměnu RDF metadat, ať už v rámci jednoho klienta do lokálního repository, anebo jako interface do distribuovaných RDF repository. Jedním z hlavních předpokladů je abstrakce od lokálního dotazovacího jazyka (např. SQL, RQL,

TRIPLE). Syntaxi a sémantiku takového rozhraní poskytují dotazovací jazyk QEL (Query Exchange Language) a ECDM (Edutella Common Data Model), přičemž připojení a překlad do lokálních formátů klienta je implementován pomocí wrapper API.



Obrázek.5 – JXTA Framework (cit.[10])

Interní reprezentace dotazů a jejich výsledků přebírá sémantiku z jazyka Datalogu, což je neprocedurální dotazovací jazyk odvozený od jazyka Prolog. Datalog je stejně jako Prolog odvozen od Hornovy klauzule (disjunkce literálů) a neobsahuje žádné funkční symboly. Program v Datalogu tvoří množina výroků (kde každé pravidlo tvoří jeden kladný literál (hlava) a jeden nebo více záporných literálů (tělo)), množina faktů (jeden kladný literál) a dotazovací literál (jeden nebo více záporných literálů). Literály zde mají význam predikátů popisující vztah mezi proměnnými či konstantami, které v RDF terminologii představují objekty a subjekty (např. `title(http://www.tul.cz/kurzy#WWW1, 'Technologie WWW')`. Každý výraz se tedy skládá z jednoho literálu tvořící hlavu (predikát) a sjednocení libovolného počtu kladných literálů tvořících tělo. Průnik vyjádříme zapsáním několika pravidel se stejnou hlavou. Dotaz dostaneme naopak sjednocením dotazovacích literálů. Koncepce Datalogu, kdy data jsou seskupována okolo vlastností, umožňuje snadné mapování do relačních databázových jazyků, jako je SQL.

V následujícím příkladě bude demonstrován jednoduchý příklad aplikace Datalogu. Vycházejme ze znalostní báze:

```
dc:title(http://www.tul.cz/kurzy#src13, 'Protokol HTTP').
type(http://www.tul.cz/kurzy#src13, elektronicky_zdroj).
dc:title(http://www.tul.cz/kurzy#src28, 'Protokol TCP/IP').
type(http://www.tul.cz/kurzy#src28, elektronicky_zdroj).
dc:title(http://www.tul.cz/kurzy#src32, 'Hypertext Transport').
type(http://www.tul.cz/kurzy#src32, zdroj).
```

Nyní položme dotaz, ve kterém nás zajímají všechny prameny typu `elektronicky_zdroj` a současně vlastnosti `title` rovnou konstantě `'Protokol HTTP'`, nebo jakýkoliv zdroj s názvem `'Hypertext Transport'`:

```
pramen(X) :- type(X, elektronicky_zdroj),
             dc:title(X, 'Protokol HTTP').
pramen(X) :- dc:title(X, 'Hypertext Transport').
?- pramen(X).
```

Výsledkem budou všechny možné vazby na proměnnou `X` (na rozdíl od Prologu, kde výsledkem je pouze první vazba):

```
pramen(http://www.tul.cz/kurzy#src13)
pramen(http://www.tul.cz/kurzy#src32)
```

Syntaxi distribuce dotazů mezi klienty definuje QEL – RDF dotazovací jazyk založený na sémantice Datalogu. QEL není vázán pouze na projekt Edutella, je univerzálně použitelný a bude aplikován v praktické části práce.

Jazyk QEL má několik předdefinovaných predikátů (zavedeme jmenný prostor s prefixem `qel` odkazující na `http://www.edutella.org/qel#`). V první řadě jsou definovány dva predikáty tvořící nové vazby na proměnné. V konstrukci `qel:s(X,Y,Z)` proměnná `X` představuje subjekt, `Y` predikát a `Z` objekt, přičemž na místě subjektu a predikátu povolujeme anonymní či adresovatelné zdroje, v případě objektu je možné navíc použít RDF literál. Chceme-li se dotázat na příslušnost ke kontejneru, použijeme predikát `qel:member(X, Member)`, kde `X` je kontejner a `Member` zdroj. Další předdefinované predikáty již nové vazby nevytvářejí, nýbrž omezují ty již zavedené. K určení dvou identických uzlů RDF grafu byla definována konstrukce `qel:equals(X,Y)`. Potřebujeme-li omezit zdroje z hlediska jejich typu, použijeme `qel:nodeType(X,Type)`, pro který platí, že `Type` je neanonymní zdroj a nikoliv proměnná. QEL specifikuje čtyři třídy typů zdrojů (`Type`): `qel:Literal` (RDF literál zapíšeme do uvozovek, jazyk lze specifikovat prefixem `@`, datový typ `^^` - např. `"Sweden"@en`, nebo `"1024"^^<xsd:int>`), `qel:Resource` a jeho disjunktní podmnožiny `qel:AnonymousResource` (např. `_:xyz`) a

`qel:NonAnonymousResource` (např. `<dc:title>`). Datový typ definovaný atributem RDF literálu (typicky XML Schema) zjistíme `qel:dataType(X, DataType)`. Další možností omezujícího predikátu je `qel:stringValue(X, String)`, který je pravdivý v případě, že `X` i `String` jsou literály tvořené stejnou řetězcovou hodnotou. Konstrukci `qel:like(X,S)` lze použít ve dvou případech, přičemž ani v jednom nesmí `S` vystupovat jako proměnná. Pokud je `X` literál, musí obsahovat `S` jako svůj podřetězec. V případě, že `X` je adresovatelný zdroj, musí jeho URI obsahovat řetězec `S`. Numerické porovnání literálů zprostředkují predikáty `qel:lessThan(X, Y)` a `qel:greaterThan(X, Y)`. Možný je také omezit jazyk literálů dle standardu [RFC 3066] predikátem `qel:language(X,Lang)`. Negaci omezujících predikátů zapíšeme operátorem „-“. Jestliže se chceme dotázat např. na všechny anglické zdroje obsahující v názvu řetězec HTTP, dotaz může vypadat takto:

```
?- qel:s(X, <dc:title>, Z), qel:like(Z, "HTTP"), qel:language(Z, "en")
```

Stejně jako Datalog i QEL umožňuje zápis omezujících pravidel, avšak každý predikát by měl mít vlastní URI:

```
moje:pramen(X)  :- qel:s(X, <rdf:type>, <moje:elektronicky_zdroj>),
 qel:s(X, <dc:title>, Z),
 qel:stringValue(Z, "Protokol HTTP")
moje:pramen(X)  :- qel:s(X, <dc:title>, Z),
 qel:stringValue(Z, "Hypertext Transport")
?- moje:pramen(X)
```

U složených dotazů není ojedinělý případ, kdy pracujeme s hodnotou, která se však nemusí vyskytovat u všech vrácených hodnot. Řešením je vnější spojení (outer join), v podobném významu jak ho známe z relačních databází a SQL, kde hodnota, v případě že není definována, nabývá hodnoty *null*. V Datalogu vnější spojení znamená, že takto označený predikát je vždy pravdivý. QEL označuje takto připojené predikáty symbolem „*“:

```
?- qel:s(X, <dc:title>, Z), qel:like(Z, "Informatika"), qel:s*(X, <dc:subject>, S)
```

Na rozdíl od Datalogu QEL specifikuje způsob vrácení výsledku dotazu. Můžeme totiž specifikovat pořadí proměnných, na které se dotazujeme. Vrácená sada výsledků (result set) pak toto pořadí musí respektovat. Pokud pořadí nespecifikujeme, je vygenerováno a výsledek dotazu obsahuje všechny proměnné uvedené v těle dotazu. Hodnotu *null* může výsledná

proměnná nabývat ve třech případech: uvedeme-li ji v hlavičce pravidla, nikoliv v jeho těle; uvedeme-li ji v pořadí proměnných dotazu, ale ne v těle dotazu; jedná-li se o vnější spojení dotazovacího literálu. Provedení dotazu v QELu vlastně znamená, že všechny proměnné z těla dotazu jsou nahrazeny konkrétními hodnotami, a pokud jsou daná tvrzení pravdivá, jsou tyto hodnoty zařazeny do výsledku. Pokud se chceme dotázat např. na URI kurzu s názvem „Technologie WWW" a počet jeho studentů:

```
?(Kurz, Pocet)  - qel:s(Kurz, <dc:title>, X),
                  qel:StringValue(X, "Technologie WWW"),
                  qel:s(Kurz, <moje:pocet_studentu>, Pocet)
```

s výsledkem:

```
{(<http://www.tul.cz/kurzy#WWW1>, "15")}
```

Nikoliv všechny implementace QELu podporují všechny konstrukce. Rozlišujeme pět základních úrovní složitosti dotazů: 1. dotazy neobsahující žádné pravidlo (Rule-less Query); 2. dotazy obsahující maximálně jedno pravidlo na jeden predikát, čímž nelze aplikovat průnik (Conjunctive Query); 3. dotazy s neomezenými pravidly bez rekurze (Disjunctive Query); 4. dotazy obsahující rekurzivní predikáty, avšak rekurze je lineární, což teoreticky umožňuje transformaci do SQL99 (Linear Recursive Query); 5. dotazy s nelineární rekurzí predikátů, což vyžaduje spuštění příslušného Datalog nebo Prolog procesoru (General Recursive Query).

Všechny uvedené konstrukce QELu lze vyjádřit také prostřednictvím RDF tvrzení. V dotazech, jež označuje element `qel:Query`, rozlišujeme několik elementů: proměnné, predikáty, dotazovací literál a pravidla. Proměnné a predikáty definujeme jako instance typu `qel:Variable` respektive `qel:Predicate`, což mohou být anonymní nebo neanonymní zdroje s libovolným počtem vlastností, které však QEL nijak nepoužije. Specielním případem je konstrukce `qel:BuiltinPredicate` určující neanonymní zdroj, který identifikuje vestavěnou vlastnost QELu. Dotazovací literály rozlišujeme podle charakteru predikátu: použijeme-li specielní predikát `qel:s`, dotazovací literál bude typu `qel:StatementLiteral`, v opačném případě použijeme typ `qel:QueryLiteral`. Třída `qel:QueryLiteral` má definovány tři vlastnosti: `qel:predicate` jako instance třídy `qel:Predicate` určuje vlastnost; `qel:arguments` obsahuje kontejner `rdf:Seq`, který specifikuje hodnoty vlastnosti v dotazu, může obsahovat proměnné, RDF literály nebo zdroje; a volitelně `qel:negated` obsahuje hodnotu `"true"^^<xsd:boolean>` nebo

"false"^^<xsd:boolean> v případě, že chceme uvedenou vlastnost negovat, což lze pouze u předdefinovaných vlastností. Dotazovací literál typu qel:StatementLiteral implementuje predikát qel:s (viz formalismus Datalogu), přebírá vlastnosti z RDF reifikace: rdf:subject, rdf:predicate a rdf:object. Pro zápis pravidla použijeme typu qel:Rule a jeho vlastností: qel:head určuje hlavičku pravidla, typicky predikát a proměnné a ukazuje na instanci typu qel:QueryLiteral; libovolné množství vlastností qel:literal definující tělo pravidla typu qel:QueryLiteral nebo qel:StatementLiteral; vlastnost qel:outerJoinLiteral se shoduje s předchozí s tím rozdílem, že definuje vnější spojení. Typ celého dotazu zavádí element qel:Query s libovolným počtem vlastností, ve kterých využívá předešlých konstrukcí. V instanci dotazu qel:Query lze definovat libovolné množství pravidel qel:rule, libovolné množství vlastností qel:literal a qel:outerJoinLiteral odkazující na instance tříd qel:QueryLiteral nebo qel:StatementLiteral a jednu vlastnost qel:resultVariables, která implementuje kontejner rdf:Seq s proměnnými, jež mají být zahrnuty do výsledku dotazu. Nyní ukážeme dotaz, ve kterém nás zajímají všechny prameny typu elektronicky_zdroj a současně vlastnosti title rovnou konstantě *'Protokol HTTP'*:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:moje="http://www.tul.cz/schemata/kurzy#"
  xmlns:qel= "http://www.edutella.org/qel#"
>

<!--Definice proměnných-->
<qel:Variable rdf:nodeID="X" />

<!--Definice pravidla - dotazujeme se na zdroj typu el.zdroj s nazvem
   „Protokol HTTP“-->
<qel:Rule rdf:nodeID="pravidlo1">
   <!--Hlavička-->
    <qel:head>
      <qel:QueryLiteral>
        <qel:predicate
           rdf:resource="http://www.tul.cz/schemata/kurzy#pramen"/>
        <qel:arguments>
          <rdf:Seq>
            <rdf:li rdf:nodeID="X"/>
          </rdf:Seq>
        </qel:arguments>
   </qel:QueryLiteral>
   </qel:head>

   <!--Tělo-->
   <qel:literal>
     <qel:StatementLiteral>
        <rdf:subject rdf:nodeID="X" />
        <rdf:predicate
           rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#type"/>
        <rdf:object
           rdf:resource="http://www.tul.cz/schemata/kurzy#elektronicky_zdroj"
        />
      </qel:StatementLiteral>
   </qel:literal>

   <qel:literal>
     <qel:StatementLiteral>
        <rdf:subject rdf:nodeID="X" />
        <rdf:predicate rdf:resource="http://purl.org/dc/elements/1.1/title"/>
        <rdf:object>Protokol HTTP</rdf:object>
      </qel:StatementLiteral>
    </qel:literal>

  </qel:Rule>


  <!-Dotaz složený pouze z jednoho pravidla-->
  <qel:Query rdf:about="http://www.tul.cz/dotazy/0015">

    <qel:rule rdf:nodeID="pravidlo1"/>

    <qel:resultVariables>
      <rdf:Seq>
        <rdf:li rdf:nodeID="X"/>
      </rdf:Seq>
    </qel:resultVariables>

  </qel:Query>

</rdf:RDF>
```

Výsledek dotazu je poskytován opět v RDF jako instance třídy `qel:ResultSet`, která má definována tři typy vlastností: jednu `qel:query` jako odkaz na instanci třídy `qel:Query` identifikující dotaz; libovolný počet vlastností `qel:result`, kde každý zahrnuje kontejner `rdf:Seq` s hodnotami výsledků dotazu; a jeden element `qel:resultVariables`, což je vlastnost určující pořadí proměnných (v případě, že pořadí bylo definováno v dotazu, je tato vlastnost nepovinná). Výsledek předcházejícího dotazu bude vypadat následovně:

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:qel= "http://www.edutella.org/qel#"
>

   <qel:ResultSet>

     <qel:query rdf:resource="http://www.tul.cz/dotazy/0015"/>

     <qel:result>
       <rdf:Seq>
         <rdf:li rdf:resource="http://www.tul.cz/kurzy#src13"/>
       </rdf:Seq>
     </qel:result>

   </qel:ResultSet>

 </rdf:RDF>
```

Další služby klientů Edutelly jsou volitelné: Replication Service (poskytuje persistentní data), Mapping Service (překlad mezi jednotlivými metadatovými slovníky, zajišťuje interoperabilitu), Mediation Service (prostřednictvím metadat definuje pohledy spojující data z různých zdrojů) a Annotation Service (umožňuje vytvářet anotace jakéhokoliv zdroje v rámci sítě Edutella).

Kapitola 3

Webové služby

3.1 Základy webových služeb

Existuje mnoho definic webových služeb od technických po velmi zjednodušující. Mezinárodní webové konsorcium (W3C), jež webové služby standardizovalo, definuje webové služby asi takto: „Webová služba je softwarový systém identifikovaný svým URI, jehož rozhraní a dostupnost jsou popsány XML a lze je tak nalézt pomocí jiných softwarových systémů. Tyto systémy mohou spolupracovat se službou definovaným rozhraním prostřednictvím XML zpráv a internetového protokolu (typicky HTTP)."

Webové služby lze označit jako další stupeň přirozeného vývoje poskytování informací na webu. Nejdříve to byla síť čistých HTML stránek, později pak webových aplikací, které tyto stránky dynamicky generovaly. Silným omezením webových aplikací je však jejich pevná svázanost s jejich HTML GUI. Naproti tomu webové služby poskytují data ve formě XML, datová složka je zde zcela oddělena od prezentační, což zajišťuje interoperabilitu a snadnou integraci Webových služeb do různých aplikací.

Koncepce webových služeb vychází z technologií jako RPC, CORBA nebo RMI pro distribuované provádění kódu. RPC (Remote Procedure Call) je protokol definující způsob, jak parametry a návratovou hodnotu volané funkce posílat po síti, a tím nabízí mechanismus volání funkcí umístěných na vzdálených systémech. RPC nepodporuje přenášení strukturovaných datových typů a asynchronní komunikaci. Distribuované volání objektů CORBA či RMI umožňují pracovat se vzdálenými objekty (proxy objekty) stejně, jako by byly přímo k dispozici na lokálním systému. Existuje také jazykově nezávislý popis rozhraní objektů IDL (Interface Definition Language). Co se týče podpory různých platforem, tak RMI je technologie určená pouze pro Javu, zatímco CORBA má dobrou podporu většiny platforem. Nevýhodou těchto technologií je vyšší složitost psaní komponent a svázanost s určitou platformou (např. reprezentací datových typů).

Architektura orientovaná na služby (Service-Oriented Architecture – SOA) umožňuje sdílení softwarových prostředků definicí služby jako znovupoužitelné entity. Tento způsob většinou poskytuje lepší náhled na problematiku, než u objektově orientovaného sdílení tříd či objektů. Služby mívají charakter obecnějších komponent, jejichž funkci specifikuje formou zpráv nadefinované rozhraní. Komunikace pak může probíhat buď jako jednoduché zasílání zpráv mezi službami, nebo pro řešení komplexnějších úloh, jako sada několika vzájemně spolupracujících služeb. Tento scénář předpokládá interakci tří rolí:

- Poskytovatel webové služby nabízí služby klientům.
- Adresář obsahuje popisy služeb a umožňuje vyhledání služby.
- Klient vyhledává služby v adresáři a přistupuje k nim přímo přes definované rozhraní.

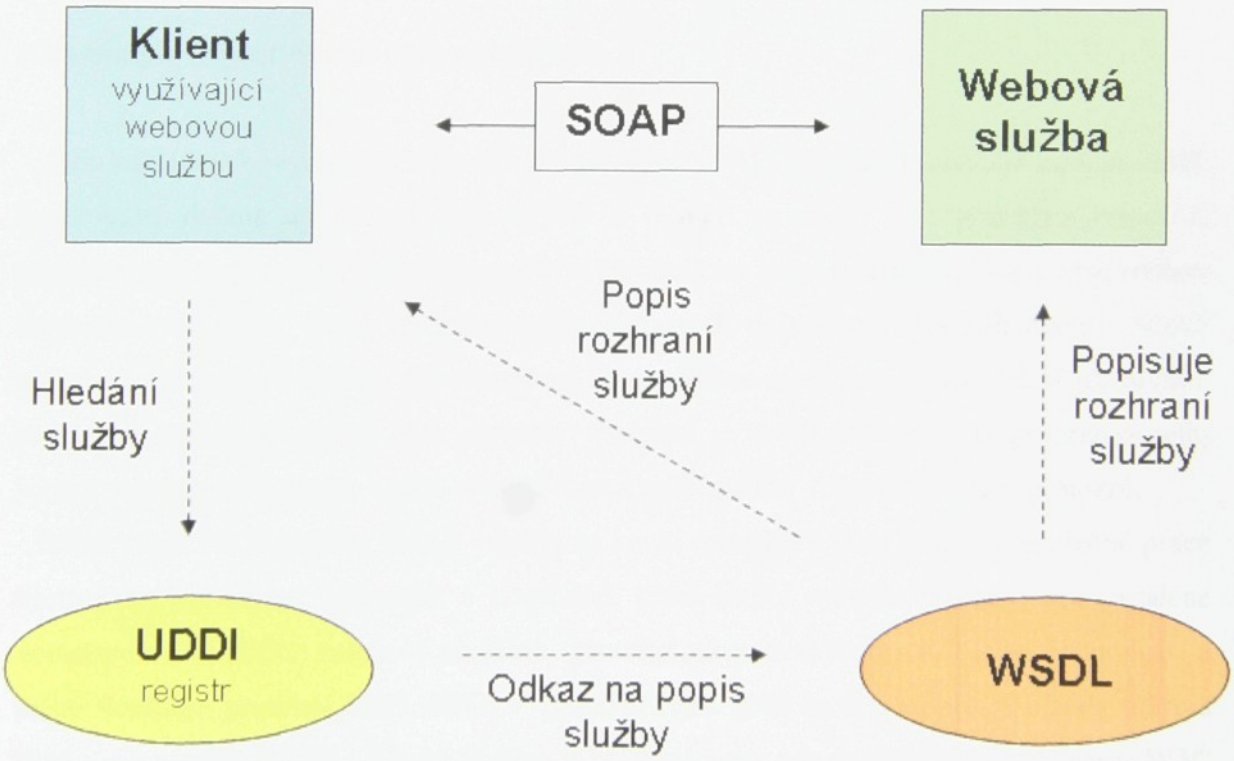
Výhodou této architektury je možnost implementovat tzv. loosely-coupled systémy, tedy systémy, kde klient a server o sobě prakticky nic nepředpokládají. Funkčnost tohoto modelu požaduje implementaci:

- Mechanizmu pro přístup klientů ke službě.
- Mechanizmu pro registraci rozličných služeb v adresáři, v němž služba prezentuje svoji existenci. Webová služba je síťově dostupná aplikace s transparentně určeným umístěním, což znamená, že klient může v adresáři dynamicky volit službu, kterou chce použít.
- Mechanizmu pro specifikaci rozhraní služby využitelné klienty.

Nasazení webových služeb přináší oproti ostatním technologiím řadu výhod, díky kterým se v současnosti těší velké popularitě. Klíčovou vlastností je zřejmě interoperabilita napříč heterogenními systémy. Model webových služeb byl navrhnut tak, aby distribuované služby byly zcela nezávislé na softwarových platformách, architekturách a mohly být vytvořeny v jakémkoliv programovacím jazyku. Dalším pozitivem je jistě snadná integrace do stávajících systémů, kdy není třeba dělat radikální změny stávajících kódů, pouze např. některé funkce zpřístupníme jako službu. Jednoduše lze také do systému integrovat již vytvořené služby. Webové služby nabízí také dobré možnosti rozšíření o nové XML technologie, nebo efektivní týmovou práci při vytváření aplikací. Většina těchto výhod tedy přímo souvisí s použitím XML, z čehož ovšem vyplývá i nevýhoda, kterou jsou vyšší výpočetní nároky při parsování XML souborů.

3.2 Technologie a standardy webových služeb

Standardem nazýváme soubor specifikací a pravidel, jež nepředjímá žádnou konkrétní implementaci. Aplikace jednotlivých standardů dávají vzniknout rozličným technologiím, což umožňuje jejich vzájemnou kooperaci nezávisle na detailech implementace. Aby byla technologie webových služeb úspěšná, musí zahrnovat široce akceptované standardy a tím zajistit interoperabilitu jejich aplikací. Jde především o to, aby klient přistupoval ke službě nezávisle na platformě, ve které je implementován, a současně ani nebyl závislý na žádné konkrétní platformě služby. Tím docílíme toho, že programy na zcela různých platformách (JavaScript, Java, C, MS .NET, mobilní telefony) budou moci spolu snadno komunikovat.



Obrázek č.6 – Technologie webové služby

V první řadě musíme zajistit, aby si poskytovatel a klient služby vyměňovali vzájemně srozumitelné informace, neboli je nutné stanovit určitý formát pro výměnu dat. Neefektivním řešením jsou různé interpretery nebo slovníky. Webové služby za tímto účelem používají univerzální značkovací jazyk XML. Dalším bodem je stanovení formátu zpráv pro vzájemnou komunikaci, jako nutného předpokladu pro komunikaci dvou stran, které o sobě původně nic

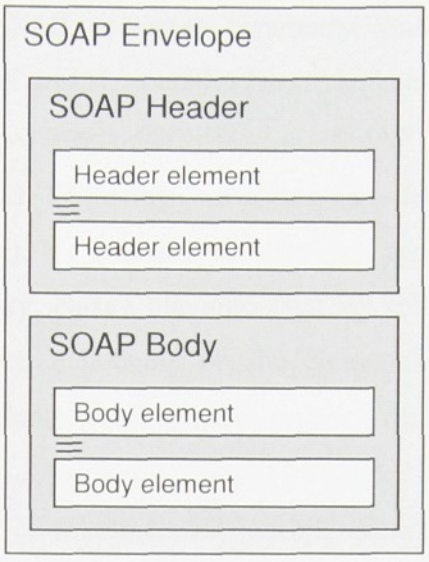
nevědí. Jinými slovy je nutné přenášená data opatřit obálkou, aby bylo zřejmé, z jakého důvodu se přenáší. Simple Access Object Protocol (SOAP) je formátem zpráv pro webové služby. Kromě toho může být také velmi výhodné specifikovat formální jazyk pro popis služeb, např. typ služby, umístění, rozhraní atd. Libovolný klient tak získá strojově čitelnou informaci o tom, jakým způsobem k vybrané službě přistupovat. V souvislosti s webovými službami se používá zejména jazyk Web Service Description Language (WSDL). Aby byl klient schopen vyhledat službu a získat její popis, musí mít k dispozici nějaké obecně známé umístění s registrem pro publikaci služeb. Přístup do registru musí být také standardizovaný. Mechanismus vyhledávání webových služeb specifikuje Universal Description Discovery and Integration (UDDI).

3.2.1 Simple Object Access Protocol (SOAP)

Jako základ webových služeb lze označit protokol SOAP. Protokol umožňuje zasílání XML zpráv mezi dvěma aplikacemi přes HTTP či SMTP protokol. Tím prakticky odpovídá principu peer-to-peer. Dle standardu probíhá komunikace jednosměrně, aplikace však mohou implementovat různé druhy interakce (dotaz/odpověď, dotaz/více odpovědí apod.). SOAP nedefinuje žádnou sémantiku pro přenášená data, představuje pouze rámec, jehož rozšířením je možné aplikačně závislé data přenášet. Protokol je zcela nezávislý na programovacím jazyku, operačním systému a jeho textová forma zajišťuje tzv. firewall-friendly protokol.

První verze 1.0 protokolu SOAP vznikla na konci roku 1999 jako výsledek společné práce firem DevelopMentor, Microsoft a UserLand, které chtěly vytvořit protokol pro vzdálené volání procedur (RPC) založený na XML. Protokol navazoval na o rok mladší, jednodušší a méně flexibilní protokol XML-RPC. V průběhu roku 2000 se k podpoře přihlásila i firma IBM a nová verze SOAP 1.1 byla zaslána W3C konsorciu, kde vznikla taky další verze W3C SOAP 1.2 z roku 2003, která má již status doporučení (tedy hotový standard).

Tělo SOAP zprávy je jednoduchý XML dokument s definovaným jmenným prostorem `env="http://www.w3.org/2003/05/soap-envelope"`. Kořenovým elementem je `env:Envelope`, který obsahuje dva sub-elementy: hlavičku `env:Header` a tělo `env:Body`. Obsah těchto elementů je aplikačně závislý a není součástí specifikace.



Obrázek č.7 – Struktura SOAP zprávy

Hlavička je nepovinnou součástí SOAP zprávy, kde typicky nese informace týkající se dalšího zpracování dat. Využití nacházejí např. pokud zpráva prochází dodatečným zpracováním přes několik uzlů (služeb), pro autentizaci (jméno, heslo), identifikaci uživatele apod. Jaká data budou umístěna v hlavičce a jaká v těle zprávy bývá vždy otázkou konkrétní aplikace. Tělo zprávy mohou tvořit jakákoliv XML data, přičemž se samozřejmě předpokládá, že koncovému příjemci zprávy bude zřejmá i jejich sémantika.

Jak již bylo řečeno, v dřívější době byl protokol SOAP spojován s technologií RPC pro vzdálené volání procedur. V současnosti je však podporováno také zasílání čistě XML zpráv, jejichž interpretace je úkolem konkrétní aplikace. Uveďme nyní příklad právě takové zprávy zaslané pomocí HTTP POST:

```
POST /Soucet HTTP/1.1
Host: www.priklad.net
Content-Type: text/xml; charset="utf-8"
Content-Length: nnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
     <x>5.2</x>
     <y>4.0</y>
  </env:Body>
</env:Envelope>
```

Velmi časté je použití SOAP jako RPC, kdy komunikace odpovídá modelu požadavek/odpověď. Klient tedy vyvolá metodu na straně služby a její návratová hodnota je

zaslána zpět opět ve formě XML, přičemž parametry, jména metod, respektive jejich návratové hodnoty jsou serializovány specifickým způsobem v těle požadavku. Způsob serializace udává atribut `env:encodingStyle`. SOAP kódování (jmenný prostor `http://www.w3.org/2003/05/soap-encoding`) definuje serializaci běžných datových typů (řetězců a čísel), ale i složených datových typů (pole, struktury), nebo lze serializovat i odkazy na objekty. Název hlavního tagu ve volání odpovídá volané metodě, vnořené tagy parametrům a jejich hodnoty obsahu elementů. Analogicky v odpovědi tag operace obaluje návratové hodnoty. Atribut `env:encodingStyle` však může nabývat i jiných hodnot, např. `http://www.w3.org/1999/02/22-rdf-syntax-ns#` pro serializaci RDF. Vzhledem k minimalizaci zdrojů nekompatibilit se v současnosti upřednostňuje typování XML Schema namísto SOAP kódování datových typů. Následující příklad demonstruje volání metody `double Secti(double x, double y)`:

```
<?xml version="1.0" encoding="utf-8"?>
<env:Envelope
  xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <env:Body>
    <prns:Secti
     xmlns:prns="http://www.tul.cz/ws/priklad2/schema"
      <prns:x xsi:type="xsd:double">5.2</prns:x>
      <prns:y xsi:type="xsd:double">4.0</prns:y>
    </prns:Secti>
  </env:Body>
</env:Envelope>
```

V příkladu vidíme použití datových typů XML Schema, tedy jednoznačný, rozšiřitelný a srozumitelný způsob typování dat. Každá volaná operace by měla mít celosvětově jedinečnou identifikaci, čehož jsme dosáhli zavedením jmenných prostorů. Odpověď na předcházející požadavek vypadá takto:

```
<?xml version="1.0" encoding="utf-8"?>
<env:Envelope
  xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <env:Body>
    <prns:SectiResponse
     xmlns:prns="http://www.tul.cz/ws/priklad2/schema"
      <prns:z xsi:type="xsd:double">9.2</prns:z>
    </prns:SectiResponse>
  </env:Body>
</env:Envelope>
```

Protokol SOAP přímo podporuje také identifikaci chybových stavů, kdy chyby aplikační i protokolu udává tag `env:Fault` v těle zprávy. Tento element má dva povinné subelementy `env:Code` specifikuje chybu a `env:Reason` uvádí, jak vznikla.

Původně byl SOAP zamýšlen pouze pro HTTP protokol, momentálně jednotlivé implementace dovolují SOAP zprávy přenášet např. i prostřednictvím SMTP, FTP, JMS, aj.

3.2.2 Web Service Description Language (WSDL)

Web Service Description Language je jazyk vycházející z XML, který nabízí standardizovaný platformově nezávislý způsob popisu webové služby. Obecně lze říci, že se jedná o XML schema týkající se popisu rozhraní, vazby a umístění služby. WSDL se tedy netýká sémantiky volaných operací, pouze popisuje syntaxi jejich volání, což umožňuje vytvářet automatizované nástroje, které z daného WSDL umí vygenerovat zástupný kód (tzv. stub) pro volání služby ve zvoleném programovacím jazyce. Vznikl sloučením tří jazyků firem IBM, Microsoft a Ariba s názvy NASSL (Network Accessable Service Specification Language), SCL (SOAP Contract Language) a SDL (Service Description Language) do jednoho jazyka nazvaného WSDL 1.1. V současnosti aktuální verze WSDL 2.0 má status kandidáta na doporučení W3C.

Struktura každého WSDL 2.0 dokumentu se skládá z několika základních tagů, jejichž základní možnosti nyní stručně popíšeme (viz příklad dále). Kořenový element se jmenuje `description`. Zde je třeba nejprve definovat potřebné jmenné prostory pro konstrukty WSDL `http://www.w3.org/ns/wsdl`, existující absolutní URI atributu `targetNamespace` by mělo odkazovat na WSDL dokument popisující danou službu (od vlastního dokumentu se může lišit v případě, že je tento popis složen z více souborů). Dále jsou použity některé jmenné prostory pro specifikaci protokolu. Elementem `types` určíme typy zpráv, konkrétně jejich názvů a datových struktur typicky odvozených od XML Schema, které přenášejí (`Secti` a `SectiResponse`). Element `interface` definuje abstraktní rozhraní služby jako sady abstraktních operací, kdy každá operace reprezentuje jednotlivé interakce mezi klientem a službou. Každá operace dále specifikuje vzor komunikace, který definuje způsob výměny zpráv. V našem příkladu jde o In-Out model, což určuje atribut `pattern="http://www.w3.org/ns/wsdl/in-out"` (další možnosti jsou In-Only či Out-Only). Elementy `input` a `output` rodičovského elementu `interface` pak stanoví,

které konkrétní zprávy se v daném modelu operace použijí. Do této chvíle jsme určili, jaké zprávy se při komunikaci použijí, nedefinovali jsme však způsob, jakým se budou přenášet po síti. K tomu je určen další potomek elementu `description` element `binding`, jehož atributy určují vazbu na rozhraní definované v předchozí části (`interface="tns:sectiInterface"`), strukturu zprávy `type="http://www.w3.org/ns/wsdl/soap"` pro SOAP 1.2 a protokol HTTP specifikujeme atributem `wsoap:protocol= "http://www.w3.org/2003/05/soap/bindings/HTTP/"`. Dále můžeme uvnitř elementu `binding` určit pro jednotlivé operace např. metodu GET či POST apod. Nakonec zbývá pouze definovat umístění služby, k čemuž slouží element `service`, kde pro každý `binding` definujeme koncový bod elementem `endpoint`.

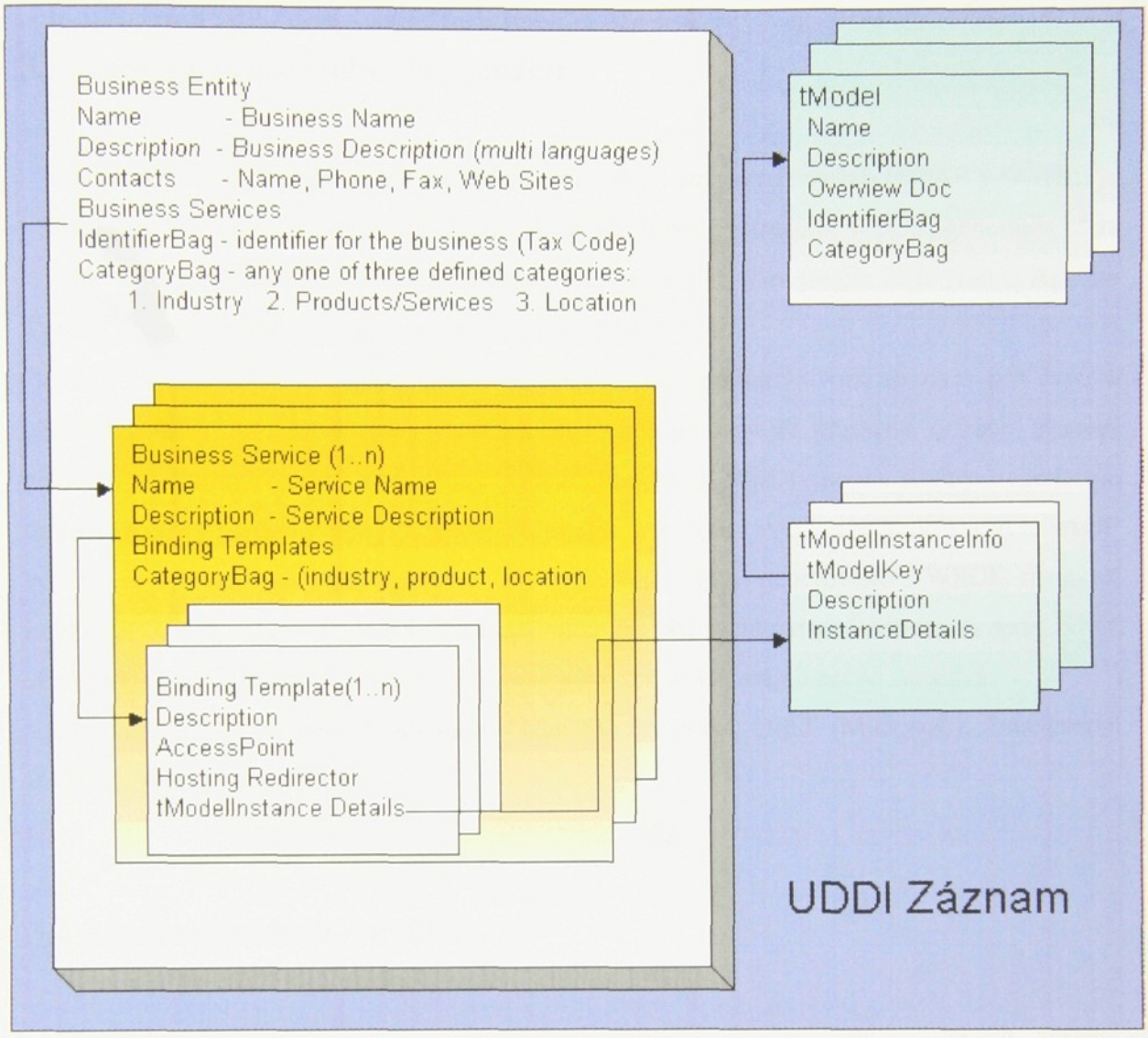
```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace="http://www.tul.cz/ws/priklad2/wsdl"
    xmlns:tns="http://www.tul.cz/ws/priklad2/wsdl"
    xmlns:prns="http://www.tul.cz/ws/priklad2/schema"
    xmlns:wsoap="http://www.w3.org/ns/wsdl/soap">

  <documentation>
 Služba sčítá dvě čísla typu double.
  </documentation>

  <types>
    <xs:schema
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace=" http://www.tul.cz/ws/priklad2/schema"
        xmlns="http://www.tul.cz/ws/priklad2/schema">

      <xs:element name="Secti" type="tSecti"/>
      <xs:complexType name="tSecti">
        <xs:sequence>
          <xs:element  name="x" type="xs:double"/>
          <xs:element  name="y" type="xs:double"/>
        </xs:sequence>
      </xs:complexType>
      <xs:element name="SectiResponse" type="tSectiResponse"/>
      <xs:complexType name="tSectiResponse">
        <xs:sequence>
          <xs:element  name="z" type="xs:double"/>
        </xs:sequence>
      </xs:complexType>

    </xs:schema>
  </types>

  <interface name="sectiInterface">
    <operation name="opSecti"
               pattern="http://www.w3.org/ns/wsdl/in-out"
               style="http://www.w3.org/ns/wsdl/style/iri">
        <input messageLabel="In"
              element="prns:Secti" />
        <output messageLabel="Out"
              element="prns:SectiResponse" />
    </operation>
  </interface>

  <binding name="sectiSOAPBinding"
           interface="tns:sectiInterface"
           type="http://www.w3.org/ns/wsdl/soap"
           wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">
  </binding>

  <service name="sectiService"
           interface="tns:sectiInterface">

     <endpoint name="sectiEndpoint"
               binding="tns:sectiSOAPBinding"
               address ="http://www.tul.cz/ws/priklad2"/>
  </service>
</description>
```

3.2.3 Universal Description, Definitions and Interface (UDDI)

Universal Description, Definitions and Interface představuje standardizovaný mechanismus pro registraci, kategorizaci a vyhledávání webových služeb (obecně pro aplikace architektury SOA). Tato technologie funguje jako adresář, který obsahuje informace o subjektech (např. firmách) a jimi poskytovaných služeb. Samotný registr pracuje rovněž jako webová služba protokolem SOAP. UDDI není standardem W3C, v současnosti je dispozici UDDI v3 pod hlavičkou OASIS Committee Specifications.

UDDI záznam definuje komplexní datovou strukturu (viz Obrázek č.8), která obsahuje informace potřebné k nalezení služby. Kompletní UDDI registr obsahuje miliony takových záznamů. Základní entitou UDDI záznamu je tzv. `Business Entity`, jež obsahuje data jako název, popis a kontakt (typicky nějaké firmy). Jednotlivé položky mohou mít charakter struktury, např. kontakt je seznamem různých kontaktních údajů. Další entita `Business Service` je součástí té předchozí a obsahuje seznam služeb, jež firma nabízí. Kromě popisu služeb obsahuje ke každé službě libovolné množství entit šablon vazeb - `Binding Template`, které odkazují na údaje týkající se vlastní implementace a umístění služeb. Položka `Access Point` šablony vazeb představuje umístění služby, je tedy jednou z nejdůležitějších. Každá šablona vazeb obsahuje také strukturu nazvanou `tModelInstanceInfo` odkazující na entitu `tModel`, což je druhá velmi důležitá sada informací pro každou službu. Jelikož jsou entity `tModel` technické specifikace popisující povahu služby vycházející z konkrétní aplikace, nabízí UDDI rozšiřitelný rámec pro jejich implementaci.

Klasický scénář vyhledání služby v UDDI registru charakterizují dva druhy metod – `find` a `get`. Vyhledávání začíná tím, že klient specifikuje oblast zájmu (produkty, kategorie, klíčová slova atd.). Výsledkem je tzv. `BussinessInfo` struktura odvozená od `Business Entity` obsahující relevantní výsledky. Dalším krokem je vytvoření seznamu služeb, které jednotlivé firmy nabízí. Klient následně poskytuje další kriteria výběru, na jejichž základě vrátí registr sadu entit `tModel`, jež tyto kriteria splňují. V zápětí klient vyvolá metody pro získání všech vazeb zvolené služby, přičemž registr odpoví zasláním `Binding Template` odkazující na WSDL soubor nebo jiný popis tzv. *fingerprint* (nebo na oba současně). Na závěr je klientovi zaslána detailní sada informací `Binding Template` metodami `get` v závislosti na výsledcích metod `find`.



Obrázek č. 8 – Struktura UDDI záznamu

Co se týče implementací UDDI registrů, tak tři centrální databáze spravovali společnosti IBM, Microsoft a SAP. Časem se však ukázalo, že velké procento záznamů je neplatných a nelze zaručit důvěryhodnost jejich poskytovatelů, což byl taky oficiální důvod jejich vypnutí. Jde v podstatě o analogický problém se zaručením věrohodnosti zdrojů z pohledu sémantického webu (viz kapitola 2.5). Dalším nedostatkem UDDI je zřejmě přílišná obecnost poskytovaných informací, kdy se zaznamenávají i informace nikterak související s webovou službou (např. telefon firmy apod.). Zřejmé jsou jistě také nevýhody spojené s tím, že jde o centrální systémy. Z těchto důvodů byly centrální UDDI registry vypnuty v lednu 2006, podpora pro privátní sítě by však měla pokračovat.

3.3 Implementace webových služeb

Komerčních i open source nástrojů pro implementaci webových služeb (SOAP a WSDL) je v současnosti k dispozici velké množství. Obsahují obvykle generátor WSDL, generátor kódu klienta a serverové části z WSDL, aplikační server, ve kterém služba běží (Jetty, Apache Tomcat apod.) a komerční nástroje obvykle také GUI.

Velké popularitě se těší především dva open source projekty: Apache Axis pro Javu a gSOAP pro C/C++. Tyto produkty však mají k implementaci poněkud odlišný přístup. gSOAP je optimalizovaný na vysoký výkon, pro daný WSDL popis služby vygeneruje optimalizovaný zdrojový kód v C nebo v C++. Naproti tomu Axis umožňuje sestavit SAOP volání i dynamicky za běhu aplikace, kdy zástupné třídy vygenerované z WSDL sestavují konkrétní SAOP volání. To je také důsledek řádově vyšší rychlosti gSOAP oproti Axis. Nová verze Axis2 je asi dvakrát rychlejší než verze původní, avšak stále značně zaostává.

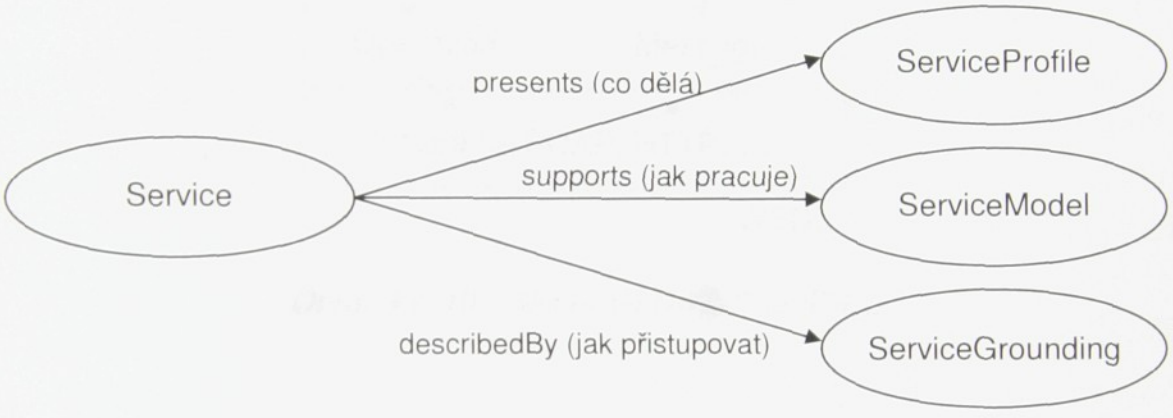
Z komerčních implementací jmenujme One Studio (Sun), .NET (Microsoft), Oracle9iAS (Oracle), Websphere (IBM).

3.4 Sémantické webové služby

Využívání webových služeb v současnosti vyžaduje velkou míru lidských zásahů, především z hlediska vyhledání relevantní služby. Řešením by mohli být anotace služeb pro vyjádření podstaty a použitelnosti prostřednictvím ontologie. Webové služby by se tak staly velmi vhodnou technologií pro realizaci aplikací odpovídající koncepci sémantického webu. Jelikož sémantický web je zatím stále spíše myšlenkou, větší pozornost se věnuje zajištění syntaktické interoperability, která může přinést praktické výsledky dříve. Existují však také aktivity zabývající se integrací využitelné sémantiky do webových služeb. Jde především o vzájemně si konkurující projekty OWL-S, WSDL-S a WSMO (Web Services Modeling Ontology).

3.4.1 Ontology Web Language for Services (OWL-S)

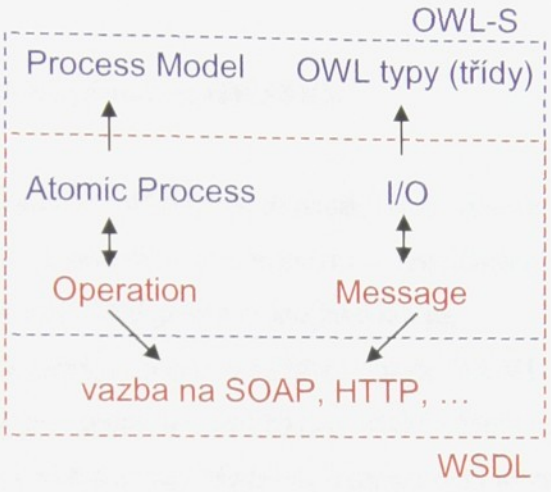
OWL-S (OWL for Services) rozšiřuje slovník jazyka OWL o sadu značek pro vytváření popisu webových služeb. Jazyk vyvíjený skupinou akademiků vychází z DAML-S a momentálně je k dispozici ve verzi 1.1. Rozšíření OWL-S jsou vázány na WSDL a jejich cílem je specifikace strojově srozumitelného sémantického popisu (ontologie) pro automatické vyhledání služby, stanovení významu parametrů a návratových hodnot a zaznamenání informací o procesním skládání v případě komplexních služeb, které se skládají z několika kroků.



Obrázek č. 9 – OWL-S - Nejvyšší vrstva ontologie služby

Každá publikovaná služba odpovídá v OWL-S instanci třídy `Service`, která má definovány tři vlastnosti `presents`, `supports` a `describedBy`. Oborem hodnot (`range`) těchto vlastností jsou instance tříd `ServiceProfile`, `ServiceModel` a `ServiceGrounding` (viz Obrázek č. 9). Třída `ServiceProfile` poskytuje popis podstaty služby, který využije především vyhledávací agent k výběru relevantní služby. Prostřednictvím třídy `Pofile` (viz Příloha 3) jsou publikovány tři druhy informací: údaje o poskytovateli služby, funkcionalita služby a vlastnosti hostitele služby (kategorizace, hodnotící systém a různorodé parametry jako maximální čas odezvy, geografické umístění apod.). Třída `ServiceModel` obsahuje podtřídu `Process` (viz Příloha 3), jež popisuje chování služby při jejím vyvolání, zejména tedy vstupy a výstupy jednotlivých úkonů `AtomicProcess` (analogie WSDL `operation`). Službu zde tedy chápeme jako proces, kdy v případě služby skládající se z několika kroků může vyhledávací agent tyto informace

využít k hlubší analýze funkcionality služby. Poslední třída nejvyšší úrovně ontologie služby `ServiceGrounding` specifikuje, jakým způsobem ke službě přistupovat, neboli popisuje její rozhraní. Typicky jde o protokol, formát zpráv, serializaci, porty apod. Jedná se prakticky o analogii WSDL elementu `binding`, avšak zmíněné vlastnosti se mapují na OWL-S model procesů. Mapování mezi OWL-S a WSDL 1.1 naznačuje obrázek č.10.



Obrázek č. 10 – Mapování OWL-S na WSDL

3.4.2 WSDL-S

WSDL-S rozšiřuje WSDL o sémantické anotace elementů jako `interface`, `operation`, `input` a `output` a jejich typová schémata. Způsob, jakým WSDL-S přiřazuje význam těmto abstraktním elementům je podobný OWL-S. WSDL-S však nabízí odlehčený a přehlednější přístup, který lze následně využít i např. pro tvorbu ontologie v OWL-S. Výhody WSDL-S spočívají v tom, že zatímco OWL-S ontologie je publikována v samostatném souboru a pracuje se svým vlastním modelem, sémantický popis ve WSDL-S je přímo součástí WSDL dokumentu, s jehož strukturou by měli být vývojáři již dobře obeznámeni. Další výhodou je skutečnost, že vyjádření sémantiky nepředjímá žádnou konkrétní formalizaci, je tedy možné použít libovolný ontologický jazyk (UML, OWL).

WSDL-S definuje pět elementů a atributů, jejichž stručný popis nyní uvedeme. Atribut `modelReference` definuje asociaci mezi WSDL entitou a nějakým sémantickým modelem (ontologií). Může být aplikován na elementy `xsd:complexType`, `xsd:element`, `wsdl:operation` a také na nové elementy `precondition` a `effect`. Atribut

`schemaMapping` se připojuje k elementům, které mají definovaný typ (XML Schema, komplexní typ) a vyjadřují korespondenci s typy sémantického modelu. Dva nové elementy `precondition` a `effect` jako potomci elementu `wsdl:operation` a rozšiřující atribut `category` elementu `wsdl:interface` se používají pro určení funkcionality a kategorizace služby a uplatňují se především pro vyhledání služby.

3.4.3 Web Services Modeling Ontology (WSMO)

WSMO je další alternativou určenou k sémantickému popisu služeb za účelem jejich automatického vyhledání, spolupráce a kompozice. Specifikace je vyvíjena pod WSMO Working Group, W3C jej roku 2005 publikovalo jako návrh.

Formální vyjádření sémantiky služeb umožňuje jazyk WSML (Web Service Modeling Language), návrh ontologie používá specifikace MOF (Meta Object Facility). Koncept WSMO nazvaný WSMF (Web Service Modeling Framework) je založen na čtyřech hlavních entitách: `Ontology` poskytuje doménově závislou terminologii ostatním elementům, `Web services` popisují přístup ke službě nějaké domény z hlediska funkčnosti, rozhraní a technologie, `Goals` reprezentují možné cíle klienta a `Mediators` řeší problémy s interoperabilitu na úrovni spolupráce ontologií, protokolů a procesního skládání (kombinace služeb).

Oproti OWL-S se přístup WSMO liší v několika aspektech. V první řadě OWL-S neodděluje popis služby a kritéria hledané klientem. Pro popis služby nefunkcionálními daty (jméno služby, kontakt) nejsou v OWL-S spojeny s žádným standardem, WSMO naproti tomu doporučuje užití např. Dublin Core. Další odlišností je, že v OWL-S můžeme specifikovat k určité službě vždy jen jedno rozhraní, kdežto WSMO počítá s několika možnými způsoby interakce se službou, což se týká především procesního skládání služeb. WSMO také umožňuje přímo implementovat zajištění interoperability např. mezi službou popsanou WSMO a službou OWL-S (prostřednictvím `Mediators`). Obecně lze shrnout, že WSMO poskytuje komplexnější model, kdežto OWL-S se považuje za vyspělejší technologii především co se týče vazby na službu a popisu její činnosti.

Kapitola 4

Realizace webové služby pro dotazování do repository

Cílem praktické části práce je vytvoření aplikace na bázi webové služby, která dokáže procházet repository obsahující metadata, vyhledávat relevantní informace a vracet požadované výsledky ve formátu RDF/XML. Systém lze označit jako tzv. loosely-coupled, tedy systém, kde o sobě klient ani server předem nemusí prakticky nic předpokládat. Pomocí naimplementovaného rozhraní bude klient schopen systém konfigurovat a ovlivnit tak způsob dotazování a formát výsledků. Aplikace webové služby toto rozhraní jasně specifikuje pomocí WSDL a umožňuje tak velmi dynamickou interakci. Záleží pak pouze na konkrétní realizaci služby, jaké možné přístupy implementuje. Jde tedy v podstatě o řešení obecného způsobu dotazování, které nepředjímá žádnou konkrétní reprezentaci dotazu, způsobu uložení dat a formátu výsledku.

V této práci bude demonstrována webová služba implementována pomocí Java 2 Enterprise Edition (J2EE) s aplikačním serverem Apache Tomcat a implementací SOAP protokolu na bázi Javy Apache Axis. Služba bude schopna prostřednictvím specifikace rozhraní Simple Query Interface (SQI) dotazování do repository obsahující RDF metadata standardu LOM, která budou fyzicky uložena v databázi MySQL. Podporované formáty dotazu budou RDF-QEL nebo DATALOG-QEL. Výsledky dotazů budou formátovány pomocí tamplate enginu Jakarta Velocity do RDF/XML.

Jedním z možných scénářů nasazení aplikace je využití jako nástroje pro získávání metadat o objektech např. v rámci P2P sítě, na jejichž základě se bude moci klient rozhodnout, zda k samotnému objektu přistupovat, či nikoliv. Kompletní zdrojové kódy aplikace jsou k dispozici na přiloženém CD-ROM (viz Příloha 5).

4.1 Automatizované nástroje Apache Axis

Apache Axis mimo implementace SOAP protokolu zahrnuje také nástroje Java2WSDL WSDL2Java pro automatické generování zdrojových kódů. Java2WSDL je nástroj pro vygenerování popisu webové služby z Java rozhraní. Nejdříve tedy vytvoříme a zkompilujeme rozhraní v Javě, nástrojem Java2WSDL vygenerujeme WSDL popis služby, ze kterého nástrojem WSDL2Java vygenerujeme jak serverovou, tak klientskou část webové služby. Tyto části budou obsahovat kompletní infrastrukturu potřebnou k provozu služby, je třeba pouze implementovat metody uvedené v Java rozhraní.

4.2 Rozhraní služby – SQI, WSDL

Služba je založená na vzdáleném volání metod JAX-RPC prostřednictvím protokolu SOAP. Aby bylo možné službu volat, je nutné specifikovat určitou sadu metod pro komunikaci. Vytvoření rozhraní se tak stává výchozím bodem pro vývoj služby. Aplication Program Interface (API) pro implementaci metod za účelem dotazování definuje např. specifikace SQI (Simple Query Interface). Jedná se o neutrální specifikaci z pohledu dotazovacího jazyka i formátu výsledků, podporuje synchronní i asynchronní komunikaci a je založen na práci se session. Od SQI se samozřejmě odvíjí i specifikace rozhraní služby pomocí WSDL 1.1.

4.2.1 Rozhraní pro Session Management

Komunikace mezi klientem (v SQI označen source) a službou (target) vždy začíná vytvořením session, proto také popis rozhraní zahájíme sadou metod pro Session Management. SQI obecně umožňuje vytváření jak anonymních sessions, tak sessions se specifikovaným jménem a heslem.

Prvním krokem, který klient musí uničit za účelem navázání komunikace, je zavolání metody pro vytvoření session. SQI specifikuje metodu pro neanonymní session `createSession`, jejíž parametry jsou identifikátor uživatele `userID` a heslo `password`, oba typu `String`. Anonymní session vytvoříme zavoláním metody bez parametrů `createAnonymousSession`. Jako návratová hodnota obou metod je vráceno

sessionID opět typu String. Třetí povinnou metodou rozhraní je metoda destroySession, jejíž parametr sessionID identifikuje session, která má být zrušena. Kromě SQI metod specifikujeme také sada výjimek ošetřujících chybové stavy, které mohou být zaslány klientovi pro upřesnění chybového stavu (SessionCreationFailureException a pro autentizaci také WrongCredentialsException).

Uvedené rozhraní uložíme do souboru SessionManagement.java jako součást balíku service.session_mgmt. Nyní použijeme nástroj Java2WSDL pro vygenerování WSDL popisu následujícím příkazem:

```
java org.apache.axis.wsdl.Java2WSDL -o SessionMNG.wsdl
        -l"http://localhost:8080/services/SessionMNG" -p"service.session_mgmt"
        "urn:SessionMNG" service.session_mgmt.SessionManagement
```

, kde přepínač -o specifikuje jméno WSDL, který se vygeneruje; -l umístění služby; -p mapování mezi balíkem a jmenným prostorem; a uvedená třída obsahuje rozhraní služby.

Vygenerovaný WSDL soubor bude obsahovat příslušné metody, které budou mapovány jako operace tagem wsdl:operation, jejich vstupy a výstupy jako zprávy elementem wsdl:message (vždy s postfixem Request a Response pro každou metodu). V rámci zpráv budou specifikovány přenášené parametry a jejich typ (mapování typů z Javy do XML Schema specifikuje Axis). Pro každou metodu (operation) jsou specifikovány možné výjimky elementem wsdl:fault, jež dříve definujeme jako zprávy. Na závěr je uvedena vazba na konkrétní umístění, která slouží klientům pro vyvolání služby.

4.2.2 Rozhraní pro dotazování

Jakmile je navázáno spojení pomocí session, systém očekává dotaz. V prvé řadě rozhraní definuje sadu metod pro parametrizaci služby. Metoda setQueryLanguage určuje formát dotazu, setResultsSize počet výsledků v rámci jedné zprávy, setMaxQueryResults definuje maximální počet výsledků dotazu (tudíž by měl být větší než parametr předchozí metody), maximální dobu odezvy lze nastavit metodou setMaxDuration a formát výsledku prostřednictvím metody setResultsFormat.

Metoda	Syn.	Asyn.	Parametry	Návratová hodnota
setQueryLanguage	X	X	String targetSessionID String queryLanguageID	void
setResultsFormat	X	X	String targetSessionID String resultsFormat	void
setMaxQueryResult	X	X	String targetSessionID int maxQueryResults	void
setMaxDuration	X	X	String targetSessionID int maxDuration	void
setResultsSetSize	X		String targetSessionID Int resultsSetSize	void
synchronousQuery	X		String targetSessionID String queryStatement int startResult	String
getTotalResultsCount	X		String targetSessionID String queryStatement	int
setSourceLocation		X	String targetSessionID String sourceLocation	void
asynchronousQuery		X	String targetSessionID String queryStatement String queryID	String
queryResultsListener *(implementace na klientovi)*		X	String queryID String queryResults	void

Tabulka č. 3 – SQI rozhraní pro dotazování do repository

Následuje volání metod sloužících ke zpracování dotazu. Rozhraní pro synchronní komunikaci definuje metodu `synchronousQuery`, pro asynchronní přenos metodu `asynchronousQuery`. V synchronním módu je výsledek vrácen přímo jako návratová hodnota. Metoda `getTotalResultsCount` vrací celkový počet relevantních metadatových záznamů. V případě asynchronní komunikace se výsledky předávají metodě `queryResultsListener`, která musí být implementována na straně klienta, kde je zavolána jako služba. Parametrem metody `setSourceLocation` je nutné nastavit umístění klienta právě za účelem volání předešlé metody. Přehled všech metod, parametrů a návratových hodnot viz Tabulka č.3. Dále opět specifikuje množinu výjimek. Těch je tentokrát definováno celkem patnáct, přičemž od rodičovské výjimky `QueryServiceException` odvozujeme specializovanější, které se týkají buď chybné parametrizace (např. `NoValidResultsSetSizeException`,

`NoValidMaxDurationException` apod.) nebo špatného formátu dotazu a výsledku (např. `NoValidQueryStatementException`).

WSDL soubor vygenerujeme nástrojem Java2WSDL podobným způsobem, jako v případě Session Managementu.

4.3 Generování stubs, skeletons a datových typů

Máme-li k dispozici WSDL popisy služby, použijeme nástroj WSDL2Java pro vygenerování Java kódu jak pro klientskou, tak serverovou část. Tímto nástrojem jsme schopni vygenerovat vše potřebné k popisu služby: *stubs* (zástupný kód pro volání služby), *skeletons* (třídy, které stojí mezi Axis enginem a konkrétní implementací) a potřebné datové typy.

Vytvoření kostry služby demonstrujme příkazem pro vygenerování serverové části z rozhraní pro dotazování:

```
java org.apache.axis.wsdl.WSDL2Java -o gen -d "Application" -s -S true
 -N urn:TargetQueryMNG service.sqi.target TargetQueryMNG.wsdl
```

, kde přepínač -o určuje umístění pro vygenerované soubory; -d specifikuje způsob vytváření objektů při volání služby (tzv. *scope*); -s specifikuje generování serverové části; -S pak nastavuje vytváření *skeletons*; -N specifikuje mapování na namespace.

Příkazem jsou mimo jiné vytvořeny následující soubory:

- `Query.java` obsahuje nové rozhraní, které obsahuje použití `java.rmi.Remote`;
- `QueryService.java` je třída obsahující rozhraní klientské části služby;
- `TargetQueryMNGSoapBindingImpl.java` je výchozí implementací serverové části služby (je třeba doplnit implementace metod);
- `TargetQueryMNGSoapBindingSkeleton.java` je kostra serverové části;
- `TargetQueryMNGSoapBindingStub.java` je zástupný kód klientské části;
- `deploy.wsdd` a `undeploy.wsdd` jsou deskriptory pro nasazení služby na server.

4.4 Implementace služby

4.4.1 Session Management – balík `service.session_mgmt`

Implementace Session Managementu je realizována prostřednictvím tříd `SessionMNGSoapBindingImpl.java`, jejíž kostru jsme vygenerovali v předchozí kapitole, a třídy `Session.java`, jako datový typ. Všechny relevantní třídy jsou součástí balíku `service.session_mgmt`, odpovídající výjimky pak `service.session_except`. Každý objekt `Session` má definovány proměnné třídy `sessionID` (jednoznačné ID pro každou session), `pID` (identifikátor uživatele, který je implicitně nastaven na -1 pro anonymní session), `uID` (jméno uživatele implicitně „anon") a dále timestamps indikující vytvoření a expiraci session. Vytvoření session provádí konstruktor, ve které je vygenerováno unikátní `sessionID` pomocí časové a hešovací funkce MD5.

Metody popsané ve specifikaci rozhraní Session Managementu (kapitola 4.2.1) jsou implementovány ve třídě `SessionMNGSoapBindingImpl.java`. V práci jsem vytvořil pouze metody pro práci s anonymními sessions, avšak s přihlédnutím na snadnou implementaci i těch neanonymních (metoda `createSession`), což se týká především práce s osobním identifikátorem `personID`, které je u anonymních sessions implicitně nastaven na `-1`. Nepracujeme tedy přímo s touto hodnotou, ale s proměnou, což umožní snadnou implementaci i neanonymních sessions při zachování stávajících metod. Důležitou proměnnou třídy je kolekce všech session `sessMap`. Vytvoření anonymní session provádí metoda `createAnonymousSession`, v níž nejprve prověříme, zda jsme nepřesáhli limit vytvořených anonymních session, následně vytvoříme nový objekt `Session` s implicitními anonymními hodnotami a vrátíme klientovi `sessionID`. Zrušení session poskytuje metoda `destroySession`, jež na základě předané `sessionID` danou session, pokud existuje, odstraní z kontejneru.

Součástí třídy `SessionMNGSoapBindingImpl` je také vnitřní třída `SessionCleaner`. Tato třída je inicializována v konstruktoru a periodicky spouštěna jako vlákno, které vyhledává a maže expirované sessions. Za účelem aktualizace session seznamů, které jsou implementovány dva – jeden v rámci služby Session Managementu a jeden jako součást služby pro dotazování, jsou ve třídě `Session.java` implementovány metody

destroyRemoteSession a setDestroySessionNotifier. Metody slouží jednak k inicializaci parametrů umístění služby (typicky umístění TargetQueryMNG, ačkoli metody může volat i klient) a jména vzdálené metody, tak pro vlastní zavolání vzdálené metody (pro TargetQueryMNG je to metoda destroySessionNotify) s parametrem SessionID indikující rušenou session.

4.4.2 Rozhraní služby – balík service.sqi.target

Metody dotazovacího rozhraní, tak jak jsme ho specifikovali v kapitole 4.2.2, implementuje třída TargetQueryMNGSoapBindingImpl.java. V konstruktoru třída inicializuje implicitní hodnoty parametrů, vytvoří potřebné instance pro volání metod SessionManagementu a rovnou volá metodu setDestroySessionNotifier pro nastavení synchronizace sessionMap. Téměř každá metoda třídy pracuje s instancí třídy QuerySession.java, která reprezentuje datový typ pro jednotlivé dotazy. Proměnné třídy QuerrySession definují dotaz, výsledek, parametry jako počet výsledků, indexy výsledku, formát dotazu apod. Dále jsou zde mimo jiné implementovány dvě důležité metody getParsedQELQuery a getFormattedResults pro formátování dotazu a výsledku (viz dále).

Metody rozhraní služby (kapitola 4.2.2) implementované ve třídě TargetQueryMNGSoapBindingImpl vždy nejprve volají privátní metodu verifySession, ta ověřuje platnost session, jak v lokální struktuře, tak v záznamech session managementu, a v kladném případě vrací instanci QuerySession. Následně se zjišťuje, zda-li vzdálená parametrizace systému náleží do povoleného oboru hodnot, který stanovují implicitní parametry služby. V případě korektních hodnot se tyto nastaví v instanci dotazu. Nejvýznamnější metoda této třídy je bezpochyby synchronousQuery, která kromě identifikátoru session přebírá od klienta vlastní dotaz. Návratová hodnota předaná zpět klientovi představuje výsledek dotazu. Po ověření session a případném vytvoření objektu QuerySession je zavolána metoda této třídy getParsedQELQuery, která prostřednictvím knihoven Edutelly (viz kapitola 2.7.3) a na základě formátu dotazu (Datalog, XML serializace) parsuje dotaz (String). Metoda vrací objekt typu Query, ten je následně předán v konstruktoru třídy QueryToSQL z balíku service.utils. Vytvořením instance této třídy dostáváme k dispozici metody, jež opět s využitím knihoven Edutelly

konvertují dotaz z QELu do SQL. Pokud máme dotaz ve formátu SQL, nic již nebrání tomu, abychom položili dotaz do vlastního repository. K tomu slouží balík `service.repository`, v němž je definováno rozhraní `QueryInterface`, jeho implementace `QueryInterfaceFactory` se statickou vnitřní třídu `MySQLQueryInterface`. Ta implementuje vlastní dotazování do MySQL databáze veřejnou metodou `getResults`. Návratovou hodnotou je kontejner typu `ArrayList` obsahující jednotlivé výsledky pro každý podporovaný predikát. V další fázi metody `synchronousQuery` se v instanci dotazu `QuerySession` nastaví výsledek, jeho velikost a aktuální index výsledku. Nyní potřebujeme výsledek zformátovat do zvoleného formátu. Učiníme tak voláním metody `getFormattedResults`, jež pomocí instance třídy `ResultFormatter` vrací výsledky ve zvoleném formátu (v našem případě RDF/XML), a ty jsou také návratovou hodnotou metody `synchronousQuery`.

Oproti definici SQI rozhraní třída `TargetQueryMNGSoapBindingImpl` navíc implementuje metodu `getAdditionalQueryResults`. Metodu může klient využít v případě, že výsledek dotazu obsahuje více záznamů, než povoluje parametr stanovující maximální počet výsledků v rámci jedné zprávy. Celkový počet výsledků dotazu je klientovi distribuován prostřednictvím metody `getTotalResultsCount`.

Stejně jako implementační třída Session Managementu i třída `TargetQueryMNGSoapBindingImpl` definuje vnitřní třídu `SessionCleaner`, která udržuje v kontejneru `sessionMap` pouze platné sessions. Navíc třída obsahuje metodu `destroySessionNotify`, jež volá služba session managementu v případě, jestliže je některá ze session zneplatněna.

4.4.3 Pomocné třídy – balík `service.utils`

Součástí balíku `service.utils` jsou tři třídy – `QueryColumnMapper`, `QueryToSQL` a `QueryResultSetParser`. Ve třídě `QueryColumnMapper` definujeme dvě pole `ArrayList intCols a extCols`, čímž stanovíme mapování mezi externím formátem vlastností-predikátů (např. `lom-tech:size`) a interní reprezentací pro názvy sloupců MySQL (např. `TEC_SIZE`), přičemž mapování realizujeme prostřednictvím společného indexu odpovídajících názvů.

Jak již bylo řečeno dříve, třída QueryToSQL konvertuje dotaz typu `Query` (Edutella), jež získáme parsováním QEL dotazu, do SQL. Využíváme přitom řadu knihoven projektu Edutella, v rámci něhož je také jazyk QEL specifikován. Jedná se především o metody a datové typy, které umožňují pracovat s literály QELu. V metodě `importLiterals` tak procházíme jednotlivé literály, a pokud např. narazíme na instanci `StatementLiteral`, pokusíme se nejprve převést jmenný prostor predikátu na odpovídající formát `ColumnMapperu` (např. z `http://ltsc.ieee.org/2002/09/lom-life#` na `lom-edu:`), následně v případě kladného testu objektu na literál (nikoliv proměnnou) je tento nahrazen odpovídajícím interním názvem. Další možností je instance typu `BuiltinLiteral`, načež odpovídající mapování vestavěných vlastností QELu do SQL zprostředkuje metoda `mapOperator` (např. pomocí `equals(QEL.lessThan)` testujeme predikát, který v kladném případě nahradíme řetězcem "<").

Třída `QueryResultSetParser` je určena, jak sám název napovídá, k formátování výsledku do RDF/XML. Za tímto účelem využijeme template engine Velocity. Nejprve je třeba vytvořit šablonu dokumentu obsahující kostru a všechny podporované predikáty, kde na místo hodnoty vepíšeme název proměnné (klíč), jejíž hodnota bude později substituována (viz soubory *.vm Příloha X). Pro nepovinné součásti dokumentu navíc implementujeme makro, které v případě, že klíč nebude mít definovanou hodnotu, daný predikát vynechá. Veřejné metodě `getFormatedResult` třídy `QueryResultSetParser` jsou předány dva kontejnery obsahující jednak pole názvů proměnných pro Velocity (`keys`), tak jejich hodnot jako sadu výsledků dotazu (`values`). V metodě procházíme jednotlivé dvojice klíč-proměnná, kterými nejprve inicializujeme strukturu `VelocityContext`. Ten je následně aplikován na šablonu metodou `merge`. Návratová hodnota této metody pak putuje až ke klientovi coby zformátovaný výsledek dotazu.

4.4.4 Repository a jeho rozhraní – balík `service.repository`

Jako úložiště dat jsem zvolil databázi MySQL. Metadata jsou uchovávány v tabulce, kde názvy sloupců odpovídají jednotlivým podporovaným vlastnostem. Řádky tabulky reprezentují jednotlivé dokumenty, přičemž jako primární klíč slouží URI dokumentu. Na úrovni databáze jsou také řešeny povinné prvky výsledku, kdy každý platný záznam musí mít

tyto prvky definovány (konkrétně jsou to vlastnosti `URI`, `identifier`, `title`, `language` a `description`).

Dotazování do databáze implementuje třída `MySQLQueryInterface`, jakožto vnitřní třída `QueryInterfaceFactory`. Počítá se tedy se snadnou implementací i dalších možností uložení dat. Třída `MySQLQueryInterface` definuje metodu `getResults`, ve které se provádí vlastní připojení do databáze. Metodě jsou předány omezující podmínky v SQL. Na úrovni SQL je omezen také počet výsledků (`LIMIT`) odpovídající konfiguraci služby. Návratová hodnota metody typu `List` představuje sadu výsledků dotazu.

4.4.5 Parametry služby – třída `service.ServiceConf`

Aplikace nabízí poměrně značné možnosti konfigurace. Třída `ServiceConf` zahrnuje jednak parametry konfigurovatelné přes rozhraní (viz kapitola 4.2.2) a jejich implicitní hodnoty, tak vnitřní konfigurace pro Session Management, Velocity a MySQL.

4.5 Implementace klienta

Implementace klienta je velmi jednoduchá, jelikož prakticky všechen kód lze vygenerovat z WSDL nástrojem WSDL2Java. Demonstraci klientského volání služby obsahuje balík `client`, prostřednictvím třídy `ClientSQI` a metody `ClientSynQueryTest`. Zde nevoláme přímo s instancí vygenerovaných *stubs*, nýbrž vytvoříme instanci lokátoru (např. `QueryServiceLocator`). V dalším kroku na tuto instanci zavoláme metodu get (např. `getTargetQueryMNG`), jež vrací *stub* implementující rozhraní služby. V tomto okamžiku již můžeme volat všechny metody odpovídající rozhraní služby.

4.6 Nasazení služby

Apache Axis podporuje několik variant nasazení služby na webový server. Jelikož nám nástroj WSDL2Java pro obě dvě služby automaticky vygeneroval dva WSDD (Web Service Deployment Descriptor) pro nasazení a odebrání služby ze serveru (`deploy.wsdd` a `undeploy.wsdd`). Deployment deskriptor obsahuje vše, co chce uživatel zpřístupnit skrze Axis engine. Zápis opět vychází z XML a je velice dobře čitelný. Vnější element `deployment` říká enginu, že jde o WSDD deskriptor a definuje jmenný prostor java. Element `service` definuje vlastní službu, její styl a základní ovladač (provider), vnořené elementy `parameter` nesou informaci o tom, jaké třídy instancovat, případně jaké metody volat.

Kapitola 5

Závěr

Sémantický web je stále spíše vizí, proto drtivá většina technologií a standardů zmiňovaných v první kapitole je stále předmětem výzkumu a procházejí poměrně rychlým vývojem. Klíčová otázka realizace této vize se odvíjí především od tvorby ontologií. Zde je třeba učinit ještě mnoho práce a to nejen z hlediska standardizace určité sady prvků postihující danou doménu, ale především oboru hodnot prvků tak, aby byly všeobecně akceptovatelné. Teprve potom bude reálná myšlenka strojového odvozování nad sémantickými informacemi, a bude jistě možné alespoň nastínit možnosti implementací vyšších vrstev sémantického webu, které nemohly být do této doby implementovány.

Naproti tomu webové služby jsou technologií již v dnešní době značně využívanou. Tuto popularitu získávají především tam, kde je třeba zajistit interoperabilitu mezi aplikacemi různých platforem. Webové služby lze tedy označit za velmi vhodnou technologii pro strojovou interakci. Avšak pro využití celého potenciálu této technologie zbývá dořešit především otázku veřejné publikace služeb, tedy UDDI registrů či jeho alternativ. Co se týče sémantické integrace webových služeb, neboli možnosti automatického vyhledání služby na základě jejího sémantického popisu, je situace obdobná jako v případě sémantického webu. Omezujícím faktorem je tedy standardizace metadatových schémat, ale i jejich formálního vyjádření.

Pokud se tyto problémy podaří uspokojivě vyřešit, budeme tyto technologie potkávat jistě mnohem častěji než doposud. Některé aplikace však bude třeba provádět obzvláště citlivě s ohledem na svobodu a soukromí lidí.

Praktická část práce si kladla za cíl vytvoření tzv. losely-coupled systému pro dotazování do repository s metadaty. To se v první řadě podařilo prostřednictvím rozhraní, které umožňuje pomocí sady metod parametrizovat službu tak, jak požaduje klient. Také implementace na bázi webové služby se ukázala jako velmi vhodná, jelikož klient tak může být implementován na prakticky libovolné platformě a na základě strojově čitelného rozhraní automaticky generovat volání služby. Celkově byl tedy vytvořen systém demonstrující obecné dotazovací rozhraní do obecného úložiště dat, jehož možnosti byly implementovány tak, aby splňovaly zadání práce.

Zdroje:

[1] SVÁTEK, V. *Ontologie a WWW.* [online]. [cit. 2007-01-22]. URL: <http://nb.vse.cz/~svatek/onto-www.pdf>

[2] SKLENÁK, V. *Metadata, sémantika a sémantický web.* [online]. [cit. 2007-01-22]. URL: <http://www.inforum.cz/inforum2004/pdf/Sklenak_Vilem1.pdf>

[3] HERMAN, I. *Semantic Web.* [online]. [cit. 2007-01-22]. URL: <http://www.w3.org/2001/sw/>

[4] MANOLA, F., MILLER, E. *RDF Primer.* [online]. [cit. 2007-03-11]. URL: <http://www.w3.org/TR/rdf-primer/>

[5] BECKETT, D. *RDF/XML Syntax Specification.* [online]. [cit. 2007-03-11]. URL: <http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/>

[6] BRICKLEY, D., GUHA R.V. *RDF Vocabulary Description Language 1.0: RDF Schema.* [online]. [cit. 2007-03-11]. URL: <http://www.w3.org/TR/rdf-schema/>

[7] MCGUINNESS, D. L., VAN HARMELEN, F. *OWL Web Ontology Language Overview.* [online]. [cit. 2007-03-19]. URL: <http://www.w3.org/TR/owl-features/>

[8] DCMI Usage Board. *DCMI Metadata Terms.* [online]. [cit. 2007-03-25]. URL: <http://dublincore.org/documents/dcmi-terms/>

[9] IEEE Learning Technology Standards Committee. *WG12: Learning Object Metadata.* [online]. [cit. 2007-03-25]. URL: <http://ltsc.ieee.org/wg12/>

[10] NEJDL, W. *Edutella: A P2P Networking Infrastructure Based On RDF.* [online]. [cit. 2007-04-02]. URL: <http://edutella.jxta.org/reports/edutella-whitepaper.pdf>

[11] NILSSON, M., SIBERSKI, W. *RDF Query Exchange Language (QEL).* [online]. [cit. 2007-04-02]. URL: <http://edutella.jxta.org/spec/qel.html>

[12] SINGH, I., BRYDON, S., MURRAY, G., RAMACHANDRAN, V., VIOLLEAU, T., STEARNS, B. *Designing Web Services with the J2EE 1.4 Platform.* [online]. [cit. 2007-04-17]. URL: <http://java.sun.com/blueprints/guidelines/designing_webservices/html/>

[13] KOSEK, J. *Inteligentní podpora navigace na WWW s využitím XML.* [online]. [cit. 2007-04-17]. URL: <http://www.kosek.cz/diplomka/html/index.html>

[14] GUDGIN, M., HADLEY, M., MENDELSOHN, N., MOREAU, J-J., NIELSEN, F. H., KARMARKAR, A., LAFON, Y. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition).* [online]. [cit. 2007-04-22]. URL: <http://www.w3.org/TR/soap12-part1/>

[15] BOOTH, D., LIU, C.K. *Web Services Description Language (WSDL) Version 2.0 Part0: Primer.* [online]. [cit. 2007-04-25]. URL: <http://www.w3.org/TR/2007/PR-wsdl20-primer-20070326/>

[16] COVER, R. *Universal Description, Discovery, and Integration (UDDI).* [online]. [cit. 2007-04-30]. URL: <http://xml.coverpages.org/uddi.html>

[17] MARTIN, D. *OWL-S: Semantic Markup for Web Services.* [online]. [cit. 2007-05-01]. URL: <http://www.daml.org/services/owl-s/1.1/overview/>

[18] AKKIRAJU, R. FARRELL, J., MILLER, J., NAGARAJAN, M., SCHMIDT, M-T., SHETH, A., VERMA, K. *Web Service Semantics - WSDL-S.* [online]. [cit. 2007-05-01]. URL: <http://www.w3.org/Submission/WSDL-S/>

[19] ECKEL, B., *Thinking In Java 3rd Edition.* [online]. [cit. 2007-01-05]. URL: <http://www.mindview.net/Books/TIJ/>

[20] HEROUT, P., *Učebnice jazyka Java.* 2.vyd. České Budějovice: Kopp, 2005. 349s. ISBN 80-7232-115-3

[21] The Apache Software Foundation. *Web Services – Axis: Documentation.* [online]. [cit. 2007-01-15]. URL: <http://ws.apache.org/axis/java/index.html>

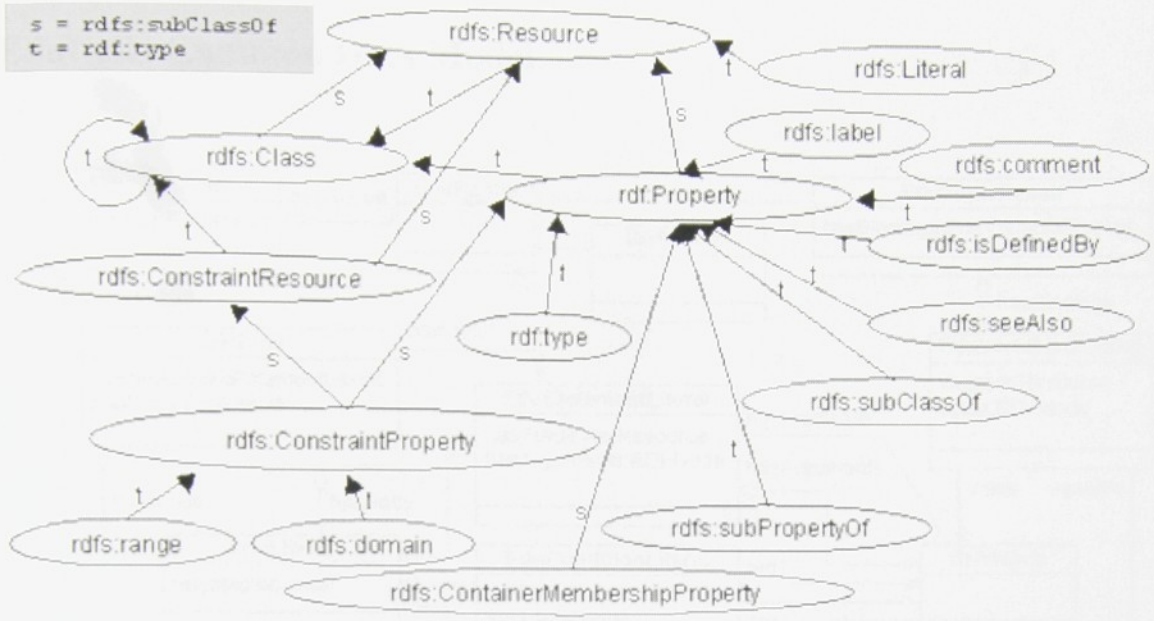
Příloha 1

Přehled předdefinovaných tříd a vlastností jazyků RDF, RDFS

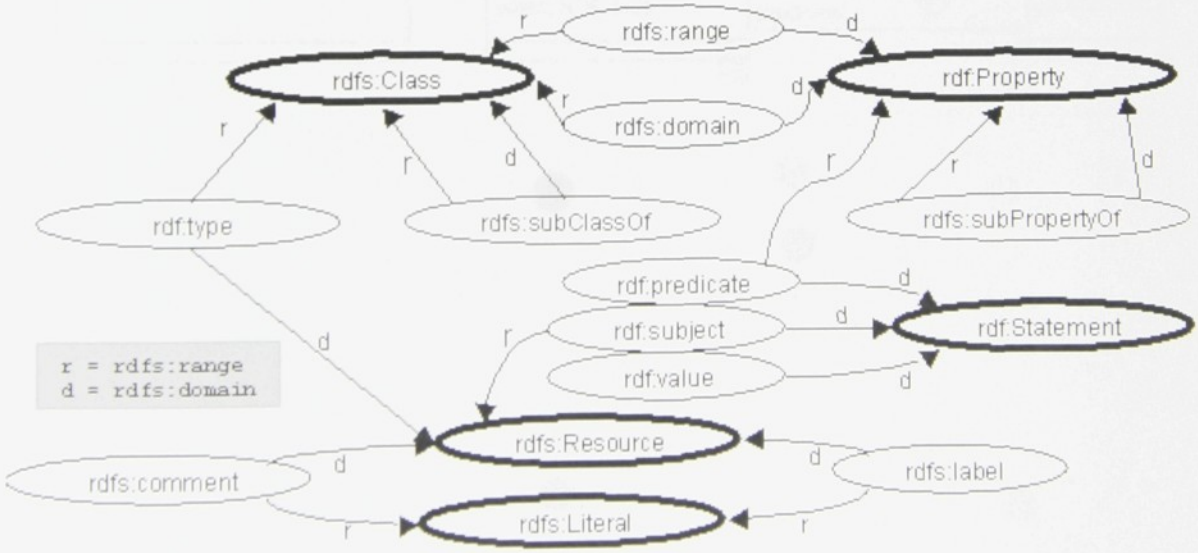
Název třídy	Komentář
rdfs:Resource	Rodičovská třída všech tříd.
rdfs:Literal	Třída všech literálů.
rdf:XMLLiteral	Třída XML literálů.
rdfs:Class	Třída všech tříd.
rdf:Property	Třída RDF vlastností.
rdfs:Datatype	Třída identifikující datový typ.
rdf:Statement	Třída RDF tvrzení.
rdf:Bag	Třída nesetříděných kontejnerů.
rdf:Seq	Třída setříděných kontejnerů.
rdf:Alt	Třída kontejnerů alternativ.
rdfs:Container	Třída kontejnerů.
rdfs:ContainerMembershipProperty	Třída příslušnosti ke kontejneru, rdf:_1, rdf:_2, ..., podvlastnost vlastnosti rdfs:member.
rdf:List	Třída kolekcí.

Název vlastnosti	Komentář	domain	range
rdf:type	Subjekt je instancí třídy.	rdfs:Resource	rdfs:Class
rdfs:subClassOf	Subjekt je podtřídou třídy.	rdfs:Class	rdfs:Class
rdfs:subPropertyOf	Subjekt je podvlastností vlastnosti.	rdf:Property	rdf:Property
rdfs:domain	Doména subjektu vlastnosti (obor hodnot)	rdf:Property	rdfs:Class
rdfs:range	Definiční obor vlastnosti.	rdf:Property	rdfs:Class
rdfs:label	Popisek	rdfs:Resource	rdfs:Literal
rdfs:comment	Komentář	rdfs:Resource	rdfs:Literal
rdfs:member	Člen zdroje subjektu	rdfs:Resource	rdfs:Resource
rdf:first	První položka kolekce	rdf:List	rdfs:Resource
rdf:rest	Zbytek kolekce (mimo první položky)	rdf:List	rdf:List
rdfs:seeAlso	Detailnější informace o subjektu	rdfs:Resource	rdfs:Resource
rdfs:isDefinedBy	Definice subjektu	rdfs:Resource	rdfs:Resource
rdf:value	Idiomatická vlastnost pro strukturované typy	rdfs:Resource	rdfs:Resource
rdf:subject	Subjekt RDF tvrzení	rdf:Statement	rdfs:Resource
rdf:predicate	Predikát RDF tvrzení	rdf:Statement	rdfs:Resource
rdf:object	Objekt RDF tvrzení	rdf:Statement	rdfs:Resource

Hierarchie tříd:



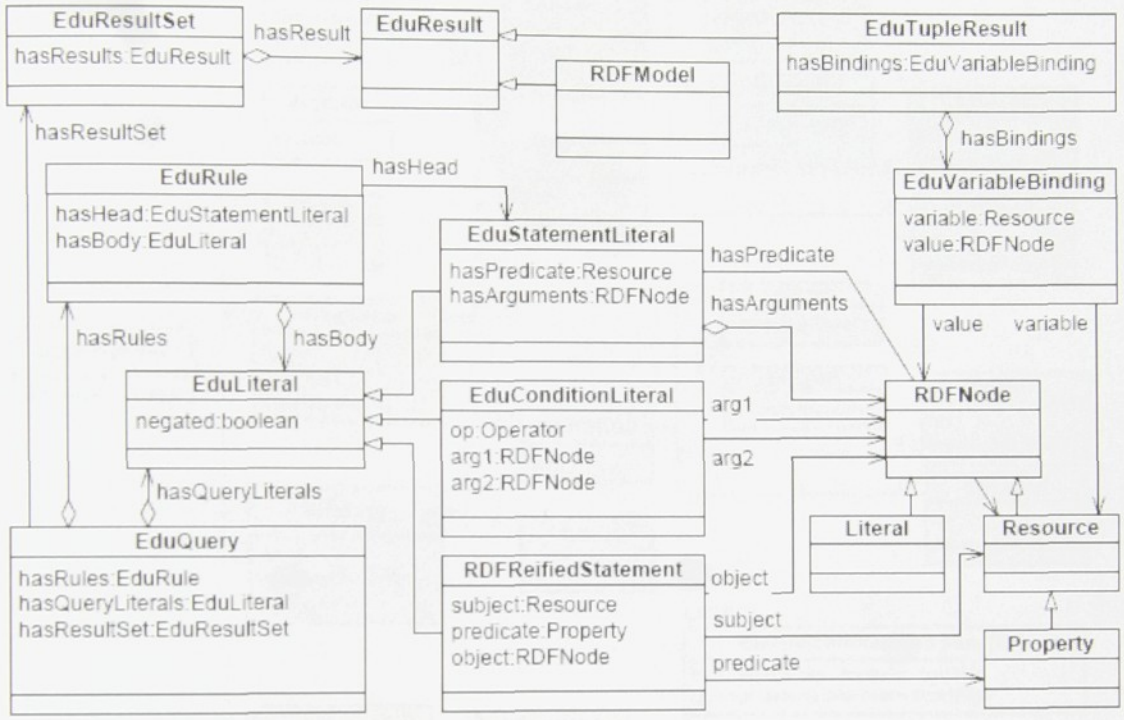
Omezení RDFS:



Příloha 2

Eduttella Common Data Model

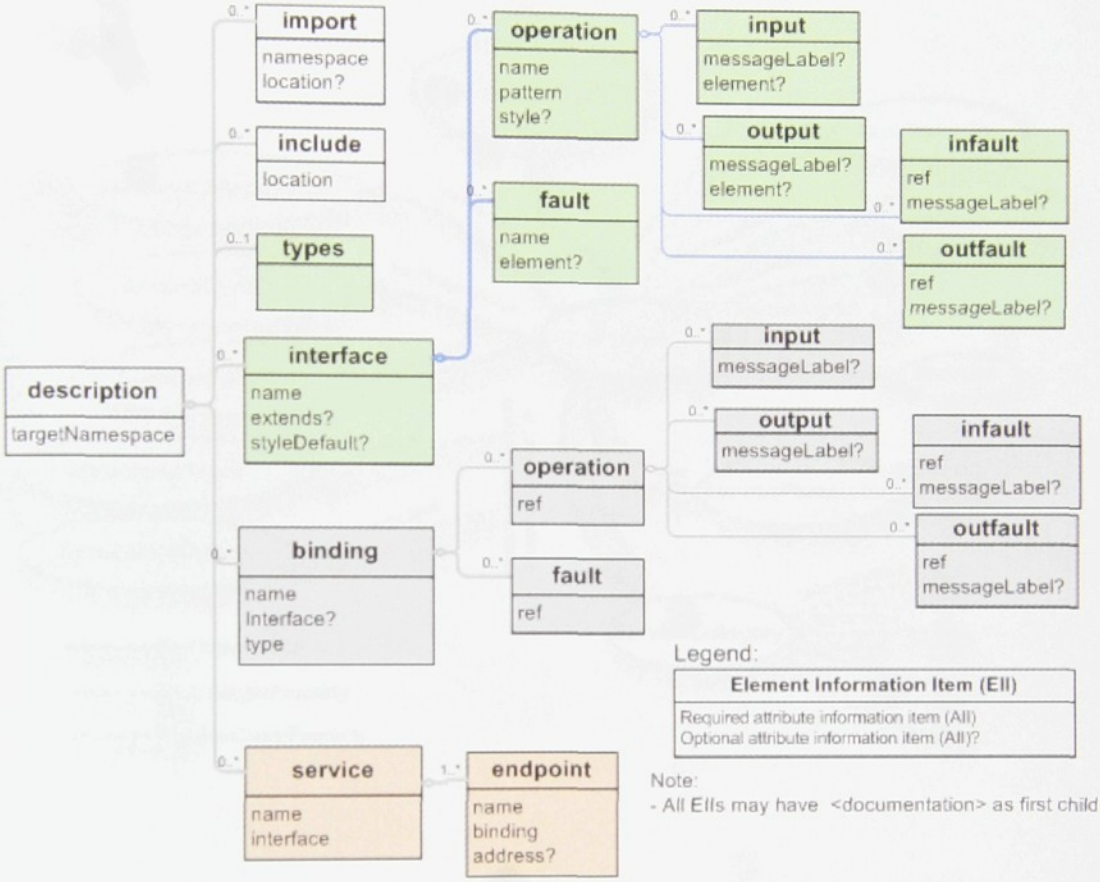
Převzato z [10].



Příloha 3

Struktura WSDL

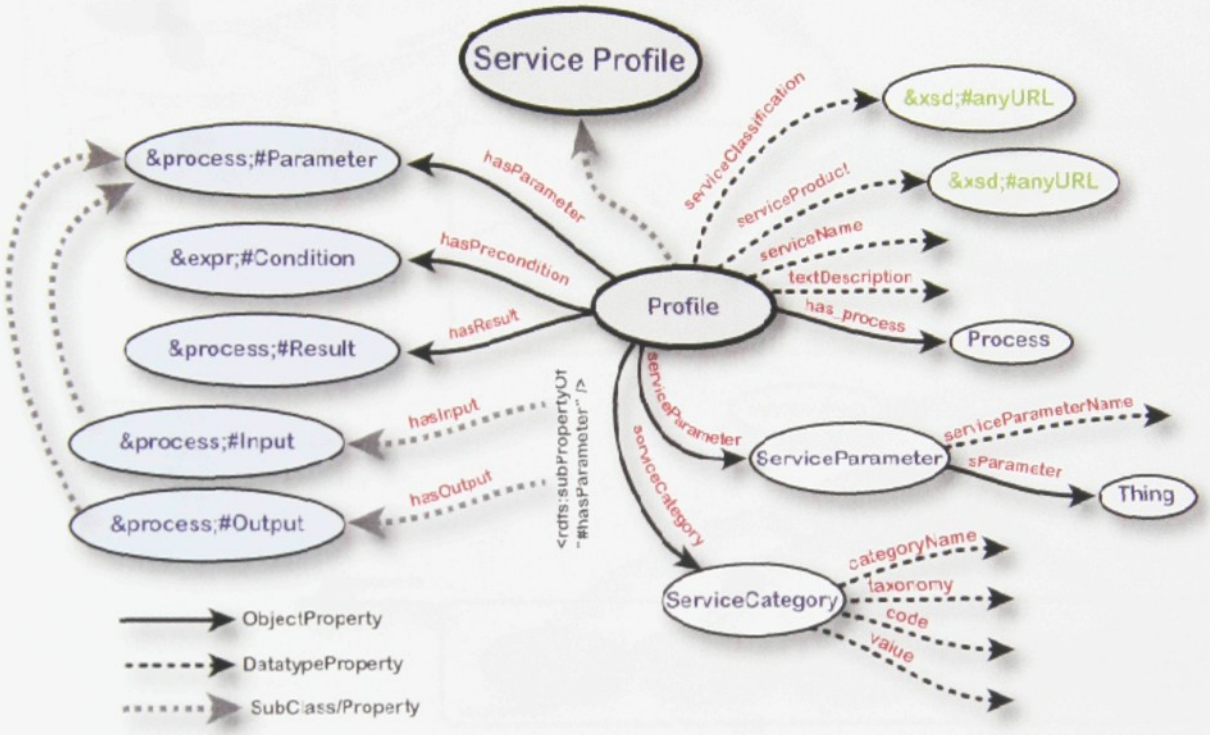
Převzato z [15].

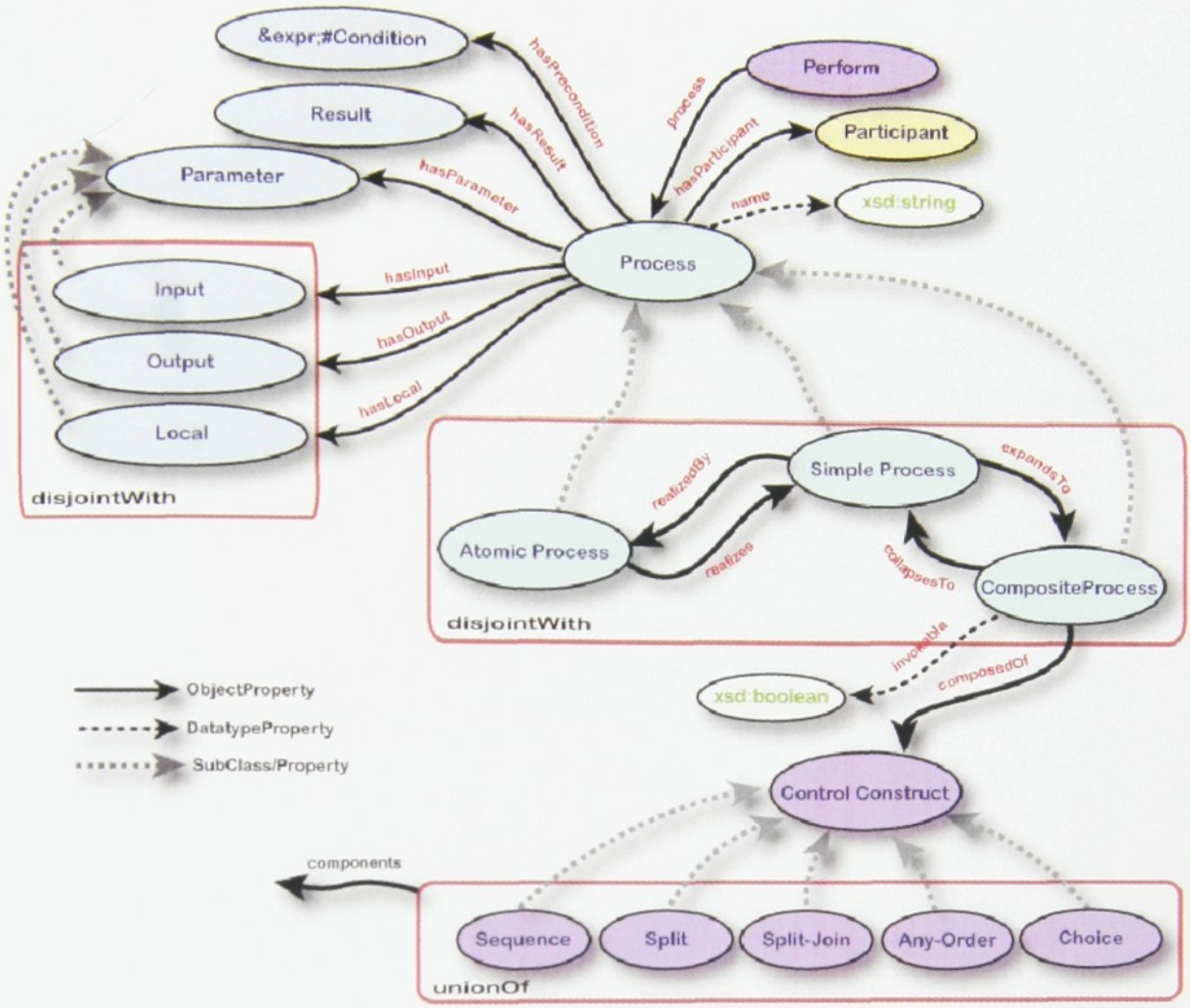


Příloha 4

OWL-S - Třídy *Profile* a *Process* a jejich vlastnosti

Převzato z [17].




&expr;#Condition
Result
Parameter
Input
Output
Local
disjointWith
hasPrecondition
hasResult
hasParameter
hasInput
hasOutput
hasLocal
Perform
Participant
xsd:string
process
hasParticipant
name
Process
Simple Process
Atomic Process
CompositeProcess
realizedBy
realizes
expandsTo
collapsesTo
disjointWith
invokable
xsd:boolean
composedOf
Control Construct
components
Sequence
Split
Split-Join
Any-Order
Choice
unionOf
ObjectProperty
DatatypeProperty
SubClass/Property

Příloha 5

CD-ROM

Na přiloženém CD je uložen text práce ve formátu DOC a PDF, zdrojové kódy služby i klienta včetně všech potřebných součástí a knihoven, WSDL soubory i WSDD deskriptory. Pro lokální testování je k dispozici také projekt pro IDE NetBeans 5.5.