VYSOKÁ ŠKOLA EKONOMICKÁ V PRAZE

Fakulta informatiky a statistiky

Katedra informačního a znalostního inženýrství

Inteligentní podpora navigace na WWW s využitím XML

Diplomová práce

Jiří Kosek

Vedoucí práce: Ing. Vojtěch Svátek, Dr.

květen 2002

Anotace

Diplomová práce se zabývá problémy efektivní navigace v prostředí WWW. V první části práce jsou nastíněny problémy současných systémů pro vyhledávání a navigaci na Internetu a je popsána architektura systému RAINBOW, který se snaží tyto nedostatky překonat. V dalších kapitolách jsou popsány možnosti implementace vybraných komponent systému – komunikační infrastruktury, navigačního rozhraní a modulu pro stahování stránek. V rámci celé práce je zvažována možnost použití XML a příbuzných technologií pro dílčí části systému. Praktickým výstupem diplomové práce je funkční modul pro stahování stránek a navigační rozhraní pro prohlížeč Mozilla. Navigační rozhraní spolupracuje se třemi dalšími analytickými moduly.

Poděkování

Rád bych poděkoval Vojtěchu Svátkovi za vedení a podporu při tvorbě diplomové práce.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a použil pouze literaturu uvedenou v přiloženém seznamu. Nemám námitek proti půjčení práce se souhlasem katedry ani proti zveřejnění práce nebo její části.

V Praze dne 6. května 2002

Jiří Kosek

Obsah

Kapitola 1

Úvod

Jeden z problémů, které Internet přináší, je informační zahlcení. V současné době je na Internetu přístupných několik miliard webových stránek. Nalezení určité informace v takovém množství dokumentů je velice obtížné. Existují samozřejmě služby jako internetové vyhledávače nebo katalogy, které nalezení potřebných informací výrazným způsobem usnadňují.

Osobně vidím největší problém současných vyhledávacích nástrojů v jejich ovládání. Jedná se o samostatné stránky a uživatel při hledání informace neustále přechází mezi stránkami vyhledávače a nalezenými stránkami výsledku. Celý proces by přitom mohl být mnohem interaktivnější – rozhraní vyhledávače může být začleněno přímo do prohlížeče, může uživateli „inteligentně“ nabízet nejen výsledky hledání, ale i odkazy na další související stránky – ať již třeba obsahově, nebo od stejného autora, patřící stejné firmě apod.

O vytvoření prostředí pro snazší vyhledávání a orientaci ve stránkách usiluje projekt RAINBOW (Reusable Architecture for INtelligent Brokering Of Web information access),[1] na kterém se podílí pracovníci z Katedry informačního a znalostního inženýrství a Laboratoře inteligentních systémů VŠE. Moje diplomová práce se zabývá některými částmi tohoto projektu, především celkovou architekturou systému, jeho komunikační infrastrukturou, uživatelským rozhraním a modulem pro stahování stránek. Výsledkem práce je prototypová implementace RAINBOW, která může být dále rozšiřována pomocí modulů vytvořených ostatními účastníky projektu.

V následující kapitole jsou podrobněji rozebrány problémy vyhledávání na současném webu a nastíněny možnosti jejich řešení v rámci projektu. Třetí kapitola podává stručný přehled XML a na něj navazujících technologií, které mohou být v projektu využity. Následně jsou v samostatných kapitolách podrobně rozebrány otázky komunikace mezi komponentami systému RAINBOW (4 – *Komunikační infrastruktura*), implementace uživatelského navigačního rozhraní (5 – *Navigační rozhraní*) a implementace modulu pro stahování stránek (6 – *Stahování a ukládání stránek*).

[1] `http://rainbow.vse.cz/`

Kapitola 2

Přístupy k navigaci a vyhledávání na webu

Internet (a mám teď na mysli především webové stránky) nabízí obrovské množství informací. Tyto informace mohou být užitečné, mohou být k ničemu, nebo se dokonce jedná o dezinformace. Problém je, jak v tomto kvantu[1] najít ty správné stránky a ještě ověřit, zda na nich prezentované informace jsou hodnověrné.

Samotné nalezení stránky, která obsahuje požadované informace, na základě zadaných klíčových slov dnes nabízí mnoho služeb. V současné době patří mezi nejznámější a nejoblíbenější vyhledávač Google.[2] Přehled dalších vyhledávacích serverů je možné nalézt např. na adrese `http://dir.yahoo.com/Computers_and_Internet/Internet/World_Wide_Web/Searching_the_Web/Search_Engines_and_Directories/`.

Jak vypadá interakce uživatele s vyhledávací službou? Dnešní vyhledávače důsledně oddělují fázi hledání a prohlížení stránek. Uživatel musí nejprve co nejpřesněji formulovat dotaz pro vyhledávací službu. Ta mu vrátí seznam stránek, které by mohly uživatele zajímat. Uživatel si ze seznamu podle názvu stránky a stručného výtahu vybere potencionálně zajímavou stránku a načte ji do prohlížeče. Pokud stránka neodpovídá hledané oblasti, musí se uživatel vrátit zpět a zkusit jinou stránku z výsledku dotazu. Pokud hledaná informace není nalezena na většině vrácených stránek, musí uživatel přejít zpět na stránku vyhledávací služby a snažit se upřesnit dotaz tak, aby byly výsledky hledání přesnější.

Tento postup při hledání informací není pro uživatele příliš pohodlný. Neustálé přecházení mezi stránkami a vyhledávací službou zbytečně zdržuje a navíc zdaleka nepředstavuje nejefektivnější způsob pro rychlé nalezení požadovaných informací. Kdyby se nám podařilo inteligentní vyhledávání integrovat s prohlížením stránek, dostali bychom prostředí, které by bylo pro uživatele mnohem efektivnější a snazší na ovládání. V této kapitole práce popíši, jak by takový inteligentní systém navigace po webových stránkách mohl vypadat. Další části diplomové práce pak popisují prototypovou implementaci dílčích částí tohoto systému. Nejprve se však podíváme, jak vypadají a fungují současné běžně dostupné nástroje pro vyhledávání a navigaci na webu.

[1] Odhadnout celkový počet stránek dostupných na webu je velmi těžké. Například podle studie S. Lawrence a L. Gilese [28] bylo v roce 1999 na webu 800 miliónů stránek. Na jaře roku 2002 měl vyhledávač Google ve svém indexu již 2 miliardy stránek a to jistě nezaindexoval celý web.

[2] `http://www.google.com`

2.1 Vyhledávací služby

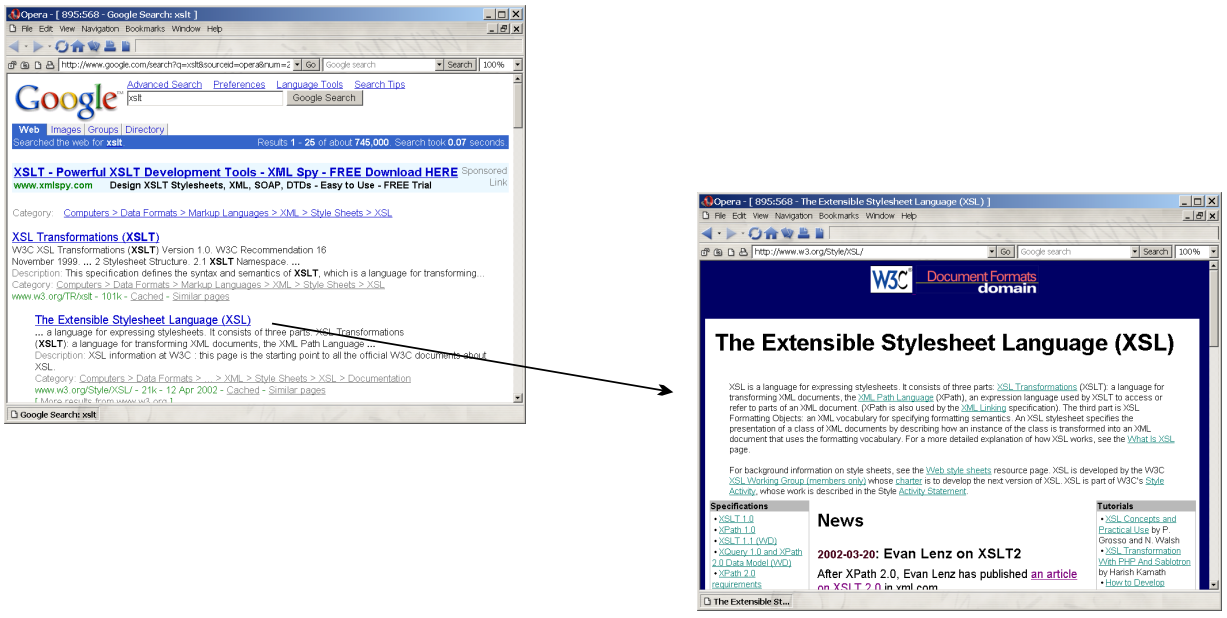
Všechny vyhledávací služby (např. Google,[3] Altavista,[4] nebo český WebFast[5] či Webseek[6]) pracují na stejném principu. Skládají se ze tří do velké míry samostatných modulů – webového robota (crawlera), indexátoru a vyhledávacího modulu.

Webový robot stahuje webové stránky a ukládá je do databáze dokumentů. Ze stránek se vyberou všechny odkazy na další stránky a ty se také stáhnou. Na počátku se stahování zahajuje obvykle pro ručně vybrané servery, případně pro odkazy z nějakého katalogu. Některé vyhledávače umožňují ruční zadání URL adres, které se mají zpracovat. Dost často se pro získání adres ještě nezpracovaných webových serverů používá vytažení nově registrovaných domén ze služby DNS a test, zda na nich běží webový server.

Stažené stránky se poté zařadí do indexu používaného pro fulltextové vyhledávání. Používají se přitom klasické techniky plnotextového hledání – vyřazení stop-slov, lemmatizace a zařazení slov do invertovaného seznamu nebo podobné vyhledávací struktury.

Když pak chce uživatel najít stránku obsahující určitá slova nebo fráze, pracuje s vyhledávacím modulem. Ten prohledá index, seřadí nalezené stránky podle relevance a vrátí je uživateli.

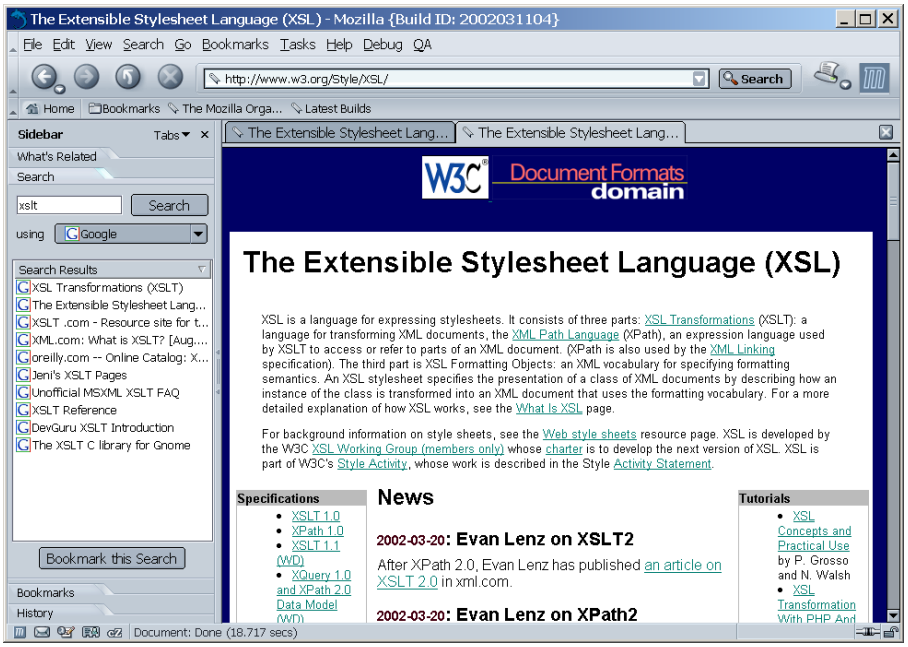
Výsledek je typicky prezentován jako samostatná webová stránka se seznamem odkazů na stránky výsledku (viz obrázek 2.1). Uživatel se kliknutím na název stránky ve výsledku dostane přímo na danou stránku. Pokud však zjistí, že nalezená stránka neobsahuje jím hledané informace, musí se ručně vrátit zpět na stránku s výsledky a zkusit jinou stránku, případně upřesnit dotaz.



Obrázek 2.1: Klasické vyhledávací služby nutí uživatele přecházet mezi nalezenými stránkami a výsledkem hledání

[3] http://google.com
[4] http://altavista.com
[5] http://webfast.cz
[6] http://webseek.cz

Tento způsob práce s prohledávačem není příliš pohodlný. Když se chvíli pohybujeme po webu, na který patří stránka z výsledku, můžeme se dostat poměrně daleko (myšleno počtem postupně aktivovaných odkazů) od stránky s výsledkem hledání. Tento problém do jisté míry řeší novější prohlížeče nebo přídavné moduly, které zobrazují výsledek dotazu v samostatném okně, nezávisle na prohlížené stránce (obrázek 2.2).



Obrázek 2.2: Některé prohlížeče zobrazují výsledek hledání v samostatném okně

I když jsou podobné vyhledávací panely z uživatelského hlediska velkým krokem vpřed, stále nenabízejí vše, co by uživatel potřeboval. Po nalezení stránky s požadovaným tématem chce uživatel často nalézt tematicky stejně zaměřené stránky. V tomto okamžiku by se hodila funkce, která by pomocí shlukové analýzy nebo podobné metody nabídla uživateli seznam podobných stránek.

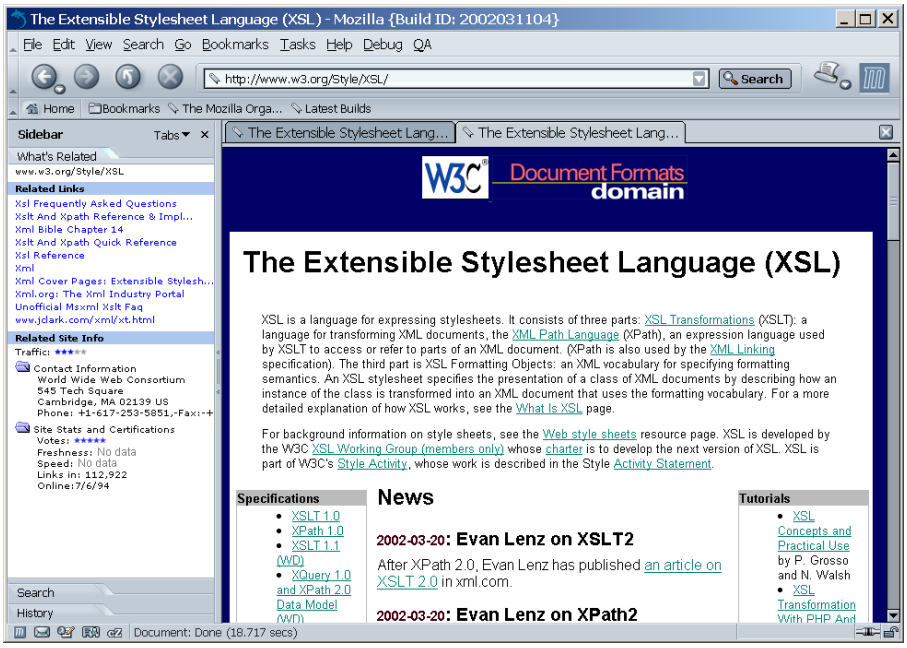
2.2 Navigační asistenti

Navigační asistenti již nejsou zdaleka tak rozšířená aplikace jako vyhledávače. Pod pojmem navigační asistent myslím komponentu webového prohlížeče, která usnadňuje navigaci automatickým nabízením obsahově podobných stránek apod.

Asi nejznámější aplikací tohoto druhu je panel *What's Related* v prohlížeči Netscape Navigator, resp. Mozilla. Ke každé stránce, kterou si prohlížíme, nám automaticky nabízí odkazy na související stránky, kontaktní informace získané z databáze doménových jmen a jednoduchou statistiku o celém webu (viz obrázek 2.3). Seznam podobných stránek nabízených v panelu je přitom získáván pomocí služby Alexa.[7]

Služby jako Alexa jsou užitečné zejména v druhé fázi hledání, kdy už uživatel nalezl alespoň jednu stránku s požadovanými informacemi. Může pak velice snadno přejít na další stránky věnující se danému tématu. Nicméně i tato služba by mohla být lepší. Mohla

[7] `http://www.alexa.com/`



Obrázek 2.3: Navigační panel What's Related v Mozille

by nabízet více druhů informací a metainformací o stránce. Podobné stránky by mohly být členěny do několika kategorií – např. stejné obsahově, od stejného autora, se stejnou strukturou apod. To je směr, kterým se ubírá projekt RAINBOW, jehož vybrané části jsou vytvářeny v rámci této diplomové práce.

2.3 Projekt RAINBOW a jeho možnosti při podpoře navigace

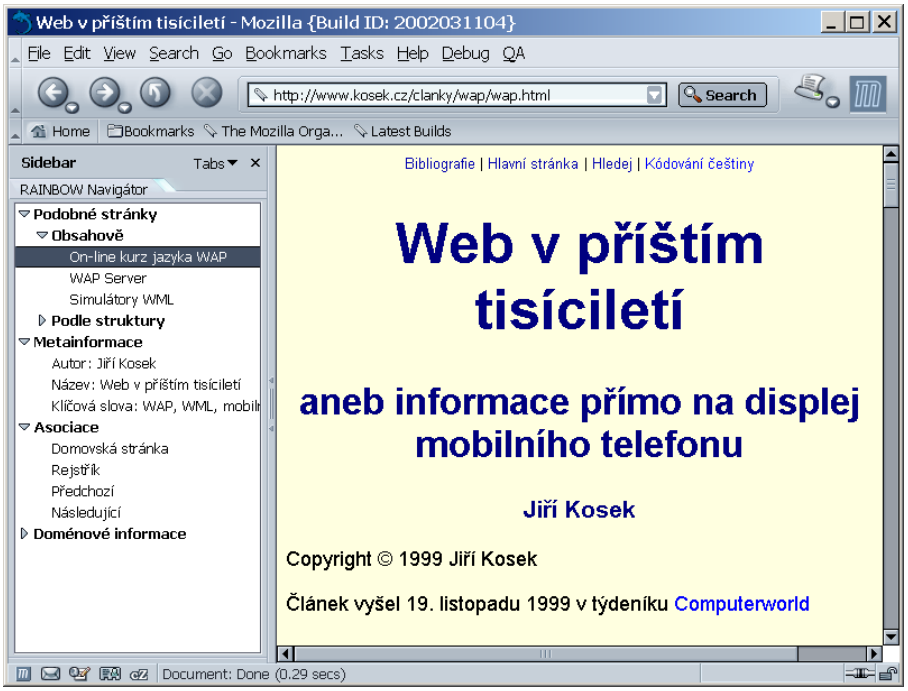
Cílů projektu RAINBOW je několik. Lze je rozdělit do tří skupin:

- Vytvoření navigačního rozhraní pro snazší pohyb po souvisejících stránkách.
- Další rozšíření systému VŠEvěd [1]. Systém VŠEvěd pracuje jako nadstavba nad vyhledávacími službami. Výsledky vyhledávání umí inteligentně seskupovat a kategorizovat. Uživatel tak dostane užitečnější výsledek.
- Systém pro audit stránek, který by hledal chyby na stránkách a upozorňoval na ně.

V současné době se pracuje především na první části projektu, jejímž cílem je usnadnění a zlepšení navigace na webových stránkách. Z uživatelského pohledu je cílem vyvinout rozšířený prohlížeč webových stránek, který bude ve speciálním panelu zobrazovat přídavné informace vztahující se k aktuálně zobrazované stránce. Na rozdíl od panelu *What's Related* by měl systém nabízet větší množství informací o aktuálně zobrazené stránce (viz obrázek 2.4).

Ve finální podobě by navigační asistent RAINBOW mohl ke každé stránce automaticky nabízet následující údaje:

- Sekci *metainformací o dokumentu* obsahující důležité informace o aktuálním dokumentu. Přitom by se systém nemusel soustřeďovat jen na klasické metainformace jako autor,



Obrázek 2.4: Prototyp navigačního rozhraní

název, datum vzniku, klíčová slova, ale i na méně obvyklé – např. to, že jde o akademickou stránku, o obsah vícestránkového dokumentu apod.

- Sekci *podobných stránek*, která bude pro většinu uživatelů zřejmě nejdůležitější, protože umožní rychlý přechod na stránky podobné podle různých kritérií – systém nabídne stránky shodné obsahově, stránky od stejného autora, nebo například stránky se stejnou formální strukturou. Uživatel by se tak mohl snadno pohybovat v prostoru pro něj zajímavých dokumentů bez nutnosti ruční práce s nějakou vyhledávací službou.

- Sekci *asociovaných stránek* odkazující na stránky, které s tou aktuální jistým způsobem souvisejí (a nejde přitom o symetrickou podobnost). Patří sem například odkazy na předchozí a následující stránku v rámci dokumentu složeného z více fyzických stránek, na hlavní stránku, na stránku s hledáním apod. To umožňuje snadný pohyb i po špatně navržených stránkách, které neobsahují vlastní navigační prvky.

- Sekci *doménově závislých informací* obsahující v kondenzované podobě jak metainformace, tak asociace, ovšem takové, které jsou specifické pro jistou problémovou oblast a jako takové popsané specializovanou ontologií. V rámci webového místa univerzity může jít například o identifikaci stránky kurzu, vyučujícího a katedry a jejich vzájemné odkazování. V rámci firemního webu firmy pak např. o hlavní stránku, stránku s referencemi na klienty a stránky produktů.

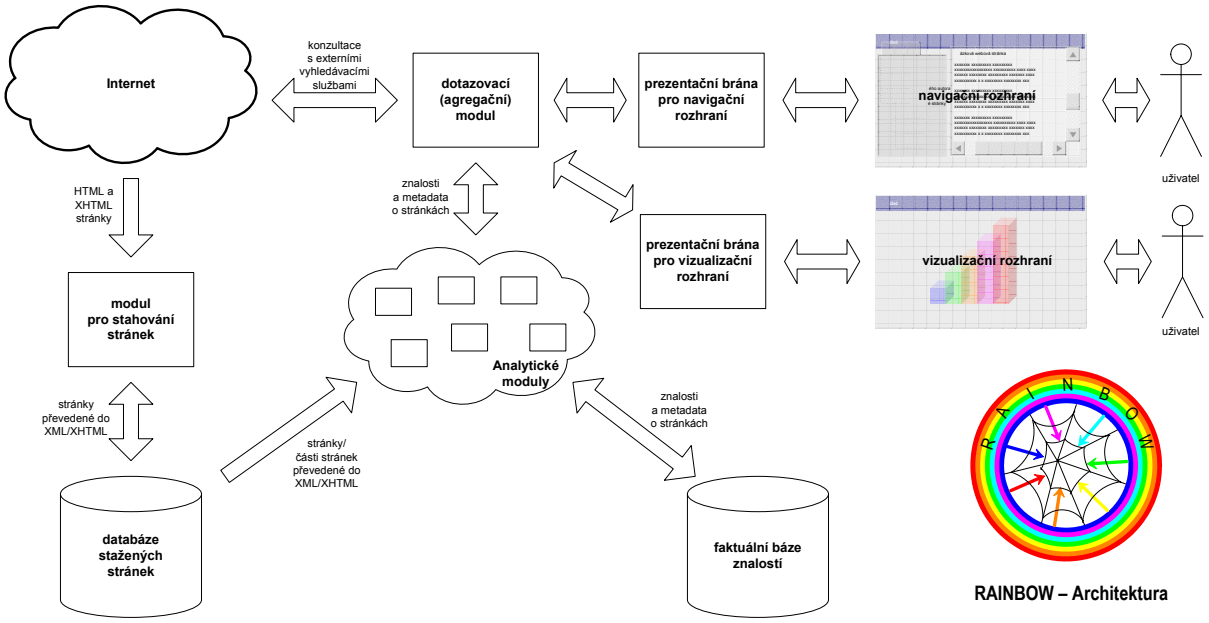
Má-li být navigační rozhraní opravdu ergonomické, mělo by nabízet možnost personalizace – individuálního nastavení podle potřeb uživatele. Uživatel by měl mít možnost vybrat si, jaké typy informací mu má systém nabízet.

Výše popsané úlohy nelze většinou realizovat v reálném čase tak, aby měl celý systém dostatečně rychlou odezvu. RAINBOW proto pracuje podobným způsobem jako klasické internetové vyhledávače. Předem se vždy zpracuje určitá část webu – například stránky jedné

univerzity, firmy, státu, nebo třeba průmyslového odvětví. Stránky se stáhnou, provede se jejich analýza a získané informace se uloží do faktuální znalostní báze. Z této báze znalostí se pak získávají potřebné údaje pro navigační rozhraní. Při analyzování stránek některými moduly mohou být extrahovány i informace důležité pro audit stránek – například syntaktické chyby, chybějící metadata – a odeslány autorovi stránky (pokud se ze stránky podaří získat jeho e-mailovou adresu).

Vlastní analýzou obsahu a struktury stránek a příslušnými analytickými moduly se v této práci nezabývám. Jde o velmi náročnou záležitost, která vyžaduje zapojení technik umělé inteligence, znalostního inženýrství apod. V rámci projektu RAINBOW se počítá s vývojem následujících modulů:

- extrakce informací z textu;
- analýza metadat explicitně přítomných ve webových dokumentech;
- analýza URL adres;
- analýza struktury značek;
- frekvenční analýza termínů;
- analýza topologie odkazů;
- analýza obrazových informací.



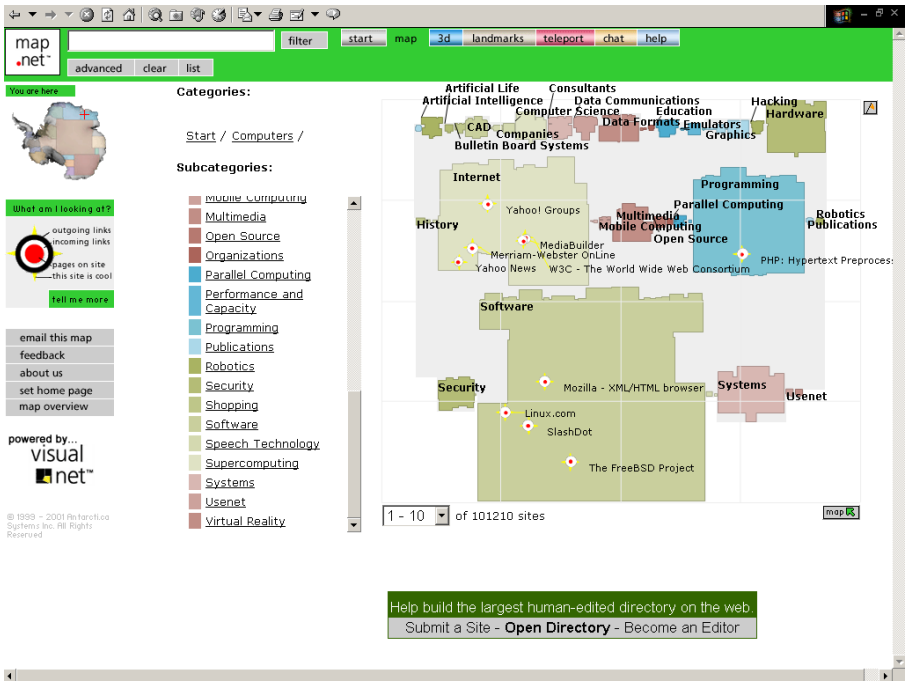
Obrázek 2.5: Architektura RAINBOW

Přehledové schéma celého systému je zachyceno na obrázku 2.5. Modul pro stahování stránek se stará o stažení stránek z vybrané části Internetu. Jelikož mnoho současných webových stránek obsahuje syntaktické chyby, které by ztížily práci dalším analytickým modulům, jsou tyto chyby odstraněny, stránky převedeny do XML a následně uloženy do databáze stažených stránek.

Z této databáze se stránky předají analytickým modulům, které se v nich pokusí nalézt metadata, různé souvislosti a další zajímavé údaje. Tyto údaje se pak uloží do faktuální báze znalostí.

Bude-li chtít uživatel pracovat se systémem, musí mít nainstalován speciální prohlížeč, který pro právě prohlíženou stránku kontaktuje dotazovací modul. Ten zjistí od ostatních modulů a z faktuální báze informace o aktuální stránce a předá je zpět navigačnímu rozhraní. Podle povahy údajů může dotazovací modul některé informace získat „on-line“ od určitých analytických modulů, případně od externích služeb dostupných mimo RAINBOW. Tyto výsledky se zkombinují se složitěji získanými informacemi, které jsou uloženy „off-line“ v již zmíněné faktuální bázi znalostí.

Dotazovací modul přitom nekomunikuje přímo s uživatelem, resp. rozhraním, které uživatel používá. Mezi uživatele a dotazovací modul je vložena ještě jedna vrstva, která umožňuje převedení získaných informací o stránce do různých podob. Kromě dříve popsaného navigačního rozhraní tak může později vzniknout i další rozhraní, které bude výsledky prezentovat odlišnou formou – např. vizuálně. Inspirovat se v tomto případě můžeme například systémem Visual.net firmy Antarcti.ca, který umožňuje prostorovou vizualizaci informačního prostoru (a tedy i Internetu).



Obrázek 2.6: Antarcti.ca – katalog stránek prezentovaný jako shluky v 2D prostoru

Cílem mé diplomové práce je vytvoření modulu pro stahování stránek, navigačního rozhraní, navržení a otestování vhodné komunikační infrastruktury pro moduly.

Kapitola 3

Přehled XML technologií a možností jejich využití

Z předchozího popisu je patrné, že RAINBOW je již poměrně komplikovaný systém, při jehož implementaci se musíme vypořádat s řadou praktických úkolů. Musíme uchovávat zdrojové webové stránky a z nich získané znalosti, zajistit komunikaci různých programových komponent a prezentaci informací uživateli. Abychom nemuseli používat příliš rozdílné nástroje a formáty, rozhodli jsme se v maximální možné míře využít jazyk XML a návazné technologie. Zároveň jsem chtěl ověřit, na co všechno lze XML použít, a jaké jsou jeho limity zejména při rychlosti zpracování.

XML (eXtensible Markup Language) [6] je jednoduchý značkovací jazyk, který umožňuje uchování dat téměř libovolného typu. V odvozených formátech založených na XML můžeme používat vlastní značky a s jejich pomocí zachytit význam a vztah mezi jednotlivými informacemi (viz např. [24]). Syntaxe však vždy vychází z jazyka XML, a proto můžeme používat již existující standardní knihovny pro zpracování XML dokumentů. To nám může ušetřit spoustu práce.

Kromě toho je nad XML vystaveno mnoho dalších užitečných jazyků, které lze v aplikaci využít. Hlavní výhoda spočívá v tom, že se používají známé technologie a že i pro tyto jazyky jsou k dispozici již existující knihovny. V následujícím přehledu se podíváme na některé XML jazyky a technologie, které úzce navazují na XML, a posoudíme možnost jejich využití v RAINBOW.

3.1 XML

Popisovat na tomto místě syntaxi XML by bylo asi zbytečné. Připomeňme si tedy jen, že XML dokument se skládá z pojmenovaných elementů. Každý element přitom může obsahovat text a další vnořené elementy. XML dokument si tak lze představit jako hierarchickou stromovou strukturu. Tato datová struktura je velmi oblíbená a lze do ní uložit v podstatě libovolné informace. Každý element může mít u sebe přítomných ještě několik atributů, což jsou uspořádané dvojice obsahující jméno atributu a jeho hodnotu.

3.1.1 Infoset

Z teoretického hlediska je pro XML velmi důležitý pojem tzv. infosetu – informační množiny (XML Information Set) [10]. Z historických důvodů popisuje standard jazyka XML [6] přímo

syntaxi jazyka, nezabývá se datovým modelem, který XML dokumenty používají. Tento přístup sice umožnil celkem rychlé přijetí XML a jeho širokou podporu v aplikacích, na druhou stranu poněkud kompikuje vývoj dalších standardů navazujících na XML, zejména programátorských rozhraní (API) pro čtení XML dokumentů a dotazovacích jazyků.

Infoset definuje abstraktní datový model pro XML dokumenty. Model přitom vychází ze stromové struktury, kde má každý uzel definovány vlastnosti popisující jeho typ a další navázané uzly (tj. vnořené elementy, atributy apod.). Na rozdíl od samotného standardu XML již infoset neoperuje s jednotlivými znaky v souboru, ale s elementy, atributy, textovým obsahem elementů apod. Je tedy o jednu úroveň abstrakce výše než samotný standard XML, který přesně popisuje syntaxi XML pomocí EBNF.[1]

3.1.2 Jmenné prostory

Pokud potřebujeme do XML ukládat nějaký druh informací, vytvoříme si pro tento účel obvykle vlastní značkovací jazyk, vlastní sadu značek. Rozhodneme se tedy, jak se budou jednotlivé elementy jmenovat a jak mohou být navzájem kombinovány. Tuto definici můžeme zapsat i formálně pomocí nějakého jazyka pro popis schématu dokumentu jako jsou DTD, XML schémata (XML Schema) nebo Relax NG.

Problém ovšem nastane, chceme-li v jednom dokumentu použít více sad značek – může dojít například ke konfliktu názvů elementů apod. Tento problém řeší jmenné prostory [7]. Jedná se o samostatný standard, který velmi úzce doplňuje samostný standard XML. Dokonce se plánuje, že v dalším vydání standardu XML bude vše shrnuto v jednom dokumentu.

Jmenné prostory fungují na jednoduchém principu. Každý element a atribut může být přiřazen ke jmennému prostoru, který je identifikován URI adresou. Pro zkrácení zápisu se pak v XML dokumentu deklarují pro použité jmenné prostory krátké prefixy, které se používají pro přiřazení elementu nebo atributu do určitého jmenného prostoru. Z následující ukázky je patrné, jak se zápis zkrátí, neboť poměrně dlouhé URI je v dokumentu uvedeno jen jednou.

```
<a:x xmlns:a="urn:x-pokus:a" xmlns:b="urn:x-pokus:b">
  <a:y>Obsah elementu Y ve jmenném prostoru urn:x-pokus:a</a:y>
  <b:y>Obsah elementu Y ve jmenném prostoru urn:x-pokus:b</b:y>
</a:x>
```

3.1.3 Jazyky pro popis schématu dokumentů

XML nám sice umožňuje libovolné pojmenování elementů, ale pro většinu praktických aplikací není přílišná volnost žádoucí. Pro každou třídu XML dokumentů proto můžeme pomocí speciálního jazyka přesně formálně definovat, jaké elementy lze v dokumentu použít, jaké mohou mít atributy a obsah. Těmto jazykům se říká jazyky pro definování *schématu dokumentu*.

Historicky nejstarším a nejznámější je bezesporu definice typu dokumentu – DTD (Document Type Definition), která vznikla již pro jazyk SGML a byla převzata i do XML.

DTD pro současné aplikace v mnoha ohledech nedostačují. Jejich asi největším problémem je špatná podpora jmenných prostorů a datových typů. Postupem času proto vznikly i další jazyky pro popis schématu, které nedostatky DTD odstraňují. Nejpoužívanější jsou

[1] Extended Backus-Naur Form (EBNF) je často používaný způsob zápisu bezkontextové gramatiky.

dnes XML schémata [14], která jsou standardem W3C. Jedná se o jazyk postavený na definici datových typů, který nabízí mnoho funkcí – včetně definovaní vlastních datových typů, referenční integrity a integritních omezení. Problém XML schémat je v tom, že se jedná o poměrně složitý standard, kde jde jednu věc dělat mnoha různými způsoby. Nicméně velké IT firmy včetně Microsoftu XML schémata podporují (Microsoft na nich ostatně založil kompletní práci s relačními daty v .NET Frameworku).

Alternativou k XML schématům je jazyk Relax NG [8], který je podobně jako DTD postaven na definici gramatiky a pro většinu použití je tedy mnohem jednodušší než XML schémata. Lze v něm navíc určit i datové typy. V současné době probíhá standardizační proces na půdě organizace ISO [18].

Pokud máme k dokumentu schéma, můžeme během čtení kontrolovat, zda dokument schématu vyhovuje. Provádíme pak tzv. *validaci*. Výhodou tohoto přístupu je především zjednodušení aplikací – ty již nemusí kontrolovat tolik chybových stavů ve vstupních datech, mnoho chyb odhalí již validace.

Validace přináší ještě jeden důležitý koncept, který využívají především dotazovací jazyky nad XML. Ke každému XML dokumentu existuje jeho abstraktní reprezentace v podobě infosetu. Pokud máme k dokumentu schéma, můžeme podle něj jednotlivým uzlům infosetu přiřadit datové typy – získáme tak tzv. PSVI (Post-Schema Validation Infoset) – otypovaný infoset. V dokumentu již nejsou všechna data chápána jako textové řetězce, ale jako konkrétní datové typy – čísla, datumy, řetězce, měnové údaje apod. Nad PSVI již můžeme budovat dotazovací jazyk, který dokáže jednoduše provádět operace jako řazení či výpočty agregačních funkcí, jež jsou závislé na datovém typu zpracovávaných údajů.

3.1.4 Parsery a programátorská rozhraní

Pokud chceme s XML dokumenty pracovat v nějaké aplikaci, nemusíme si sami psát analyzátor XML, ale může využít některý z mnoha již existujících parserů. Parser[2] je program nebo programátorská knihovna, jež čte XML dokument ze souboru (nebo z jiného zdroje – např. z webového serveru přes HTTP protokol) a stará se o nízkoúrovňovou syntaktickou analýzu XML dokumentu – za menšítkem (<) očekává název elementu, extrahuje hodnotu atributu uzavřeného v uvozovkách nebo apostrofech apod. Přes programátorské rozhraní (API) pak parser nabízí abstraktní model XML dokumentu – infoset.

Samotný XML dokument resp. jeho infoset nám může být nabízen několika způsoby a historicky se proto vyvinulo několik různých API, které se hodí pro různé aplikace.

První parsery nabízely rozhraní řízené událostmi. Parser postupně četl XML dokument a volal naše funkce pro obsluhu důležitých událostí, jako začátek a konec elementu, textový obsah elementu apod. Funkci byly navíc předány další důležité parametry – například název elementu, seznam atributů apod.

Událostmi řízené zpracování XML dokumentu má dvě velké výhody – je rychlé a má malé paměťové nároky. V aplikacích, kde je hlavní důraz kladen na rychlost, se proto tento přístup používá nejčastěji. Nevýhodou je naopak nutnost zpracovat XML dokument během jednoho sekvenčního průchodu. Zejména začínající programátoři mají s touto technikou problémy, neboť důležité informace si programátor musí pamatovat ve vlastních stavových proměnných apod.

[2]Správně bych měl používat pojem „parser XML“, protože pojem parser obecně označuje libovolný syntaktický analyzátor, ne jen ten specializovaný na XML. Pro snazší čitelnost textu přívlastek XML neuvádím.

Asi nejznámější rozhraní používající událostmi řízený přístup je SAX (Simple API for XML) [31]. Toto rozhraní vzniklo velmi rychle jako výsledek společného úsilí několika vývojářů e-mailové konference xml-dev. Původně byl SAX navržen pro Javu, ale existují jeho implementace pro mnoho dalších jazyků. V současné době se již používá novější verze rozhraní SAX2, která podporuje jmenné prostory a některé další užitečné vlastnosti.

Pro práci programátora je mnohem pohodlnější, když může kdykoliv přistupovat k libovolné části XML dokumentu. Aby to bylo možné, je nutné načíst celý dokument do vhodné paměťové struktury. XML dokumenty a jejich strukturu lze velmi přirozeně modelovat pomocí stromu. Nejznámější rozhraní, které pracuje se stromovou reprezentací dokumentu, je DOM (Document Object Model) [19]. Toto rozhraní vytvořilo W3C konsorcium a je zcela nezávislé na použitém jazyku. XML dokument je zpřístupněn pomocí objektů, které zastupují jednotlivé důležité prvky – elementy, atributy, textový obsah, komentáře apod. Každý objekt odpovídá jednomu uzlu ve stromu XML dokumentu a nabízí metody pro zjištění svého typu a hodnoty, svých dětí a rodičů.

Strom dokumentu můžeme procházet libovolně a opakovaně. Díky tomu je zpracování XML dokumentu velmi jednoduché. Zaplatíme za to nižší rychlostí a velkou paměťovou náročností. Při načítání XML dokumentu do paměti musíme počítat s tím, že dokument v paměti zabere v závislosti na použité implementaci dvakrát až desetkrát více prostoru než v souboru. U souborů s hodně elementy a atributy je navíc načtení dokumentu dost pomalé, protože se pro každý element a atribut vytváří v paměti objekt.

Na rozdíl od rozhraní SAX sloužícího pouze pro čtení XML dokumentů, umožňuje DOM s reprezentací XML dokumentu v paměti manipulovat, dokonce můžeme vytvořit nový XML dokument přímo v paměti. Je poněkud zarážející, že současná verze DOMu nenabízí standardní metody pro načtení a uložení XML dokumentu, takže každý parser je musí řešit po svém. Tento nedostatek odstraňuje až právě připravovaná verze DOM3.

Rozhraní DOM má za sebou poměrně dlouhou historii sahající do poloviny 90. let minulého století. O DOM se poprvé hovořilo v souvislosti s JavaScriptem a jazykem HTML. JavaScript mohl přes rozhraní DOM pracovat s důležitými informacemi na stránce zobrazené ve webovém prohlížeči – přístupné byly např. obrázky, formuláře a rámy. Další rozvoj DOM vyústil až v dynamické HTML, které umožňuje vytvářet skutečně interaktivní stránky. Velkým problémem však byla (a bohužel stáje ještě trochu je) nekompatibilita DOM rozhraní v jednotlivých prohlížečích. W3C konsorcium proto toto rozhraní standardizovalo a vznikl tak DOM1, který ve skutečnosti obsahuje dvě velmi odlišná rozhraní. Jedno pro práci s HTML dokumenty a druhé pro zpracování XML – tzv. XML Core. Tato část rozhraní je pak podporována DOM XML parsery. Další verze rozhraní DOM2 přinesla zejména podporu jmenných prostorů. Navíc byla přidána další rozhraní, která umožnila mj. snazší průchod stromem dokumentu. Do té doby bylo nutné pro průchod stromem využívat nestandardní třídy (tzv. treewalkery) nebo si psát vlastní kód, který obvykle rekurzivně procházel strom.

DOM a SAX poskytují velmi komplexní podporu pro čtení XML dokumentů. Postupem času se však ukázalo, že pro typické úlohy je jejich použití zbytečně komplikované. Během necelých posledních dvou let se proto stále častěji objevují nová rozhraní nebo rozšíření těch stávajících, která mají jeden společný cíl – usnadnit programátorům práci s XML dokumenty. Autoři těchto nových přístupů mají na mysli zejména zkrácení doby nutné na naučení práce s novým rozhraním a na zkrácení délky kódu nutného pro provedení určité operace (což je ekvivalentní se zkrácením doby vývoje aplikace).

Po počátečním nadšení z rozhraní DOM většina vývojářů zjistí, že je poměrně pracné napsat kód, který zpracuje určité specifické části dokumentu. Přišlo se proto s možností položit nad DOM stromem jednoduchý dotaz, který vybere jen určité uzly stromu – tedy

určité elementy, atributy či textové uzly. Jasným kandidátem na takový dotazovací jazyk je XPath (XML Path Language), který se používá v mnoha dalších XML standardech (XSLT, XML schémata, XPointer aj.). XPath umožňuje zadávat jednoduché dotazy, podobné cestě v adresářové struktuře disku, které vybírají části ze struktury XML dokumentu.

Zjednodušení se dočkaly i parsery, které pracují nad XML dokumentem sekvenčně. O parserech podporujících rozhraní SAX se někdy říká, že jsou to tzv. push parsery – tlačí do aplikace proud událostí odpovídající jednotlivým prvkům XML dokumentu. Tento na událostech postavený přístup není všem programátorům vlastní, a tak vznikly pull parsery. Mají všechny přednosti událostmi řízených parserů – rychlost a paměťovou nenáročnost – a navíc se s nimi velmi jednoduše pracuje. Programátor si podle potřeby říká o další a další části dokumentu a parser mu je postupně předává, dokud nedorazí na konec dokumentu.

3.1.5 Využití XML při ukládání a zpracování dokumentů

Pro potřeby systému RAINBOW musíme načítat webové stránky a ty pak analyzovat různými metodami. Pro úspěšnou analýzu dokumentu potřebujeme znát i strukturu webových stránek. To by teoreticky neměl být problém, protože HTML stránky by měly odpovídat standardu SGML [23] a pro jejich snadné čtení by tedy mělo jít využít SGML parser, který nám zpřístupní i informace o struktuře dokumentu.

Většina SGML parserů je ovšem pro naše účely příliš těžkopádná a navíc většina současných webových stránek standardu HTML (a tím pádem ani SGML) nevyhovuje, protože obsahuje spoustu syntaktických chyb. Poměrně snadno však můžeme pomocí programu HTML-Tidy[3] odstranit ze stávajících HTML stránek syntaktické chyby a převést je do formátu XML. Tím získáme dokumenty, které půjde snadno číst pomocí velkého množství dostupných XML parserů a navíc budeme mít zpřístupněnou strukturu dokumentů.

Tento přístup – převod HTML stránek do XML – se nám vyplatí i z dlouhodobějšího hlediska, protože se na webu začínají postupně objevovat stránky, které jsou zapsány přímo v XML nebo v XHTML, což je nástupce jazyka HTML založený na XML syntaxi. Tím, že analytické moduly budou již od počátku připraveny na zpracování XML, nebude v budoucnu problém se zpracováním nové generace webu založené na XML.

3.2 Odkazy mezi dokumenty

3.2.1 XPointer – identifikace fragmentů dokumentu

XPointer [12] je jednoduchý jazyk, který umožňuje jednoznačnou identifikaci libovolné části dokumentu. Fakticky se jedná o rozšířený jazyk XPath. XPointer mohou s výhodou využívat jednotlivé analytické moduly, které si pro zpracování budou předávat části dokumentů. Nebude potřeba předávat celé části dokumentů, ale jen poměrně krátkou adresu dokumentu doplněnou o XPointer identifikující konkrétní část dokumentu. Například první tabulku obsaženou v XML dokumentu pojmenovaném `dokument.xml`, můžeme identifikovat pomocí následujícího XPointeru:

```
dokument.xml#xpointer(table[1])
```

XPointer můžeme používat pro adresování, protože všechny HTML stránky převádíme do XML.

[3] `http://www.w3.org/Status.html`

3.2.2 XLink a RDF – popis vazeb mezi stránkami

Pro účely RAINBOW nás zajímají i vztahy mezi jednotlivými stránkami – např. asociace podle obsahové podobnosti nebo topologického uspořádání. Reprezentovat vztahy mezi stránkami můžeme samozřejmě mnoha způsoby. Jedna z možností je i využití jazyka XLink (XML Linking Language) [11], který slouží k zápisu odkazů mezi dokumenty. Oproti jiným jazykům, lze v XLinku vytvářet i vícesměrné odkazy, které propojují více stránek. Dalším důležitým rysem XLinku je možnost přiřazení typu jednotlivým odkazům. Samotné odkazy tak mohou přímo nést nějakou sémantickou informaci. Následující ukázka obsahuje asociace pro stránku zapsané pomocí XLinku:

```
<xlink:extended xmlns:xlink="http://www.w3.org/1999/xlink"
                xlink:type="extended">
 <xlink:locator xlink:href="http://www.kosek.cz/clanky/wap/wap.html"
                xlink:role="urn:x-rainbow:linkroles:basedocument"
                xlink:title="Web v příštím tisíciletí"
                xlink:type="locator"/>
 <xlink:locator xlink:href="http://www.kosek.cz/clanky/index.html"
                xlink:role="urn:x-rainbow:linkroles:toc"
                xlink:title="Seznam článků"
                xlink:type="locator"/>
 <xlink:locator xlink:href=
                      "http://www.kosek.cz/clanky/wap/sluzby.html"
                xlink:role="urn:x-rainbow:linkroles:next"
                xlink:title="Jak budou služby dostupné uživateli"
                xlink:type="locator"/>
 <xlink:locator xlink:href="http://www.kosek.cz/hledej.html"
                xlink:role="urn:x-rainbow:linkroles:search"
                xlink:title="Fulltextové prohledávání"
                xlink:type="locator"/>
</xlink:extended>
```

K podobnému účelu můžeme kromě XLinku použít i RDF (Resource Description Format) [27]. RDF je formát založený na XML, který umožňuje připojování libovolných sémantických metadat k libovolnému dokumentu (informačnímu zdroji). RDF dokument umožňuje zapsat skupinu výroků, kdy je subjektu (informačnímu zdroji) přiřazena jedna nebo více dvojic vlastností a jejich hodnot. Informační zdroje, vlastnosti i jejich hodnoty jsou identifikovány URI adresou. Jedinou výjimku tvoří hodnota vlastnosti, která může mít i typ textového řetězce.

Výše uvedonou informaci o asociovaných stránkách můžeme klidně zapsat i v RDF.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/TR/REC-rdf-syntax#"
         xmlns:rainbow="uri:x-rainbow:roles">
 <rdf:Description about="http://www.kosek.cz/clanky/wap/wap.html">
  <rainbow:ToC>http://www.kosek.cz/clanky/index.html</rainbow:ToC>
  <rainbow:Next>http://www.kosek.cz/clanky/wap/sluzby.html
  </rainbow:Next>
  <rainbow:Search>http://www.kosek.cz/hledej.html</rainbow:Search>
 </rdf:Description>
</rdf:RDF>
```

Význam dokumentu je přitom celkem jasný. Říká, že k dokumentu `http://www.kosek.cz/clanky/wap/wap.html`, existují stránky, které jsou ve vztahu „obsah“, „další stránka“ a „vyhledávací stránka“, a rovněž říká, jaké to jsou stránky (identifikace pomocí jejich URI adres).

3.3 Komunikace mezi moduly – XML-RPC a SOAP

RAINBOW se skládá z několika modulů, které mezi sebou musí komunikovat. Aby byl celý systém dostatečně škálovatelný, je potřeba, aby byly jednotlivé moduly schopné spolupráce, i když poběží na různých počítačích umístěných v síti. Moduly mohou být navíc implementovány v různých jazycích.

Pro komunikaci bychom samozřejmě mohli využít takové technologie jako CORBA nebo DCOM, ale jejich implementace je poměrně složitá. Existují přitom komunikační protokoly založené na XML, které se pro naše účely výborně hodí a jejich implementace je mnohem jednodušší než u dříve zmíněných technologií.

Mezi nejpoužívanější a nejperspektivnější protokoly pro komunikaci aplikací dnes patří SOAP (Simple Object Access Protocol) [4]. Tento protokol umožňuje předávání zpráv ve formátu XML, jako komunikační protokol se přitom nejčastěji používá protokol HTTP (Hypertext Transfer Protocol). Na podobných základech staví i o něco jednodušší protokol XML-RPC. Podrobnější popis SOAPu je v následující kapitole.

Výhodou SOAPu je to, že staví na technologiích, které jsou podporovány v mnoha jazycích a na mnoho platformách. Umožňuje proto integraci služeb, které pracují v heterogenním a distribuovaném prostředí.

3.4 Uživatelské rozhraní

Pro uživatele se RAINBOW jeví jako navigační klient, který pomocí přídavného okna v prohlížeči zpřístupňuje informace o stránce a funkce usnadňující navigaci. Možností, jak vytvořit takového klienta, je mnoho. Pomineme-li možnost vytvoření rozhraní jako klasického programu využívajícího nějakou knihovnu grafických prvků uživatelského rozhraní, máme i několik možností postavených čistě na XML technologiích.

První možností je vytvoření rozhraní přímo v HTML (případně XHTML) s použitím JavaScriptu. Pro naše účely to však není zrovna nejvhodnější řešení, protože HTML neobsahuje prvky pro snadné vytváření rozbalovacích seznamů, které jsou přirozeným způsobem pro prezentování informací v navigačním asistentovi.

Toto omezení překonává speciální jazyk pro tvorbu formulářů XForms [13]. Problém je v tom, že tento jazyk je zatím jen ve stavu návrhu a neexistují žádné běžně použitelné implementace.

Existují však i speciální jazyky pro deklarativní popis uživatelského rozhraní. Mezi nejznámější patří jazyk XUL (XML-based User Interface Language) [21], který používá prohlížeč Mozilla[4] a od něj odvozený Netscape Navigator 6.

Pomocí XUL můžeme snadno vytvořit systém hierarchických nabídek, který zpřístupní všechny navigační funkce. XUL má tu výhodu, že je snadno přenositelný mezi různými platformami, lze jej použít všude, kde pracuje Mozilla. Uživatelské rozhraní je definováno pomocí pojmů jako dialogové okno, nabídka, položka nabídky, seznam, vstupní pole apod.

[4] `http://www.mozilla.org`

Na jednotlivé prvky pak můžeme navázat volání kódu v JavaScriptu nebo v C++. Toho bude využívat i náš navigační klient. Interaktivní chování aplikace je zajištěno právě pomocí JavaScriptu.

Samotné rozhraní Mozilly je definováno v XUL, proto můžeme jednoduchou modifikací několika souborů změnit vzhled prohlížeče. Snadno tak můžeme do Mozilly přidat další panel s rozhraním RAINBOW. Podrobněji jsou možnosti Mozilly, XUL a vytvoření navigačního rozhraní popsány v kapitole 5 – *Navigační rozhraní*.

Kdybychom kromě navigačního rozhraní chtěli vytvořit i vizualizační rozhraní, můžeme s výhodou použít jazyk SVG (Scalable Vector Graphics) [15]. Tento jazyk opět založený na XML syntaxi umožňuje popsat obrázek složený ze základních grafických objektů jako čáry, texty, oblouky, obdélníky, křivky apod. Přes rozhraní DOM pak můžeme pomocí JavaScriptu s obrázkem manipulovat a vytvořit interaktivní grafické aplikace.

3.5 XSLT – konverze a transformace dokumentů

Pokud v jakémkoliv větším než malém projektu využíváme jazyk XML, určitě budeme dříve či později postaveni před problém konverze mezi XML dokumenty s různými schématy. Konverzi lze samozřejmě implementovat pomocí klasických programů, které budou XML dokumenty číst přes nějaké API jako je SAX nebo DOM. Existuje však mnohem jednodušší možnost – transformační jazyk XSLT.

XSLT (XSL Transformations) [9] je speciální jazyk navržený pro transformaci XML dokumentů. Transformace se řídí pomocí stylu, který obsahuje šablony a může se dotazovat na strukturu vstupního dokumentu. Navíc lze použít cykly, podmínky a další konstrukce, které známe z klasických jazyků. Výsledkem transformace může být buď další XML dokument, HTML stránka nebo libovolný textový soubor.

V systémech jako RAINBOW nalezne XSLT bohaté uplatnění. Současné znalostní systémy málokdy přímo podporují formát XML. Není však problém jakýkoliv XML dokument pomocí vhodného stylu převést do požadovaného formátu – např. do zdrojového kódu pro interpret Prologu, textového souboru pro nějaký produkt pro dolování dat apod.

V budoucnu plánujeme přidání dalších koncových rozhraní pro uživatele. Výsledky dotazů můžeme pomocí stylu konvertovat do požadované podoby – např. do klasické HTML stránky, do XUL pro prohlížeč Mozilla nebo třeba do vektorového grafického formátu SVG.

3.6 Dotazovací jazyky

Jelikož lze do XML ukládat informace, je celkem logické, že vyvstane nutnost později v těchto informacích vyhledávat. V současné době existuje pouze jediný skutečně standardizovaný jazyk pro dotazování – XPath. Umožňuje vytváření dotazů, které se mohou dotazovat na strukturu dokumentu a provádět navigaci po stromové reprezentaci XML dokumentu. XPath však nepodporuje důležité funkce jako seskupování, řazení, složitější agregační funkce a definice struktury výstupního XML.

Pomineme-li pouze návrhy dotazovacích jazyků, které nebyly nikdy standardizovány a široce přijaty (např. XQL a Quilt), existují dnes dva hlavní proudy dotazovacích jazyků určených pro XML. Prvním je snaha o rozšíření SQL o speciální konstrukce podporující práci s XML daty a jejich integraci do objektově-relačního modelu. V rámci standardizační skupiny

pro SQL vznikla skupina SQLX,[5] která pracuje na integraci XML do SQL (jedná se 14. část SQL standardu tzv. SQL/XML). Projekt zahrnuje definici mapování mezi identifikátory a datovými typy SQL a XML schémat, pracuje se na možnosti generování XML přímo jako výsledku SQL dotazu apod.

Zatímco SQL/XML se snaží rozšířit existující jazyk pro práci s relačními a objektově-relačními daty o možnosti XML, na půdě W3C se připravuje dotazovací jazyk speciálně určený pro dotazování pouze nad XML dokumenty. Jazyk by měl v sobě spojit možnosti XPathu a SQL, částečně i s možnostmi XSLT. Jedná se o dotazovací jazyk XQuery (XML Query Language) [2]. Existuje již i několik jeho testovacích implementací.[6]

V našem projektu zatím nepočítáme s použitím komplexních dotazovacích jazyků jako XQuery nebo SQL/XML, naopak XPath bude vzhledem k povaze projektu pravděpodobně využíván ve velké míře.

[5] `http://www.sqlx.org/`

[6] `http://www.w3.org/XML/Query#products`

Kapitola 4

Komunikační infrastruktura

Celý systém RAINBOW se skládá z několika modulů, které jsou schopny provádět dílčí operace – stahovat HTML stránky, analyzovat jejich kód, extrahovat metadata z HTML, lematizovat slovo apod. Tyto moduly spolu musí spolupracovat při analýze stránek i při vyhodnocování dotazů. V této části práce se proto podíváme na způsoby, jakými lze komunikaci vyřešit, a podrobněji popíšeme technologii webových služeb, kterou používá prototypová implementace systému.

4.1 Požadavky systému

Předtím než začneme porovnávat jednotlivé technologie a techniky, které lze použít pro zajištění komunikace mezi moduly, si musíme stanovit požadavky na komunikační infrastrukturu.

4.1.1 Distribuovanost

První prototypová implementace sice pracuje jen na jednom počítači, ale je velmi pravděpodobné, že se v budoucnu systém rozroste a jednotlivé moduly poběží na různých počítačích. Komunikační infrastruktura proto musí umožňovat spolupráci modulů i v distribuovaném internetovém prostředí.

4.1.2 Nezávislost na implementačním prostředí

Činnost jednotlivých modulů je dosti rozdílná – počínaje čistě technickými moduly (stahování stránek) až po moduly implementující různé lingvistické metody a metody expertních systémů. Moduly jsou navíc vyvíjeny různými lidmi. Jednotlivé moduly mohou být napsány v různých programovacích jazycích a mohou pracovat i v různých operačních systémech. Komunikační protokol by proto neměl být svázán s nějakým konkrétním jazykem nebo platformou.

4.1.3 Flexibilita

Jednotlivé moduly mají dost odlišné požadavky na druh přenášených dat. Někdy se přenášejí jen jednoduché skalární hodnoty (čísla, řetězce), někdy celé HTML dokumenty, někdy jen

části HTML dokumentů, můžeme uvažovat i o seznamech či strukturách složených z předchozích hodnot. Komunikační protokol by proto měl umožnit přenášení dat libovolného druhu.

4.1.4 Asynchronnost

Část systému, která bude obstarávat komunikaci s uživatelem – navigační rozhraní – si vystačí se synchronním způsobem komunikace. Uživatel si do prohlížeče načte stránku, tím vyvolá požadavek na zjištění informací o stránce, požadavek se předá modulům a ty vrátí odpověď, kterou navigační rozhraní zobrazí.

Druhá strana systému, která zajišťuje stahování stránek a jejich analýzu, už nemusí nutně pracovat jen synchronně. Můžeme si představit, že se po stažení stránky předá ostatním modulům požadavek k jejich zpracování. Toto zpracování však může proběhnout i za delší dobu a mohlo by proto pro něj být vhodnější použít asynchronní způsob komunikace, kdy jednotlivé moduly nečekají na okamžitý výsledek operace jako v jednoduchém modelu požadavek/odpověď.

Komunikační infrastruktura by proto měla umožňovat i asynchronní komunikaci, s tím, že prototypová implementace systému ji nejspíš zatím nevyužije.

4.2 Možnosti řešení komunikace mezi moduly

V současné době existuje mnoho technologií a postupů, jak po síti propojit softwarové moduly. V následujícím textu stručně charakterizujeme nejpoužívanější metody a popíšeme jejich výhody a nevýhody pro použití v našem projektu. Rozdělení do kategorií je přitom potřeba brát s jistou rezervou, jednotlivé technologie jsou si v mnohém blízké a často se překrývají.

4.2.1 Vzdálené volání procedur

Vzdálené volání procedur (RPC – Remote Procedure Call) je jednou z nejstarších metod pro komunikaci programů na dálku. RPC nabízí mechanismus, jak z jednoho programu volat funkci umístěnou na vzdáleném systému. Existuje několik protokolů implementujících RPC – mezi nejznámější patří asi Distributed Computing Environment (DCE).[1] RPC protokol definuje způsob jak parametry a výsledek volané funkce převádět do formátu, ve kterém se RPC požadavky posílají po síti.

Pro naše potřeby je RPC příliš jednoduché. Jednak RPC podporuje jen základní datové typy, přenos strukturovaných dat jako jsou např. fragmenty XML není přímo podporován. Navíc RPC předpokládá jednoduchý model požadavek/odpověď, který nelze v případě potřeby změnit na asynchronní model.

4.2.2 Distribuované objektové technologie

Mezi nejznámější distribuované technologie patří bezesporu CORBA, DCOM a RMI. Ty umožňují volání vzdálených objektů, tak jako bychom je měli přímo k dispozici. Využívá se přitom mechanismus zástupných (proxy) objektů. CORBA je standard vzniklý v rámci

[1] `http://www.opengroup.org/dce/`

skupiny OMG a v současné době existuje mapování z jazykově nezávislého popisu rozhraní objektů IDL (Interface Definition Language) do většiny používaných jazyků. DCOM je proprietární technologie Microsoftu, ale jeho podpora je dostupná rovněž v širokém spektru jazyků. RMI (Remote Method Invocation) je technologie použitelná pouze v jazyce Java.

Ce se týče podpory různých platforem je na tom CORBA velmi dobře. Modul, který zajišťuje komunikaci – ORB (Object Request Broker) – je dostupný pro několik desítek platforem. Jednotlivé ORB mezi sebou nejčastěji komunikují pomocí protokolu IIOP (Internet Inter-ORB Protocol), který umožňuje distribuovaný systém provozovat přímo v internetové síťové infrastruktuře.

Oproti tomu technologie DCOM vznikla primárně pro Windows a její portace na další platformy není nijak excelentní. Existuje sice několik implementací DCOMu pro unixové prostředí, ale většinou se jedná o komerční produkty, což DCOM pro naše potřeby předem diskvalifikuje.

Využití technologie CORBA by pro náš projekt bylo samozřejmě možné. Na druhou stranu je potřeba říci, že psaní CORBA komponent by bylo pro naše účely příliš složité. CORBA nabízí i mnoho funkcí, které v současné době nepotřebujeme – např. transakce a vysokou bezpečnost.

4.2.3 Messaging

Messaging je způsob komunikace mezi softwarovými komponentami nebo softwarovými aplikacemi založený na výměně zpráv [38]. O samotné doručování a směrování zpráv mezi jednotlivými aplikacemi se stará tzv. MOM (Message Oriented Middleware). Jednotlivé aplikace využívají služby MOM přes aplikační rozhraní. Většina producentů MOM systému nabízí vlastní API, které je dostupné jen pro omezený počet platforem. Největší je dnes segment MOM systémů pro Javu, pro kterou existuje standardní API pro využívání messagingových služeb – JMS (Java Message Service).

MOM systémy v té nejjednodušší podobě zaručují doručení zprávy cílové aplikaci. Komunikace přitom probíhá pomocí virtuálních kanálů (tzv. destinací). Aplikace může zprávy posílat na určitou destinaci, k jejímu odběru se může přihlásit jiná aplikace (může jich být dokonce více). Systémy mají obvykle systém pro samotné doručování a směrování zpráv oddělen od aplikačního rozhraní, takže se o tyto detaily nemusíme příliš starat. Kromě těchto základních funkcí nabízejí MOM systémy i pokročilé funkce jako transakce (několik operací/zpráv je považováno za nedělitelnou operaci) a bezpečnost (šifrování přenášených dat, autentizace a autorizace). MOM systémy jsou z funkčního hlediska hodně vyspělé, většinou se však jedná o komerční produkty, což se pro náš akademický projekt nehodí.

Vzhledem k tomu, že v celém projektu RAINBOW se používá mnoho metod znalostního inženýrství, hodně jsme zvažovali použití jazyka a protokolu KQML (Knowledge Query and Manipulation Language),[2] který je de-facto standardem pro komunikaci v distribuovaných znalostních systémech.

KQML je velmi obecný a flexibilní protokol navržený speciálně pro komunikaci v multiagentních systémech. Obsahuje proto mnoho tzv. performativ, která kromě samotné komunikace obstarávají i velké množství podpůrných funkcí – dynamické objevování agentů v systému, přesměrování požadavků, zjištění funkcí daného agenta apod. V důsledku toho je úplná implementace KQML poměrně složitá a již hotové implementace pokrývají jen velice úzký okruh jazyků – Java, C/C++ a Lisp.

[2] `http://www.cs.umbc.edu/kqml/`

Během práce na projektu jsme zjišťovali možnosti využití již hotové knihovny pro tvorbu multiagentních systémů založených na KQML, kterou vytvořila Gerstnerova laboratoř (FEL ČVUT) pro svůj systém ProPlanT.[3] Problémem knihovny však byla její malá propustnost. Zvládala obsluhu jen několika zpráv za sekundu, což pro naše účely bylo opravdu málo.

V posledních dvou letech vyvolala velkou pozornost koncepce tzv. *webových služeb* (web-services). Z technologického hlediska nepřinášejí webové služby nic převratně nového a za technologiemi jako je MOM dokonce silně pokulhávají svou funkčností a rychlostí. Velkou výhodou webových služeb je však jejich schopnost velmi levně integrovat aplikace napsané v různých jazycích a bežících na různých platformách. Je toho dosaženo tím, že aplikace spolu komunikují posíláním XML zpráv pomocí HTTP protokolu. Podpora HTTP a XML je široce dostupná, a proto jsou i webové služby dostupné v podstatě všude.

Základem webových služeb je protokol SOAP (Simple Object Access Protocol) [4], který definuje strukturu zpráv a způsob, jakým se do XML kódují běžné datové typy. Nejčastěji se dnes v SOAPu používá synchronní komunikace. Klientská aplikace pošle v XML požadavek webové službě (serveru), ta dekóduje požadavek v XML, zavolá příslušný kód, a výsledek opět ve formátu XML zašle zpět klientovi.

Popis rozhraní webové služby (kde je služba k dispozici, jaké má vstupní/výstupní parametry) je možné formálně specifikovat pomocí popisu v jazyce WSDL (Web Services Description Language) [22]. Pro mnoho jazyků existují knihovny, které jsou z WSDL popisu schopné automaticky generovat klientský kód, který umožní pohodlné volání webové služby stejně jako by to byla lokálně dostupná funkce nebo metoda.

Právě možnost použití webových služeb napříč platformami a dostupnost knihoven usnadňujících implementaci byla asi hlavním důvodem, proč jsme se nakonec rozhodli v projektu RAINBOW použít jako komunikační infrastrukturu právě protokol SOAP. Navíc je protokol vystaven nad XML a můžeme si na reálném projektu vyzkoušet praktickou použitelnost nově vznikající technologie. Podrobnější popis webových služeb a SOAPu naleznete v další části této kapitoly.

4.3 Význam ontologie při komunikaci

Ontologie [42], [37] je explicitně specifikovaná konceptualizace. Ontologie popisuje určitou oblast zájmu (doménu) formálním způsobem – definuje třídy objektů, které se v dané oblasti nacházejí, a vztahy, které mezi nimi mohou existovat. Význam ontologie spočívá především v usnadnění komunikace mezi lidmi, zlepšení spolupráce softwarových systémů a ve zdokonalení systémového inženýrství. Ontologie do všech těchto oblastí přinášejí možnost sjednocení pohledu, udržení konzistence a jednoznačnosti.

Z dlouhodobého hlediska se v systému RAINBOW počítá s vytvořením ontologie, která by pokrývala doménu našeho problému – především by umožnila klasifikaci objektů, se kterými pracujeme, jako jsou HTML stránka, HTML element, text, věta, URL adresa apod. Při větším počtu modulů, které by si vyměňovaly složitější informace, je ontologie velmi důležitá, protože usnadní a zjednoznační popis dat vyměňovaných mezi moduly.

Kdyby se systém rozrostl do větších rozměrů a skládal by se z velkého množství nezávislých modulů, které by mohly být do systému přidávány i dynamicky, bylo by použití ontologie (nebo jiného obdobného nástroje) v podstatě nevyhnutelné. V dynamicky se měnícím systému je nutné jednotlivé zprávy posílané mezi moduly a jejich obsah přesně sémanticky

[3] `http://cyber.felk.cvut.cz/gerstner/dai/proplant/index.html`

označit. Jen tak může být zaručeno, že bude komunikace probíhat mezi správnými moduly, a že data nebudou chybně interpretována.

V současné době je množství modulů v systému RAINBOW minimální a navíc jsou moduly velmi jednoduché. Celý systém je statický a neměnný. Použití formální ontologie by v tuto chvíli nepřineslo žádný podstatný užitek. Samozřejmě, že i tak jsou moduly vyvíjeny na základě neformální ontologie, kterou autoři všech modulů nosí ve své hlavě. V určité fázi projektu RAINBOW se počítá s přidáním sémantických rolí odkazujících na ontologii do všech zpráv.

4.4 Využití webových služeb a protokolu SOAP při komunikaci

Jak jsme již naznačili, webové služby umožňují jednoduchou komunikaci mezi aplikacemi ve velmi heterogenním prostředí, protože komunikace je založena na platformě nezávislých standardech – především na jazyce XML a protokolu HTTP. Aplikace si mezi sebou posílají XML zprávy, které přenášejí dotazy a odpovědi jednotlivých aplikací. Celá infrastruktura webových služeb je založena na třech základních technologiích [39]:

- SOAP (Simple Object Access Protocol) – protokol používaný pro komunikaci;
- WSDL (Web Services Description Language) – standardní formát pro popis rozhraní webové služby;
- UDDI (Universal Description, Discovery and Integration) – standardní mechanismus umožňující registraci a vyhledávání webových služeb.

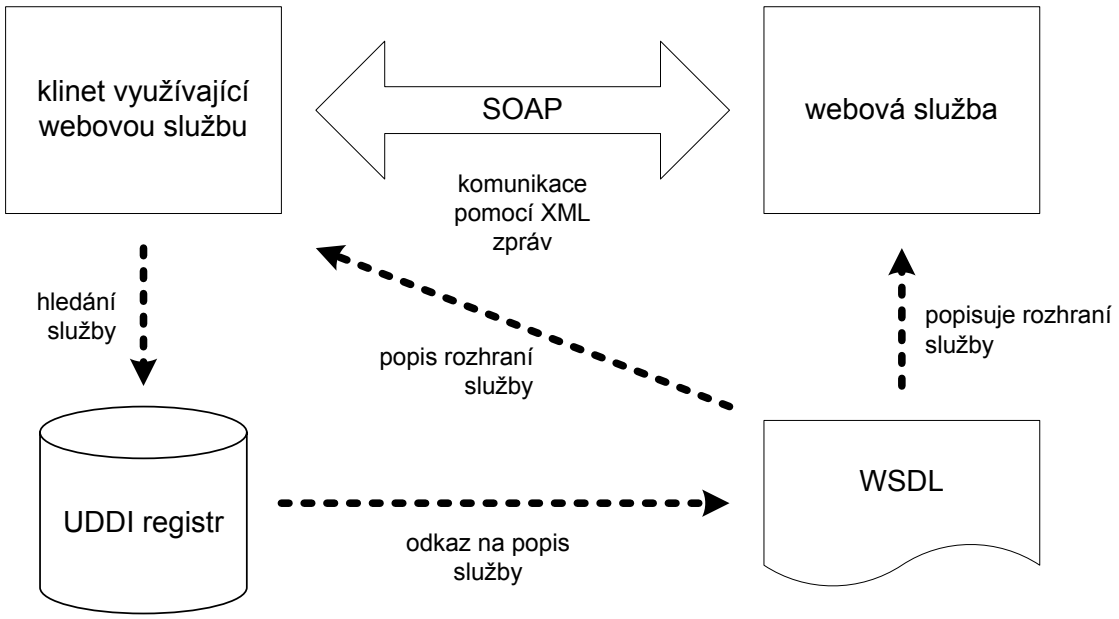
Vzájemné vztahy mezi těmito třemi technologiemi jsou zachycené na obrázku 4.1. Ke každé webové službě by měl být k dispozici její formální popis v jazyce WSDL. Z tohoto popisu již jde automaticky vygenerovat soapový požadavek. Ve větších systémech nebo přímo v otevřeném prostředí Internetu se popis služby může zaregistrovat do UDDI registru. Ten slouží jako jakýsi telefonní seznam („zlaté stránky“), který umožňuje vyhledávání služeb s určitými parametry.

Klient, který chce využít webovou službu, získá buď přes UDDI, nebo přímo její popis. Z něj je jasné, jakou strukturu má mít soapová zpráva a kam se má webové službě poslat, aby ji rozpoznala.

4.4.1 SOAP

SOAP je protokol pro posílání zpráv XML a je základem webových služeb. Ostatní standardy jako WSDL a UDDI vznikly až později po uvedení SOAPu a jen dále rozšiřují jeho možnosti a snadnost použití. SOAP umožňuje zaslání XML zprávy mezi dvěma aplikacemi a pracuje tedy na principu peer-to-peer. Zpráva je jednosměrný přenos informace od odesílatele k příjemci, ale díky kombinování několika zpráv můžeme pomocí SOAPu snadno implementovat běžné komunikační scénáře.

Nejčastěji se SOAP používá jako náhrada vzdáleného volání procedur (RPC), tedy v modelu požadavek/odpověď. Jedna aplikace pošle v XML zprávě požadavek druhé aplikaci, tak požadavek obslouží a výsledek zašle jako druhou zprávu zpět původnímu iniciátorovi komunikace. V tomto případě bývá webová služba vyvolána webovým serverem, který čeká



Obrázek 4.1: Vztah tří základních technologií (SOAP, WSDL a UDDI) webových služeb

na požadavky klientů a v okamžiku, kdy přes HTTP přijde soapová zpráva, spustí webovou službu a předá jí požadavek. Výsledek služby je pak předán zpět klientovi jako odpověď.

První verze (1.0) protokolu SOAP vznikla na konci roku 1999 jako výsledek společné práce firem DevelopMentor, Microsoft a UserLand, které chtěly vytvořit protokol pro vzdálené volání procedur (RPC) založený na XML [3]. Protokol navazoval na o rok mladší, jednodušší a méně flexibilní protokol XML-RPC.[4] V průběhu roku 2000 se k podpoře přihlásila i firma IBM a nová verze SOAPu 1.1 byla zaslána W3C konsorciu [4]. Verze SOAPu 1.1 je dnes nejpoužívanější a v diplomové práci se budeme zabývat právě jí. Na půdě W3C konsorcia nyní probíhá práce na uvolnění prvního skutečného standardu SOAP 1.2 v rámci pracovní skupiny pro XML protokol.[5] Pracovní verze specifikace je dostupná v [17].

4.4.1.1 Struktura zprávy

Zpráva v SOAPu je jednoduchý XML dokument, který má kořenový element `Envelope`. V této obálce jsou pak uzavřeny dva elementy `Header` (hlavička) a `Body` (tělo). Hlavička je přitom nepovinná a používá se pro přenos pomocných informací pro zpracování zprávy – například identifikaci uživatele, autentizační informace (jméno, heslo) apod.

O to nejdůležitější se stará tělo zprávy, v němž se přenášejí informace identifikující volanou službu a předávané parametry, resp. návratové hodnoty služby. SOAP používá jmenné prostory pro identifikování jednotlivých částí XML zprávy. Obálka, hlavičky a tělo zprávy patří do jmenného prostoru `http://schemas.xmlsoap.org/soap/envelope/`.

Příklad 4.1: Ukázka jednoduché zprávy SOAP

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
```

[4] `http://www.xmlrpc.com/`
[5] `http://www.w3.org/2002/ws/`

```
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="urn:x-example:services:StockQuote">
      <symbol>MOT</symbol>
    </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Příklad 4.1 ukazuje velmi jednoduchý SOAP požadavek na zjištění posledního známého kurzu akcie s kódem `MOT`. Ukázková zpráva pro jednoduchost neobsahuje hlavičku, ale pouze tělo. V něm je požadavek na vyvolání vzdálené funkce `GetLastTradePrice` s parametrem pojmenovaným `symbol` s hodnotou `MOT` (kód akcií firmy Motorola).

Jak by mohla vypadat XML zpráva s výsledkem přibližuje příklad 4.2. Oba dva příklady jsou drobně modifikované ukázky přímo z [4].

Příklad 4.2: Ukázka zprávy s odpovědí

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
 <SOAP-ENV:Body>
  <m:GetLastTradePriceResponse
   xmlns:m="urn:x-example:services:StockQuote">
   <Price>14.5</Price>
  </m:GetLastTradePriceResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

4.4.1.2 Kódování dat

V ukázkách SOAP zpráv je použit ještě jeden jmenný prostor – `http://schemas.xmlsoap.org/soap/encoding/`. Ten identifikuje způsob kódování přenášených dat do XML. Se SOAPem můžeme použít libovolný způsob serializace, standard SOAPu jeden rovnou definuje. Standardní serializace je schopná do XML převést graf obsahující otypované objekty. V praxi se nejčastěji používají běžné skalární datové typy (jako čísla, řetězce apod.), které definují XML schémata. Navíc soapové kódování definuje způsob serializace složených datových typů – seznamů skalárních hodnot (polí) a struktur. Lze serializovat i reference na objekty.

Obecně platí, že se hodnoty ukládají vždy jako obsah elementů, jedna hodnota do jednoho elementu. Datové typy mohou být definovány buď v externím XML schématu nebo přímo v XML zprávě. Druhý způsob je jednodušší a tedy i používanější. Např. mé osobní údaje by mohly být zakódovány do XML pro potřeby SOAPu následujícím způsobem:

```
<jméno xsi:type="xsd:string">Jirka Kosek</jméno>
<email xsi:type="xsd:string">jirka@kosek.cz</email>
<věk xsi:type="xsd:int">26</věk>
```

Předpokládáme přitom, že prefixy `xsi` a `xsd` jsou svázány se jmenným prostorem `http://www.w3.org/2001/XMLSchema-instance`, resp. `http://www.w3.org/2001/XMLSchema`. Odpovídající samostatné XML schéma by pak vypadalo následovně:

```
<xsd:element name="jméno" type="xsd:string"/>
<xsl:element name="email" type="xsd:string"/>
<xsl:element name="věk" type="xsd:int"/>
```

4.4.1.3 Transportní mechanismy

Jelikož se dnes SOAP typicky používá pro RPC volání, je celkem přirozené, že se pro přenos požadavku/odpovědi nejčastěji používá protokol HTTP (HyperText Transfer Protocol) [16]. Důvodem je zejména široká podpora HTTP v různých aplikacích. Navíc webovou službu lze nahrát přímo na běžný webový server, jenž slouží jako „dispečer", který jednotlivé požadavky předává odpovídající webové službě ke zpracování. Výhoda použití HTTP také spočívá v tom, že stávající síťová infrastruktura, zvláště ve firemní sféře, dovoluje v podstatě neomezenou komunikaci na portu vyhrazeném pro HTTP (TCP port 80). Webové služby je možné používat bez nutnosti zásahu do konfigurace aktivních síťových prvků jako jsou firewally. Při použití technologií DCOM nebo CORBA je potřeba povolit komunikaci na portech, které používají příslušné přenosové protokoly (např. IIOP pro CORBA).

SOAP požadavek se zasílá v těle HTTP požadavku. Používá se přitom metoda POST, která dovoluje posílat data v těle HTTP požadavku. Požadavek musí obsahovat HTTP hlavičku `SOAPAction`, která identifikuje SOAP požadavek. Tuto hlavičku mohou používat jednak firewally k filtrování požadavků a jednak může obsahovat URI s identifikací služby, která se má vyvolat. Pokud je obsahem hlavičky prázdný řetězec, služba ke spuštění je identifikována přímo adresou, na kterou směřuje požadavek.

Příklad 4.3: SOAP požadavek zaslaný přes HTTP

```
POST /StockQuote HTTP/1.1
Content-Type: text/xml; charset=utf-8
Content-Length: nnn
SOAPAction: ""

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 <SOAP-ENV:Body>
  <m:GetLastTradePrice xmlns:m="urn:x-example:services:StockQuote">
   <symbol>MOT</symbol>
  </m:GetLastTradePrice>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Odpověď na požadavek nemá při přenosu pomocí HTTP žádné přídavné informace. Obsah odpovědi musí být identifikován jako XML dokument pomocí příslušného MIME typu `text/xml` v hlavičce `Content-Type`.

Příklad 4.4: SOAP odpověď přenášená pomocí HTTP

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: nnn
```

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
 <SOAP-ENV:Body>
  <m:GetLastTradePriceResponse
   xmlns:m="urn:x-example:services:StockQuote">
   <Price>14.5</Price>
  </m:GetLastTradePriceResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Další možností, jak posílat SOAP zprávy pomocí HTTP, je využití rozšiřujícího systému pro HTTP popsaného v [32]. Požadavek a odpověď se pak drobně liší v hlavičkách, jak ukazují příklady 4.5 a 4.6.

Příklad 4.5: SOAP požadavek zaslaný přes rozšíření HTTP

```
M-POST /StockQuote HTTP/1.1
Man: "http://schemas.xmlsoap.org/soap/envelope"; ns=42
Content-Type: text/xml; charset=utf-8
Content-Length: nnn
42-SOAPAction: ""

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
   <m:GetLastTradePrice xmlns:m="urn:x-example:services:StockQuote">
    <symbol>MOT</symbol>
   </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Příklad 4.6: SOAP odpověď přenášená pomocí rozšíření HTTP

```
HTTP/1.1 200 OK
Ext:
Content-Type: text/xml; charset=utf-8
Content-Length: nnn

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
 <SOAP-ENV:Body>
  <m:GetLastTradePriceResponse
   xmlns:m="urn:x-example:services:StockQuote">
   <Price>14.5</Price>
```

```
  </m:GetLastTradePriceResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Jednotlivé implementace webových služeb podporují i další přenosové mechanismy. Patří mezi ně například přenos pomocí e-mailových zpráv pomocí protokolu SMTP (Simple Mail Transfer Protocol) nebo pomocí javové messagingové služby JMS.

4.4.2 WSDL

Jazyk WSDL [22] slouží k popisu síťových služeb jako množiny koncových bodů zpracovávajících zprávy. Operace a zprávy jsou popisovány na abstraktní úrovni a teprve poté jsou svázány s konkrétním síťovým protokolem a datovým formátem. To umožňuje snadné vytvoření popisu rozhraní, které nabízí jednu službu několika způsoby. V praxi WSDL popisy nejčastěji popisují služby, které si posílají zprávy pomocí formátu SOAP a protokolu HTTP.

WSDL vzniklo jako společná iniciativa firem Microsoft a IBM, které si uvědomily potřebu sjednocení jazyka používaného pro popis rozhraní webových služeb. Navazuje tak na předchozí aktivity, zejména na jazyky NASSL (Network Accessable Service Specification Language), SCL (SOAP Contract Language) a SDL (Service Description Language). WSDL [22] je v současné době vydán jako informativní poznámka W3C a v rámci pracovní skupiny pro popis webových služeb[6] se pracuje na vytvoření skutečného standardu.

WSDL soubor s definicí rozhraní služby je XML dokument. Skládá se zejména z následujících elementů, které tvoří základní části každého WSDL popisu.

`types` Obsahuje definici datových struktur používaných ve zprávách. K definici lze použít teoreticky libovolný typový systém, ale nejčastěji se používají XML schémata. Nástroje pro webové služby se starají o mapování datových typů podle XML schémat na nativní datové typy použitého jazyka.

`message` Definuje formát předávaných zpráv pomocí dříve definovaných datových typů. Zprávy fungují jako vstupní anebo výstupní struktury pro operace. Každá zpráva se může skládat z několika logických částí s vlastním datovým typem. Při použití SOAPu pro RPC odpovídá jedna část zprávy jednomu parametru vzdálené metody.

`operation` Abstraktní definice operací, které jsou službou podporovány. U operace se definuje jaké má vstupy a výstupy. Vstup a výstup je popsán již existující zprávou (`message`). V SOAP RPC modelu odpovídá operace metodě.

`portType` Sdružuje dohromady několik operací.

`binding` Slouží pro navázání určitého typu portu (`portType`) na konkrétní protokol a formát přenosu zpráv.

`port` Jeden koncový bod služby definovaný jako kombinace síťové adresy a dříve definované vazby (`binding`).

`service` Sdružuje několik koncových bodů (portů) do jedné služby.

[6] `http://www.w3.org/2002/ws/desc/`

Naším cílem není popsat zde detailně všechny možnosti WSDL, pro ilustraci uvádím jen ukázku WSDL souboru, který definuje rozhraní výše popsané služby pro zjišťování aktuálního kurzu zadané akcie.

Příklad 4.7: Ukázka WSDL souboru

```
<?xml version="1.0" encoding="utf-8"?>
<definitions name="StockQuote"
             targetNamespace="urn:x-example:services:StockQuote"
             xmlns:tns="urn:x-example:services:StockQuote"
             xmlns:xsd1="http://example.com/stockquote.xsd"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

  <types>
   <schema targetNamespace="http://example.com/stockquote.xsd"
           xmlns="http://www.w3.org/2000/10/XMLSchema">
     <element name="GetLastTradePrice">
       <complexType>
         <all>
           <element name="symbol" type="string"/>
         </all>
       </complexType>
     </element>
     <element name="GetLastTradePriceResponse">
       <complexType>
         <all>
           <element name="Price" type="float"/>
         </all>
       </complexType>
     </element>
   </schema>
  </types>

  <message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:GetLastTradePrice"/>
  </message>

  <message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:GetLastTradePriceResponse"/>
  </message>

  <portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
      <input message="tns:GetLastTradePriceInput"/>
      <output message="tns:GetLastTradePriceOutput"/>
    </operation>
  </portType>
```

```
  <binding name="StockQuoteSoapBinding"
           type="tns:StockQuotePortType">
    <soap:binding style="document"
                  transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetLastTradePrice">
      <soap:operation soapAction=""/>
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>

  <service name="StockQuoteService">
    <documentation>Moje první služba</documentation>
    <port name="StockQuotePort" binding="tns:StockQuoteBinding">
      <soap:address location="http://example.com/StockQuote"/>
    </port>
  </service>

</definitions>
```

4.4.3 UDDI

UDDI nabízí mechanismy pro registrování, kategorizování a vyhledávání webových služeb. UDDI funguje jako velký adresář, který obsahuje informace o subjektech (firmách) a jimi poskytovaných službách. Samotný registr pracuje rovněž jako webová služba a komunikace s ní tedy opět probíhá pomocí SOAPu.

UDDI registr obsahuje následující čtyři druhy entit:

podnikatelské entity (firmy) – business entity U každé firmy v registru jsou zaznamenány základní údaje jako název, stručný popis a kontaktní údaje. Každé firmě mohou být přiřazeny klasifikační identifikátory, které určují oblasti jejího podnikání a geografickou polohu.

služby – business service Ke každé firmě jsou v registru uloženy seznamy služeb, které firma poskytuje. Každá služba je opět popsána a obsahuje seznam šablon vazeb, které ukazují na technické údaje nutné pro využití služby.

šablony vazeb – binding template Šablony popisují, jak a kde je možné se službou komunikovat. Typicky je tato informace popsána odkazem na WSDL soubor s definicí rozhraní služby. Každá šablona kromě toho odkazuje na typ služby, který implementuje.

typy služeb – service typ Typ služby definuje abstraktní službu. Funguje tedy jako obdoba rozhraní, jak je známe např. z Javy. Několik firem může nabízet stejný druh služby se stejným rozhraním a tedy i typem služby. Typ služby je popsán tzv. technickým modelem (tModel).

Typická práce s UDDI probíhá tak, že vývojář prohledá registr a najde si služby, které potřebuje. Získá pro ně popis WSDL a může je začít rovnou používat. Dodejme ještě, že UDDI nemusí obsahovat jen popisy webových služeb ve WSDL, lze do něj ukládat popisy služeb v libovolném formátu. Z důvodu interoperability se však společně s UDDI používá právě SOAP a WSDL.

Vzhledem k tomu, že náš projekt není tak rozsáhlý a zatím neobsahuje nijak velké množství služeb, nebudeme zatím UDDI registr potřebovat. Nebudu se jím proto ani podrobněji dále zabývat.

4.5 Ukázka vytvoření a použití jednoduché webové služby

Na začátku kapitoly jsem došel k závěru, že použití webových služeb bude pro náš projekt asi nejlepší, mimo jiné z důvodu snadného použití. Většině čtenářů však asi soapové zprávy ani WSDL popisy nebudou připadat zrovna průhledné a jednoduché. Nezbývá s nimi než souhlasit. Pravdou však je, že od technických detailů je vývojář většinou zcela odstíněn díky podpoře webových služeb ve vývojových nástrojích. Abych demonstroval snadnost použití webových služeb, rozhodl jsem se do práce zařadit popis vytvoření a použití jednoduché webové služby. Služba nesouvisí nijak přímo s cílem projektu. Vzhledem k tomu, že by tato kapitola diplomové práce měla sloužit i jako návod pro tvůrce ostatních modulů systému RAINBOW, uvádím zde tento jednoduchý příklad. Vytvoření dalších modulů a jejich úspěšné zapojení do komunikační infrastruktury je klíčovým předpokladem úspěšnosti celého projektu.

V Javě vytvoříme jednoduchou webovou službu, která bude schopná sečíst dvě čísla. Pak ukáži, jak můžeme jednoduše napsat klienty využívající tuto službu v Javě, Perlu a C# v prostředí .NET.

4.5.1 Vytvoření služby

Pro vytvoření a provozování služby použijeme balík WASP (Web Applications and Services Platform)[7] od firmy Systinet. Existuje mnoho dalších implementací webových služeb, jejich přehled lze nalézt např. na adrese `http://www.soapware.org/directory/4/implementations`.

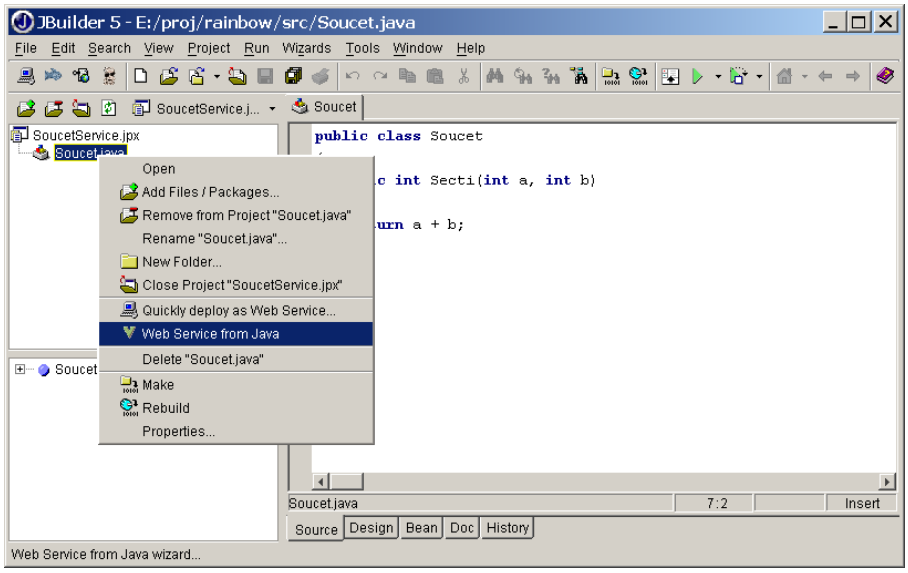
Nejprve si vytvoříme jednoduchou třídu, která bude obsahovat metodu umožňující sečtení dvou čísel (viz příklad 4.8). Všimněte si, že se jedná o zcela běžnou třídu, neobsahuje nic specifického, co by indikovalo, že se má v budoucnu jednat o webovou službu.

Příklad 4.8: Javová třída s metodou pro součet čísel

```
public class Soucet
{
  public int Secti(int a, int b)
  {
    return a + b;
  }
}
```

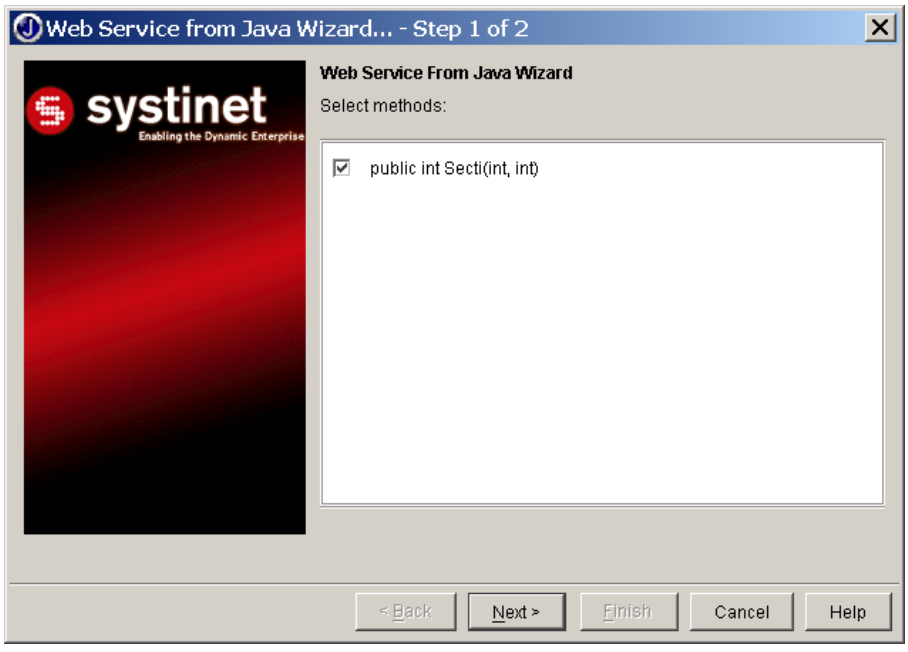
[7] `http://www.systinet.com/`

Nyní chceme z této třídy udělat webovou službu. WASP pro tyto účely nabízí jednak sadu nástrojů ovladatelných z příkazové řádky, jednak mnohem pohodlnější nástroje (WASP Developer), které se integrují přímo do javového vývojového prostředí jako je JBuilder nebo Forte.



Obrázek 4.2: Vytvoření webové služby z javové třídy

Přímo ve vývojovém prostředí proto řekneme, že chceme z třídy udělat webovou službu. Průvodce převodem se nás zeptá, které metody mají být dostupné v rámci webové služby (obrázek 4.3).

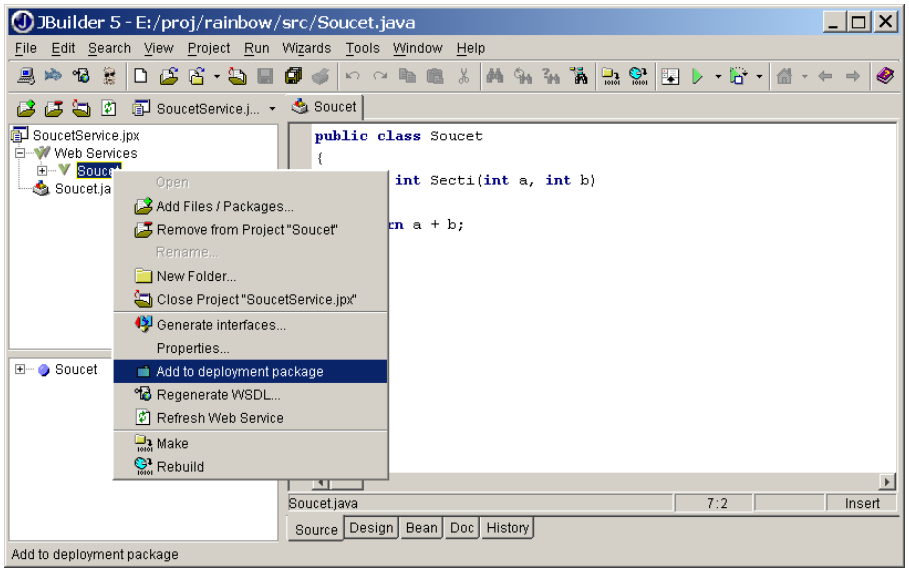


Obrázek 4.3: Výběr metod převáděných na webovou službu

V dalším kroku si vybere jmenný prostor pro službu, její název a další parametry nezbytné pro správné vygenerování WSDL popisu (obrázek 4.4).

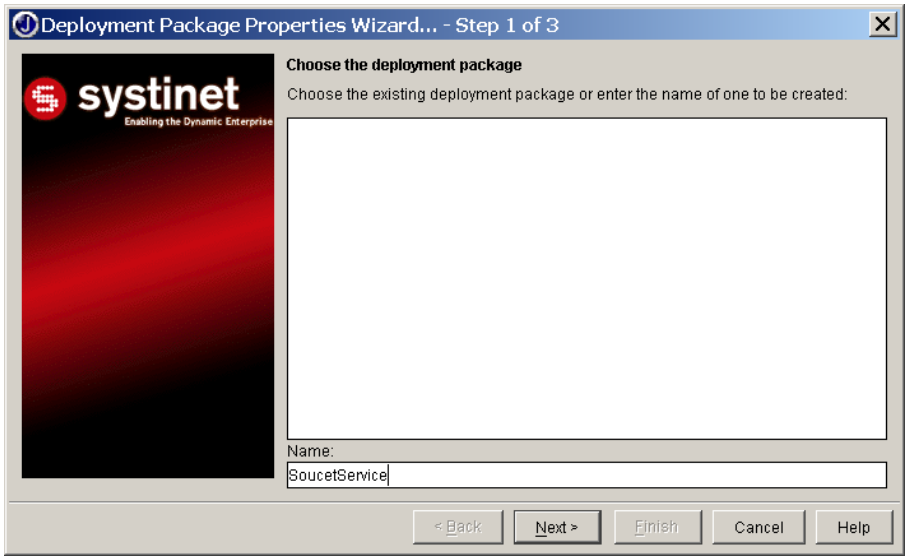
Obrázek 4.4: Nastavení názvu webové služby

WASP Developer nám nyní vygeneroval kostru webové služby a základ WSDL souboru. Z ní vygenerujeme kostru instalačního balíčku (obrázky 4.5 a 4.6).

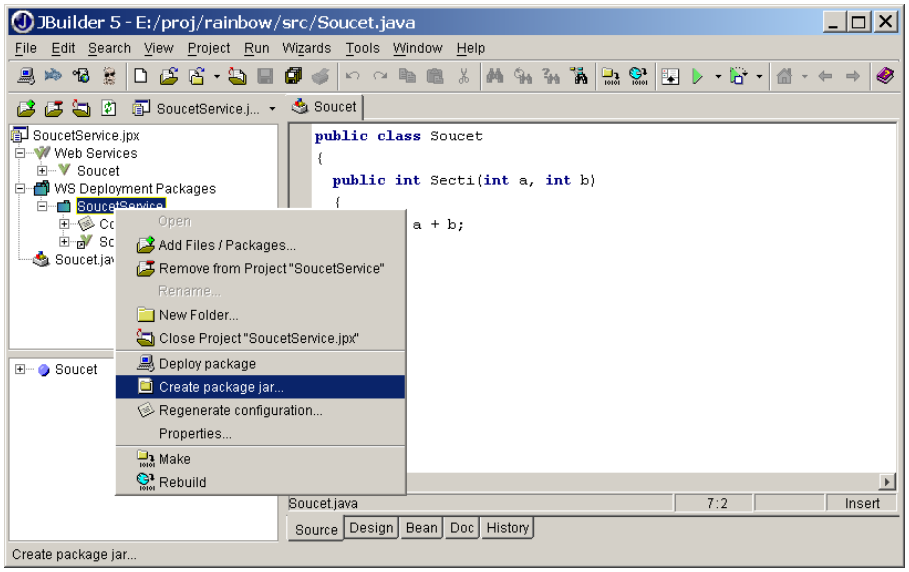


Obrázek 4.5: Vygenerování kostry instalačního balíčku pro webovou službu

Nyní můžeme přikročit k vytvoření plně funkční webové služby. Možností je mnoho, jednou z nich je vytvoření javového archivu, který obsahuje vše podstatné provoz služby – její kód, WSDL popis a další konfigurační soubory, které vyžaduje WASP Server. Vytvoření archivu je zase jen otázkou kliknutí (obrázek 4.7).



Obrázek 4.6: Zadání názvu instalačního balíčku



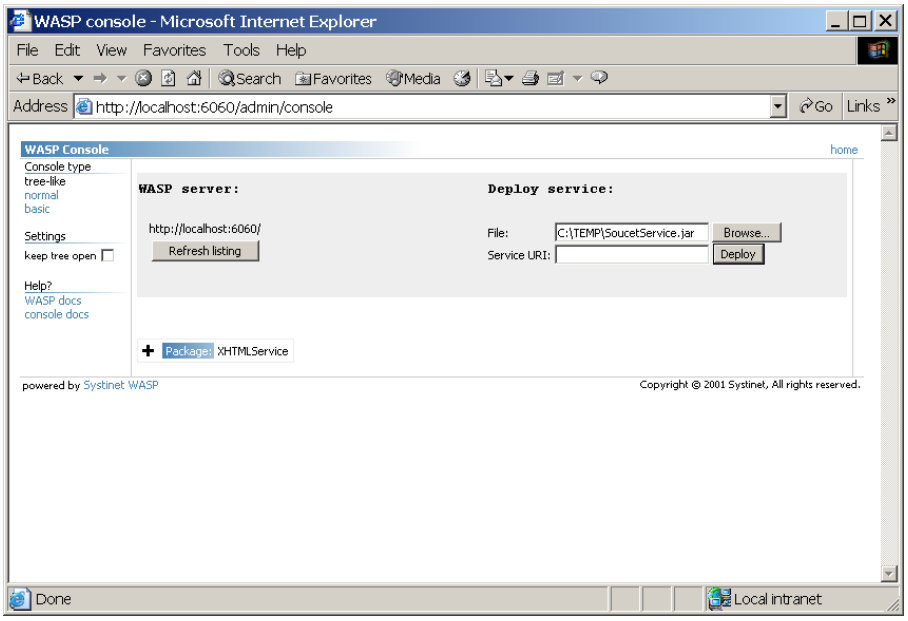
Obrázek 4.7: Vytvoření instalačního archivu

Tento javový archiv pak můžeme použít k instalaci webové služby na server. WASP Server obsahuje administrační konzoli s webovým rozhraním. S její pomocí můžeme konfigurovat existující služby a nové přidávat (nahráním instalačního archivu – viz obrázek 4.8).

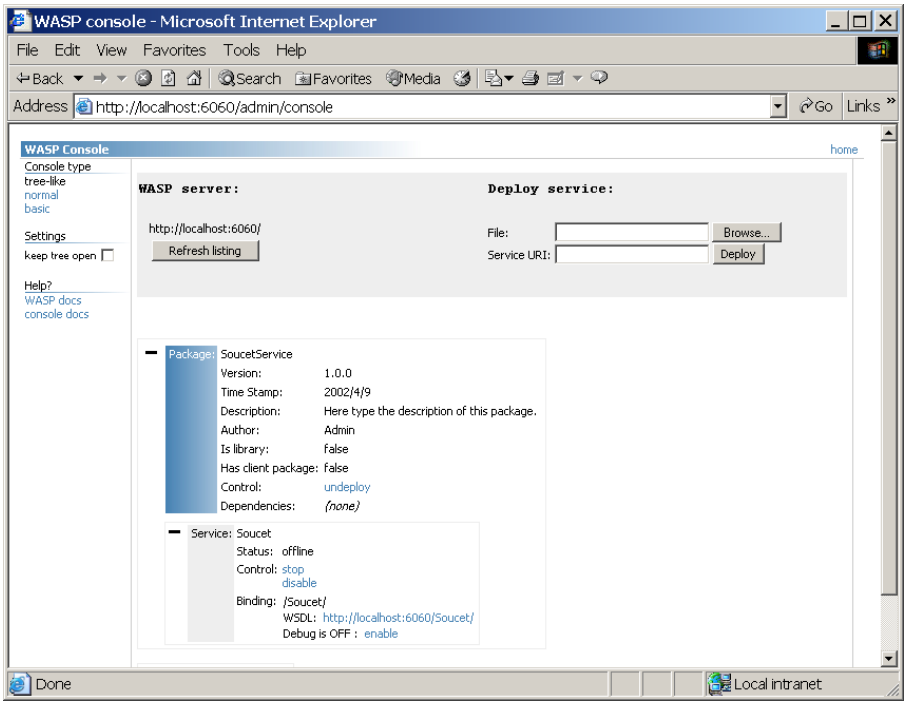
Pomocí administračního rozhraní (obrázek 4.9) lze jednotlivé služby zastavovat a spouštět, odinstalovat, případně si zapnout trasování odesílaných a přijímaných SOAP zpráv. Je zde k dispozici i URL adresa, kde je dostupný popis služby ve WSDL. S jeho znalostí mohou naši webovou službu využívat i další aplikace.

Příklad 4.9: WSDL popis ukázkové webové služby

```
<?xml version='1.0'?>
<wsdl:definitions name='Soucet'
```



Obrázek 4.8: Instalace webové služby na server



Obrázek 4.9: Administrační rozhraní WASP Serveru

```
targetNamespace='urn:x-kosek:services:Soucet'
xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
xmlns:xsd='http://www.w3.org/2001/XMLSchema'
xmlns:tns='urn:x-kosek:services:Soucet'
xmlns:http='http://schemas.xmlsoap.org/wsdl/http/'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
```

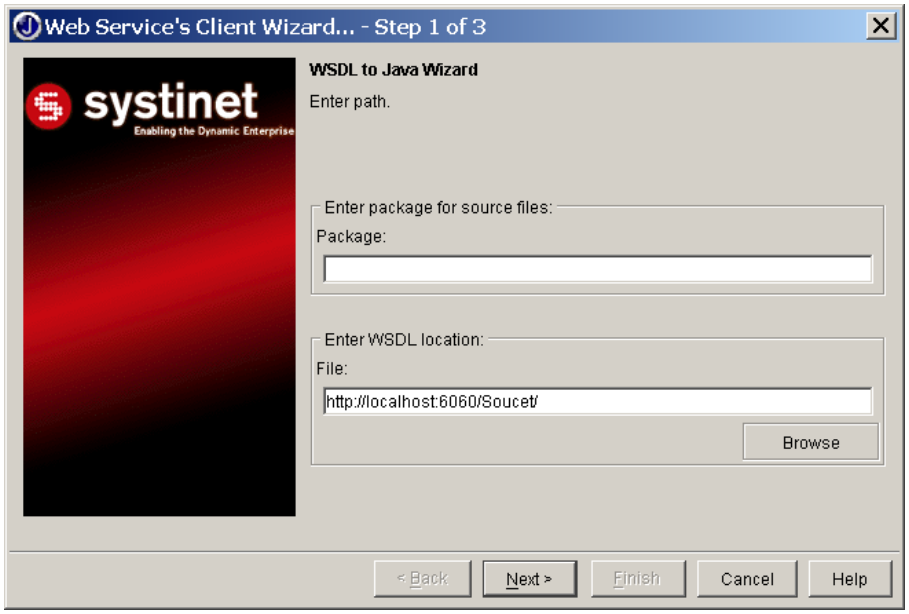
```
  xmlns:mime='http://schemas.xmlsoap.org/wsdl/mime/'
  xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
  xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'>
  <wsdl:message name='Soucet_Secti_Response'>
    <wsdl:part name='response' type='xsd:int'/>
  </wsdl:message>
  <wsdl:message name='Soucet_Secti_Request'>
    <wsdl:part name='p0' type='xsd:int'/>
    <wsdl:part name='p1' type='xsd:int'/>
  </wsdl:message>
  <wsdl:portType name='Soucet'>
    <wsdl:operation name='Secti' parameterOrder='p0 p1'>
      <wsdl:input name='Secti'
                  message='tns:Soucet_Secti_Request'/>
      <wsdl:output name='Secti'
                   message='tns:Soucet_Secti_Response'/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name='SoucetSOAPBinding0' type='tns:Soucet'>
    <soap:binding
     transport='http://schemas.xmlsoap.org/soap/http' style='rpc'/>
    <wsdl:operation name='Secti'>
      <soap:operation soapAction='' style='rpc'/>
      <wsdl:input name='Secti'>
        <soap:body use='encoded'
          encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
          namespace='urn:x-kosek:services:Soucet'/>
      </wsdl:input>
      <wsdl:output name='Secti'>
        <soap:body use='encoded'
          encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
          namespace='urn:x-kosek:services:Soucet'/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name='Soucet'>
    <wsdl:port name='Soucet' binding='tns:SoucetSOAPBinding0'>
      <soap:address location='http://localhost:6060/Soucet/'/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

4.5.2 Javový klient

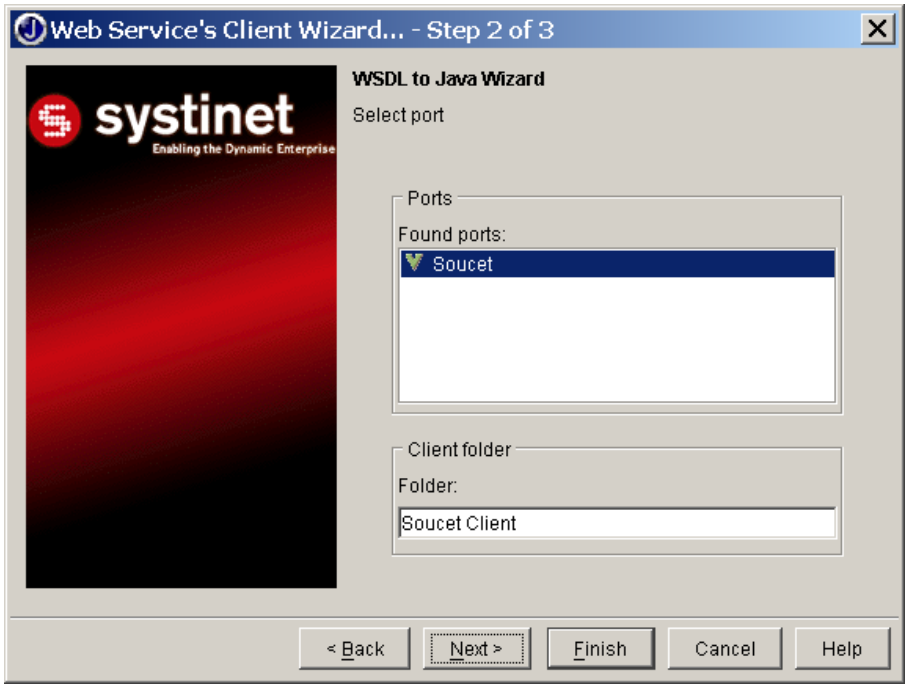
WASP není určen jen pro tvorbu webových služeb, ale i pro tvorbu konzumentů webových služeb. Konzument (klient) potřebuje pro připojení k webové službě znát její WSDL. WASP

Developer nabízí průvodce, který je schopný z WSDL popisu vygenerovat všechen potřebný kód, existují samozřejmě i nástroje spustitelné z příkazové řádky.

Průvodce si od nás nejprve vyžádá URL adresu, na které lze získat WSDL (obrázek 4.10). Analýzou WSDL jsou zjištěny všechny dostupné služby (4.11) a můžeme potvrdit vytvoření kódu pro jejich přístup.



Obrázek 4.10: Generování klienta z WSDL popisu I



Obrázek 4.11: Generování klienta z WSDL popisu II

V našem jednoduchém případě průvodce automaticky vygeneruje dva javové soubory. První obsahuje definici rozhraní, které mapuje WSDL rozhraní na odpovídající javové rozhraní:

```
package client.iface;

public interface Soucet
{
    int Secti(int p0, int p1);
}
```

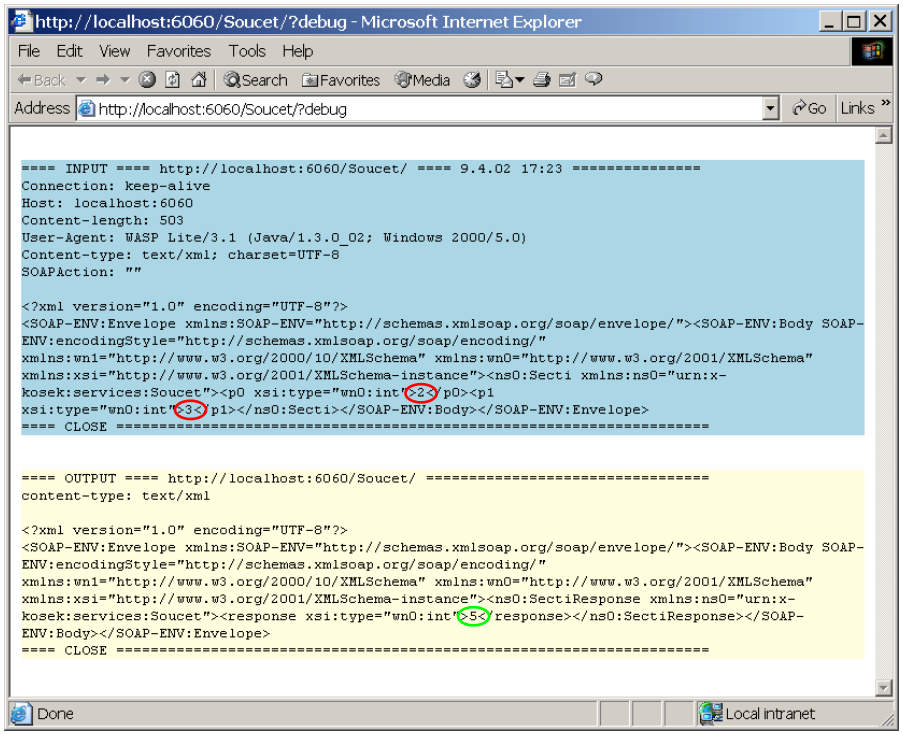
Druhý soubor obsahuje kód, který vytvoří proxy objekt umožňující volání metod webové služby. Tento kód můžeme použít ve vlastním programu nebo jej použít pro otestování služby. Pro názornost si ukážeme, jak sečíst čísla 2 a 3 pomocí webové služby.

```
package client;

import client.iface.*;
import org.idoox.wasp.Context;
import org.idoox.wasp.MessageAttachment;
import org.idoox.webservice.client.WebService;
import org.idoox.webservice.client.WebServiceLookup;

public class SoucetClient{
  public static void main(String args[]) throws Exception {
    String host = ”http://localhost:6060/Soucet/”;

    //init the lookup
    WebServiceLookup lookup = (WebServiceLookup)Context.getInstance
        (”org.idoox.webservice.client.WebServiceLookup”);

    //get the instance of the Web Service interface from the lookup
    //change the interface class to your Web Service’s interface
    Soucet service = (Soucet)lookup.lookup
        (”http://localhost:6060/Soucet/”, Soucet.class,host);

    //now call the methods on your Web  Service’s interface
    System.out.println(service.Secti(2,3));
  }
}
```

Nejzajímavější je poslední řádka programu, kde voláme metodu `Secti()` jakoby byla dostupná lokálně. Objekt `service` je přitom proxy objekt, který volání metody předá webové službě – postará se tedy o vytvoření SOAP zprávy, její doručení, příjem odpovědi, její dekódování a převod zpět do nativních datových typů Javy.

Administrační konzole WASP Serveru umožňuje zapnout sledování SOAP komunikace. Můžeme se pak podívat, jak přesně vypadají SOAP požadavky/odpovědi, které si klient vyměňuje se serverem. Na obrázku 4.12 je zachycen požadavek na sečtení čísel 2 a 3. Je z něj jasně vidět, že SOAP rozhodně není optimalizován na velikost přenášených dat. Požadavek

i odpověď obsahuje velké množství servisních informací. Ty však zaručují snadné použití webových služeb na různých platformách.



Obrázek 4.12: Ukázka SOAP komunikace mezi webovou službou a jejím konzumentem

4.5.3 C# klient pro .NET

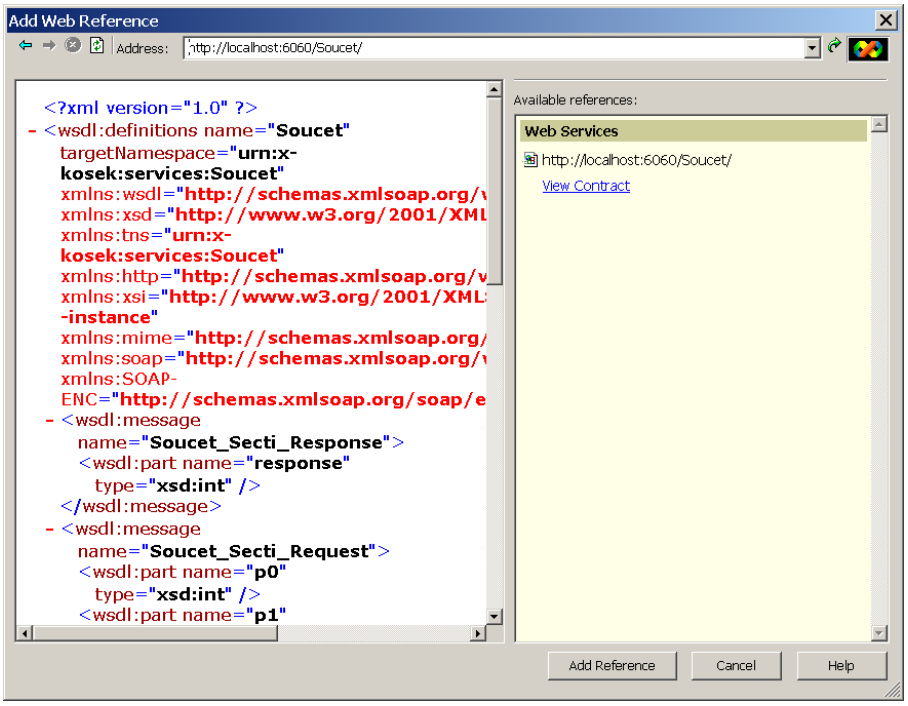
.NET je nová platforma Microsoftu založená na principu podobném Javě. Programy se nyní nepřekládají přímo do binárního kódu pro určitou hardwarovou platformu a operační systém, ale překládají se do mezikódu – CIL (Common Intermmediate Language). Pro spuštění programu pak musí být na počítači k dispozici CLR (Common Language Runtime), který se postará o překlad CIL do nativního binárního kódu a jeho spuštění [40].

.NET také obsahuje velké množství knihoven s výbornou podporou XML a webových služeb. .NET je ostatně kolem webových služeb postaven – Microsoft si uvědomuje potřebu distribuovaných aplikací a s webovými službami je jejich implementace snazší než s použitím DCOM.

Vývojové prostředí Visual Studio .NET obsahuje nástroje pro pohodlnou práci s webovými službami. Pomocí jednoduchého průvodce (4.13) nám Visual Studio .NET vygeneruje z WSDL popisu proxy objekt.

Automaticky se vytvoří třída `Soucet`, která patří do jmenného prostoru `localhost` (ten je určen z adresy serveru, kde je umístěna webová služba – v našem případě se jedná o lokální webový server). Zavolání webové služby je pak hračka:

```
// vytvoření instance proxy objektu
localhost.Soucet service = new localhost.Soucet();
```



Obrázek 4.13: Generování konzumenta z WSDL ve Visual Studiu .NET

```
// zavolání webové služby
int vysledek = service.Secti(2,3);
```

4.5.4 Klient v Perlu

Do třetice si ukážeme volání webové služby z Perlu. Použijeme přitom modul SOAP:Lite,[8] který umožňuje využít webové služby přímo v Perlu. Knihovna opět z WSDL přímo za běhu programu vytvoří proxy objekt, ve kterém můžeme volat vzdálené metody webové služby.

```
use SOAP::Lite;

print "Program na sčítání přes webovou službu\n";
print "Zadej A: ";
$a = <STDIN>;
chop($a);
print "Zadej B: ";
$b = <STDIN>;
chop($b);

print "\nA + B = ";
print SOAP::Lite->service('http://localhost:6060/Soucet/')
        ->Secti($a,$b);
```

[8] http://www.soaplite.com/

Kapitola 5

Navigační rozhraní

Z uživatelského hlediska je navigační rozhraní nejdůležitější částí celého systému. V této kapitole se podíváme na funkce navigačního rozhraní, možnosti jejich implementace a stručně popíšeme implementaci funkčního prototypu navigačního rozhraní.

5.1 Základní funkce

Základní funkce navigačního rozhraní je velmi jednoduchá. Webový prohlížeč by měl v samostatném okně zobrazovat přídavné informace o stránce. Při každé změně aktuálně zobrazené stránky (například po kliknutí na odkaz nebo po ručním zadání nové adresy) by se měly odpovídajícím způsobem aktualizovat i informace v okně navigačního asistenta.

Navigační rozhraní přitom při každé změně načtené stránky musí kontaktovat dotazovací modul RAINBOW a vyžádat si od něj informace o právě zobrazované stránce. Dotazovací modul předá dotaz analytickým modulům a jejich odpovědi propojí dohromady. Výsledek se pak odešle zpět do navigačního rozhraní, které se postará o zobrazení.

5.2 Možnosti implementace

Při výběru vhodného implementačního prostředí musíme zvážit zejména následující faktory:

- pohodlnost a přirozenost ovládání pro uživatele;
- snadnost implementace;
- velikost uživatelské skupiny, která bude moci řešení využít (např. kvůli omezení danému použitou platformou apod.).

5.2.1 Rámy

Jedna z možností implementace zcela staví na možnostech jazyka HTML. Ten umožňuje rozdělit okno prohlížeče na několik částí (tzv. rámů), ve kterých jsou zobrazené samostatné stránky. V jednom rámu by mohlo být načteno navigační rozhraní a ve druhém prohlížená stránka. Pomocí JavaScriptu bychom mohli detekovat změny v hlavním okně.

Výhodou tohoto řešení je schopnost spolupráce s libovolným webovým prohlížečem na libovolné platformě. Samotná implementace by také nebyla příliš složitá. Nevýhodou by

však bylo těžkopádné ovládání pro uživatele. Ten by nemohl pro zadání adresy prohlížené stránky použít klasické vstupní pole prohlížeče, ale musel by adresu psát do speciálního formuláře, který by načetl rámy. To je pro většinu uživatelů zcela neakceptovatelný postup. Navíc není použití rámu dostatečně robustní, neboť mnoho komerčních stránek detekuje načtení do rámu a pomocí JavaScriptu umí rámy zrušit a načíst se do celého okna.

5.2.2 Vlastní prohlížeč

Druhou extrémní možností je napsání vlastního prohlížeče. Většina prohlížečů je dnes dostupná v podobě komponenty (např. ActiveX), která je schopná zobrazovat HTML stránky. Stačí tedy napsat program, který bude mít menu a tlačítka s příkazy známými z běžných prohlížečů a použije již hotovou komponentu pro zobrazování stránek. Do takového prohlížeče můžeme snadno přidat další panel s navigačním rozhraním.

Pro uživatele by sice toto prostředí mohlo být poměrně pohodlné, zvláště kdyby se náš prohlížeč hodně podobal existujícím prohlížečům. Existuje však jen málo uživatelů, kteří jsou ochotni změnit svůj oblíbený prohlížeč a používat jiný. Kdyby se měl náš prohlížeč funkčností přiblížit dnešním prohlížečům, nebyla by implementace příliš snadná ani při využití již existující komponenty pro zobrazování stránek.

5.2.3 Modifikace existujícího prohlížeče

Jako nejvhodnější se nakonec jeví možnost modifikace stávajícího prohlížeče. Většina dnešních prohlížečů umožňuje svou modifikaci přidáním dalších komponent. Uživatel si tak nemusí zvykat na novou aplikaci a způsob práce. Používá stále stejný program, který nabízí nové funkce díky přídavné komponentě.

Mezi nejpoužívanější prohlížeče dneška patří Internet Explorer a Netscape Navigator. Do obou prohlížečů lze přidat další panely. Internet Explorer umí vložit přídavný panel v HTML nebo jako ActiveX komponentu. Netscape Navigator, resp. jeho open source varianta Mozilla umožňují vytvoření panelu v HTML nebo ve speciálním jazyce XUL (XML User Interface Language) pro tvorbu prvků uživatelského rozhraní.

Rozhodl jsem se panel vytvořit pro prohlížeč Mozilla. Jednak je tento prohlížeč dostupný pro mnoho platforem, a tím pádem bude i navigační rozhraní dostupné pro všechny uživatele bez rozdílu. Navíc mi osobně připadá jednodušší vytvoření panelu pomocí jazyka XUL (je popsán dále), než psaní speciální ActiveX komponenty použitelné pouze s Internet Explorerem ve Windows.

5.3 Architektura prohlížeče Mozilla

Prohlížeč Mozilla,[1] na němž je založen i komerční Netscape Navigator, je zajímavý z mnoha důvodů. Jednou z jeho zvláštností je i jeho uživatelské rozhraní, které je popsáno ve speciálním jazyce XUL. Díky tomu lze vzhled Mozilly velice snadno měnit, lze do ní přidávat další funkce a navíc lze jádro Mozilly ve spojení s XUL použít pro vytvoření uživatelského rozhraní ve vlastních programech.

Proč je vlastně v Mozille použit tento poněkud neobvyklý způsob vytváření uživatelského rozhraní? Ve většině jazyků se při tvorbě aplikací používá speciální knihovna, která

[1] `http://www.mozilla.org`

umožňuje vytvářet dialogová okna, vkládat do nich různé prvky uživatelského rozhraní a komunikovat s nimi. Vývojáři aplikací pro Windows nejspíše znají knihovnu MFC, existují i knihovny určené původně pro Linux jako Gtk a Qt. Problém je však v tom, že tyto knihovny podporují jednu a v lepším případě několik málo platforem. Existují verze knihoven Gtk a Qt určené i pro Windows, a existují i další multiplatformní knihovny jako wxWindows. S většími či menšími obtížemi tak lze vyvíjet aplikace, které půjde spustit na několika málo platformách.

Když se však podíváme na stránky projektu Mozilla, zjistíme, že prohlížeč je k dispozici pro opravdu širokou škálu platforem: Windows, MacOS 9, MacOS X, Linux, AIX, BeOS, OpenVMS, HPUX, FreeBSD, NetBSD, BSD/OS, Caldera OpenUNIX8, OS/2, Solaris, Irix, Tru64 Unix. Těžko bychom hledali nějakou knihovnu, která by umožnila vyvíjet grafické aplikace jednotným způsobem pro všechny tyto platformy.

Mozilla podporuje opravdu mnoho platforem, a tak se její vývojáři museli s problémem přenositelnosti vypořádat vlastními silami. Vytvořili přitom velice flexibilní systém, který přináší i další výhody. Jádrem Mozilly je zobrazovací jádro Gecko. To umí zobrazovat HTML stránky a podporuje samozřejmě kaskádové styly (CSS) a JavaScript. Kromě HTML umí interpretovat i jazyk XUL, který popisuje prvky uživatelského rozhraní jako menu, okna, vstupní pole, tlačítka apod. Mozilla, kterou vidíme na obrazovce našeho počítače, tedy není nic jiného než jedno okno, ve kterém Gecko zobrazuje uživatelské rozhraní definované pomocí XUL.

Samotný XUL nijak přesně nedefinuje vzhled rozhraní. Barvy, způsob zarovnání, velikost okrajů a obrázky, ze kterých se rozhraní skládá, jsou definovány nezávisle pomocí kaskádových stylů. Souboru kaskádových stylů a obrázků, které definují vzhled určité části aplikace, se říká skin. K jednomu XUL rozhraní můžeme mít několik definic vzhledu.

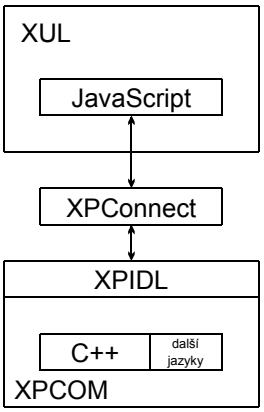
XUL a kaskádové styly řeší pouze zobrazení uživatelského rozhraní. Každá smysluplná aplikace však musí reagovat na požadavky uživatele. Přímo v XUL lze proto pro jednotlivé události vyvolávané uživatelem definovat kód v JavaScriptu, který se postará o obsluhu události. Samozřejmě že v JavaScriptu nelze psát nějaké velké a složité aplikace. Mozilla proto obsahuje sadu několika technologií, které umožňují pomocí JavaScriptu vyvolávat nativní kód napsaný v C++ a dalších jazycích.

Všechny technologie, které se používají pro propojení JavaScriptu s nativním kódem, mají v názvu písmena „XP“, které symbolizují přenositelnost mezi platformami. Základem psaní přenositelného binárního kódu je XPCOM (cross-platform component object model). Jedná se obdobu rozhraní COM známého z Windows. Pokud píšeme nějaký kód například v C++, vytvoříme pro něj XPCOM obálku, která obsahuje rozhraní nezávislé na jazyku a použité platformě. Ostatní komponenty pak mohou snadno využívat služby nabízené tímto objektem. Samotný kód v C++ samozřejmě musíme vždy přeložit pro konkrétní platformu, ale způsob vyvolání tohoto kódu je díky XPCOMu stejný na všech systémech.

XPIDL (cross-platform interface definition language) umožňuje popsat rozhraní definovaná jednotlivými objekty. Jeho syntaxe je velice blízká jazyku IDL. Pomocí speciálního překladače pak můžeme z XPIDL automaticky vygenerovat hlavičkové soubory a další potřebné soubory pro snazší vytvoření XPCOM objektů.

O samotné propojení JavaScriptu a objektů XPCOM se stará technologie XPConnect, které umožňuje přímé volání XPCOM objektů z JavaScriptu a naopak. Vzájemné propojení jednotlivých XP-technologií je zachyceno na obrázku 5.1.

Jelikož je Mozilla poměrně mladý projekt, byl pro jeho autory výběr syntaxe jazyka XUL poměrně jednoduchý – XUL je jazyk založený na XML. Pro jednotlivé prvky uživatelského rozhraní existují odpovídající XML elementy. Chování prvků je pak ovlivňováno pomocí



Obrázek 5.1: XP-architektura Mozilly

parametrů, které se zapisují jako XML atributy. Vzhled prvků je definován pomocí kaskádového stylu.

K dispozici máme všechny běžné prvky uživatelského rozhraní:

- okna a dialogová okna;
- menu, submenu a položky menu;
- přepínací tlačítka;
- zaškrtávací tlačítka;
- lišty nástrojů;
- textové popisky a obrázky;
- vstupní textová pole;
- tlačítka;
- karty se záložkami;
- pop-up menu;
- seznamy a hierarchické (stromové) seznamy;
- a další.

Kromě těchto standardních prvků můžeme v XULu používat libovolné HTML elementy, které se do dokumentu vkládají pomocí zvláštního jmenného prostoru.

Vzájemné umístění prvků lze ovládat třemi způsoby. První možností je uzavírat skupiny prvků do boxů, které se skládají vedle sebe nebo nad sebe. Tento způsob je velice jednoduchý a pro většinu aplikací s ním bohatě vystačíme. Další možností je použít některý ze zabudovaných layout manažerů.[2] Poslední možností je nastavit umístění prvků pomocí kaskádového stylu.

[2]Layout manažer je neviditelná komponeta uživatelského rozhraní, která se stará o vzájemné rozmístění ostatních komponent uživatelského rozhraní.

Propojení kódu s uživatelskými akcemi se děje podobně jako v HTML pomocí událostí. U každého elementu můžeme použít speciální atributy, které odpovídají jednotlivým akcím provedeným uživatelem. Hodnotou atributu bude javascriptový kód, který se použije pro obsloužení příslušné události. Jednoduché aplikace vystačí pouze s JavaScriptem, ty složitější pak zavolají binární objekty pomocí XPConnectu. Mezi nejpoužívanější událost patří `oncommand`, která se vyvolává např. výběrem položky z menu nebo stiskem tlačítka.

V současné podobě lze pomocí jazyků XUL a CSS dosáhnout oddělení definice rozhraní od jeho vzhledu. Mozilla umožňuje oddělit i definici chování rozhraní. Nový jazyk XBL (Extensible Bindings Language) [20] umožňuje definovat obsluhu událostí mimo XUL (případně HTML) kód. Základní princip je přitom stejný jako při použití oddělených definic chování (behavior), se kterými přišel už Internet Explorer 5. Podobné zaměření má i standard XML events [30] vznikající na půdě W3C. Kaskádový styl může pro každý element kromě prezentačních vlastností definovat i jméno souboru, který obsahuje definice funkcí obsluhujících jednotlivé události. V závislosti na použitém stylu se pak může změnit i chování celé aplikace.

I když lze XUL a Gecko použít pro tvorbu aplikací přenositelných mezi platformami, nemyslím si, že by to byl hlavní přínos. Samotné jádro Mozilly, které XUL kód interpretuje zabere v paměti poměrně dost místa, které by jinak mohla využívat aplikace. Otevřená architektura Mozilly se podle mne využije především při rozšiřování schopností prohlížeče. Pomocí XULu můžeme jednoduše upravit rozhraní Mozilly – přidat okno pro aktuální firemní zprávy, burzovní zpravodajství, pohodlné vyhledávání pomocí navigačního asistenta atd. Z prohlížeče se tak dá velice snadno vytvořit pohodlné prostředí pro vstup do digitálního komunikačního světa.

5.4 Implementace

Pro implementaci prototypu navigačního rozhraní jsem se rozhodl použít prohlížeč Mozilla. Ten obsahuje postranní pruh (sidebar), do kterého lze velmi snadno přidat vlastní panely vytvořené v HTML nebo XUL. Panel musí čekat na změnu stránky zobrazené v hlavním okně prohlížeče. Změnu lze detekovat dvěma způsoby:

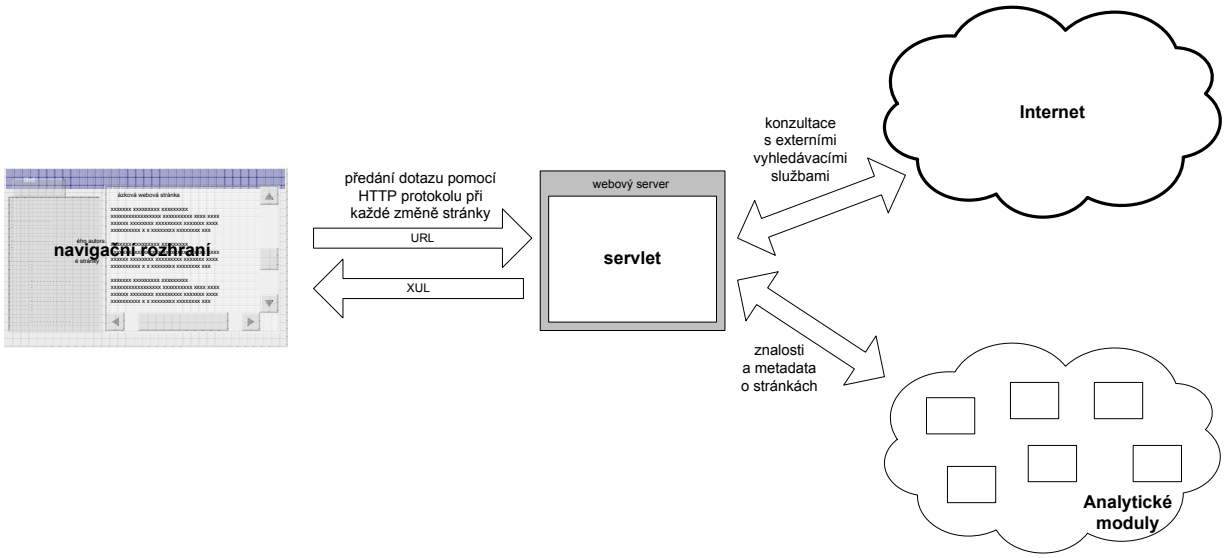
- panel se zaregistruje jako speciální komponenta, která obdrží událost při změně zobrazené stránky;
- panel bude v pravidelných časových intervalech kontrolovat, zda nedošlo ke změně stránky.

První varianta je samozřejmě elegantnější. Bohužel ji nemůžeme použít. Klasicky nainstalované panely nemohou kvůli bezpečnostním omezení Mozilly odposlouchávat změny v hlavním okně.[3] Pokud chceme, aby měl panel vyšší oprávnění, můžeme ho nainstalovat pomocí instalačního balíčku XPI, což je formát používaný pro instalaci přídavných komponent Mozilly. Bohužel v Mozille je dosud neodstraněná chyba, která znemožňuje instalaci vlastních komponent s vyššími právy právě do postranního panelu.

Musel jsem proto použít druhé, méně elegantní, řešení. Do panelu se vloží jednoduchý skript v JavaScriptu, který každou sekundu zjišťuje právě aktivní stránku v hlavním okně.

[3] Důvod je podle vývojářů Mozilly v ochraně soukromí uživatele. Kdyby mohl libovolný panel sledovat pohyb uživatele po webu, jednalo by se o významný zásah do jeho soukromí.

Pokud se stránka změní, pošle se HTTP požadavek na javový servlet, který slouží jako dotazovací modul (viz obrázek 5.2). V parametrech se pošle URL adresa právě načtené stránky. Dotazovací modul kontaktuje analytické moduly a další služby, vyžádá si od nich informace, zapíše je do hierarchické podoby pomocí XUL, které je odesláno zpět do navigačního rozhraní. Uživatel tak získá přídavné informace o právě načtené stránce.



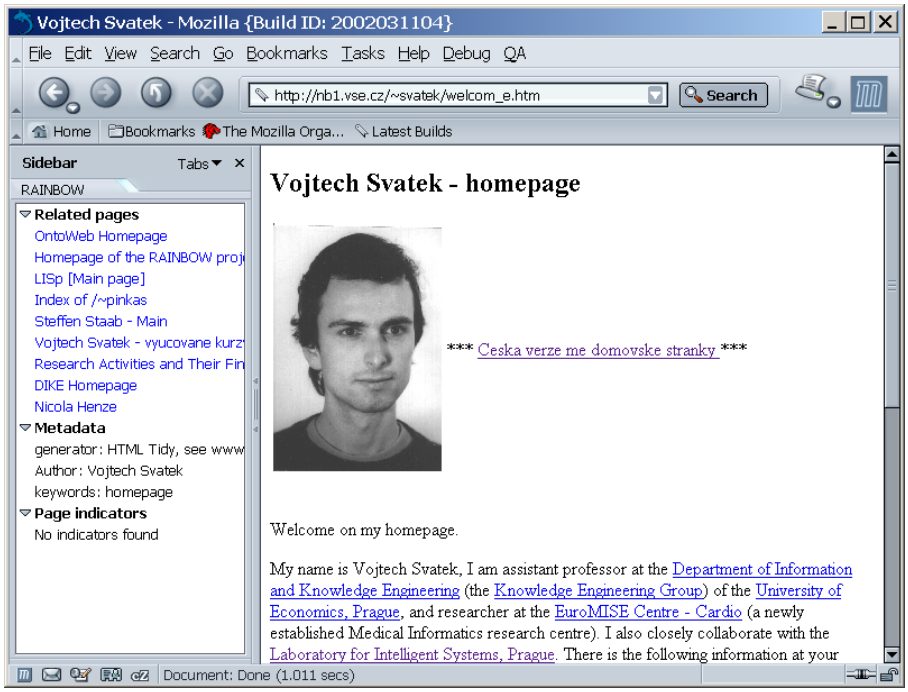
Obrázek 5.2: Schéma obsloužení požadavku navigačním rozhraním

Oproti původnímu návrhu RAINBOW jsem v prototypové implementaci sloučil dohromady dotazovací modul a modul překládající výsledek dotazu do formátu vhodného pro navigační rozhraní. Funkčních analytických modulů je zatím tak málo, že by bylo zbytečným zdržováním a komplikováním rozdělovat tuto funkčnost do dvou nezávislých komponent. V budoucnu, až bude více funkčních modulů, se toto rozdělení samozřejmě provede. Servlet pak pomocí SOAPu bude komunikovat s dotazovacím modulem, který se postará o agregaci výsledků z dalších modulů (s nimi bude opět komunikovat pomocí SOAPu).

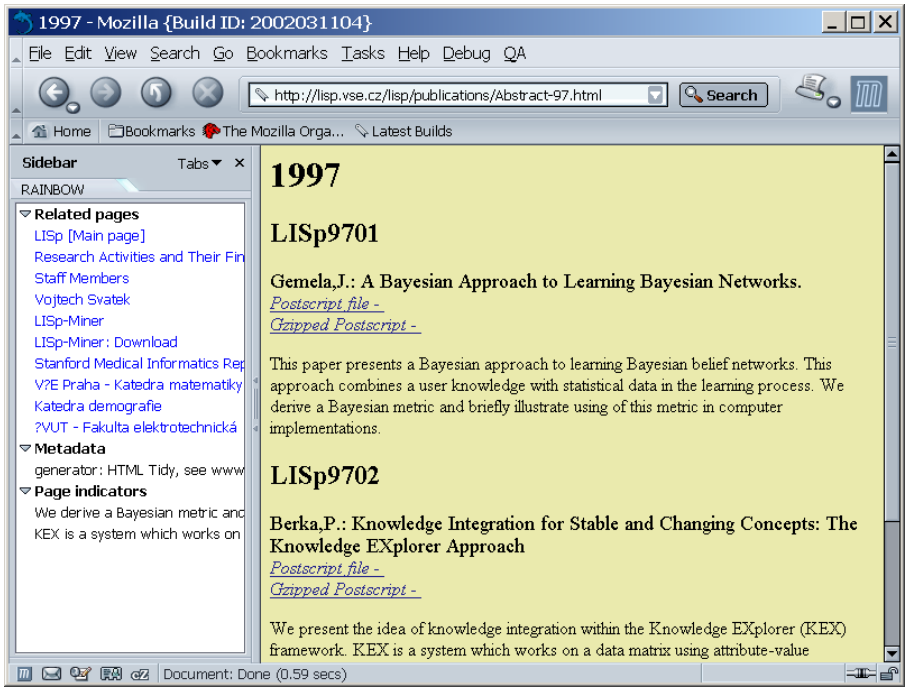
Zdrojové kódy servletu a navigačního rozhraní naleznete v příloze B – *Navigační rozhraní*. Dotazovací modul v současné době používá externí vyhledávací službu Google[4] pro zjištění souvisejících stránek a dvě původní služby RAINBOW pro získání metadat a indikátorů stránky. Na obrázcích 5.3[5] a 5.4 si můžete prohlédnout navigační rozhraní v provozu.

[4]Google bohužel na mnoha stránkách v češtině nahradí znaky s diaktritikou otazníkem. Názvy některých stránek v navigačním rozhraní jsou proto zkomolené.

[5]Stránka vedoucího mé diplomové stránce nebyla vybrána ani náhodně, ani z vypočítavosti. Je jednou z mála, která obsahuje metadata.



Obrázek 5.3: Ukázka navigačního rozhraní v provozu



Obrázek 5.4: Navigační rozhraní zobrazuje indikátory stránky získané analytickým modulem

Kapitola 6

Stahování a ukládání stránek

Současný návrh systému RAINBOW počítá s tím, že se zdrojové webové stránky předzpracují do podoby znalostní báze. Systém tedy bude funkční pouze pro omezenou část webových stránek, které budou předem zpracovány. V této části práce se proto podíváme na způsoby, jakými lze z Internetu stahovat kolekce webových stránek pro další zpracování. Nejprve stručně zhodnotíme jednotlivé přístupy k tomuto problému a poté se seznámíme s implementací vlastního stahovacího robota použitého v projektu RAINBOW.

6.1 Možné přístupy ke stahování a ukládání stránek

Ještě než můžeme ze stránek získávat nějaké informace a ukládat je do znalostní báze, musíme někde stránky získat. Dnes se pro tuto činnost běžně používají weboví roboti,[1] což jsou programy, které stáhnou zadanou webovou stránku a uloží ji. Kromě toho stejným způsobem zpracují i všechny stránky, na které z původní stránky vedly odkazy. Tímto způsobem lze po určité době získat podstatnou část webových stránek. Většina robotů samozřejmě stahuje stránky jen z omezeného počtu serverů – např. pro účely českých vyhledávacích serverů roboti obvykle stahují jen dokumenty z národní domény `.cz`. Pokud naopak chceme získat co nejvíce stránek, můžeme z WHOIS serverů obsahujících zaregistrované domény získat seznam potencionálnách webových serverů a pokusit se je všechny zpracovat.

Pro projekt RAINBOW webového robota samozřejmě potřebujeme. Stáli jsme proto před rozhodnutím, jakého robota použít. Při rozhodování byla nejdůležitější především následující dvě kritéria:

- uložené stránky musí být snadno přístupné pro další aplikace;
- proces stahování by měl jít ovlivnit – např. by mělo být možné upravovat seznam povolených a zakázaných domén, měla by být podporována různá česká kódování apod.

Pro samotné uložení stahovaných stránek se nabízejí tři možnosti:

- uložení do souborů na disku tak, že každé stránce odpovídá jeden soubor;
- uložení do relační databáze tak, že každé stránce odpovídá jeden záznam;
- uložení do vlastního formátu, kde bude více stránek v jednom fyzickém souboru.

[1] Někdy též známí pod angl. názvem *spider* nebo *crawler*.

Výčet není samozřejmě kompletní. Pro ukládání bychom mohli využít i některou z nativních XML databází nebo dokumenty dekomponovat na úroveň elementů a ty ukládat jako jednotlivé záznamy do relační databáze. Dekompozice na elementy pro naší aplikaci však nepřináší žádnou výhodu, spíše naopak, protože většina dokumentů pracuje s dokumentem jako s celkem. Nativní XML databáze jsou zatím v počátku svého vývoje.

6.1.1 Ukládání stránek do souborů

Stahování stránek do souborů je implementačně nejjednodušší, protože lze využít již existující programy (např. WGET [44] nebo w3mir [43]). Stahování stránek do souborů má však i následující nevýhody:

- většina operačních systémů má limit pro počet souborů na disku, který se pohybuje obvykle v řádu desítek až stovek tisíc;
- nad procesem stahování nemáme kontrolu, takže nemůžeme snadno detekovat duplicitní dokumenty, sjednotit kódování dokumetů apod.;
- procesem stahování na disk jsou změněny odkazy, kdy některé vedou k dalším stránkám na disku a některé stále odkazují na Internet – sestavení orientovaného grafu znázorňujícího odkazy mezi stránkami je obtížné, ne-li nemožné;
- pro další zpracování a využití ostatními moduly není uložení do souborů moc šikovné.

6.1.2 Ukládání stránek do relační databáze

Ukládání stránek do relační databáze si vynutí napsání vlastního robota (nebo úpravu stávajícího) určeného pro stahování stránek. To však není zas tak obtížný úkol. Získáme tím navíc plnou kontrolu nad celým procesem stahování stránek. Můžeme pak detekovat například duplicitní stránky, sjednotit kódování dokumentů apod.

Samotné využití relační databáze přinese následující výhody:

- do relační databáze lze velice snadno přistupovat z téměř libovolného jazyka, nebude proto problém využívat získaná data v dalších analytických modulech;[2]
- databázové servery obvykle ukládají všechna data do jednoho souboru, proto nejsme limitováni počtem souborů (webových stránek);
- k databázovému serveru lze přistupovat i vzdáleně pomocí TCP/IP – není problém celý systém rozložit na několik počítačů.

Na druhou stranu existují limity pro maximální velikost souboru (obvykle okolo 2 GB) a některé starší databázové servery obvykle neumějí rozdělit jednu tabulku nebo databázi do více souborů.

[2]Později při implementaci RAINBOW jsme se rozhodli jít cestou webových služeb a přístup ke stránkám je zprostředkován samostatnou službou, která ostatní moduly od skutečného úložiště stránek odstíní.

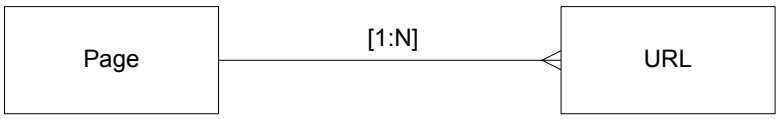
6.1.3 Ukládání do vlastního formátu

Nevýhodou databázového serveru je, že používané datové struktury a přístupové algoritmy nejsou plně optimalizovány pro naši úlohu. Kdybychom napsali vlastní specializovanou jednoúčelovou databázi, mohli bychom dosáhnout vyšší efektivity. Náročnost této úlohy, nutnost implementovat síťový přístup apod. nás myslím opravňují použít již existující databázový systém, který možná bude o pár procent pomalejší a paměťově náročnější, ale ušetří nám spoustu práce. Zvláště pokud celý systém RAINBOW chápeme jako prototypovou implementaci.

6.1.4 Datový model pro uložení stránek

Rozhodneme-li se pro uložení stažených stránek do databáze, musíme se shodnout na datovém modelu, který se pro ukládání stránek bude používat.

Ačkoliv by se na první pohled mohlo zdát, že si pro ukládání stránek vystačíme s jednou tabulkou, opak je pravdou. Není neobvyklé, že jedna stránka se na Webu objevuje v několika exemplářích s různými URL adresami.[3] To je pro nás sama o sobě důležitá informace a navíc nám to umožní ušetřit místo potřebné pro uložení dat. Informace o stažených stránkách proto uložíme do dvou tabulek `Page` a `URL` se vzájemným vztahem 1:N (viz obrázek 6.1).



Obrázek 6.1: ER-diagram databáze pro ukládání stránek

Název atributu	Typ	Popis
id	int, automaticky zvětšovaný	Identifikátor stránky, slouží jako umělý primární klíč tabulky
md5	char(32)	Výtah zprávy používaný při kontrole změny obsahu stránky a hledání kopií jedné stránky
md5dia	char(32)	Výtah zprávy bez diakritiky používaný při kontrole změny obsahu stránky a hledání kopií jedné stránky
type	varchar(128)	MIME typ stránky (např. `text/html`, `text/plain`)
data	longvarchar/clob	Samotný obsah stránky
modified	timestamp	Datum a čas poslední modifikace záznamu tj. stránky

Tabulka 6.1: Struktura tabulky Page

[3] Do této skupiny patří i verze stránky v různých kódováních, které jsou dostupné na drobně odlišných URL adresách.

Název atributu	Typ	Popis
url	varchar(1024)	URL zdroje, primární klíč tabulky
page_id	int	Identifikátor stránky, kterou dané URL obsahuje
fetched	datetime	Datum a čas prvního stažení URL
checked	datetime	Datum a čas poslední kontroly URL
expires	datetime	Datum a čas vypršení platnosti zdroje
status	int	Poslední HTTP stavový kód získaný při čtení URL

Tabulka 6.2: Struktura tabulky URL

Některé moduly mohou potřebovat i informaci o odkazech mezi dokumenty. Struktura odkazů není nic jiného než orientovaný graf. Ten lze v relačním modelu zachytit například jako seznam uspořádaných dvojic zdrojového a cílového uzlu.

Název atributu	Typ	Popis
source	int	Identifikátor zdrojové stránky
target	int	Identifikátor cílové stránky

Tabulka 6.3: Struktura tabulky Link pro ukládání odkazů mezi stránkami

Primární klíč je v tabulce `Link` přitom tvořen oběma dvěma atributy. Tato struktura umožňuje velice jednoduše zjistit přímého předka a následníka každé stránky. Pro hledání nepřímých vazeb (přes více stránek) však jazyk SQL, který se používá pro dotazování v tabulkách, nenabízí vhodné prostředky.

6.2 Návrh postupu pro stahování a ukládání stránek

Modul pro stahování stránek bude sloužit k naplnění a údržbě výše popsané databáze stránek. V následujícím textu se pokusím popsat, jak by měl modul pro stahování stránek pracovat a jaké další pomocné datové struktury budeme potřebovat.

URL stránek, jež potřebujeme stáhnout, budeme průběžně ukládat do fronty, ze které budou postupně čtena. Stahnutí a zpracování stránky by mělo probíhat zhruba podle následujícího scénáře:

1. *Stažení stránky z daného URL.* Pokud je stránka nedostupná, zapíše se do tabulky URL chybový kód a stahování se ukončí.

2. *Normalizace kódování stránek.* Pro další zpracování je nezbytně nutné sjednotit kódování češtiny používané na stránkách. Pro naše potřeby bude asi nejvhodnější všechny stránky převést do kódování ISO 8859-2, nebo UTF-8. Detekce kódování není zcela triviální záležitost, ale můžeme se inspirovat například zdrojovými kódy vyhledávače Sherlock [29], který tento problém poměrně uspokojivě řeší.

3. *Kanonizace do XML.* Převážná většina dnešních HTML stránek obsahuje mnoho syntaktických chyb a znemožňuje jejich snadné zpracování pomocí jednoduchého parseru.

Při načítání proto všechny stránky převedeme do XML, a tím usnadníme jejich zpracování dalším aplikacím.Tento krok se vyplatí i s ohledem na budoucnost. Další verze HTML – XHTML – má již přímo syntaxi XML, na Webu se postupně objeví informace dostupné pouze v XML.

4. *Testování změny stránky a opakovaného výskytu.* V tomto kroku zjistíme, zda se načtená stránka již v databázi nevyskytuje pod jiným URL, nebo zda nedošlo ke změně stránky od jejího posledního zařazení do databáze.

5. *Aktualizace databáze.* V závislosti na předchozím kroku jsou příslušné tabulky v databázi modifikovány – buď je přidána nová stránka, nebo jsou upraveny údaje o stávajících stránkách a URL.

6. *Aktualizace fronty dokumentů ke stažení.* Ze zpracovaného dokumentu získáme všechny odkazy. Ty, které ještě nemáme uložené v tabulce `URL` nebo ve frontě, přidáme do fronty URL adres ke zpracování.

6.2.1 Stahování stránek z daného URL

Samotné stažení stránky z dané URL adresy není příliš složité, protože mnoho programovacích jazyků již přímo obsahuje funkce, které toto umožňují. Pokud však stahujeme větší množství stránek z více serverů, musíme zvážit i další okolnosti.

Existuje všeobecně akceptovaný standard [36], kterým může správce webového serveru zakázat stahování stránek pro automaticky pracující roboty. Informace o stránkách, které by se neměly stahovat, se ukládají do souboru `robots.txt` v kořenovém adresáři webového serveru. Pro naše potřeby bychom si měli pro každý server (doménové jméno + port) pamatovat obsah tohoto souboru a při stahování se jím řídit.

Zákaz indexování a zpracování dalších odkazů lze uvést i přímo v záhlaví dokumentu. Náš robot by měl respektovat i toto nastavení v hlavičce stránky:

```
<META name=”ROBOTS” content=”NOINDEX, NOFOLLOW”>
```

Pro efektivní stahování by bylo výhodné stahovat stránky paralelně z několika serverů najednou. Aby nedocházelo k přetěžování serverů, nemělo by probíhat stahování více souborů z jednoho serveru zároveň. Fronta URL ke stažení proto bude muset fungovat jako několik samostatných front pro jednotlivé servery.

Z našeho úhlu pohledu budeme jako webový server chápat dvojici jeho doménového jména (případně IP adresy, pokud doménové jméno nebude k dispozici) a portu. Pro každý webový server si budeme pamatovat informace získané z `robots.txt` a stavový kód poslední operace. Pokud bude výsledkem poslední operace na serveru chyba, která znamená nefunkčnost serveru, budeme muset na nějaký čas přerušit stahování ze serveru.

6.2.2 Normalizace kódování

Jazyk HTML používá jako znakovou sadu Unicode [35], [41]. Pro zápis unicodových znaků však mohou různé stránky používat různá kódování. Pro češtinu se nejčastěji používají kódování `windows-1250`, `iso-8859-2` a `utf-8`. Mnoho stránek je z historických důvodů přístupných pomocí různých modulů pro dynamickou změnu kódování i v několika dalších kódováních – např. Mac CE, CP852, kódování bratří Kamenických apod.

Správné rozpoznání kódování je klíčové pro namapování získané sekvence bajtů na unicodový textový řetězec. Ten pak může posloužit jako vstup pro další moduly. Specifikace HTML [35] v části 5.2 popisuje, jakým způsobem se má zjišťovat kódování dokumentu. Nejvyšší prioritu má obsah hlavičky `Content-type` z HTTP odpovědi. Ten může obsahovat např.:

```
Content-type: text/html;charset=iso-8859-2
```

Hlavička říká, že se jedná o HTML dokument (MIME typ `text/html`) a že je v kódování `iso-8859-2`. Pokud není kódování určeno v HTTP hlavičkách, hledá se odpovídající meta tag v přímo v HTML dokumentu.

```
<meta http-equiv="Content-type"
      content="text/html;charset=iso-8859-2">
```

Jediný problém je v tom, že některé stránky neobsahují ani meta tag, ani se informace o kódování neposílá v hlavičkách. V tomto případě musíme kódování uhodnout. Jednak můžeme použít analýzu četnosti výskytu jednotlivých znaků, která se pro různá kódování samozřejmě liší. Pro stránky v češtině navíc platí, že nejčastěji bývají právě v kódování `windows-1250`.

6.2.3 Kanonizace do XML

Pro převod HTML stránek do XML můžeme použít program HTML-Tidy [34]. Ten kromě převodu do XML odstraní mnoho chyb, které se dnes v HTML kódu vyskytují.

Převod XML a XHTML[4] do XML není vůbec nutný. Dokument zkrátka jen uložíme do databáze.

Převod textových dokumentů do XML je velice jednoduchý. Můžeme se pokusit vytvořit heuristiky, které budou hledat například jednotlivé odstavce textu a uzavřou je do nějakého elementu.

Převod SGML dokumentů také nebude nikterak obtížný. Pokud SGML dokument znormalizujeme,[5] dostaneme formát, z něhož lze XML vyrobit velice jednoduše.

Dalšími složitějšími formáty, jako je PDF nebo DOC, se zatím zabývat nebudeme. Nicméně je vhodné převod do XML napsat modulárně tak, aby šla v budoucnu snadno přidat podpora pro další vstupní formáty.

6.2.4 Testování změny a opakovaného výskytu

Pokud budeme chtít nějakým efektivním způsobem zjistit, zda právě získaná stránka již není v naší databázi pod jiným URL, s výhodou využijeme vlastnosti algoritmu MD5 nebo obdobného „message digest“ algoritmu. Funkce MD5 přiřazuje každému textu 128bitové číslo. Z vlastností algoritmu vyplývá, že pokud jsou dva texty různé, je pravděpodobnost, že budou mít stejnou hodnotu MD5, velmi malá.

[4] XHTML je verze HTML zapisovaná striktně v XML syntaxi.

[5] Normalizací SGML dokumentu se označuje proces, kdy se do instance dokumentu doplní všechno značkování, které mohlo být díky speciálním minimalizačním pravidlům definovaným v SGML, v SGML deklaraci nebo v DTD vynecháno.

Pro stránku proto spočítáme MD5 a snadno tak zjistíme, zda již stejnou stránku nemáme zaindexovanou – měla by stejné MD5. Pro jistotou můžeme ještě stránky se stejným MD5 klasicky porovnat.

Některé stránky jsou na webu dostupné v několika různých kódováních včetně `us-ascii` – bez diakritiky. U každé stránky bychom proto také měli spočítat MD5 jejího textu po odstranění diakritiky. Jen tak můžeme dostatečně důkladně detekovat duplicitní stránky. Při ukládání dáváme samozřejmě přednost stránce s diakritikou před stránkou bez diakritiky – ocení to zejména lingvistické moduly.

Kromě hledání duplicitních stránek nám MD5 umožní snadno zjistit, zda se stránka s daným URL od posledního stažení nezměnila. Při změně obsahu stránky se samozřejmě změní i její MD5.

6.3 Implementace stahování stránek

V předchozím textu jsme došli k závěru, že pro naše potřeby bude nejvhodnější implementovat vlastní program pro stahování a ukládání stránek. V následujícím textu se proto zaměřím na některé zajímavější aspekty implementace robota.

6.3.1 Výběr implementačního prostředí

Abychom mohli vybrat implementační prostředí, musíme si nejprve stanovit požadavky. Pro implementaci systému potřebujeme jazyk, který:

- podporuje HTTP protokol, kterým budeme stahovat stránky;
- umožňuje práci s XML dokumenty;
- podporuje vícevláknové aplikace pro paralelní stahování stránek z několika serverů najednou;
- umožňuje přístup do databáze;
- autor ho alespoň trochu ovládá.

Tato kritéria (s výjimkou posledního) nám výběr příliš neusnadní, protože jim vyhoví každý běžně používaný jazyk. Rozhodl jsem se proto použít Javu, která obsahuje bohaté knihovny a navíc je nezávislá na platformě – výslednou prototypovou aplikaci si bude moci vyzkoušet každý nezávislé na použité platformě.

6.3.2 Architektura modulu pro stahování stránek

Celý stahovací modul je poměrně jednoduchý. Hlavní program po spuštění načte konfiguraci, která nastavuje důležité parametry – stránku, kterou se má zahájit stahování, z kolika serverů se mají stránky stahovat najednou atd. Poté se spustí několik vláken a každé stahuje stránky z jednoho serveru. Před samotným stažením je testováno, zda adresa stránky vyhovuje regulárním výrazům, které definují filtr pro stránky, jež nás zajímají.

Po stažení stránky se na základě jejího typu rozhodne, jak bude zpracována. Zatím je implementována jen třída pro obsluhu HTML kódu. Ta využívá knihovnu JTidy,[6] která umí

[6] `http://sourceforge.net/projects/jtidy`

opravit syntaktické chyby v HTML kódu (jedná se o javovou verzi programu HTML-Tidy) a výsledek pak nabízí pomocí DOM rozhraní. Pomocí DOM parseru se projde celý dokument a získají se z něj všechny odkazy, které se vloží do fronty. Stránka je pak převedena do textové podoby XML dokumentu, spočítají se její výtahy (MD5) s diakritikou i bez ní a stránka se uloží do databáze stažených stránek (předtím se samozřejmě testuje, zda již v databázi neexistuje stejná stránka s jinou URL adresou).

Takto stažené stránky jsou pak ostatním modulům k dispozici pomocí služby, která používá webové rozhraní. Její rozhraní definované pomocí WSDL naleznete v příloze A – *Modul pro stahování stránek*. Samotné zdrojové kódy modulu pro stahování nejsou nijak zajímavé, případní zájemci si je mohou vyžádat přímo od autora.

Kapitola 7

Závěr

Cílem diplomové práce bylo navrhnout a vytvořit nástroj, který by usnadnil navigaci po webových stránkách. Navržené řešení spočívá ve vytvoření speciálního navigačního panelu pro prohlížeč. V tomto panelu se pak zobrazují přídavné informace o stránce, jako podobné stránky, metadata apod.

Výsledkem práce je funkční navigační rozhraní, které integruje výsledky z několika modulů dostupných pomocí protokolu SOAP. Má práce ověřila, že distribuovaná architektura využívající protokol SOAP je použitelná. Kromě navigačního modulu vznikl i modul pro stahování stránek. Pro další rozvoj celého systému RAINBOW je klíčové především vytvoření a integrování dalších analytických modulů.

Literatura

[1] Berka, P. – Sochorová, M. – Svátek, V. – Šrámek, D.: *The VSEved System for Intelligent WWW Metasearch.* In: Rudas, I. – Madarasz, L.: *INES'99 – IEEE Intl. Conf. on Intelligent Engineering Systems.* 1999.

[2] Boag, S. – Chamberlin, D. – Fernandez, M. – Florescu, D. – Robie, J. – Siméon, J. – Stefanescu, M.: *XQuery 1.0: An XML Query Language. W3C Working Draft.* 2002. URL: `http://www.w3.org/TR/xquery/`

[3] Box, D.: *A Brief History of SOAP.* O'Reilly XML.com. URL: `http://www.xml.com/pub/a/2001/04/04/soap.html`

[4] Box, D. – Ehnebuske, D. – Kakivaya, G. – Layman, A. – Mendelsohn, N. – Nielsen, H. – Thatte, S. – Winer, D.: *Simple Object Access Protocol (SOAP) 1.1. W3C Note.* 2000. URL: `http://www.w3.org/TR/SOAP/`

[5] Bradley, N.: *The XML companion.* Addison Wesley Longman Limited, 1998. ISBN 0-201-34285-5.

[6] Bray, T. – Paoli, J. – Sperberg-McQueen, C. – Maler, E.: *Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation.* 2000. URL: `http://www.w3.org/TR/REC-xml`

[7] Bray, T. – Hollander, D. – Layman, A.: *Namespaces in XML.* W3C, 1999. URL: `http://www.w3.org/TR/REC-xml-names/`

[8] Clark, J. – Makoto, M.: *RELAX NG Specification.* OASIS, 2001. URL: `http://www.oasis-open.org/committees/relax-ng/spec-20011203.html`

[9] Clark, J.: *XSL Transformations (XSLT) Version 1.0. W3C Recommendation.* 1999. URL: `http://www.w3.org/TR/xslt`

[10] Cowan, J. – Tobin, R.: *XML Information Set. W3C Recommendation.* 2001. URL: `http://www.w3.org/TR/xml-infoset/`

[11] DeRose, S. – Maler, E. – Orchard, D.: *XML Linking Language (XLink) Version 1.0. W3C Recommendation.* 2001. URL: `http://www.w3.org/TR/xlink/`

[12] DeRose, S. – Maler, E. – Daniel, R.: *XML Pointer Language (XPointer) Version 1.0. W3C Candidate Recommendation.* 2001. URL: `http://www.w3.org/TR/xptr/`

[13] Dubinko, M. – Dietl, J. – Klotz, L. – Merrick, R. – Raman, T.: *XForms 1.0. W3C Working Draft.* 2002. URL: `http://www.w3.org/TR/xforms/`

[14] Fallside, D.: *XML Schema Part 0: Primer. W3C Recommendation.* 2001.
URL: `http://www.w3.org/TR/xmlschema-0/`

[15] Ferraiolo, J.: *Scalable Vector Graphics (SVG) 1.0 Specification. W3C Recommendation.* 2001.
URL: `http://www.w3.org/TR/SVG/`

[16] Fielding, R. – Gettys, J. – Mogul, J. – Frystyk, H. – Masinter, L. – Leach, P. – Berners-Lee, T.: *RFC2616: Hypertext Transfer Protocol – HTTP/1.1.* IETF, 1999.
URL: `http://www.ietf.org/rfc/rfc2616.txt`

[17] Gudgin, M. – Hadley, M. – Moreau, J. – Nielsen, H.: *SOAP Version 1.2 Part 1: Messaging Framework. W3C Working Draft.* 2001.
URL: `http://www.w3.org/TR/soap12-part1/`

[18] Holman, G.: *CD 19757-0 – DSDL Part 0 – Overview. Document Schema Definition Language.* ISO/IEC JTC 1/SC34, 2001.
URL: `http://www.jtc1.org/FTP/Public/SC34/DOCREG/0275.htm`

[19] Le Hors, A. – Le Hégaret, P. – Wood, L.: *Document Object Model (DOM) Level 2 Core Specification. W3C Recommendation.* 2000.
URL: `http://www.w3.org/TR/DOM-Level-2-Core`

[20] Hyatt, D.: *XBL – XML Binding Language. W3C Note.* 2001.
URL: `http://www.w3.org/TR/xbl/`

[21] Hyatt, D.: *XML User Interface Language (XUL) 1.0.*
URL: `http://www.mozilla.org/projects/xul/xul.html`

[22] Christensen, E. – Curbera, F. – Meredith, G. – Weerawarana, S.: *Web Services Description Language (WSDL) 1.1. W3C Note.* 2001. URL: `http://www.w3.org/TR/wsdl`

[23] ISO (International Organization for Standardization): *ISO 8879:1986(E). Information processing – Text and Office Systems – Standard Generalized Markup Language (SGML).* 1986.

[24] Kosek, J.: *XML pro každého.* Grada Publishing, Praha 2000. ISBN 80-7169-860-1. Str. 164.

[25] Kosek, J. – Svátek, V.: *XML a ontologie jako integrační nástroje pro analýzu a zpřístupňování WWW.* Str. 183-192. In: Valenta, J.: *Datasem 2000 Proceedings.* 2000. ISBN 80-210-2428-3.

[26] Kosek, J.: *XUL.* Str. 88-91. In: *Softwarové noviny.* 8/2001. ISSN 1210-8472.

[27] Lassila, O. – Swick, R.: *Resource Description Framework (RDF) Model and Syntax Specification. W3C Recommendation.* 1999.
URL: `http://www.w3.org/TR/REC-rdf-syntax/`

[28] Lawrence, S. – Giles, L.: *Accessibility and Distribution of Information on the Web.*
URL: `http://wwwmetrics.com/`

[29] Mareš, M.: *Sherlock – A Network Document Space Search System.*
URL: `ftp://atrey.karlin.mff.cuni.cz/pub/local/mj/sherlock/papers/sherlock-1.0.ps.gz`

[30] McCaron, S. – Pemberton, S. – Raman, T.: *XML Events. W3C Working Draft.* 2001. URL: `http://www.w3.org/TR/xml-events/`

[31] Megginson, D.: *SAX 2.0.* URL: `http://www.saxproject.org/`

[32] Nielsen, H. – Leach, P. – Lawrence, S.: *RFC 2774: An HTTP Extension Framework.* IETF, 2000. URL: `http://www.ietf.org/rfc/rfc2774.txt`

[33] Pokorný, J.: *Databáze a Web.* Str. 4-22. In: *Moderní databáze 2001.* URL: `http://www.komix.cz/napsal/md2001/md_mffuk.zip`

[34] Raggett, D.: *Clean up your Web pages with HTML TIDY.* URL: `http://www.w3.org/People/Raggett/tidy/`

[35] Raggett, D. – Le Hors, A. – Jacobs, I.: *HTML 4.01 Specification. W3C Recommendation.* 1999. URL: `http://www.w3.org/TR/html401/`

[36] *A Standard for Robot Exclusion.* URL: `http://info.webcrawler.com/mak/projects/robots/norobots.html`

[37] Šimek, P.: *Ontologie a WWW. Diplomová práce.* Praha, VŠE, 1999.

[38] Štumpf, J.: *Systémy řízení výměny zpráv a XML (XML Messaging).* Str. 67-98. In: Bieliková, M.: *Datakon 2001 Proceedings.* 2001. ISBN 80-227-1597-2.

[39] Systinet: *Vývoj Web Services. Praktické ukázky technologií SOAP, WSDL a UDDI.* 2001.

[40] Thai, T. – Lam, H.: *.NET Framework Essentials.* O'Reilly, 2001. ISBN 0-596-00165-7.

[41] The Unicode Consortium: *The Unicode Standard. Version 3.0.* Addison-Wesley, 2000. ISBN 0-201-61633-5.

[42] Uschold, M. – Gruninger, M.: *Ontologies: principles, methods and applications.* Str. 93-136. In: *The Knowledge Engineering Review.* 1996. Vol. 11:2.

[43] *w3mir's homepage.* URL: `http://www.math.uio.no/~janl/w3mir/`

[44] *Wget.* URL: `http://www.gnu.org/software/wget/wget.html`

Příloha A

Modul pro stahování stránek

Stažené stránky jsou ostatním modulům dostupné jako webová služba. Její rozhraní je definováno následujícím WSDL souborem.

Příklad A.10: WSDL soubor služby poskytující stažené stránky

```
<?xml version='1.0'?>
<wsdl:definitions name='cz.vse.rainbow.services.XHTMLService'
  targetNamespace='urn:x-rainbow:services:XHTMLService'
  xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
  xmlns:tns='urn:x-rainbow:services:XHTMLService'
  xmlns:http='http://schemas.xmlsoap.org/wsdl/http/'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns:mime='http://schemas.xmlsoap.org/wsdl/mime/'
  xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
  xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'>
  <wsdl:message name='XHTMLService_getString_Request'>
    <wsdl:part name='p0' type='xsd:string'/>
  </wsdl:message>
  <wsdl:message name='XHTMLService_getString_Response'>
    <wsdl:part name='response' type='xsd:string'/>
  </wsdl:message>
  <wsdl:message
    name='XHTMLService_getString_java.sql.SQLException_Fault'>
    <wsdl:part name='idoox-java-mapping.java.sql.SQLException'
               type='xsd:string'/>
  </wsdl:message>
  <wsdl:portType name='XHTMLService'>
    <wsdl:operation name='getString' parameterOrder='p0'>
      <wsdl:input name='getString'
                  message='tns:XHTMLService_getString_Request'/>
      <wsdl:output name='getString'
                   message='tns:XHTMLService_getString_Response'/>
      <wsdl:fault name='getString_fault1'
        message=
  'tns:XHTMLService_getString_java.sql.SQLException_Fault'/>
```

```
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name='XHTMLServiceSOAPBinding0'
                type='tns:XHTMLService'>
    <soap:binding transport='http://schemas.xmlsoap.org/soap/http'
                  style='rpc'/>
    <wsdl:operation name='getString'>
      <soap:operation soapAction='' style='rpc'/>
      <wsdl:input name='getString'>
        <soap:body use='encoded'
          encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
          namespace='urn:x-rainbow:services:XHTMLService'/>
      </wsdl:input>
      <wsdl:output name='getString'>
        <soap:body use='encoded'
          encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
          namespace='urn:x-rainbow:services:XHTMLService'/>
      </wsdl:output>
      <wsdl:fault name='getString_fault1'>
        <soap:fault use='encoded'
          encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
          namespace='urn:x-rainbow:services:XHTMLService'/>
      </wsdl:fault>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name='XHTMLService'>
    <wsdl:port name='XHTMLService'
               binding='tns:XHTMLServiceSOAPBinding0'>
      <soap:address
        location='http://rainbow.vse.cz:8000/wasp/XHTMLService/'/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Příloha B

Navigační rozhraní

B.1 Navigační modul pro Mozillu

Aby měl navigační modul přístup k hlavnímu oknu prohlížeče, ze kterého zjišťuje informace o právě načtené stránce, musíme navigační panel nainstalovat pomocí instalační technologie XPInstall. Kvůli chybě v Mozille je instalace trochu krkolomná, nicméně funkční. Postup je následující:

1. Nainstalujte si prohlížeč Mozilla z adresy `http://www.mozilla.org`. Rozhraní bylo testováno s verzí 0.9.9. Verze 1.0 RC1 obsahuje chybu, kvůli které navigační panel nefunguje. Čistě teoreticky by mělo vše fungovat s libovolnou verzí Mozilly, ale prakticky je to odzkoušené jen s verzí 0.9.9.

2. V Mozille si otevřete adresu `http://rainbow.vse.cz:8000/rainbow/rainbow.xpi`. Prohlížeč se zeptá, zda může instalovat software. Po potvrzení a instalaci restartujte Mozillu.

3. Vyberte příkaz z menu *Tasks* → *RAINBOW Add Panel*. Opět restartujte Mozillu.

4. V postraním pruhu by měl být vidět další panel – *RAINBOW*. Funguje zcela automaticky, stačí zcela klasicky brouzdat po stránkách a do několika sekund po načtení nové stránky by se měly objevit i informace o stránce v panelu.

Příklad B.11: Navigační panel pro Mozillu

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>

<window xmlns=
  "http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

<script type="application/x-javascript">

// base URL of xulservlet
var baseUrl = "http://rainbow.vse.cz:8000/rainbow/xulservlet?url="

// old URL
```

```
var oldUrl = "";
var newUrl = ""

// reload results when displayed page is changed
function update()
{
  newUrl = window._content.location;

  if (newUrl != oldUrl)
  {
    oldUrl = "" + newUrl; // make copy, not reference
    document.getElementById('results').
      setAttribute('src', baseUrl + newUrl);
 }
}

var timer = window.setInterval("update()", 1000);

</script>

<iframe id="results" src="about:blank" flex="1" />

</window>
```

B.2 Servlet sloužící jako dotazovací modul

Servlet je speciální třída v jazyce Java, která běží v rámci servletové kontejneru. To je speciální web server, který je schopný hostovat právě servlety. Funkční verze servletu je dostupná na adrese `http://rainbow.vse.cz:8000/rainbow/xulservlet`.

Příklad B.12: Servlet generující XUL kód

```
package cz.vse.rainbow.servlets;

// standard and servlet imports
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

// WS imports
import client.iface.*;
import client.iface.struct.*;
import org.idoox.wasp.Context;
import org.idoox.wasp.MessageAttachment;
import org.idoox.webservice.client.WebService;
import org.idoox.webservice.client.WebServiceLookup;
```

```
import org.idoox.webservice.client.WebServiceLookupException;

// servlet which returns information about specific URL
// as XUL document
// URL should be passed by GET method in ”url” parameter
public class XULServlet extends HttpServlet
{
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
    throws IOException, ServletException
    {
      // generate begin of XUL page
      response.setContentType(
        ”application/vnd.mozilla.xul+xml; charset=utf-8”);
      PrintWriter out = response.getWriter();
      out.println(”<?xml version=’1.0’?>\n” +
        ”<?xml-stylesheet href=’chrome://global/skin/’
                          type=’text/css’?>\n” +
        ”<window xmlns=’http://www.mozilla.org/keymaster/
                        gatekeeper/there.is.only.xul’>\n” +
        ”<script type=’application/x-javascript’>\n” +
        ”function openTopWin( url )\n” +
        ”{ window._content.location = url; }\n” +
        ”</script>\n” +
        ”<tree flex=’1’>\n” +
        ” <treecols>\n” +
        ”   <treecol flex=’1’/>\n” +
        ”   <treecol flex=’1’/>\n” +
        ” </treecols>\n” +
        ” <treechildren flex=’1’>\n”);

      String googleHost = ”http://api.google.com/search/beta2”;
      String metadataHost =
               ”http://rainbow.vse.cz:8000/wasp/Metadata/”;
      String lingHost = ”http://rainbow.vse.cz:7878”;

      try
      {
        //init the lookup
        WebServiceLookup lookup =
          (WebServiceLookup)Context.getInstance
            (”org.idoox.webservice.client.WebServiceLookup”);

        // ask Google for related pages
        GoogleSearchPort serviceGoogle = (GoogleSearchPort)lookup.
          lookup(”http://api.google.com/GoogleSearch.wsdl”,
                 GoogleSearchPort.class,googleHost);
        GoogleSearchResult r = serviceGoogle.doGoogleSearch
```

```
  (”...heslo pro přístup ke službě...”,
   ”related:” + request.getParameter(”url”), 0, 10,
   false, ””, false, ””, ””, ””);
out.println(”<treeitem container=’true’ open=’true’>\n” +
            ”<treerow>\n” +
            ”<treecell class=’treecell-indent’
                       label=’Related pages’
                       style=’font-weight: bold’/>\n” +
            ”</treerow>\n” +
            ”<treechildren>\n”);
for (int i=0; i < r.resultElements.length; i++)
{
  String title = escapeXML(r.resultElements[i].title);
  String url = escapeXML(r.resultElements[i].URL);
  if (title.equals(””)) title = url;
  out.println(”<treeitem><treerow>\n” +
              ”<treecell class=’treecell-indent’
                         style=’color: blue’ ” +
              ”label=’” + title + ”’ ” +
              ”onclick=\”openTopWin(’” + url + ”’)\”/>\n” +
              ”</treerow></treeitem>\n”);
}
if (r.resultElements.length == 0)
{
  out.println(”<treeitem><treerow>\n” +
              ”<treecell class=’treecell-indent’
                         style=’color: blue’ ” +
              ”label=’No related pages’/>\n” +
              ”</treerow></treeitem>\n”);
}
out.println(”</treechildren></treeitem>\n”);

// ask Metadata service for metadata
Metadata serviceMetadata = (Metadata)lookup.
  lookup(”http://rainbow.vse.cz:8000/wasp/Metadata/”,
         Metadata.class,metadataHost);
Metalist listtags = new Metalist();
MetalistHolder objmeta = new MetalistHolder();
objmeta.setValue(listtags);
serviceMetadata.getTags(request.getParameter(”url”),
                        objmeta);
listtags = objmeta.getValue();
out.println(”<treeitem container=’true’ open=’true’>\n” +
            ”<treerow>\n” +
            ”<treecell class=’treecell-indent’
                       label=’Metadata’
                       style=’font-weight: bold’/>\n” +
            ”</treerow>\n” +
```

```
            "<treechildren>\n");
    for (int i=0; i< listtags.count; i++)
    {
      String metadata = escapeXML(listtags.metatags[i].name +
                        ": " + listtags.metatags[i].content);
      out.println("<treeitem><treerow>\n" +
                  "<treecell class='treecell-indent' " +
                  "label='" + metadata + "'/>\n" +
                  "</treerow></treeitem>\n");
    }
    if (listtags.count == 0)
    {
      out.println("<treeitem><treerow>\n" +
                  "<treecell class='treecell-indent' " +
                  "label='No metadata found'/>\n" +
                  "</treerow></treeitem>\n");
    }
    out.println("</treechildren></treeitem>\n");

    // ask Ling service for data
    LingService serviceLing = (LingService)lookup.
      lookup("http://rainbow.vse.cz/support/rb-ling.wsdl",
             LingService.class,lingHost);
    String sentences[] = serviceLing.extractSentences
      (request.getParameter("url"));
    out.println("<treeitem container='true' open='true'>\n" +
                "<treerow>\n" +
                "<treecell class='treecell-indent'
                           label='Page indicators'
                           style='font-weight: bold'/>\n" +
                "</treerow>\n" +
                "<treechildren>\n");
    for (int i=0; i<sentences.length; i++)
    {
      out.println("<treeitem><treerow>\n" +
                  "<treecell class='treecell-indent' label='" +
                  escapeXML(sentences[i]) + "'/>\n" +
                  "</treerow></treeitem>\n");
    }
    if (sentences.length == 0)
    {
      out.println("<treeitem><treerow>\n" +
                  "<treecell class='treecell-indent' " +
                  "label='No indicators found'/>\n" +
                  "</treerow></treeitem>\n");
    }
    out.println("</treechildren></treeitem>\n");
  }
```

```
      catch (WebServiceLookupException e)
      {
        out.println("Unable to call web-service.");
      }

      out.println("</treechildren></tree>\n</window>\n");
    }

    // escapes XML significant characters by entities
    String escapeXML (String in)
    {
      String out = "";
      for (int i=0; i< in.length(); i++)
      {
        if (in.charAt(i) == '<')
          out += "&lt;";
        else if (in.charAt(i) == '&')
          out += "&amp;";
        else if (in.charAt(i) == '\'')
          out += "&apos;";
        else if (in.charAt(i) == '\"')
          out += "&quot;";
        else
          out += in.charAt(i);
      }
      return out;
    }
}
```