VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

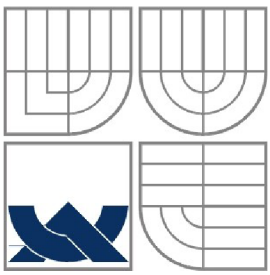
TRANSFORMACE WEBOVÝCH APLIKACÍ NA WEBOVÉ SLUŽBY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

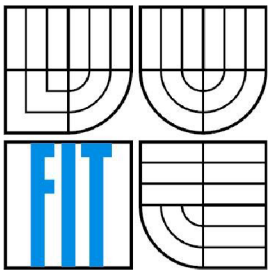
AUTOR PRÁCE
AUTHOR

Bc.Miroslav Zámečník

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

TRANSFORMACE WEBOVÝCH APLIKACÍ NA WEBOVÉ SLUŽBY

TRANSFORMATION OF WEB APPLICATIONS TO WEB SERVICES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Miroslav Zámečník

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Petr Weiss

BRNO 2007

Abstrakt

V současně době se web ubírá směrem k možnosti automatizovat chování uživatelů webových aplikací. Přidání sémantiky a vytvoření rozhraní webové služby jsou hlavní cesty ke splnění tohoto uživatelského komfortu. Tato cesta, nicméně také přináší komplikace jako je složitější publikování a implementace dokumentů na webu. Webové služby mohou propojovat velmi různorodé systémy, protože jsou založeny na jazyku XML, což je neutrální půda, na které se všechny programovací jazyky mohou střetnout beze ztráty nezávislosti na platformě. Automatická transformace webových aplikací na webové služby by mohla být podstatně efektivnější než pouze vytvářet webovou službu jako novou aplikaci. Tento krok je ovšem pro některé aplikace bez znalosti vnitřní struktury téměř nereálný. Transformaci bude ve většině případů možné provést jen poloautomaticky za přispění lidského rozhodnutí.

Klíčová slova

Sémantický web, Ontologie, Webová služba, XML, XML schema, XSD, XML namespace, URI, SOAP, WSDL, UDDI, Transformace, Webová aplikace

Abstract

Present web is aiming to the possibility of automatization of user behavior on web applications. Adding of semantics and creation of web service interface are the main approaches for accomplishment of this user comfort. Nevertheless, this direction brings some problems which can make more difficult publishing and implementation of web documents. Web services can connect heterogeneous systems, because they are based on XML markup language that is a place where all applications can meet without lost of platform independence. The automatic transformation of a web application into a web service could be considerably more effective than to create a web service from the beginning. However, this step is for some applications almost unreal without knowledge of their inner structure. In most cases, the transformation will be done semiautomatically with help of human decisions.

Keywords

Semantic web, Ontology, Web services, XML, XML schema, XSD, XML namespace, URI, SOAP, WSDL, UDDI, Transformation, Web service, Web application

Citace

Miroslav Zámečník: Transformace webových aplikací na webové služby. Brno, 2008, diplomová práce, FIT VUT v Brně.

Transformace webových aplikací na webové služby

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Petra Weisse.
Další informace mi poskytli...
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Miroslav Zámečník
15.5.2008

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu práce Ing. Petru Weissovi za cenné rady a připomínky při řešení toho projektu.

Obsah

1 Úvod

Lidé používají web stále více. A stále častěji využívají služeb, které zpřístupňují webové aplikace. Hledají letenky, rezervují hotely, nakupují v internetových obchodech, atd. Většina klasických služeb už má svou webovou podobu. A když ji nemá je možné službu alespoň po internetu objednat. Problém je, že lidé musí projít spoustou irelevantních textů, než se konečně propracují k výsledku. Jako příklad si můžeme představit uživatele webové aplikace, který si chce objednat jeho oblíbenou levnou pizzu, která je v dosahu. K uskutečnění objednávky je třeba přečíst a projít mnoho stránek. Ideou webových služeb a sémantického webu je tuto činnost zautomatizovat a od uživatele ji odstínit. Představa je, aby počítačový agent sám našel všechny blízké pizzerie, zjistil jejich uživatelské hodnocení, spočítal poměr cena ku oblíbenosti typu pizzy, podle toho vybral tu nejvhodnější pizzu pro konkrétního uživatele, objednal ji a informoval ho o čase kdy by mu měla být doručena.

Tato práce volně navazuje na semestrální projekt stejného jména, který byl vypracován o semestr dříve než tato diplomová práce. V jednotlivých kapitolách budou postupně rozebrány všechny standardy a protokoly, které je nutné znát v souvislosti s webovými službami. Mezi nejdůležitější standardy patří SOAP, WSDL nebo UDDI. Dále budou rozebrány jednotlivé koncepty a základní standardy sémantického webu. Dozvíme se něco o sémantických službách, které spojují webové služby se sémantickým webem. Pokusíme se definovat budoucí směr sémantického webu.

Poté co bude probrána potřebná teorie, budou následující kapitoly věnovány samotnému návrhu a implementaci transformace. V návrhu transformace rozebereme možnosti jak postupovat při vyhledávání vstupů a výstupů webových aplikací. Dále se budeme věnovat vytváření popisu webové služby a pokusíme se tuto službu použít.

Vyzkoušíme převést obyčejnou webovou aplikaci na sémantickou a podrobíme ji aplikacím sémantického webu. Vyzkoušíme vyhledávání v sémantické aplikaci pomocí dotazovacího jazyka SPARQL.

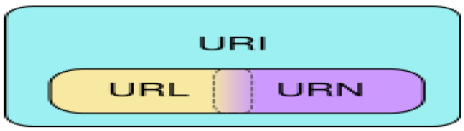
2 Technologie současného webu

Pro uvedení do problematiky webových služeb a sémantického webu je vhodné nejprve říci něco o technologiích, na kterých jsou založeny.

2.1 URI

URI (Uniform Resource Identifier) vznikne složením URL (Uniform Resource Locator) a URN (Uniform Resource Name). URL je všeobecně známé jako adresa konkrétního uzlu v síti, který je pod touto adresou přístupný. URN určuje zdroj jménem bez uvedení jeho umístění. URI musí být složeno pouze z ASCII znaků. Proto bylo vytvořeno ještě IRI (Internation Resource Identifier), které podporuje více bajtovou kódovou sadu unicode.

URI se používá často ve specifikacích XML, pro celosvětově jednoznačnou identifikaci. Předpokladem je nízká pravděpodobnost toho, že si dva lidé zvolí stejné řetězce, když použijí na začátku řetězce URL toho serveru, který mají pod kontrolou. Na některém místě, kde je požadováno URL, lze použít URI, protože je to vlastně také URL. Ovšem toto URL nemusí existovat, což je trochu kontroverzní a pro neznalé poněkud matoucí. Bývá tedy zvykem na toto URI umístit dokument, který popisuje smysl jeho použití nebo popis dané specifikace.



Ilustrace 2.1: Rozdíl mezi URI a URL (převzato z [11])

2.2 XML

Extensible Markup Language nebo-li XML je jednoduchý a flexibilní textový formát derivovaný ze značkovacího jazyku SGML. Hraje důležitou roli při výměně dat mezi internetovými aplikacemi. Umožňuje vytvářet nové značkovací jazyky jakou jsou například XHTML, WSDL, RSS a další. Jedná se o otevřený standard doporučovaný organizací W3C. Aby byl XML dokument validní, je třeba, aby bylo specifikováno schéma, jak má vypadat. Starší DTD má různá omezení a hlavně to není XML dokument. Proto bylo nově vytvořeno XML schéma, které je mnohem mocnější než DTD a přitom jednoduší, i když to tak na první pohled nevypadá.

2.3 XML namespaces

XML je metajazyk, umožňující definovat nové značkovací jazyky. Někdy se stává, že je třeba v jednom XML dokumentu smísit značky z více jazyků. XML namespace je specifikace standardizovaná konsorciem W3C, která umožňuje rozlišovat značky různých jazyků. V XML dokumentu si můžeme definovat jakákoliv jména elementů, které splňují podmínky well-formed dokumentu. Může nastat problém s duplicitou některých elementů. Tento problém je úspěšně eliminován zavedením jmenných prostorů, které dokáží zaručit unikátnost elementů.

Jmenný prostor je deklarován rezervovaným XML atributem "xmlns", jehož hodnota musí obsahovat URI, i když ve skutečnosti se tento dokument v tomto umístění nemusí nacházet. Je lepší používat URI namísto jednoduchého řetězce, protože se tím snižuje riziko duplicity názvu jmenných prostorů. I když tedy jmenný prostor nemusí splňovat konvenci zápisu webových adres, ve většině případů tomu tak je. Název jmenného prostoru nesmí být v žádném případě `xml a xmlns` je možné použít jen v případě definice nových elementů.

Elementy nemusí mít nutně deklarován jmenný prostor (pomocí prefixu), ale i tak patří do prostoru rodiče. Z tohoto prostoru se mohou vymanit použitím jiného prefixu. Jmenný prostor můžeme také oddeklarovat (změnit na žádný), pokud jako jeho hodnotu nezadáme nic. Je možno definovat výchozí jmenný prostor pro daný dokument pro úsporu psaní prefixu elementů. Výchozí jmenné prostory jsou ty jmenné prostory, které jsou z pohledu aktuálního elementu deklarovány bez prefixu. Je nutné dát ovšem pozor na to, že výchozí namespace není aplikován na atributy elementů XML schéma.

2.4 XML schéma

XML schéma, neboli trošku neformálně XSD, vzniklo jako standard organizace W3C. XSD stejně jako DTD vytváří pravidla struktury validních XML dokumentů. DTD je velice rozšířené, ale oproti XSD má několik nevýhod. Na rozdíl od DTD je XSD specifikováno ve formátu XML, což zvyšuje jeho přehlednost. DTD má složitější pro člověka hůře čitelnou strukturu. Navíc DTD nepodporuje některé vlastnosti XSD jako je definování vlastních typů a jmenných prostorů. XSD je zkrátka mocnější než DTD a přitom snáze čitelné, což praktický vyzývá k nahrazení staršího DTD.

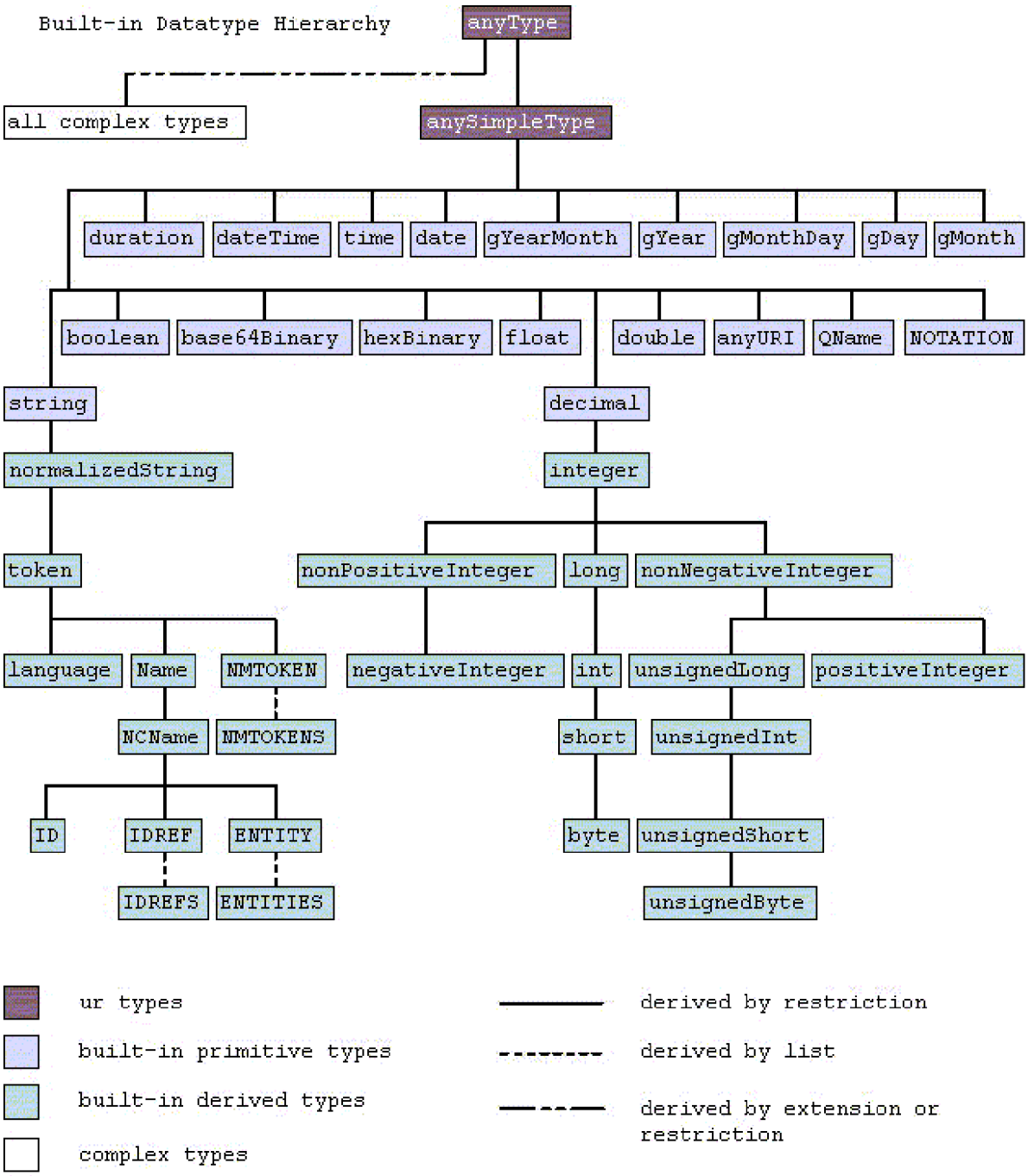
Asi nejmocnější vlastností XSD je možnost vytváření vlastních datových typů. XSD rozlišuje typy podle toho jestli mohou obsahovat potomky (komplexita) nebo nemohou (simpleType). Pro každý element musí schéma určit jeho typ. Jako namespace elementů XML schéma se může použít například zkratky xs nebo xsd. Atributy se deklarují až za vnořenými elementy pomocí elementu "attribute". U atributu rovněž musíme určit jeho název a datový typ.

2.4.1 Datové typy XML schéma

Datové typy jsou samotným základem XML schémat, protože všechny elementy v nich jsou v podstatě datové typy. Jak jsme již řekli, datové typy mohou být jednoduché či komplexní. XML schémata obsahují běžné základní datové typy, jako textový řetězec, celá a desetinná čísla, binární data, logická hodnota, datum, čas, časový interval a několik typů převzatých z DTD pro jejich snazší konverzi do XML schémat.

Vyžadujeme-li složitější kontrolu dat nebo deklarovat vlastní typy je možné použít restrikce. Pomocí sady integritních omezení řídíme přijímané vstupní hodnoty. Integritní omezení mohou být různých typů. Jsou to například omezení na datový typ, interval hodnot, délku řetězce, regulární výrazy syntaxe Perlu a další. Z jednoduchých datových typů je možné vytvářet datové typy komplexní, které jsou též různých typů. Komplexní typy mohou být typu "all", "sequence" nebo "choice". Komplexní datové typy slouží přímo k modelování struktury dokumentu.

2.4.2 Hierarchie typů XML schéma



Ilustrace 2.2: Diagram typů XML schéma (převzato z [18])

2.4.3 Příklad XML schéma

Mějme jednoduchý XML dokument, který obsahuje informaci o ceně z internetového obchodu.

```
<?xml version ='1.0' encoding ='UTF-8' ?>
<cena xmlns="http://www.wa2ws/eshop_xsd">
   <product>
      <category>Mobilní telefon</category>
      <manufacturer>Nokia</manufacturer>
      <name>Nokia 3110</name>
   </product>
   <eshop>Elektromedia.cz</eshop>
   <sum>1260</sum>
   <imported>2007-12-18</imported>
</cena>
```

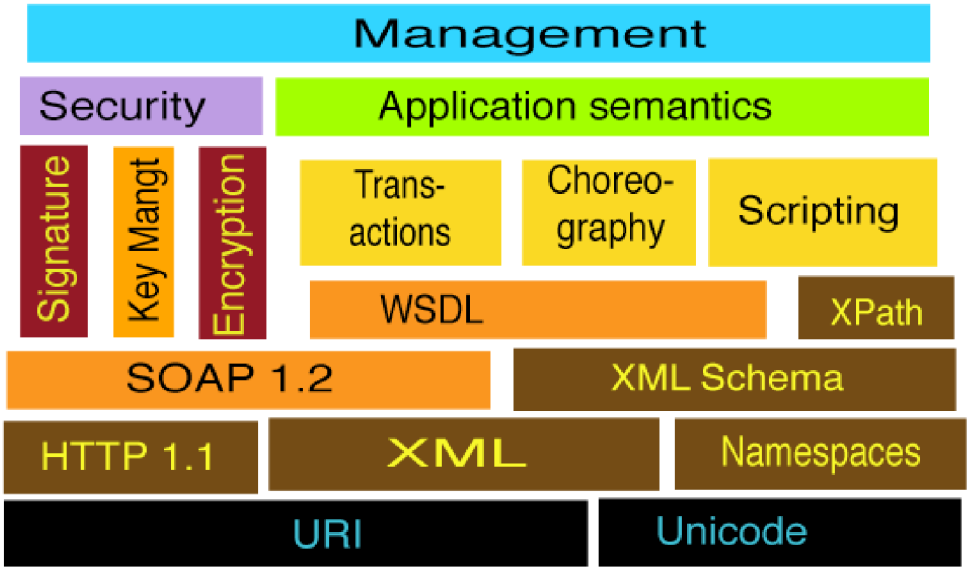
Nyní si ukážeme jak bude pro tento dokument vypadat XML Schéma.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="cena">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="jmeno">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="category" type="xs:string"/>
                     <xs:element name="manufacturer" type="xs:string"/>
                     <xs:element name="name" type="xs:string"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
            <xs:element name="eshop" type="xs:string"/>
            <xs:element name="sum" type="xs:decimal"/>
            <xs:element name="imported" type="xs:date"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Na první pohled se to může zdát nepřehledné, nicméně po bližším zkoumáním se zjistí, že se jedná o velmi jednoduchý formát definice typů, který je mnohem čitelnější a přitom komplexnější než DTD.

3 Webové služby

Webové služby podporují myšlenku sémantického webu tím, že umožňují distribuci internetových aplikací pomocí standardizovaného rozhraní. Webové služby poskytují data, ke kterým je možné strojově přistupovat. Na rozdíl od webových aplikací totiž vracejí výsledky v sémanticky definované struktuře XML dokumentu. Webové služby tak umožňují snadnou a platformě nezávislou integraci aplikací. Popis webových služeb je popsán XML dokumentem v jazyce WSDL (Web Service Description Language). Služby je možné zanést do registru UDDI, aby je potenciální uživatelé mohli snadno nalézt. Jako protokol komunikace je prakticky všude použit protokol SOAP. Aby SOAP byl opravdu platformě nezávislý využívá definici datových typů v XML Schéma, které je určeno k popisu XML dokumentu a je náhradou k DTD. Webové služby běží standardně na aplikačním protokolu HTTP. Když se vše shrne, webová služba se dá definovat jako softwarová služba dostupná na webu prostřednictvím protokolu SOAP, popsaného souborem WSDL a registrovaná v registru UDDI.



Ilustrace 3.1: Vrstvy webové služby (převzato z [19])

3.1 SOAP

SOAP je protokol pro výměnu zpráv ve formátu dokumentů XML, při použití klasických internetových protokolů HTTP nebo HTTPS. Je to hlavní protokol komunikace webových služeb. SOAP je zkratka ze Simple Object Access Protocol a bývá často zaměňována se SOA, což je Service-oriented Architecture. Celkově je tento název poněkud matoucí, protože SOAP není jednoduchý a není určen pro přístup k objektům. Proto SOAP od verze 1.2 již oficiálně není zkratka. Vznikl dříve

než WSDL i UDDI. SOAP se nejčastěji používá jako náhrada RPC pro vzdálené volání procedur. Byl navržen firmami IBM a Microsoft a vycházel z Object Access Protokolu, který byl vytvořen v roce 1998. Je nezávislý na platformě a snadno průchodný přes firewally, protože využívá protokolu HTTP. Všudypřítomnost HTTP a jednoduchost SOAP vytváří ideální základnu k implementaci webových služeb, které mohou být volány téměř z jakéhokoli prostředí.

SOAP původně definoval vlastní typový systém, protože jeho vývoj začal dříve než vzniklo XML schéma. Tento systém však měl velké problémy s kompatibilitou. Snažil se totiž jednotlivé typy zapsat pomocí XML specificky pro různé programovací jazyky. Tato cesta se ukázala býti slepou. Byl tedy zvolen opačný přístup, kdy SOAP neslouží k výměně datových typů, ale XML dokumentů, protože každý programovací jazyk je chápe stejně. To znamená, že se implementace SOAP musí v každém programovacím jazyku přizpůsobit. Je tedy potřeba vyvarovat se generování webové služby z existujícího kódu, protože by vygenerovaná služba nesla závislosti na jazyku kódu.

Nejdůležitější funkcí SOAP je možnost implementace na mnoha různých hardwarových a softwarových platformách. Znamená to, že protokol SOAP může být využit ke spojení neslučitelných systémů. SOAP je první komunikační protokol sloužící k integraci systému, který dosáhl tak širokého uplatnění. Hlavní příčinou toho je, že je mnohem jednodušší k implementaci než starší protokoly. Implementace SOAP vyžaduje řádově jen několik měsíců, zatímco implementace protokolů jako jsou CORBA nebo DCE řádově několik let. Pro SOAP jsou napsány desítky implementací, což se žádnému z jeho předchůdců nepovedlo. Jako některé z mnoha implementací můžeme z mnoha uvést například PHP SOAP[1], SOAP::Lite[2], NuSoap[3], Apache Axis[4].

3.1.1 Transportní mechanismus SOAP

SOAP umožňuje přenos přes SMTP nebo HTTP. V praxi se používá hlavně přenos přes HTTP metodou POST, protože má lepší průchodnost přes proxy servery a firewally. Metoda POST dovoluje posílat data v těle HTTP požadavku. Dále je možné použít přenos přes HTTPS, který přidává šifrování pomocí SSL. Každý HTTP požadavek musí obsahovat hlavičku "SOAPaction", která identifikuje požadavek SOAP. Tuto hlavičku mohou používat firewally k filtrování nebo může nést URI služby, která se má volat. Pokud je tato hlavička prázdná je volána přímo URI, na kterou je tento požadavek směřován. Odpověď ze serveru obsahuje také speciální hlavičku. Název této hlavičky je složen z názvu volané procedury a slova "response". Tato hlavička obsahuje vnořené značky, které představují návratové hodnoty.

1 http://cz.php.net/soap
2 http://www.soaplite.com/
3 http://sourceforge.net/projects/nusoap/
4 http://ws.apache.org/axis/

3.2 WSDL

WSDL je formát XML dokumentu pro popis webových služeb. WSDL dokument obsahuje popis dat, které mohou být předány webové službě tak, aby odesílatel i příjemce věděli, která data si vyměňují. Dále WSDL dokument obsahuje popis operací, které mohou být na takovýchto datech provedeny, tak že příjemce ví, jak tato data zpracovat a následně je odeslat. WSDL je typicky používáno s protokolem SOAP. WSDL bylo vyvinuto firmami Microsoft, Ariba a IBM a je schváleno ve verzích 1.1 a 2.0 (přejmenováno z verze 1.2) konsorciem W3C. WSDL představuje de facto definici webových služeb. Hlavním přínosem verze 2.0 je oddělení abstraktního popisu funkčnosti webové služby od podrobného popisu jak službu volat a používat.

Klient i poskytovatel webové služby musí mít přístup ke stejnému WSDL dokumentu, aby si byli vzájemně schopni porozumět. Klient musí znát jak správně vytvořit dotaz a server musí vědět jak tento dotaz zpracovat. V následujících podkapitolách si popíšeme datový model WSDL.

Doporučovaným postupem při tvorbě webové služby je začínat právě vytvořením dokumentu s popisem pomocí WSDL. Jsou sice nástroje jak vytvořit WSDL dokument přímo ze zdrojového kódu, nicméně to je postup špatný, protože se výsledné WSDL stává závislé na implementaci a tudíž ztrácíme přenositelnost.

Pokud je třeba změnit webovou službu je důležité rozhodnout, zda změny umožní zachovat kompatibilitu s předchozí verzí. Pokud ji zachovají je možné službu změnit, protože starší klienti nepřestanou fungovat v důsledku nových změn. Pokud však změny nebudou kompatibilní se starší verzí je doporučeno zachovat stávající službu a vytvořit službu novou v novém jmenném prostoru. To nám umožní, že budou fungovat stávající klienti a nový klienti budou moci využívat nových vylepšení webové služby.

3.2.1 Datový model WSDL 2.0

3.2.1.1 Description

Kořenová struktura `description` je pouze kontejnerem pro dvě kategorie vnořených struktur. První kategorií jsou WSDL 2.0 struktury a druhou kategorií jsou struktury systémových typů zapsaných ve formátu XML schéma. Přímí potomci struktury `description` jsou struktury `interface`, `binding`, `service`, `import` a `types`. Těmto strukturám se v dokumentu WSDL říká top-level struktury. Povinné jsou pouze struktury `interface`, `binding` a `service`.

3.2.1.2 Types

Struktura systémových typů obsahuje omezení na obsah jednotlivých zpráv. Definuje nové typy a elementy použité v obsahu zpráv pomocí XML schéma. XML schéma je výchozí a prakticky se nic jiného neužívá, nicméně použití definic typů pomocí DTD je však také možné.

3.2.1.3 Interface

Struktura `interface` popisuje sekvenci zpráv, které služba přijímá nebo odesílá. Zprávy se seskupují do operací, kde každá operace je sekvence vstupních a výstupních zpráv. Na struktury `interface` může být uplatněn princip dědičnosti. Každá struktura `interface` musí být pojmenována, aby jí mohlo být použito ve struktuře `binding`. Dále může struktura interface obsahovat struktury `fault`. Jedná se o strukturu shodnou se strukturou `portType` z WSDL verze 1.0, která nedefinuje komunikační protokol. Na tomto místě vzniká most mezi abstraktním rozhraním služby s definicemi typů, operacemi, které jsou absolutně nezávislé, a implementací služby, která přesně říká jaký protokol použita na které URL adrese webovou službu nalézt.

3.2.1.4 Interface Fault

Chyba je událost, kdy během výměny zpráv dochází k narušení normální posloupnosti poslaných zpráv. Chyba nastává například, když je jedna strana nedostupná nebo si přeje ukončit komunikaci. Používá k posílání zpráv mimo pásmo. Tyto zprávy typicky popisují původ, a vysvětlení chyby.

3.2.1.5 Interface Operation

Element "operation" popisuje dostupné operace na rozhraní. Operace je interakce s webovou službou skládající se z množiny mezi sebou vyměněných zpráv. Pořadí a kardinalita zpráv je řízena atributem "message exchange pattern".

3.2.1.6 Interface Message Reference

Jedná se o elementy `input` a `output` souhrnně nazývané jako reference zprávy. Jsou to odkazy na definici zprávy, která je definována ve struktuře `types`. Jde o přiřazení zpráv, které bude daná operace na rozhraní přijímat, a které odesílat.

3.2.1.7 Interface Fault Reference

Jedná se o elementy `infault` a `outfault`. Jako v předchozí kapitole se jedná o reference na elementy chyby definované ve struktuře `types`.

3.2.1.8 Binding

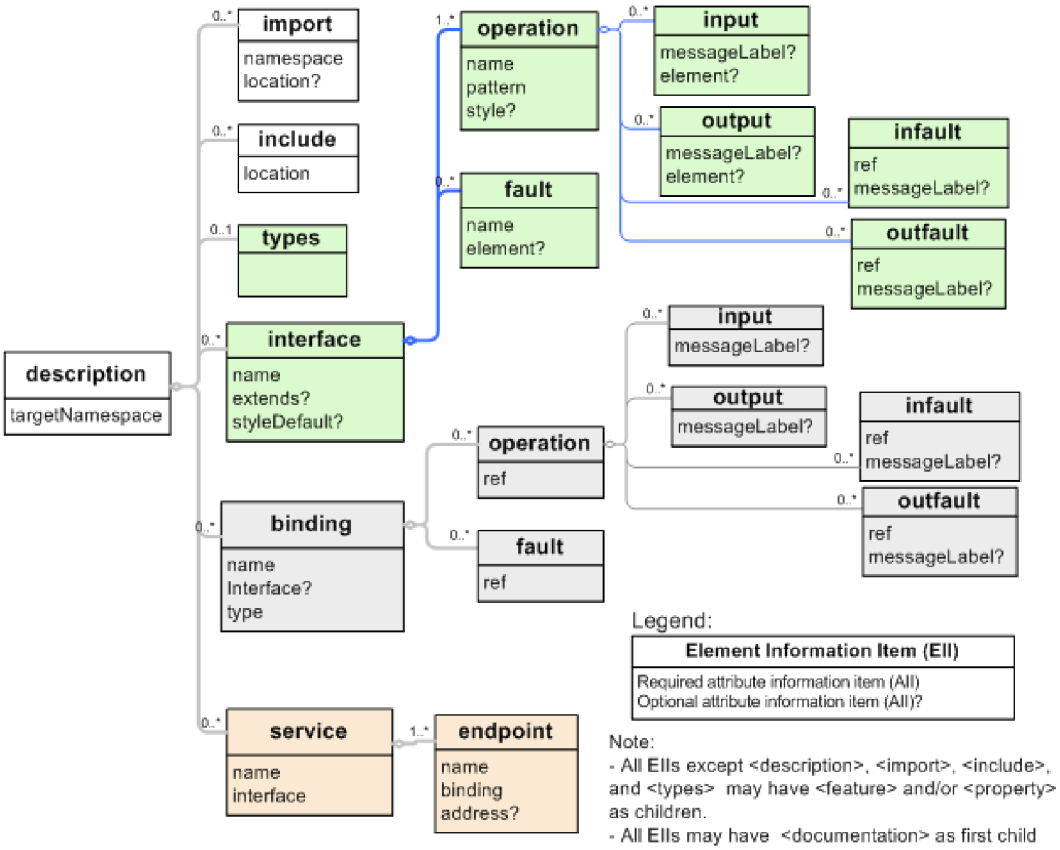
Tato top-level struktura popisuje formát dané zprávy a protokol, což umožňuje definovat koncový bod webové služby. Struktura `binding` obsahuje potřebné detaily pro přístup ke službě. Specifikuje jakými komunikačním protokoly je možné službu volat. Stejně jako struktura `interface` může obsahovat podstruktury s podobným významem, které již nebudou popisovány. Jsou jimi `binding fault`, `binding fault reference` a `binding message reference`. Typicky je komunikačním protokolem SOAP.

3.2.1.9 Service

Tato struktura popisuje množinu koncových bodů služby, na kterých je daná služba dostupná. Její součástí je několik struktur `endpoint`.

3.2.1.10 Endpoint

Struktura je obsažena jednou nebo vícekrát ve struktuře `service`. Struktura definuje, kde se služba nalézá, což je určeno nejčastěji pomocí URL, na kterém je dostupná přes protokol HTTP.



Ilustrace 3.2: Schéma datového modelu WSDL (převzato z [21])

3.2.2 Příklad WSDL 2.0 dokumentu

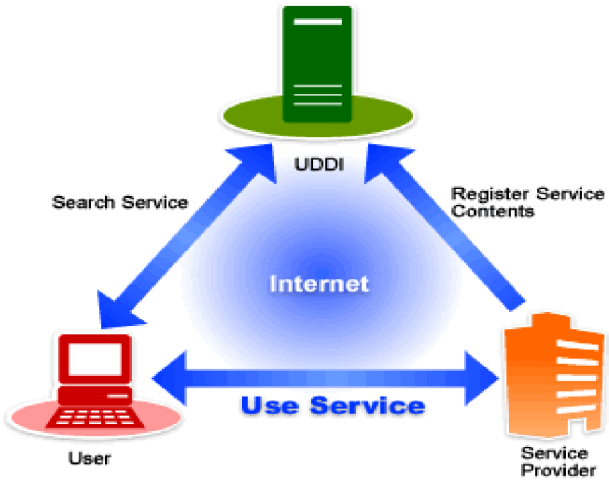
```
<?xml version ='1.0' encoding ='UTF-8' ?>
<description
   xmlns="http://www.w3.org/ns/wsdl"
   xmlns:tns="http://www.example.com/wsdl20sample"
   xmlns:whttp="http://www.w3.org/ns/wsdl/http"
   xmlns:wsoap="http://www.w3.org/ns/wsdl/soap"
   targetNamespace="http://services.multi-eshop.cz/getprices">

   <!-- Deklarace typů zpráv -->
   <types>
      <xs:schema
         xmlns="http://services.multi-eshop.cz/getprices"
         xmlns:xs="http://www.w3.org/2001/XMLSchema"
         targetNamespace="http://services.multi-eshop.cz/getprices">
            <xs:element name="pricelistrequest">
            <xs:complexType>
               <xs:all>
                  <xs:element name="category" type="xs:string"/>
                  <xs:element name="manufacturer" type="xs:string"/>
                  <xs:element name="name" type="xs:string"/>
               </xs:all>
            </xs:complexType>
         </xs:element>
         <xs:element name="pricelistresponse">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="product" type="xs:string"/>
                  <xs:element name="price" minOccurs="0">
                     <xs:complexType>
                        <xs:all>
                           <xs:element name="eshop" type="xs:string"/>
                           <xs:element name="sum" type="xs:decimal"/>
                           <xs:element name="imported" type="xs:date"/>
                        </xs:all>
                     </xs:complexType>
                  </xs:element>
               </xs:sequence>
            </xs:complexType>
         </xs:element>
      </xs:schema>
   </types>
   <!-- Abstraktní rozhraní operací -->
   <interface name="pricelistinterface">
      <operation name="getPrices" pattern="http://www.w3.org/ns/wsdl/in-out">
         <input messageLabel="pricelistmessagereq" element="tns:pricelistrequest"/>
         <output messageLabel="pricelistmessageresp"
element="tns:pricelistresponse"/>
      </operation>
   </interface>
   <!-- Nyní začíná implementačně závislá část  -->
   <binding name="pricelistbinding" interface="tns:pricelistinterface"
      type="http://www.w3.org/ns/wsdl/soap"
      wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/"
      wsoap:mepDefault="http://www.w3.org/2003/05/soap/mep/request-response">
      <operation ref="tns:getPrices" />
   </binding>
   <service name="pricelistservice" interface="tns:pricelistinterface">
      <endpoint name="pricelistendpoint" binding="tns:pricelistbinding"
         address="http://services.multi-eshop.cz/price_list.php/" />
   </service>
</description>
```

Tento WSDL dokument popisuje webovou službu aplikace, která sbírá ceny z různých internetových obchodů a nabízí jejich porovnání. Tato služba má pouze jednu nadefinovanou operaci kvůli jednoduchosti. Úkol služby je nalézt a vrátit ceny z různých e-shopů pro daný produkt. Produkt je určen kategorií, výrobce a názvem.

Teď již konkrétněji pro daný WSDL dokument. Operace pro vrácení cen produktu se skládá z jedné vstupní zprávy, která obsahuje kategorii produktu, výrobce produktu a název produktu, a jedné výstupní zprávy, která je tvořena elementem s celým názvem produktu a poté žádnou nebo několika strukturami, které obsahují cenu produktu, čas importu a e-shop, ve kterém byla cena nalezena. Dále je definováno abstraktní rozhraní služby nad touto operací. Následuje přidělení rozhraní komunikační protokol SOAP pomocí struktury `binding`. Na závěr dokumentu je definice koncových bodů služby.

3.3 UDDI

UDDI (Universal Description, Discovery and Integration) je platformě nezávislá specifikace pro registraci a vyhledávání webových služeb. UDDI je otevřenou firemní iniciativou sponzorovanou společností OASIS. UDDI se též označuje jako zlaté stránky webových služeb. Stejně jako u tradičních zlatých stránek zde můžete hledat firmy nabízející požadované služby, přečíst si základní fakta o nabízených službách a kontaktovat někoho k získání dalších informací. Webovou službu můžete samozřejmě nabízet bez registrace v UDDI, stejně jako když otevřete obchůdek v přízemí vašeho domku a spolehnete se na „ústní“ reklamu mezi sousedy. Pokud však chcete obsáhnout významnější trh, potřebujete UDDI, aby vás zákazníci mohli najít.



Ilustrace 3.3: Vyhledávaní webové služby pomocí UDDI (převzato z [20])

UDDI je internetový adresář společností a jejich webových služeb, kde jednotlivé služby jsou popsány pomocí WSDL. Je reprezentováno strukturovaným dokumentem založeným na XML schéma a podporuje přístup pro vložení a získávání dat pomocí protokolu SOAP. Jedná se o první standard pro zveřejňování webových služeb na internetu, který byl založen ve spolupráci společností Microsoft, IBM a Ariba. Nyní je součástí komunity UDDI více než 300 společností, přičemž 15 z nich je hlavních, které mají hlavní slovo při organizačních záležitostech a schvalování rozhodnutí. UDDI bylo poprvé představeno světu v roce 2000 a nyní je již ve své třetí verzi, která byla publikována roku 2004.

Veřejné UDDI pracuje podobně jako služba DNS. Společnosti se mohou zaregistrovat u jakéhokoliv poskytovatele (IBM, HP, SAP nebo Microsoft). Informace, které poskytnou jsou umístěny do databáze poskytovatele a po určitém čase replikovány do databází ostatních poskytovatelů.

Velký důraz je u UDDI kladen na bezpečnost. Společnosti, které se chtějí registrovat musí nejprve obdržet autorizační token, který jim umožní se přihlásit k poskytovateli UDDI. Další podstatnou vlastností registru je konzistence a validita data. Někdo navíc musí zajistit, že data zadaná společností jsou správná a úplná.

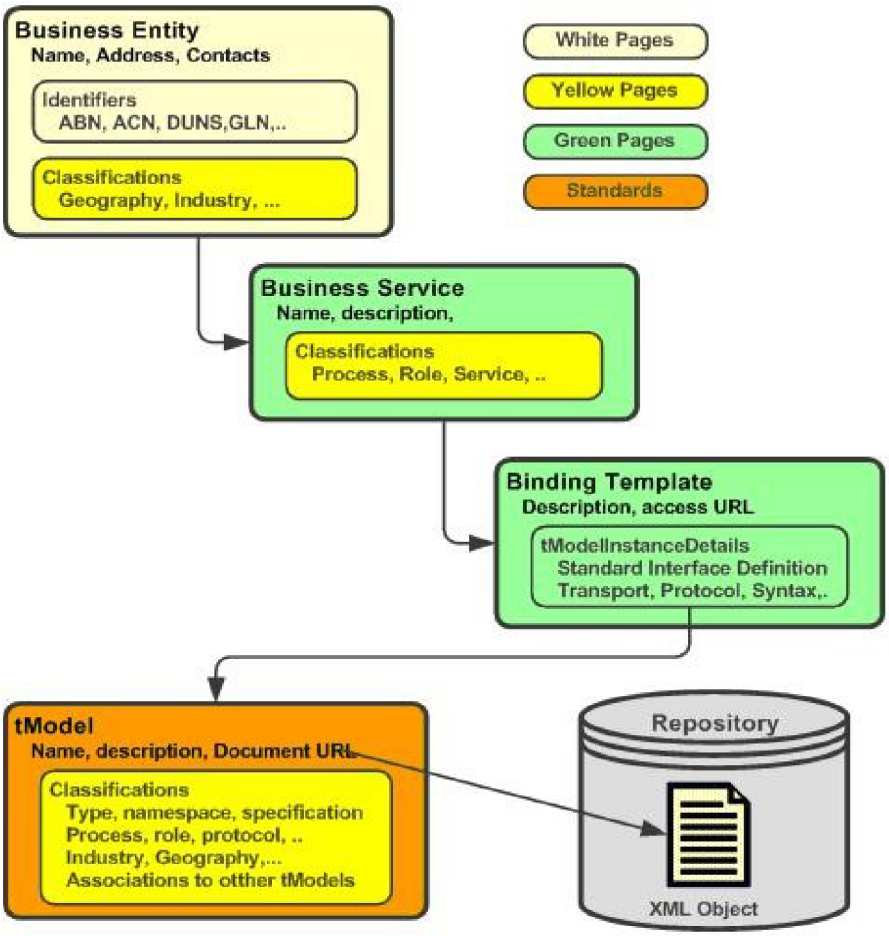
UDDI má dvě hlavní části: registraci a vyhledávání služby. Registrace umožňuje společnosti zadat informace, na základě kterých je možné je v registru nalézt. Společnosti přidávají popisy svých webových služeb. UDDI rozlišuje WSDL soubory pro registraci a pro v hledání služeb, které má vlastní formát XML dokumentu.

Informace v UDDI jsou často rozdělovány do tří kategorií. Bílé stránky popisující společnosti nabízející webovou službu (kontakt, adresa, …). Žluté stránky zahrnující průmyslové kategorie založené na standardních systematikách. Zelené stránky obsahující technické informace o zveřejněných službách, aby kdokoliv mohl napsat aplikaci využívající webovou službu.

Adresář UDDI rovněž obsahuje několik způsobů, jak vyhledávat služby potřebné ke stavbě vašich aplikací. Lze například hledat poskytovatele služby v určeném geografickém místě nebo podnik určitého typu. Adresář UDDI pak dodá informace, kontakty, odkazy a technická data, podle kterých vyhodnotíte, které služby splňují vaše požadavky.

Problémem UDDI je otázka důvěry. Jinými slovy je třeba přesvědčit společnosti, aby důvěřovali hostitelům služby. Důležité je jestli se UDDI stane uznávaným standardem. Dalším konkurentem může být používání papírové formy řešení komunikace mezi společnostmi, protože UDDI se může jevit až jako příliš komplexní. Na druhou stranu informace v registru UDDI nebudou nikdy dost komplexní a konzistentní, aby to vyřešilo celosvětový problém kategorizace. Nicméně pro potřeby webových služeb představuje velmi mocný a široce uplatňovaný model. Alternativou k UDDI může být použití dokumentů specifikace WSIL (Web service inspection

language). WSIL je použito v případě, kdy známe server, se kterým chceme komunikovat a potřebujeme procházet služby, které nabízí.



Ilustrace 3.4: UDDI - Rozdělení na žluté, zelené a bíle stránky (ořevzato z [22])

3.3.1 Datový model UDDI

Datový model registrace společností do UDDI je složen z pěti datových struktur popsaných v následujících kapitolách. Ke každé struktuře je přidělen unikátní klíč, který je ve formě UUIDs (universally unique identifiers). UDDI je navrženo tak, aby podporovalo jakýkoliv typ popisu webových služeb a ne pouze WSDL.

3.3.1.1 businessEntity

Kořenová struktura popisující společnost nebo obsahující jiné struktury, které obsahují informace s registračními údaji. Další struktury jsou vnořeny v této struktuře. Od této entity se ve většině případů začíná vyhledávat.

3.3.1.2 Element publisherAssertion

Struktura popisující asociaci mezi dvěma a více strukturami `bussinessEntity`. Tato asociace může být různého typu.

3.3.1.3 businessService

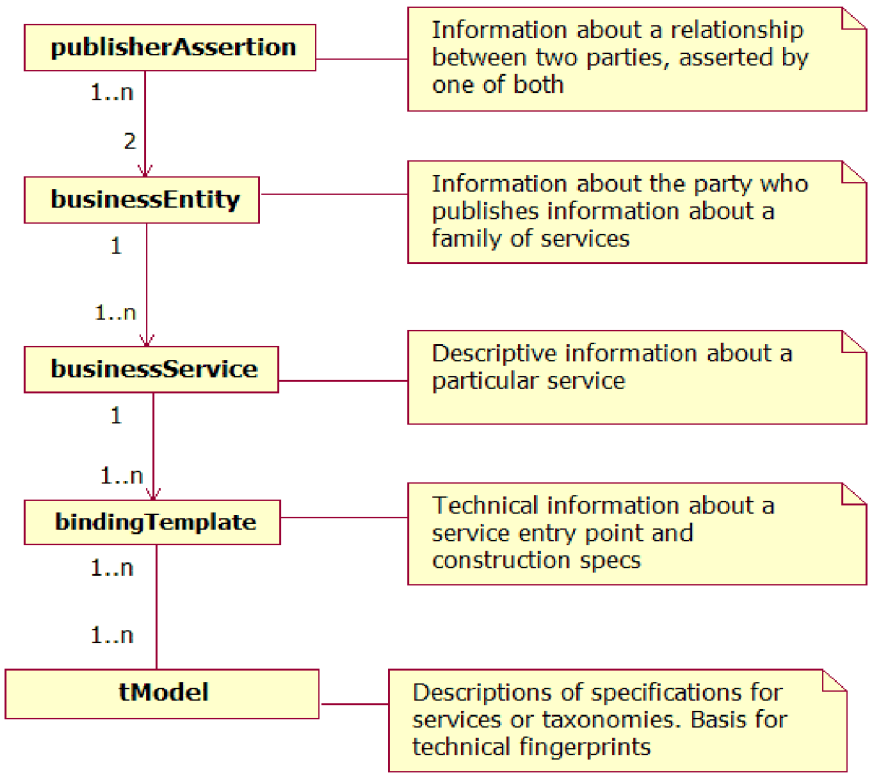
Jméno a popis publikované služby.

3.3.1.4 bindingTemplate

Informace o službě obsahující vstupní bod pro přístup k této službě. Vstupním bodem je URL, na kterém daná služba funguje.

3.3.1.5 tModel

Složení informací jednoznačně identifikující specifikaci služby. V mnoha případech obsahuje `tModel` soubor WSDL, který popisuje rozhraní SOAP k Webové službě XML, ale `tModel` je dostatečně pružný, aby mohl popsat téměř jakýkoli druh služby. Umožňuje vyhledávání dle typu služby.



Ilustrace 3.5: Datový model UDDI (převzato z [23])

4 Sémantický web

Současné webové aplikace využívají jako značkovací jazyk různé standardy jazyka HTML. Ať je to HTML 4 nebo XHTML, stále se však jedná o jazyk, který se sebou nese pouze informaci o formátu a umístění dat, nikoliv však o jeho významu. Tedy jediní, kdo současným webovým aplikacím rozumí jsou lidé. Aby jim mohli porozumět i stroje je potřeba provádět sofistikovanou analýzu dat s potřebným slovníkem pojmů a nejlépe specializovaným počítačem.

Sémantický web je nový pojem, který byl poprvé prezentován v květnu roku 2005, i když tato myšlenka je již mnohem starší. Někde se též můžeme setkat s označením Web 3.0. Sémantika je nauka o významech. Tim Berners-Lee, tvůrce současného webu a ředitel konsorcia W3C, spolu s dalšími spoluautory upozornil, že web není nic než jen halda webových stránek, ve které je čím dál složitější nalézt relevantní informace. Ideou sémantického webu je tuto haldu stránek přepracovat tak, aby byla strojově zpracovatelná. Aktuální web je decentralizovaná platforma pro distribuování prezentací, zatímco sémantický web je decentralizovaný s distribuováním znalostí.

V dnešním světě jsou lidé přes internet schopni uskutečnit téměř cokoliv. Tento proces však může být uskutečněn pouze lidmi, protože dnešní webové aplikace jsou tak napsány. Dobré by bylo, kdyby bylo možné místo lidí nechat pracovat softwarové agenty, kteří by tuto činnost prováděli automaticky za ně. Tento agent by zákazníkovi při nákupu letenky rovnou nabídl v cílové destinaci dostupné hotely a vybraný hotel poté sám rezervoval. Více než o webovou aplikaci se v tomto případě jedná o webovou službu.

Aby něco takového bylo možné je třeba zavést sémantiku webu. Sémantika webu umožňuje počítačové zpracování psaného textu, snadné vyhledávání a lepší kategorizaci dat. Vytvoření sémantického webu je podmíněno vznikem příslušných standardů, slovníků a ontologií. Sémantický web si můžeme představit jako vypsání obsahu relační databáze i s názvy sloupců tabulek. Poté je jasně vidět, co která část znamená. Sémantický web by měl poté zajistit komunikaci mezi různými databázemi, které by si jinak nebyly schopny data vyměnit aniž by někdo jejich komunikaci zvlášť implementoval.

Pro vývoj sémantického webu jsou momentálně nejpodstatnější dvě technologie. Jsou to Extensible Markup Language(XML) a Resource Description Framework(RDF). XML umožňuje snadné vytváření konkrétních značkovacích jazyků pro různé účely a široké spektrum různých typů dat. Skript, který XML dokument zpracovává, ovšem musí znát význam jednotlivých elementů. XML tedy umožňuje vytváření strukturovaných dokumentů, ale neříká nic o významu této struktury.

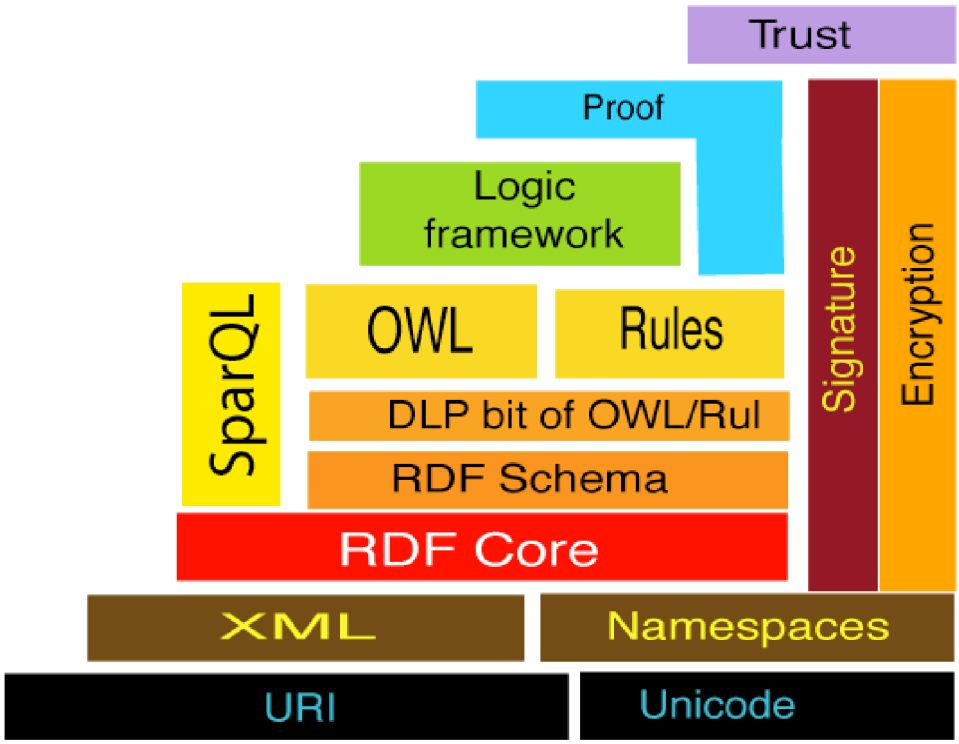
Význam XML dokumentu popisuje RDF, který v sobě kóduje množinu trojic nazvaných *tvrzení*. Každé tvrzení se skládá z trojice *podmět*, *vlastnost* a *předmět*. Tato trojice tak fakticky tvoří

jednoduchou větu. Mohou být zapsány pomocí XML elementů. Tvrzení formuluje to, že dané věci (člověk, web, zpráva) mají vlastnost nebo-li predikát(má bratra, je autorem, má příjemce) s určitou hodnotou (člověk, web). Každá z těchto tří součástí může být zvána jako term. Term může být URI nebo literál.

Přirozený jazyk může selhat u mnohoznačných slov. Pokud neznáme souvislosti, nevíme jaký význam slovo má. Tento problém ovšem neplatí při strojovém zpracování. Problém řeší používání jednotlivých URI, které specifikují jednoznačný význam tvrzení. Jednotlivá URI jsou vázána do definičních tříd, které jsou každému na webu dostupné.

Tohle ovšem není vše, protože dvě databáze mohou používat rozdílné identifikátory pro stejnou věc. Řešení tohoto problému poskytuje třetí základní komponenta sémantického webu, zvaná ontologie.

Uvedeme si schéma s jednotlivými vrstvami sémantického webu. Poté je možné toto schéma porovnat se schématem vrstev webových služeb.



Ilustrace 4.1: Vrstvy sémantického webu (převzato z [24])

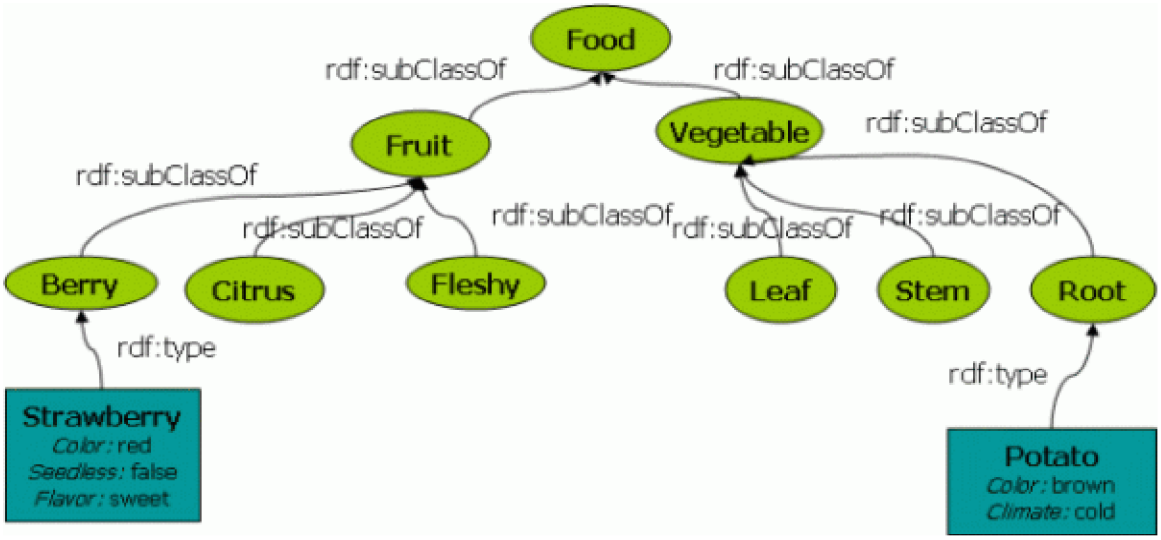
4.1 Ontologie

Ve filozofii se výraz ontologie chápe jako nauka o bytí. V rámci výpočetní techniky, ale bývá definována jinak. V informatice je ontologie specifikována jako explicitní specifikace konceptualizace. Konceptualizace (tj. systém pojmů modelující určitou část světa) musí být specifikována explicitně, tj. nikoliv jen skryta v hlavě svého autora. Účelem ontologie je podpora porozumění mezi lidmi, podpora komunikace mezi počítačovými systémy a podpora návrhu znalostně orientovaných systémů.

Ontologie umožňuje tvorbu metadat neboli informací o informacích. Popisuje význam dané části textu. Dokument se poté skládá z několika významových částí. Metadata se mohou vyskytovat buď v hlavičce dokumentu nebo přímo u dané části textu. Ontologie by měli vytvářet lidé znalí daného oboru, aby bylo možné vytvořit komplexní obecné slovníky, které bude možné použít k rozpoznání typů dat.

Typická ontologie pro web má taxonomii a množinu dedukčních pravidel. Například adresa může být definována jako typ popisu umístění. Třídy, podtřídy a vztahy mezi nimi jsou mocným prostředkem pro účely webu. Na třídy můžeme aplikovat principy dědičnosti. Vztahy mohou být tranzitivní. Pokud je adresa asociována s městem a město se státem, tak poté je též asociována adresa se státem.

Ontologie mohou vylepšit vlastnosti webu mnoha způsoby. Mohou jednoduše zvedat přesnost vyhledávaní. Vyhledávací robot bude vyhledávat pouze dokumenty s danou ontologickou třídou a nedojde tak k nejednoznačnému hledání víceznačných klíčových slov. Dále zvyšují množství znalostí potenciálně zjistitelných z webové stránky.



Ilustrace 4.2: Ukázka hierarchie v ontologii (převzato z [25])

4.2 RDF

RDF (Resource Description Framework) by se podle W3C konzorcia mělo stát technologickým základem sémantického webu. Umožňuje strojově zpracovávat data webových aplikací. Je to v principu metajazyk, který může vytvářet další jazyky, obdobně jako XML. Je založen na přiřazování jednotlivým zdrojům tvrzení. Tvrzení představuje trojice zdroj-predikát-hodnota. Tvrzení vyjadřuje vztah mezi zdrojem a hodnotou. Jako termy jsou použity URI nebo literální hodnoty.

RDF je orientováno na data, na rozdíl od XML, které je orientováno na dokument. V XML jde prakticky vyjádřit to stejné jako v RDF, ale mnohem komplikovanějším a těžko zpracovatelným způsobem. XML umožňuje vytvořit strukturu dokumentu a RDF umožňuje v této struktuře definovat význam. Typické je skloubení RDF s XHTML. Člověk tak dostává informace v klasické textové podobě, kterým rozumí, ale navíc i stroj dokáže ze stránky čerpat znalosti. Nicméně RDF se na webu zatím vyskytuje pouze zřídka. Jediný portál, který jej naplno používá je samotné W3C. Z aplikací RDF získal velkou podporu formát RSS a je jedinou aplikací RDF, která prozatím dosáhla využití jaké se od RDF předpokládá.

4.2.1 RSS

RSS je rodina XML formátů pro syndikaci obsahu webových portálů. Umožňuje uživatelům přihlásit se k odběru novinek. Tento formát se zpravidla používá na portálech, které často mění a přidávají obsahy (např. zpravodajský portál). Původně sloužil jako prostředek k předávání aktuálních novinek mezi jednotlivými servery, což umožňovalo odkazovat na nejaktuálnější obsah. Tohle je schopnost, pro které bylo navrženo samotné RDF, a proto je RSS názornou ukázkou aplikace RDF. Zkratka RSS lze přeložit více způsoby a zastřešuje několik rozdílných protokolů. RSS je možné rozdělit na dvě paralelní vývojové větve. První skupinou jsou verze RSS založeny na RDF. Jsou to formáty RSS 1.*

- RSS 0.9 (RDF Site Summary) bylo původně vyvinuto firmou Netscape, bylo založeno na neúplném pracovním návrhu RDF a není se finální verzí RDF kompatibilní.
- RSS 1.0 (RDF Site Summary) je novější verzí, která již je s RDF plně kompatibilní
- RSS 1.1 je nejnovější verzí založenou na RDF

Druhou skupinu RSS 2.* tvoří formáty, které jsou založeny pouze na XML

- RSS 0.91 (Rich Site Summary) je zjednodušenou verzí RSS 0.9 vyvinutou firmou Netscape. Tato verze nepoužívá RDF a její použití je velmi jednoduché. Jedná se o nejběžnější formát.
- RSS 2.01 (Really Simple Syndication) umožňuje rozšiřitelnost RSS pomocí XML namespace.

Verze v jednotlivých větvích jsou zpětně kompatibilní. Verze v různých větvích již kompatibilní nejsou, nicméně většina dnešních RSS čteček podporuje obě vývojové větve, proto se tímto problémem nekompatibility uživatel vůbec nemusí zabývat. Stejně tak čtečky podporují jiné konkurenční formáty pro syndikaci obsahu, jakým je například Atom.

Na příkladu RSS vidíme, že RDF přístup zde nemá jednoznačnou podporu a má jak své zastánce tak i odpůrce. Pokud má formát RDF dosáhnout většího rozšíření, je třeba více prosazovat jeho výhody. Nastává zde tedy boj mezi obecností a specializací. Momentálně se jeví nejlepší variantou RSS 2.0.

4.2.2 Zápis RDF pomocí N3 a jeho podmnožin

N3 (Notation 3) je jazyk, který má stejné vyjadřovací schopnosti jako RDF, ale na rozdíl od něj je psán ve zjednodušené a přehlednější formě. Je to docíleno tím, že není nutné dodržovat definici jmenných prostorů a dalších pravidel validního XML dokumentu. V této práci je uveden, protože jde o prostředek, který umožňuje jednoduše vytvářet sémantičnost webů bez znalosti implementačních detailů. Pro akademické účely je to naprosto ideální jazyk pro pochopení základů sémantického webu. Po stručném popsání syntaxe bude vše vysvětleno na konkrétním příkladu z praxe.

Každý term je identifikován pomocí URI. Pokud ovšem odstraníme konkrétní zdrojové URI a ponecháme pouze lokální název, tak nás nezajímá, z kterého ontologického slovníku daný term pochází. Tedy URI <http://www.example.com/dir/subdir/file#term> nahrazujeme pouhým <#term>.

Tvrzení jsou zapisovány jako trojice elementů nebo literálů (čísla a řetězce). V trojicích je možno používat speciální oddělovače. Středník odděluje vlastnosti podmětů a čárka odděluje jednotlivé hodnoty. Poté je možné do trojice vložit pomocná slova, která pomáhají pochopit význam tvrzení, ale samy o sobě neznamenají nic. Dále se v jazyce mohou vyskytovat hranaté závorky, které se používají pro tvrzení, kde nechceme mít zdrojový term. Umožňuje nám říci to, že něco existuje, ale nevytvářet referenci na jiný term.

```
<#john> <#age>  64 ;
        <#eyes> "blue";
        <#child> <#jim> , <#elizabeth> .
<#jim>  <#child> [ #age 2  , #age 7 ].
```

Tenhle příklad v N3 nám poskytuje znalosti o Johnovi. John má 64 let, modré oči a syna Jima a dceru Elizabeth. Jim má dvě děti ve věku 2 a 7 let. Nicméně soubor znalostí není kompletní, protože stroj nepozná, že <#john> zastupuje osobu jménem John. Je tudíž nutné přidat další dvě tvrzení, které říkají, že <#john> je osoba, která má jméno "John".

RDF nemůže definovat v jednom dokumentu významy daných termů. V přirozeném jazyce by to možné bylo, ale tomu stroje nerozumí. Je proto nutné definovat sdílené ontologické slovníky.

V jednotlivých slovnících se nachází hierarchicky řazené třídy objektů. Jako jednoznačná identifikace objektu slouží URI. Jednou z organizací, která vytváří ontologické slovníky je Dublin Core.

```
<> <#title> "Semantic web"
<> <http://purl.org/dc/elements/1.1/title> "Semantic web"
```

Na předchozím příkladě je místo termu <#title> použita identifikace pomocí URI, z ontologického slovníku Dublin Core. Takovýchto slovníků existuje více a mohou být nalezeny mimo jiné na serveru http://www.purl.org. Term <> odkazuje na aktuální dokument. Tedy tvrzení udává nadpis aktuálního dokumentu.

Pokud by každý term měl být zapisován takhle dlouhým URI, byl by zápis poněkud rozvláčný. Proto N3 umožňuje přiřazování jmenným prostorům prefixy. Je to obdobné jako atribut xmlns v dokumentu XML. Termy, které čerpají ze jmenných prostorů určených prefixem se poté již nezapisují do závorek, ale jako prefix a term, oddělené dvojtečkou. Pokud neuvedeme název prefixu, budeme předpokládat, že se jedná o aktuální N3 dokument. Následující příklad demonstruje výhody prefixů.

```
@prefix dc:  <http://purl.org/dc/elements/1.1/> .
@prefix : <#> .
<>   dc:title   "Semantic web".
:john :child [ :age 4 ] , [ :age 3 ].
```

Místo využívání sdílených ontologických slovníků si můžeme vytvářet své vlastní, protože se nejedná o nic jiného než o další metadata. Termy jako dc:title jsou RDF vlastnosti. Když chceme vytvořit nový slovník vytváříme tak nové RDF třídy a vlastnosti. Když chceme vyjádřit co daná třída znamená, musíme říci jakého je typu. V RDF by to muselo být termem rdf:type, ale N3 umožňuje použít pouhé písmeno a. Pomocí tříd můžeme vytvářet objekty. Třída vyjadřuje pouze určitý koncept. Objekt může být obecně instancí několika tříd. Ve slovnících je možné vytvářet různé hierarchie tříd.

```
:Woman a rdfs:Class; rdfs:subClassOf :Person .
```

Příklad definuje třídu žena, která je podtřídou třídy osoba. Dále je možné vytvářet vlastnosti, což je schopnost vyjádření vztahu mezi dvěma třídami.

```
:sister a rdf:Property.
```

Někdy existuje takový vztah mezi třídami, o kterých bezprostředně něco víme. Znalost třídy podmětu nazýváme "domain" a znalost třídy předmětu nazýváme "range". Tímto způsobem můžeme jednoduše definovat vlastnost sestra, tak že daná žena je něčí sestra nějaké osoby.

```
:sister rdfs:domain  :Person;
        rdfs:range   :Woman.
```

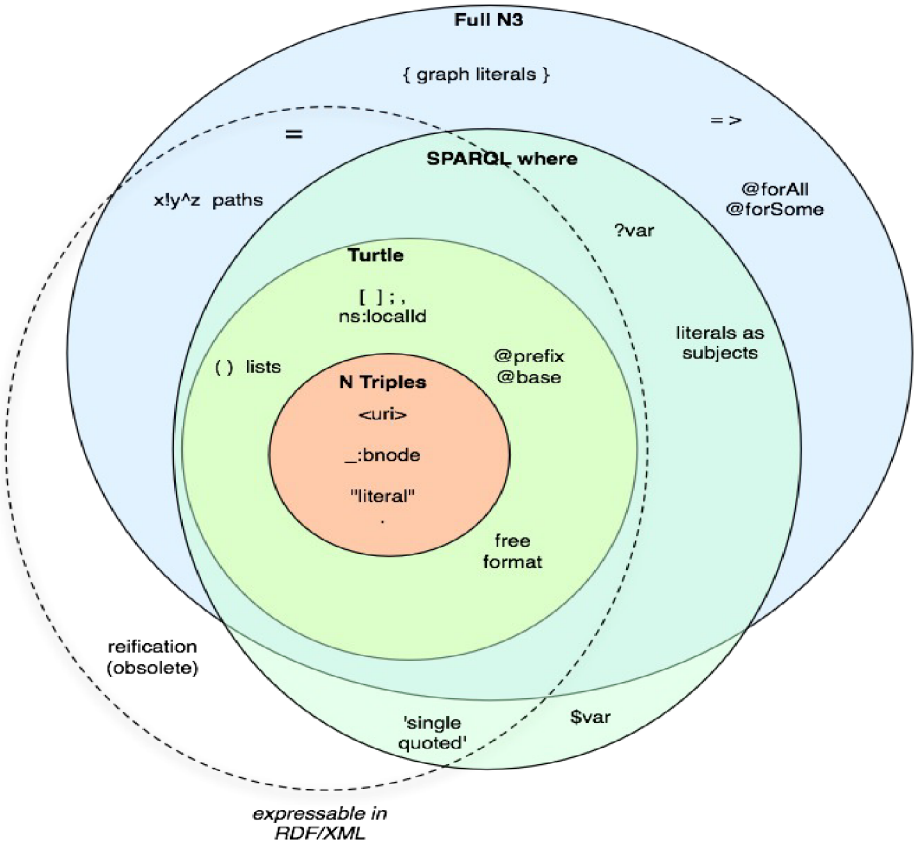
Stává se, že některá třída je definována ve více slovnících. V tomto případě je možné použít vlastnost ekvivalence mezi těmito třídami. Žena je dospělá osoba ženského pohlaví. A nadpis definovaný v našem N3 dokumentu je ekvivalentní nadpisu definovanému v Dublin Core.

```
:Woman = foo:FemaleAdult .
:Title a rdf:Property; = dc:title .
```

Pokud je to možné je třeba používat obecné slovníky, protože to v budoucnosti usnadní agentům významově zpracovávat naše dokumenty.

V případě, že vytvoříme svůj vlastní slovník bývá zvykem a slušností na URI jmenného prostoru slovníku zpřístupnit dokument HTML, který slovně popisuje tento slovník. Slovník může být obecně vytvořen ve formátu RDF schéma nebo OWL a říká se mu schéma, ontologie nebo ontologický slovník.

Existují dvě známe podmnožiny N3. První z nich je N-Triples, což je jazyk, který je značně restriktivní a je psán pro snadné zpracování stroji. Je tedy mnohem hůře čitelný. Další podmnožinou je Turtle (Terse RDF Triple Language), která vybírá ty vlastnosti N3, tak aby byl dodržen RDF model. Turtle je mimo jiné používáno v dotazovacím jazyku SPARQL. A oproti RDF/XML neaplikuje některé restrikce na typ URI.

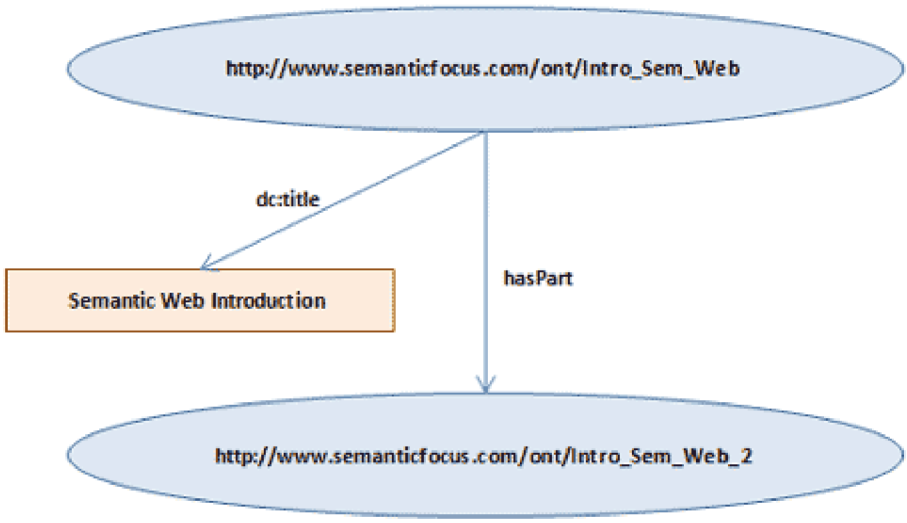


Ilustrace 4.3: Množinové znázornění různých RDF serializací (převzato z [26])

4.2.3 RDF Graf

V článcích o RDF se poměrně často skloňuje pojem RDF graf. RDF graf je množina definovaných trojic někdy nazývána též jako RDF model nebo RDF data. RDF graf lze graficky vyjádřit ohodnoceným orientovaným multigrafem. V tomto grafu se ovály používají jako termy, obdélníky pro literály a orientované úsečky jako vlastnosti. Vlastnosti jsou zapisovány k hranám ve formátu URI. Prefix x: vyjadřuje jakýkoliv URI. Dva uzly mohou být spojeny jednou nebo více hranami. Zdroje mohou mít více vlastností. Literály se v tvrzením mohou vyskytovat pouze jako objekty.

Podgraf RDF grafu je množina trojic vyskytující se v RDF grafu. Znamená to tedy, že každá trojice v RDF grafu tvoří podgraf. RDF grafy se používají v dotazovacím jazyku SPARQL jako podmínky v klauzuli WHERE. Na následujícím obrázku lze vidět příklad RDF grafu. Tento RDF graf reprezentuje několik tvrzení. RDF graf znázorňuje článek, který má titulek a své pokračování. V reálu jsou přitom RDF grafy mnohem složitější a při vysokém počtu tvrzení již prakticky nezobrazitelné.



Ilustrace 4.4: Příklad RDF grafu (převzato z [26])

4.2.4 RDF schéma

RDF schéma (RDFS) slouží k popisu tříd použitých v RDF tvrzeních. Je to rozšiřitelný jazyk pro reprezentaci znalostí, který poskytuje základní elementy pro popis ontologií neboli RDF slovníků. RDF schéma je strukturované a vytváří hierarchickou strukturu. Finální verze doporučení W3C byla vydána roku 2004. Hlavní komponenty RDF schématu jsou použity v komplexnějším jazyce OWL.

Zápis RDF schématu je schodný s vytvářením slovníku v jazyce N3. Oproti N3 je vyžadováno striktnější dodržování XML.

4.2.5 Použití RDF

V této kapitole si na příkladech budeme ukazovat použití RDF. Jak již bylo řečeno, každá část z trojice tvořící tvrzení představuje buď unikátní URI nebo literál. RDF prezentuje informaci o zdrojích na webu. Prezentuje taková data jako jsou autor, titulek, datum vytvoření webových stránek a další. Tyto data již můžeme na webových stránkách naleznout a jsou k tomu použity metatagy. Tyto tagy můžeme označit jako RDF pravidla, které mají vždy jako zdrojový term danou webovou stránkou. Nicméně RDF je obecně platné pro jakékoliv zdroje. RDF umožňuje vytvářet vlastní slovníky, které se nazývají RDF schéma a je tu jistá souvislost s XML schéma. Většinou je lepší nejdříve vyhledat zda daná definice již neexistuje v některém ze standardizovaných slovníků.

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">
   <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
      <contact:fullName>John Smith</contact:fullName>
      <contact:mailbox rdf:resource="mailto:john@smith.com"/>
   </contact:Person>
</rdf:RDF>
```

RDF je možné skloubit s XML tak, že vznikne formát XML/RDF. Je to prakticky XML, které využívá RDF slovníků. Nicméně je jedno jestli použijeme jazyk N3, Turtle, XML/RDF nebo něco jiného. Vše je pouze "zlaté pozlátko", ale sémantický význam trojic zůstává stejný.

Začněme rozvíjet příklad textovým zápisem tvrzení:

```
http://www.example.org/index.html has a creator whose value is John Smith
http://www.example.org/index.html has a language whose value is english.
```

Předmětem tvrzení je v obou případech URL adresa. Tvrzení říkají, že webová stránka ležící na tomto URL je v angličtině a jejím autorem je jistý John Smith.

Následující příklad je zápis prvního tvrzení v jazyku RDF s tím rozdílem, že je zde použito identifikační číslo zaměstnance. O tom, že se tento zaměstnanec jmenuje John Smith se můžeme dozvědět například pomocí dalšího RDF pravidla. Zde je vidět, že každý term je identifikován jednoznačným URI. Vlastnost autor je čerpána ze standardizovaného slovníku pro vlastnosti dokumentů Dublin Core.

```
<http://www.example.org/index.html> <http://purl.org/dc/elements/1.1/creator>
<http://www.example.org/staffid/85740> .
```

Druhé tvrzení na místo URI využívá v případě hodnoty zápis pomocí literálu. Zpracující agent ovšem nemusí vědět, že řetězec "en" identifikuje angličtinu.

```
<http://www.example.org/index.html> <http://purl.org/dc/elements/1.1/language>
"en".
```

Protože tímto zápisem vznikají poměrně dlouhé řetězce, je lepší používat prefixy stejně jako v N3. S těmi vypadá zápis o poznání lépe.

```
@prefix dc:     <http://purl.org/dc/elements/1.1/ .
@prefix staff:  <http://www.example.org/staffid> .
@prefix ex:     <http://www.example.org/> .

ex:index.html   dc:creator   staff:85740 .
ex:index.html   dc:language  "en"
```

4.3 SPARQL

SPARQL (SPARQL Protocol And RDF Query Language) je dotazovacím jazykem a protokolem pro RDF. SPARQL silně podporuje nasazení webových služeb. Znamená to totiž to, že již není třeba definovat žádné funkce zvlášť od generování webových stránek. Postačí vytvoření webové stránky ve formátu XML/RDF nebo případně jiné alternativě a nad těmito daty již je možné provádět specializované dotazy, které jsou typické pro webové služby.

Teprve v lednu roku 2008 byl SPARQL standard označen jako oficiální doporučení W3C. Jde tedy o velmi mladý standard. Nicméně se jazyk SPARQL jeví již jako dostatečně stabilní a vývojáři mohou velmi seriózně uvažovat o jeho nasazení. SPARQL je již implementován v CMS systému Drupal a v dalších.

Jelikož jednotlivá RDF tvrzení jdou přirovnat k řádkům relační tabulky, má jazyk SPARQL hodně společného s SQL. Jednotlivé dotazy jazyku SPARQL se samy o sobě chovají jako RDF tvrzení a proto samotné SQL využít nelze. Není možné využít ani XQuery, protože by nebylo možné vyhledávat v ničem jiném než XML/RDF. Na rozdíl od SQL neumí SPARQL dotazy INSERT, UPDATE a DELETE. Umožňuje pouze získávání dat. Nicméně je možné, že ve verzi 2 již SPARQL tyto operace bude podporovat.

SPARQL se skládá z dotazovacího jazyka a protokolu. Dotazovací jazyk umožňuje dotazy nad RDF, zatímco SPARQL protokol zprostředkovává výsledky dotazů jako webové služby. Pro prezentaci rozhraní dotazů využívá protokol formátu WSDL 2.0, který byl již popsán dříve.

4.3.1 Dotazovací jazyk SPARQL

Na dotazovací jazyk SPARQL se poměrně dlouho čekalo a jeho oficiální vypuštění znamená velký krok kupředu v globálním nasazení sémantického webu. Syntaxe jazyka je podobná syntaxi jazyka SQL. Většina dotazů jazyka obsahuje sadu trojic zvaných *základní model grafu*. Trojice se liší od RDF tvrzení tím, že podmět, vlastnost i předmět mohou být proměnné. Ke splnění podmínky dochází, pokud se základní model grafu obsažený v dotazu shoduje s nějakým podgrafem RDF dat. Přičemž proměnné vyskytující se v dotazu mohou být nahrazeny libovolným termem nebo literálem.

Trojice jsou v dotazech reprezentovány pomocí jazyku Turtle, což je podmnožina popsaného jazyka N3.

Následující příklad zobrazuje jednoduchý SPARQL dotaz, který se snaží nalézt titulek knihy z daného RDF grafu. Dotaz je složen ze dvou částí: Klauzule SELECT určuje proměnné, které se budou vyskytovat ve výsledku a klauzule WHERE obsahuje základní model grafu, který se bude porovnávat s RDF grafem. Tento základní model grafu se skládá z jedné trojice s jednou proměnou (? title) na pozici předmětu.

```
<http://example.org/book> <http://purl.org/dc/elements/1.1/title> "SPARQL MANUAL" .

SELECT ?title
WHERE
{
  <http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> ?title .
}
```

Výsledkem tohoto dotazu bude jediný řádek a ten bude obsahovat literál "SPARQL MANUAL". Dotaz ale obecně může vracet více řádků.

4.3.2 Protokol SPARQL

SPARQL protokol je popsán abstraktně pomocí WSDL 2.0. Má předdefinované jediné `interface SparqlQuery`, a to má jedinou operaci `query`. Stačí pouze definovat jednotlivé SOAP a HTTP `binding` a ty poté přidat do `service`. Zde je vidět hlavní výhoda WSDL 2.0 oproti svému předchůdci WSDL 1.1. SPARQL služba je abstraktně definována a v dokumentu měníme pouze implementačně závislé části WSDL dokumentu.

4.4 OWL

OWL (Web Ontology Language) je rodina jazyků pro popis ontologií a je to jeden ze základních kamenů sémantického webu. Jako ostatní je doporučovaným standardem W3C konzorcia. Je možné je rozdělit na tři podjazyky s různou vyjadřovací silou (OWL Lite, OWL DL, OWL Full). OWL pracuje o úroveň výše než RDF. Nicméně pouze RDF Full může být chápáno jako rozšíření RDF, zatímco OWL Lite a OWL DL jsou pouze rozšíření omezené části RDF. Každý OWL dokument je zároveň RDF dokument a každý RDF dokument je OWL Full dokument. Ontologie napsané v OWL jsou nejčastěji serializovány pomocí syntaxe RDF/XML.

OWL a RDF jsou prakticky dvě stejné věci, ale jazyk OWL na rozdíl od RDF má silnější vyjadřovací schopnosti, mocnější syntax a větší slovník.

4.5 Sémantické webové služby

Již byly popsány klasické webové služby. Sémantické webové služby v sobě spojují výhody webových služeb a sémantického webu. Je to dynamická součást sémantického webu.

OWL-S(Ontology Web Language for Services) je dialektem jazyka OWL, který vznikl pro potřeby sémantických webových služeb. OWL-S zajišťuje skloubení sémantického webu a webových služeb, tak že popisuje vlastnosti webových služeb v jednoznačné počítačově zpracovatelné podobě. Jazyk OWL-S zajišťuje automatické nalezení, spuštění, sestavování a součinnost webových služeb.

Propojení jazyku OWL-S s webovými službami je možné přímo v dokumentu WSDL a to díky jeho rozšířitelnosti. Toto propojení je poměrně přímočaré a nenásilné. Složitější situace nastává při tvorbě konkrétní ontologie.

Sémantické webové služby jsou dnes nejvizionářštější oblastí sémantického webu a jejich nasazení je v prvopočátcích. Pokud si, ale představíme situaci, kdy se nám podaří propojit webové služby, sémantický web a možnosti současného webu 2.0, jakým může být třeba komunikace AJAX, dostáváme se do úplně jiného prostředí webu, kde lidé již nebudou potřebovat klasické webové prohlížeče tak jak je známe dnes.

4.6 Agenti sémantického webu

Opravdová síla sémantického webu může být využita, když lidé vytvoří takové programy, které budou sbírat různé informace na sémantickém webu, tyto informace zpracovávat a vyměňovat si je navzájem. Efektivita těchto agentů bude růst exponenciálně s množstvím strojově zpracovatelných informací na sémantickém webu.

Důležitá je také otázka důvěry v tyto agenty. Proto agenti budou muset být velmi restriktivní a sbírat data opravdu jen z ověřených zdrojů. Samotný agent bude též muset být vytvořen důvěryhodným výrobcem. Bezpečnost bude jistě velmi závažnou věcí co v praxi řešit, aby nic nenarušovalo výsledky, které budou agenti zodpovídat.

Na současném webu již existuje mnoho služeb, které teprve čekají na své objevení. Pokud totiž nejsou zrovna v databázi webových služeb UDDI , není již další možnost, jak je nalézt. Tohle by měli též řešit sémantičtí agenti, kteří vyhledají webové služby, které mohou být sémanticky definovány například jazykem OWL-S.

4.7 Současnost a vize sémantického webu

V současnosti je vize i detaily sémantického webu stále součástí diskuzí. Jsme stále na začátku nové epochy webu. Prozatím jde převážně o vývojovou aktivitu, na níž se podílí řada univerzitních pracovišť obou stran Atlantiku, nicméně existují již první aplikace používané v průmyslu. Pravdou je, že sémantický web zvedne komfort uživatelů, nicméně zase velmi zkomplikuje publikování nových dokumentů. Pro potřebu přístupného publikování by bylo třeba implementovat dobrý publikační nástroj, což může být komplikované vzhledem ke komplexnosti ontologie sémantického webu i vzhledem k nalezení školených publikátorů. I když takovýto nástroj bude existovat, je nutné vytrénovat lidi, aby se naučili myslet sémanticky. Tato vlastnost by mohla radikálně změnit poměr mezi množstvím publikovaných dat a úsilí o jejich publikaci. Na druhou stranu by mohli publikovat jen lidé znalí věci, což by mohlo potencionálně vést ke zkvalitnění webu.

5 Transformace webových aplikací

5.1 Analýza

Předcházející kapitoly položily potřebný teoretický základ k řešení samotné transformace webových aplikací na webové služby. Všechny potřebné znalosti načerpané v předchozích kapitolách nám pomohou nalézt různé možnosti jak transformaci provést. Postupů je několik a liší se obecností a náročností. Problém transformace webových aplikací je obecně ten, že nemůžeme u transformovaných webových stránek počítat s žádným dodržováním standardů nebo moderních metod.

V ideálním případě by byly všechny aplikace napsány v jazyce XML/RDF a my bychom žádnou transformaci nemuseli provádět, protože by stačilo použít dostupných protokolů. Byl by použit dotazovací jazyk SPARQL, který sám v sobě představuje webovou sémantickou službu.

Nicméně k tomuto stavu webových aplikací jsme ještě hodně daleko a je třeba přistoupit k mnohem méně přímočarým řešením. Nejhorším případem je webová stránka, která ani nedodržuje standardy pro psaní HTML. V této stránce je potom velmi těžké identifikovat základní vstupy a výstupy služby.

Naštěstí existují nástroje jako HTML_Tidy[5], které umožňují převést stránku s nedodrženým standardem HTML na validní XHTML dokument. Kromě samotného převodu umožňuje různě modifikovat vstupní HTML, dle konfiguračních parametrů. Tyto nástroje jsou spolehlivé a knihovny pro tyto nástroje existují v mnoha programovacích jazycích. Vidíme, že hned na začátku transformace se setkáváme s poměrně zásadním problémem, ale tento problém je ve většině případů naštěstí snadno řešitelný.

5.1.1 Nalezení služeb

Webové stránky jsou napsány v jazyce HTML a většina z tohoto dokumentu je pro nás nezajímavá. Je třeba hledat pouze webové služby, které webová stránka poskytuje. Typicky může webová stránka obsahovat několik webových služeb. Webovou službu na stránce identifikuje buď formulář nebo také URL dané stránky. Pokud počítáme i URL jako vstup webové služby, poté každá webová stránka obsahuje minimálně jednu webovou službu. Vstupy, které získáváme z URL můžeme rozdělit na ty, které jsou vytvářeny strukturou webu, a ty které vytvářejí parametry URL.

```
http://library/book/nokia/n90.
```

5 Více o projektu HTML_Tidy se dozvíte zde http://tidy.sourceforge.net/

Na této URL bude zobrazen detail telefonu Nokia N90 s příslušným popisem, obrázky a parametry. Máme zde tedy typickou webovou službu, kde na základě jména telefonu získáváme jeho detail.

```
http://www.eshop.cz/?kategorie=mobilni-telefon&vyrobce=nokia
```

Tato URL je rozdílná v tom, že zobrazuje produkt dle parametrů v URL. Typicky se na tomto URL nachází seznam telefonů Nokia. Na rozdíl od předchozí stránky je URL této stránky výsledkem odeslání formuláře metodou GET a je tedy nepřesné, když v tomto případě mluvíme o získávání vstupů webové služby z URL. Spíše se stavím k tomu, že vstupem služby je virtuální formulář.

Formuláře na webových stránkách mají omezený počet typů elementů a je tedy možné identifikovat typy vstupů webových služeb. Zdánlivě jednoduché získání vstupů služby nám ale kazí několik záludností, které formuláře poskytují.

Každý element ve formuláři může mít udané omezení, dle kterého je validován. Například element pro měsíc může přijímat pouze přirozená čísla od 1 do 12. I tohle musíme v naší transformaci zohlednit. Dále mohou formuláře obsahovat elementy, které nemají na výstup služby žádný vliv. Podobným případem jsou povinné a nepovinné elementy. Téměř neřešitelný problém způsobuje to, když je formulář před odesláním pozměněn javascriptem nebo javascriptem přímo odeslán. Nicméně rozumně napsané webové aplikace by měly fungovat i bez javascriptu.

Pomocí formulářů jsme tedy schopni automaticky identifikovat vstupy webových služeb. Otázka zní jak přesně. Za samotného HTML bez znalosti zpracujícího skriptu na serveru nepoznáme jaký formát vstupu daný element přesně vyžaduje. V lepší situaci jsme, pokud je formulář vystavěn na standardech, které využívají sémantiky. Pokud víme co daný element vyžaduje za data, jsme schopni určit také formát těchto dat. Poté můžeme generovat WSDL dané služby s širšími znalostmi o daném vstupu.

U každého formuláře poté zjistíme na jakou URL míří jeho zpracování (atribut action) a jakou metodou je odeslán(GET nebo POST).

5.1.2 Nalezení a identifikace výstupů služeb

Pokud vyplynulo z předchozí kapitoly, že přesná identifikace vstupu webových služeb z HTML je velmi obtížná, tak v případě nalezení výstupu webových služeb je situace ještě horší. Při hledání vstupů webových služeb lze totiž využít toho, že formuláře mají pevně danou strukturu a tedy víme, že vstupy můžeme hledat vždy v jednotlivých elementech formulářů.

V případě hledání výstupů tomu tak ale není. Výstupy webové služby se typicky mohou vyskytovat v jakémkoliv HTML tagu, nebo ani nemusí být HTML tagem vymezeny. Musíme nyní udělat rozhodnutí mezi tím, jestli chceme nalézt pouze výstupní rozhraní služby nebo konkrétní

výstupy. Při hledání výstupního rozhraní služby potřebujeme znát pouze to, jakých typů jsou výstupy. To půjde z HTML kódu zjistit jen velmi složitě, protože implicitně je chápáno všechno jako typ řetězec (xs:string). Při hledání konkrétních výstupů služby se prakticky jedná o konstrukci wrapperu, který by se stal prostředníkem mezi webovou aplikací a klientem, což už je absolutní transformace na webovou službu. Webová služba by byla závislá na HTML kódu, a proto je toto řešení spíše teorií. Při hledání služeb budeme rozlišovat různé úrovně znalostí o výstupech služby.

5.1.2.1 Žádné znalosti o výstupech služby

Touto situací myslíme to, že výstupem služby je HTML stránka, která nepoužívá žádné atributy, třídy či metaznaky k určení výstupu. Je velmi složité vůbec identifikovat výstupy. Jedinou možností je několikanásobné odeslání vstupních dat s vyhledáváním rozdílů mezi jednotlivými výstupními HTML stránkami. Toto řešení je poměrně náročné a nepřesné. Je proto mnohem lepší tímto způsobem webové aplikace na webové služby raději vůbec nepřevádět.

5.1.2.2 Výstupy mají přiděleny atributy

V této situaci jsme na tom již lépe, protože jednotlivé výstupy služeb máme identifikovány pomocí atributů (id, class). Jsme schopni dosáhnout poloautomatické transformace. Transformace nemůže být automatická, protože potřebujeme, aby nějaký člověk rozhodl, které výstupy HTML stránky využít jako výstupy webové služby. Tento člověk určí seznam výstupů výpisem tříd nebo identifikátorů. V ideálním případě definuje také typy výstupů služeb.

Tento případ již umožňuje transformaci na webové služby. Řešením je vytvoření wrapperu, který vstupy webové služby naformátuje tak, aby odpovídali tomu, jak by je odeslal formulář. Poté vrácenou HTML stránku rozebere, a tím, že je schopen identifikovat výstupy služby, vytvoří odpověď webové služby.

5.1.2.3 Známe XPath všech výstupů

Pokud nemůžeme přidělit jednotlivým výstupním tagům třídy, můžeme tyto tagy identifikovat pomocí XPath. Najít správnou a jednoznačnou XPath tagu ovšem také nemusí být úplně triviální.

5.1.2.4 Výstupní stránka používá sémantiky

Toto je ideální případ pro poloautomatickou transformaci webové aplikace na webovou službu, aniž bychom znali kódy na serveru, které aplikaci generují. Nyní již víme, co znamená pojem sémantický web a víme co umožňuje. Jsme tedy schopni ze stránky, která využívá RDF získat konkrétní typy výstupních dat. Opět je třeba lidský zákrok tak, že někdo rozhodne, které RDF tvrzení budou výsledky webové služby.

Nicméně tento člověk nemusí vyjmenovávat konkrétní RDF tvrzení, ale postačí, pokud napíše dotaz nebo dotazy v jazyce SPARQL, které vyberou pouze ta tvrzení, která budou výstupy webové služby.

5.1.2.5 Známe kódy na serveru

V tomto případě jsme na tom úplně nejlépe, protože jsme schopni generovat takové výstupy, jaké chceme. A můžeme se například rozhodnout generovat přímo SOAP odpověď, což je naprosto nejelegantnější a také nejefektivnější.

5.2 Implementace

5.2.1 Testovací aplikace

Aby bylo možné vyzkoušet v praxi principy popsané v předcházejících kapitolách analýzy, je třeba vytvořit jednoduchou webovou aplikaci. Po důkladném zvážení jsem ustoupil od aplikování těchto principů na reálné aplikace. Důvodem je, že implementace by musela řešit spoustu záležitostí, které by nás odváděly od samotného problému transformace. Dále tato testovací aplikace je lehce měnitelná a umožňuje nám pozorovat chování transformace při změně vstupních dat.

Samotná aplikace implementuje velice jednoduché webové rozhraní knihovny. Aplikace zpřístupňuje seznam knih a jejich autorů. Webové rozhraní nám umožňuje vyhledávat knihy dle autora, názvu knihy, ISBN nebo žánru. Výsledky vyhledávání jsou poté různými způsoby vypisovány. Vše je napsáno v duchu pravidla KISS[6], aby byla zřejmá logika generování testovací HTML stránky a nic nás neodvádělo k složitému procházení zdrojového kódu. Aplikace je ovšem dostatečně pružná pro případné změny a dodržuje oblíbené principy MVC[7] architektury. Více o této aplikaci uvádět nebudeme, protože není předmětem této diplomové práce.

5.2.2 Hledání vstupů a výstupů služby

Princip hledání služeb byl popsán v analýze, takže můžeme nyní přímo přikročit k implementačnímu popisu procesu vyhledávání vstupů a výstupů služeb. Budeme uvažovat, že službu identifikuje pouze formulář, i když prakticky by to mohly být i URL stránky. Pro vytvoření formuláře bylo použito frameworku HLEN[8] z důvodů jednoduchosti a čistoty kódu. Bylo by možné použít i jiné třídy pro vytvoření formuláře nebo si ho vytvářet vlastním skriptem. Řekněme, že jde o takové zpestření. Nicméně důležité je jak vypadá formulář po vygenerování v HTML.

---

6 Keep It Stupid Simple. Žádné obskurní algoritmy a psaní čistého jednoznačného kódu.
7 Model View Controller. Architektura odděluje model, aplikačním a prezenční logiku.
8 Jednoduchý a snadno použitelný framework HLEN http://hlen.programujte.com/

```
<div id="bookform">
<form action="/list/" method="get">
<fieldset>
   <legend>Search form</legend>
   <label for="form-title" id="form-title-label">Title</label>
   <input class="text" id="form-title" name="title" type="text"/>
   <label for="form-author" id="form-author-label">Author</label>
   <input class="text" id="form-author" name="author" type="text"/>
   <label for="form-isbn" id="form-isbn-label">ISBN</label>
   <input class="text" id="form-isbn" name="isbn" type="text"/>
   <label for="form-genre" id="form-genre-label">Genre</label>
   <select class="select" id="form-genre" name="genre">
      <option value="">--</option>
      <option value="roman">Roman</option>
      <option value="horror">Horror</option>
   </select>
   <input class="submit" name="submit" type="submit" value="Search"/>
</fieldset>
</form>
</div>
```

Tento formulář obsahuje pouze čtyři vstupní pole s názvy `title`, `author`, `isbn` a `genre`. Pro parsování formuláře použijeme implementovanou třídu `WAParser`, která umí načíst formulář a jeho elementy pak zařadí do pole vstupů služby. Třída `WAParser` používá hojně PHP funkce `SimpleXml`, což je momentálně nejlepší PHP parser XML dokumentů v případě, že nepotřebujeme nad tímto dokumentem provádět složité operace jako jsou přesuny elementů. `SimpleXML` implementuje XPath, čehož je velmi hojně využíváno.

Výsledek po odeslání tohoto formuláře je zobrazen v jednoduché HTML tabulce. Tato výstupní HTML tabulka pro nás ovšem nemá pro definování výstupů služby velký význam.

```
$wa = new WAParser( 'http://library' );
$inputs = array( 'useXpath'=> true ,'value'=> '//*[contains(@id,\'bookform\')]');
$outputs = array( 'useXpath'=> true ,'value'=> array( '//table/tr/th' ));
if( ! $wa->findService( 'booksearch' , $inputs , $outputs ))
   die( $wa->getErrors() );
```

Třída `WAParser`, potřebuje v konstruktoru zadat adresu webové aplikace. Poté se volá její metoda `findService`, která vyžaduje jako parametry jméno služby a pomocná pole pro hledání vstupů a výstupů služby. Abychom nemuseli procházet celou stránku, nastavíme do parametru `$inputs` hodnotu `id` nebo XPath HTML elementu, v jehož obsahu budeme formulář hledat. Výstupy je možné hledat dvěma způsoby a to pomocí obohacujících tříd a nebo též XPath HTML elementů obsahujících výstup služby. V tomto případě jsme použili řešení s využitím XPath. Nutno podotknout, že hledaní výstupů je zde značně pochybné a ideální je výstupy zadávat ručně. Takovéto hledání výstupů by mohlo být dobré pouze v případě, že bychom se snažili zkonstruovat wrapper této služby.

Z tohoto formuláře tedy získáme 3 vstupy typu `xs:string` a jeden vstup, pro nějž musíme vytvořit speciální typ pomocí XML schéma. Bez znalosti kódu je velice obtížné určit přesné typy

vstupů služby a tedy typ elementu v XML schématu. Problém by mohlo řešit přidání extra třídy, která by identifikovala možné vstupy (např. `class="numeric max_20"`). Tohle řešení je ovšem hodně kostrbaté. Dále by daný vstup mohl být identifikován typem v RDF, což už je mnohem lepší přístup. Nejlepším možným řešením je ovšem klasické HTML formuláře nahradit novými formuláři XFORMS, které poskytují mnohem více možností.

Výstupy služby jsou určeny poměrně specificky a jejich hledání je obecně tímto způsobem hodně neohrabané a nedoporučuji se při transformaci tímto směrem vůbec ubírat.

Podobným způsobem můžeme přidávat další služby, a to nám umožní generovat jediný WSDL dokument, který bude obsahovat více služeb. Třída `WAParser` umožňuje vrátit pole, které má takovou strukturu, která je vhodná pro generování WSDL, ale není specifické pro žádnou verzi WSDL. Třída `WAParser` je napsána obecně a je možné ji bez problému vylepšovat. Pro naši transformaci je dostatečná, ale pro složitější transformace je stále na čem pracovat. Nicméně v tomto způsobu transformace nevidím budoucnost a více doporučuji použít další implementované principy. Největší přínos je v možnosti převést formulář na vstupy služby, což má potencionální využití, ale stejně neodpadá nutnost složitého hledání výstupů služby.

5.2.3 Generování WSDL dokumentu

Poté co jsme úspěšně identifikovali vstupy a výstupy služeb, můžeme přejí k jejich projekci do WSDL dokumentu. Použijeme pro to třídu `SimpleWSDL`, kde jako základní parametr využijeme strukturu, kterou nám vrací metoda `toArray` třídy `WAParser`. `SimpleWSDL` tedy přímo navazuje na `WAParser`, což je žádané, protože třída `SimpleWSDL` je implementována zvlášť pro WSDL verze 1.1 a WSDL verze 2.0.

```
$wsdlArr = $wa->toArray();
$wsdl = new SimpleWSDL($wsdlArr);
$wsdl->toXml();
```

5.2.3.1 DOM nebo SAX

Třída `SimpleWSDL` generuje XML dokument a pro tvorbu XML dokumentů jsou známé dva hlavní přístupy a jsou pro ně ve většině programovacích a skriptovacích jazyků vytvořeny vlastní API. Při výběru, který přístup při generování WSDL použít hraje velkou roli v jeho dalším vývoji.

Důležitou roli při výběru mezi těmito API je i samotná preference programátora. Nicméně každé API má své výhody a nevýhody a jejich nasazení je třeba podřídit situaci. Zatímco SAX (Simple API for XML) modeluje parser, DOM (Document Object Model) modeluje XML dokument. Dom je vhodný tam, kde se trvale s XML dokumentem pracuje nebo se generuje. Celý dokument je totiž uložen v paměti na rozdíl od SAX, který zpracovává XML dokument proudově a je řízen událostmi. Pro generování WSDL dokumentu bude tedy použito DOM API.

5.2.3.2 Generování WSDL 2.0

Trochu netypicky začneme od novější verze, protože ta je pro nás důležitější. WSDL 2.0 je oficiálním doporučením W3C a je třeba to dodržovat. Navíc verze 2.0 je mnohem propracovanější než verze 1.1. Dalším důvodem proč přejít na novější verzi je protokol jazyka SPARQL, který je postaven nad WSDL 2.0, i když existuje port i pro verzi 1.1.

Při generování WSDL musíme dávat pozor na správné použití XML jmenných prostorů a XML schéma. DOM API nám k tomu nabízí nějaké pomocné funkce, nicméně jejich použití není úplně jednoznačné a někdy je lepší použít zápis přímo pomocí textového řetězce.

Výsledné WSDL bude generováno na základě pole, které bylo vytvořeno z webové aplikace, pomocí třídy `WAParser` Třída `SimpleWSDL` se stará čistě o prezenční logiku generování WSDL dokumentu pro webovou službu.

WSDL dokument můžeme rozdělit na několik částí. První část tvoří definice jmenných prostorů a definování cílového jmenného prostoru. Jmenné prostory `wsoap` a `whttp`, jsou specifické pro WSDL verze 2.0 a slouží pro zpřístupnění služby.

```
<?xml version="1.0" encoding="utf-8"?>
<description xmlns="http://www.w3.org/ns/wsdl"
   xmlns:tns="http://www.wa2ws.cz/simpleWSDL"
   xmlns:wsoap="http://www.w3.org/ns/wsdl/soap"
   xmlns:whttp="http://www.w3.org/ns/wsdl/http"
   targetNamespace="http://www.wa2ws.cz/simpleWSDL">
   < ... >
</description>
```

Dále je třeba definovat abstraktní datové typy. Tyto typy budou identifikovat data v přenesených zprávách. Typy mohou být jednoduché nebo komplexní. Součástí komplexních typů mohou být typy jednoduché i komplexní a celé to vytváří stromovou strukturu XML schématu. Typy jsou jedinou společnou částí pro obě verze WSDL, protože používají stejný standard XML schéma. Je potřeba pouze dodržet jmenné prostory.

```
<types>
   <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns="http://www.wa2ws.cz/simpleWSDL"
      targetNamespace="http://www.wa2ws.cz/simpleWSDL">
   < ... >
   </xs:schema>
</types>
```

Vstup služby je abstrahován pomocí elementu `booksearchRequest`. Tento element se skládá ze tří jednoduchý typů a jednoho komplexního typu. Jednoduché typy jsou řetězce (`xs:string`) a komplexní typ je jednoduchého typu `tns:genreOptions`. Jednoduchý datový typ `genreOptions` není nic jiného než jednoduchý typ `xs:string`, který má omezení na svůj obsah dle hodnot jednotlivých `option` elementu `select`.

```
<xs:simpleType name="genreOptions">
   <xs:restriction base="xs:string">
      <xs:enumeration value=""/>
      <xs:enumeration value="roman"/>
      <xs:enumeration value="horror"/>
   </xs:restriction>
</xs:simpleType>
<xs:element name="booksearchRequest">
   <xs:complexType>
      <xs:sequence>
         <xs:element type="xs:string" name="title"/>
         <xs:element type="xs:string" name="author"/>
         <xs:element type="xs:string" name="isbn"/>
         <xs:element type="tns:genreOptions" name="genre"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Výstup služby je tvořen elementem `booksearchResponse`. Jednoduše řečeno je typ `booksearchRespons` prostorově neomezené pole struktur `booksearchResponseItem`. Struktura `booksearchResponseItem` je tvořena čtyřmi jednoduchými typy `xs:string`. Teoreticky by mohl být element `genre` omezen podobně jako v případě vstupu, ale tohle nelze jednoznačně automaticky zjistit.

```
<xs:complexType name="booksearchResponseItem">
   <xs:sequence>
      <xs:element type="xs:string" name="title"/>
      <xs:element type="xs:string" name="author"/>
      <xs:element type="xs:string" name="isbn"/>
      <xs:element type="xs:string" name="genre"/>
   </xs:sequence>
</xs:complexType>
<xs:element name="booksearchResponse">
   <xs:complexType>
      <xs:sequence>
         <xs:element maxOccurs="unbounded" type="tns:booksearchResponseItem"
            name="booksearchItem"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Nyní se dostáváme k částem specifických pro každou verzi WSDL dokumentu. Poslední abstraktní částí WSDL dokumentu je deklarace jednotlivých struktur `interface`. V našem případě máme jediné rozhraní, které definuje dvě operace. Dotaz typu `booksearchRequest` a odpověď typy `booksearchResponse`.

```
<interface name="booksearchInterface">
   <operation name="booksearchPost" pattern="http://www.w3.org/ns/wsdl/in-out">
      <input messageLabel="PostMsg" element="tns:booksearchRequest"/>
      <output messageLabel="SuccessfullMsg" element="tns:booksearchResponse"/>
   </operation>
</interface>
```

Element `binding` již není abstraktní a přiřazuje konkrétní protokol pro daný interface. My budeme používat protokoly SOAP a HTTP, což jsou základní protokoly a nic jiného se ani moc nepoužívá.

```
<binding name="booksearchHttpBinding" interface="tns:booksearchInterface"
   type="http://www.w3.org/ns/wsdl/http">
   <operation ref="tns:booksearchPost" whttp:method="POST"/>
</binding>
<binding name="booksearchSoapBinding" interface="tns:booksearchInterface"
   type="http://www.w3.org/ns/wsdl/soap"
   wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/"
   wsoap:mepDefault="http://www.w3.org/2003/05/soap/mep/request-response">
   <operation ref="tns:booksearchPost"/>
</binding>
```

Poslední částí je definice jednotlivých služeb. Z našeho WSDL dokumentu získáváme pouze jedinou službu, jež implementuje rozhraní. Element `service` obsahuje seznam jednotlivých koncových bodů, kde každý koncový bod určuje URL adresu konkretního `binding`.

```
<service name="booksearchService" interface="tns:booksearchInterface">
   <endpoint name="booksearchHttpEndpoint" binding="tns:booksearchHttpBinding"
      address="library/list"/>
   <endpoint name="booksearchSoapEndpoint" binding="tns:booksearchSoapBinding"
      address="library/list_soap"/>
   </service>
</description>
```

5.2.3.3 Generování WSDL 1.1

WSDL 1.1 je již sice překonané, ale ve většině současných aplikací je to jediná dostupná verze. Je jen málo aplikací, které již implementovali WSDL 2.0. Zásadním důvodem, proč je v této diplomové práci implementováno i generování WSDL 1.1, je to, že PHP nemá žádného klienta, který by podporoval SOAP komunikaci ve WSDL režimu i pro novou verzi WSDL. Proto, aby bylo možné vygenerovaný dokument WSDL použít musíme vygenerovat i starší verzi. Naštěstí to oproti generování WSDL 2.0 nepřináší moc práce navíc, protože deklarace abstraktních typů, což je nejkomplikovanější část, je shodná s novější verzí WSDL. Je dobré znát rozdíly obou verzí, proto si uvedeme i starší verzi, abychom je mohli porovnat.

Kořenový element se na rozdíl od novější verze jmenuje `definitions`. Dále jsou použité jiné jmenné prostory, protože starší verze nemá vlastní SOAP a HTTP implementace. WSDL 1.1 má navíc elementy `message`, což je prakticky typ XML schéma zastřešený pod WSDL element. Dále nemá element interface, což je abstraktní zastřešení rozhraní služby.

```
<message name="booksearchRequestMessage">
   <part element="tns:booksearchRequest" name="parameters"/>
</message>
<message name="booksearchResponseMessage">
   <part element="tns:booksearchResponse" name="parameters"/>
</message>
```

Místo definice rozhraní je použito elementu `portType`, kde je jednotlivým `message` určeno, zda jde o vstup nebo výstup. Vstupy a výstupy jsou poté spojeny do operací, které tvoří `portType`.

```
<portType name="booksearchPortType">
   <operation name="booksearchOperation">
      <input message="tns:booksearchRequestMessage"/>
      <output message="tns:booksearchResponseMessage"/>
   </operation>
</portType>
```

Element `binding` poté implementuje `portType`. V našem případě vytváříme `binding` pouze pro komunikací pomocí protokolu SOAP.

```
<binding name="booksearchSoapBinding" type="tns:booksearchPortType">
   <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="booksearchOperation">
      <input>
         <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="urn:wa2ws" use="literal"/>
      </input>
      <output>
         <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="urn:wa2ws" use="literal"/>
      </output>
   </operation>
</binding>
```

A nakonec konkrétní služba slučuje jednotlivé `binding` a přiřazuje jim URL adresu. Žádné `interface` zde není, aby bylo možné považovat službu jako abstraktně definovanou v případě, že neexistuje žádné `binding`.

```
<service name="booksearchService">
   <port name="booksearchPort" binding="tns:booksearchSoapBinding">
      <soap:address location="http://library/list_soap"/>
   </port>
</service>
```

5.2.4 Konstrukce a použití služeb

Tím, že jsme vytvořili popis webové služby, dostáváme do ruky mocnou zbraň. Nyní můžeme tuto webovou službu prezentovat, aniž bychom někomu prozrazovali implementační detaily naší aplikace. Tento popis je zároveň platformě nezávislý, což nám umožní snadnou migraci na implementaci v jiném programovacím jazyce. V reálu existuje více způsobů, jak službu prezentovat, a jak ji používat.

PHP od verze 5 obsahuje knihovnu pro SOAP. Pomocí této knihovny je použití SOAP naprostou hračkou. O samotném SOAP v tomto případě dokonce ani nemusíme nic vědět. SOAP komunikace se dá rozdělit na to zda využíváme WSDL dokument nebo ne. Pokud ano, říkáme, že voláme SOAP ve WSDL módu.

5.2.4.1 Webová služba bez WSDL

K tomu, abychom mohli volat webovou službu nepotřebujeme WSDL. Musíme však znát jména operací a jejich parametry. Tenhle přístup může být použit pouze u takových aplikací, kde známe kód serveru. Poté jde spíše o volání vzdálených procedur než o webovou službu.

Nejdříve je třeba vytvořit SOAP server, což znamená vytvoření objektu SOAPServer a přiřazení funkcí, které bude server nabízet. Na základě SOAP zprávy poté server provede některou z funkcí a vrátí její výstupy v SOAP odpovědi klientovi. Tedy to, že přidáme funkci `booksearchOperation` neznamená to, že ji budeme spouštět, ale pouze to, že ji server může na základě SOAP zprávy vyvolat.

```
$soapServer = new SoapServer(null, array('uri' => 'http://wa2ws/'));
$soapServer->addFunction('booksearchOperation');
$soapServer->addFunction('authorsearchOperation');
$soapServer->handle();
```

Nyní je možné vytvořit SOAP klienta, který volá funkci přiřazenou SOAP serveru. V našem případě klient žádá server, aby vyhledal všechny knihy od spisovatele Remarqua. Funkce musí být na serveru definována a přidaná jako callback funkce do SOAP serveru. V opačném případě SOAP klient vyvolá výjimku, kterou je nutné zpracovat. Výjimku můžeme vyvolat též na SOAP serveru například na základě nesprávného formátu vstupního parametru. Tohle je prakticky jediná možnost, jak oznámit klientovy k jaké chybě došlo na serveru.

```
$soapClient = new SoapClient(null, array(
   'location' => 'http://library/list_soap',
   'uri' => 'http://wa2ws/'));
$result = $soapClient->booksearchOperation( array('author'=>'Remarque') );
```

5.2.4.2 Webová služba s WSDL

Obecně lepším přístupem je použití WSDL dokumentu. Pokud použijeme WSDL dokument může SOAP klient sám zjišťovat dostupné funkce webové služby, jejich parametry a návratovou hodnotu. Server s webovou službou může vlastnit jiná společnost, ale na základě popisu služby pomocí WSDL ví uživatel této služby, jak službu zavolat, jaké parametry a jaký výstup očekávat. Pokud mluvíme o očekávaném masovém rozšíření webových služeb, je popis pomocí WSDL nutností.

SOAP klient může přímo volat skript, který vytváří WSDL dokument z webové aplikace. Zde je vidět hlavní důvod proč bylo generováno také WSDL 1.1. PHP knihovna SOAP by si totiž s novou verzí WSDL dokumentu neporadila.

```
$soapClient = new SoapClient( 'http://waws/wsdl1.php' );
```

U SOAP serveru se obdobně jako u klienta pouze poupraví konstruktor. Přiřazení callback funkce pomocí metody `addFunction` musí zůstat.

Dokázali jsme dostat výsledek celého snažení transformace prakticky na jeden řádek kódu. Nutno podotknout, že jsme si hodně pomohli tím, že jsme si server napsali samy. Jinak by totiž bylo nutné zkonstruovat wrapper, který by výslednou stránku parsoval a snažil se v ní nalézt výstupy. Ne že by to bylo zcela nemožné, ale do obecnosti použití má tento postup opravdu daleko.

5.2.5 Přidání sémantiky

Všechny předchozí implementace se týkaly aktuálního stavu webu, kde možnosti klasických webových aplikací i současných webových služeb dosahují svých limitů. Vymýšlejí se "zkrášlovadla" jako AJAX, ale ty už omezenou technologii aktuálních webových aplikací a služeb příliš nezlepší. Když chceme něco víc, musíme zapojit sémantiku.

Knihovna jako testovací aplikace byla zvolena především proto, že bibliografie má již vytvořenou ontologii, a není nutné se složitě dopátrávat jaké zdroje a predikáty použít. Není nutné ani definovat nové termy. Nyní zobrazíme výpis všech knih v knihovně ve formátu RDF/XML. Správné RDF se skládá v prvé řadě z definice jmenných prostorů a ontologií. Pro náš příklad si vystačíme se základním RDF slovníkem Dublin Core a bibliografickou ontologií.

```
<rdf:RDF xmlns:foaf="http://xmlns.com/foaf/0.1/"
   xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:bibo="http://purl.org/ontology/bibo/">
   < ... >
</rdf:RDF>
```

Po definici jmenných prostorů následuje seznam jednotlivých tvrzení, které tvoří RDF graf. V jednotlivých tvrzeních používáme pouze dva zdroje. Jedním zdrojem je kniha (`bibo:Book`), která je identifikována URI, kde hraje důležitou roli ISBN. Tvrzení s tímto zdrojem používají v tomto RDF dokumentu tří různých predikátů. Vlastností predikátu `dc:creator` je URI identifikující autora dané knihy. Vlastnost predikátu `dc:title` obsahuje název knihy a konečně predikát `bibo:isbn` přiřazuje knize ISBN. V tomto případě nejde o duplicitu, protože ISBN v URI knihy slouží pouze jako jednoznačný identifikátor knihy, ale nic nevíme o tom, jak z daného URI získat ISBN. Predikát `bibo:isbn` je čerpán z bibliografické ontologie, která popisuje různé typy bibliografií, kde kniha je jedním z nich.

Druhým použitým zdrojem je spisovatel (`foaf:Person`). Tento zdroj využívá v tomto dokumentu pouze jeden predikát. Tímto predikátem je `foaf:name`, jehož vlastnost obsahuje jméno a příjmení spisovatele.

Když použijeme trochu představivosti, můžeme toto RDF chápat jako relační databázi se dvěma tabulkami. Jedna tabulka obsahuje seznam knih, kde primárním klíčem je URI identifikující knihu a cizím klíčem je predikát `dc:creator`. Tento cizí klíč je vázán na primární klíč tabulky

spisovatelů, který je určen URI identifikujícím konkrétního spisovatele. Z tohoto srovnání s relační databází můžeme říci, že RDF a jazyk SPARQL nejsou ničím převratným ani složitým. U RDF dat není složité je vytvořit a získávat z nich informace. Složité je vytvořit globálně použitelné ontologické slovníky popisující všechny objekty světa, které potřebujeme v tvrzeních použít. Tohle nikdy nebude možné a proto vznikají specifické ontologie pro konkrétní aplikace. Pokud však pro danou věc již ontologie existuje, je lepší používat tuto, namísto vytváření nové duplicitní.

U tohoto příkladu stálo poměrně značné úsilí nalézt takovou ontologii, která definuje třídu kniha a její derivace. Toto RDF navíc postrádá žánr knihy, který v této ontologii není definován a jeho typ by musel být vytvořen nebo dohledán v jiných ontologiích.

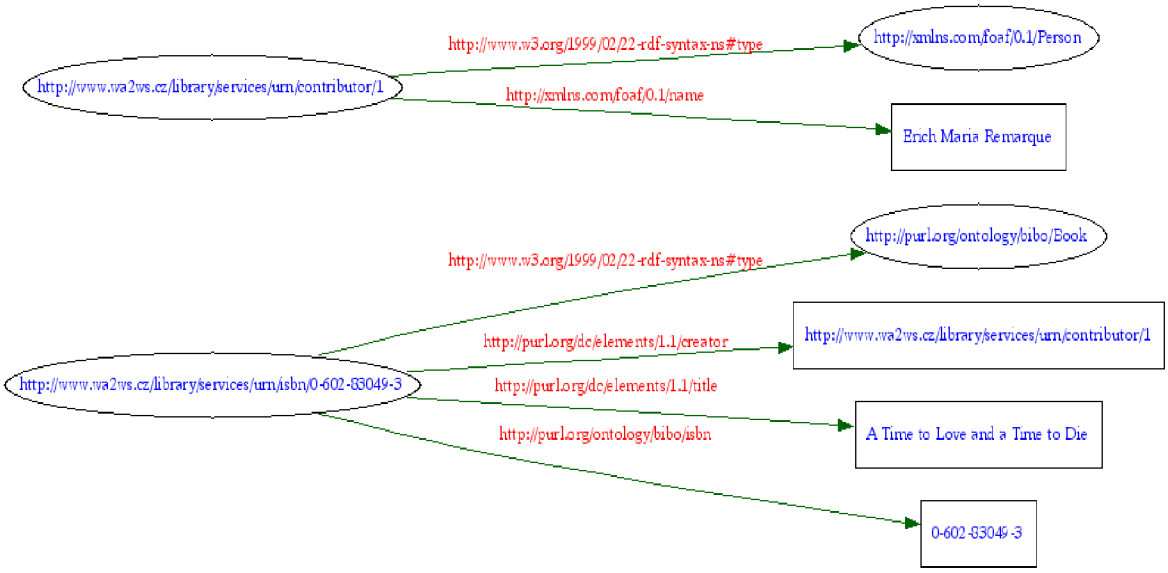
```
<bibo:Book
   rdf:about="http://www.wa2ws.cz/library/services/urn/isbn/0-602-83049-3"
   dc:creator="http://www.wa2ws.cz/library/services/urn/contributor/1"
   dc:title="A Time to Love and a Time to Die"
   bibo:isbn="0-602-83049-3">
</bibo:Book>
   <bibo:Book ... />
<foaf:Person rdf:about="http://www.wa2ws.cz/library/services/urn/contributor/1"
   foaf:name="Erich Maria Remarque">
</foaf:Person>
   <foaf:Person ... />
```

Tento zápis v jazyce RDF/XML se dá snadno přepsat do jazyka N3, popřípadě Turtle. Uvedeme si příklad v jazyce Turtle, který je na první pohled mnohem přehlednější než RDF/XML. To je hlavní důvod, proč se Turtle používá na webových portálech zaměřujících se na sémantický web. Wiki Bibliografické ontologie rovněž uvádí všechny příklady v Turtle.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix bibo: <http://purl.org/ontology/bibo/> .
@prefix dc: <http://purl.org/dc/terms/> .

<http://examples.net/contributors/1> a foaf:Person ;
    foaf:name "Erich Maria Remarque"

<urn:isbn:0-602-83049-3> a bibo:Book ;
    dc:creator <http://www.wa2ws.cz/library/services/urn/contributor/1> ;
    dc:title "A Time to Love and a Time to Die" .
```

Pokud tato RDF data proženeme W3C RDF validátorem[9], tak kromě informace o validnosti RDF data, získáme také jednoduchý RDF Graf. Bohužel obrázek není příliš čitelný, ale to je způsobeno, tím co nám poskytuje RDF validátor.

9 http://www.w3.org/RDF/Validator/



Ilustrace 5.1: RDF Graf vygenerovaný W3C RDF validátorem

5.2.6 Implementace sémantické webové služby

Nyní, když již pracujeme s víceméně sémantickou webovou aplikací, můžeme začít využívat možnosti sémantického webu. Určitě existuje více přístupů, jak se k těmto RDF datům postavit. Pokud nechceme tento RDF/XML dokument složitě parsovat, nabízí se v prvé řadě využití dotazovacího jazyka SPARQL, který je k získávání informací z RDF určený.

Pro potřeby vyzkoušení bylo potřeba nalézt nástroje, které práci s jazykem SPARQL mohou implementovat. Po zvážení byl nakonec vybrána open-source implementace ARC[10], což je flexibilní RDF systém pro systémy LAMP. Je napsán v PHP a pro uchování předzpracovaných tvrzení může využívat například databázi MySQL. Navíc je velmi jednoduše začlenitelný do jakéhokoliv projektu včetně tohoto.

Pro demonstraci možností si napíšeme jednoduchý SPARQL dotaz, kterým vybereme všechny knihy od spisovatele Remarqua. Nejprve definujeme potřebné jmenné prostory.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/> .
PREFIX bibo: <http://purl.org/ontology/bibo/> .
PREFIX dc: <http://purl.org/dc/elements/1.1/> .
```

Vytvoříme dotaz typu SELECT, což je momentálně jediný typ dotazu, který jazyk SPARQL podporuje, i když implementace ARC dokáže provádět nad RDF daty i dotazy INSERT, UPDATE

10 Stránky projektu ARC http://arc.semsol.org/

a DELETE. Tento dotaz bude vybírat knihy s názvem jejich autora a ISBN stejně jako je tomu v aplikaci library.

```
SELECT ?name ?title ?isbn
WHERE
{
   ?person    a  foaf:Person ;
   foaf:name  ?name .
   ?book      a  bibo:Book ;
   bibo:isbn ?isbn;
   dc:creator ?person;
   dc:title ?title .
   FILTER regex(?author, "remarque")
}
```

Na tomto dotazu je jasně patrná podobnost s dotazovacím jazykem SQL. Pouze klauzule WHERE odpovídá podmínkám sémantického webu. Všechny podmínky v klauzuli WHERE jsou napsány pomocí jazyka Turtle. Na rozdíl od Turtle však SPARQL používá proměnných. Proměnné jsou identifikovány otazníkem na začátku slova. Ačkoliv to není z dotazu úplně zřejmé, dochází při něm k operaci typu JOIN nad seznamem knih a seznamem autorů. Tento SPARQL dotaz lze, samozřejmě, modifikovat v závislosti na vyhledávajících kriterií a je tedy možné implementovat vyhledávání podobně jako pomocí SQL.

Zde se dostáváme k implementačnímu cíli této diplomové práci. Povedlo se nám transformovat webovou aplikaci na webovou službu pouze tím, že jsme prosté HTML nahradili RDF, což je základní myšlenka toho, jak vidí budoucnost webu jeho zakladatel Tim Berners-Lee. Tím že se nám to podařilo na téhle jednoduché aplikaci však neznamená, že takhle jednoduše bude možné transformovat všechny aplikace.

6 Závěr

V této diplomové práci bylo mým cílem převést webovou aplikaci na webovou službu. Pro uskutečnění této transformace bylo potřeba položit dostatečný teoretický základ. Byl popsán stav současných webových aplikací, základní principy webových služeb a teorie sémantického webu. Webové služby jsou na vzestupu a jejich nasazení usnadňuje vytváření systémů, které spojují jednotlivé služby napříč celým internetem. Webové služby jsou reálné a jsou mezi námi.

Sémantický web je oproti tomu blízkou i vzdálenou budoucností. Již existují aplikace postavené na principech sémantického webu, ale je jich velmi málo. Sémantický web je stále v plenkách a to i navzdory tomu, že je již několik let prezentován jako technologie zítřka. Nicméně sémantickému webu je v této práci věnováno dostatečné množství prostoru, již z důvodu, že česky psaných dokumentů na toto téma není mnoho. Navíc je sémantický web zajímavou vizí, která mě opravdu oslovila.

Samotná transformace je pojata jako množství postupů a doporučení. Není možné vytvořit obecný postup, který by se dal aplikovat na jakoukoliv webovou aplikaci. Vše záleží na tom, jak je tato aplikace napsána, a na jaké úrovni k ní můžeme přistupovat. Pokud máme přístup k samotnému zdrojovému kódu generujícímu HTML, můžeme postavit opravdu efektivní webovou službu. Pokud chceme transformovat aplikaci jen na základě obsahu webových stránek, je situace jen složitě řešitelná. Není problém nalézt vstupy služeb, ale výstupy služeb. Zatímco vstupy služeb získáváme pouhým zpracování formulářů, na webových stránkách, výstupy musíme dohledávat mnohem sofistikovanějšími postupy a ne vždy je možné je nalézt.

Díky této práci jsem získal poměrně slušný přehled v oblastech webových služeb a sémantického webu. Prakticky mě to připravilo na budoucí kariéru v oblasti webových systémů, protože vidím dál než současný tvůrce webu. Vím, kterým směrem budoucnost webu směřuje a jsem tedy schopen přizpůsobovat své aplikace budoucím potřebám.

Největší budoucnost přikládám v transformaci současný webových aplikací na sémantické aplikace a až následovném transformovaní na sémantické webové služby. Sémantickým webovým službám patří budoucnost a měli by být klíčové pro případný další rozvoj této práce.

Literatura

[1] Newcomer, E. *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. Addison-Wesley, 2002, ISBN 0-201-75081-3.
[2] W3C Web Services activity, Dokument dostupný na URL http://www.w3.org/2002/ws/
[3] W3C Semantic activity, Dokument dostupný na URL http://www.w3.org/2001/sw/
[4] W3C Standard XML Schema, Dokument dostupný na URL http://www.w3.org/XML/Schema
[5] W3C Standard WSDL 2.0, Dokument dostupný na URL http://www.w3.org/TR/wsdl20/
[6] W3C Standard WSDL 1.1, Dokument dostupný na URL http://www.w3.org/TR/wsdl
[7] W3C Standard SOAP 1.2, Dokument dostupný na URL http://www.w3.org/TR/soap12-part0/
[8] W3C Standard SPARQL Query Language, Dokument dostupný na URL http://www.w3.org/TR/rdf-sparql-query/
[9] W3C Standard SPARQL Protocol, Dokument dostupný na URL http://www.w3.org/TR/rdf-sparql-protocol/
[10] Introducing SPARQL, Leigh Dodds, Dokument dostupný na URL http://xml.com/pub/a/2005/11/16/introducing-sparql-querying-semantic-web-tutorial.html
[11] SPARQL FAQ, Dokument dostupný na URL http://thefigtrees.net/lee/sw/sparql-faq
[12] RDF Primer, Dokument dostupný na URL http://www.w3.org/TR/rdf-primer/
[13] Wiki of Bibliographic Ontology, Dokument dostupný na URL http://wiki.bibliontology.com/index.php/Ontology_Working_Draft
[14] W3C RDF Semantic web activity, Dokument dostupný na URL http://www.w3.org/RDF/
[15] About RDF, Dokument dostupný na URL http://www.rdfabout.net/intro/?section=contents
[16] W3C Standard OWL features, Dokument dostupný na URL http://www.w3.org/TR/owl-features/
[17] Wikipedie - URI, Dokument dostupný na URL http://en.wikipedia.org/wiki/Uniform_Resource_Identifier
[18] XML schémata, Jiří Kosek, Dokument dostupný na URL http://www.kosek.cz/xml/schema/wxs.html
[19] Základní popis webových služeb, Tim Berners-Lee, Dokument dostupný na URL http://www.w3.org/DesignIssues/WebServices.html
[20] UDDI tutorial, Nuwan Bandara, Dokument dostupný na URL http://nndo.com/?p=59uwanba
[21] WSDL 2.0, Primer, Dokument dostupný na URL http://www.w3.org/TR/2006/CR-wsdl20-primer-20060106/
[22] Introduction to UDDI, Dokument dostupný na URL https://www.govdex.gov.au/confluence/display/GTM/Introduction+to+UDDI
[23] Use of UDDI datastructure, Dokument dostupný na URL http://java.boot.by/wsd-guide/ch03s03.html
[24] W3C Semantic web tutorial, Eric Prud'hommeaux and Lee Feigenbaum, Dokument dostupný na URL http://www.w3.org/2008/Talks/0305-C-SHALS/
[25] OWL for Services, Dokument dostupný na URL http://www.sei.cmu.edu/isis/guide/technologies/owl-s.htm
[26] Semantic focus, fresh thoughts, Dokument dostupný na URL http://www.semanticfocus.com/
[27] Notation 3, Dokument dostupný na URL http://www.w3.org/DesignIssues/Notation3
[28] Transformace webových aplikací na webové služby, Miroslav Zámečník, semestrální projekt, FIT VUT v Brně, 2008

Seznam ilustrací

Použité zkratky

DOM (Document Object Model) - Objekt reprezetující XML dokument

DTD (Document Type Definition) - Definice typu XML dokumentu

N3 (Notation 3) - Serializér RDF dat

OWL (Web Ontology Language) - Jazyk pro popis ontologií

OWL-S (Web Ontology Language for describing Services) - Pro sémantický popis webových služeb

PHP (PHP Hypertext Processor) - Skriptovací jazyk pro

RDF (Resource Description Framework) - Pro vložení sémantiky do dokumentů

RSS (Rich Site Summary) - Jazyk pro syndikaci obsahu

SAX (Simple API for XML) - Proudový parser XML dokumentu

SOAP (Oficiálně už není zkratka, ale protokol) - Protokol pro výměnu XML zpráv

SPARQL (SPARQL Protocol and RDF Query Language) - Dotazovací jazyk pro RDF

SQL (Structured Query Language) - Základní dotazovací jazyk

Turtle (Terse RDF Triple Language) - Serializér RDF dat

UDDI (Uniform Description Discovery and Integration) - Registr webových služeb

URI (Uniform Resource Identifier) - Unikátní identifikátor zdroje

URL (Uniform Resource Locator) - Jednoznačná adresa, podmnožina URI

WSDL (Web Service Description Language) - Jazyk pro popis webových služeb

XML (Extensible Markup Language) - Rozšiřitelný značkovací jazyk

XPath (XML Path Language) - Jazyk pro vyhledávání XML dokumentech

XSD (XML Schema) - Jazyk pro popis XML dokumentu

Seznam příloh

1. CD disk s následujícím obsahem
    - /text - Text této diplomové práce v různých elektronických formátech
    - /podklady - Podklady diplomové práce
    - /obrazky - Posbírané obrázky
    - /app - Zdrojové kódy
    - /app/readme - Základní příručka k zprovoznění testovací aplikace a její transformace
    - /app/library - Testovací aplikace knihovny
    - /app/library/dump - Dump MySQL databáze pro testovací aplikaci knihovny
    - /app/wa2ws/ - Skripty pro transformaci webových aplikací na webové služby