Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra Informačních Technologií

BAKALÁŘSKÁ PRÁCE

XML a World Wide Web Aplikace

Jan Rázga

Vedoucí bakalářské práce: Ing. Pavel Šimek, Ph.D.

Studijní program: Informatika

Čestné prohlášení

Prohlašuji, že jsem svou bakalářskou práci na téma XML a World Wide Web Aplikace jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 28. dubna 2009 .....

Jan Rázga

Poděkování

Rád bych touto cestou poděkoval vedoucímu bakalářské práce Ing. Pavlu Šimkovi, Ph.D. za cenné rady, připomínky a metodické vedení práce. Poděkování patří také všem, kteří mě při tvorbě baklářské práce podporovali, zejména rodině.

XML a World Wide Web Aplikace

XML and World Wide Web Applications

Anotace

Bakalářská práce se zabývá aplikací formátu XML v prostředí World Wide Web. Zaměřuje se zejména na problematiku podpory tvorby webových služeb v jazycích Java, PHP a v prostředí .NET. V práci jsou představeny současné technologie a prostředky pro tvorbu webových služeb a je uvedena jejich podpora ve zvolených jazycích. Dále je na základě implementace webové služby v prostředí konkrétního jazyka zhodnocena snadnost, s jakou se služba dala implementovat.

Annotation

Bachelor's thesis deals with aplication of XML in World Wide Web. In particular, it focuses on issues related to the support of web service development in Java and PHP programming languages, and .NET environment. The work introduces current technologies and tools for creating web services and presents their support in chosen languages. Furthermore there is made an assessment of ease of web service implementation in particular language, which is based on concrete web service implementation in that language.

Klíčová slova

XML, .NET, Java, PHP, SOA webové služby, SOAP, WSDL, webové aplikace

Keywords

XML, .NET, Java, PHP, SOA, web services, SOAP, WSDL, web aplications

Obsah

1 Úvod

World Wide Web, jakož to celosvětová informační síť, je pro běžného uživatele soustavou zdrojů (HTML stránky, obrázky, videa, soubory atd.), které lze zobrazit, případně jiným způsobem zpracovat, prostřednictvím programového vybavení počítače. Pohled programů a aplikací na internet je odlišný. Programy vyžadují od internetu možnost získávat a posílat data (nebo v nich vyhledávat), a to v určitém formátu, který je pro daný program srozumitelný. Bez obecného standardu není nemožné informace získané touto cestou nějak obecně strojově zpracovávat, proto začaly vznikat technologie, které takovéto služby programům umožňují. Asi nejvýznamnějším prvkem této strojové zpracovatelnosti se stal značkovací jazyk XML, který umožňuje vytvářet dokumenty s jasně danou strukturou, zpracovatelné na všech platformách, tím pádem i ve všech programovacích jazycích.

Značkovací jazyk XML původně vznikl jako univerzální prostředek pro výměnu dat. Pro svou univerzálnost a jednoduchou syntaxi se stal standardem, který se rozšířil a získal uplatnění v mnoha nejen webových technologiích. Kromě zmíněného uplatnění při výměně dat se XML stalo základem sémantického webu, také jej můžeme považovat za databázový model, který rozšiřuje možnosti databázového zpracování zejména hierarchických dat, pro něž je zpracování v relačních databázích prakticky nepoužitelné. V neposlední řadě je na XML založena servisně orientovaná integrace.

Dnešní informační svět spěje k maximální propojenosti informačních zdrojů. Díky existenci internetu, představujícímu soubor technologií umožňující přístup ke vzdáleným

zdrojům prostřednictvím celosvětové sítě, je trendem nejen v oblasti podnikání usnadňovat konzumentům přístup k informacím a službám právě jeho prostřednictvím. K tomuto účelu jsou vyvíjeny nové technologie, které umožňují integraci jednotlivých systému ve stále větší míře. Architektura, na které jsou stavěny moderní Business-to-business (B2B) řešení se nazývá servisně orientovaná a jednou z technologií, které ji využívají, jsou webové služby, založené na komunikaci prostřednictvím XML zpráv. Webové služby (anglicky web services), tedy strojově zpracovatelné webové zdroje, které dokážou XML nejen vytvářet, ale i přijímat, jsou ústředním tématem této bakalářské práce.

2 Cíl práce a metodika

Cílem bakalářské práce je analyzovat podporu XML a webových služeb v nejrozšířenějších programovacích jazycích pro tvorbu XML a webových služeb. Současně si dala práce za cíl výstižně shrnout základní principy XML formátu a jeho zpracování. Dále pak popsat podstatu nejběžnějších technologií v oblasti World Wide Web, založených na bázi XML. Součástí bakalářské práce je také praktická implementace jednoduché webové služby ve zkoumaných jazycích.

Základní metodou při zpracování literární rešerše bylo studium odborné literatury a internetových článků uvedených v seznamu literatury. V této teoretické části je hojně využita metoda citací s odkazem na zdrojovou literaturu. Kromě citací byla použita metoda tvorby vlastního textu na základě konfrontace několika informačních pramenů.

Praktická část je zaměřena na výběr konkrétních jazyků pro tvorbu webových služeb a porovnání možností jejich tvorby. V dnešní době nejvyužívanějšími, tím pádem i nejperspektivnějšími, jazyky pro tvorbu webových jsou jazyky Java, PHP a platforma .NET (zástupcem této platformy byl zvolen jazyk Visual Basic). Podpora tvorby webových služeb byla ověřena implementací jednoduché webové služby ve zvolených jazycích. Součástí implementace byla i instalace softwarových součástí potřebných k naprogramování a zprovoznění webové služby na počítači. Rozhodující kriteria pro výběr programovacích jazyků byla:

- rozšířenost jejich využívání v prostředí World Wide Web,

- s tím související technologické možnosti,
- podpora daného jazyka s ohledem na možnosti dalšího vývoje.

V první kapitole bakalářské práce jsou uvedeny základní principy XML zahrnující syntaxi jazyka a jeho obecné vlastnosti. Dále jsou zde uvedena aplikační rozhraní, která se při zpracování dokumentu využívají. Druhá kapitola zmiňuje a osvětluje současné technologie založené na XML, které se využívají v prostředí World Wide Web. Třetí kapitola je zaměřena na problematiku webových služeb. Kapitola vysvětluje základní pojmy a vysvětluje principy fungování webových služeb. Také jsou zde zmíněny používané architektury a standardy týkající se webových služeb. Čtvrtá kapitola tvoří praktickou část bakalářské práce. Porovnává podporu webových služeb v programovacích jazycích Java, PHP a jazycích založených na platformě .NET. Závěrem je porovnána využitelnost jazyků při vývoji XML aplikací a webových služeb, a zhodnocena jednoduchost implementace webové služby v těchto jazycích.

3 Značkovací jazyk XML

XML je zkratka pro eXtensible Markup Language, tedy rozšiřitelný značkovací jazyk. Formát XML definovalo konsorcium W3C pro přenos obecných dokumentů a dat.

Vznik jazyka XML byl inspirován potřebou univerzálního, pokud možno otevřeného, standardu, který by byl schopen jednoduchým způsobem popsat obecnou strukturu dat. Dalším požadavkem byla přenositelnost jazyka mezi různými platformami s různými národními prostředími a také nezávislost na konkrétním programovacím jazyku.

Pro přenositelnost je u jazyka XML důležitý jeho textový charakter. Na XML soubor tedy není nahlíženo jako na binární strukturu, často závislou na konkrétní platformě, ale jako na posloupnost znaků, jejichž interpretace je jednoznačná. Přenositelnost mezi různými národními prostředími je zajištěna samotnou strukturou XML dokumentu, jejíž součástí je definice použitého kódování.

Mluvíme-li o nevýhodách XML, můžeme zmínit redundanci dat u XML dokumentů. Ta může být v dnešní době palčivá zejména v oblasti nativních XML databází kde je potřeba ukládat velké množství nadbytečných dat. Hlavní příčinou redundance je časté opakování XML značek (tagů) v dokumentech a textový charakter XML formátu, který je oproti binární reprezentaci neúsporný. Tato redundance byla zpočátku překážkou i pro masovější rozšíření XML v oblasti webových aplikací, ale vzhledem k dnešním možnostem a dostupnosti přenosových kapacit, je redundance dat při přenosu textových souborů prakticky zanedbatelná a proto se XML mohlo stát základem mnoha webových technologií.

3.1 Nástupce SGML

Koncept XML vychází z obecnějšího standardu SGML (Standard Generalized Markup Language) a je reakcí na nedostatečnou flexibilitu jazyka HTML (HyperText Markup Language) vytvořeného jazykem SGML. Problémem HTML je nemožnost rozšíření poměrně úzké množiny značek (tagů), které mohou být v dokumentu použity. Tím je použitelnost HTML, jako jazyka pro tvorbu složitějších strukturovaných dokumentů, značně omezena a pro některé konkrétní aplikace je tím jazyk nepoužitelný. potřeba rozšiřitelné množiny značek za současné nenáročnosti údržby aplikací (programátorské i nákladové), kterou jazyk SGML neuspokojoval, vedla ke vzniku nového standardu, jazyka XML. I když XML tvoří pouze zjednodušenou podmnožinu SGML, základní požadavek flexibility byl zachován a přílišná složitost SGML odstraněna.

3.2 XML dokument

XML dokument je určitým způsobem uspořádaná posloupnost znaků jisté abecedy. Implicitně se předpokládá Unicode – kód ISO-IEC 10646. Na rozdíl od formátu HTML rozlišuje XML důsledně velká a malá písmena. [1]

Fyzicky se XML dokument skládá z posloupnosti prvků nazvaných entity. Fyzická entita ještě není logicky element dokumentu. Z hlediska procesoru XML může každá fyzická entita obsahovat buď rozpoznatelná data, nebo nerozpoznatelná data. Nerozpoznatelná data mohou být textová nebo binární, která se buď procesoru nepodaří interpretovat jako znaky, nebo se jedná o data pro jinou aplikaci. Rozpoznatelná data jsou sestavena ze znaků zvolené abecedy a představují buď znaková data nebo značky (markups). Značky vyznačují logickou strukturu dokumentu a tím i jeho rozložení (layout). Dokument začíná entitou nazývanou kořen (root entity). [1]

Logicky se dokument skládá z prologu, deklarací, elementů, komentářů a instrukcí pro zpracování jinými aplikacemi. Logické elementy jsou v dokumentu vyznačeny značkami. Sada značek je obecně libovolná, může být ale předepsána deklaracemi. Deklarace nejsou povinné, pokud ale chceme strukturu dokumentu kontrolovat, musíme ji deklaracemi předepsat. XML je dobře vytvořen, pokud všechny rozpoznatelné entity v dokumentu jsou správně vytvořeny a navíc, všechny rozpoznatelné entity, na které existují v dokumentu odkazy, jsou rovněž dobře vytvořeny. Každá dvojice závorek musí být korektně spárována v rámci elementu a tyto dvojice musí být dobře vnořeny do sebe – závorky se nesmí křížit. Z toho plyne, že dobře vytvořený XML dokument má stromovou strukturu. [1]

3.2.1 Formální struktura

Základní XML struktura se skládá podobně jako dokument HTML z jednotlivých tagů. Tyto jsou podobně jako u HTML vyznačeny lomenými závorkami. Mezi jednotlivými tagy se nachází element. Každý z elementů ještě může mít v rámci svého uvozujícího tagu atributy.

V XML je množina značek neuzavřená. Pro různé dokumenty můžeme definovat různé množiny značek. Značky slouží k označení určitých částí dokumentu, tedy k vyznačení syntaktické struktury dokumentu. Sémantika značek i význam jejich obsahu jsou ponechány na aplikacích zpracovávajících dokument. Definice sady značek může být součástí dokumentu nebo může být specifikována odkazem.

Značky v XML mají otevírací závorku (start-tag) a uzavírací závorku (end-tag). Dvojici je možné nahradit prázdným elementem, pokud je text mezi závorkami prázdný:

<nazev>Element</nazev>	- element obsahující řetězec “Element”
<znacka />	- prázdný element

Pokud se v textu vyskytují znaky, které jsou vyhrazeny pro definice značky, je třeba užít zástupných řetězců. Například '<' zapíšeme jako <. Prvek zapsaný s užitím otazníků <?poznamka?> není standardními XML nástroji zpracováván. Tyto prvky se užívají buď jako prostor pro poznámky autora dokumentu, nebo jako prostor, kam si SW nástroje třetích stran mohou zapisovat svá data. Pak mají tyto poznámky zpravidla tvar <?ProgramXYZ potrebna data?>. Jednotlivé prvky se mohou vzájemně vnořovat, tj. jeden prvek může jako svou součást obsahovat jiné prvky. [11]

Zdrojový kód 3.1: Příklad vnořování v XML

```
<osoba>
  <jmeno>Jan</jmeno>
  <prijmeni>Valter</prijmeni>
  <narozen>1999</narozen>
</osoba>
```

(zdroj: [11])

Při vnořování není dovoleno vzájemně "křížit"značky. Křížením rozumíme stav, kdy je jedna část textu bezprostřední součástí dvou a více prvků. Příkladem takové chyby je následující fragment <jaro>duben květen<leto>červen </jaro>červenec srpen září</leto> kde je text červen současně částí prvku <jaro> i prvku <leto>. Taková konstrukce není v XML povolena. Správně definovaný dokument v XML může na nejvyšší úrovni obsahovat právě jeden prvek. Všechny ostatní prvky do něj musí být vhodným způsobem vnořeny. [11] Následující dokument je nesprávný:

Zdrojový kód 3.2: Nesprávně vytvořený XML dokument

```
<osoba><jmeno>Jan</jmeno></osoba>
<osoba><jmeno>Dan</jmeno></osoba>
<osoba><jmeno>Igor</jmeno></osoba>
```

(zdroj: [11])

Z formálních důvodů je třeba dokument doplnit o prvek, který zmíněné prvky zastřeší:

Zdrojový kód 3.3: Správně vytvořený XML dokument

```
<seznam>
  <osoba><jmeno>Jan</jmeno></osoba>
  <osoba><jmeno>Dan</jmeno></osoba>
  <osoba><jmeno>Igor</jmeno></osoba>
</seznam>
```

(zdroj: [11])

Prvky mohou dále obsahovat tzv. atributy, což jsou dvojice řetězců *název*=”hodnota”. Například:

<auto *typ*=”skoda” *barva*=”cervena” />

nastaví hodnoty atributů *typ* a *barva* elementu *auto*.

3.3 Jmenné prostory

Občas se stane, že je potřeba v jednom XML dokumentu použít značek z více jazyků (dokument používá více definic). V takovém případě může dojít k nejednoznačnosti použití značek, kdy v alespoň dvou použitých jazycích je definována stejně pojmenovaná značka, jež označuje data mající různý význam. Jmenné prostory v XML jsou mechanismem, který dokáže zabránit takovýmto konfliktům.

Značky každého jazyka jsou ve vlastním jmenném prostoru, který je identifikován pomocí jistého URI (Uniform Resource Identifier)[1]. Značky v dokumentu jsou se jmenným prostorem svázány pomocí prefixu, zapisovaného před jméno značky

[1]jednotný identifikátor zdroje

a odděleného dvojtečkou. Prefixu musí být přiřazeno příslušné URI před jeho použitím pomocí atributu tvaru *xmlns:prefix*="URI". Samotný prefix není podstatný, důležité je jen URI jemu přidělené, tedy dva různé prefixy vázané se stejným URI označují stejný jmenný prostor. Jméno značky společně s URI jmenného prostoru tvoří tzv. kvalifikované jméno. [3]

Pro ušetření znaků nemusí jeden ze jmenných prostorů použitých v nějaké části dokumentu mít definovaný prefix, a všechny značky bez prefixu jsou pak automaticky v tomto prostoru. Tento tzv. implicitní jmenný prostor se definuje pomocí atributu *xmlns*="URI", a stejně jako u definic prefixů je jeho rozsah platnosti jen v části dokumentu uvnitř značky nesoucí tento atribut, tedy v různých částech jednoho dokumentu můžou být různé implicitní jmenné prostory. [3]

3.4 XML schéma

XML je metajazyk, a lze s jeho pomocí vytvářet nové značkovací jazyky kladoucí určitá omezení na dokumenty v daném jazyku. Definice použitelné sady značek, definice logické struktury a rozložení dokumentu se nazývá XML schéma dokumentu.

Původně byl pro zápis struktury nově definovaných jazyků navržen jazyk DTD (Document Type Definition), ten však měl jinou syntaxi než XML a neumožňoval zadat typová omezení, např. že obsahem nějaké značky smí být pouze celé číslo. Proto byl standardizován nový definiční jazyk XML Schema [3], který umožňuje popsat strukturu definovaného jazyka a typová omezení. Poskytuje typový systém, který je vhodný pro definice struktury dokumentů, ale který lze bohužel jen velmi obtížně přenést do objektově orientovaných programovacích jazyků. Poskytuje sadu tzv. atomických typů (řetězce, čísla, časové údaje, atd.), seznamy (pole), variantní

typy, složené typy a tzv. typy vzniklé omezením, např. číselný interval nebo řetězec odpovídající nějakému regulárnímu výrazu. Zejména vyjádření typů vzniklých omezením v objektově orientovaných programovacích jazycích je problém, který stále není uspokojivě vyřešen. Proto současné nástroje provádějící převod mezi dokumenty odpovídajícími určitému XML schematu a objekty v paměti nikdy neplní svoji úlohu zcela správně.

3.4.1 Validita XML dokumentu

Každý dobře vytvořený XML dokument může být označen validní, pokud splňuje omezení správnosti (validity constraints) daná specifikací XML schématu. Validita dokumentu proto vyžaduje ověření proti definované specifikaci obsahu. [1] Definice struktury je součástí XML souboru a to buď formou přímé definice, nebo formou odkazu v hlavičce XML dokumentu.

Za validní dokument považujeme takový dokument, který má všechny části, které jsou považovány za povinné, a současně nemá takové části, které nejsou u konkrétního typu dokumentu přípustné. Je ale možné pracovat s volitelnými částmi dokumentu. [11] Validitu dokumentů vyhodnocují nástroje zvané procesory.

3.5 Aplikační rozhraní

(citováno ze zdroje [1]) [2]

K tomu, aby se relevantní data obsažená v XML dokumentu dala použít v konkrétní aplikaci, musí existovat způsob, jak tyto informace zprostředkovat. Prostředníkem mezi aplikací a XML souborem je tzv. procesor. Aby mohly aplikace s XML dokumentem jednoduše pracovat, využívají standardizovaná aplikační rozhraní procesorů. Existují čtyři „generická“ API (Application Programming Interface)[3] k procesorům, která zároveň poskytují jistý datový model XML, ve kterém jsou XML data reprezentována jako strom:

- SAX – Simple API for XML (založené na událostech),
- DOM – Document Object Model (založené na stromech a objektech),
- JDOM – Java Document Object Model (založené na stromech a objektech),
- StAX – Streaming API for XML (založené na proudech dat a událostech).

Veškeré algoritmy konstruující XML strom nebo vracející jeho uzly jsou založeny na sekvenčním čtení XML textu. [...] Protože sekvenční čtení XML dokumentu odpovídá průchodu příslušného stromu do hloubky ve variantě preorder, dostáváme uzly v pořadí odpovídajícímu standardnímu uspořádání.

[2] Mlýnkova, I. - Nečaský, M. - Pokorný, J. - Richta, K. - Toman, K. - Toman, V. Technologie XML: Principy a aplikace v praxi

[3] programátorské rozhraní aplikace

3.5.1 XML procesor

XML procesor (parser) je modul, který umí číst XML dokumenty a zpřístupňuje jejich prvky aplikacím. XML procesor kontroluje, zda je dokument dobře vytvořen (well-formed). Porušení těchto pravidel představuje fatální chybu, kterou musí XML procesor detekovat a hlásit aplikaci. Po detekci fatální chyby může XML procesor poskytovat další data aplikaci, ale nesmí pokračovat v normálním zpracování. XML procesor může být validující – pak umí kontrolovat, zda je vstupní dokument validní (valid) vzhledem k dané specifikaci struktury. Validující XML procesor musí navíc hlásit neshodu s deklarovanou strukturou XML dokumentu.

3.5.2 SAX

Rozhraní SAX (Simple API for XML) představuje ve zpracování XML de facto standard. [...] Jde o rozhraní založené na událostech, tj. situacích, kdy se při sekvenčním čtení zpracovávaného dokumentu narazí např. na:

- počátek dokumentu,
- počáteční značku elementu,
- koncovou značku elementu,
- znaková data,
- instrukci pro zpracování.

SAX informuje o události, kdykoliv vidí uzel pro značku, atribut, text nebo externí entitu. [...] SAX samo o sobě není XML procesorem. Programátor připojuje vlastní

funkce (handlery) pro zacházení s událostmi. Každá událost tedy vyvolá korespondující funkci (metodu), kterou píše programátor. Tento způsob zpracování je tedy proudově orientovaný. Zpracování dává výsledky, aniž by byl k dispozici celý XML dokument. Procesor založený na tomto způsobu zpracování je orientovaný na data (data centric view). Základní myšlenkou je zpracovat element a pak ho „zapomenout“.

3.5.3 DOM

DOM poskytuje objektově orientované rozhraní nezávislé na platformě a jazyku. Při zpracování XML dat generuje procesor ve vnitřní paměti strom, který odpovídá zpracovanému dokumentu. Rozhraní DOM definuje metody pro přístup a (na rozdíl od SAX) také modifikaci stromu.

DOM definuje jazykově nezávislé rozhraní, ale i vazby na různé programovací jazyky jako C++, COBRA, JavaScript a OMG IDL. Standard zahrnuje DOM Level 1,2 a 3. Procesor založený na tomto zpracování je orientovaný na dokument (dokument-centric view). Na rozdíl od SAX lze tedy na DOM založit jednoduchou implementaci jazyků jako XPath či XQuery.

3.5.4 JDOM

Standard DOM je velmi jednoduchá datová struktura, ve které se míchají uzly různých typů, což znepříjemňuje použití DOM v praxi. JDOM na druhé straně pro XML data vytváří strom objektů odpovídajících typů, tj. představuje objektově orientované rozhraní určené speciálně pro jazyk Java. Procesor opět generuje ve vnitřní paměti strom odpovídající vstupnímu dokumentu, který je pro použití jednodušší.

3.5.5 StAX

Procesor s rozhraní SAX API pracuje tak, že zpracovává všechny události bez ohledu na to, zdali ji aplikace potřebuje či nikoliv. Naproti tomu existuje další kategorie proudových API, ve které spíše klientský program „žádá“ XML procesor o další informaci, tj. API je řízeno klientem a nikoliv naopak.

4 Využití XML ve webových aplikacích

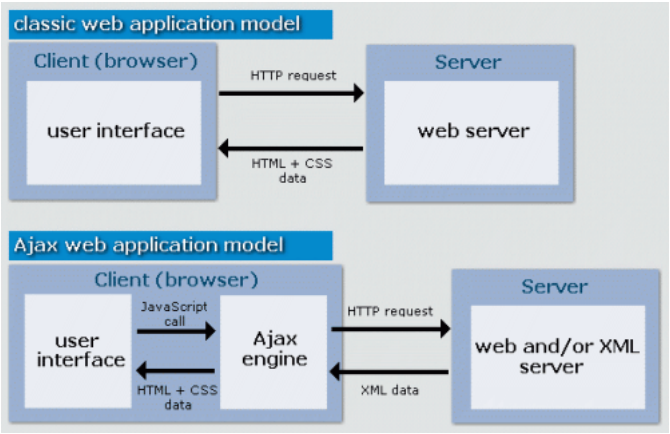
Pro svou jednoduchost a flexibilitu jazyk XML rychle dosáhl značné popularity a stal se základem mnoha technologií současného světa informačních technologií. Jednou z nejvýznamnějších oblastí, kde se jazyk uplatnil, jsou webové aplikace a technologie. Jakožto metajazyk XML umožňuje vytvářet nové struktury „šité na míru“ konkrétním aplikacím. V této kapitole několik z nich zmíníme.

4.1 AJAX

Zkratka AJAX znamená Asynchronous JavaScript and XML (asynchronní JavaScript a XML). AJAX není sám o sobě implementací technologie či softwarovým produktem, ale jedná se o obecný koncept nebo lépe návrhový vzor pro RIA (Rich Internet Application). [15] AJAX je představitel cesty využívající maximální možné hodnoty dnešních technologií. AJAX ve skutečnosti není žádnou novou technologií, pouze novou kombinací technologií již známých, tj. HTML (nebo XHTML), JavaScriptu, XML a XMLHttpRequest. Je to koncept využití těchto technologií, a to především XMLHttpRequest k volání serveru. Bližší srovnání klasického pří-

stupu oproti AJAXu ilustruje následující obrázek:

Obrázek 4.1: Porovnání klasického konceptu s AJAXem, (zdroj: [15])



4.1.1 XMLHttpRequest

Základním stavebním kamenem je objekt XMLHttpRequest, který umožňuje asynchronní volání serveru. V klasickém webovém modelu každá změna stavu na klientu vyžaduje obnovení celého uživatelského rozhraní. Nejdříve je tedy žádost o změnu stavu, odeslání požadavku na server, vyřízení požadavku a vše končí zasláním kompletního uživatelského rozhraní s daty, přičemž jednotlivé kroky jsou vzájemně synchronizovány. Naopak AJAX, díky XMLHttpRequest, může vyvolat libovolný počet nezávislých požadavků, jejichž výsledky mohou ovlivnit pouze patřičné části uživatelského rozhraní, bez nutnosti jeho celkového znovunačítání. Tedy, žádost o změnu stavu, vygenerování požadavku přes XMLHttpRequest, vyřízení požadavku serverem a zpracování vrácené odpovědi XMLHttpRequestem a změna patřičné části uživatelského rozhraní (logika obsluhující XMLHttpRequest je na obrázku znázorněna jako AJAX engine). [15]

4.2 RSS

RSS je zkratkou pro Really Simple Syndication (0.9x) nebo RDF Site Summary (RDF - Resource Description Framework - standardizován organizací W3C). RSS je v podstatě dialekt XML, který umožňuje publikování seznamu odkazů spolu s dalšími informacemi, které blíže popisují daný odkaz. Zdrojový soubor RSS (tzv. RSS kanál) je uloženo na serveru (případně může být generováno dynamicky) a je přístupné návštěvníkům webu. RSS kanál se stal v současnosti nedílnou součástí téměř každého zpravodajského serveru nebo weblogu.

Soubor RSS musí dodržovat pravidla XML dokumentu. RSS má několik povinných značek, které každý soubor musí obsahovat. Další značky jsou nepovinné. Povinnými značkami jsou:

<channel>	- informace o daném kanále
<description>	- textový popis pro item, channel, image a textinput
<language>	- specifikace jazyka daného kanálu (cs pro češtinu)
<link>	- URL dané položky
<title>	- textová identifikace zdroje

Alternativou k rozšířenému RSS se stal ATOM, který je stejně jako RSS založen na publikovaní XML dokumentu s obsahem na webu a vychází ze zkušeností z RSS.

4.3 Sémantický web

(citováno ze zdroje [9]) [1]

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

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

Jde tedy o to, aby data prezentovaná na internetu měla přesně definovaný význam a dovolovala do značné míry automatizované (strojové) zpracování.

[1] Matulík, P. – Pitner, T. Sémantický web a jeho technologie.

4.3.1 Hlavní prvky sémantického webu

Jedním ze základních kroků k vytvoření sémantického webu je konceptualizace dat dostupných na internetu. Jedním z klíčových nástrojů konceptualizace jsou ontologie. Ontologie lze charakterizovat jako formalizované reprezentace znalostí určené k jejich sdílení a znovupoužití. Ontologie jsou často doménového (oborového) zaměření a bývají konstruovány jako pojmové (konceptuální) hierarchie nebo sítě.

(Polo)automatizované zpracování informací v sémantickém webu může být realizováno pomocí softwarových agentů, což jsou do určité míry autonomní inteligentní programové komponenty pohybující se obvykle v distribuovaném prostředí a schopné realizovat "na účet toho, kdo je pověřil"požadavky na vyhledávání informací, realizaci transakcí apod.

Důležitým předpokladem sémantického webu je rovněž standardizovaný popis webových zdrojů. Zdrojem se v této souvislosti rozumí cokoliv, co je dosažitelné prostřednictvím sítě WWW, tedy textové dokumenty, obrázky, videosekvence, zvukové soubory apod. Každý zdroj by byl vybaven stejnými charakteristikami (autor, typ zdroje, klíčová slova atd.), což by umožnilo uživatelům internetu pracovat se sítí WWW jako s relační databází a dotazovat se na její obsah prostřednictvím jazyků podobných SQL. Významným důsledkem by například byla velmi vysoká přesnost a relevance odpovědi na vyhledávací dotaz, což znamená, že by byl uživateli při vyhledávání určité informace vrácen seznam všech zdrojů, které se této informace týkají, a žádný zdroj navíc.

4.3.2 Metadata

Metadata mohou být stručně definována jako (strukturovaná) data o datech. Zachycují obsah, kontext a strukturu dat, která popisují. Síťová metadata, na která

se zaměříme, jsou nejčastěji zapisována prostřednictvím XML, které svými vlastnostmi nejvíce odpovídá požadavkům na otevřenost, přenositelnost a interoperabilitu formátu pro výměnu a ukládání dat.

Pro vyjádření vztahů mezi jednotlivými metadatovými prvky a schématy byl navržen standard RDF (Resource Description Framework) a skutečné zachycení sémantiky popisovaných dat je zajištěno prostřednictvím klasifikačních schémat a řízených slovníků.

4.4 WML

Jazyk WML (Wireless Markup Language) je značkovací jazyk založený na XML umožňující tvorbu online dokumentů pro mobilní zařízení. WML stránky jsou obdobou HTML stránek v mobilních zařízeních.

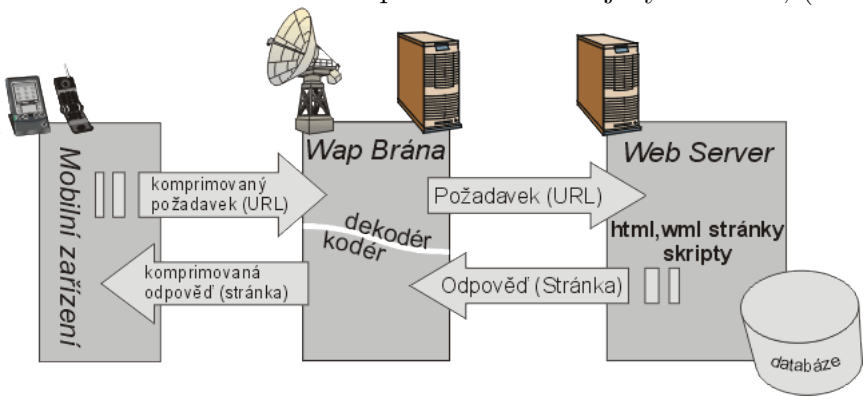
Díky omezenosti zobrazení a nemožnosti přesného umístění prvků HTML stránek na displeji mobilních zařízení vznikl jazyk WML. WML minimalizuje objem dat, která se přenášejí přes internet do mobilního zařízení a zjednodušuje zobrazení stránek.

Komunikace mezi webovým serverem a mobilním zařízení probíhá prostřednictvím WAP brány poskytovatele mobilní telefonní sítě. Toto brána převádí požadavky mobilního zařízení na webovou stránku v požadavky na webový server a zprostředkovává WML stránky mobilnímu zařízení.

4.4.1 WML jazyk

Jazyk WML je založen na jazyce XML. Každá značka tedy musí být párová a navíc napsaná malými písmeny - hned dvě zpřísnění oproti HTML. Pro značky, které nemají párový ekvivalent, existují tzv. prázdné značky (viz jazyk XML, např.
).

Obrázek 4.2: Schéma komunikace prostřednictvím jazyka WML, (zdroj: [8])



Každá WML stránka se skládá z několika karet (značka card). Při nahrání stránky z určité adresy do mobilního zařízení je zobrazena první karta. Jelikož je displej malý, nemůže obsahovat rozmanité strukturované informace. Díky systému karet je však možné na každou kartu umístit určité údaje a vzájemnými odkazy mezi nimi přecházet, aniž by mobilní zařízení muselo znovu kontaktovat server a žádat o stránku.

Protože každá karta může obsahovat ovládací prvky s akcemi, které jsou většinou pro všechny karty společné, je možno společné akce nadefinovat pomocí značky template pro všechny karty na začátku zdrojového textu.

Zdrojový kód 4.1: Ukázka WML dokumentu

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card id="HTML" title="HTML Tutorial">
    <p>
      Our HTML Tutorial is˜an award winning tutorial from W3Schools.
    </p>
  </card>
  <card id="XML" title="XML Tutorial">
    <p>
```

```
    Our XML Tutorial is~an award winning tutorial from W3Schools.
   </p>
  </card>
</wml>
```

(zdroj: [14])

5 Webové služby

Jako další perspektivní aplikaci formátu XML v oblasti World Wide Web můžeme jmenovat webové služby (anglicky web services). Vznik webových služeb se pojí s potřebou obecnějšího řešení při komunikaci v distribuovaných systémech, tedy systémech, jejichž části běží na různých počítačích spojených komunikační sítí. Webové služby jsou postavené na tzv. architektuře SOA.

5.1 Architektura orientovaná na služby (SOA)

V poslední době je jedním ze základních informatických problémů integrace různých softwarových komponent. Ukazuje se, že pro úspěšné zvládnutí tohoto problému je nutná dostatečná abstrakce implementace těchto komponent. Zjednodušeně řečeno je takové abstrakce dosahováno tak, že každá komponenta definuje rozhraní, které popisuje, v jaké podobě komponenta přijímá data od ostatních komponent a v jaké podobě naopak data ostatním komponentám poskytuje. Komponenty pak komunikují výměnou zpráv, jejichž podoba je dána jejich rozhraními. Závislost mezi jednotlivými komponentami pak existuje pouze na úrovni jejich rozhraní a ne na úrovni jejich implementace. Pro takto volně provázané komponenty se v odborné terminologii používá termín loosely-couplet components a výsledné architektuře se dnes

již běžně říká architektura orientovaná na služby (Service Oriented Architecture – SOA). V této souvislosti nazýváme komponenty architektury služby. Každá služba poskytuje určitou funkcionalitu a skládáním služeb můžeme realizovat složitější procesy. [1]

Jednoduchou definicí architektury může být: „SOA je rámec pro integraci obchodních procesů a podpůrné IT infrastruktury ve formě bezpečných, standardizovaných komponent – služeb, které mohou být znovupoužitelné a komponované tak, aby adresovaly měnící se obchodní priority a požadavky.“ [17]

SOA tak přináší řadu výhod, za ty nejvýznamnější jsou považovány především následující:

- *interoperabilita*, tj. možnost propojovat služby nezávisle na platformách, jazycích či operačních systémech, ve kterých jsou implementovány,
- *znovupoužitelnost*, tj. schopnost opakovaně využívat již existující služby v různých složitějších procesech,
- *přehled* o tom, jak jsou složitější procesy realizovány pomocí jednotlivých služeb poskytujících jednodušší funkcionalitu,
- *agilita*, tj. schopnost rychle implementovat nové požadavky a procesy a jejich změny. [1]

5.2 Předpoklady pro webovou službu

SOA lze s výhodami aplikovat i v otevřených prostředích webu. To je přirozená myšlenka, protože webové protokoly jako např. HTTP umožňují snadno vyměňovat zprávy mezi službami na webu. Takto můžeme zpřístupnit přes web funkcionalitu

realizovanou danou službou a vytvářet dynamické systémy, kde jsou služby realizovány službami různých poskytovatelů. Je také možné tyto služby nahrazovat jiným v případě jejich výpadku či nevýhodnosti jejich použití v dané situaci. V takové architektuře ale musíme být schopni především:

1. nalézt službu odpovídající našim požadavkům,
2. zjistit formát zpráv vyžadovaných nalezenou službou,
3. vyměňovat si s nalezenou službou zprávy v předepsaném formátu.

V tak otevřeném prostředí, jakým je web, je nutné tyto činnosti standardizovat. Standardizaci pro webové prostředí představují webové služby (web services). [1]

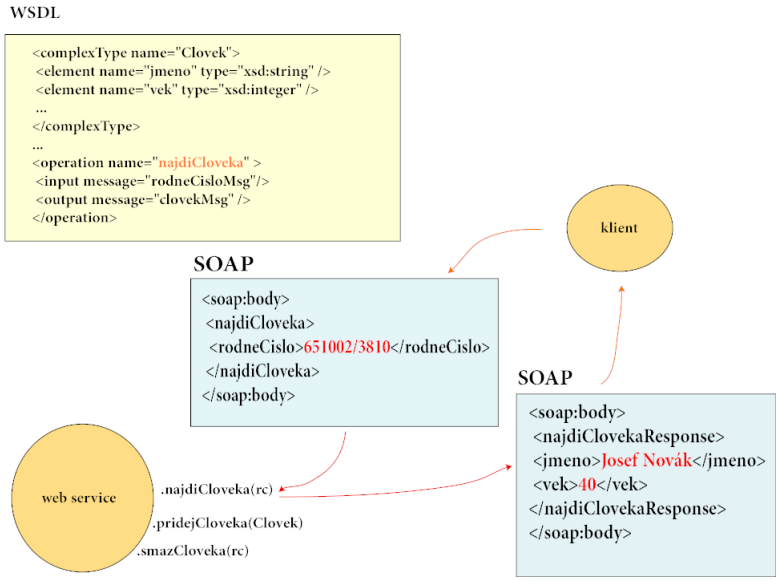
Pojmem webová služba označujeme výpočetní entitu přístupnou přes web prostřednictvím přesně definovaného, implementačně nezávislého rozhraní. Pokud chceme v tomto pojetí zpřístupnit službu zákazníkům/klientům přes web, pak ji můžeme zpřístupnit jako webovou službu. To znamená implementovat softwarovou komponentu realizující tuto službu a doplnit ji o rozhraní, které ke službě umožní přistupovat. [1]

5.3 Definice webové služby

Podle dokumentu W3C, sepsaného Pracovní skupinou pro architekturu webových služeb, zní definice webové služby takto:
Webová služba je softwarový systém zkonstruovaný k podpoře interakce mezi stroji přes síť. Má rozhraní popsané ve strojově zpracovatelném formátu (specificky WSDL). Ostatní systémy interagují s webovou službou způsobem předepsaným jejím popisem

za pomoci SOAP zpráv, typicky dopravovaných použitím HTTP s XML zápisem v součinnosti s ostatními webovými standardy.

Obrázek 5.1: Schéma komunikace SOAP webové služby, (zdroj: [3])



Tato definice je kompromisní, ve skutečnosti představy o webových službách sahají od podnikových messagingových systémů zasílajících XML zprávy mezi službami (tedy nic společného s webem), až po služby přístupné přes web server pomocí zasílání čistého XML (tedy žádné SOAP). [3]

Výhody webových služeb oproti jiným systémům vyplývají především z použití XML – nevyskytují se problémy se zápisem českých znaků či znaků z různých jazyků, díky důslednému používání znakové sady UNICODE; interakce je nezávislá na programovacím jazyku, objektové orientovanosti či platformě; vstupní bariéra je nízká; jsou vhodné pro volně vázané systémy, kdy klient a server o sobě musí předpokládat jen naprosté minimum. Nevýhodou je prostorová neúspornost a pomalost syntaktické analýzy XML. [3]

5.4 SOAP

SOAP (Simple Object Access Protocol) je jedním z protokolů pro výměnu zpráv mezi webovými komponentami a je založený na XML. To znamená, že každá SOAP zpráva je XML dokument se strukturou popsanou protokolem SOAP. Díky nezávislosti na konkrétní platformě či jazyku je hojně využíván pro výměnu zpráv mezi webovými službami. Za jeho nevýhody je někdy považována přílišná „upovídanost“. [1]

5.4.1 Struktura SOAP Zprávy

Kořenový element zprávy SOAP je element Envelope. V něm především uvádíme jmenné prostory. Obsah kořenového elementu má dvě části, hlavičku a tělo, reprezentované podelementy Head a Body. Obsah hlavičky a těla není protokolem specifikován, to už je záležitost aplikace protokolu. Hlavička je nepovinná. Obsahuje obvykle pomocné údaje jako např. identifikační či autentifikační data, transakční data či různé specifičtější informace o kontextu pro zpracování zpráv. [1]

Tělo SOAP zprávy (element Body) je povinné. Tělo tvoří nejdůležitější část zprávy, protože obsahuje informace identifikující volanou službu a předávané parametry, resp. návratové hodnoty služby. K identifikování jednotlivých částí XML zprávy používá SOAP jmenné prostory.

SOAP zprávy lze přenášet podle různých přenosových protokolů. Nejběžnější je protokol HTTP. [1]

Zdrojový kód 5.1: HTTP požadavek klienta na službu zprávou SOAP

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn
```

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

(zdroj: [3])

Zdrojový kód 5.2: HTTP odpověď serveru zprávou SOAP

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

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

(zdroj: [3])

Jde o volání operace s názvem jePrvocilo, ve jmenném prostoru urn:mojeURI, která má jeden vstupní parametr nazvaný cislo. Prvních pět řádků jsou HTTP

hlavičky. Celá SOAP zpráva je zabalena ve značce Envelope, ve které jsou umístěny i definice jmenných prostorů. Samotný obsah volání je uvnitř značky Body. [3]

5.5 WSDL

WSDL je jazyk vyvinutý jako standardizovaný popis rozhraní webových služeb je jazyk založený na XML. WSDL vymezuje množinu abstraktních operací, jaké jsou operace povahy (zda jsou obousměrné či pouze jednosměrné), jak tyto operace volat, jaké mají vstupní a výstupní parametry a jakých jsou tyto datových typů. Současně určuje, kde je WS dostupná a jaké mapování (na jaký protokol) se používá pro definované abstraktní zprávy. [13]

WSDL popis webové služby je opět XML dokument s předepsanou strukturou. Kořenovým elementem je element description. V něm zavádíme potřebné jmenné prostory. [1]

Pomocí elementu <import> se do dokumentu importují externí definice, tedy jiné WSDL soubory. U složitých, rozsáhlých WS tak můžeme zlepšit jejich čitelnost a správu. Element <types> definuje elementy a jejich datové typy použité v definici, čímž současně deklaruje datové typy parametrů operací. [13] Elementy <message> určují, jaká data jsou předávána při komunikaci konzumenta s webovou službou. Současně je propojují s XML elementy uvedenými v <types> (to v případě použitého modelu document exchange) nebo přímo nastavují jejich datový typ (v případě RPC modelu). V elementu <portType> se udávají operace, které endpoint WS poskytuje. K daným operacím jsou pak přivázány (pomocí elementů <input> a <output>) vstupní a výstupní zprávy. Jednosměrné operace se stanovují jednoduše vypuštěním jednoho z elementů. Propojením operací se zprávami si tak operace přiřazují své

vstupní a výstupní parametry. [13]

Element <binding> definuje, jaký binding je pro které operace použit, tedy jaký protokol je použit pro přenos jejich dat. WSDL specifikace popisuje HTTP, SOAP a MIME binding, přičemž ale vývojářům nebrání stanovit si mapování vlastní. Nicméně, jak bude uvedeno v části věnované interoperabilitě, WS-I stejně povoluje pouze SOAP, případně MIME mapování (a to pouze SwA). [13]

Kde se webová služba nalézá (obvykle pomocí URL, na kterém je dostupná protokolem http) nakonec říká element <service>.

5.6 UDDI

S rostoucím významem webových služeb a rapidním nárůstem jejich množství v celém Internetu se objevil problém, jak vytvořenou webovou službu publikovat, jak informovat Internet o její existenci a jak naopak v případě potřeby rychle vyhledat webovou službu podle našich požadavků. [13]

UDDI (Universal Description, Discovery and Integration) je vlastně framework pro adresářové služby. Obsahuje registr, kde se mohou registrovat různí poskytovatelé služeb a klienti mohou služby vyhledávat. UDDI standardizuje všechny činnosti s tím spojené. Registr je rozdělen tří částí:

- *bíle stránky* (white pages), kde jsou obsaženy základní informace o poskytovatelích služeb (např. Jméno, kontakt atd.),
- *žluté stránky* (yellow pages), kde jsou obsaženy podrobnější informace o poskytovatelích (např. Zařazení poskytovatele do různých kategorizačních schémat průmyslu atd.),

- *zelené stránky* (green pages), kde jsou obsaženy popisy samotných webových služeb nabízených poskytovateli (popisy webových služeb určených pro jejich vyhledávání a samozřejmě jejich WSDL popisy).

K registru můžeme přistupovat různými způsoby. Pro nás je zajímavé, že může fungovat jako webová služba, která je popsána standardizovaným WSDL popisem a se kterou můžeme komunikovat pomocí výměny XML zpráv. Rozhraní registru definuje několik různých operací pro manipulaci s registrem, jako např. save_service pro ukládání informací o webových službách do registru nebo find_service pro vyhledání webových služeb v registru splňujících daná kritéria. [1]

5.7 WS-I

Výhodou webových služeb oproti jiným technologiím pro síťovou komunikaci je jejich schopnost spolupracovat s heterogenními systémy, platformami, aplikacemi a programovacími jazyky. Tento požadavek na webové služby, být interoperabilní, však nemusí být vždy automaticky splněn. Díky velkému počtu a někdy i složitosti původních specifikací týkajících se webových služeb může dojít k jejich nesprávnému výkladu a implementaci, která ve výsledku nebude schopná spolupráce s jinými aplikacemi. Dosažení interoperability se ztěžuje i zaváděním dalších specifikací, které obohacují funkcionalitu webových služeb. [13]

Částečným východiskem pro zajištění interoperability jsou ustanovení organizace WS-I (Web Services Interoperability), která vydává zpřesňující specifikace (tzv. profily). Tyto profily obsahují pravidla týkající se nejasných pasáží specifikací z ruzných oblastí webových služeb (protokoly, zabezpečení, přílohy zpráv, atd.). Stěžejní specifikací této organizace je WS-I Basic Profile. [3]

5.8 REST

Specifikace SOAP a WS-* jsou vhodnější pro podniková prostředí, kde jsou zprávy zpracovávány mezilehlými prostředníky či jejich putování systémem je určováno dynamicky. Pro některé webové služby (zejména služby používající cyklus dotaz-odpověď) architektura založená na protokolu SOAP zbytečně složitá. Proto vznikly alternativy, které v konkrétních aplikacích některé nedostatky SOAP webových služeb mohou odstranit. [3]

Jednou z možností, jak zjednodušit komunikaci aplikací je použití vzoru POX (Plain Old XML), který spočívá v zasílání čistých XML dat, bez „zbytečné obálky“ (SOAP). V dnešní době asi zajímavější je použití architektury REST.

Architektura REST se oproti „klasické“ architektuře založené na SOAP a WS-* specifikacích vyznačuje svou jednoduchostí. Zjednodušeně řečeno se jedná o sadu architektonických principů k tomu, jak definovat, adresovat a používat informační zdroje na Internetu. REST představuje geniálně elegantní a jednoduchý způsob, jak realizovat dokumentově orientované webové služby, a to bez komplexnosti protokolů SOAP/WSDL. Důležité je i to, že v podstatě jakýkoliv moderní webový prohlížeč se velmi snadno stane REST klientem. Naopak chtít po webovém prohlížeči, aby přímo komunikoval s „klasickou”SOAP webovou službou, není nejlepší krok. [17]

Základní myšlenkou REST (REpresentational State Transfer) architektury je, že každý zdroj na webu je identifikován pomocí URI a má reprezentaci, např. XML dokument, obrázek, atd. S touto reprezentací může být nakládáno pomocí pevné sady operací, např. čtyř základních HTTP metod, PUT, GET, POST a DELETE na tomto URI. [3]

5.9 Bezpečnost

Bezpečnost webových služeb lze řešit na dvou úrovních, na úrovni zpráv nebo na úrovni transportní vrstvy. [3]

Snadnější, rychlejší a historicky starší je řešení na úrovni transportní vrstvy, kdy SOAP a HTTP komunikace probíhá nad SSL (Secure Socket Layer) šifrovaným a autentizovaným spojením. Toto řešení má jako výhody rychlost a značnou robustnost, protože SSL existuje již dlouhou dobu. Má však také nevýhody. SSL je ze své podstaty dvoubodové spojení, které chrání komunikaci před odposlechem a změnou po cestě, ale neumožňuje digitálně podepisovat zasílaná data, tedy zpětně dokázat, kdo co zaslal (non-repudiation). Jako dvoubodové spojení také neumožňuje použít složitější architektury než je klient-server, tedy zpracování mezilehlými prostředníky. [3]

Druhým řešením bezpečnosti webových služeb je řešení na úrovni SOAP zpráv. SOAP zpráva je XML dokument, je proto možné použít W3C standardy XML Encryption a XML Signature pro šifrování resp. podepisování částí zpráv nebo celých zpráv. Standardizační organizace OASIS vydala specifikaci WS-Security, která stanovuje, jakým způsobem podepisování a šifrování na úrovni SOAP zpráv používat, a jak využít stávající bezpečnostní mechanismy (PKI, Kerberos, hesla atd.) v SOAP zprávách. [3]

Bezpečnost na úrovni zpráv má tedy opačné výhody a nevýhody než bezpečnost na úrovni transportní vrstvy. Umožňuje sice podepisování zpráv a složité scénáře s prostředníky při zpracování zpráv, není však ještě dost vyzrálá a i rychlost zpracování je zatím nižší než u SSL. [3]

6 Podpora tvorby webových služeb v programovacích jazycích

Webové služby založené na výměně XML zpráv pomocí standardních internetových protokolů jsou nezávislé na platformě, proto je možné jednotlivé komponenty distribuovaných systémům, které mezi sebou komunikují na základě popsaného rozhraní, vyvíjet v různých programovacích jazycích aniž by bylo ovlivněno jejich použití.

V této kapitole se zaměříme na vývoj webových služeb v prostředí .NET, Javy a PHP. Tvorba webové služby zahrnuje:

- vytvoření samotné webové služby,
- její publikovaní na serveru,
- vytvoření klientské aplikace, která bude se serverovou službou komunikovat.
- Java Web Services Metadata (JSR 181).

Klientská aplikace může mít buďto formu webové aplikace umístěné na serveru nebo desktopové aplikace spuštěné z počítače. V obou případech klient komunikuje se službou zasíláním zpráv prostřednictvím standardních internetových protokolů.

6.1 prostředí pro tvorbu webových aplikací

Tato kapitola se zaměřuje na popis problematiky vývoje zejména webových služeb ve vybraných programovacích jazycích a shrnuje možnosti použití volně dostupných nástrojů k jejich tvorbě.

6.1.1 .NET

.NET je označení pro technologie založené na .NET frameworku, platformě firmy Microsoft pro vývoj desktopových i webových aplikací, mimo jiné i webových služeb. Aplikace lze psát v několika programovacích jazycích, které mají v .NET frameworku společný základ. Podporované jazyky jsou C#, C++ a Visual Basic. .NET Framework je standardně součástí operačního systému Windows, ale existují i open source implementace založené na .NET frameworku (např. MONO) umožňující vývoj aplikací pod jinými operačními systémy.

Základem .NET frameworku jsou Common Language Runtime (CLR) a .NET framework class library (tedy knihovny .NET frameworku). Knihovny .NET frameworku obsahují celou řadu typů objektů (pro všechny podporované jazyky) pro vývoj nejrůznějších aplikací, počínaje desktopovými a konče aplikacemi založenými na nejnovějších technologiích ASP.NET (jako jsou Web Forms a XML Web Services). Existence Common Language Runtime zajišťuje možnost nezávislého vývoje aplikací ve všech podporovaných programovacích jazycích. CLR tvoří jakéhosi agenta, který se stará o vykonávání kódu a poskytuje aplikacím základní služby systému.

Základem pro tvorbu webových služeb v prostředí .NET je součást frameworku nazvaná ASP.NET (nyní ve verzi 2.0), která obsahujíce implementace všech protokolů a služeb potřebných k vývoji webových aplikací i webových služeb.

V souvislosti s vývojem .NET webových aplikací na platformě Windows je vhodné použít jako server IIS (Internet Information Services), který je součástí operačního systému Windows. vývojové prostředí Microsoft Visual Studia nabízí integrovanou součinnost s touto službou, takže není nutná prakticky žádná konfigurace. Navíc je služba IIS součástí průvodců, což do jisté míry usnadňuje vývoj aplikací.

V případě potřeby využití datového zdroje při tvorbě aplikace se nabízí mnoho možností jakou databázi využít a jak k ní přistupovat. Jednoduchým způsobem v prostředí .NET je využití databázového serveru SQL Server ve volně dostupné verzi Express Edition. Pro přístup k datům je vhodné použít komponenty .NET frameworku ADO.NET, která poskytuje standardizované rozhraní pro přístup k různým typům databází.

6.1.2 Java

Java jako platforma zastřešuje různé varianty použití programovacího jazyka Java pro vývoj nejrůznějších typů aplikací. Platforma Java, na rozdíl od platformy .NET, staví zejména na nezávislosti na operačním systému a otevřeném vývoji. Platformní nezávislost je zajištěna existencí Java Virtual Machine (JVM), tedy jakéhosi virtuálního stroje (sada programů), který spouští všechny zdrojové kódy zkompilované do Java bytecodu. Otevřenost reprezentují zejména technologie, založené na platformě Java, které jsou dostupné pod open source licencí, a tudíž je možno se na jejich vývoji aktivně podílet. Navíc v prostředí Javy často existuje možnost rozhodnutí o volbě produktu, protože mnoho komunit vyvíjí Java software a každý a záleží jen na uživateli, kterou alternativu si vybere.

Souborem základních nástrojů pro tvorbu a spouštění Java aplikací je Java Development Kit (JDK). Základní API pro práci s webovými službami je obsaženo

již v základním balíku Java SDK (Software Development Kit), který je označován jako Java Platform, Standard Edition (Java SE). Nadstavbou Java SE je potom platforma určená k vývoji webových apliakcí všeho druhu, která je označována jako Java Platform, Enterprise Edition (Java EE). Obsahuje podporu vývoje i složitějších podnikových řešení založených na architektuře SOA s širokou podporou standardů Webových služeb.

Java SE 6 obsahuje následující technologie (založené na Javě) pro práci s webovými službami:

- Java API for XML Web Services (JAXWS),
- Java Architecture for XML Binding (JAXB),
- SOAP with Attachments API for Java (SAAJ),
- Java Web Services Metadata (JSR 181).

Implementace jednotlivých technologií jsou obsaženy v Java balících (packages), které zprostředkovávají jejich funkčnost formou tříd (class). K použití je tedy potřeba vytvořit instanci třídy, která zvolenou technologii implementuje. Dnešními technologiemi pro tvorbu dynamických webových stránek na platformě Javy jsou Servlety a JSP (JavaServer Pages).

Servlety slouží vývojářům k jednoduššímu rozšiřování funkcionality webových serverů a umožňují přístup k existujícím business systémům. Na servlety lze nahlížet jako na aplety běžící na straně serveru bez možnosti grafického rozhraní. Servlety pracují na bázi požadavek – odpověď a jsou hojně využívány při tvorbě webových služeb. Server, který zprostředkovává komunikaci ze servlety se nazývá servlet kontejner a asi nejpoužívanějším je v dnešní době Apache Tomcat vyvíjeny konsorciem Apache Software Foundation.

JSP je technologie, která umožňuje tvorbu dynamického obsahu na straně serveru kombinací formátovacích značek a příkazů Javy. Tvoří v podstatě vyšší vrstvu abstrakce servetů, protože při prvním volání JSP dokumentu je vytvořen Java program, který dále funguje stejně jako servlet.

Jako engine pro vývoj webových služeb je dnes často používán Apache Axis2 vyvíjený konsorciem Apache Software Foundation. Apache Axis je implementací SOAP protokolu podle specifikace konsorcia W3C. Axis2 nabízí kromě podpory existujících verzí SOAP protokolu také podporu REST/POX a REST architektury standardů WS-* a dalších technologií z oblasti webových služeb. Kromě toho umožňuje jednoducho rozšiřitelnost formou modulů.

Apache Axis2 je dostupný ve dvou implementacích: Apache Axis2/Java and Apache Axis2/C. V souvislosti s Javou je Axis2 často používán jako Java servlet v kombinaci s Tomcat servlet kontejnerem, ale je ho možné použít i v samostatném módu.

Co se týče vývojových prostředí, tak mezi nejpoužívanější prostředí pro tvorbu Java aplikací patří Eclipse, vyvíjené open source komunitou Eclipse Foundation a založené firmou IBM, a prostředí NetBeans, vyvíjené také jako open source a založené firmou Sun Microsystems.

6.1.3 PHP

Skriptovací jazyk PHP je jednostranně zaměřen na běh na serveru, bez kterého samostatně nedokáže fungovat, a proto v něm není možné (alespoň co se týče klasické distribuce) vytvářet desktopové aplikace komunikující s webovou službou jako v ostatních popsaných jazycích. Proto se v něm můžeme zaměřit pouze na tvorbu servrových aplikací webových služeb a klientů na straně serveru.

Nativní podpora v PHP 4 byla limitována pouze na základní XML technologie. Kromě několika technologií pro práci s XML (SAX, domxml, XSLT) nabízí PHP 4 podporu XML vzdáleného volání procedur (XML-RPC)a pro výměnu obecných XML dat WDDX. Z hlediska tvorby webových služeb však neposkytuje mnoho pomoci, navíc jsou všechna tato řešení součástí vlastního rozšíření, takže neumožňují jednoduchou součinnost.

Nativní podpora v PHP 5 je založena na knihovně libxml2, která implementuje značné množství XML standardů a díky své implementaci v jazyku C je její kód i dostatečně rychlý. Pro podporu webových služeb nabízí třídy pro podporu SOAP webových služeb a webových služeb založených na vzdáleném volání procedur (XML-RPC). Rozšíření SOAP umožňuje poměrně jednoduše vytvářet SOAP servery a klienty a podporuje specifikace SOAP 1.1, SOAP 1.2 a WSDL 1.1. PHP 5 nativně nepodporuje standardy související s webovými službami.

Kromě nativní podpory lze využít rozšíření třetích stran. Jedním z nich je implementace nuSOAP, která byla aktivní zejména před vznikem PHP 5 a rozšiřovala možnosti PHP 4 o podporu SOAP a WSDL. Nevýhodou oproti nativní podpoře u nuSOAP rozšíření je jeho implementace formou PHP tříd, takže nedosahuje takového výkonu jako PHP 5. V dnešní době už se nuSOAP nevyvíjí.

V dnešní době se těší značné oblibě vývojářské rozšíření WSO2 Web Services Framework for PHP (WSO2 WSF/PHP), které vyvíjí společnost WSO2 zabývající se vývojem open source řešení v oblasti SOA architektury. Rozšíření WSF/PHP implementuje nejnovější standardy v oblasti webových služeb a jeho implementace je založena na Axis2/C Web Services enginu. Samotný WSF/PHP framework je implementován v jazyce C, takže vyvíjené aplikace mohou pracovat maximálně efektivně. WSF/PHP např. implementuje standardy WS-*.

6.2 Implementace webové služby

Cílem implementace webové služby je ověřit dostupnost prostředků, které usnadňují tvorbu webových služeb v jazycích Java, PHP a Visual Basic .NET. Tato kapitola tedy popisuje jednotlivé kroky, které je třeba udělat.

Pro praktickou implementaci byla navržena webová služba pro správu databáze osob. Její funkce je poskytovat klientům jméno nebo příjmení osoby, na základě podaného rodného čísla. Služba pracuje s databází osob, jejíž jednoduchá struktura je popsána následujícím příkazem SQL:

Zdrojový kód 6.1: SQL příkaz pro vytvoření struktury databáze

```
CREATE TABLE [dbo].[osoby](
    [id] [int] IDENTITY(1,1) NOT NULL,
    [rc] [int] NOT NULL,
    [jmeno] [varchar](50) NOT NULL,
    [prijmeni] [varchar](50) NOT NULL
)
```

Databázovým zdrojem pro všechny implementace je databáze Microsoft SQL 2005 Express.

Samotná služba je pojmenována SpravaOsob a uspokojuje následující požadavky:

- přidání osoby do databáze (zadává se rodné číslo, jméno a příjmení),
- zjištění jména na základě rodného čísla,
- zjištění příjmení na základě rodného čísla.

K tomuto účelu jsou definovány funkce pridejOsobu, dejJmeno a dejPrijmeni s příslušnými parametry. Zdrojové kódy tříd implementujících tuto webovou službu jsou uvedeny v příloze. [1]

[1] Všechna předvedená řešení byla implementována na platformě .NET

6.2.1 .NET

Nástrojem pro implementaci v prostředí .NET bylo Microsoft Visual Studio 2008 a funkci serveru plnila služba IIS 7 (Internet Information Services). Ke spolupráci vývojového prostředí se serverem bylo třeba pouze nastavit podporu ASP.NET a IIS metabase a IIS 6 konfigurační kompatibilu ve funkcích systému. Připojení k databázi proběhlo pomocí funkcí ADO.NET.
Postup vytvoření služby ve Visual Studiu je následující:

1. vytvoření nového webové stránky (*File* -> *New* -> *Web Site*) s přednastavenou šablonou *ASP.NET Web Service*,

2. v nastavení je třeba vybrat umístění na serveru IIS a ve výchozím adresáři webového serveru založit nový adresář pomocí volby *Create New Web Site*,

3. dále se vygeneruje nový soubor s výchozím kódem s funkcí HalloWorld. Pro vytvoření vlastní webové služby je třeba do projektu přidat novou webovou službu pomocí volby *Add New Item* -> *Web Service*, příslušně ji pojmenovat a nahradit vygenerovaný kód kódem vlastní třídy (s prefixem <*WebMethod()*> u každé metody),

4. spuštěním projektu se automaticky vytvoří kód pro publikování webové služby na serveru a ve webovém prohlížeči se zobrazí je odkazy na vygenerovaný WSDL kód (součást přílohy) a ukázky volání služby protokolem SOAP a přes HTTP Post.

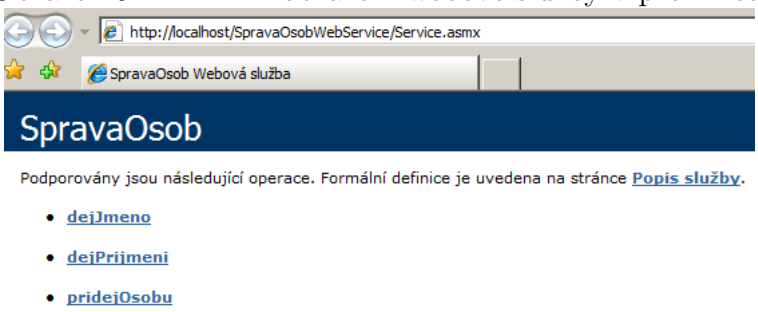
Takto je připravena služba k použití. Základem webové služby v .NET je soubor s příponou .asmx, na který se odkazuje v klientovi služby. Tvorba klienta je také

jednoduchá. Spočívá ve vytvoření ASP.NET stránky a předání reference na webovou službu do projektu:

1. založení nové stránky (*File* -> *New* -> *Web Site* -> *ASP.NET Web Site*),
2. přidání reference na vytvořenou službu (v projektu *Add Web Reference*) zadáním adresy služby na server
   (např. *http://localhost/SpravaOsobWebService/Service.asmx*),
3. přidání ovládacích prvků do .aspx stránky (například pomocí Designeru) a nastavením příslušných akcí (například volání služby na stisknutí tlačítka) .

Poslední krok by měl zaručit propojení klienta s webovou službou.

Obrázek 6.1: .NET: zobrazení webové služby v prohlížeči



Zdrojový kód 6.2: Visual Basic kód vložený do aspx stránky

```
Partial Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Button1_Click(ByVal sender As Object, _
      ByVal e As System.EventArgs) Handles Button1.Click
        Dim so As New localhost.SpravaOsob()
        Label1.Text = so.dejJmeno(888).ToString()
    End Sub
End Class
```

6.2.2 Java

Jako vývojové prostředí pro tvorbu služby v jazyce Java byla zvolena platforma Eclipse s rozšířením Web Tools Platform (WTP) [2] pro vývoj webových a Java EE aplikací. Jako server pro běh služby byl použit Apache Tomcat v kombinaci s Axis2. Pro připojení k Microsoft SQL databázi bylo nutné stáhnout a nainstalovat Miscrosoft SQL JDBS driver.
Postup pro vytvoření webové služby byl následující:

1. nakonfigurovat Tomcat server a Axis2 runtime pro běh v prostředí Eclipse,
2. vytvořit novou webovou aplikaci (*Web* -> *Web Dynamic Project*) a nakonfigurovat spolupráci se serverem Tomcat a Axis2,
3. vytvořit zdrojový kód třídy pro webovou službu (součást přílohy) jako součást balíčku wtp,
4. vytvořit nový projekt (*Web Service* -> *Web Service*) a označit jako Service Implementation třídu webové služby (nakonfigurovat pro server Tomcat a Axis2 runtime),
5. spuštěním projektu se vygeneruje příslušný kód pro běh na serveru a v okně prohlížeče se zobrazí úvodní stránka.

Klienta (jako konzolovou aplikaci) vytvoříme pomocí nového projektu:

1. vytvořit nový projekt (*Web Service* –> *Web Service Client*) s vazbou na vytvořený popis služby

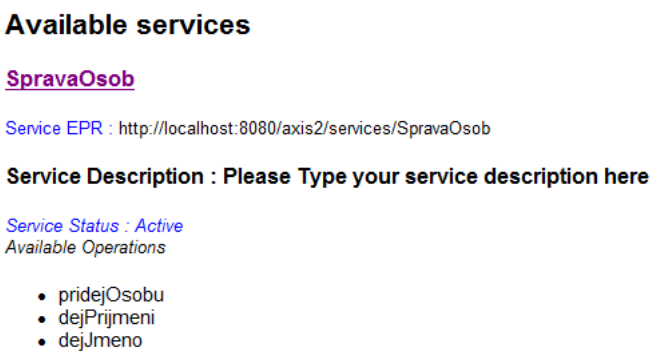
[2] použitá verze Eclipse WTP 3.4.2

(např. *http://localhost:8080/axis2/services/SpravaOsob?wsdl*), čímž se vygenerují tzv. Stubs (tj. třídy implementující interakci s třídami, ze kterých byli vytvořeny),

2. vytvořit kód aplikace, který bude webovou službu spouštět a tuto aplikaci spustit.

Výstup této aplikace se vypisuje do konzole.

Obrázek 6.2: Java: zobrazení webové služby v prohlížeči



6.2.3 PHP

K implementaci řešení v prostředí jazyka PHP bylo použito prostředí Eclipse, ale pouze jako editor zdrojového kódu. Instalace frameworku WSF/PHP zahrnovala stažení binární distribuce a potřebných knihoven, dále pak konfiguraci PHP ve verzi 5.2.10 pro práci s knihovnami WSF. Služba byla spouštěna na serveru Apache ve verzi 2.2. Připojení k Microsoft SQL databázi proběhlo za použití standardních funkcí PHP na bázi ODBC driveru.

Vytvoření služby ve frameworku WSF/PHP zahrnovalo následující kroky:

1. vytvoření třídy implementující funkce služby (je obsahem přílohy),
2. napsání funkce generující odpověď služby a její registrace v objektu webové služby (třída *WSService*),
3. publikování vytvořeného skriptu na serveru (zkopírování .php souboru na server).

Zdrojový kód 6.3: Jednoduchá služba v jazyku PHP

```
<?php
require_once("class_SpravaOsob.php");

$so = new SpravaOsob();

function dejJmeno($message) {
$jmeno = $so->dejJmeno(888);
$responsePayloadString = <<<XML
    <dejJmenoResponse>$jmeno</dejJmenoResponse>
XML;

    $returnMessage = new WSMessage($responsePayloadString);

    return $returnMessage;
}
$service = new WSService(array("operations" => array("dejJmeno")));
$service->reply();
?>
```

Po těchto krocích je možno se službou komunikovat. Pro vytvoření klienta je nutné znát formát zprávy, která se má posílat, aby na ní služba mohla zareagovat. Tvorba klienta probíhá následovně:

1. vytvoření požadavku na službu,

2. vytvoření objektu klienta (třída *WSClient*) se specifikací protokolu,

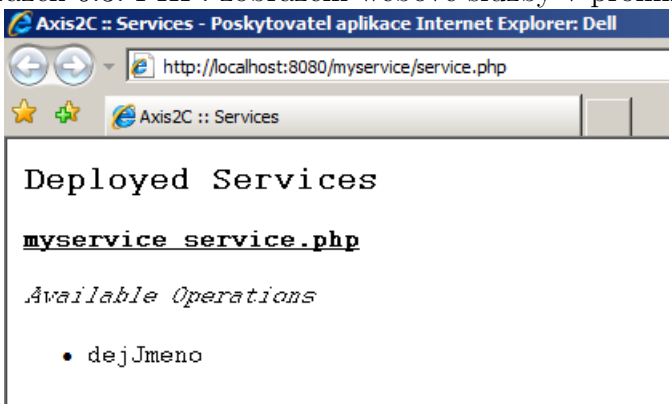
3. předání požadavku a adresy služby klientovy.

Nakonec se přečte opověď služby a příslušně se zpracuje. K ověření klienta je potřeba soubor umístit na server.

Zdrojový kód 6.4: Klient webové služby v jazyku PHP

```
<?php
$reqestPayloadString = <<<XML
<dejJmeno>888</dejJmeno>
XML;

$message = new WSMessage($reqestPayloadString,
       array("to" => "http://localhost/myService/service.php"));

$client = new WSClient(array("useSOAP" => 1.2));

$response = $client->request($message);

echo $response->str;
?>
```

Obrázek 6.3: PHP: zobrazení webové služby v prohlížeči



Závěr

Závěrem lze konstatovat, že jazyky, které byly podrobeny analýze v této bakalářské práci, poskytují zcela dostatečné nástroje pro tvorbu komplexních webových a XML aplikací. I když například PHP neobsahuje nativní podporu standardů webových služeb, existují volně dostupná rozšíření, která poskytují programátorům snadno použitelná rozhraní pro implementaci i těch nejnovějších standardů z oblasti webových služeb.

Co se týče programátorského komfortu a dostupnosti knihoven nebo rozšíření potřebných k vývoji aplikací, nabízí Java a .NET komplexnější vývojová prostředí, která značně usnadňují a urychlují proces získání potřebného programového vybavení. Tento trend souvisí s vývojem těchto IDE (Integrated Development Environment)[1] pod záštitou komunit zabívajících se vývojem celé platformy (nejen pro webové aplikace), což umožňuje vyšší míru integrace funkcí do celého prostředí.

Naproti tomu v PHP, jako proprietárnímu jazyku pro vývoj aplikací na straně serveru, probíhá instalace komponent a rozšíření převážně manuálně. Často také neexistuje dostatečná dokumentace k rozšíření, což může působit komplikace při instalaci.

[1] integrované vývojové prostředí

Obtížnost vývoje samotné jednoduché webové služby v situaci, kdy jsou k dispozici potřebné vývojové komponenty, je ve všech jazycích srovnatelná. Některá vývojová prostředí Javy a .NET jsou při založení projektu schopna automaticky generovat část kódu, ale základní funkčnost webové služby je nutné naprogramovat ručně. Možnosti zakládání projektů a následného publikování webových služeb jsou v IDE Eclipse a Visual Studia na vysoké úrovni a programátor se v podstatě nemusí starat o konfiguraci serveru.

Pro robustnější řešení klient-server se stále častěji používají technologie Javy nebo .NET, a to hlavně díky možnosti rozšíření na desktopové aplikace a integraci modernějších technologií do řešení. I programátorský komfort je v těchto prostředích větší, protože jazyky nabízejí větší množství předdefinovaných tříd a balíků implementující i složitější konstrukce.

Také pro začínajícího programátora v oblasti webových aplikací je zajímavější soustředit se na technologie Java a .NET, protože nabízejí více možností, k jejichž implementaci vede menší pracovní úsilí. V oblasti jazyka PHP nevznikají prakticky žádné nové technologie a jeho vývoj spočívá spíše v implementaci současných standardů.

Literatura

[1] Mlýnkova, I. - Nečaský, M. - Pokorný, J. - Richta, K. - Toman, K. - Toman, V. **Technologie XML: Principy a aplikace v praxi**. 1. vyd. Praha: Grada, 2008. 267 s. ISBN 978-80-247-2725-7.

[2] Richards, Robert. **Pro PHP XML and Web Services**. Berkley: Apress, 2006. 919 s. ISBN 1590596331.

[3] Kuba, Martin. **Web Services**. Peter Vojtáš (ed.), DATAKON 2006, Brno, 14.-17. 10. 2006, pp. 93-112.

[4] Harold, Elliotte Rusty - Means, W. Scott. **XML v kostce**. 1. vyd. Praha: Computer Press, 2002. 439 s. ISBN 80-7226-712-4.

[5] Møller, Anders - Schwartzbach, Michael I. **An Introduction to XML and Web Technologies**. Boston: Addison-Wesley, 2006. 542 s. ISBN 0321269667.

[6] Herout, Pavel. **Java a XML**. 1. vyd. České Budějovice : Kopp, 2007. 313 s. ISBN 978-80-7232-307-4.

[7] Řepišová, Z. **XML - o čem a pro koho?**. Zpravodaj ÚVT MU. ISSN 1212-0901, 2000, roč. X, č. 4, s. 2-5.

[8] Ocelka, J. **Mobilem do Internetu**. Zpravodaj ÚVT MU. ISSN 1212-0901, 2000, roč. X, č. 5, s. 6-12.

[9] Matulík, P. – Pitner, T. **Sémantický web a jeho technologie**. Zpravodaj ÚVT MU. ISSN 1212-0901, 2004, roč. XIV, č. 3, s. 15-17.

[10] Kuba, M. **XML snadno a rychle**. Zpravodaj ÚVT MU. ISSN 1212-0901, 2003, roč. XIII, č. 3, s. 4-9.

[11] Vochozka, J. **Značkovací jazyky a XML (2)**. Zpravodaj ÚVT MU. ISSN 1212-0901, 2001, roč. XI, č. 3, s. 5-9.

[12] Altmann, Petr. **Srovnání implementační náročnosti a výkonu webových služeb v .NET Framework a J2EE**. Diplomová práce. Zlín: UTB MU, 2006. 83 s.

[13] Vrabec, Ondřej. **Validace webových služeb podle profilu WS-I v prostředí NetBeans**. Diplomová práce. Praha: FEL ČVUT, 2008. 63 s.

[14] **W3C Schools** [online]. [cit. 2009-04-15]
<http://www.w3schools.com>

[15] Pichlík, Roman. **Rich Internet Application** [online]. Publikováno 14.6.2005 [cit. 2009-04-15]
<http://interval.cz/clanky/rich-internet-application>

[16] Bureš, Jiří. **RSS? RSS!** [online]. Publikováno 4.3.2003 [cit. 2009-04-15]
<http://interval.cz/clanky/rss-rss>

[17] Kadlec, Tomáš. **Praktické využití architektury SOA** [online]. Publikováno 27.8.2008 [cit. 2009-04-20]
<http://www.itbiz.cz/tomas-kadlec-soa>

Seznam obrázků

Seznam zdrojových kódů

Příloha A

Zdrojový kód A.1: Zdrojový kód implementace třídy v jazyku Visual Basic

```
Class SpravaOsob
    Private connectionString As String = _
      System.Configuration.ConfigurationManager.AppSettings("connectionString")
    Private tableName As String = _
      System.Configuration.ConfigurationManager.AppSettings("tableName")

    Public Function pridejOsobu(ByVal rc As Integer, ByVal jmeno As String, _
       ByVal prijmeni As String) As Boolean
        Dim sqlConnection As New SqlConnection(connectionString)
        Dim sqlCommand As New SqlCommand("INSERT INTO " & tableName & _
          " ([rc],[jmeno],[prijmeni]) VALUES (" & CStr(rc) & ",'" & jmeno & _
             "','" & prijmeni & "')", sqlConnection)

        sqlConnection.Open()
        sqlCommand.ExecuteNonQuery()
        sqlConnection.Close()
        Return True
    End Function

    Public Function dejJmeno(ByVal rc As Integer) As String
        Dim sqlConnection As New SqlConnection(connectionString)
        Dim sqlCommand As New SqlCommand("SELECT [jmeno] FROM " & tableName & _
          " WHERE [rc] = " & CStr(rc), sqlConnection)

        sqlConnection.Open()
        Dim dataReader As SqlDataReader = sqlCommand.ExecuteReader()
```

```
        Dim jmeno As String = ""
        If dataReader.Read Then
            jmeno = dataReader("jmeno")
        End If
        sqlConnection.Close()
        Return jmeno
    End Function

    Public Function dejPrijmeni(ByVal rc As Integer) As String
        Dim sqlConnection As New SqlConnection(connectionString)
        Dim sqlCommand As New SqlCommand("SELECT [prijmeni] FROM " & tableName _
         & " WHERE [rc] = " & CStr(rc), sqlConnection)

        sqlConnection.Open()
        Dim dataReader As SqlDataReader = sqlCommand.ExecuteReader()
        Dim prijmeni As String = ""
        If dataReader.Read Then
            prijmeni = dataReader("prijmeni")
        End If
        sqlConnection.Close()
        Return prijmeni
    End Function
End Class
```

Zdrojový kód A.2: Zdrojový kód implementace třídy v jazyku Java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class MSSQLConnector {
    private static final String connectionUrl = Constants.connectionString;
    private Connection con;
    private ResultSet result = null;

    MSSQLConnector() {
        try {
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        this.con = null;
        try {
            this.con = DriverManager.getConnection(connectionUrl);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public boolean executeUpdate(String sql) {
        try {
            Statement stmt = this.con.createStatement();
            stmt.executeUpdate(sql);
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
        return true;
    }

    public void executeQuery(String sql) {
```

```
        try {
            Statement stmt = this.con.createStatement();
            result = stmt.executeQuery(sql);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public String fetchField(String fieldName) {
        String ret = "";
        try {
            if (this.result != null && this.result.next()) {
                ret = this.result.getString(fieldName);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return ret;
    }
}

public class SpravaOsob {
    private MSSQLConnector con = new MSSQLConnector();

    public void pridejOsobu(int rc, String jmeno, String prijmeni) {
        String sql = "INSERT INTO " + Constants.tableName +
    " ([rc],[jmeno],[prijmeni]) VALUES (" + rc + ",'" + jmeno + "','" + prijmeni + "')";
        this.con.executeUpdate(sql);
    }

    public String dejJmeno(int rc) {
        String sql = "SELECT [jmeno] FROM " + Constants.tableName + " WHERE [rc] = " + rc;
        this.con.executeQuery(sql);
        return this.con.fetchField("jmeno");
    }

    public String dejPrijmeni(int rc) {
        String sql = "SELECT [prijmeni] FROM " + Constants.tableName + " WHERE [rc] = " + rc;
```

```
        this.con.executeQuery(sql);
        return this.con.fetchField("prijmeni");
    }
}
```

Zdrojový kód A.3: Zdrojový kód implementace třídy v jazyku PHP5

```
<?php
require_once ('constants.php');
require_once ('class_ODBCConnector.php');

class ODBCConnector {
    private $resource;
    private $result;
    private $statement;

    public function __construct($dsn, $user, $password) {
        try {
            $this->resource = odbc_connect($dsn, $user, $password);
            if ($this->resource === false) {
                $this->resource = null;
            }
        } catch (Exception $e) {
            var_dump($e->getMessage());
        }
    }

    public function __destruct() {
        odbc_close($this->resource);
    }

    public function execute($statement, $params = null, $flags = null) {
        $this->statement = $statement;
        if (($params != null) && !empty ($params)) {
            return @ odbc_execute($this->statement, $params, $flags);
        } else {
            return @ odbc_execute($this->statement, null, $flags);
        }
    }

    public function query($sql, $flags = null) {
        $this->free();
        $this->result = @ odbc_exec($this->resource, $sql, $flags);
        if ($this->result === false) {
```

```
            $this->result = null;
            return false;
        } else {
            return true;
        }
    }

    public function free() {
        if ($this->result != null) {
            odbc_free_result($this->result);
            $this->result = null;
        }
    }

    public function fetch($rowNumber = null) {
        if ($this->result != null) {
            $fetched = @ odbc_fetch_array($this->result, $rowNumber);
            if ($fetched) {
                return $fetched;
            } else {
                return array ();
            }
        } else {
            return array ();
        }
    }
}
class SpravaOsob {
    private $db;

    public function __construct() {
        $this->db = new ODBCConnector(DB_DSN, DB_LOGIN, DB_PASSWORD);
    }

    public function pridejOsobu($rc, $jmeno, $prijmeni) {
        $query = "INSERT INTO " . TABLE_NAME . " ([rc],[jmeno],[prijmeni]) VALUES ('
    $rc','$jmeno','$prijmeni')";
        if ($this->db->query($query)) {
```

```
            return true;
        }
        return false;
    }

    public function dejJmeno($rc) {
        $query = "SELECT [jmeno] FROM " . TABLE_NAME . " WHERE [rc] = $rc";
        if ($this->db->query($query)) {
            $record = $this->db->fetch();
            return $record['jmeno'];
        }
        return false;
    }

    public function dejPrijmeni($rc) {
        $query = "SELECT [prijmeni] FROM " . TABLE_NAME . " WHERE [rc] = $rc";
        if ($this->db->query($query)) {
            $record = $this->db->fetch();
            return $record['prijmeni'];
        }
        return false;
    }
};
?>
```

Zdrojový kód A.4: Část vygenerovaného WSDL popisu webové služby

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
   xmlns:tns="http://tempuri.org/" xmlns:s="http://www.w3.org/2001/XMLSchema"
   xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
   xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
   targetNamespace="http://tempuri.org/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
      <s:element name="pridejOsobu">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="rc" type="s:int" />
            <s:element minOccurs="0" maxOccurs="1" name="jmeno" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="prijmeni" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="pridejOsobuResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="pridejOsobuResult" type="s:boolean" />
          </s:sequence>
        </s:complexType>
      </s:element>

      ...

    </s:schema>
  </wsdl:types>
  <wsdl:message name="pridejOsobuSoapIn">
    <wsdl:part name="parameters" element="tns:pridejOsobu" />
  </wsdl:message>
  <wsdl:message name="pridejOsobuSoapOut">
```

```
    <wsdl:part name="parameters" element="tns:pridejOsobuResponse" />
  </wsdl:message>

      ...

  <wsdl:portType name="SpravaOsobSoap">
    <wsdl:operation name="pridejOsobu">
      <wsdl:input message="tns:pridejOsobuSoapIn" />
      <wsdl:output message="tns:pridejOsobuSoapOut" />
    </wsdl:operation>

      ...

  </wsdl:portType>
  <wsdl:binding name="SpravaOsobSoap" type="tns:SpravaOsobSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="pridejOsobu">
      <soap:operation soapAction="http://tempuri.org/pridejOsobu" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>

      ...

  </wsdl:binding>
  <wsdl:service name="SpravaOsob">
    <wsdl:port name="SpravaOsobSoap" binding="tns:SpravaOsobSoap">
      <soap:address location="http://localhost/SpravaOsobWebService/Service.asmx" />
    </wsdl:port>
    <wsdl:port name="SpravaOsobSoap12" binding="tns:SpravaOsobSoap12">
      <soap12:address location="http://localhost/SpravaOsobWebService/Service.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```