Srovnání implementační náročnosti a výkonu webových služeb v .NET Framework a J2EE

Bc. Petr Altmann

Diplomová práce
2006

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2005/2006

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Petr ALTMANN**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Srovnání implementační náročnosti a výkonu webových služeb na platformách .NET Framework a J2EE.**

Zásady pro vypracování:

1. Vytvořte rešerši srovnávající implementační náročnost a výkon webových služeb vytvářených pomocí různých programovacích technologií a aplikačních serverů. Zaměřte se zejména na tyto technologie a produkty:
a. .NET Framework a libovolný vhodný programovací (skriptovací) jazyk váprostředí MS Visual Studio 2003 nebo 2005 (nejlépe C)
b. J2EE a Java (Sun Application Server, Tomcat + Axis) váprostředí NetBeans a Ecplise
c. Apache a PHP válibovolném vhodném vývojovém prostředí
2. Dokument by měl obsahovat:
a. Formální popis jak serverové, tak klientské části webové služby (daná webová služba by měla pokud možno demonstrovat všechny možnosti této technologie) sávyužitím UML
b. Podrobný popis tvorby serverové/klientské části webové služby ve všech zmiňovaných vývojových prostředích sávyužitím UML
c. Srovnávací tabulku hodnotící výkon webové služby vázávislosti na použitých technologiích

Rozsah práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. Sharp, J., Jagger, J.: MS Visual C .NET krok za krokem, Mobil Media, 2002
2. Duthie, G. A.: MS ASP.NET krok za krokem, Mobil Media, 2003
3. Richter, J.: .NET Framework, Grada, 2003
4. Esposito, D.: XML – efektivní programování pro .NET, Grada, 2004
5. Bollinger, G., Natarajan, B.: JSP – Java Server Pages, Grada, 2003
6. Eckel, B.: Myslíme vájazyce Java, Grada, 2001
7. Welling, L., Thomson, L.: PHP a MySQL – rozvoj webových aplikací, SoftPress, 2002
8. Arlow, J.: UML a unifikovaný proces vývoje aplikací, Computer Press, 2003
9. Kanisová, H., Miller, M.: UML srozumitelně, Computer Press, 2004
10. Internetové zdroje:

a. IDE NetBeans – http://www.netbeans.org/
b. JAVA – http://java.sun.com/
c. IDE Elipse – http://www.eclipse.org/
d. Vývojové nástroje firmy MS – http://msdn.microsoft.com/vstudio/express/

Vedoucí diplomové práce: **Ing. Michal Bližňák**
Ústav aplikované informatiky

Datum zadání diplomové práce: **14. února 2006**

Termín odevzdání diplomové práce: **26. května 2006**

Ve Zlíně dne 14. února 2006

prof. Ing. Vladimír Vašek, CSc.
pověřený děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.
1

doc. Ing. Ivan Zelinka, Ph.D.
ředitel ústavu

ABSTRAKT

Tato diplomová práce zkoumá možnosti vývoje webových služeb a snaží se tyto možnosti mezi sebou srovnat. Srovnání je dvouúrovňové. V prvním případě srovnává náročnost a možnosti implementace webových služeb na různých programátorských platformách, jako jsou Microsoft .NET, J2EE a PHP, za použití příslušných vývojových nástrojů. Druhým pohledem na srovnání je pak výkon těchto implementací.

Klíčová slova:

Webové služby, SOAP, WSDL, XML, parser, UML, databáze, ADO.NET, JDBC, .NET Framework, Java 2 Enterprise Edition, nuSOAP, PHP, C#, Java, Visual Studio .NET, NetBeans IDE, Eclipse.

ABSTRACT

This diploma thesis deals with possibilities of development of web services and gives a comparison of these possibilities. This is a two-level comparison. In the first step, I compare difficulty and possibilities of implementation of web services based on different platforms such as Microsoft .NET, J2EE a PHP while using relevant development environments. The second step is comparison of performance of these implementations.

Keywords:

Web services, SOAP, WSDL, XML, parser, UML, database, ADO.NET, JDBC, .NET Framework, Java 2 Enterprise Edition, nuSOAP, PHP, C#, Java, Visual Studio .NET, NetBeans IDE, Eclipse.

Na tomto místě bych rád poděkoval vedoucímu diplomové práce Ing. Michalu Bližňákovi, za podnětné připomínky a rady, při sestavování tohoto textu.

Dále bych rád poděkoval všem blízkým, za trpělivost a podporu, kterou mi vyjádřili, když jsem pracoval na této diplomové práci.

OBSAH

ÚVOD

Webové služby jsou poměrně novým fenoménem, se kterým se v oblasti IT můžeme setkat. Jedná se o softwarové komponenty, které jsou schopny plnit nejrůznější funkce přes Internet. Navíc protože jsou jejich popisy otevřené, není problém pro vývojáře zjistit, co služba dělá a co ke své činnosti potřebuje. Pokud toto ví, může jeho program, ať už je napsán v kterémkoli z programovacích jazyků a běží na kterémkoli operačním systémů, využívat webovou službu.

Tato diplomová práce se zabývá srovnáním webových služeb na různých platformách. Toto srovnání probíhá ze dvou základních hledisek. Jedná se především o možnosti a náročnost implementace webové služby jak po serverové tak po klientské stránce. Dále se tato práce pokouší zodpovědět otázku výkonu jednotlivých variant implementace webových služeb porovnáním jejich rychlosti.

První kapitola přináší základní teoretický přehled o formátu XML, základním kameni, na kterém celé technologie webových služeb stojí. Zdůrazněny jsou zde především prvky XML, kterým se ve webových službách dostává nejvíce pozornosti – systém jemnných prostorů a schémat.

Druhá kapitola zodpovídá otázku, co to vlastně webové služby jsou, jaké jsou možnosti vývoje těchto služeb a na jakých principech webové služby pracují. Podrobněji se zde věnuji komunikaci mezi klientskou a serverovou částí webové služby na standardu XML – SOAP messaging. Dále způsobu, jak jsou webové služby univerzálně popsány ve formátu WSDL a také jaké existují možnosti vyhledávání existujících webových služeb.

Třetí kapitola se zaměřuje na implementaci webových služeb na platformě firmy Microsoft .NET Framework. Jedná se především o možnosti práce s databází a vývojové nástroje, pomocí kterých je možné webovou službu vytvořit.

Čtvrtá kapitola popisuje implementaci na platformě Java. Ve své struktuře je velmi podobná kapitole předcházející.

Pátá kapitola sleduje možnosti práce s webovými službami ve skriptovacím jazyku PHP.

Šestá kapitola popisuje standardní způsoby návrhu webových služeb technikami jazyka UML s možnostmi, jaké poskytu při návrhu velkých projektů.

Praktická část této diplomové práce se zabývá z velké části popisem implementace konkrétní webové služby navržené jako příklad pro snadné pochopení.

Sedmá kapitola názorně ukazuje proces návrhu webové služby DVDMovie pomocí jazyka UML. Jedná se o službu, která má simulovat funkcionalitu jednoduché půjčovny DVD disků, na kterých jsou obsaženy filmy.

Osmá, devátá a desátá kapitola se pak zaměřuje na konkrétní popis vývoje webové služby DVDMovie nejprve v .NET, poté v Javě a nakonec v PHP

Jedenáctá kapitola přináší srovnání jednotlivých implementací jak po stránce náročnosti postupů a možných vzniklých problémů, tak také po stránce výkonu, kdy byla sestavena metodika měření uplatněná na každou implementaci webové služby DVDMovie. Výsledky jsou prezentovány ve formě tabulek a srovnávacího grafu.

I. TEORETICKÁ ČÁST

1 XML JAKO ZÁKLAD WEBOVÝCH SLUŽEB

XML (eXtensible Markup Language) je termín, na který můžeme narazit v posledním desetiletí opravdu často, navíc v různých souvislostech. O co se tedy jedná? XML je nástroj, často se také používá termín jazyk, který umožňuje standardizovat způsoby výměny informací. Kouzlo XML je především v jeho **otevřenosti** a **univerzálnosti**.

Otevřeností mám na mysli především to, že specifikace formátu XML je veřejně dostupná[1]. To znamená, že každý kdo má zájem použít jazyk XML, může se podívat, jaké pravidla pro tvorbu dokumentů v tomto jazyce musí dodržet. Toto je častý a prakticky také největší problém všech neotevřených formátů. Jejich specifikace bývá často velice těžko dostupná, případně není dostupná vůbec. To ztěžuje jejich použití v aplikacích, které jsou vytvářeny třetími stranami. Typickým příkladem je formát DOC americké firmy Microsoft, který není otevřený. Pokusy o jeho implementaci sice vznikají dnes a denně (spousta konkurenčních aplikací se snaží zpřístupnit funkce čtení tohoto formátu), avšak opravdu jediná aplikace, která tento formát zpracuje bez problémů je textový editor Word, pro nějž byl formát DOC vytvořen. V tomto případě se jedná zcela jistě o záměr firmy Microsoft. Avšak otevřený formát by přinesl daleko lepší možnosti využití i v konkurenčním software a zákazník by na tom byl podstatně lépe.

Univerzálnost jazyka XML vidím především v jeho možnostech fungování na různých platformách a v různých národních prostředích. Této vlastnosti bylo docíleno hlavně textovou podobou XML jazyka. Nejedná se tedy o binární strukturu, často závislou právě na platformě. XML soubor je tedy možné otevřít v libovolném textovém editoru a číst jeho obsah. Možnost fungovat v různých národních prostředích je dána vlastností samotné struktury XML, která v sobě nese též informaci o použitém kódování v dokumentu. Podle této informace pak musí klientská aplikace reagovat, často však bývá tato reakce ošetřena již na úrovni programátorských knihoven jednotlivých jazyků.

Abychom nezmiňovali jenom výhody, tak je třeba upozornit také na stinné stránky XML jazyka. Hlavním nedostatkem, který prakticky zpočátku velmi bránil jeho rozšíření je velká redundance dat. Už samotná textová forma uložení je proti binární velmi neefektivní. Do-

slova kamenem úrazu ale jsou jednotlivé řídící informace (tagy), které se v jednom dokumentu často i několikrát opakují. Toto je daň za vynikající vlastnosti, uvedené v předchozích odstavcích. Ovšem jedním dechem je třeba podotknout, že tento problém je v dnešní době čím dál tím méně palčivý, jelikož dochází opravdu k velkým nárůstům paměťových a přenosových kapacit u hardwarových zařízení a je možné, že tento problém se za pár let stane opravdu nepodstatným. Dnes lze tento problém také velmi dobře odstínit za pomoci moderních datových komprimačních technik.

O XML formátu se zmiňují také některé webové stránky, kde je možné získat velmi dobré informace [7], [8].

1.1 Důležité prvky XML používané ve webových službách

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.

1.1.1 Jmenné prostory

Jmenné prostory v XML jsou mechanismem, který umožňuje rozšiřitelnost XML z hlediska formátů. V případě že máme data o různém významu v rámci jednoho nebo více XML souborů označeny stejnými elementy, dojde ke konfliktu. Jmenné prostory jsou právě tím mechanismem, který konfliktu zabrání. Příklad je uveden na následující struktuře:

```
<clients xmlns="http://data.clients.cz/names">
    <client>Novák Jan</client>
    <city>Zlín</city>
</clients>
```

Obrázek 1: Ukázka jmenného prostoru v XML struktuře

Element `<clients>` má nadefinován jmenný prostor http://data.clients.cz/names a tím je odlišen od případných ostatních elementů `<clients>`, které tento jmenný prostor nadefi-

[1] Specifikaci zveřejňuje konsorcium W3C, které zastřešuje spoustu dalších technologií a standardů. Specifikaci XML formátu můžete najít na webu: http://www.w3c.org/xml/

novaný nemají. Tvar definice jmenného prostoru se skládá z atributu `xmlns` a přiřazenému URI, což je jednoznačný textový identifikátor. Nejčastěji má podobu URL adresy, protože ty jsou v rámci internetu unikátní. Ovšem tato adresa význam jako URL nemá a klidně vůbec nemusí směřovat na existující stránku. Častou koncepcí ovšem je, že právě na této adrese je k nalezení popis elementu, ke kterému je toho URL jako jmenný prostor přiřazeno.

Další důležitou vlastností nadefinovaného jmenného prostoru je obor jeho platnosti. Ten se totiž vztahuje na všechny potomky aktuálního elementu. V našem příkladě tedy i pro elementy `<client>` a `<city>`. Výjimku z tohoto výčtu mohou tvořit pouze elementy, které mají nadefinovaný svůj vlastní jmenný prostor. Ten potom platí opět pro ně a všechny jejich potomky.

V případě, že je žádoucí u potomka elementu s nadefinovaným jmenným prostorem, aby do tohoto prostoru nepatřil, můžeme tento jmenný prostor zrušit nadefinováním prázdného jmenného prostoru: `xmlns=""`.

Pro zvýšení přehlednosti ve jmenných prostorech byl zaveden souběžně ještě jeden mechanismus a to předpony jmenných prostorů. V případě, že totiž potřebujeme kombinovat více jmenných prostorů museli bychom bez použití předpon deklarovat jmenné prostory pokaždé a na několika místech. To není příliš efektivní např. z důvodu překlepu. Když rozšíříme předcházející příklad o nadefinování jmenného prostoru s příponou bude vypadat následovně:

```
<cln:clients xmlns:cln="http://data.clients.cz/names">
    <cln:client>Novák Jan</cln:client>
    <city>Zlín</city>
</cln:clients>
```

Obrázek 2: Příklad XML struktury s předponami jmenných prostorů

Předponu je nutné vložit i do elementu, ve kterém dochází k samotné její deklaraci. Je to z toho důvodu, že jmenný prostor s příponou se vztahuje výlučně na elementy s uvedenou předponou. V našem příkladě se tedy jmenný prostor vztahuje k elementům `<clients>` a `<client>`, ne však `<city>`.

Poslední zmínkou ke jmenným prostorům je jejich vztah k atributům jednotlivých elementů. Atributy nepatří do jmenného prostoru, pokud jim není tento jmenný prostor explicitně přiřazen pomocí předpony.

Podrobně se jmennými prostory zabývá článek [9].

1.1.2 Šablony

Technologie XML šablon (templates) je postavena stylech XSL. Tento jazyk má dnes po záštitou konsorcium W3C. Pomocí definic provedených v jazyce XSL můžeme provádět tzv. transofmace XML dokumentů (XSLT). Soubor obsahující XSLT styl je obyčejným XML dokumentem a s jako takovým se s ním také pracuje.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
            version="1.0">

      vlastní definice stylu

</xsl:stylesheet>
```

Obrázek 3: Základní struktura definice XSLT stylu

Obrázek 3 ukazuje základní strukturu definice XSLT stylu. Přitom jmenný prostor s uvedenou adresou je velmi důležitý a bez přesného uvedení je definice stylu neplatná.

Samotný styl potom obsahuje jednotlivé šablony, které definují, jakým způsobem se budou transformovat příslušné části XML dokumentu. Vlastní šablony XML pak mají svůj hlavní význam pro kontrolu správnosti XML dokumentu, kdy přesně definují obsah jednotlivých elementů.

1.2 Jednotlivá rozhraní parserů

XML formáty dat, jak už bylo uvedeno výše, jsou univerzální. Jejich programové zpracování a celkově práce s XML je řízena programy, které jsou nazývány parsery. Tyto parsery se dělí podle své vlastní funkčnosti do skupin, které uvedu později.

Protože je XML postaven na textovém souboru, vyplynulo z této skutečnosti několik komplikací při zpracování dat. Pro programátora by nebylo zrovna jednoduché vždy znovu vyvinout knihovny pro práci s XML soubory. Právě proto došlo ke vzniku parserů, které mají podobu jednotlivých knihoven pro programovací jazyky a výrazně usnadňují práci s daty získanými z XML. V dnešní době se používá základní dvě rozhraní parserů. Jsou jimi SAX a DOM.

1.2.1 SAX

Rozhraní SAX (Simple API for XML) je založeno na řízení pomocí událostí. Toto rozhraní vytvoří vazby mezi událostmi, které generuje parser, a programovým kódem. Tyto události (events) se generují jako reakce na pozici při čtení XML dokumentu. Když parser narazí na začátek elementu (úvodní tag), konec elementu, komentář nebo atribut, vždy vyvolá příslušnou proceduru a předá jí aktuální parametry – název elementu, atributu atd.

Mezi základní výhody tohoto událostmi řízeného přístupu patří: rychlost a malé paměťové nároky. Výhody vyplývají ze samotného přístupu. Události jsou vyvolávány postupně v pořadí, jak následují za sebou v dokumentu. To má za následek to, že se v paměti vždy drží odkaz jen na jednu událost a po dokončení jejího zpracování dojde k uvolnění alokované paměti a posunu zpracování směrem k další události. Sekvenční zpracování je také dostatečně rychlé, což je druhou důležitou vlastností rozhraní SAX.

1.2.2 DOM

Rozhraní DOM (Dokument Object Model) má v principu opačnou funkčnost než rozhraní SAX. Nejprve dojde k načtení celého dokumentu nebo jeho části do paměti do podoby stromové struktury. Každému elementu v této podobě odpovídá jeden uzel stromu. Výhody tohoto přístupu spočívají především ve větších možnostech při práci s XML dokumentem. Vždy můžeme modifikovat jednotlivé uzly pomocí příslušných metod. Tyto uzly můžeme také mazat přidávat a pohybovat se celým stromem, aniž bychom byli nuceni procházet celou strukturu od začátku. Nevýhody tohoto rozhraní jsou přesně tam, kde má SAX výhody.

Tabulka 1: Srovnání předností a záporů jednotlivých rozhraní

	SAX	DOM
Výhody (+)	Rychlost Nízké paměťové nároky	Objektová reprezentace dokumentu Možnost přímého zpracování
Nevýhody (-)	Omezené možnosti při manipulaci s elementy Nutnost zpracovávat dokument sekvenčně	Nižší rychlost Vyšší paměťové nároky

1.3 Práce s XML v jednotlivých platformách

XML jako v dnešní době velmi rozšířený datový formát si již nemůže dovolit nepodporovat žádný z „velkých“ jazyků. Proto se s implementací XML do programovacích jazyků setkáváme prakticky všude. Novější jazyky jako Java a C# mají navíc tu výhodu, že již bylo s XML počítáno při jejich vzniku, popř. vzniku částí těchto jazyků. Do starších programovacích jazyků byla podpora XML „doprogramována“ dodatečně.

Podpora práce s XML bývá označována jako API toho kterého jazyka. Zahrnuje v sobě především zpracování nejčastějších činností, které můžeme s XML dokumentem provádět- především tedy čtení, kontrolu správnosti – validity, zápis a modifikaci, přeměnu XML struktur na příslušné objektové struktury, přeměnu XML struktur na třídy konkrétního jazyka.

1.3.1 Java

Práce s XML v Javě se může zdát na první pohled velmi složitá díky množství různých větví, které mají na starosti nejrůznější funkce. Ovšem opak je pravdou. Celá struktura knihoven pro práci s XML je soustředěna do balíčků, které jsou rozděleny podle jednotlivých funkcí a cílových činností. Cílem je především uspořádání knihoven v takové formě, aby při použití různých parserů nedocházelo pokud možno k žádným změnám aplikačního kódu a přístup k parserům byl tak standardizován.

Rozhraní Javy pro práci s XML bylo pojmenováno JAX (Java APIs for XML) a skládá se z několika částí:

- JAXP – základní balíček, který definuje rozhraní pro přístup ke všem parserům, tedy např. SAX, DOM atd. Definuje jejich napojení na konkrétní implementace. Hlavní výhodou je, jak již bylo uvedeno, že programátor může zvolit jiný parser než je dosud používán, aniž by provedl zásah do aplikačního kódu.
- JDOM – nástavba DOM parseru speciálně vytvořeného pro Javu.
- JAXB – třídy sloužící pro mapování XML struktur na jednotlivé Java třídy. Tyto knihovny velmi usnadňují práci při vývoji webových služeb, protože je jich využí-

váno především v oblasti převodu zdrojového kódu Javy na WSDL[2] soubor a naopak.

- JAXM – soubor tříd sloužící k odesílání a přijímaní zpráv ve formátu XML. JAXM funguje na principu asynchronní komunikace – aplikace po odeslání zprávy pokačuje dále ve své činnosti
- JAX-RPC – velmi důležité třídy pro webové služby. Toto rozhraní dovoluje volat vzdálené metody pomocí standardizovaných XML zpráv[3].
- JAXR – třídy umožňující přístup do databází určených k vyhledávání jednolivých webových služeb.

Většina výše uvedených rozhraní je od sebe oddělena záměrně, protože se vždy starají o konkrétní druhy XML struktur – např. WSDL nebo SOAP.

Příklad použití výše uvedených rozhraní je uveden v příloze PI. Jedná se o zdrojový kód v Javě, který názorně ukazuje vytvoření XML dokumentu.

1.3.2 .NET Framework

Také .NET Framework podobně jako Java obsahuje v rámci své „superknihovny“ FCL (.NET Framework Class Library) poměrně velké množství nástrojů pro práci s XML strukturami. Tyto knihovny umožňují pohodlné vytváření XML souborů, jejich čtení, transformace nebo například ukládání do databází. Jedná se především o následující velmi důležité součásti:

- XmlTextReader – velmi rychlé rozhraní, určené pro čtení dokumentů. Obdoba SAX rozhraní.
- XmlTextWriter – toto rozhraní umožňuje vytváření XML souborů.
- XmlDocument – programové rozhraní odpovídající svojí funkcionalitou DOM parserům.

[2] O WSDL popisu webové služby se zmiňuje následující kapitola
[3] Tyto zprávy se nazývají SOAP a podrobně o nich pojednává následující kapitola

- XmlValidatingReader – tato třída je nadstavbou třídy XmlTextReader. Umožňuje provádět kontrolu správnosti XML dokumentu. Zda je XML struktura validní, tedy platná.

Příklad vytvoření XML dokumentu pomocí rozhraní jazyka C# je uveden v příloze PI.

1.3.3 PHP

Stejně jako u předešlých dvou programovacích jazyků, také PHP obsahuje možnosti pro práci s XML dokumenty, ovšem ty nejsou zdaleka tak bohaté. Standardně je součástí PHP EXPAT parser, což je sada open-source knihoven.

Tento parser je velmi jednoduchý a jeho implementace spočívá v procházení XML struktury po jednotlivých znacích a zpracovávání při definici počátečního tagu a koncového tagu. Ještě v rámci počáteční inicializace parseru je nutné nadefinovat funkce, na které se parser odvolá v případě že narazí na počáteční tag nebo jeho atribut, koncový tag a data uvnitř tagu. Bohužel to programátora neoprostí o velké spousty práce na vlastní stavbě algoritmu.

Do PHP je možné nainstalovat a použít i jiné parsery (např. Serializer). Těchto parserů vzniklo za dobu, co existuje PHP velké množství a za většinou z těchto implementací stojí nejrůznější skupiny či jednotlivci velké open-source komunity, která se kolem PHP vytvořila. Není tak problémem najít parsery fungující na principu objektové struktury DOM, stejně tak jako parsery typu SAX.

2 WEBOVÉ SLUŽBY

Webové služby (Web services) se začaly dostávat do povědomí IT veřejnosti kolem roku 2000. Jde tedy vidět, že se jedná o velmi mladou technologii, která se dnes začíná stále více prosazovat.

Celá technologie webových služeb je postavena na platformě nezávislých standardech. Jedná se především o XML struktury a HTTP protokol. Z toho pramení také hlavní deviza webových služeb a to možnost komunikace mezi aplikacemi napříč různými prostředími a platformami. V zásadě se jedná o to, že webová služba může běžet na libovolném k tomu určeném serveru, běžícím na libovolném operačním systému a také hardware. Webové služby pod sebe zahrnují jednotící standardy vývoje aplikací jak pro malé klientské aplikace běžící ve webových prohlížečích tak třeba klientské desktopové aplikace nebo třeba java aplikace v mobilním zařízení.

Než přistoupíme k vlastnímu popisu webových služeb, je třeba objasnit rozdíl mezi termíny webová služba a webová aplikace. V případě webové služby vždy hovoříme mezi komunikačním vztahem automat – automat. Tento komunikační vztah je určen svým rozhraním – většinou standardizovaným protokolem. Naproti tomu webová aplikace zprostředkovává spojení mezi automatem na jedné straně a člověkem na straně druhé. Zde jako příklad webové aplikace může posloužit internetový obchod, který zabezpečuje komunikaci se zákazníkem, prostřednictvím formulářů a funkčních obrazovek, které zákazník vyplňuje popř. jej informují o různých stavech aplikace. Webovou službou v tomto jednoduchém příkladu by mohlo být rozhraní, odesílající požadavky zákazníka na vzdálený server [10].

Podstatné na webových službách je to, že umožňují vyváření webových a dektopových aplikací, které jsou na tyto služby přímo navázané a používají je. Tyto webové a desktopové aplikace pak mohou využívat více těchto služeb najednou.

Technologie webových služeb stojí na třech základních pilířích. V prvé řadě jde o protokol vzdáleného volání procedur SOAP, dále jazyk pro popis webových služeb jako takových – WSDL a mechanismus nalezení služeb UDDI[4].

[4] Ten ovšem není jediný. Existuje více druhů standardů. Mezi další nejznámější patří WSIL

Vlastní popis webových služeb a jejich standardů nejlépe popisuje web konsorcia W3C, které za touto specifikací stojí [12].

2.1 Dva způsoby vývoje webových služeb

Obrovskou výhodou pro programátory webových služeb je existence nepřeberného množství nástrojů na jejich tvorbu, které pracují nejrůznějšími způsoby. Tyto nástroje také přebírají částečně tvorbu programového kódu, který se opakuje a je používán ve standardní formě. Zde dojde tedy k velkému zrychlení práce, protože programátor se nestará o vlastní systémovou vrstvu komunikace a zajištění jejího bezproblémového chodu, ale pouze o „své“ metody.

Webové služby můžeme vytvářet dvěma základními přístupy.

- Přístup shora dolů (top-down)
- Přístup zdola nahoru (bottom-up)

V přístupu shora dolů nejprve vytvoříme popis služby v jazyce WSDL a z něj následně necháme automatickými nástroji vytvořit rozhraní služby ve vlastním programovacím jazyce, kam posléze doplníme vlastní implementaci jednotlivých metod.

Druhý přístup předpokládá, že již rozhraní pro webovou službu naprogramováno máme a z něj potom dojde ke generování odpovídajícího popisu webové služby (WSDL).

Oba výše uvedené přístupy jsou sice přesnými opaky, ale při tvorbě webové služby je můžeme využít zároveň. Abychom tedy nemuseli psát popis webové služby ručně nebo s pomocí nějakého specializovaného nástroje, použijeme některý z programovacích jazyků pro napsání rozhraní služby. Z tohoto rozhraní následně vygenerujeme popis služby ve WSDL jazyku a z něj potom použijeme přístup generování kódu webové služby. Tím se zdánlivě vrátíme na začátek, ale budeme mít již vytvořen kód oné systémové komunikační vrstvy, kterou jsem již zmínil.

2.2 Protokol SOAP jako základ webových služeb

Před příchodem webových služeb, se data přes HTTP přenášela na server jen velice omezeně. Prvním způsobem, který se objevil, byla možnost přidat parametr přímo do URL za znak ?. Toho se využívalo u jednoduchých skriptů běžících na serveru a prakticky je využí-

vá v předávání dodnes nehledě na problémy s omezením délky URL a množstvím parametrů, které je možné touto cestou předat. Když GET metoda přestala stačit, objevila se na světě metoda POST, která umožnila předávat parametry přímo v těle HTTP struktury. To do jisté míry na několik let stačilo do všech možných implementací, protože předávání parametrů metodou POST netrpělo hlavním omezením metody GET a ještě bylo možno toto předávání také lépe zabezpečit. Tato metoda se postupem času stala velmi oblíbenou a dodnes je její použití doslova masivní. Umožňuje dnes předávání binárních dat v souborech po zavedení datového typu multipart/form-data.

2.2.1 Výhody SOAP protokolu a jeho vývoj

Když se objevil jazyk XML, bylo už poměrně blízko k tomu, aby se data mohla přenášet přes HTTP v XML strukturách. Tyto struktury byly pojmenovány jako SOAP zprávy (Simple Object Access Protocol). Tento posun přinesl několik výhod:

- formát dat je textový a tedy platformě nezávislý
- XML formát umožňuje přenášet i složitější struktury dat proti pouhým primitivním typům
- typy dat je možné dále vytvářet kombinací předchozích pomocí použití definic z XML schémat
- nedochází také ke konfliktům jednotlivých definic díky systému jmenných prostorů
- je možno takto předávat dokonce definice objektů nebo celé kolekce definic objektů

Tvůrcem původní struktury SOAP protokolu byla firma Microsoft, která dala vzniknout tomuto standardu v roce 1998. Vedle SOAP protokolu se však podobně jako u jiných standardů zpočátku objevilo velké množství podobných protokolů, např. XML-RPC od firmy UserLand Software, WDDX od firmy Macromedia, XMI od konsorcia OMG. Konsorcium W3C zařadilo SOAP do svého programu v roce 2000 a vytvořilo verzi SOAP 1.2

Původně uměl SOAP přenášet zprávy pouze přes HTTP protokol, ale postupně do něj byla přidána podpora také pro SMTP, takže se rozšířila možnost přenosu komunikace také přes mailové zprávy.

2.2.2 Struktura SOAP zpráv

Podmínkou, kterou SOAP jasně stanovil je, že každá funkce byla definována jmenným prostorem. V terminologii vývojáře to znamená, že jmenný prostor odpovídá objektu (neboli třídě) a funkce metodě tohoto objektu. Uveďme příklad SOAP zprávy.

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
   <n:priority>1</n:priority>
   <n:expires>2001-06-22T14:00:00-05:00</n:expires>
  </n:alertcontrol>
 </env:Header>
 <env:Body>
  <m:alert xmlns:m="http://example.org/alert">
   <m:msg>Pick up Mary at school at 2pm</m:msg>
  </m:alert>
 </env:Body>
</env:Envelope>
```

Obrázek 4: Příklad struktury SOAP zprávy

O interpretaci SOAP zprávy se stará na serveru běžící příslušný program (např. Apache Axis). V uvedeném příkladu dojde k vyskočení zprávy o vyzvednutí ratolesti ze školy k uvedenému času.

2.3 Popis webové služby v jazyce WSDL

Webová služba je ve své obecné podobě definována pomocí XML struktury, která je též označována jako samostatný jazyk. WSDL (jazyk popisu webových služeb) obsahuje ve své struktuře všechny funkce, které webová služba využívá, včetně jejich parametrů a hodnot, které tyto funkce vracejí. Samotné WSDL je postaveno na XML a jeho specifických vlastnostech, jako jsou jmenné prostory a schémata, která byla popsána v první kapitole toho textu.

Historicky vzniklo WSDL sloučením samostatných jazyků NASSL od firmy IBM, SCL of firmy Microsoft a SDL firmy Ariba. Toto sloučení dalo za vznik WSDL 1.1, což byla specifikace, které si později všimlo konsorcium W3C, které začalo pracovat na verzi WSDL 1.2 [13].

2.3.1 Struktura WSDL

Struktura WSDL může vypadat na první pohled poměrně složitě, ale tak jako všechny XML dokumenty má jasnou strukturu. Nepřehledně působí především množství jmenných prostorů, které jednoznačně identifikují elementy. Konvence určuje vždy jako příponu souboru WSDL zkratku tohoto názvu. Pro přehlednost uvedu příklad jednoduché WSDL struktury:

```
<?xml version="1.0"?>
<definitions name="StockQuote"
targetNamespace="http://example.com/stockquote/service"
          xmlns:tns="http://example.com/stockquote/service"
          xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
          xmlns:defs="http://example.com/stockquote/definitions"
          xmlns="http://schemas.xmlsoap.org/wsdl/">
   <import namespace="http://example.com/stockquote/definitions"
           location="http://example.com/stockquote/stockquote.wsdl"/>
    <binding name="StockQuoteSoapBinding" type="defs:StockQuotePortType">
        <soap:binding style="document" trans-
port="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetLastTradePrice">
           <soap:operation soapActi-
on="http://example.com/GetLastTradePrice"/>
           <input>
               <soap:body use="literal"/>
           </input>
           <output>
               <soap:body use="literal"/>
           </output>
        </operation>
    </binding>
    <service name="StockQuoteService">
        <documentation>My first service</documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteBinding">
           <soap:address location="http://example.com/stockquote"/>
        </port>
    </service>
</definitions>
```

Obrázek 5: Ukázka struktury WSDL

WSDL dokument obsahuje popis jedné webové služby. Ta je definována elementem service. Každá webová služba má brány - jednu nebo i více. Ty jsou definovány elementem port. Aby mohla být každá z bran zavolána, musí být nadefinován způsob, jakým se tak stane. Toto zajišťuje vazba definovaná atributem binding. Způsob volání definuje vnořený element. V našem případě jde o způsob volání SOAP zprávou přes adresu v umístění, které definuje atribut location. Pro jednu službu může existovat tedy i více bran s různými vazbami. Toto vše nám volnost WSDL popisu dovoluje.

Dále je v našem příkladu definováno rozhraní pomocí elementu binding, kam se odkazuje vazba z definice služby. Rozhraní je souhrnem operací (operation), které může webová služba volat. Každá z těchto operací má ještě zvlášť nadefinováno, jaké vstupní a výstupní zprávy bude obsahovat. V řeči programátora existují tedy základní příměry. Rozhraní je analogický prvek k objektu (neboli třídě), operace jsou metody a vstupní a výstupní zprávy jsou parametry nebo návratové hodnoty, které tyto metody využívají. Pro lepší přehlednost uvedu na často uváděný příklad webové služby a její objektové reprezentace.

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

```
      </wsdl:input>
      <wsdl:output name='Secti'>
        <soap:body use='encoded'
          encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'>
      </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>
```

Obrázek 6: Příklad WSDL struktury pro jednoduchou třídu

Jednotlivé typy parametrů funkcí a jejich návratových hodnoty jsou také definovány pomocí XML schéma. V našem případě pomocí XSD. Jako primitivní typy jsou podporovány následující: string, int, float, decimal, boolean, Base64, datetime, duration, anyURI. WSDL dále podporuje také celá čísla různého rozsahu (int, byte, short, long). Všechny tyto primitivní typy je možné také sdružovat do struktur a výčtových typů.

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

Obrázek 7: Příklad jednoduché třídy s jednou metodou v jazyce Java

2.3.2 Nástroje pro práci s WSDL

Psát WSDL dokument ručně může být velmi obtížné a nepraktické. Proto vznikla v minulosti (a dále vzniká) celá řada více či méně kvalitních nástrojů pro vytváření, editaci a celkovou práci s WSDL. Vůbec nejpohodlnějším způsobem jak vytvořit WSDL strukturu je její generování z hotového zdrojového kódu konkrétního jazyka, jak již bylo uvedeno předcházející kapitole. K této činnosti se objevila velká spousta nástrojů integrovaných přímo

do vývojových prostředí, která s webovými službami mohou pracovat, ať již se jedná o .NET nebo Javu.

Na poli .NET jde především o editory ve Visual Studiu, které velmi dovedně zpracovávají vlastní postup vytvoření WSDL dokumentu a vývojář tak s vlastním obsahem struktury vůbec nemusí přijít do styku.

V Javě je situace velmi obdobná. V prostředí NetBeans IDE je balík nástrojů pro práci s WSDL integrován přímo do prostředí již v základní verzi. V Eclipse existuje velice komplexní skupina editorů a utilit, které je možné nechat doinstalovat (např. pomocí automatického update IDE). Tyto nástroje jsou pojmenovány jako WTP (Web Tool Project).

Všechny výše uvedené implementační nástroje mají společné možnosti práce s WSDL. Jsou to především samotný generátor WSDL, generátor kódu pro klienta služby, generátor pro serverovou implementaci služby, aplikační server, na kterém může daná služba běžet.

2.4 Vyhledávání webových služeb

Webové služby, které již jsou vytvořeny a běží někde v internetu na některém ze serverů často mohou poskytovat velice kvalitní výsledky a zajímavé funkce. Jak se ale o takovýchto službách dovědět, abychom je mohli využívat?

Řešením tohoto problému se stal vznik několika veřejných katalogů, kde jsou služby, jejich umístění a jejich popis uvedeny v předem známém formátu a lze tak velmi dobře a snadno vyhledávat vhodné služby pro naše využití.

Pak stačí v tomto katalogu najít příslušnou službu a zjistit její funkci. Z příslušného WSDL popisu služby potom již není problém negenerovat kostru (skeleton) aplikace klienta, ať již jako běžné desktopové aplikace, či webové aplikace.

Posledním standardem, který se v souvislosti s webovými službami dostává do popředí je UDDI (Universal Description, Discovery and Integration). Tento standard zpřístupňuje veřejnou databázi, do které je možno registrovat jednotlivé webové služby a jejich popisy a také v těchto databázích posléze vyhledávat. Centrální databáze mají na starost dvě velké firmy: IBM a Microsoft.

3 IMPLEMENTACE WEBOVÝCH SLUŽEB V PROSTŘEDÍ .NET FRAMEWORK

S trochou nadsázky se dá říci, že kolem webových služeb se v prostředí .NET Framework všechno točí. Toto prostředí pro vývoj nejrůznějších typů aplikací je jakousi odpovědí firmy Microsoft na úspěch Javy. Zatímco Java šla historicky trochu jinou cestou – především platformní nezávislosti a nejrůznější technologie byly dodávány až posléze, .NET se již zpočátku soustředil na možnosti využití webových služeb a XML formátu obecně. Podpora webových služeb je tedy již pevnou součástí .NET a tato podpora nenese speciální označení, jako v případě Javy.

Knihovny .NET tedy zahrnují implementace pro nejrůznější stávající programovací jazyky, jako jsou Visual Basic, C++, ASP apod. Pro tyto jazyky byly vytvořeny jejich následovníci vždy s přídomkem .NET. Důvod byl historicky prostý: usnadnit programátorům přechod na platformu .NET ze stávajícího prostředí.

Aplikace vytvořené pomocí .NET, na rozdíl od současných aplikací, již nejsou překládány přímo do kódu pro daný procesor, ale jsou překládány do mezijazyka Microsoft Intermediate Language (MSIL).

Všechny tyto nové programovací jazyky jsou založeny na CLR (Common Language Runtime), který je sdílen všemi aplikacemi běžícími na knihovnách .NET Framework a tím pádem je možné použít jakýkoliv z předešlých jazyků, neboť mají v CLR společný základ.

Pro .NET byl ovšem vytvořen také zcela nový programovací jazyk C#. Zde již Microsoft inspiraci Javou nepopře, nepoť C# je Javě ve své podstatě velmi podobný a obsahuje také podobné mechanismy programování.

3.1 Webové služby ukryté v .NET Framework

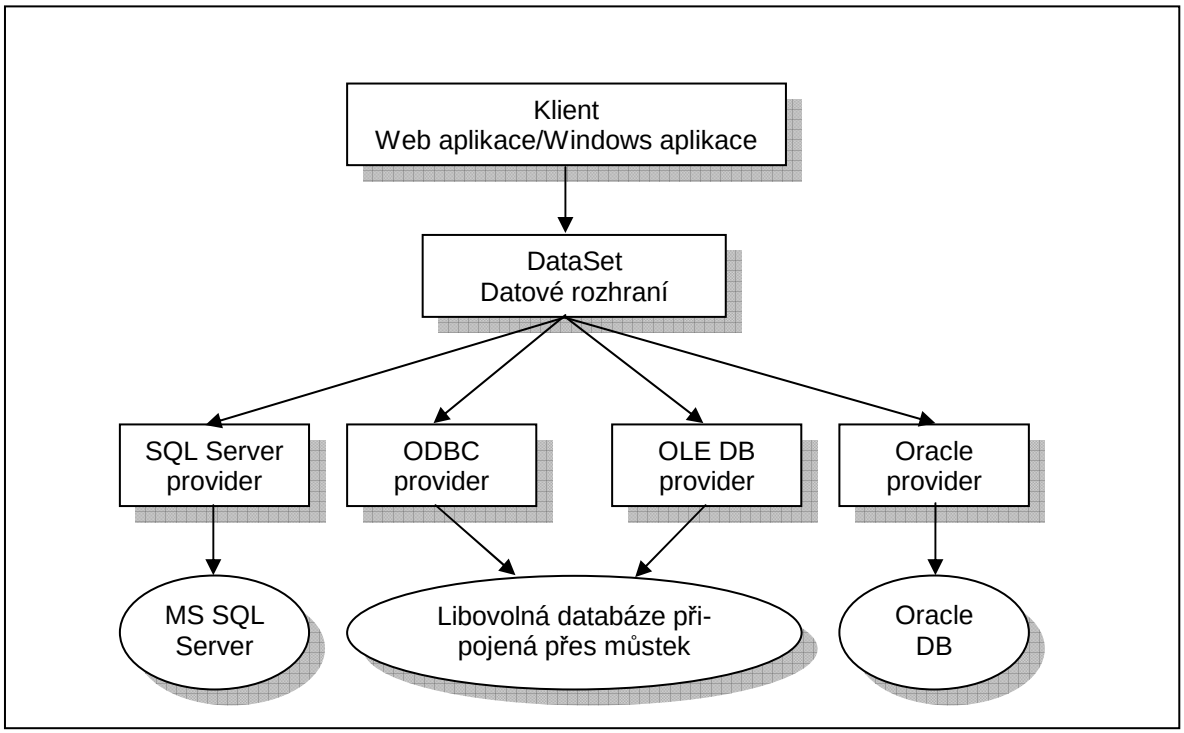
Klíčovou vlastností .NET je schopnost snadno a rychle vytvářet webové služby. K tomu můžeme použít prakticky všechny programovací jazyky, které v sobě platforma .NET implementuje.

Na rozdíl od Javy – kde je podpora webových služeb dodána v rámci různých řešení (např. Apache nebo SUN), .NET tuto podporu obsahuje již v základu a po nainstalovaní knihoven .NET Framework se programátor nemusí již starat o integraci dalších modulů.

To ovšem neznamená, že by se základ webových služeb a jejich technické pozadí nějak výrazně lišilo od ostatních implementací. Stále se používají základní technologie pro komunikaci pomocí XML zpráv (SOAP) a popis webových služeb (WSDL). Firma Microsoft však více dbá na jakési „skrytí“ tohoto pozadí a jednoduchý vývoj těchto služeb sestávající se z vytvoření služby a její nahrání na server a následně z negenerování a doplňující úpravy klienta webové služby.

3.2 ADO.NET – rozhraní .NET pro připojení databáze

Když potřebuje aplikace napsaná v některém z rodiny jazyků v .NET přistupovat k datovému zdroji, nabízí se použití komponenty ADO.NET. Tato kolekce nejrůznějších typů, rozhraní, výčtových typů a tříd umožňuje aplikacím napojení do databází nejrůznějších typů. Tento přístup k datům je možné provést standardizovaným způsobem podobně jako je tomu u konkurenční technologie JDBC[5]. Dá se také říci, že ADO.NET vznikla jako odpověď na existenci technologie JDBC [19].



Obrázek 8: Schéma architektury připojení ADO.NET

[5] O této technologii se píše dále v textu v kapitole 4.

ADO.NET je nástupcem technologie ADO firmy Microsoft. Ta byla určena pro standardizovaný přístup k datům v aplikacích, které byly programovány v nativním kódu a tedy před příchodem .NET Framework. Velká část architektury ADO byla přepracována tak aby umožňovala jednodušší a intuitivnější přístup k datům sjednoceným pod společné rozhraní. Zároveň byla přidána podpora XML formátu, která je velmi důležitou vlastností pro práci s webovými službami.

Základní knihovna tříd obsahuje několik tzv. data providerů, kteří implementují nejčastěji se vyskytující datové zdroje:

- System.Data.SqlClient – jmenný prostor obsahující rozhraní pro práci MS SQL Serverem.
- System.Data.Odbc – jmenný prostor pro připojení libovolného datového zdroje přes prostředníka – rozhraní ODBC
- Systém.Data.OleDb – jmenný prostor pro připojení datového zdroje přes rozhraní OLE DB
- Systém.Data.Oracle – jmenný prostor pro připojení databáze Oracle

Je patrné, že firma Microsoft zahrnula do standardních řešení především své technologie. K výše uvedeným providerům je však možné samozřejmě přidat další externí providery ve formě knihoven pro zatím nepodporované databáze.

Z výše uvedených skutečností vyplývá, že existují podobné možnosti a principy připojení k databázím jako v případě JDBC.

Mnohem podrobnější popis práce s databázemi na platformě .NET je možno také najít v knize [4], vlastní popis programování v .NET potom přináší kniha [5].

3.3 Vývojové prostředí Visual Studio .NET Express

Výrazným pomocníkem pro programování vlastních webových služeb a jejich klientů je vývojové prostředí firmy Microsoft Visual Studio .NET Express. Visual Studio tak jako většina softwarových produktů prošla dlouhým vývojem. První verze Visual Studia umožňovaly programování pouze v jazyku C++. K dalším rozšířením docházelo až během následujících let.

Milníkem ve vývoji se stala verze 6.0, která přinesla podporu jazyka Visual Basic, jehož vznik byl reakcí na existenci konkurenčních RAD (Rapid Application Development) nástrojů od firmy Borland postavených na knihovně VLC (Visual komponent Library). Dalším významným vývojovým prvkem se stala integrace technologie .NET a s tím souvisejících nových jazyků. Tento stav tak prakticky trvá dodnes.

Visual Studio .NET Express je novinkou v rodině vývojových nástrojů od Microsoftu. Na rozdíl od verze Visual Studio .NET je verze Express volně stažitelná z internetu. Jedná se o produkt, který Microsoft prezentuje jako jednoduchý, intuitivní RAD nástroj, který podporuje všechny moderní způsoby programování. Proti plné verzi je ale také omezen. Asi největším problémem je, že Express edice neobsahuje plný web server Internet Information Server (IIS), ale pouze runtime web server, pro testování a debug webových služeb.

Vlastní edice je rozdělena na několik částí:

- Visual Web Developer 2005 Express – nástroj pro vývoj dynamických webových stránek v technologii ASP.NET a webových služeb v libovolném programovacím jazyku.
- Visual C# 2005 Express – základní nástroj VS.NET, pro vývoj desktopových aplikací v jazyce C#. Podobně existují i další části pro Visual Basic, C++ a J#.
- SQL Server Express – zjednodušená verze MS SQL Serveru.

4 MOŽNOSTI IMPLEMENTACE WEBOVÝCH SLUŽEB V PROSTŘEDÍ JAVA

Možnosti implementace webových služeb jsou k dnešnímu dni poměrně široké. Je to dáno především otevřeností přístupu k tomuto druhu programování. Rovněž otázka ceny zde není palčivá, jako u jiných softwarových řešení. Především v oblasti Javy se objevuje dostatek kvalitních řešení, vyvíjených pro poskytování zdarma a také pod licencí open-source. Obsahem této kapitoly bude seznámení s nejčastějšími možnostmi implementace webových služeb, ke kterým se můžeme dostat. Nejčastějšími ovšem neznamená jedinými. Proto přehled ostatních možností implementace webových služeb je možno najít na adrese: http://www.soapware.org/directory/4/implementations.

4.1 Axis + Tomcat

Jedním z nejčastěji používaných a jmenovaných řešení pro webové služby je rozhodně kombinace dvou produktů konsorcia Apache Foundation.

Tyto nástroje vytváří vlastní nezávislé běhové prostředí pro webové služby v prostředí jazyka Java. Postupně se s oběma produkty seznámíme a představíme si je.

4.1.1 Apache Tomcat

Apache Tomcat je servletový kontejner, který byl původně vyvinut v rámci Apache Foundation jako součást projektu Jakarta, jehož cílem bylo vytvářet open-source řešení serverových aplikací pro Java platformu. V současné době je projekt Jakarta pravděpodobně největším a nejkomplexnějším open-source řešením pro serverové aplikace napsané v Javě. V dnešní době již došlo k vyčlenění serveru Tomcat z projektu Jakarta a nadále je vyvíjen samostatně.

Tomcat je oficiální referenční implementací specifikací JSP[6] a Java Servlets[7]. Je tedy jakýmsi zásobníkem (odtud výše zmíněný termín kontejner) jednotlivých servletů. K základním funkcím Tomcatu patří tedy spouštění, běh, ukončování jednotlivých servletů.

[6] JavaServer Pages je technologie Javy, která umožňuje vývojářům dynamicky generovat zdrojové texty HTML stránek, XML struktur atd. Zjednodušeně se dá také říci, že jde o obdobu PHP či ASP.

Tomcat jako takový se dočkal na tomto poli velkého uznání a také obliby. Je třeba podotknout, že je zde rozdíl oproti klasickému webovému serveru Apache, který je nejčastěji zmiňován v souvislosti s provozováním webových stránek obvykle s technologií PHP a MySQL. Přestože je Tomcat naprosto samostatným serverem, je možné jej použít také v kombinaci s klasickým Apache Web serverem. Výhoda je potom především ve výkonnostním nárůstu, jelikož Apache Web server zpracovává klasické požadavky – běžné webové stránky, ostatní skriptovací jazyky apod. a serveru Tomcat předává ke zpracování pouze požadavky pro jednotlivé servlety.

Tomcat je vyvíjen pod záštitou společnosti SUN. Různé verze serveru Tomcat nabízejí podporu pro různé verze výše uvedených specifikací. Proto jsou běžně na webových stránkách projektu přístupno zároveň více stabilních vývojových verzí serveru Tomcat. Jejich rozdělení ukazuje následující tabulka:

Tabulka 2: Specifikace Java technologií a jejich příslušnost k verzím Tomcat

Apache Tomcat verze	Java Servlet specifikace	JSP specifikace
5.5	2.4	2.0
4.1	2.3	1.2
3.3	2.2	1.1

4.1.2 Apache Axis

Axis je implementace SOAP protokolu v jazyce Java. Obsahuje též kolekci nástrojů pro práci s webovými službami. Mezi dva nejdůležitější nástroje Axisu patří Java2WSDL, pomocí kterého se generují popisy webových služeb v jazyce WSDL ze zdrojových textů v Javě. Inverzním nástrojem je potom WSDL2Java, pomocí kterého se generují klientské serverové zdrojové kódy rovněž v Javě.

U Axisu není oficiálně podporován režim použití bez servletového kontejneru. Přesto tato možnost existuje, ale je tvůrci označena jako tajná a silně se nedoporučuje používat. Slouží

[7] Java Servlet Technology – technologie Servletů. Servlet je program napsaný v jazyce Java , který běží na serveru. Je schopen přijímat http požadavky a je schopen na ně stejnou cestou, tedy přes protokol http, také odpovídat.

jen pro interní účely testování. Je možné, že se ale s touto variantou v následujících letech setkáme.

Protože tedy Axis „nedokáže“ běžet sám, potřebuje ke svému provozu server, který je zároveň servletovým kontejnerem. Tomuto požadavku bez výhrad odpovídá v minulé kapitole uvedený produkt Tomcat. Výhody tohoto spojení jsou nasnadě. Oba produkty jsou úzce spojeny a dá se předpokládat bezproblémová spolupráce. Ovšem protože je Axis napsán v Javě a projektován k obecnému použití, můžeme jej spustit také na jiných serverech.

4.2 Java WSDP + Sun Application Web Server

Druhou velmi často používanou možností pro práci s webovými službami je kombinace nástrojů od firmy SUN. Základem podobně jako v předchozím případě je aplikační server Sun Application Web Server. Rozšířením funkcionality o práci s webovými službami je potom Java Web Services Developer Pack.

4.2.1 Sun Application Web Server

Aplikační server, který představuje výkonnou platformu pro vývoj velkého množství aplikací postavených na nejrůznějších javovských technologiích skloubených do balíku J2EE. Pro tuto práci je důležitá především podpora pro vývoj a provoz webových aplikací a webových služeb.

Mezi základní výhody Sun Application Web Serveru patří:

- Kompatibilita s J2EE 1.4[8]
- Je zdarma pro vývoj a deployement.
- Jednoduše integrovatelný do vývojových prostředí (především pak do NetBeans IDE od firmy SUN)
- Používá velmi pohodlnou administrační konzoli, jejíž uživatelské rozhraní je postaveno na webových stránkách s maximální přehledností a uživatelským komfortem.

4.2.2 Java WSDP

Java WSDP je kompletní sada API a nástrojů vyvinutých v rámci Java Community Process (JCP). Ty vývojářům umožňují budovat přenositelné, univerzální a otevřené webové služby a distribuované podnikové aplikace. Java WSDP zahrnuje nástroje pro budování SOAP zpráv, podporu WSDL a podporu přístupu k registrům UDDI a registrům a zdrojům ebXML.

Balík zahrnuje také aplikační server Apache Tomcat a knihovny JavaServer Pages (JSP) Standard Tag Libraries. Navíc jsou do tohoto rozšířeného souboru Java WSDP plně integrována všechna API pro XML z dříve vydaného balíku Java XML Pack, takže je nyní k dispozici vše v jediném výhodném balíku integrovaných nástrojů pro webové služby na platformě Java. Technologie je prostřednictvím Java komunity již podporována významnými výrobci softwarových nástrojů.

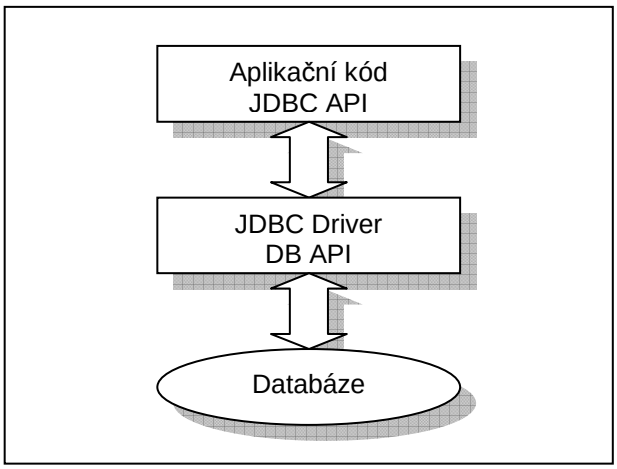
4.3 JDBC – rozhraní Javy pro připojení databáze

JDBC [18] představuje v Javě univerzální rozhraní pro spojení s relačními databázemi. Základní princip spočívá ve vytvoření mezivrstvy mezi samotnou databází a aplikací. JDBC ovladač překládá požadavky vznášené na databázi do nativního tvaru. Výhody tohoto přístupu by se daly označit jako 2J:

- **Jednoduchost** – připojení k databázi zajišťuje poměrně triviální API, které je navíc velmi dobře zdokumentováno na stránkách firmy SUN, případně jiného vydavatele Javy.
- **Jednotnost** – vývojář nemusí řešit rozdílnost přístupů a používání jednotlivých databází. Prostě se k databázi připojí přes rozhraní JDBC a pomocí tohoto rozhraní získá také výsledky vrácené databází. Také v případě, že se rozhodne vývojář změnit databázi, není třeba dělat úpravy do zdrojových kódů, stačí zaměnit JDBC ovladač.

[8] Alespoň jak tvrdí tvůrce firma SUN – to se ovšem dá předpokládat, protože přece jenom by mělo toto řešení mít k SUNovské implementaci Javy daleko nejblíže.

JDBC jak již napovídá podobnost názvů principielně vychází z ODBC, což je rozhraní firmy Microsoft. Dnes se toto rozhraní používá již méně často, přesto je stále velmi oblíbené především u menšíc informačních systémů. Samotný Microsoft tento koncept ve svých nejnovějších technologiích opouští a nahrazuje jej novými a modernějšími standardy.



Obrázek 9: Architektura JDBC

J2SE SDK od SUNu obsahuje standardně ovladače JDBC, které umožňují připojení na ODBC ovladače. Původně byl tento ovladač vyvíjen samostatně, ale později došlo k jeho zařazení do SDK.

Architektura JDBC zobrazená výše (viz. Obrázek 9) je sice zjednodušeným pohledem na strukturu JDBC, avšak pro naši představu je dostatečné. Logická struktura JDBC je závislá na druhu ovladače, který je použit. Rozeznáváme základní čtyři druhy ovladače:

- Typ 1 – tento typ ovladače je nejjednodušší variantou a využívá ke spojení s databází ovladač ODBC. K tomuto ovladači se připojuje přes tzv. JDBC-ODBC bridže. Jedná se opravdu doslova o přemostění. Výhodou je velká univerzálnost[9], která je zaplacena jaksi „delší“ cestou komunikace mezi Javou a databází. Tím, že v celém řetězci přibývá další prostředník, dochází ke snížení výkonu aplikace.
- Typ 2 – je jistým zobecněním předchozího typu. Jedná se o přemostění na jakýkoliv jiný ovladač databáze, nainstalovaný na systému. Prostředníků v komunikačním řetězci proti prvnímu případu neubylo, dokonce došlo ke konkretizování druhého pro-

[9] Ovšem pouze v rámci produktů Microsoft a prostředí Windows

středníka v podobě přímého ovladače, avšak výhodou tohoto řešení je zvýšení výkonu spojení s databází.

- Typ 3 – tento případ nepoužívá žádný nativní ovladač jako prostředník. To je zjevná výhoda pro vývojáře, co chtějí udržet aplikaci platformě nezávislou. JDBC v tomto případě převádí dotazy na síťový protokol, pomocí kterého se spojuje s databázovým serverem. Výhodou je velké zvýšení efektivity a možnosti připojení k velké sadě různých databázových systémů (Oracle, MS SQL, Sybase apod.).
- Typ 4 – posledním typem je ovladač napsaný čistě v Javě optimalizovaný přímo na konkrétní databázový systém. Výhoda spočívá především v tom, že klientská aplikace nemusí být nijak konfigurována. Vhodné především pro menší systémy.

4.4 Vývojová prostředí Javy

Na poli vývojových prostředí (IDE) je Java velice silná. Podobně jako pro C++ zde existuje velké množství vývojových prostředí. Velké množství z nich je též velmi kvalitních. Tato diplomová práce se zaměřuje především na softwarové prostředky, které jsou k dispozici zdarma, nejlépe jako open-source. Zde jsou příklady nejčastěji se vyskytujících vývojových prostředí:

- NetBeans (http://www.netbeans.org)
- Eclipse (http://www.eclipse.org)

4.4.1 NetBeans IDE

Vývojové prostředí firmy SUN. Dříve stála za tímto prostředím firma NetBeans s českými kořeny. SUN ovšem tuto tento produkt převzal a dnes je jedním z největších hitů. Sun souběžně dál vyvíjí své vlastní prostředí SunONE.

NetBeans má tedy nejblíže k edici Javy, která je velmi často používaná. Je vyvíjen jako open-source projekt. Má obrovskou uživatelskou základnu a často se s ním setkáváme u profesionálních vývojářů. Jedná se o bezplatně šířený produkt.

NetBeans je úspěšným vývojovým prostředím z několika důvodů. Samotný uživatelský komfort, který je mimo jiné vynikající, by sám o sobě nestačil. NetBeans přichází s integrací velkého množství nástrojů pro vývoj všech možných druhů aplikací v Javě a

nejen v ní. S trochou nadsázky se dá říct, že není technologie, kterou by toto vývojové prostředí nepodporovalo. NetBeans reaguje velmi pružně také díky modulární struktuře, kterou převzal částečně po vzoru vývojového prostředí Eclipse.

Pro webové služby je NetBeans vybaveno velmi dobře. Jednak ve své poslední verzi obsahuje přímo integrované nástroje pro běh webových služeb od konsorcia Apache – server Tomcat se SOAP interpretem Axis. Dále umožňuje velmi triviální integraci Sun Application Web Server. Postup nastavení je blíže popsán v praktické části tohoto textu.

4.4.2 Eclipse

Vývojové prostředí Eclipse je dalším velkým hráčem na poli IDE pro Javu. Stejně jako u NetBeans se jedná o projekt, který je zdarma. Eclipse je od začátku projektováno jako platforma pro vývoj v různých jazycích. Největších úspěchů však dosáhlo právě na poli Javy, pro kterou je původně navrženo.

Z Eclipse podobně jako za NetBeans stojí velká firma. V tomto případě se jedná o IBM. Nespornou výhodou Eclipse oproti NetBeans je, že jeho jádro je napsáno pro různé operační systémy nativně a tudíž jako aplikace je Eclipse přece jen svižnější především na méně výkonných stanicích.

Eclipse se však od NetBeans liší i v pojetí základní koncepce. Zatímco u NetBeans si stáhnete kompletní balík vývojového prostředí a velké spousty dalších nástrojů, Eclipse se k této věci staví přesně opačně. Základní vývojové prostředí umožňuje velmi dobrou práci, avšak pokud potřebujeme další funkce, je třeba sáhnout mezi pluginy, které jsou dostupné v opravdu velké míře. Eclipse tak například v základní instalaci neobsahuje ani RAD nástroje pro jednoduchý vizuální návrh formulářů. Pluginy je nutné tedy vyhledat a doinstalovat. Výhodou ovšem je, že co uživatel skutečně nepotřebuje, nemusí v Eclipse mít.

Pro vývoj webových služeb je potřeba mít v prostředí Eclipse tedy následující součásti:

- GEF (Graphical Editor Framework) – komponenta obsahující nástroje pro tvorbu komplexních grafických editorů.
- EMF (Eclipse Modeling Framework) – komponenta obsahující nástroje pro datové modelování a generování zdrojových kódů z těchto modelů.
- JEM (Java EMF Model) – plugin pro převod modelování do java kódu.

- WTP (Web Tools Platform) – modul pro rozšíření funkcionality Eclipse pro vývoj webových aplikací postavených na J2EE. Obsahuje editory zdrojových kódů, grafické editory pro práci s XML a WSDL, automatické návrháře pro vývoj webových aplikací a webových služeb.

Tyto součásti mohou být staženy a nainstalovány automaticky pomocí nástroje Update Manager, který je obsažen v Eclipse verzi 3.1, nebo staženy každá zvlášť a nakopírována do instalačního adresáře Eclipse.

5 IMPLEMENTACE WEBOVÝCH SLUŽEB VE SKRIPTOVACÍM PHP JAZYKU

V práci s konkrétními implementacemi XML jako jsou SOAP zprávy či WSDL dokumenty v PHP podpora standardně není.

Na rozdíl od předcházejících komplexních řešení, v PHP panuje trochu jiná situace. Především PHP jako skriptovací jazyk na straně serveru nedokáže fungovat samostatně a tudíž v něm, alespoň co se týká klasické distribuce PHP nelze vytvořit desktopovou aplikaci. Zde je tedy výběr klienta omezen pouze na skript běžící na serveru.

Vlastní připojení k databázím obsahuje PHP také. Jeho podrobný popis je dostupný v mnoha publikacích (např. [6]).

5.1 Knihovna nuSOAP

Existuje open-source řešení NuSOAP, což je balík tříd pro PHP, který umožňuje vytváření webových služeb v PHP a práci se SOAP zprávami. Jedná se o prostou knihovnu napsanou v jazyku PHP, jejímž autorem je Dietrich Ayala. Domovská stránka tohoto autora sídlí na adrese: `http://dietrich.ganx4.com/nusoap/index.php`, odkud je možné též tuto volně šiřitelkou open-source knihovnu stánout.

Knihovna má podobu pouze jednoho souboru `nusoap.php`. Tento soubor se vždy přidá k vlastnímu skriptu a pomocí příkazů knihovny dojde k vytvoření příslušné SOAP zprávy, která putuje ze serveru nebo naopak.

Knihovna vypadá jen zdánlivě jednoduše, ovšem poskytuje veškeré základní důležité postupy pro práci s webovými službami. Jedná se především o tvorbu jak serverového, tak klientského kódu. Knihovna zajišťuje tvorbu SOAP zpráv, pomocí kterých mezi sebou komunikují serverové skripty v rolích služeb a klientů. Dále také může vývojář pomocí nuSOAP vytvořit plnohodnotný popis webové služby ve WSDL. Toto je nutné zejména pro případy, kdy chceme službu zveřejnit a poskytnout její popis pro snadnější vytváření klientů na jiných platformách.

Tato diplomová práce se na implementaci webových služeb v PHP dívá pouze zjednodušeně a uvádí základní příklad, jak vytvořit webovou službu i v rámci jednouchého prostředí, jakým PHP bezesporu je.

6 PROCES NÁVRHU WEBOVÉ SLUŽBY TECHNIKAMI UML

Každá webová služba je vlastně jakousi objektovou třídou, která obsahuje několik metod, přes které jsou jednak parametry webové službě předávány ke zpracování a také pomocí návratových hodnot vrací tato webová služba výsledky.

Z výše uvedeného vyplývá, že webová služba stejně jako ostatní třídy musí být nějakým způsobem navržena, promyšlena a v konečném stádiu implementována.

V případě jednoduché webové služby lze tyto kroky sloučit do prostého návrhu a naprogramování. Zkušenosti vývojářů složitějších systémů (především podnikových, ve kterých se webové služby začínají prosazovat nejvíce) ovšem hovoří pro použití některé z modelovacích technik, která umožní kvalitní návrh a pokud možno co nejmenší problémy při následné implementaci.

6.1 Modelovací technika UML

Modelovací technika nebo také jazyk UML (Unified Modeling Language) vznikla jako výsledek práce mnoha designérů přibližně v polovině 90. let. Jedná se o souhrn metod[10] k vyjádření analytických a návrhových modelů. UML jako takové umožňuje vizuální návrh, specifikaci, stavbu a dokumentaci softwarových systémů a to v předem stanovené, jednoznačné a přehledné podobě.

Pro lepší představu uvedu výběr činností spojených s modelováním pomocí UML:

- Stanovení požadavků – vyjádření přání funkčnosti nejčastěji od zákazníka. Zde rozlišujeme zdroje požadavků (kromě zákazníků může jít např. o softwarové a hardwarové vybavení).
- Označení případů užití – specifikace zachycující přesnou funkčnost, která bude informačním systémem (v našem případě webovou službou) pokryta. Stanovením případů užití dojde také k jednoznačnému vymezení rozsahu prací, kdy každý případ užití popisuje jeden ze způsobů použití systému.

[10] Nejčastěji grafických metod.

- Modelování tříd objektů – zde již dochází ke specifikaci vlastní implementace. Jak budou vypadat jednotlivé objekty (třídy) a jaké mezi sebou budou mít vztahy – toto blíže specifikují další modely objektové spolupráce
- Seskupení tříd – krok nezbytný při modelování větších systémů. Jedná se o namodelování větší celků (opět tříd), které seskupují třídy, které spolu vzájemně velmi často pracují do větších rozhraní.
- Stanovení stavových diagramů – techniky znázornění chování systému v různých stavech, ve kterých se může objevit. Hodí se zejména pro znázornění a uvědomění si chování objektů napříč klasickými případy užití.

Tento soubor technik nemusí být vždy použit celý včetně všech způsobů namodelování. V UML je důležité, že zde existuje možnost, jak všechny tyto problémy dostatečně popsat a zdokumentovat jednotným způsobem.

6.2 Vhodné nástroje pro modelování UML technikami

Nástroje pro modelování UML technikami se nazývají CASE (Computer Aided Software Engineering). V současnosti prakticky všechny ve světě rozšířené objektově orientované CASE nástroje vycházejí z modelovacího jazyka UML. Návrh rozsáhlejších informačních systémů se již bez těchto nástrojů nemůže obejít.

Jednotlivé diagramy UML je možné kreslit pomocí klasických kreslících nástrojů, avšak největší výhoda dnešních CASE nástrojů spočívá v propojení jednotlivých technik, tak aby se minimalizovaly chyby návrhu.

Jak je specifikováno v knize [3], dobrý CASE nástroj by měl obsahovat především následující funkcionality:

- Integrační mechanismy pro spojení jednotlivých modelovacích technik
- Podpora všech etap softwarového vývoje
- Podpora generování databázových skriptů
- Schopnost konverze objektových modelů do vývojového prostředí objektově orientovaných programovacích jazyků
- Podpora týmové práce analytiků a designérů včetně možnosti správy verzí.

Techniky UML dnes podporují především jednotlivá vývojová prostředí včetně těch, které byly uvedeny v předcházejících kapitolách – Visual Studio .NET, NetBeans, Eclipse. Podpora UML je implementována často jako integrované editory, případně v rámci vývojové platformy Eclipse jako doplňující pluginy.

II. PRAKTICKÁ ČÁST

7 NÁVRH WEBOVÉ SLUŽBY DVD FILMY

Praktická část této diplomové práce se zabývá konkrétním návrhem webové služby, její implementací na jednotlivých platformách, srovnáním náročnosti implementace a srovnáním výkonu těchto implementací.

Webová služba DVD filmy bude tedy postupně vytvořena:

- v prostředí .NET Framework za použití C# programovacího jazyku a vývojového nástroje Microsoft Visual Studio .NET 2005
- v prostředí J2EE za použití programovacího jazyku Java a implementace Apache Axis a běhového serveru Tomcat v prostředí Eclipse 3.1
- v prostředí J2EE za použití programovacího jazyku Java a implementace Sun WSDP a běhového serveru Sun Application Web Server ve vývojovém prostředí NetBeans 5.0
- v prostředí PHP za použití programovacího jazyku PHP a knihoven nuSOAP na serveru Apache.

Součástí webové služby bude vždy její klient, který bude mít za úkol změřit její výkon a to jako doba odezvy při zavolání webové služby přes jednu z jejích metod.

Uvedená webová služba je velmi jednoduchá, protože jejím úkolem není praktická implementace, ale pouze naznačení řešení vývoje na jednotlivých platformách. Proto níže navržená webová služba neobsahuje velké množství dalších metod, které by jistě přispěly k jejímu lepšímu využití v praxi, ale zároveň by došlo k poklesu přehlednosti uvedených postupů při vývoji.

S trochou nadsázky se dá tato služba označit jako složitější Hello World příklad.

7.1 Proces návrhu webové služby DVD filmy

Tato kapitola obsahuje kroky pro vytvoření návrhu webové služby podle specifikací jazyka UML uvedených v kapitole 6.1.

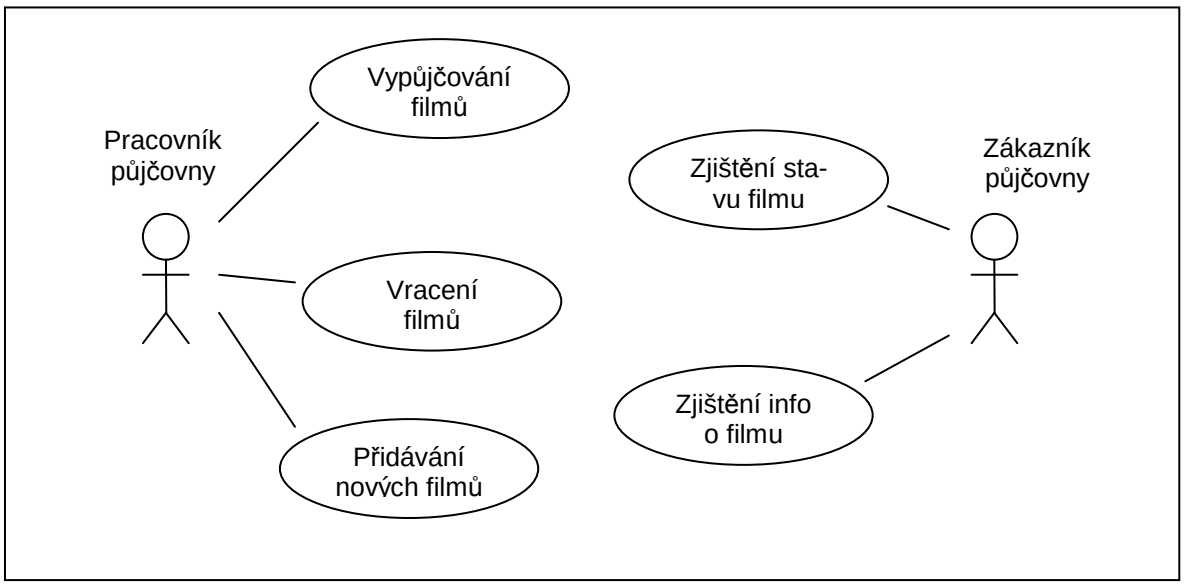
7.1.1 Požadavky na službu DVD filmy

Služba DVD filmy bude jednoduchou webovou službou umožňující přístup do databázové tabulky, ve které budou uloženy záznamy o jednotlivých filmech, které vlastní DVD půjčovna filmů. Tato služba by měla splňovat především následující požadavky:

1. vypůjčování jednotlivých titulů, obsažených v databázi
2. evidenci zrušení výpůjček
3. zjištění, jestli je konkrétní titul vypůjčen, či nikoli
4. v případě, že vypůjčen je, kdy se tak stalo – pro stanovení doby výpůjčky
5. zjištění informací o filmech uložených v databázi
6. přidávání nových titulů do databáze

7.1.2 Stanovení případů užití

Při stanovení případů užití je důležitá volba tzv. Aktérů. Tímto pojmem, jak je uvedeno v knize [3], je role v jaké přistupuje uživatel k webové službě. V zásadě můžeme jako aktéry webové služby DVD filmy označit zákazníka a pracovníka půjčovny. Zatímco zákazník je uživatel služby zvenčí, pracovník půjčovny má přístup také k systémovým operacím. Důležitým prvkem tohoto bodu návrhu tedy bude rozdělení jednotlivých případů užití z kapitoly 7.1.1.



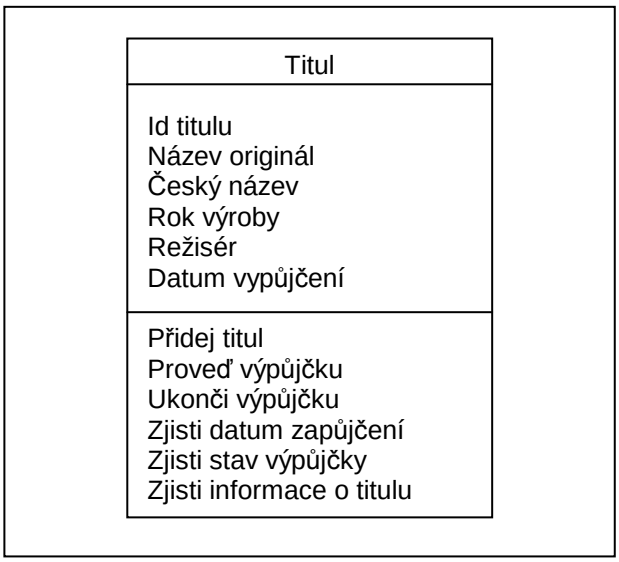
Obrázek 10: Stanovení aktéři a jejich činnosti

Podle správného postupu metodikou UML by měla po stanovení aktérů následovat specifikace jednotlivých kroků uvedených operací. Jelikož ale v tomto případě vždy jedna operace znamená jednu metodu webové služby, můžeme tento upřesňující krok přeskočit.

7.1.3 Modelování třídy aplikace

Opět je tento postup vhodný především pro větší případové studie, avšak zde můžeme vytvořit první konkrétní návrh třídy – objektu – webové služby, kterou později převedeme na zdrojový kód v konkrétním programovacím jazyce.

Tato třída bude ale přesto v něčem zvláštní. Projeví se absencí atributů, protože k vytvoření popisu webové služby je nebudeme potřebovat. Ve schématu však jednotlivé atributy obsaženy budou, protože mají význam z hlediska vytvoření databázové tabulky.



Obrázek 11: Objektová třída titul – DVD film

Z důvodu triviálnosti webové služby a tedy i celého objektového návrhu tady postup použití UML technik končí. Z výše uvedené třídy v následujících kapitolách dojde k vyčlenění seznamu atributů, které se stanou základem pro implementaci tabulky konkrétní databáze. Zbylé metody navržené třídy se stanou základem pro vytvoření rozhraní implementujícího webovou službu běžící na serveru.

7.2 Volba databázového zdroje

Z nepřeberného množství různých databázových serverů, které jsou dnes k dispozici jsem zvolil databázový server MySQL a to z několika důvodů:

- Velmi časté použití v praxi
- Možnost získat a používat tento server zdarma
- Vytvořené nástroje pro připojení standardních rozhraní jak pro ADO.NET tak také pro JDBC
- Dobrá podpora mySQL v jazyce PHP

Pro správné fungování MySQL ve všech plánovaných platformách bylo potřeba získat následující součásti:

1. vlastní instalátor databáze MySQL – instalace probíhá naprosto standardně jako u jiných aplikací pro Windows. Soubor instalátoru je možné stáhnout na adrese: `http://dev.mysql.com/downloads/mysql/5.0.html`
2. Connector/J – soubor tříd pro přístup JDBC k MySQL
3. Connector/Net – soubor tříd pro přístup ADO.NET k MySQL
4. MySQL Query Browser – aplikace pro pohodlnější zadávání sql dotazů v prostředí Windows. Tato není nutná, protože stejnou funkci, ale s menším komfortem odevede také příkazová konzole, která je součástí hlavní instalace MySQL serveru. Tuto aplikaci je možné stáhnout na: `http://dev.mysql.com/downloads/query-browser/1.1.html`

Oba Connectory je možné získat: `http://dev.mysql.com/downloads/connector/`

Po nainstalování databázového serveru a provedení nezbytných systémových nastavení – nastavení hlavního uživatelského účtu, nastavení hesla pro přístup k databázi jsem založil novou databázovou tabulku SQL příkazem:

```
CREATE TABLE `video` (
  `id` INT NOT NULL AUTO_INCREMENT ,
  `name` VARCHAR( 50 ) NOT NULL ,
  `locname` VARCHAR( 50 ) NOT NULL ,
  `year` INT NOT NULL ,
  `director` VARCHAR( 50 ) NOT NULL ,
  `borrowdate` DATE NOT NULL ,
PRIMARY KEY ( `id` )
);
```

Obrázek 12: SQL příkaz pro vytvoření nové tabulky k webové službě DVD film

8 IMPLEMENTACE WEBOVÉ SLUŽBY V .NET

Implementace webové služby v .NET se podobně jako v ostatních případech rozdělila na dva základní kroky. Vývoj vlastní implementace webové služby a vytvoření klienta webové služby.

Nejprve jsem v jazyce C# sestavil třídu, která se stala později základem pro webovou službu. Zdrojový kód této třídy můžete najít v příloze PII.

Každá z metod používá připojení k databázovému serveru MySQL pomocí rozhraní ADO.NET. Nadefinování tohoto připojení ukazuje Obrázek 13: Vytvoření připojení k databázi MySQL v jazyku C#. Toto připojení používá jmenný prostor `MySql.Data.MySqlClient`, který ovšem není standardní součástí knihoven .NET Framework. Je nutné si tyto knihovny opatřil na webu MySQL. Jedná se o nástroj Connector/Net, který obsahuje DLL knihovny, pro jednotlivé verze .NET. Příslušný soubor DLL tedy přidejte ke svému projektu do složky BIN.

```
using MySql.Data.MySqlClient;
...
       MySqlConnection conn =
            new MySqlConnection("server=localhost;database=dvdmovie;uid=root;pwd=");
```

Obrázek 13: Vytvoření připojení k databázi MySQL v jazyku C#

Pro samotný vývoj webové služby a její ladění nebudeme potřebovat web server, jelikož instalace Visual Studia .NET obsahuje vestavěný webový server, který se nastavuje a spouští automaticky. Ovšem pro vývoj klienta, již budeme potřebovat server, který je schopen běžet samostatně a nezávisle na vývojové prostředí. Takovým serverem může být např. Microsoft IIS – Internet Information Service (česky jako Internetová informační služba).

Pozor tento server není součástí instalace nástroje Visual Studio .NET, ale součástí operačního systému Windows XP – Professional[11]. Přitom při nastavení normální instalace se tento server nenainstaluje a je nutné jej doinstalovat přes nastavení *Ovládací panely* > *Přidat nebo odebrat programy* > *Přidat nebo odebrat součásti systému*.

[11] Pro verzi Windows XP Home Edition bohužel IIS není k dispozici a nejde ani doinstalovat.

8.1 Postup tvorby webové služby v .NET

Pro vytvoření webové služby a webového klienta v .NET budeme potřebovat aplikaci Microsoft Visual Web Developer .NET 2005 Express, pro vytvoření desktopové klientské aplikace budeme potřebovat aplikaci Microsoft Visual C# .NET 2005. Dále budeme postupovat podle následujících bodů, které byly inspirovány zdroji [20] a [21]:

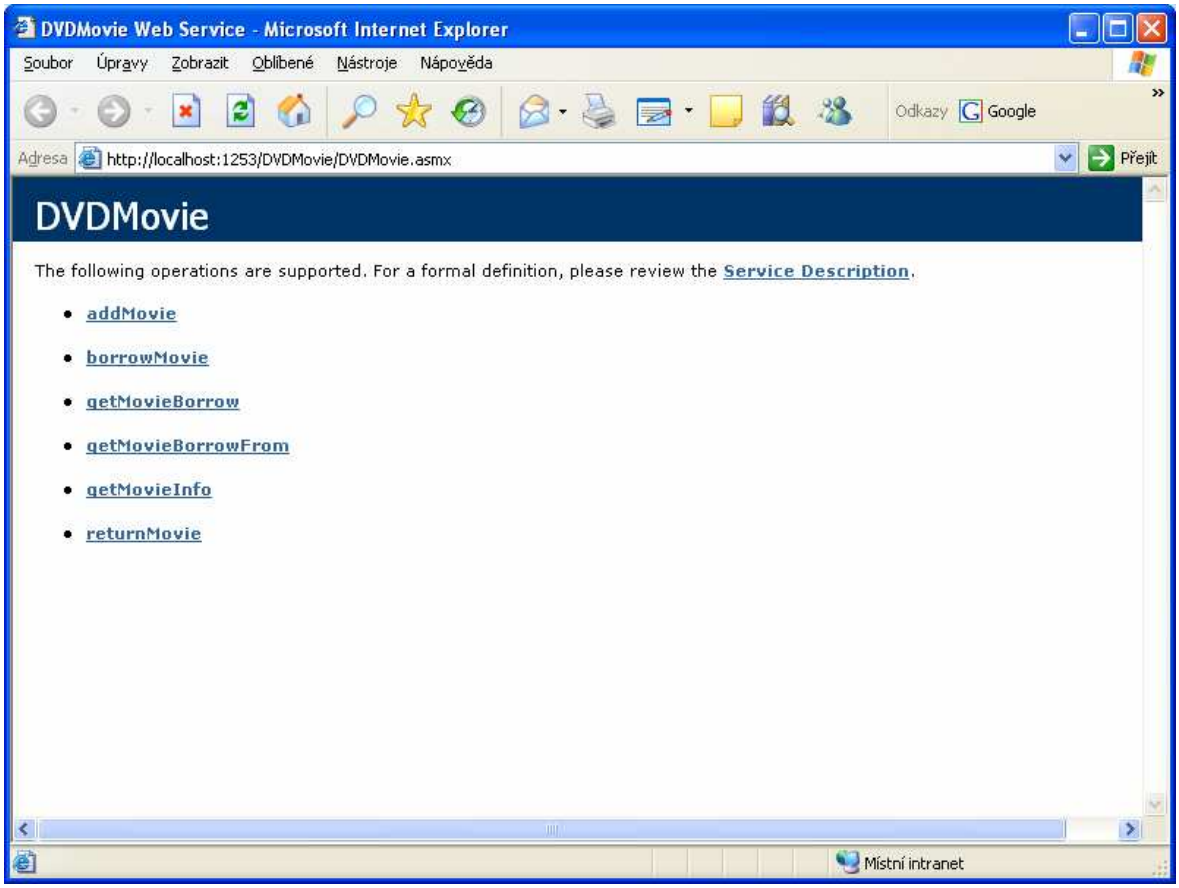
1. V hlavním menu aplikace zvolte *New Web Site* > *ASP.NET Web Service*. V dolní části dialogu můžete specifikovat druh umístění (na disku, na vzdáleném serveru apod.) a konkrétní složku umístění. Jako jazyk nezapomeňte nastavit C#. Klikněte na *OK*.

2. Dojde k vytvoření projektu *Hello World*, přičemž tuto metodu můžete nahradit kódem metod z přílohy PII.

3. Po spuštění samotné aplikace, dojde nejprve ke spuštění lokálního debug serveru na náhodném volném portu (standardně se jedná o 1253) a poté se spuštění vlastní webové služby v náhledovém zobrazení, tak jak představuje Obrázek 14: Spuštěná webová služba v prostředí Visual Web Developer .NET 2005

4. Kliknutím na odkaz *Service Description* můžete zobrazit soubor popisu služby WSDL. Rovněž kliknutím na jednotlivé odkazy metod služby můžete provádět testování, jestli služba vrací správné výsledky. Tyto výsledky jsou zobrazovány ve formátu SOAP zpráv.

5. Pokud je vše v pořádku a webová služba je hotova, měli byste ji nahrát na server pomocí nabídky *Web Site* > *Copy Web Site*. V zobrazeném okně klikněte na tlačítko *Connect* pro připojení k serveru a v okně výběru cílové oblasti v levém sloupci zvolte *Local IIS*. Poté nakopírujte zdrojové soubory na server.



Obrázek 14: Spuštěná webová služba v prostředí Visual Web Developer .NET 2005

8.2 Postup tvorby klienta webové služby v ASP.NET

Pro vytvoření klienta webové služby jako skriptové webové stránky v ASP.NET budeme podobně jako v předcházející kapitole potřebovat Visual Web Developer .NET 2005 Express. Postup je také velmi podobný, pouze s rozdílem definování referencí projektu na webovou službu [21].

1. Z hlavní nabídky zvolte *File* > *New Web Site*. Jako nový projekt označte volbu *ASP.NET Web Site*. Zkontrolujte nastavení programovacího jazyka a specifikujte umístění na svém disku. Klikněte na *OK*, čímž dojde v vytvoření projektu webové stránky.
2. Když je kostra webové aplikace nagenerována, v hlavním menu zvolte *Website* > *Add Web Reference*. Ve výběru položek zvolte možnost hledání webové služby na lokálním počítači. Pokud služba běží na IIS, zadejte adresu, na které se nachází. Pro představu se bude jednat o podobnou adresu z předchozí kapitoly:
    - `http://localhost:1253/DVDMovie/DVDMovie.asmx`

3. Po kliknutí na *Add Reference* dojde k úpravě struktury projektu a vytvoření odkazů na metody webové služby tak, že je možno je nyní volat, jako by se jejich definice nacházela přímo v klientském zdrojovém textu. Všimněte si, že v *Solution Exploreru* ve struktuře vznikla složka *App_WebReferences* s definicí webové služby. O všechno ostatní je již postará vývojové prostředí.
4. Proveďte úpravu klientského kódu aplikace. Např. výpis data, od kterého je titul zapůjčen (tedy zavolání metody getMovieBorrowFrom()) je následující:
    - `Console.WriteLine(myService.getMovieBorrowFrom(2));`
5. Pro umístění webové výsledné webové stránky na server zvolte položku hlavního menu *Website > Copy Web Site*. V hodní části okna zvolte tlačítko *Connect* a v zobrazeném okně připojení zvolte z možností Local IIS. Poté nakopírujte strukturu aplikace na server IIS. Tím je webová stránka klienta umístěna na server.

8.3 Tvorba klienta jako desktop aplikace v C#

Protože webové služby umožňují volání z nejrůznějších druhů klientů, vyzkoušel jsem také tvorbu jednoduchého klienta, který měl charakter desktopové aplikace. Tento klient je jen pokusil zavolat každou z metod a vrátit výsledky. Kontrolou obsahu databázové tabulky jsem ověřil, že se klientská aplikace skutečně s webovou službou spojila.

Postup tvorby klientské desktopové aplikace má vývojové prostředí Visual Studio zvládnuto na jedničku. Postupuje se totiž velmi podobně, jako v předchozím příkladu s tím rozdílem, že jako vývojové prostředí použijeme Visual C# .NET 2005 Express.

1. V hlavní nabídce zvolte *File > New Project*. Nyní je možnost zvolit druh aplikace, která se stane konzumentem webové služby. Máme na výběr konzolovou aplikaci (jednodušší případ) a klasickou Windows aplikaci. Po volbě specifikujte název aplikace a stiskněte *OK*.
2. V tomto kroku již postupujte stejně, jako v předchozí kapitole. Tedy volba *Project > Add Web Reference*. Po dokončení vytváření odkazu na webovou službu je již aplikace připravena ze svých zdrojových kódů na volání metod webové služby.

Vlastním vývojem a návrhem aplikace se tento postup již zabývat nebude, jelikož s webovými službami nesouvisí.

9 IMPLEMENTACE WEBOVÉ SLUŽBY V J2EE

Implementace webové služby v J2EE proběhla dvakrát, protože zde došlo k porovnání implementace čistého řešení od firmy SUN (Web Service Developer Pack, Sun App Server a vývojové prostředí NetBeans) a proti tomu kombinace nástrojů od firem Apache a IBM (Axis, Tomcat a Eclipse).

Nejprve jsem v jazyku Java sestavil implementační kód webové služby na základě specifikace z kapitoly 7.1.3. Tento kód je uveden v příloze PIII.

Uvedené metody jsou ve zdrojovém kódu pojmenovány svými anglickými ekvivalenty. Implementace dále obsahuje v tělech jednotlivých metod použití rozhraní JDBC pro připojení k databázi. Toto rozhraní používá nadefinovanou konstantu:

```
private static final String CONNSTRING =
"jdbc:mysql://localhost/video?user=root&password=tequila";
```

Obrázek 15: Definice řetězcové konstanty pro připojení k databázi přes JDBC

Jak je patrné, jedná se o sestavovací řetězec, specifikující všechny parametry, které webová služba potřebuje pro připojení k databázovému serveru MySQL.

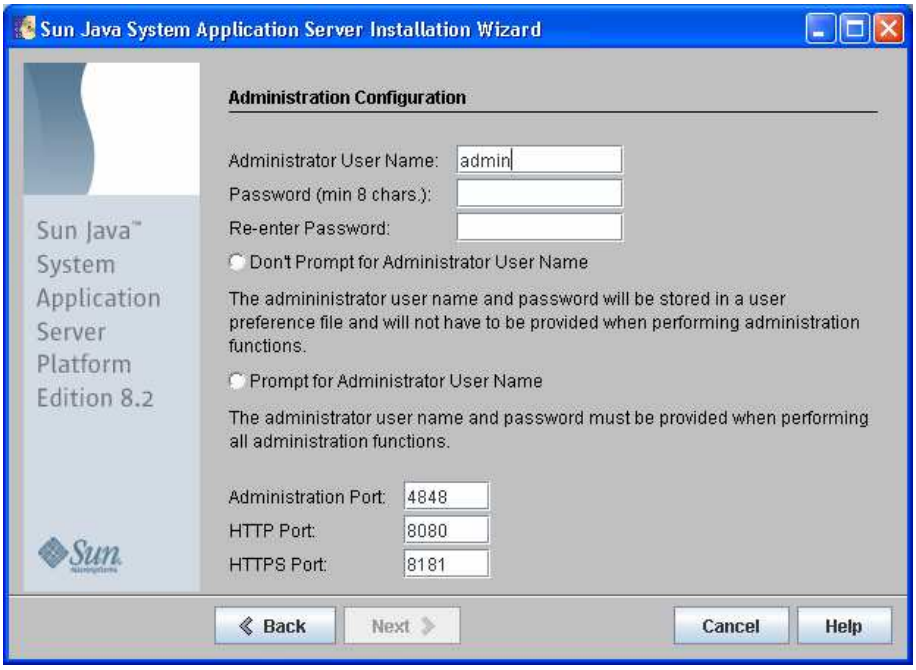
Pro vytvořený projekt webové služby je také důležité, aby v externích knihovnách byl zahrnut JAR archív Connector/J, který poskytne rozhraní JDBC důležitou mezivrstvu pro spojení s databází. Popis přidání této knihovny je uveden v následujícím textu vždy u postupu pro konkrétní vývojové prostředí.

9.1 Řešení WSDP + Sun Java Application Server + NetBeans IDE

Vytvoření webové služby pomocí nástrojů firmy SUN se skládá z několika kroků. Především neprve musí dojít k instalaci všeho potřebného vybavení a následnému nastavení prostředků. Vlastní postup tvorby webové služby potom spočívá ve vytvoření projektu webové aplikace, vytvoření samotné webové služby v rámci tohoto projektu a umístění webové služby na server a vývoj klienta, komunikujícího s webovou službou.

9.1.1 Instalace a nastavení

Instalace Sun Application Web Serveru je jednoduchá a má podobu většiny instalátorů v systému Windows. Instalační wizard si vyžádá pouze několik údajů jako jméno a heslo k administrátorského účtu a nastavení portů, na kterých server poběží.



Obrázek 16: Základní nastavení Sun Application Serveru

Vlastní administrace serveru může probíhat buď klasickým způsobem prostřednictvím konzole nebo pomocí pohodlného a přehledného webového rozhraní.

Instalace vývojového balíku pro programování webových služeb Java WSDP probíhá rovněž automaticky. Během instalace si průvodce vyžádá specifikaci umístění Sun Application Web Server. Pokud ještě nebyl nainstalován, musí dojít nejprve k jeho instalaci, aby bylo možné dokončit průvodce instalací Java WSDP.

Posledním krokem je instalace vývojového prostředí NetBeans, které je možné stáhnout z webových stránek společnosti SUN také v balíčku společně s Java WSDP. Po nainstalování vývojového prostředí je nutné ještě jeho nastavení.

9.1.2 Nastavení spojení NetBeans se Sun Java App Serverem

Tento postup je možné přeskočit v případě, že došlo k nainstalování NetBeans společně se serverem. Toto spojení je potom nastaveno automaticky.

1. Z hlavní nabídky zvolte *Tools* > *Server Manager*
2. V zobrazeném okně zvolte tlačítko *Add Server* a následně zadejte v průvodci jméno instance serveru, kterou vytváříte.
3. Nastavte cestu, kde je Sun App Server nainstalován.

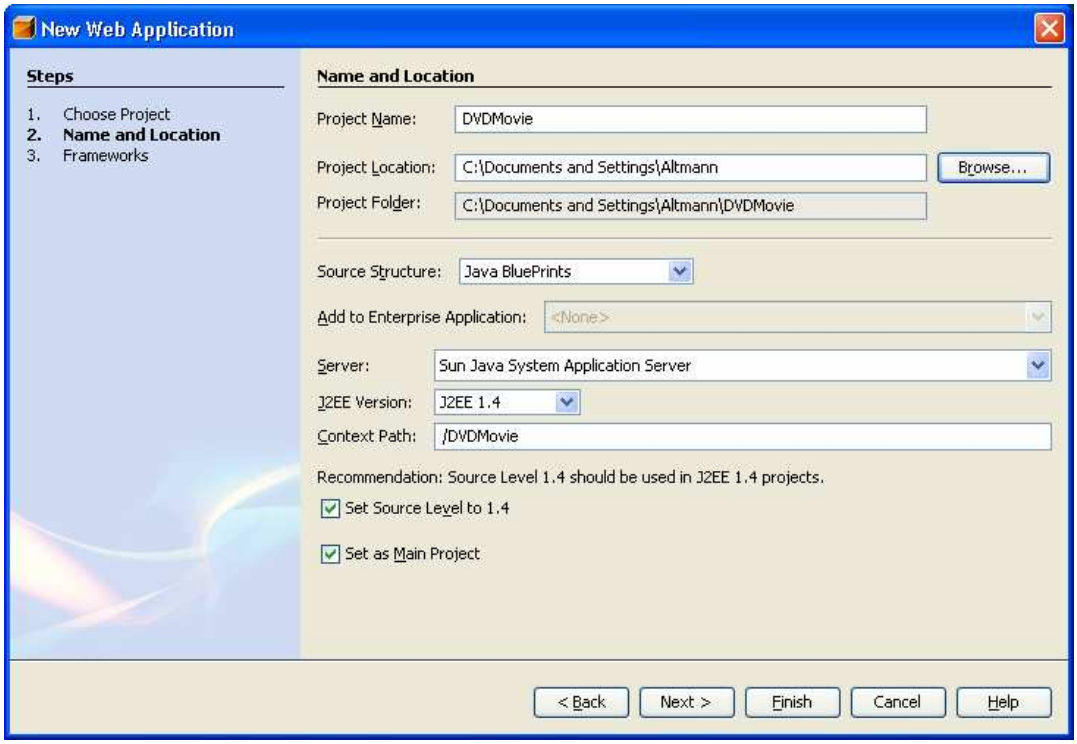
4. Zadejte uživatelské jméno a heslo, které po vás bylo požadováno při instalaci Sun App Serveru, jak bylo zobrazeno na Obrázek 16.
5. Klikněte na část okna v hlavní obrazovce – *Runtime*. Rozbalte vlákno *Servers*. Zde by se měla objevit právě založená instance serveru. Pravým tlačítkem klikněte na ni a zvolte *Start Server*.
6. V případě, že se server nepodařilo spustit, zkontrolujte, jestli jste spustili server jako službu Windows z vaší nabídky start.

V případě, že byl tento postup úspěšný, dojde ke spuštění serveru, což je označeno změnou ikony u instance v okně Runtime. Server také vypíše do konzole běhové informace.

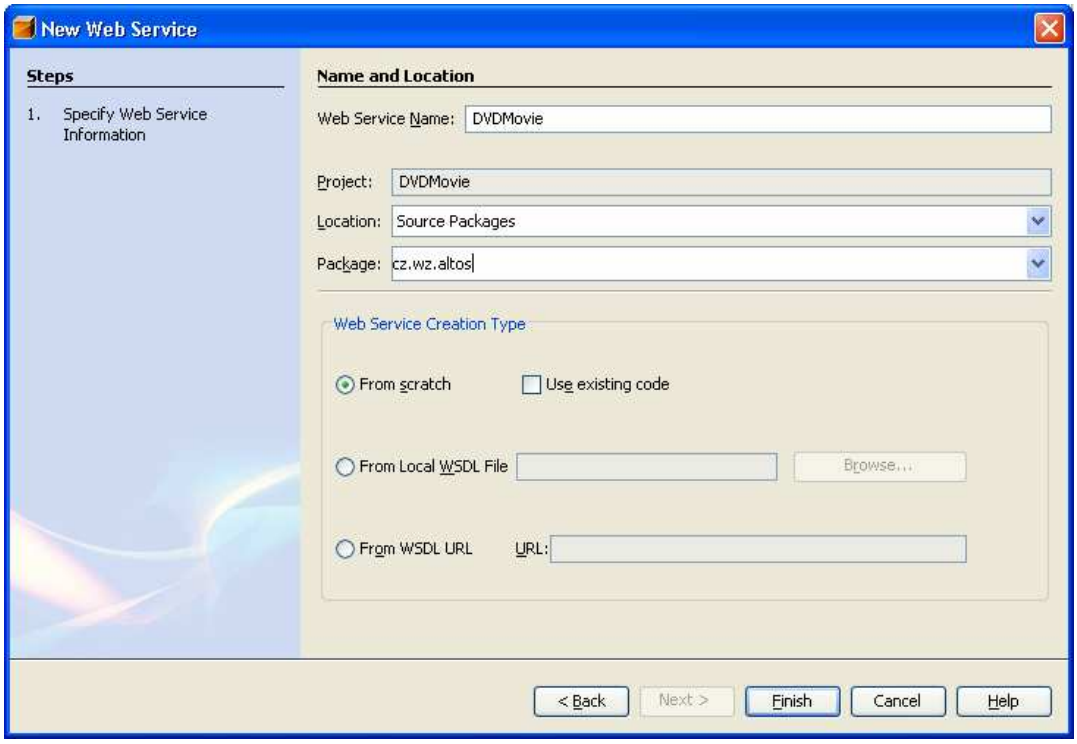
9.1.3 Postup tvorby webové služby v prostředí NetBeans

Vytvořením webové služby je myšleno především vytvoření WSDL popisu ze struktury, kterou již máme naprogramovanou. Dále potom umístění této služby na server a její otestování. Celý postup je převzatý a upravený z článků [16] a [17] a doplněn o vlastní zkušeností při vývoji webové služby.

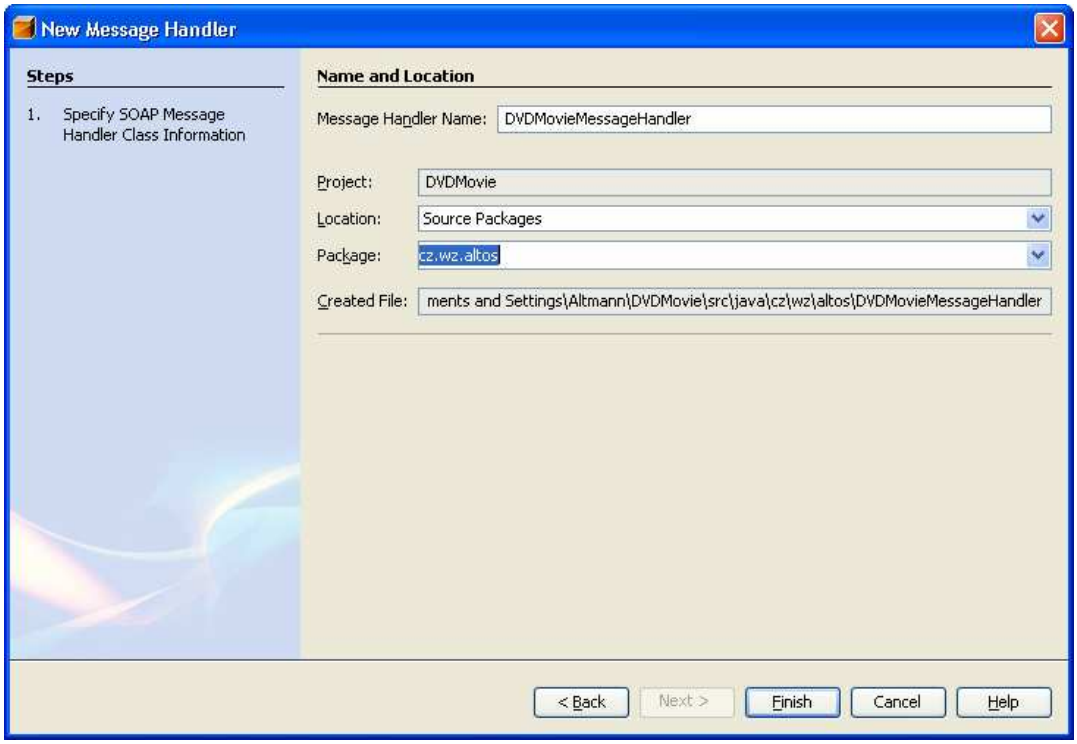
1. Vytvořte nový projekt (*File > New Project*) a v kategoriích zvolte *Web* a *Web Application*. Po kliknutí na tlačítko další pojmenujte projekt a zkontrolujte umístění na disku (*Project Folder*) a nastavení běhového serveru na Sun App Server. Nastavení parametrů je uvedeno na Obrázek 17: Nastavení nového projektu v prostředí NetBeans
2. Klikněte pravým tlačítkem na projekt vytvořený v předchozím bodě a zvolte z nabídky *New > Web Service*. Nastavte parametry webové služby jak je uvedeno na Obrázek 18: Nastavení nové webové služby v prostředí NetBeans. Po kliknutí na tlačítko *Finish* dojde k negenerování kostry webové služby.
3. Vygenerovaný zdrojový soubor `DVDmovieImpl.java` nahraďte zdrojovým kódem napsaným již dříve. Tento kód je uveden v příloze PIII.
4. Klikněte pravým tlačítkem na projekt vytvořený v prvním bodě a zvolte z nabídky *New > File/Folder*. V kategoriích zvolte *Web Services* a ve *File Types* vyberte možnost *Message Handler*. V okně specifikace nastavte parametry podle Obrázek 19: Nastavení nového rozhraní pro zasílání zpráv v prostředí NetBeans.



Obrázek 17: Nastavení nového projektu v prostředí NetBeans



Obrázek 18: Nastavení nové webové služby v prostředí NetBeans



Obrázek 19: Nastavení nového rozhraní pro zasílání zpráv v prostředí NetBeans

5. V okně přehledu projektů rozbalte vlákno *Web Services* a klikněte na webovou službu vytvořenou v bodě 2. Zvolte nabídku *Configure Handlers*. V okně *Configure SOAP Message Handlers* klikněte na *Add* a zvolte třídu *DVDMovieMessageHandler*.

6. Zkontrolujte v konfiguračních souborech projektu soubor `webservices.xml`, jestli je v něm handler deklarován.

7. Klikněte pravým tlačítkem ve výpise projektu na *Libraries* a přidejte knihovnu. Zvolte jako druh knihovny soubor *.jar a zvolte umístění knihovny Connector/J pro spojení s databází. Tato knihovna je jeden soubor ve formátu JAR obsažený v archivu staženém z webových stránek produktu MySQL.

8. Pravým tlačítkem klikněte v projektovém okně na projekt *DVDMovie*. Zvolte možnost *Run Project*. Bude odstartován server a webová služba bude přeložena a umístěna do adresářové struktury instalace serveru.

9. Pravým tlačítkem klikněte na webovou službu v projektovém okně a zvolte *Add to Registry* pro zaregistrování služby na serveru.

10. V běhovém okně rozbalte instanci serveru Sun App Server a potom rozbalte vlákno *Web Service*. Rozbalením dojde k zobrazení jednotlivých operací, které jsou

v rámci webové služby implementovány. Klikněte pravým tlačítkem na jednu z operací a zvolte *Test Operation*. Nyní po zadání vstupních hodnot můžete testovat konkrétní operaci, zda vrací korektní výsledek.

9.1.4 Postup tvorby klienta webové služby v NetBeans

Podobně jako je v NetBeans jednoduché vytvoření webové služby, stejně jednoduše se postupuje při vytváření klienta webové služby. NetBeans umožňuje vytvoření klienta jak jako webovou aplikaci (servlet), tak také jako desktop aplikaci. Následující postup ukazuje vytvoření webové aplikace:

1. Z hlavní nabídky zvolte *New Project*, v kategoriích zvolte *Web* > *Web Application*. Po kliknutí na tlačítko *Next* napište název nového projektu *DVDMovieClient*. Zkontrolujte složku, ve které bude nový projekt uložen na disku a také jestli je jako server nastaven *Sun Java App Server*.
2. Pravým tlačítkem klikněte v okně projektů na nově vzniklý projekt a zvolte *New* > *Web Service Client*. Vložte také URL na kterém momentálně webová služba *DVDMovie* běží. Pokud je vše nastaveno standardně měla by to být adresa:
   - `http://localhost:8080/DVDMovie/DVDMovie?WSDL`
3. Klikněte na *Retrieve WSDL*, čímž dojde k přečtení WSDL popisu služby, který jsme vytvořili v předcházející kapitole. Zadejte dále balíček (v mém případě cz.wz.altos) a zkontrolujte, že je nastaveno *J2EE Container – generated static stub*.
4. Pravým tlačítkem klikněte na projekt a zvolte *New* > *Servlet*. Pojmenujte nový servlet jako *DVDMovieServlet* a jako balíček nastavte stejnou hodnotu, jako v bodě 3. Nyní je negenerován zdrojový kód klientského servletu. Když kliknete někam do zdrojového kódu třídy *DVDMovieServlet* a zvolíte *Web Service Klient Resources* > *Call Web Service Operation*, vložíte syntakticky správné vzdálené volání jedné z metod služby. Její kód již můžete upravit podle libosti, stejně jako kód okolní pro tvorbu uživatelského interface.

9.1.5 Tvorba klienta jako desktop aplikace v jazyku Java

Také v jazyce Java jsem vytvořil jednoduchého desktopového klienta webové služby. Přestože toto řešení nemělo za úkol změřit odezvu, uvádím jej zde z důvodu uvedení úplných možností práce s webovými službami.

1. V hlavní nabídce prostředí NetBeans zvolte *File > New Project*. V jednotlivých kategoriích vyberte možnost *General > Java Application*, klikněte na *Next*.
2. Zadejte jméno nové aplikace (např. *DVDMovieClientApp*) a specifikujte cesty k projektu a jeho umístění. Zkontrolujte, zda je označeno vytvoření hlavní třídy a klepněte na *Finish*.
3. Nyní je zapotřebí vytvořit u nově vzniklého projektu vazbu na webovou službu. Klikněte pravým tlačítkem na hlavní uzel projektu a zvolte *New > Web Service Client*. Pokud webová služba právě běží na Sun Java App Serveru, zadejte URL k WSDL souboru. Je to stejná adresa jako v předchozím příkladu:
   - `http://localhost:8080/DVDMovie/DVDMovie?WSDL`
4. Následující postup je totožný: Klikněte na *Retrieve WSDL*, čímž dojde k přečtení WSDL popisu služby, který jsme vytvořili v předcházející kapitole. Zadejte dále balíček (v mém případě cz.wz.altos) a zkontrolujte, že je nastaveno *J2EE Container – generated static stub*.
5. Pokud kliknete do prostoru hlavní smyčky *Main* pravým tlačítkem, zvolte *Web Services Klient Resources > Call Web Service Operation* a zvolte metodu, kterou chcete zavolat. Dojde k negenerování kódu volání metody ve správné syntaxi včetně ošetření všech výjimek.
6. Tímto způsobem můžete postupovat dále při tvorbě zdrojového kódu klientské aplikace.

Vlastním vývojem a návrhem aplikace se tento postup již zabývat nebude, jelikož s webovými službami nesouvisí.

9.2 Řešení Axis + Tomcat + Eclipse

Podobně jako v případě kapitoly 9.1 se řešení za pomoci prostředků firem Apache Foundation a IBM skládá z několika kroků. Celý proces je velmi podobný, přesto existují drobné rozdíly při tvorbě, které budou zhodnoceny v následujících kapitolách.

9.2.1 Instalace a nastavení Axis a Tomcat

Open-source řešení Tomcat+Axis, je v dnešní době velmi oblíbené. Tato kapitola představuje postup instalace tohoto řešení, které se zpočátku uživateli Windows nemusí zdát triviální, protože není vedeno žádným instalátorem. Celý postup je rozepsán do několika kroků.

Nutné komponenty pro instalaci:

- JDK 1.4 nebo vyšší (http://java.sun.com)
- Apache Tomcat (http://tomcat.apache.org)
- Apache Axis (http://ws.apache.org/axis)
- JavaBeans Activation Framework – pomocí tohoto rozhraní může Java aplikace pracovat s MIME daty. (http://java.sun.com/products/javabeans/glasgow/jaf.html)
- JavaMail API – rozhraní sloužící pro práci s emaily. Jedná se o množinu tříd umožňující pohodlnou implementaci mailových zpráv v java aplikacích. Je součástí J2EE, ovšem pro J2SE se musí doinstalovat (http://java.sun.com/products/javamail).
- Apache XML security (http://xml.apache.org/security/dist/java-library).

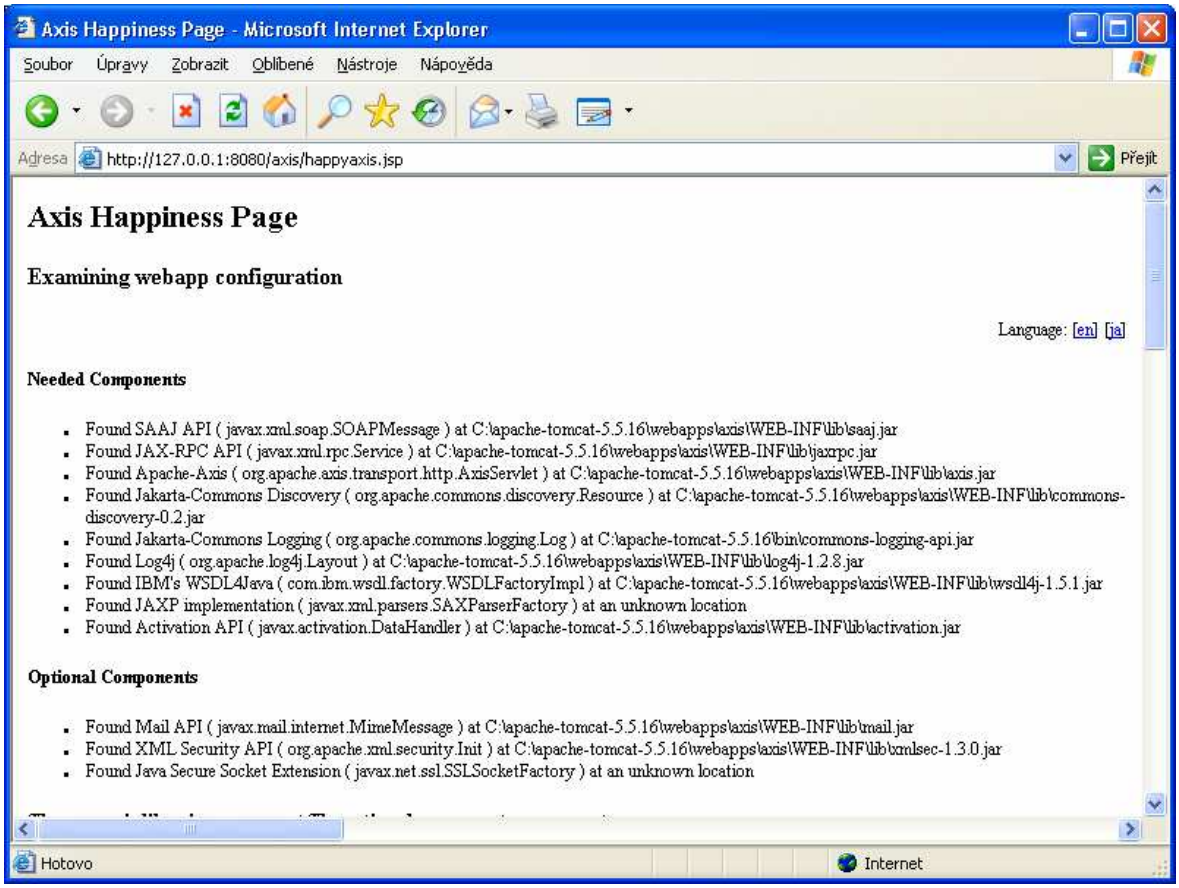
Všechny tyto části jsou volně stažitelné z internetu. Uvedené zdroje jsou rovněž přístupné na CD, přiloženém k tomuto textu.

Postup instalace krok za krokem:

1. Pokud tak nebylo učiněno dříve, nainstalujte Javu do počítače. Tento krok je uveden v samostatném postupu instalace.
2. Archiv zip s Apache Tomcat rozbalte na disk do adresáře dle vlastního uvážení. Instalátor zde není třeba a server je tímto připraven ke spuštění.

3. V archivu Axisu najděte adresář `..\webapps\axis` a celý jeho obsah zkopírujte do adresáře Tomcatu `..\webapps`. Tímto krokem jste nainstalovali Axis na server Tomcat.
4. Z archivu JavaBeans Activation Framework rozbalte soubor `activation.jar` do adresáře Tomcatu `..\webapps\axis\WEB-INF\lib`.
5. Z archivu JavaMail rozbalte soubor mail.jar do stejného adresáře, jako v předchozím případě, tedy `..\webapps\axis\WEB-INF\lib`.
6. Z archivu Apache XML security rozbalte všechny soubory umístěné v `..\libs` do `..\webapps\axis\WEB-INF\lib`.

Pokud byla instalace provedena správně, proběhne také v pořádku validace instalace. Spusťte tedy server Tomcat (viz obr) a ve webovém browseru zadejte adresu `http://127.0.0.1:8080/axis/`, čímž se spustí úvodní stránka SOAP interpretu Axis. Pokud se tak nestane, zkontrolujte jestli běží server Tomcat, případně jestli jste správně umístili Axis do adresářové struktury Tomcatu. Pokud se úvodní stránka načetla v pořádku, pokračujte kliknutím na odkaz Validation, čímž dojde ke zkontrolování instalace (viz Obrázek 20). Pro bezproblémový chod je především důležité, aby všechny komponenty Axisu v sekci Needed Components byly ve stavu FOUND. Tento postup také zahrnuje instalaci nepovinných komponent, jejichž výčet je v sekci Optional Components. V případě, že některá z komponent nebyla nalezena, bude to touto stránkou zobrazeno společně s odkazem na stránku, kde lze tuto komponentu najít. V tomto případě znovu překontrolujte správnost celé instalace v bodech 4-6.



Obrázek 20: Validace instalace Axisu

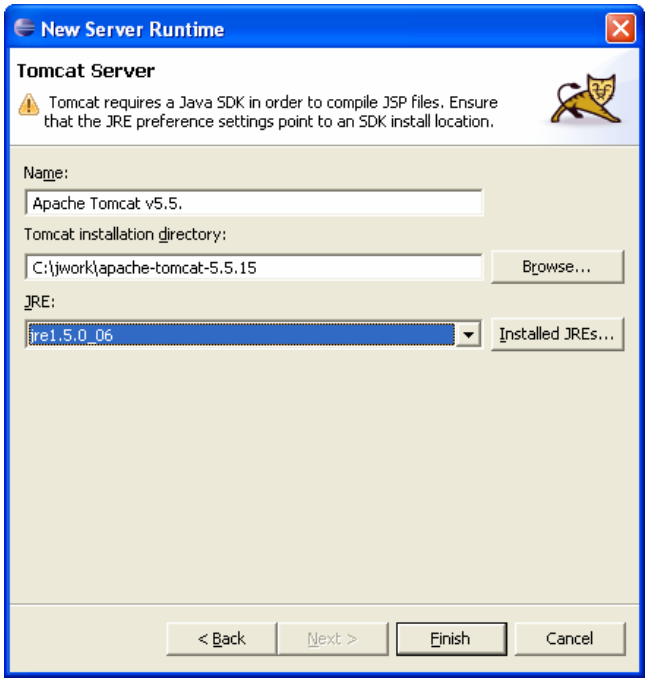
9.2.2 Nastavení spojení Eclipse s Tomcat serverem

Po správném nainstalování serveru Tomcat včetně všech jeho součástí je potřeba pro pohodlné vytváření webové služby v Eclipse podobně jako v NetBeans zajistit spojení vývojového prostředí se serverem. Je potřeba postupovat podle následujících kroků:

1. Z hlavní nabídky zvolte položku *File* > *New* > *Other* > *Server*. V zobrazeném okně nechejte jako jméno hositele serveru *localhost* a specifikujte typ serveru jako Tomcat v5.5, pokud jste tedy nenainstalovali jinou verzi serveru a klikněte na tlačítko *Installed Runtimes*.

2. V tomto kroku jsme se přenesli do hlavního nastavení prostředí Eclipse, kde můžeze specifikovat jednotlivé běhové prostředí různých serverů. Kliknutím na tlačítko *Add* přidáme nové běhové prostředí serveru. Znovu specifikujte typ serveru a po kliknutí na tlačítko *Next* určete jméno serveru (můžete ponechat stávající) a především specifikujte instalační adresář serveru a běhové prostředí JRE, pokud jich má-

te na svém počítači nainstalováno více. Nastavení tohoto okna je uvedeno na Obrázek 21: Specifikace Runtime pro server.



Obrázek 21: Specifikace Runtime pro server

3. Po kliknutí na *Finish* můžete zobrazit novou instanci serveru v pracovním prostředí Eclipse volbou *Windows* > *Show view* > *Other* > *Server* > *Servers*

9.2.3 Postup tvorby webové služby včetně klienta v prostředí Eclipse

Nyní se již dostáváme k vlastnímu popisu tvorby webové služby. Tento postup je opět velmi podobný jako v případě prostředí NetBeans. Detaily jsou popsány v [14] a [15] a dokumentaci Eclipse WTP.

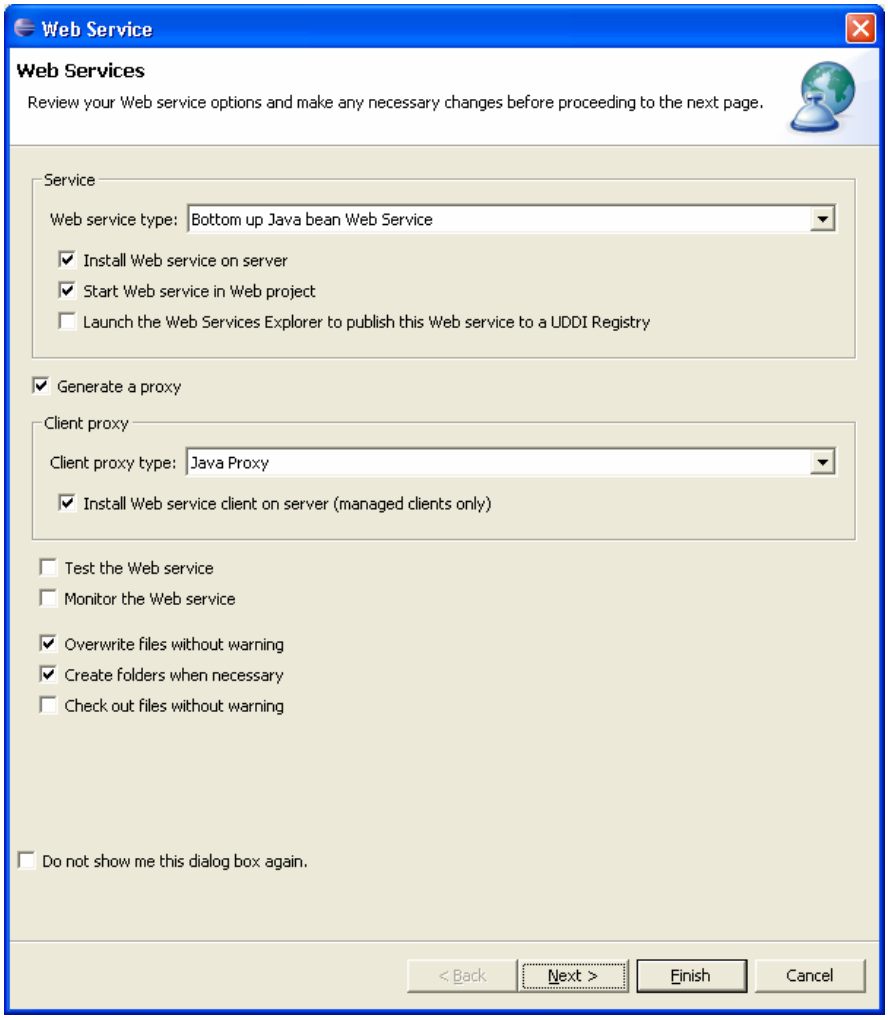
1. Nejprve vytvořte projekt volbou z hlavní nabídky *New* > *Web* > *Dynamic Web Project*. Poté vložte všechny potřebné údaje (jméno projektu, specifikujte verzi servletu na 2.3, vyberte v předchozím kroku nainstalovanou instanci spojení se serverem Tomcat) Poté klepněte na *Finish* a dojde k vytvoření nového projektu, který má vazbu na server Tomcat.

2. Do zdrojové složky projektu (nejčastěji `/src`), který jste vytvořili v předchozím bodu importujte zdrojový kód třídy, která je uvedena v příloze PIII.

3. Pravým tlačítkem klikněte na soubor se zdrojovým kódem webové služby a zvolte *Web services* > *Create web service*. Dojde ke spuštění průvodce tvorbou webové

služby. V prvním kroku zadejte údaje podle Obrázek 22: Vytvoření nové webové služby v Eclipse

4. V následujícím kroku specifikujte třídu, ze které se bude webová služba generovat. Klikněte na *Next*.
5. V kroku specifikace serverů, na kterých má služba a klient běžet ponechte vše v nastavení, jaké vám nabídl průvodce. Příslušná nastavení změňte jen v případě, že máte nainstalováno více serverů a potřebuje specifikovat, na kterém se má služba objevit.
6. Také v kroku nabídky výběru metod, které se mají zahrnout do procesu generování služby ponechte vše beze změn. V horní části je automaticky specifikováno umístění webové služby na serveru a přístupová adresa a stejně tak adresa pro přístup k WSDL popisu služby.
7. V případě, že váš Tomcat server ještě neběží, nabídne vám průvodce krok pro spuštění. Klikněte na tlačítko *Start server* a vyčkejte spuštění serveru. Zároveň dojde k umístění webové služby (jak její implementace tak také jejího popisu WSDL) na server.
8. V tomto kroku můžete nechat nagenerovat tzv. Proxy třídy do implementace klienta pro automatické volání metod. Tyto třídy jsou programovým pozadím pro zajištění spojení se serverem na principu vzdáleného volání metod.
9. Zároveň na rozdíl od ostatních řešení je možné v dalších bodech vytvořit také klienta, který připravený *Dynamic Web Project* s nadefinovaným rozhraním pro volání metod webové služby. Přidáním třídy kódu aplikace klienta služby do stejného balíčku je vytvoření klienta hotovo.
10. Vlastní kód klienta je umístěn v souborech JSP v projektové složce WebContent. Pokud zde soubory nejsou, klikněte na popis služby ve WSDL souboru pravým tlačítkem a zvolte Web Services > Generace Sample JSP, aby došlo k vytvoření zdrojových souborů. Tyto soubory jsou pro úplnost také na přiloženém CD.
11. Vlastní úprava souborů JSP už je individuální. V těchto souborech je naznačeno, jak volat jednotlivé metody webové služby.



Obrázek 22: Vytvoření nové webové služby v Eclipse

10 IMPLEMENTACE WEBOVÉ SLUŽBY V PHP

Implementace webové služby v PHP si proti svým dvěma předchůdcům nesla s sebou jistá specifika. Především pak to, že jsem pro knihovnu nuSOAP nenašel patřičný vývojový nástroj a vývoj webové služby tak musel proběhnout ručně. Stejně tak se stal poměrně pracnou záležitostí vývoj klienta služby, ze stejného důvodu.

Při studiu práce s knihovnou nuSOAP mi byly kromě projektové dokumentace knihovny nejvíce nápomocné stránky s konkrétními a okomentovanými příklady [22] a [23].

10.1 Postup tvorby webové služby v PHP

Vlastní proces implementace webové služby se však nelišil od předchozích postupů. Nejprve jsem vytvořil v PHP jazyku zdrojový text implementace služby s obsluhou všech nadefinovaných metod včetně připojení k MySQL databázi. Tento zdrojový kód je přiložen v příloze PIV.

Protože PHP na rozdíl od např. Javy není přísně objektovým jazykem, knihovně nuSOAP stačí nadefinovat příslušné funkce v hlavním těle skriptu a pomocí standardních metod knihovny nuSOAP, jak je uvedeno na Obrázek 23: Vytvoření instance SOAP serveru a zaregistrování metody.

```
require_once('nusoap.php');

$server = new soap_server;
$server->register('addMovie');
...
```

Obrázek 23: Vytvoření instance SOAP serveru a zaregistrování metody

Klient webové služby je také skriptem, který běží na stejném serveru. Po zaregistrování příslušných metod se tyto z klientského skriptu volají způsob, jaký ukazuje názorně Obrázek 24: Volání jedné z metod webové služby.

```
$result = $client->call('addMovie', array('name' => $name,
                        'locname' => $locname,
                        'year' => $year,
                        'director' => $director));
```

Obrázek 24: Volání jedné z metod webové služby

Do proměnné `result` se vždy uloží hodnoty, které metoda ve své implementaci vrací. Je potom na klientské aplikaci, jak bude s těmito hodnotami nakládat. Metoda z uvedeného příkladu žádný výsledek nevrací.

Protože je PHP skriptovacím jazykem, který dokáže běžet jen v rámci webových stránek, musel být zdrojový kód klienta „roztříštěn“ do několika samostatných souborů. Jejich skladba je velmi podobná a ve většině případů se jedná jen o malé změny. Zdrojový kód klienta služby zde v tomto textu proto záměrně neuvádím, je však přiložen na doprovodném CD disku.

11 SROVNÁNÍ JEDNOTLIVÝCH IMPLEMENTACÍ

Poslední kapitola této práce se zabývá srovnáním jednotlivých implementací webových služeb a to ze dvou hledisek.

Srovnání implementační náročnosti vychází částečně ze zhodnocení rychlosti a snadnosti vývoje pomocí nejrůznějších vývojových nástrojů vhodných k té které platformě, částečně pak ze subjektivních pocitů a problémů autora při tvorbě webové služby DVD film.

Srovnání výkonu je druhým kritériem porovnání jednotlivých implementací. Zde jsem se snažil o co nejpřesnější změření výkonu jednotlivých webových služeb při zajištění pokud možno rovných podmínek.

11.1 Srovnání implementační náročnosti

Při studiu webových služeb – tedy co to je a k čemu to vlastně slouží, jsem došel k závěru, že jednou z nejdůležitějších vlastností pro vývojáře je jejich snadnost tvorby. Tím mám na mysli především přístup, ve kterém bude muset programátor napsat co nejméně obslužného kódu, aby dosáhl výsledku. Webové služby a nástroje pro jejich tvorbu toto prakticky umožňují. Vývojář se tedy může soustředit jen na naprogramování vlastní implementace webové služby, která se stane jakousi černou skříňkou pro všechny ostatní. Vytvořením popisu této služby ve WSDL potom všem „zájemcům“ standardním způsobem řekne, jak s webovou službou zacházet – tedy jaké parametry a v jakém tvaru předávat a naopak, co služba všechno za výsledky vrací.

Při programování webové služby DVD film jsem došel k závěru, že na platformě .NET je toto vyřešeno velmi dobře a prakticky se jedná o velmi dobrý způsob, jak začít s webovými službami pracovat. Dá se říct, že vývojové prostředí Visual Studio .NET Express má pro začínající s webovými službami hodně co nabídnout. Ovšem na druhou stranu je také spousta věcí kolem vývoje webových služeb skryta „pod povrch“. Mám tím na mysli především ukrytí kódu vzdáleného volání metod do projektu. Žádná z běžných nabídek okna Solution Explorer nenabízí možnost prohlédnout si tento kód. Je ale možné, že toto je úprava pouze Express verze Visual Studia .NET, protože tato edice je přece jen prezentována samotným Microsoftem jako začátečnická a sloužící spíše k seznámení, přestože se jedná o plnohodnotný vývojový nástroj.

Pro vlastní naprogramování webové služby je jeví varianta .NET jako nejlepší, protože umožňuje také velmi pohodlné ladění. Vývojové prostředí Visual Web Developer 2005 Express má dokonce svůj vlastní integrovaný web server. Veškeré operace typu umístění webové služby na server apod. jsou řešeny automaticky.

Podobně je na tom i platforma Java, kde především řešení o firmy SUN, je velmi dobře zvládnuto. Oproti řešení od Microsoft, ale NetBeans dávají větší možnost nahlédnout „pod pokličku" vlastní práce, při zachování pohodlného vývoje. Opět jde především o otevřenost projektu a zdrojových textů volání metod, které se generují samy, jak byla zmíněna o několik řádků výše. Také NetBeans nabízejí integrovaný web server a to přímo Tomcat od Apache[12]. Ten jsem při práci nepoužil, protože jsem se soustředil čistě na řešení od SUNu, tedy Sun Java Application Web Server.

Při použití varianty Tomcat + Axis + Eclipse, jsem se ale setkal s problémy, které plynuly především z prostředí Eclipse jako takového. Implementace práce s webovými službami se zde potýká ještě s menšími problémy, protože se občas stávalo, že editory a průvodci webových služeb negenerovali funkční kód a v některých případech byly dokonce nestabilní. Průzkumem internetových fór jsem také zjistil, že se problém zdaleka netýká jen mé osoby. Ovšem celkově musím tyto nedostatky hodnotit jen jako malé.

Trochu stranou ovšem stojí řešení s PHP. Přístup k tvorbě webové služby v PHP je totiž úplně jiný, než v případě .NET nebo Javy. V některých případech je implementace jednodušších služeb velmi snadná na pochopení. Protože jsem během práce na tomto textu nenašel žádný nástroj, který by implementaci jako v předcházejících případech usnadňoval, musím konstatovat, že použití je sice jednoduché, ale časově náročné, protože si žádá vlastnoruční tvorbu obslužného kódu především v klientské části.

Jen programová tvorba popisu webové služby ve WSDL je na zvláštní kapitolu. Jedná se o relativně složitý proces použití několika metod knihovny nuSOAP. Vytvoření WSDL popisu, je však v řadě případů nezbytné pro použití služby třetími stranami. WSDL popis není nutné tvořit v případě, že na stejném serveru jako služba běží také klientské skripty a nechceme službu použít pro další potenciální zájemce ze stran vývojářů. Ovšem nutno po-

[12] Bohužel však jsem jeho testováním zjistil, že neobsahuje standardně interpret SOAP zpráv Axis.

dotknout, že tvorbu WSDL popisu zvládnou i jiné externí nástoje. Mezi tyto nástroje můžeme započítat jednak všechna vývojová prostředí konkurenčních platforem a dále také ostatní editory (např. WSDL Generator).

S přihlédnutím na fakt, že knihovna nuSOAP byla vytvořena jedním člověkem, kdežto řešení firem SUN, Apache, Microsoft a IBM zaměstnalo bezesporu několik desítek lidí, nemohlo srovnání ani jinak dopadnout a je na zvážení, jestli vůbec mělo nějaký smysl.

Implementace PHP tedy byla zařazena do této práce jenom pro rozšíření představy o možnostech webových služeb.

11.2 Srovnání výkonu webových služeb

Vlastní srovnání výkonu webových služeb probíhalo na principu měření rychlosti odezvy webové služby po zavolání jejích metod.

Měření probíhalo na úrovni klientů, kdy byl vždy upraven kód klienta tak, aby před zavoláním příslušné metody vytvořil první časové razítko a hned po skončení metody s přijetí výsledků bez dalšího zpracování vytvořil druhé časové razítko. Porovnáním těchto časových razítek vznikl čas, který klient webové služby potřeboval k:

1. Připojení na server
2. Vytvoření SOAP požadavku
3. Zpracování předávaných parametrů v rámci implementačního kódu v daném jazyce
4. Vytvoření SOAP odpovědi
5. Přijetí výsledku ze zpracované zprávy

Přestože servery byly umístěny na localhostu, měření každé metody proběhlo desetkrát a po odstranění nejrychlejších a nejpomalejšího výsledku se spočítal průměr. Tato metoda byla zvolena z důvodu nenadálých systémových swapů, které by mohly výsledky podstatným způsobem ovlivnit.

V následujícím textu jsou tabulky měření výkonu jednotlivých implementací webových služeb. K těmto tabulkám je následující označení:

- A – addMovie(), B – borrowMovie(), C – returnMovie(), D – getMovieBorrow(), E – getMovieBorrowFrom(), F – getMovieInfo().

V tomto pořadí jsou označeny metody ve všech tabulkách. V posledním řádku každé z tabulek je jako p označen průměr hodnot ve sloupci. Poslední řádek každé tabulky tvoří převrácené hodnoty jednotlivých průměrů.

11.2.1 Webová služba v .NET

Pro webovou službu implementovanou v .NET jsem vytvořil jako klienta webovou aplikaci, která otestovala každou metodu desetkrát. Výsledky vypsala do HTML stránky, která byla převedena do tabulky. Aplikaci klienta jsem musel upravit tak, aby po spuštění nespouštěla okamžitě procedudu samotného testu, protože se to výrazně podepsalo na prvním zavolání metody addMovie(). Čas byl v tomto případě výrazně horší.

Při testu, jehož výsledky ukazuje tabulka, jsem nezjistil jinak žádné další výkyvy výkonu a dá se říct, že je tedy implementace v .NET stabilní. Je možné, že je toto způsobeno webovým serverem Microsoft (IIS), který webovou službu hostoval.

Tabulka 3: Výsledky měření odezvy webové služby v .NET

	A	B	C	D	E	F
1	0,101	0,020	0,050	0,040	0,070	0,050
2	0,050	0,040	0,020	0,020	0,023	0,034
3	0,060	0,033	0,040	0,050	0,040	0,057
4	0,014	0,010	0,010	0,020	0,029	0,020
5	0,070	0,040	0,071	0,040	0,050	0,040
6	0,020	0,020	0,010	0,023	0,020	0,030
7	0,030	0,050	0,030	0,050	0,040	0,093
8	0,029	0,029	0,020	0,020	0,010	0,020
9	0,130	0,040	0,050	0,080	0,046	0,040
10	0,040	0,020	0,020	0,020	0,050	0,010
p	0,050	0,030	0,030	0,033	0,037	0,036
1/p	19,969	33,022	33,285	30,392	26,870	27,415

Celkový průměr, stanovený jako průměrná hodnota průměrných dob odezvy jednotlivých metod, potom po výpočtu činil **0,036 s**.

11.2.2 Webová služba v Java – řešení Sun App Server

Pro webovou službu běžící na Sun App Serveru jsem sepsal servlet, který opět všechny metody otestovat desetkrát. Výsledek byl uložen do HTML formátu a převeden do následující tabulky.

Tabulka 4: výsledky měření odezvy webové služby v Javě (Sun)

	A	B	C	D	E	F
1	0,202	0,077	0,031	0,156	0,031	0,031
2	0,031	0,032	0,047	0,031	0,031	0,046
3	0,031	0,031	0,093	0,047	0,046	0,047
4	0,016	0,062	0,016	0,046	0,031	0,031
5	0,217	0,031	0,015	0,031	0,032	0,078
6	0,171	0,047	0,032	0,047	0,046	0,046
7	0,032	0,046	0,046	0,684	0,031	0,032
8	0,009	0,031	0,002	0,078	0,047	0,046
9	0,015	0,047	0,047	0,031	0,031	0,031
10	0,047	0,155	0,046	0,047	0,047	0,047
p	0,068	0,047	0,035	0,060	0,037	0,041
1/p	14,679	21,448	28,571	16,563	27,119	24,540

Celkový průměr, stanovený jako průměrná hodnota průměrných dob odezvy jednotlivých metod, potom po výpočtu činil **0,048 s**.

11.2.3 Webová služba v Java – řešení Apache Tomcat a Axis

Pro webovou službu běžící na komponentách od Apache, jsem napsat v prostředí Eclipse JSP servlet, jehož výsledkem se staly hodnoty časů uvedené v tabulce.

Tabulka 5: výsledky měření odezvy webové služby v Javě (Apache)

	A	B	C	D	E	F
1	0,016	0,047	0,016	0,031	0,016	0,031
2	0,264	0,015	0,031	0,031	0,031	0,016
3	0,047	0,031	0,016	0,016	0,015	0,016
4	0,015	0,016	0,031	0,031	0,031	0,279
5	0,016	0,016	0,031	0,015	0,032	0,031
6	0,046	0,015	0,015	0,016	0,015	0,032
7	0,016	0,031	0,016	0,031	0,031	0,015
8	0,046	0,031	0,031	0,016	0,016	0,031
9	0,015	0,016	0,015	0,031	0,015	0,016
10	0,031	0,015	0,016	0,015	0,031	0,031
p	0,029	0,021	0,022	0,023	0,023	0,026
1/p	34,335	46,784	46,512	42,781	43,011	39,216

Celkový průměr, stanovený jako průměrná hodnota průměrných dob odezvy jednotlivých metod, potom po výpočtu činil **0,024 s**.

11.2.4 Webová služba v PHP

Pro webovou běžící na PHP a Apache serveru jsem napsal speciální skript `benchmark.php`, který výsledky prezentoval v HTML formátu.

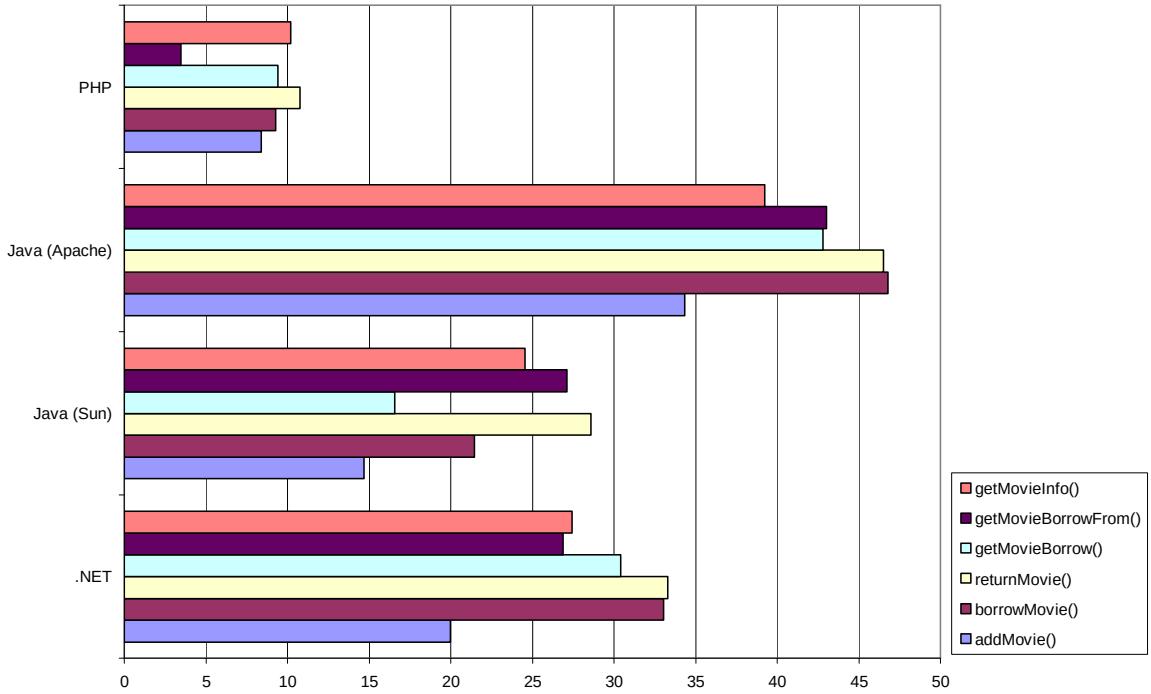
Tabulka 6: výsledky měření odezvy webové služby v PHP

	A	B	C	D	E	F
1	0,906	0,347	0,093	0,172	0,328	0,146
2	0,256	0,086	0,107	0,116	0,289	0,092
3	0,097	0,106	0,087	0,091	0,429	0,093
4	0,095	0,091	0,091	0,089	0,495	0,094
5	0,109	0,088	0,094	0,095	0,119	0,089
6	0,091	0,093	0,090	0,092	0,251	0,088
7	0,091	0,090	0,087	0,101	0,760	0,089
8	0,108	0,091	0,091	0,090	0,177	0,092
9	0,105	0,091	0,091	0,216	0,087	0,089
10	0,091	0,214	0,210	0,094	0,215	0,216
p	0,119	0,108	0,093	0,106	0,288	0,098
1/p	8,397	9,260	10,753	9,416	3,474	10,191

Celkový průměr, stanovený jako průměrná hodnota průměrných dob odezvy jednotlivých metod, potom po výpočtu činil **0,135 s**.

11.2.5 Celkové srovnání výkonu

Pro každou průměrnou vypočítanou hodnotu jsem stanovil také její převrácenou hodnotu. Tyto výsledky jsem zanesl do samostatné tabulky a pro lepší přehlednost také do grafu, který je zobrazen jako Obrázek 25: Graf srovnání výkonu jednotlivých implementací.



Obrázek 25: Graf srovnání výkonu jednotlivých implementací

Již z grafu jednoznačně vyplývá, že nejlepších hodnot odezvy se podařilo dosáhnout na serveru Apache. Následovala implementace Microsoft na serveru IIS a poté hned řešení firmy SUN. Jako poslední v testu skončila implementace v PHP, což se dalo předpokládat, jelikož se jedná o projekt, který je svojí povahou spíše „nadšenecký“.

ZÁVĚR

Webové služby jsou moderním trendem dnešních řešení nejrůznějších systémů a dá se předpokládat, že díky jednoduchosti, otevřenosti a především podpoře velkých firem se budou stále více rozšiřovat do povědomí vývojářů. Z hlediska uživatelů tak pravděpodobně dojde ke stavu, kdy časem s nárůstem rychlosti připojení k Internetu a navýšení hardwarového výkonu nebudou používání webových služeb pozorovat a funkcionalita se jim bude jevit jako lokální.

Při psaní tohoto textu jsem se pokusil shrnout a zpracovat problematiku webových služeb jak po teoretické stránce, tak také ukázat jednoduchost vlastní implementace. Navíc jsem se tímto způsobem snažil ukázat podobnost práce s webovými službami pod nejrůznějšími vývojovými prostředími a poodhalit tak jednu z jejich pro vývojáře největších výhod.

Z hlediska možností a pohodní práce – tedy náročnosti implementace, jsem došel k závěru, že je na tom nejlépe řešení od firmy Microsoft, které je určeno také pro pohodlné seznámení se s webovými službami. Platforma .NET nabízí široké možnosti práce s webovými službami, přičemž se jejich vývoj snaží co nejvíce ulehčit.

Platforma Java v tomto směru nezůstává příliš pozadu a je více než důstojným konkurentem. Přesto mám pocit, že vývoj webových služeb v Javě je spojen s většími nároky na znalost problematiky. Naproti tomu ale Java nabízí detailnější pohled do „útrob“ fungování webových služeb.

PHP je z hlediska implementace webových služeb asi nejméně přívětivé, přesto nabízí podporu všech důležitých součástí webových služeb. Přesto to s sebou nese i jisté výhody a to především v rozšíření PHP na množství freewebových serverech a tedy možnosti rozjet webovou službu s nulovými nebo velmi nízkými náklady.

Výkon je hledisko které bych při rozhodování, kterou implementaci použít, zohlednil jen v případě velkých systémů, protože jak se ukázalo, rozdíly v rychlosti jednotlivých řešení nejsou nijak závratné a pohybují se v řádu setin sekund. Přesto musím konstatovat, že překvapivě na tomto poli zvítězila platforma Java a to v kombinaci serveru Apache Tomcat se SOAP interpretem Axis. Naopak v řešení firmy Sun, které nabízelo daleko vyšší komfort při vývoji, výkonnostně Java propadla.

SEZNAM POUŽITÉ LITERATURY

[1] ECKEL, Bruce. *Myslíme v jazyku Java – knihovna programátora*. Praha: Grada Publishing, 2001. 432 s. ISBN 80-247-9010-6

[2] ECKEL, Bruce. *Myslíme v jazyku Java – knihovna zkušeného programátora*. Praha: Grada Publishing, 2001. 470 s. ISBN 80-247-0027-1

[3] KANISOVÁ, H., MÜLLER, M. *UML srozumitelně*. Brno: Computer Press. 2004. 158 s. ISBN 80-251-0231-9

[4] PRICE, Jason. *C# - programování databází*. Praha: Grada Publishing. 2005. 624 s. ISBN 80-247-0982-1

[5] SHARP, J., JAGGER, J. *MS Visual C# .NET krok za krokem*. Mobil Media, 2002. 656 s. ISBN 80-865-9327-4

[6] WELLING, L., THOMSON, L. *PHP a MySQL – rozvoj webových aplikací*. Soft-Press, 2002. 830 s. ISBN 80-864-9783-6

[7] KOSEK, Jiří. *Seriál o XML pro Softwarové noviny* [online]. 2001-2003 [cit. 2006-03-21]. Dostupný z WWW: <http://www.kosek.cz/clanky/swn-xml/index.html>.

[8] KOSEK, Jiří. *XML - eXtensible Markup Language* [online]. 2001-2003 [cit. 2006-03-21]. Dostupný z WWW: <http://www.kosek.cz/clanky/xml/index.html>.

[9] KOSEK, Jiří. *XML schémata* [online]. 2005 , 18.8.2005 [cit. 2006-03-21]. Dostupný z WWW: <http://www.kosek.cz/xml/schema/index.html>.

[10] KOZÁK, David. Jak fungují webové služby. *Interval* [online]. 2002 [cit. 2006-05-05] Dostupný z WWW: <http://interval.cz/clanky/jak-funguji-webove-sluzby/>.

[11] ŽALOUDEK, Jiří. *Java web services : ročníkový projekt* [online]. 2004 [cit. 2006-05-12]. Dostupný z WWW: <http://www.volny.cz/zaloudekjiri/rp/index.html>.

[12] *Web Services Activity* [online]. 2002-2006 [cit. 2006-05-12]. Dostupný z WWW: <http://www.w3.org/2002/ws/>.

[13] *Web Services Description Language (WSDL) 1.1* [online]. 2001 [cit. 2006-05-01]. Dostupný z WWW: <http://www.w3.org/TR/wsdl>.

[14] SCHAUB, Stephen. Creating Database Web Applications with Eclipse. *Eclipse corner article* [online]. 2006 [cit. 2006-05-01] Dostupný z WWW: <http://www.eclipse.org/articles/Article-EclipseDbWebapps/article.html>.

[15] MINKIN, Boris. Developing web services "Eclipse Web Tools Project". *JDJ : The worlds leading java resource* [online]. 2006 [cit. 2006-05-01] Dostupný z WWW: <http://java.sys-con.com/read/180402.htm>.

[16] NetBeans IDE - Quick start Guide for Web Services. *NetBeans docs and support* [online]. 2006 [cit. 2006-04-19] Dostupný z WWW: <http://www.netbeans.org/kb/50/quickstart-webservice.html>.

[17] NetBeans IDE - Quick start Guide for Web Services Clients. *NetBeans docs and support* [online]. 2006 [cit. 2006-04-19] Dostupný z WWW: <http://www.netbeans.org/kb/50/quickstart-webservice-client.html>.

[18] ŠEDA, Jan. Úvod do JDBC. Interval [online]. 2003 [cit. 2006-05-01] Dostupný z WWW: < http://interval.cz/clanky/uvod-do-jdbc/>.

[19] PUŠ, Petr. Poznáváme C# a Microsoft .NET - 54. díl : ADO.NET. *Živě.cz* [online]. 2005 [cit. 2006-05-01]. Dostupný z WWW: <http://www.zive.cz/h/Programovani/AR.asp?ARI=127344>.

[20] BERNHARDT, Peter. Building a Family History Web Service. Coding4Fun [online]. 2005 [cit. 2006-05-04]. Dostupný z WWW: <http://msdn.microsoft.com/coding4fun/xmlforfun/familyhistory/default.aspx>.

[21] BERNHARDT, Peter. Building a Family History Web Service Client. Coding4Fun [online]. 2005 [cit. 2006-05-04]. Dostupný z WWW: <http://msdn.microsoft.com/coding4fun/xmlforfun/familyhistory2/default.aspx>.

[22] NICHOL, Scott. Introduction to NuSOAP. Nusoap [online]. 2003 [cit. 2006-04-20]. Dostupný z WWW: <http://www.scottnichol.com/nusoapintro.htm>.

[23] NICHOL, Scott. Programming with NuSOAP. Nusoap [online]. 2003 [cit. 2006-04-20]. Dostupný z WWW: <http://www.scottnichol.com/nusoapprog.htm>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application Program Interface
ASP	Active Server Pages
CASE	Computer Aided Software Engineering
CLR	Common Language Runtime
DB	DataBase
DLL	Dynamic Link Library
DOM	Document Object Model
EMF	Eclipse Modeling Framework
EXPAT	XML Parser Toolkit
FCL	.NET Framework Class Library
GEF	Graphical Editor Framework
HTML	HyperText Markup Language
IDE	Integrated Development Environment
J2EE	Java 2 Enterprise Edition
J2SE	Java 2 Standard Edition
JAX	Java APIs for XML
JAXB	Java API for XML Binding
JAXM	Java API for XML Messaging
JAXP	Java API for XML Parsing
JAXR	Java API for XML Repositories
JAX-RPC	Java API for XML based Remote Procedure Calling
JDBC	Java Data Base Connectivity
JDOM	Java Document Object Model
JEM	Java EMF Model

JSP	Java Server Pages
MS	Microsoft
MSIL	Microsoft Intermediate Language
NASSL	Network Accessible Service Specification Language
ODBC	Open Data Base Connectivity
OLE	Object Linking and Embedding
OLE DB	Object Linking and Embedding for Databases
OMG	Open Management Group
PHP	PHP: Hypertext Preprocessor aka Personal Home Page tools
RAD	Rapid Application Development
SAX	Simple API for XML
SCL	SOAP Contract Language
SDK	System Development Kit
SDL	Service Description Language
SMTP	Simple Mail Transport Protocol
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
URI	Uniform Resources Identifiers
URL	Uniform Resources Locators
VCL	Visual Component Library
W3C	World Wide Web (WWW-W3) Consortium
WDDX	Web Distributed Data Exchange
WSDL	Web Service Definition Language

WSDP	Web Service Developer Pack
WTP	Web Tool Project
XMI	XML Metadata Interchange
XML	Extensible Markup Language
XML-RPC	XML – Remote Procedure Calling
XSL	Extensible Stylesheet Language
XSLT	XSL Transformation

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I: PŘÍKLAD VYTVOŘENÍ XML DOKUMENTU V PROSTŘEDÍ JAVA A C#

```
// Java
public class XMLLicence {

public void createXMLOptionsFile(String pxadr, String pxusr, String pxpswd,
   String pc, String dm) throws Exception {

   DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
   DocumentBuilder builder = dbf.newDocumentBuilder();
   Document doc = builder.parse("licence.xml");
   DOMImplementation impl = builder.getDOMImplementation();
   doc = impl.createDocument("", "UserLogin", null);

   Node JSEP = doc.getDocumentElement();
   Element MainOptions = doc.createElement("MainOptions");
   Element ProxyAdr = doc.createElement("ProxyAdr");
   Element ProxyUser = doc.createElement("ProxyUser");
   Element ProxyPasswd = doc.createElement("ProxyPasswd");

   Text ProxyAdrText = doc.createTextNode(pxadr);
   Text ProxyUserText = doc.createTextNode(pxusr);
   Text ProxyPasswdText = doc.createTextNode(pxpswd);

   JSEP.appendChild(MainOptions);
   MainOptions.appendChild(ProxyAdr);
   MainOptions.appendChild(ProxyUser);
   MainOptions.appendChild(ProxyPasswd);

   ProxyAdr.appendChild(ProxyAdrText);
   ProxyUser.appendChild(ProxyUserText);
   ProxyPasswd.appendChild(ProxyPasswdText);

   TransformerFactory tf = TransformerFactory.newInstance();
   Transformer zapisovac = tf.newTransformer();
   zapisovac.setOutputProperty(OutputKeys.INDENT, "yes");
   zapisovac.transform(new DOMSource(doc), new StreamResult(new File(fileXml)));
}

}
```

```
// C#
public class XMLWriterPriklady {
  public static void ZapisZamestnance()  {

    ArrayList lZamestnanci = new ArrayList();
    lZamestnanci.Add(new Zamestanec("Jan","Novak","Analytik"));
    lZamestnanci.Add(new Zamestanec("Jiri","Joudek","Vyvojar"));
    XmlTextWriter lWriter = null;
    try {
      lWriter = new XmlTextWriter(new StreamWriter("C:/zamestnanci2.xml"));
      //zapiseme start dokumentu
      lWriter.WriteStartDocument();
      //zapiseme korenovy element
      lWriter.WriteStartElement("zamestnanci");
      foreach(Zamestanec lZamestnanec in lZamestnanci)
      {
        lWriter.WriteStartElement("zamestnanec");
        lWriter.WriteAttributeString("jmeno",lZamestnanec.Jmeno);
        lWriter.WriteAttributeString("prijmeni",lZamestnanec.Prijmeni);
        lWriter.WriteAttributeString("pozice",lZamestnanec.Pozice);
        lWriter.WriteEndElement();
      }
      //uzavreme dokument
      lWriter.WriteEndDocument();
    } finally {
      lWriter.Close();
    }
  }
}
```

PŘÍLOHA P II: IMPLEMENTAČNÍ ČÁST WEBOVÉ SLUŽBY V JAZYCE C#

```
<%@ WebService Language="C#" Class="DVDMovie" %>

using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Data;
using MySql.Data.MySqlClient;

[WebService(Namespace = "http://altos.wz.cz/DVDMovie/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class DVDMovie  : System.Web.Services.WebService {

    [WebMethod]
    public void addMovie(String name, String locname, int year, String director) {

        MySqlConnection conn =
            new MySqlConnection("server=localhost;database=dvdmovie;uid=root;pwd=");
        string sql = "INSERT INTO video " +
                     "VALUES ('','" + name + "','" + locname + "','" + year + "','" + di-
rector + "','')";
        MySqlCommand command = conn.CreateCommand();
        command.CommandText = sql;
        conn.Open();
        command.ExecuteNonQuery();
        conn.Close();
    }

    [WebMethod]
    public void borrowMovie(int id)
    {
        DateTime myDate = DateTime.Now;
        MySqlConnection conn =
            new MySqlConnection("server=localhost;database=dvdmovie;uid=root;pwd=");
        string sql = "UPDATE video " +
                     "SET borrowdate='" + myDate.Year + "-" + myDate.Month + "-" + myDa-
te.Day + "' " +
                     "WHERE id=" + id;
        MySqlCommand command = conn.CreateCommand();
        command.CommandText = sql;
        conn.Open();
        command.ExecuteNonQuery();
        conn.Close();
    }

    [WebMethod]
    public void returnMovie(int id)
    {

        MySqlConnection conn =
            new MySqlConnection("server=localhost;database=dvdmovie;uid=root;pwd=");
        string sql = "UPDATE video " +
                     "SET borrowdate='' " +
                     "WHERE id=" + id;
        MySqlCommand command = conn.CreateCommand();
        command.CommandText = sql;
        conn.Open();
        command.ExecuteNonQuery();
        conn.Close();
    }

    [WebMethod]
    public String getMovieBorrowFrom(int id)
    {

        MySqlConnection conn =
            new MySqlConnection("server=localhost;database=dvdmovie;uid=root;pwd=");
        string sql = "SELECT * FROM video WHERE id=" + id;
        MySqlCommand command = conn.CreateCommand();
        command.CommandText = sql;
        conn.Open();
        MySqlDataReader reader = command.ExecuteReader();
        string result = null;
        while (reader.Read())
```

```
        {
            result = reader.GetString(5);
        }
        reader.Close();
        conn.Close();
        return result;
    }

    [WebMethod]
    public Boolean getMovieBorrow(int id) {

        MySqlConnection conn =
            new MySqlConnection("server=localhost;database=dvdmovie;uid=root;pwd=");
        string sql = "SELECT * FROM video WHERE id=" + id;
        MySqlCommand command = conn.CreateCommand();
        command.CommandText = sql;
        conn.Open();
        MySqlDataReader reader = command.ExecuteReader();
        string result = null;
        while (reader.Read()) {
             result = reader.GetString(5);
        }
        reader.Close();
        conn.Close();
        if (result == "0.0.0000" || result == null)
            return false;
        else
            return true;
    }

    [WebMethod]
    public DataSet getMovieInfo(String name) {

        MySqlConnection conn =
            new MySqlConnection("server=localhost;database=dvdmovie;uid=root;pwd=");
        string sql = "SELECT name,locname,year,director FROM video WHERE name='" + name +
"'";
        MySqlCommand command = conn.CreateCommand();
        command.CommandText = sql;
        MySqlDataAdapter dadap = new MySqlDataAdapter();
        dadap.SelectCommand = command;
        DataSet myDataSet = new DataSet();
        conn.Open();
        dadap.Fill(myDataSet, "video");
        conn.Close();
        return myDataSet;
    }
}
```

PŘÍLOHA P III: IMPLEMENTAČNÍ ČÁST WEBOVÉ SLUŽBY V JAZYCE JAVA

```
package cz.wz.altos;

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

public class DVDMovieImpl {


    // Enter web service operations here. (Popup menu: Web Service->Add Operation)
    private static final String CONNSTRING =
        "jdbc:mysql://localhost/video?user=root&password=tequila";

    public void addMovie(String name, String locname, int year,
                String director) {

                try {
                        Class.forName("com.mysql.jdbc.Driver");
                } catch (ClassNotFoundException e) {
                        e.printStackTrace();
                }
                Connection conn = null;
                try {
                        conn = DriverManager.getConnection(CONNSTRING);
                        Statement stmt = conn.createStatement();
                        String sql = "INSERT INTO dvdtitle " +
                                                "(name, locname, year, directed) " +
                                                "VALUES ('" + name +
                                                "','" + locname +
                                                "'," + year +
                                                ",'" + director + "')";
                        stmt.executeUpdate(sql);
                } catch (SQLException e) {
                        e.printStackTrace();
                }
    }

    public void borrowMovie(int id) {

        Date myDate = new Date(System.currentTimeMillis());

                try {
                        Class.forName("com.mysql.jdbc.Driver");
                } catch (ClassNotFoundException e) {
                        e.printStackTrace();
                }
                Connection conn = null;
                try {
                        conn = DriverManager.getConnection(CONNSTRING);
                        Statement stmt = conn.createStatement();
                        String sql = "UPDATE dvdtitle " +
                                                "SET borrowdate='" + myDate + "' " +
                                                "WHERE id=" + id;
                        stmt.executeUpdate(sql);
                } catch (SQLException e) {
                        e.printStackTrace();
                }
    }

    public void returnMovie(int id) {

                try {
                        Class.forName("com.mysql.jdbc.Driver");
                } catch (ClassNotFoundException e) {
                        e.printStackTrace();
                }
                Connection conn = null;
                try {
                        conn = DriverManager.getConnection(CONNSTRING);
                        Statement stmt = conn.createStatement();
                        String sql = "UPDATE dvdtitle " +
```

```
                                        "SET borrowdate='0000-00-00' " +
                                        "WHERE id=" + id;
                stmt.executeUpdate(sql);
            } catch (SQLException e) {
                e.printStackTrace();
            }
    }

    public String getMovieBorrowFrom(int id) {

            try {
                Class.forName("com.mysql.jdbc.Driver");
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
            Connection conn = null;
            Date result = null;
            try {
                conn = DriverManager.getConnection(CONNSTRING);
                Statement stmt = conn.createStatement();
                String sql = "SELECT * FROM dvdtitle WHERE id=" + id;
                ResultSet set = stmt.executeQuery(sql);
                while (set.next()) {
                    result = set.getDate(6);
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }

       return result.toString();
    }

    public boolean getMovieBorrow(int id) {

            try {
                Class.forName("com.mysql.jdbc.Driver");
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
            Connection conn = null;
            String result = null;
            try {
                conn = DriverManager.getConnection(CONNSTRING);
                Statement stmt = conn.createStatement();
                String sql = "SELECT * FROM dvdtitle WHERE id=" + id;
                ResultSet set = stmt.executeQuery(sql);
                while (set.next()) {
                    result = set.getString(6);
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
       if (result == null)
            return false;
       else
            return true;
    }

    public String getMovieInfo(String name) {

            try {
                Class.forName("com.mysql.jdbc.Driver");
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
            Connection conn = null;
            Date myDate = new Date(System.currentTimeMillis());
            String result = null;
            try {
                conn = DriverManager.getConnection(CONNSTRING);
                Statement stmt = conn.createStatement();
                String sql = "SELECT * FROM dvdtitle WHERE name='" + name + "'";
                ResultSet set = stmt.executeQuery(sql);
                while (set.next()) {
                    result = "<nazev>" + set.getString(2) + "</nazev>" +
                                "<locna>" + set.getString(3) + "</locna>" +
                                "<rok>" + set.getString(4) + "</rok>" +
                                "<rezie>" + set.getString(5) + "</rezie>" +
                                "<datum>" + set.getString(6) + "</datum>";
                }
            } catch (SQLException e) {
```

```
                    e.printStackTrace();
                }
        if (result==null)
            return "Nenalezen film s timto jmenem.";
        else
            return result;
    }
}
```

PŘÍLOHA P IV: IMPLEMENTAČNÍ ČÁST WEBOVÉ SLUŽBY V JAZYCE PHP

```
<?php

require_once('nusoap.php');

$server = new soap_server;
$server->register('addMovie');
$server->register('borrowMovie');
$server->register('returnMovie');
$server->register('getMovieBorrowFrom');
$server->register('getMovieBorrow');
$server->register('getMovieInfo');

function addMovie($name,$locname,$year,$director)
{
  include "dfn_mysqlconnect.php";
  if (!$spojeni):
    echo "Nepodařilo se připojit k MySQL serveru";
  endif;
  $sql="INSERT INTO video VALUES
('','".$name."','".$locname."','".$year."','".$director."','');";
  $result=mysql_DB_Query("dvdmovie",$sql,$spojeni);
  mysql_Close();
}

function borrowMovie($id)
{
  include "dfn_mysqlconnect.php";
  $sql="UPDATE video SET borrowdate='".Date("Y-m-d")."' WHERE id=".$id.";";
  $result=mysql_DB_Query("dvdmovie",$sql,$spojeni);
  mysql_Close();
}

function returnMovie($id)
{
  include "dfn_mysqlconnect.php";
  $sql="UPDATE video SET borrowdate='0000-00-00' WHERE id=".$id.";";
  $result=mysql_DB_Query("dvdmovie",$sql,$spojeni);
  mysql_Close();
}

function getMovieBorrowFrom($id)
{
  include "dfn_mysqlconnect.php";
  $sql="SELECT * FROM video WHERE id=".$id.";";
  $result=mysql_DB_Query("dvdmovie",$sql,$spojeni);
  while ($zaznam=mysql_Fetch_Row($result)):
    $vysledek = $zaznam[5];
  endwhile;
  mysql_Close();
  return $vysledek;
}

function getMovieBorrow($id)
{
  include "dfn_mysqlconnect.php";
  $sql="SELECT * FROM video WHERE id=".$id.";";
  $result=mysql_DB_Query("dvdmovie",$sql,$spojeni);
  while ($zaznam=mysql_Fetch_Row($result)):
    $vysledek = $zaznam[5];
  endwhile;
  mysql_Close();
  if (!$vysledek || $vysledek=="0000-00-00"):
    return 0;
  else:
    return 1;
  endif;
}

function getMovieInfo($name)
{
  $vysledek = 0;
  include "dfn_mysqlconnect.php";
  $sql="SELECT * FROM video WHERE name='".$name."';";
  $result=mysql_DB_Query("dvdmovie",$sql,$spojeni);
```

```
  while ($zaznam=mysql_Fetch_Row($result)):
    $vysledek = htmlspecialchars("<title><name>".$zaznam[1].
                                 "</name><locname>".$zaznam[2].
                                 "</locname><year>".$zaznam[3].
                                 "</year><directed>".$zaznam[4].
                                 "</directed></title>");
  endwhile;
  mysql_Close();
  if ($vysledek):
    return $vysledek;
  else:
    return 0;
  endif;
}

$HTTP_RAW_POST_DATA = isset($HTTP_RAW_POST_DATA) ? $HTTP_RAW_POST_DATA : '';
$server->service($HTTP_RAW_POST_DATA);

?>
```

PŘÍLOHA P V: POPIS WEBOVÉ SLUŽBY DVDMOVIE VE WSDL

```
<?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://altos.wz.cz/DVDMovie/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" targetNamespa-
ce="http://altos.wz.cz/DVDMovie/" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
- <wsdl:types>
- <s:schema elementFormDefault="qualified" targetNamespace="http://altos.wz.cz/DVDMovie/">
- <s:element name="addMovie">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="name" type="s:string" />
  <s:element minOccurs="0" maxOccurs="1" name="locname" type="s:string" />
  <s:element minOccurs="1" maxOccurs="1" name="year" type="s:int" />
  <s:element minOccurs="0" maxOccurs="1" name="director" type="s:string" />
  </s:sequence>
  </s:complexType>
  </s:element>
- <s:element name="addMovieResponse">
  <s:complexType />
  </s:element>
- <s:element name="borrowMovie">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="id" type="s:int" />
  </s:sequence>
  </s:complexType>
  </s:element>
- <s:element name="borrowMovieResponse">
  <s:complexType />
  </s:element>
- <s:element name="returnMovie">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="id" type="s:int" />
  </s:sequence>
  </s:complexType>
  </s:element>
- <s:element name="returnMovieResponse">
  <s:complexType />
  </s:element>
- <s:element name="getMovieBorrowFrom">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="id" type="s:int" />
  </s:sequence>
  </s:complexType>
  </s:element>
- <s:element name="getMovieBorrowFromResponse">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="getMovieBorrowFromResult" type="s:string" />
  </s:sequence>
  </s:complexType>
  </s:element>
- <s:element name="getMovieBorrow">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="id" type="s:int" />
  </s:sequence>
  </s:complexType>
  </s:element>
- <s:element name="getMovieBorrowResponse">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="1" maxOccurs="1" name="getMovieBorrowResult" type="s:boolean" />
  </s:sequence>
  </s:complexType>
  </s:element>
- <s:element name="getMovieInfo">
- <s:complexType>
- <s:sequence>
  <s:element minOccurs="0" maxOccurs="1" name="name" type="s:string" />
  </s:sequence>
```

```
  </s:complexType>
  </s:element>
- <s:element name="getMovieInfoResponse">
- <s:complexType>
- <s:sequence>
- <s:element minOccurs="0" maxOccurs="1" name="getMovieInfoResult">
- <s:complexType>
- <s:sequence>
  <s:element ref="s:schema" />
  <s:any />
  </s:sequence>
  </s:complexType>
  </s:element>
  </s:sequence>
  </s:complexType>
  </s:element>
  </s:schema>
  </wsdl:types>
- <wsdl:message name="addMovieSoapIn">
  <wsdl:part name="parameters" element="tns:addMovie" />
  </wsdl:message>
- <wsdl:message name="addMovieSoapOut">
  <wsdl:part name="parameters" element="tns:addMovieResponse" />
  </wsdl:message>
- <wsdl:message name="borrowMovieSoapIn">
  <wsdl:part name="parameters" element="tns:borrowMovie" />
  </wsdl:message>
- <wsdl:message name="borrowMovieSoapOut">
  <wsdl:part name="parameters" element="tns:borrowMovieResponse" />
  </wsdl:message>
- <wsdl:message name="returnMovieSoapIn">
  <wsdl:part name="parameters" element="tns:returnMovie" />
  </wsdl:message>
- <wsdl:message name="returnMovieSoapOut">
  <wsdl:part name="parameters" element="tns:returnMovieResponse" />
  </wsdl:message>
- <wsdl:message name="getMovieBorrowFromSoapIn">
  <wsdl:part name="parameters" element="tns:getMovieBorrowFrom" />
  </wsdl:message>
- <wsdl:message name="getMovieBorrowFromSoapOut">
  <wsdl:part name="parameters" element="tns:getMovieBorrowFromResponse" />
  </wsdl:message>
- <wsdl:message name="getMovieBorrowSoapIn">
  <wsdl:part name="parameters" element="tns:getMovieBorrow" />
  </wsdl:message>
- <wsdl:message name="getMovieBorrowSoapOut">
  <wsdl:part name="parameters" element="tns:getMovieBorrowResponse" />
  </wsdl:message>
- <wsdl:message name="getMovieInfoSoapIn">
  <wsdl:part name="parameters" element="tns:getMovieInfo" />
  </wsdl:message>
- <wsdl:message name="getMovieInfoSoapOut">
  <wsdl:part name="parameters" element="tns:getMovieInfoResponse" />
  </wsdl:message>
- <wsdl:portType name="DVDMovieSoap">
- <wsdl:operation name="addMovie">
  <wsdl:input message="tns:addMovieSoapIn" />
  <wsdl:output message="tns:addMovieSoapOut" />
  </wsdl:operation>
- <wsdl:operation name="borrowMovie">
  <wsdl:input message="tns:borrowMovieSoapIn" />
  <wsdl:output message="tns:borrowMovieSoapOut" />
  </wsdl:operation>
- <wsdl:operation name="returnMovie">
  <wsdl:input message="tns:returnMovieSoapIn" />
  <wsdl:output message="tns:returnMovieSoapOut" />
  </wsdl:operation>
- <wsdl:operation name="getMovieBorrowFrom">
  <wsdl:input message="tns:getMovieBorrowFromSoapIn" />
  <wsdl:output message="tns:getMovieBorrowFromSoapOut" />
  </wsdl:operation>
- <wsdl:operation name="getMovieBorrow">
  <wsdl:input message="tns:getMovieBorrowSoapIn" />
  <wsdl:output message="tns:getMovieBorrowSoapOut" />
  </wsdl:operation>
- <wsdl:operation name="getMovieInfo">
  <wsdl:input message="tns:getMovieInfoSoapIn" />
  <wsdl:output message="tns:getMovieInfoSoapOut" />
  </wsdl:operation>
  </wsdl:portType>
- <wsdl:binding name="DVDMovieSoap" type="tns:DVDMovieSoap">
```

```
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
- <wsdl:operation name="addMovie">
  <soap:operation soapAction="http://altos.wz.cz/DVDMovie/addMovie" style="document" />
- <wsdl:input>
  <soap:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="borrowMovie">
  <soap:operation soapAction="http://altos.wz.cz/DVDMovie/borrowMovie" style="document" />
- <wsdl:input>
  <soap:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="returnMovie">
  <soap:operation soapAction="http://altos.wz.cz/DVDMovie/returnMovie" style="document" />
- <wsdl:input>
  <soap:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="getMovieBorrowFrom">
  <soap:operation soapAction="http://altos.wz.cz/DVDMovie/getMovieBorrowFrom" sty-
le="document" />
- <wsdl:input>
  <soap:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="getMovieBorrow">
  <soap:operation soapAction="http://altos.wz.cz/DVDMovie/getMovieBorrow" style="document"
/>
- <wsdl:input>
  <soap:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="getMovieInfo">
  <soap:operation soapAction="http://altos.wz.cz/DVDMovie/getMovieInfo" style="document" />
- <wsdl:input>
  <soap:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
- <wsdl:binding name="DVDMovieSoap12" type="tns:DVDMovieSoap">
  <soap12:binding transport="http://schemas.xmlsoap.org/soap/http" />
- <wsdl:operation name="addMovie">
  <soap12:operation soapAction="http://altos.wz.cz/DVDMovie/addMovie" style="document" />
- <wsdl:input>
  <soap12:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap12:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="borrowMovie">
  <soap12:operation soapAction="http://altos.wz.cz/DVDMovie/borrowMovie" style="document"
/>
- <wsdl:input>
  <soap12:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap12:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="returnMovie">
```

```
  <soap12:operation soapAction="http://altos.wz.cz/DVDMovie/returnMovie" style="document"
/>
- <wsdl:input>
  <soap12:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap12:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="getMovieBorrowFrom">
  <soap12:operation soapAction="http://altos.wz.cz/DVDMovie/getMovieBorrowFrom" sty-
le="document" />
- <wsdl:input>
  <soap12:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap12:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="getMovieBorrow">
  <soap12:operation soapAction="http://altos.wz.cz/DVDMovie/getMovieBorrow" sty-
le="document" />
- <wsdl:input>
  <soap12:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap12:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
- <wsdl:operation name="getMovieInfo">
  <soap12:operation soapAction="http://altos.wz.cz/DVDMovie/getMovieInfo" style="document"
/>
- <wsdl:input>
  <soap12:body use="literal" />
  </wsdl:input>
- <wsdl:output>
  <soap12:body use="literal" />
  </wsdl:output>
  </wsdl:operation>
  </wsdl:binding>
- <wsdl:service name="DVDMovie">
- <wsdl:port name="DVDMovieSoap" binding="tns:DVDMovieSoap">
  <soap:address location="http://localhost:1253/DVDMovie/DVDMovie.asmx" />
  </wsdl:port>
- <wsdl:port name="DVDMovieSoap12" binding="tns:DVDMovieSoap12">
  <soap12:address location="http://localhost:1253/DVDMovie/DVDMovie.asmx" />
  </wsdl:port>
  </wsdl:service>
  </wsdl:definitions>
```