VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODULÁRNÍ ARCHITEKTURA DISTRIBUOVANÝCH SYSTÉMŮ

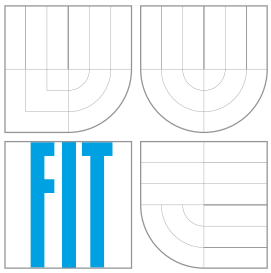
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. JIŘÍ MUSIL

BRNO 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODULÁRNÍ ARCHITEKTURA DISTRIBUOVANÝCH SYSTÉMŮ

MODULAR ARCHITECTURE OF DISTRIBUTED SYSTEMS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. JIŘÍ MUSIL

AUTHOR

VEDOUCÍ PRÁCE — Doc. Dr. Ing. PETR HANÁČEK

SUPERVISOR

BRNO 2007

Abstrakt

Tradiční architektury softwarových systémů jsou v heterogenním prostředí dnešních počítačových sítí příliš težkopadné. Tento problém se snaží řešit myšlenka architektury orientované na služby (Service Oriented Architecture – SOA). Systémy postavené nad jejími principy jsou vysoce škálovatelné a snadno integrovatelné. Praktickou možnost implementace SOA nabízejí v současné době webové služby (WS) postavené zejména nad otevřenými protokoly SOAP a XML-RPC. Tato diplomová práce se zaměřila na problém poskytování kontextových informací mobilním zařízením a způsob jeho řešení založený na principech SOA. Práce představuje návrh a implementaci webové služby poskytující kontextové informace mobilním zařízením a škálovatelný inverzní SOAP proxy server, který bude umožňovat její efektivní monitoring a management.

Klíčová slova

distribuované systémy, architektura orientovaná na služby, SOA, webové služby, SOAP, WSDL, UDDI, XML, inverzní proxy server, mobilní zařízení, kontextové informace

Abstract

Traditional architectures of software systems are in heterogenous environment of today's computer networks too heavy-footed. One of principles, which tries to solve this problem is service-oriented architecture (SOA). Practical way of its implementation is represented by web services (WS) built upon protocols like SOAP or XML-RPC. This diploma thesis focuses problem of providing contextual information to mobile devices and its solution based on SOA principles. The thesis introduces design and implementation of web service providing contextual information to mobile devices and prototype of modular inverse SOAP proxy server allowing its effective monitoring and management.

Keywords

distributed systems, service-oriented architecture, SOA, web services, SOAP, WSDL, UDDI, XML, inverse proxy server, mobile devices, contextual information

Citace

Jiří Musil: Modulární architektura distribuovaných systémů, diplomová práce, Brno, FIT VUT v Brně, 2007

Modulární architektura distribuovaných systémů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Doc. Dr. Ing. Petra Hanáčka

.....

Jiří Musil

22. května 2007

Zadání diplomové práce/5920/2006/xmusil21

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2006/2007

Zadání diplomové práce

Řešitel: **Musil Jiří, Bc.**
Obor: Informační systémy
Téma: **Modulární architektura distribuovaných aplikací**
Kategorie: Počítačové sítě

Pokyny:

1. Seznamte se tvorbou distribuovaných aplikací.
2. Seznamte se s protokoly pro vzdálené volání procedur v heterogenních sítích.
3. Navrhněte prototyp modulárního systému pro vzdálený přístup k datům přes webové služby. Zaměřte se na škálovatelnost systému.
4. Implementujte prototyp systému.
5. Implementujte ukázkové moduly a ověřte jejich funkčnost.

Literatura:

- Podle pokynů vedoucího

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body zadání 1-3

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Hanáček Petr, doc. Dr. Ing.**, UITS FIT VUT
Datum zadání: 28. února 2006
Datum odevzdání: 22. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav inteligentních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Bc. Jiří Musil**
Id studenta: 49404
Bytem: Č. p. 4, 582 22 Olešenka
Narozen: 16. 03. 1983, Havlíčkův Brod
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
diplomová práce

Název VŠKP: Modulární architektura distribuovaných aplikací
Vedoucí/školitel VŠKP: Hanáček Petr, doc. Dr. Ing.
Ústav: Ústav inteligentních systémů
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☒ 10 let po uzavření této smlouvy
   - (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Obsah

Kapitola 1

Úvod

Cílem komerčních organizací je rychle reagovat na změny na trhu a nové požadavky klientů i obchodních partnerů. Tradiční architektury softwarových systémů jsou ale v současném, vysoce heterogenním, prostředí počítačových sítí velmi težkopadné. Na tento problém odpovídá myšlenka architektury orientované na služby (Service-oriented Architecture – SOA). Systémy postavené nad principy SOA jsou vysoce škálovatelné a snadno integrovatelné. Praktickou možnost implementace myšlenky SOA nabízejí v současné době webové služby (WS) postavené nad protokoly SOAP a XML-RPC.

Tato práce se zabývá vytvořením webové služby poskytující kontextové informace mobilním zařízením (KIM) a prototypu inverzního SOAP proxy serveru pro webové služby (ISPS). KIM na základě kontextu mobilního zařízení vybere z databáze relevantní informace, které se na tomto zařízení zobrazí. Kontext mobilního zařízení je dán veškerými informacemi o zažízením samotném (výrobce, vlastnosti) ale i stavu ve kterém se nachazí (poloha, čas, spuštěná aplikace). Smyslem ISPS je vytvořit jednu vstupní bránu k webovým službám organizace, která umožňuje jednoduchý monitoring a management nabízených WS. Návrh ISPS bude vycházet z požadavku na provoz v heterogenním prostředí a zaměří se tedy především na jeho škálovatelnost.

V první časti této práce budou popsány principy distribuovaných systémů a architektury orientované na služby. Budou také představeny webové služby jako způsob implementace SOA a protokoly, které se v současné době při implementaci WS používají.

Druhá část se bude zabývat analýzou požadavků na inverzní SOAP proxy server a webovou službu poskytující kontextové informace mobilním zařízením. Na základě této analýzy bude představen návrh jejich architektury.

Závěrečná část se bude věnovat popisem implementace jednotlivých součástí tvořících architekturu představenou v návrhu a představí výsledky základních testů a návrhy na další rozvoj projektu.

Kapitola 2

Distribuované systémy

Distribuovaným systémem (DS) se rozumí množina nezávislých, obecně heterogenních výpočetních jednotek, které se uživateli jeví jako jeden výpočetní systém. Distribuovaný výpočet je pak decentralizovaný paralelní výpočet prováděný dvěma a více vzájemně propojenými a komunikujícími počítači.

2.1 Požadavky a cíle

Návrh DS je zaměřen zejména na transparentnost, distribuovanost, otevřenost, škálovatelnost a odolnost vůči poruchám. V následujícím textu budou tyto oblasti představeny podrobněji.

2.1.1 Transparentnost

Transparentnost zaručí, že uživatel nemá možnost rozlišit zda pracuje s jedním strojem či distribuovaným systémem. Komunikace mezi jednotlivými prvky DS je tedy před uživatelem skryta a celý systém se tváří jako jeden celek. Transparentnost systému je možné uvažovat na několika úrovních, kde zajistí, že před uživatelem zůstane skryta: forma přístupu k datům (při jejich různé reprezentaci uvnitř DS), způsob fyzického i logického umístění dat, migrace dat, relokace (přesuny dat mezi jednotlivými uzly během práce s těmito daty), poruchy a obnovy jednotlivých výpočetních uzlů, souběžné používání více uživateli a perzistentnost (aktuální umístění dat v paměti nebo na pevném disku).

2.1.2 Otevřenost

Otevřenost DS zaručuje, že služby které systém nabízí odpovídají příslušným standardům popisujícím syntaxi a sémantiku jejich rozhraní. Implementace standardizovaných rozhraní umožňuje přenostilnost mezi různými systémy a spolupráci aplikací postavených na odlišných plaftormách. Tato rozhraní bývájí často popsána některým platformě nezávislým jazykem. V současné době je velmi rozšířený formát Interface Definition Language (IDL).

2.1.3 Škálovatelnost

Škálovatelnost je vlastnost DS, která umožňuje jednoduché změny jeho výkonnosti přidáním nebo odebráním výpočetních zdrojů. Požadavek na dobrou škálovatelnost často přináší snížení výkonnosti systému jako celku, s tím jak se zvyšuje režie nutná pro provádění změn v DS. Škálovatelnost DS rozlišujeme ve dvou rovinách: horizontální škálovatelnost je změna výkonnosti DS přidáním/odebráním výpočetního uzlu, vertikální škálovatelnost je naopak změna výkonnosti dosažená změnou zdrojů jednoho výpočetního uzlu. Míra škálovatelnosti DS může být měřena v několika dimenzích:

- zátěž DS, kterou způsobí změna dostupných prostředků (změna by měla být rychlá a jednoduchá s co nejmenším dopadem na aktuální výkon DS),
- geografická škálovatelnost umožnuje systému neomezenou funkčnost nezávisle na vzdálenosti mezi jednotlivými výpočetními uzly,
- administrativní škálovatelnost zajistí, že funkčnost DS není omezena při používání v různých organizacích.

2.1.4 Robustnost

Robustnost jako jedna z nejdůležitějších vlastností DS zaručuje, že výpadek jednoho nebo více výpočetních uzlů neovlivní funkčnost DS jako celku. Při výpadcích se počítá se ztrátou výkonnosti danou odstavením výpočetních zdrojů, výpadek jako takový ale nikdy nemůže způsobit chybnou funkčnost DS.

2.2 Hardwarové a softwarové modely

DS mohou být postaveny na různých hardwarových a softwarových modelech. Konkretní model je určen požadavky, které plynou z daného použití DS. Některé modely kladou důraz na specifickou vlastnost DS, jiné se snaží být vyvážené a zohlednit všechny aspekty. DS můžeme dle modelu komunikace rozlišit na zejména následující typy:

- **klient-server** architektura je tvořena inteligentním klientem, který je branou pro komunikaci uživatele se serverem. Klient zpracovává data ze serveru a zobrazuje je uživateli a zároveň zpracovává uživatelské dotazy a zasílá je na server.
- **3-vrstvá architektura** přenáší inteligenci klienta do samostatné střední vrstvy a samotný klient zajišťuje pouze prezentaci dat (nese proto pak označení tenký klient). Tato architektura velmi ulehčuje nasazení daného software do provozu. V současné době na ní funguje většina Internetových aplikací.
- **N-vrstvá architektura** sestává z tenkého klienta, který pouze zprostředkovává komunikaci uživatele s aplikačními servery poskytujícími různé služby.

- **Cluster** je tvořen mnoha vysoce integrovanými výpočetními uzly, které jsou schopny paralelně zpracovávat jednu úlohu. Každý uzel clusteru zpracovává pouze část úkolu. Po jejich zpracování jsou dílčí výsledky poskladány v celek.
- **Peer-to-peer** je plně decentralizovaná architektura, postrádající jakýkoliv centrální prvek. Každý uzel má částečnou zodpovědnost a vlastní autonomii v systému.

2.3 Distribuované operační systémy

Úkolem distribuovaných operačních systémů je souběžná správa prostředků na více nezávislých počítačích takovým způsobem, aby se uživateli jevily jako jeden počítač. Jednotlivé počítače jsou v rámci systému zcela autonomní, mají vlastní kopii operačního systému, vlastní paměť a navzájem komunikují pomocí zpráv. Návrh distribuovaných operačních systémů se zaměřuje zejména na skrytí umístění jednotlivých procesů, souborů, či hardwarových prostředků, skrytí replikace souborů, transparentnost souběžného používání více uživateli (tato na dnešních systémech běžná vlastnost se v distribuovaném prostředí stává velmi komplikovanou), transparentnost paralelismu (aplikace psané běžným způsobem by měl být systém schopen rozdělit mezi více výpočetních jednotek). Dále musí návrh zajišťovat dostatečnou výkonnost. Důležitá je doba odpovědi na požadavek, množství najednou obsloužených požadavků, vyrovnávání zátěže mezi jednotlivými uzly a rychlost komunikace (propustnost komunikačních linek mezi jednotlivými uzly musí být dostatečná natolik, aby byly všechny uzly při velké zátěži systému maximálně využity).

2.4 Distribuované souborové systémy

Distribuovaný souborový systém (DSS) je souborový systém poskytující perzistentní uložiště pro soubory v podobě množiny uložných jednotek propojených počítačovou sítí. DSS musí poskytovat sadu operací se soubory, výlučný i sdílený přístup k souborům, robustní žurnálování, automatickou detekci chyb, systém práv a vlastnictví a plně transparentní přístup k souborům.

2.5 Distribuované databázové systémy

Distribuované databázové systémy (DDS) jsou databáze, jejichž úložná zařízení, nejsou fyzicky připojená k jedné výpočetní jednotce. Obecně mohou být rozmístěna na různých místech a propojena počítačovou sítí. Každé uložiště obsahuje část databáze – fragment. Tyto fragmenty jsou pak mezi jednotlivými uložišti replikována pro zajištění redundance při případné havárii některého z nich. Kromě principu fragmentace a replikace využívají DDS ještě dalších principů jako lokální autonomie, synchronní vs. asynchronní přístup k datům apd.

Každý počítač v DDS je označován jako uzel a v rámci databáze vystupuje jako klient i server. V DDS mluvíme o horizontalní, vertikální a smíšené fragmentaci. Horizontální fragment označuje podmnožinu řádků z tabulky, vertikální fragment je podmnožina sloupců z tabulky a smíšený fragment je kombinace předchozích. Dále DDS rozlišujeme na homogenní a heterogenní podle toho kolik systémů řízení báze dat (SŘBD) používají. Homogenní DDS mají pouze jeden SŘBD, heterogenní jich mají naopak více. DDS musí zajistit plnou transparentnost v přístupu k datům a databázovým transakcí. Transparentnost transakcí je zajištěna jejich rozdělením na podtransakce, které jsou zpracovávány příslušnymi uzly, kterých se týkají.

Kapitola 3

Architektura orientovaná na služby

Předchozí kapitola diskutovala na úvod obecné požadavky na distribuované systémy, možné architektury a stručně představila aspekty, které přináší distribuovanost do oblasti operačních, souborových a databázových systémů.

V této kapitole představíme technologie, jež přináší distribuovanost a její výhody do současného prostředí Internetu. V následujícím textu budou představeny principy architektury orientované na služby (Service-oriented Architecture – SOA) a protokoly, nad kterými je postavena.

Distributivita je v dnešním Internetu požadováná zejména z ekonomických důvodů. Všechny obchodní společnosti řeší stejné problémy: rychle reagovat na nové obchodní příležitosti a požadavky, snižovat náklady, rychle integrovat firemní systém se systémem obchodních partnerů či zákazníků a tak zvyšovat zisk z investic. Tyto požadavky nepřetržitě zvyšují nároky na architekturu SW systémů. SW architektury jsou nyní navrženy tak, aby umožnily distribuované zpracování, použití různých programovacích jazyků bežících na různých platformách, zredukovaly dobu implementace na minimum a využily nesčetného množství způsobů komunikace pro zrychlení jejich integrace.

Vznik SOA vycházel z myšlenky, že každá ucelená část software je firemní majetek, který by měl být maximálně využit jak v rámci jedné organizace tak být případně nabídnut jako produkt ostatním organizacím. SOA lze tedy definovat také jako architekturu, ve které jsou všechny funkce nezavislé služby s dobře definovaným rozhraním tvořící při definovaném použití obchodní proces firmy.

Myšlenka SOA není nová. Na jejím základě vzniklo mnoho různých technologií, dnes více či méně používaných. Jejich slabinou ale byla příliš náročná implementace či větší množství různých vzájemně nekompatibilních implementací. Příkladem je CORBA, jejímž cílem byla jednoduchá integrace aplikací provozovaných na různých platformách. Přílišná složitost tuto technologii ale odsoudila k použití ve velmi omezeném měřítku.

Prvotní pokusy uvést koncept SOA do praxe narazily na různé problémy dané nedostatečným definováním skutečných požadavků a přílišným zaměřením na technologii samotnou. Mezi tyto problémy patřila zejména složitost implementace. Současné systémy

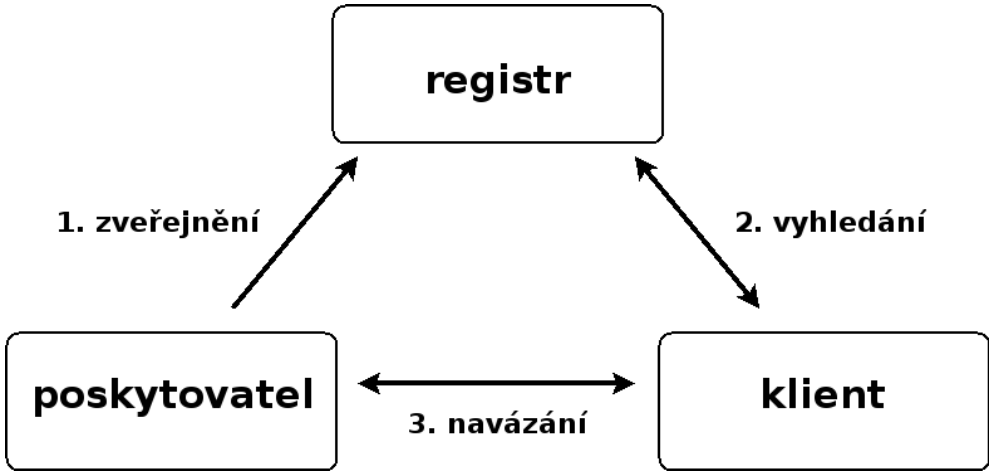
musí být vyvíjeny s důrazem na provoz v heterogenním prostředí, kde je nutné aby byly schopni pracovat na různém hardware, různých operačních systémech, middlewaru, programovacích jazycích, apd. a to s minimálním usilím při jejich implementaci v tom kterém prostředí. Dalším opomíjeným problemém byla redundance a znovupoužitelnost kódu. Konkurence schopný systém musí důsledně využívat principu opakovaného používaní stejného kódu. Tento jednoduchý princip ovšem nebylo a dodnes není jednoduché následovat z důvodu nekompatibility mezi různými techonologiemi použitými v dané organizaci a také mezi jednotlivými organizacemi. Velké těžkosti také vyvolalo mnoho různých rozhraní způsobujících velmi náročnou integraci SW. Každý integrovaný podsystém by měl mít obecně jedno rozhraní použitelné všemi ostatními podsystémy nehledě na platformu nebo programovací jazyk.

V současné době existuje již několik protokolů, nad kterými je možné efektivně budovat SOA. Tyto protokoly jsou diskutovány v dalších kapitolách. Obecně lze ale říci, že hlavním požadavkem SOA je nezávislost jednotlivých služeb a kvalitní management zahrnující například autorizaci požadavků, šifrování, validaci výsledků, logování transakcí, auditování, dynamické routovaní či možnost údržby služeb a správy verzí.

3.1 Webové služby

K implementaci SOA jsou nejčastěji využívané webové služby (WS). Organizace W3C definuje WS jako software navrhnutý pro podporu vzájemné interakce dvou počítačů komunikujících po síti. V praxi je WS rozhraní popisující sadu operací přístupných po síti pomocí standardizovaných zpráv. Toto rozhraní je je popsáno ve standardizovaném formátu a obsahuje všechny nezbytné informace pro jeho použití: transportní protokol, formáty zpráv a umístění služby. Rozhraní zakrývá implementaci služby a umožňuje její použití nezávisle na platformě a programovacím jazyku. Toto umožňuje aplikacím založeným na WS být velice volně propojené, orientované na komponenty, a implementované na různých platformách. Architektura WS je založena na interakci mezi třemi subjekty:

- poskytovatel služby je vlastníkem služby, hostuje službu na své platformě.
- Klient využívající službu je subjekt, který chce využít službu k dosažení nějakého cíle. Klient vyhledá službu a na základě jejího popisu ji využije.
- Registr WS obsahuje popisy služeb, které do něj byly vloženy jejich poskytovateli. Klient, který chce určitou službu využít prohledá registr a na základě jejího popisu použití služby zaintegruje. Zaintegrovat jí může staticky během vývoje svého systému či dynamicky za jeho běhu. Samotný popis služby není na registr nikterak vázaný. Klient může získat popis přímo od poskytovatele nebo ho stáhnout např. z FTP a webu.



Obrázek 3.1: Komunikace mezi jednotlivými subjekty WS

Samotná interakce zahrnuje zveřejnění služby, vyhledávání a navázání na službu (způsob jakým se služba zavolá). Všechny operace se pak týkají modulu, který implementuje danou službu a jejího popisu. Typické použití vypadá tak, že poskytovatel služby zveřejní popis služby buď přímo klientovi, který ji chce využít nebo na registr WS. Klient získá popis služby přímo od poskytovatele či ho vyhledá v registru WS a využije ho k navázání na službu a jejímu vyvolání. Vztah znázorňuje obrázek 3.1.

Až do této chvíle nebyla záměrně zmíněna žádná konkretní techonologie, která je použitá k implementaci WS, protože WS je obecný model komunikace. V praxi jsou však webové služby nejčastěji postaveny na následujících technologiích a protokolech:

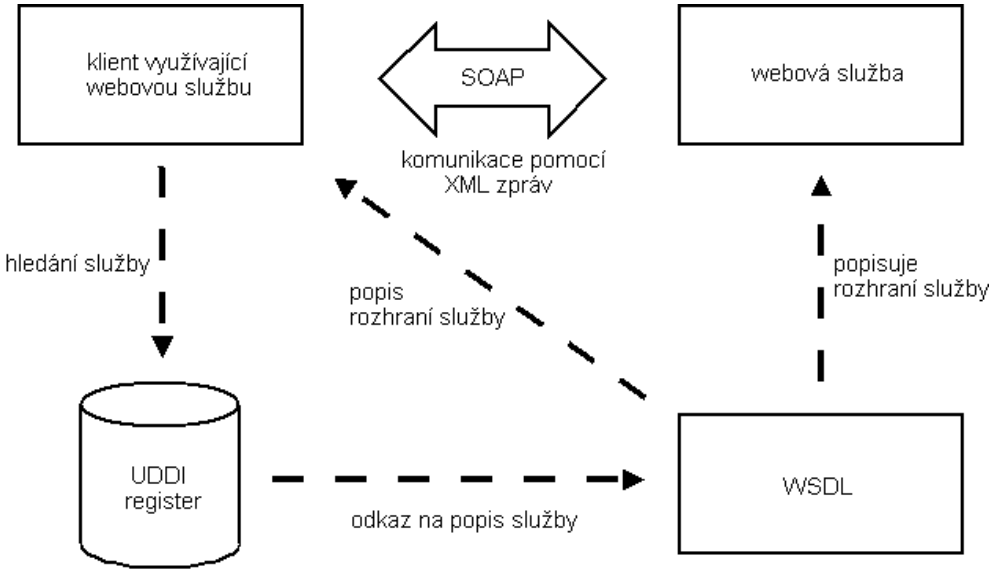
- SOAP (Simple Object Access Protocol) je protokol pro vzdálené volání procedur využívající XML pro formátování přenášených dat.
- WSDL (Web Service Definiton Language) je jazyk založený na XML sloužící pro popis poskytovaných služeb.
- UDDI (Universal Description, Discovery and Integration) je registr služeb založený na XML.

Způsob komunikace mezi výše uvedenými protokoly resp. technologiemi ukazuje obrázek 3.2, z něj je také zřejmá souvislost s obecným modelem webových služeb. Vztah jednotlivých protokolů je pak znázorněn na obrázku 3.3.

Následující podkapitoly se zabývají popisem jednotlivých technologií nad kterými jsou postaveny WS představené v této práci.

3.1.1 HTTP

Protokol HTTP (Hypertext Transfer Protocol) je ve WS nejčastějším používaným aplikačním protokolem. Důvodem je jeho jednoduchost, rozšířenost a fakt, že WS vyžadují



Obrázek 3.2: Komunikace mezi jednotlivými technologiemi WS

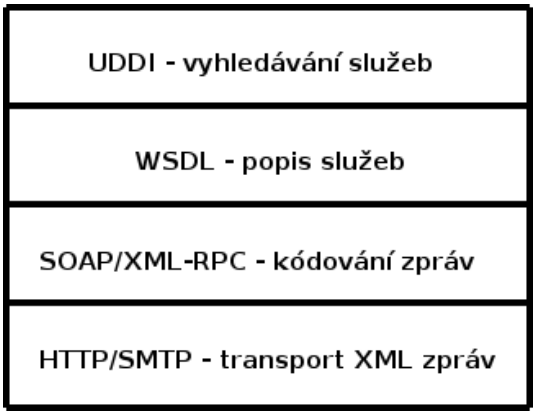
pouze přenos požadavku a odpovědi. HTTP má navíc širokou podporu v aplikacích a v podnikové sféře je to jeden z mála neblokovaných protokolů na úrovni firewallů a dalších aktivních síťových prvků.

Původní záměr pro vytvoření protokolu HTTP byl přenos HTML stránek. Standardizaci protokolu zajistili organizace W3C a IETF. Vznikl tak standard RFC 2616 popisující HTTP/1.1 – dnes používanou verzi. Tato verze přináší oproti té předchozí navíc zejména perzistentní spojení (TCP keep alive) a pipelining (klient může poslat požadavek bez obdržení odpovědi na předchozí), které snižují režii.

HTTP je protokol aplikační vrstvy založený na principu požadavek/odpověd. Klient, který naváže spojení se serverem a zašle požadavek je nazýván user-agent. Server, zpřístupňující data, vygeneruje na základě požadavku odpověd, kterou zašle zpět klientovi.

Jako transportní protokol využívá HTTP protokol TCP (Transmission Control Protocol). Klient vytvoří TCP spojení na server na odpovídají port (pro HTTP je obvyklý port 80). Server přijme spojení a čeká na požadavek klienta. Po přijetí požadavku zašle klient zpět informaci o stavu odpovědi (status line) např. “HTTP/1.1 200 OK” a samotnou zprávu v jejímž těle se nachází požadovaná data (html soubor, chybová zpráva, data ve formátu MIME, apd.).

Zdroje, které jsou přístupné přes HTTP jsou identifikovány pomocí URIs (Uniform Resource Identifiers) a HTTP nebo HTTPS URI schématu. Protokol HTTP je bezstavový a má jen několik příkazů, které jsou označovány jako metody, dále je také anonymní, pokud není autentizace explicitně vyžádána.



Obrázek 3.3: Vztah protokolů použitých pro implementaci WS

3.1.2 XML

V této kapitole nastíníme základní principy jazyka XML (eXtensible Markup Language), nad kterým jsou vytvořeny dnes běžně používané technologie webových služeb jako SOAP, WSDL nebo XML-RPC ale i další s WS přímo nesouvisející služby jako RSS, Atom, XHTML, apd.

XML je značkovací jazyk určený pro popis informací. Vychází z průmyslového standardu SGML (Standard Generalized Markup Language) a organizace W3C ho doporučuje pro popis dat. XML je podmnožina SGML a byl navržen tak, aby se dal oproti plnému SGML jednodušeji analyzovat a měl jednodušší podporu vícejazyčnosti. Tagy v XML nejsou předdefinované jako třeba u HTML a je nutné je nadefinovat.

Způsob jakým XML organizuje a popisuje data si ukážeme na příkladu:

```
1.  <?xml version=‘‘1.0’’ encoding=‘‘UTF-8’’?>
2.  <!DOCTYPE protocol [
3.          <!ENTITY prot_mainline ‘‘1’’>
4.          <!ENTITY prot_revision ‘‘13’’>
5.          <!ENTITY prot_version ‘‘&prot_mainline;.&prot_revision;’’>
6.  ]>
7.  <protocol group=’XML-RPCnew’>
8.      <head>
9.          <version>&prot_version;</version>
10.     </head>
11.     <data>
12.         <subject>Vnořená data</subject>
13.         Nějaká data.
14.     </data>
15. </protocol>
```

První řadek je volitelná hlavička udávájící znakovou sadu a verzi.

Řádky 2.-6. jsou popisem dat pomocí jazyka pro popis schémat DTD (Document Type Definition). DTD schéma je zde vložené přímo do XML dokumentu pro nadefinování vlastních entit. Entita je pojmenovaná část textu, která je při analýze XML nahrazena v jeho těle hodnotou. My jsme ji využili pro přenesení verze fiktivního protokolu do schématu.

XML schéma je obecně popis typu dat v XML dokumentu tvořený sadou omezení a zpřesnění pro jednotlivé elementy XML. Existují nástroje, které umí korektně validovat XML data, proti jejich příslušnému schématu. V současné době se používají zejména následující dva jazyky pro popis XML schémat:

- **DTD** – Document Type Definition – vychází ze SGML, v současné době už ale není schopen popsat všechny nové vlastnosti XML.
- **XSD** – XML Schema Definition – na XML postavený jazyk pro popis XML schéma dokumentu, které oproti DTD přináší zejména silné typování.

Řádky 7.-15. už jsou samotná data popsaná XML. V každém dokumentu by měl být právě jeden kořenový element (`protocol`). Každý element může mít atributy (řádek 7.) s hodnotami a obsah. Obsah mohou být další elementy nebo už samotná data. Elementy je tedy možné vnořovat.

Na jednoduchém příkladu jsme stručně popsali princip XML popisu dat, protože tvoří fundamentální součást technologie WS. V následujících kapitolách se už zaměříme přímo na protokoly WS.

3.1.3 SOAP

Protokol SOAP (Simple Object Access Protocol) je nejčastějším protokolem pro implementaci WS. Obecně je to protokol pro výměnnu informací v distribuovaném prostředí založený na XML. SOAP je nezávislý na platformě a programovacím jazyku. V současné době je aktuální W3C standard ve verzi SOAP 1.2. Jednou z velkých výhod tohoto protokolu je, že může běžet nad HTTP, který je široce podporovaný a není blokovaný na firewallech.

SOAP je obecný framework pro zasílání zpráv v XML formátu. To umožňuje nasadit ho v množství různých aplikací od zasílání zpráv po vzdálené volání procedur. Na obecné úrovni sestává SOAP ze tří částí:

- **Element envelope** obecně definuje jaké informace zpráva obsahuje, pro koho jsou určeny a zda jsou volitelné či povinné.
- **Pravidla kódování** definují mechanismus serializace jednotlivých datových typů.
- **Reprezentace RPC** (Remote Procedure Call) definuje konvence použitelné pro vzdálené volání procedur. Pro vyvolání vzdálené procedury jsou nutné následující

informace: URI cílového objektu, jméno metody, parametry metody a volitelně signatura metody.

Tato práce uvažuje SOAP jako dobrý prostředek pro vzdálené volání procedur což je také nejčastější použití tohoto protokolu. Vzdálené volání je organizováno jako model požadavek/odpověd. SOAP zpráva sestává z následujících XML elementů:

- **Envelope** je kořenový a tedy povinný element, pomocí kterého je XML dokument identifikován jako SOAP zpráva. Envelope obsahuje povinný element Body a nepovinné Header a Fault.
- **Header** je volitelný element obsahující dodatečné informace například pro identifikaci uživatele, autorizaci apd.
- **Body** je povinný element obsahující samotný požadavek nebo odpověd.
- **Fault** je volitelný element obsahující informace o připadných chybách, které se vyskytly při zpracování zprávy.

Tvar SOAP zprávy si vysvětlíme na následujícím příkladu. Požadavek směřuje na službu `StockQuote` a dotazuje se na poslední známý kurz akcií firmy označené symbolem DIS. Požadavek obsahuje pro jednoduchost pouze část Body. Pro úplnost byla ale ponechána i HTTP hlavička, kde je nejvýznamnějším prvkem pole `SOAPAction` určující, že se jedná o SOAP zprávu (používají ji například i firewally pro filtrovaní služeb). Příklad byl převzat z [18]. SOAP požadavek:

```
1.  POST /StockQuote HTTP/1.1
2.  Host: www.stockquoteserver.com
3.  Content-Type: text/xml; charset=‘‘utf-8’’
4.  Content-Length: nnnn
5.  SOAPAction: ‘‘Some-URI’’
6.  <SOAP-ENV:Envelope
7.    xmlns:SOAP-ENV=‘‘http://schemas.xmlsoap.org/soap/envelope/’’
8.    SOAP-ENV:encodingStyle=‘‘http://schemas.xmlsoap.org/soap/encoding/’’>
9.    <SOAP-ENV:Body>
10.     <m:GetLastTradePrice xmlns:m=‘‘Some-URI’’>
11.       <symbol>DIS</symbol>
12.         </m:GetLastTradePrice>
13.       </SOAP-ENV:Body>
14. </SOAP-ENV:Envelope>
```

Odpověd obsahuje opět pouze elemenet `Body` s návratovou hodnotou volané služby uzavřené v elementu `Price`. Stejně jako požadavek i odpověd musí být v HTTP hlavičce popsána jako `Content-Type` text/xml. SOAP odpověd:

```
1.  HTTP/1.1 200 OK
2.  Content-Type: text/xml; charset=‘‘utf-8’’
3.  Content-Length: nnnn
4.  <SOAP-ENV:Envelope
5.  xmlns:SOAP-ENV=‘‘http://schemas.xmlsoap.org/soap/envelope/’’
6.  SOAP-ENV:encodingStyle=‘‘http://schemas.xmlsoap.org/soap/encoding/’’/>
7.      <SOAP-ENV:Body>
8.          <m:GetLastTradePriceResponse xmlns:m=‘‘Some-URI’’>
9.              <Price>34.5</Price>
10.         </m:GetLastTradePriceResponse>
11.     </SOAP-ENV:Body>
12. </SOAP-ENV:Envelope>
```

Na řádku 5.-6. u odpovědi je možné si všimnout využití jmenných prostorů XML pro obálku a kódování. Jmenné prostory se v tomto případě používají pro odlišení SOAP a aplikačních identifikátorů. Obecně musí SOAP aplikace použít pro všechny elementy a atributy definované ve zprávě správný jmenný prostor. SOAP aplikace by měla odmítat zprávy s nesprávnými jmennými prostory. SOAP definuje následující dva jmenné prostory XML:

- Envelope: “http://schemas.xmlsoap.org/soap/envelope/”
- Kódování (serializace): “http://schemas.xmlsoap.org/soap/encoding/”

SOAP zpráva nesmí obsahovat DTD ani instrukce pro její zpracování.

3.1.4 XML-RPC

Ačkoliv tento protokol nebude v této práci použit je uveden pro doplnění přehledu technologií nad kterými jsou budovány WS. XML-RPC je, stejně jako SOAP, protokol pro vzdálené volání procedur založený na XML. Oproti SOAP je XML-RPC ale podstatně jednodušší jak specifikací tak implementačně. Tento protokol je narozdíl od SOAP pevně svázán s HTTP jako transportním protokolem a umí přenášet jen jednodušší datové typy.

Formát zpráv si ukážeme na následujícím příkladu. Z příkladu je zřejmá jednoduchost tohoto protokolu. Opět využíváme burzovních informací. Požadavek v XML-RPC:

```
1.  POST /rpcservice HTTP/1.0
2.  User-Agent: AgentX
3.  Host: www.example.com
4.  Content-Type: text/xml
5.  Content-length: nnn
6.  <?xml version=‘‘1.0’’?>
7.  <methodCall>
8.      <methodName>examples.getLastStockPrice</methodName>
```

```
9.      <params>
10.         <param>
11.             <value>
12.                 <string>DIS</string>
13.             </value>
14.         </param>
15.     </params>
16. </methodCall>
```

Povinné údaje v hlavičce jsou `User-Agent`, `Host`, `Content-Type` a `Content-length` (musí být správná). URI je nepovinná ale pomáhá webovému serveru ve vyhledání odpovídající služby. Tělo požadavku obsahuje kořenový element `methodCall`. Parametry musí být vnořené po jednom v elementu `params`. Parametry mohou být následujících typů: integer (32bit), boolean, řetězec, double, datum a čas ve formátu ISO 8601 a zakodovaná binární data do formátu base64. Ze složitějších datových typů lze v XML-RPC použít struktury a pole. Odpověd na předchozí požadavek bude v XML-RPC vypadat následovně:

```
1.  HTTP/1.1 200 OK
2.  Connection: close
3.  Content-Length: 158
4.  Content-Type: text/xml
5.  Date: Fri, 17 Jul 1998 19:55:08 GMT
2.  Server: ServerX
7.  <?xml version=‘‘1.0’’?>
8.  <methodResponse>
9.      <params>
10.         <param>
11.             <value>
12.                 <double>14.31</double>
13.             </value>
14.         </param>
15.     </params>
16. </methodResponse>
```

Odpověd na XML-RPC volání je, i při chybě, `200 OK`. Povinné údaje v hlavičce jsou stejné jako u požadavku kromě `User-Agent` a `Host`, které zde nejsou. Vrácené hodnoty jsou v kořenovém elementu `methodResponse` uložené stejně jako v požadavku parametry služby. Navíc může obsahovat element `fault`, který obsahuje elementy `faultCode` a `faultString` popisující detaily chyby, která se při zpracování vyskytla.

3.1.5 WSDL

WSDL (Web Service Description Language) je XML dokument sloužící pro popis webových služeb. Říká jak komunikovat s danou WS. Obsahuje tedy popis navázání na protokol a formát zprávy. Nabízené WS jsou popsány abstrakně a až následně navázány na konkretní protokol a formát zprávy. WSDL tedy popisuje veřejné rozhraní k WS. WSDL zatím není standardizováno organizací W3C.

Nejčastější použití WSDL je v kombinaci se SOAP a XSD pro popis Internetových WS. Teoretické použití vypadá tak, že klientský program přečte WSDL dokument a vybere jednu z nabízených služeb. Veškeré speciální datové typy jsou obsaženy v dokumentu v podobě definice pomocí XSD. Klient po vybrání služby použije SOAP pro její vzdálené volání. V praxi slouží WSDL dokumenty zatím spíše vývojářům jako formální popis nabízených WS. Rozhraní je pak využito při vývoji klienta, kde je volání služby staticky zaimplementováno podle předloženého popisu. WSDL může být rozšířeno o XLang, které popisuje službu a její zamýšlené chování v rámci celkového business procesu. WSDL dokument sestává z následujících elementů:

- **Types** – definice typů dat použitých ve zprávách, pro definici se nejčastěji používá XSD. Je pak na implementaci protokolu SOAP pro daný jazyk, aby korektně namapovala definované datové typy na své nativní.
- **Message** – definuje formát zpráv na základě definovaných datových typů, každá zpráva odpovídá parametru WS.
- **Operation** – operace, které podporuje služba. Tento element definuje vstupy a výstupy operace. Samotný vstup a výstup je popsán elementem `Message`. V SOAP modelu odpovídá tento element metodě.
- **PortType** – množina operací (Operation)
- **Binding** – definuje navázání určité operace na konkretní protokol a formát přenosu zpráv.
- **Port** – konkrétní koncový bod služby (určený síťovou adresou a definovanou vazbou – binding)
- **Service** – sdružuje několik souvisejících portů do jedné služby.

Následující příklad, převzatý z [18], ukazuje WSDL dokument popisující službu představenou v předchozí kapitole (`GetLastTradePrice`). V příkladu jsou viďet všechny výše zmíněné elementy:

```
1.  <?xml version=‘‘1.0’’?>
2.  <definitions name=‘‘StockQuote’’
3.
```

```
4.  targetNamespace=‘‘http://example.com/stockquote.wsdl’’
5.            xmlns:tns=‘‘http://example.com/stockquote.wsdl’’
6.            xmlns:xsd1=‘‘http://example.com/stockquote.xsd’’
7.            xmlns:soap=‘‘http://schemas.xmlsoap.org/wsdl/soap/’’
8.            xmlns=‘‘http://schemas.xmlsoap.org/wsdl/’’>
9.
10.     <types>
11.        <schema targetNamespace=‘‘http://example.com/stockquote.xsd’’
12.              xmlns=‘‘http://www.w3.org/2000/10/XMLSchema’’>
13.           <element name=‘‘TradePriceRequest’’>
14.               <complexType>
15.                   <all>
16.                       <element name=‘‘tickerSymbol’’ type=‘‘string’’/>
17.                   </all>
18.               </complexType>
19.            </element>
20.            <element name=‘‘TradePrice’’>
21.               <complexType>
22.                   <all>
23                        <element name=‘‘price’’ type=‘‘float’’/>
24.                   </all>
25.               </complexType>
26.            </element>
27.        </schema>
28.     </types>
29.
30.     <message name=‘‘GetLastTradePriceInput’’>
31.        <part name=‘‘body’’ element=‘‘xsd1:TradePriceRequest’’/>
32.     </message>
33.
34.     <message name=‘‘GetLastTradePriceOutput’’>
35.         <part name=‘‘body’’ element=‘‘xsd1:TradePrice’’/>
36.     </message>
37.
38.     <portType name=‘‘StockQuotePortType’’>
39.        <operation name=‘‘GetLastTradePrice’’>
40.            <input message=‘‘tns:GetLastTradePriceInput’’/>
41.            <output message=‘‘tns:GetLastTradePriceOutput’’/>
42.         </operation>
43.     </portType>
```

```
44.
45.     <binding name=‘‘StockQuoteSoapBinding’’
          type=‘‘tns:StockQuotePortType’’>
46.         <soap:binding style=‘‘document’’
              transport=‘‘http://schemas.xmlsoap.org/soap/http’’/>
47.         <operation name=‘‘GetLastTradePrice’’>
48.            <soap:operation
                 soapAction=‘‘http://example.com/GetLastTradePrice’’/>
49.            <input>
50.                <soap:body use=‘‘literal’’/>
51.            </input>
52.            <output>
53.                <soap:body use=‘‘literal’’/>
54.            </output>
55.         </operation>
56.     </binding>
57.
58.     <service name=‘‘StockQuoteService’’>
59.         <documentation>My first service</documentation>
60.         <port name=‘‘StockQuotePort’’ binding=‘‘tns:StockQuoteBinding’’>
61.            <soap:address location=‘‘http://example.com/stockquote’’/>
62.         </port>
63.     </service>
64.
65. </definitions>
```

3.1.6 UDDI

UDDI (Universal Description, Discovery and Integration) je na platformě nezávislý distribuovaný registr webových služeb založený na XML. UDDI poskytuje organizacím prostředí, kde mohou nabízet své webové služby a vyhledávát WS ostatních organizací. UDDI registr se skládá ze tří částí:

- **White Pages** – adresy, kontakty a obecné informace o organizacích nabízejících WS v registru.
- **Yellow Pages** – adresář služeb poskytovaných jednotlivými firmami, tyto služby jsou kategorizovány dle průmyslových standardů. Každá služba je popsána a odkazuje na šablonu s technickými údaji.
- **Green Pages** – technické informace o nabízených WS. Typicky se jedná o WSDL dokument s definicí rozhraní, kterou WS implementuje. Několik firem může nabízet

stejnou službu se stejným rozhraním, tyto služby jsou sjednoceny pod abstraktní službu a jsou popsány tzv. technickým modelem.

UDDI je jen málo využíváno softwarovými agenty pro dynamické navázání na služby, protože zatím není řešeno semantické hledisko WS. UDDI je tedy využíváno hlavně vývojáři, kteří po nalezení odpovídající služby získají z registru WSDL popis podle kterého naimplementují klienta. UDDI nebude v této práci využito a nebudeme se jím tedy podrobněji zabývat, bylo uvedeno jen pro ucelení přehledu technologií nad kterými jsou postaveny WS a SOA jako taková. Touto kapitolou jsme zakončili přehled technologií pro implementaci WS resp. SOA jako takové.

Kapitola 4

Návrh modulární platformy založené na WS

Předchozí kapitoly představili různé technologie spojené s distribuovaným zpracováním dat a webovými službami. Následující kapitoly představí návrh webové služby poskytující kontextové informace mobilním zařízením (KIM) a prototypu inverzního SOAP proxy serveru pro webové služby (ISPS). KIM na základě kontextu mobilního zařízení vybere z databáze relevantní informace, které se na tomto zařízení zobrazí. Kontext mobilního zařízení je dán veškerými informacemi o zažízením samotném (výrobce, vlastnosti) ale i stavu ve kterém se nachazí (poloha, čas, spuštěná aplikace). Smyslem ISPS je vytvořit jednu vstupní bránu k webovým službám organizace, která umožňuje jednoduchý monitoring a management nabízených WS. Návrh ISPS bude vycházet z požadavku na provoz v heterogenním prostředí a zaměří se tedy především na jeho škálovatelnost.

Tato práce je součástí rozsáhlého projektu platformy pro mobilní marketing a bude představovat tedy jen část celkového díla.

V první části kapitoly budou analyzovány požadavky na celé řešení, druhá část bude představovat samotný návrh celého řešení a jeho jednotlivých součástí, třetí část bude diskutovat některá bezpečnostní hlediska spojená s nasazením a provozem vyvinutého software.

4.1 Analýza požadavků

4.1.1 Modulární inverzní SOAP proxy server

ISPS pro webové služby bude v obecné rovině vstupní branou ke všem webovým službám dané organizace. To umožní centrálně kontrolovat a řídit využívaní nabízených služeb. Hlavní požadavek na ISPS bude vysoká modularita. V rámci této práce bude tedy navržena škálovatelná architektura složená z minimalistického jádra a modulů proxy serveru (dále jen PSM – proxy server modul). Jádro ISPS bude poskytovat jednoduché API (dále jen ISPS API) zpřístupňující obsah SOAP a odpovědí jednotlivým zaregistrovaným modulům proxy

serveru. Tyto moduly budou pak plnit specifické úkoly proxy serveru, kterými se zejména myslí:

- management publikovaných služeb,
- monitorování a statistiky,
- analýza obsahu,
- auditování,
- autorizace a autentizace,
- šifrovaní,
- load-balancing a failover,
- aplikace business pravidel,
- XML firewall,
- transakční cache.

V této práci vybereme pouze několik těchto funkcí a na základě nich budou implementovány jednoduché moduly proxy serveru, které otestují funkčnost jádra a odhalí jeho nedostatky. Pro otestovaní kompletního ISPS, tedy včetně nainstalovaných PSM bude vhodně využita KIM, jejíž návrh a implementace je součástí této diplomové práce. V rámci práce budou také provedeny zátěžové testy samotného ISPS i KIM. Výsledky těchto testů umožní odhalit slabá místa a navrhnout zlepšení.

PSM budou mít skrze API jádra proxy serveru přístup k zaregistrovaným webovým službám (které budou moci kdykoliv vyvolat nehledě na opravdový požadavek z Internetu), aktuálním požadavkům resp. odpovědím včetně jejich parametrů resp. návratových hodnot a službám ostatních zaregistrovaných PSM.

PSM budou moci jednotlivé části SOAP transakcí číst, upravovat, pozdržovat nebo je naopak pomocí zvýšení priority akcelerovat.

V následujících odstavcích si probereme podrobněji jednotlivé klíčové požadavky na ISPS.

Škálovatelnost

Vysoká modularita je jedním ze základních požadavků na platformu. Zajištěna bude pomocí jednoduchého jádra, které bude nabízet pouze minimální nutnou funkcionalitu. Toto jádro bude umět registrovat funkční moduly implementující jednotlivé funkce ISPS. PSM by měli mít přístup k veškerým informacím o WS, které ISPS zveřejňuje a aktuálních požadavcích na tyto WS. PSM budou mít možnost SOAP požadavky resp. odpovědi číst i měnit. V prototypu bude implementován pouze přístup k jednotlivým hodnotám parametrů

a odpovědí ale v dalším vývoji, který již je nad rámec této práce, se předpokládá přístup ke kompletnímu XML dokumentu, který tvoří SOAP dotaz resp. odpověď. To umožní širší možnosti práce s transakcemi.

Příkladem využití ISPS API může být PSM zajišťující protokolování příchozích požadavků resp. odpovědí. Takový modul potřebuje informaci o právě příchozím požadavku, kterou zpracuje a uloží, zároveň nevyžaduje blokování služby, protože nebude ovlivňovat obsah zprávy.

Druhým příkladem je load-balancer. Ten potřebuje stejně jako předchozí případ informaci o příchozím požadavku. Jeho hlavní úkol je ale rozhodnout kam bude daný požadavek přesměrován. Proto je nutné aby jeho zpracování blokovalo samotný požadavek na webovou službu. Rozhodnutí o cílovém serveru, který požadavek zpracuje je učiněno na základě load-balancing pravidel pro danou službu a aktuálního vytížení serveru, který ji zajišťuje.

Pro dosažení požadované škálovatelnosti a tedy zajištění možnosti provozu v heterogenním distribuovaném prostředí budou všechny funkce tvořící ISPS API implementovány jako webová služba nad protokolem SOAP. To umožní vytvářet moduly nejen v různých programovacích jazycích ale i na různých platformách.

Management

ISPS musí poskytovat dostatečnou funkčnost pro management publikovaných webových služeb i jednotlivých modulů proxy serveru, které využívá ke své činnosti. Takový management by měl pak poskytovat zejména následující funkčnost:

- registrace a odregistrace zveřejněných služeb (součást ISPS API)
- registrace a odregistrace modulů proxy serveru(součást ISPS API)
- konfigurace logování transakcí (PSM)
- omezení přístup k určitým webovým službám (PSM)
- parametry požadavků musí odpovídat daným pravidlům (PSM)
- transakční cache (PSM)
- apd.

Bezpečnost

SOAP proxy brána bude vstupní branou k webovým službám organizace a jejím datům, tedy k jejímu nejcenějšímu majetku. Návrh a implementace systému by měla poskytovat dostatečné bezpečnostní mechanismy k zajištění bezpečnosti SOAP brány jako takové i nabízených služeb. Bezpečnost služeb bude zajišťována jednotlivými PSM. Příkladem může

být modul zajišťující služby firewallu, monitorovací služby, auditovací služby, apd. Bezpečnostní funkce nejsou ale hlavní náplní této práce. Návrh a implementace jádra s nimi bude počítat a budou předmětem možného rozšíření systému.

Výkonnost

Výkonnost bude sledována zejména ve dvou rovinách. Jednak bude důležitá výkonnost samotné KIM, a pak jakým způsobem ovlivní výkonnost KIM její zpřístupnění přes ISPS.

U ISPS se vzhledem k rozsáhlosti použití protokolu SOAP musí provést výkonnostní test, které určí nejužší místa systému. Na základě tohoto pak budou navrhnuta opatření pro zvýšení výkonnosti v případné další práci na projektu. Obecně moduly, které pracují s aktualně vznesenými požadavky na služby by měli být blokující jen v nejnutnějším možném případě. Funkce, které tyto moduly implementují a způsobují zablokování požadavku na službu musí být krátké a efektivní a dělat jen naprosto nezbytnou činnost a zbytek odkládat na pozdější zpracování (z linuxového jádra je tato forma zpracování známa jako bottom-half).

Pro zvýšení rychlosti s jakou odpoví poskytované WS se nabízí implementace modulu pro dočasné ukládání výsledků transakcí (transakční cache). ISPS tím, že zpřístupní kompletní požadavky umožní tomuto modulu tyto požadavky shromažďovat a vyhodnocovat. Odpovědí na nejčastější případně nejnáročnější dotazy mohou být uchovávány a využity při opakování dotazu. Shromažďovaní je možné provádět v neblokujícím režimu. Vyhodnocení zda je daná odpověď aktuálně v cache je naopak nutné provést v blokujícím režimu, protože při úspěšném nahrazení se originální služba nevolá a do výsledku je uložen záznam z cache. Cache PSM je nad rámec této práce a je předmětem možného rozšíření.

4.1.2 WS poskytující kontextově závislé informace mobilním zařízením

Druhou částí serveru je webová služba poskytující kontextově závislé informace mobilním zařízením (KIM). Služba bude zveřejněna prostřednictvím SOAP protokolu. Pro zpřístupnění platformám na kterých není dostupná implementace protokolu SOAP bude možnost tuto službu využít i přes HTTP prokol, což bude zajištěno server-side skriptem umístěným na klasickém HTTP serveru. Tento skript bude transformovat parametry z URL do parametrů volání webové služby. Mezi platformy, které budou tuto WS využívat patří zejména Java2 Mobile Edition (J2ME), Flash, Symbian, Windows Mobile a jazyky pro programovaní server-side Internetových aplikací jako PHP, Perl apd. (zpřístupní službu WAPu a mobilnímu webu).

Integrace do aplikace bude provedena speciálním pluginem závislým na platformě na které aplikace běží. Tento plugin bude zobrazovat sdělení, která získá od KIM.

Aplikace na mobilním zařízení vznese pomocí zaintegrovaného pluginu požadavek na službu v rámci kterého předá KIM kontext ve kterém se nachází. KIM na základě kontextu vyhodnotí, které z uložených informací mají největší relevanci a vratí je mobilnímu zařízení.

V následujících odstavcích rozebereme jednotlivé aspekty této webové služby.

Kontext

Nejprve je nutné popsat v jakém kontextu se může mobilní zařízení nacházet. Každá informace bude mít přiřazenou sadu kontextových parametrů. Pokud těmto parametrům bude odpovídat kontext mobilního zařízení, které vzneslo dotaz, bude mu tato informace zaslána v odpovědi. Mobilní zařízení nezašle v dotazu všechny paramtery kontextu, ale pouze některé. Z parametrů, které zašle můžeme odvodit následující kontextové informace:

- **poskytovatel Internetové konektivity** může být mobilní operátor ale také provozovatel wi-fi sítě. Poskytovatele konektivity můžeme detekovat na základě IP adresy pod kterou se mobilní zařízení nachází. U mobilních operatorů je to často jedna Internetová brána, která zprostředkovává připojení pro všechny mobilní zařízení operatora, u wifi poskytovatelů to naopak může být i pevná IP pro každé zařízení zvlášt.
- **Výrobce zařízení** často určuje způsob ovládání a technické zvláštnosti.
- **Model** určí konkretní typ displeje, jeho rozměry, barevnou hloubku, množství paměti, schopnosti zařízení jako například přehravání hudby, integrovaný fotoaparát, apd.
- **Platforma** určuje v jakém formátu může resp. musí být daná informace zaslána. Wap umožňuje zobrazovat pouze textové a obrázkové informace. Naopak flash je schopný zobrazovat i animace a videa.
- **Identifikátor aplikace** určí druh informace. Například aplikace pro rezervaci zájezdů cestovních kanceláří bude informovat také o nabídce rekreačního oblečení.
- **Čas** určuje příležitost k jaké může být informace vztažena. Například zobrazení nabídky posledních vstupenek na divadelní představení.
- **Poloha** mobilního zařízení je vhodná pro zobrazení lokačně závislých dat. Například aplikace sloužící pro navigaci chce zobrazovat informace o obchodech v blízkosti, které ji poskytne třetí strana.
- **Sociodemografické údaje** jako například věk umožní vybrat informaci podle životního stádia uživatele (mladiství, střední věk, ...).
- **Klíčová slova** vyskytující se v aplikaci, která požádá o kontextové informaci přiblíží s jakými informacemi uživatel právě pracuje a umožní zobrazit podobné informace a tedy zaujmout uživatele, když se sám zajímá.

Příklad kontextu mobilního zařízení:

```
       výrobce zařízení: Sony Ericsson
                  model: K750i
              platforma: J2ME
 identifikátor aplikace: 4532
             denní doba: 7:00
                 poloha: 49.1913 lat, 16.6222 lon
sociodemografické údaje: žena, 15-20let
          klíčová slova: kino, program, film, multiplex
```

Mobilní zařízení nebude posílat kompletní kontextové informace ale pouze základ v podobě identifikace aplikace pomocí unikátního ID, HTTP user-agenta, zdrojové IP a polohy. Z těchto informací bude KIM schopna odvodit kompletní kontext tak jak byl popsán výše. V úvahu bude ale nutné brát, že jednotlivé platformy se mohou lišit v tom, jaké informace jsou schopné poskytnout. Například z WAPu jsme schopni získat user-agent a tedy parametry jako výrobce, model a vlastnosti ale nejsme schopni obdržet polohu. Naopak J2ME nám je schopna zaslat polohu, ale nezašle user-agent.

Logika pro výběr sdělení na základě kontextu

Logika pro výběr sdělení na základě kontextu je srdcem této webové služby. Tato část KIM bude rozhodovat, které informace jsou relevantní a zašle je v odpovědi mobilnímu zařízení.

Každá informace, která se má zobrazovat na mobilních zařízeních má skrz webový systém pro správu kontextových informací (dále jen CMS) přiřazený rozsah hodnot jednotlivých kontextových parametrů. Pokud parametry, které zašle mobilní zařízení v dotazu odpovídají (po transformaci na úplný kontext) hodnotám parametrů u dané informace je tato informace zařazena do množiny informací zobrazitelných na mobilním zařízení, které se nachází v daném kontextu.

Z množiny zobrazitelných informací je pak jedna až N informací náhodně vybráno. Počet informací je závislý na cílové platformě. Každá informace má navíc přiřazen určitý počet bodů (v CMS jsou označeny anglickým termínem bids), které mohou šanci na její zobrazení zvýšit přímo úměrně jejich počtu.

Statistiky

Kontextově závislé sdělení, které poskytuje KIM má podobu krátké textové nebo obrazové informace sloužící jako odkaz na další související informace v podobě wapových či webových stránek.

Každé zobrazení a případné následování odkazu (v textu dále označeno jako proklik – z anglického termínu click-through) je tedy nutné ukládat pro pozdější analýzy. U jednotlivých zobrazení chceme uchovat zejména následující informace:

- identifikaci sdělení,

- identifikaci aplikace na které byla zobrazena,
- proklik (ano/ne),
- počet pozitivních bodů (použitých jako váha při pseudonáhodném výběru),
- zdrojovou IP adresu,
- http user-agent,
- GPS souřadnice,
- čas zobrazení,
- čas prokliku pokud byl proveden.

Na základě těchto logů budou prováděny agregace podle jednotlivých položek. Vyniknou tak zajímavé informace typu nejčastější model mobilního telefonu, poměr proklik/zobrazení pro jednotlivé aplikace apd.

Webový správce

Webový správce umožňuje přehledně a uživatelsky příjemně manipulovat s daty potřebnými pro práci serveru. Obsahuje průvodce, kteří v několika krocích zaregistrují do systému novou informaci vztahující se k nějakému kontextu a nový mobilní obsah ve kterém se budou tyto informace zobrazovat. Správce je víceuživatelský s jednoduchou registrací nového uživatele. Webový správce není součástí této diplomové práce a je vyvíjen jiným členem týmu. Na obrázku 4.1 ukázán screenshot s částí, kde se konfigurují kontextové paramtery k dané informaci.

Pluginy do aplikací

Každá aplikace, která bude chtít zobrazovat kontextová data musí použít plugin připravený speciálně pro danou platformu. V případě zobrazování kontextových sdělení na WAPu se jedná o část kódu, který se vykoná na straně webového serveru při zobrazní daných WAPových stránek. Tento kód by měl být pro snadnou integraci napsaný ve stejném programovacím jazyce v jakém je vytvořen WAPový obsah v rámci, kterého bude sdělení zobrazováno. Na obr. 4.2 je screenshot z WAPového prohlížeče mobilního telefonu, který zobrazuje využití této webové služby pro zobrazení marketingových zpráv. V rámci této práce je nutné vytvořit plugin pro zobrazení kontextových marketingových sdělení ve WAPovém obsahu napsaném v jazyku PHP. Tento plugin bude zobrazovat odkaz na určité internetové stránky. Tento odkaz obdrží od serveru na základě kontextu ve kterém se nachazí. Plugin pro platformu Java2 Mobile Edition byl vytovřen v rámci [3].

Obrázek 4.1: Screenshot z CMS – část okna pro zadávání kontextových parametrů k určité informaci

Obrázek 4.2: Screenshot WAPu využívajícího KIM pro zobrazovaní marketingových sdělení

4.2 Návrh

4.2.1 Modularní inverzní SOAP proxy server

ISPS budou tvořit dva SOAP servery – jeden tzv. master server a obecně N tzv. slave serverů. Master server bude představovat jádro ISPS sloužící k registraci modulů proxy serveru a webových služeb, které má ISPS zveřejnit. Slave server bude zajišťovat samotné zveřejnění zaregistrovaných webových služeb a vyvolání triggerů, které zajistí zahájení činnosti jednotlivých modulů proxy serveru během konkrétní transakce. V následujícím textu bude představen návrh master a slave serveru a dvou jednoduchých modulů proxy serveru. Bude také představeno ISPS API pro registraci WS a PSM a nakonec návrh wrapperů WS a mechanismus spouštění triggerů modulů proxy serveru. Wrapperem webové služby se myslí stejnojmenná webová služba, která má stejné parametry a typ návratové hodnoty, ale kromě zavolání originální webové služby provede ještě další akce, jako například vyvolání triggerů modulů proxy serveru. Trigger je webová služba modulu proxy serveru, vyvolaná během SOAP transakce z wrapperu volané webové služby. Triggery umožní reakci jednotlivých modulů proxy serveru na SOAP transakce zprostředkované skrz ISPS.

Master a slave server

Master server bude poskytovat ISPS API sloužící k registrací/odregistraci modulů proxy serveru resp. webových služeb. Master server bude ve své podstatě minimalistické jádro ISPS, které bude ovládat slave servery. Slave servery budou nabízet registrované služby do prostředí Internetu. Nabízená služba bude mít uvnitř slave serveru podobu wrapperu, který kromě vyvolání originální WS vyvolá i triggery jednotlivých PSM plnících samotné funkce proxy serveru. Wrappery webových služeb a související triggery modulů proxy serveru budou rozebrány dále v textu. Na obrázku 4.3 je znázoněn návrh architektury ISPS. V rámci této práce bude implementován ISPS pracující pouze s jedním slave serverem.

ISPS API

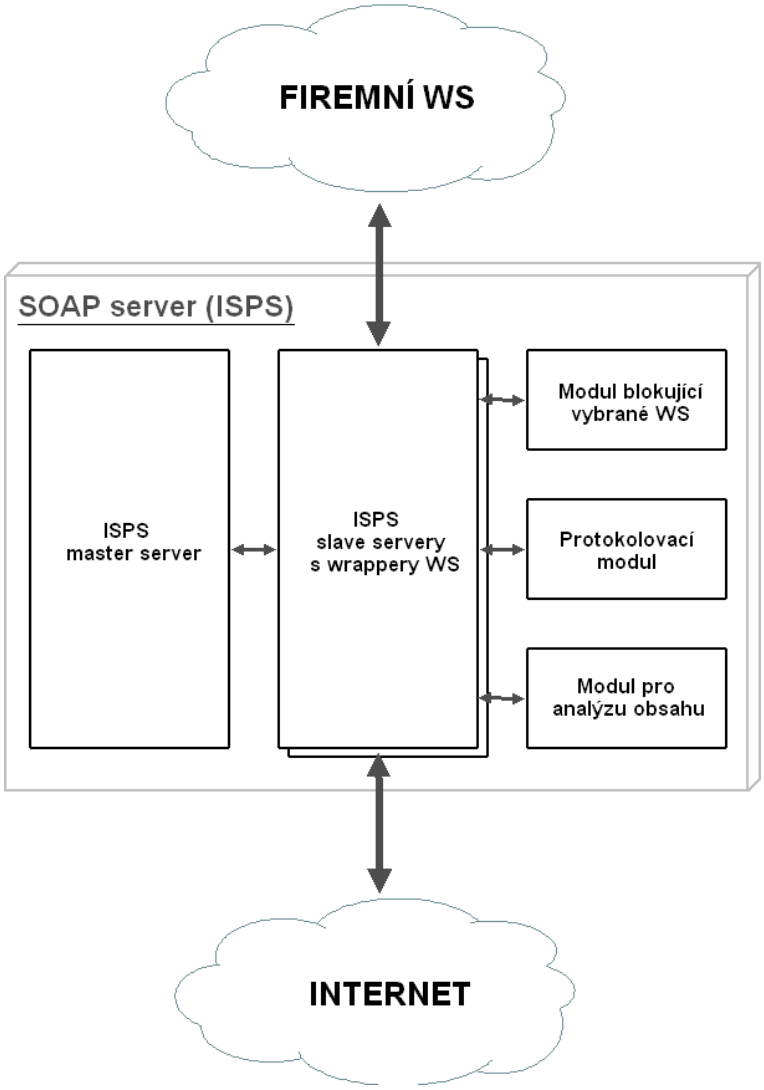
ISPS API jak již bylo uvedeno výše musí zajistit registraci/odregistraci modulů proxy serveru a webových služeb, které má server poskytovat. ISPS API bude sestávat z následujících čtyř funkcí jejichž význam výplývá z jejich názvů:

```
proxyServiceRegister(host, port, namespace, wsname, params)

proxyServiceUnregister(host, port, namespace, wsname)

proxyTriggerRegister(host, port, namespace, tgname, type)

proxyTriggerUnregister(host, port, namespace, tgname)
```



Obrázek 4.3: Koncept proxy serveru pro webové služby

Parametr `type` u funkce `proxyTriggerRegister()` je typu výčet s následujícími možnými hodnotami: `before-block, before-nonblock, after-block, after-nonblock`. Typ triggeru určí zda bude vyvolaný před či po spuštění služby a zda v blokujícím či neblokujícím režimu (viz další text).

Wrappery publikovaných webových služeb

Každé zaregistrované službě vytvoří master server stejnojmenný wrapper, který bude zveřejněn slave serverem a bude představovat jediný přístup k původní webové službě. Tento wrapper bude mít shodné parametry jako originální služba a bude se tedy volat stejným

způsobem pouze z jiného umístění. Koncept wrapperu nám umožní zajistit vyvolání jednotlivých modulů proxy serveru a tedy samotnou užitečnou činnost proxy serveru. Vyvolání PSM bude zajištěno spuštěním triggerů, které si jednotlivé PSM zaregistrovali. Jednotlivé triggery mohou být následujících typů:

- `before-block` je trigger spuštěný před zavoláním originální WS v blokujícím režimu. Může tedy změnit parametry se kterými byla WS zavolána. Měl by být efektivní, aby zbytečně nezpomaloval celou transakci.
- `before-nonblock` je trigger spuštěný před zavoláním originální WS v neblokujícím režimu. Nemůže tedy měnit parametry, protože bude spuštěný v odděleném vlákně.
- `after-block` je trigger spuštěný po zavoláním originální WS v blokujícím režimu. Může tedy změnit parametry a oproti typu `before-block` i odpověd původní WS. Opět by měl být efektivní, aby zbytečně nezpomaloval celou transakci.
- `after-nonblock` je trigger spuštěný po zavoláním originální WS v neblokujícím režimu. Nemůže tedy měnit parametry, protože bude spuštěný v odděleném vlákně. Má ale kromě parametrů k dispozici navíc i odpověď webové služby.

Parametry a návratové hodnoty jednotlivých triggerů musí odpovídat typu uvedeném při registraci. Neblokující triggery nevracejí žadnou užitečnou hodnotu naopak blokující vracejí případné upravené parametry resp. upravený výsledek. Přesný formát triggerů vypadá následovně:

```
before-block
  new_args trigger(host, port, namespace, wsname, args)

before-nonblock
  void trigger(host, port, namespace, wsname, args)

after-block
  result trigger(host, port, namespace, wsname, result, args)

after-nonblock
  void trigger(host, port, namespace, wsname, result, args)
```

Moduly proxy serveru

V rámci této práce budou implementovány dva ukázkové moduly proxy serveru, které ověří funkčnost ISPS. Tyto moduly budou sloužit k protokolování SOAP transakcí a dočasnému omezení přístupu k vybraným WS.

Protokolovací modul bude zapisovat do textového souboru záznámy o jednotlivých SOAP transakcích. Záznamy budou obsahovat informace o zavolání webové služby a o

jejím návratu. Bude z nich tedy možné určit dobu zpracování požadavku původní webovou službou. Jeden řádek bude odpovídat jednomu dotazu resp. odpovědi a bude obsahovat čas, jméno služby, hodnoty parametrů dotazu resp. odpovědi.

Namecheck modul zajistí odepření přístupu k webovým službám, které se určí v jeho konfiguračním souboru. Místo originální odpovědi pak zajistí tento modul zaslání odpovědi taktéž specifikované v konfiguračním souboru.

4.2.2 WS poskytující kontextově závislá sdělení mobilním zařízením

Po návrhu inverzního SOAP proxy serveru pro webové služby si ukážeme návrh webové služby, která ho jako první bude využívat. Na základě analýzy požadavků provedené výše představíme architekturu serveru poskytujícího tuto webovou službu, logiku pro výběr relevantních dat a způsob napojení na WURFL (Wireless Universal Resource File).

Získání úplného kontextu mobilního zařízení

Mobilní zařízení neposílá v dotazu kompletní informace o kontextu ale pouze jejich základ, ze kterého lze zbytek odvodit. Některé kontextové informace jako například čas zjistí server přímo. Jiné je nutné získat z různých datových zdrojů.

Sociodemografické údaje musí dodat provozovatel aplikace, která integruje plugin pro získavání kontextových dat ze serveru. Provozovatel většinou informuje o typické skupině uživatelů jeho aplikace.

Informace o poskytovateli připojení získá server podle adresy klienta. Informace o mobilním zařízení lze získat z databáze podle pole user-agent z hlavičky HTTP požadavku. Pokud HTTP požadavek tuto hlavičku neobsahuje musí být explicitně dodána aplikací. Na základě user-agent jsou pak získany informace jako výrobce, model, rozměry displeje, barevná hloubka, periferie zařízení jako například mp3 přehrávač, fotoaparát, apd. Zdrojem takových dat je například open source projekt WURFL. Jedná se o XML konfigurační soubor, který obsahuje informace o vlastnostech rozličných mobilních zařízení. V případě představované služby budeme tento soubor využívat k vyhledávání podle pole user-agent. Ukázka WURFL XML souboru (část popisu mobilního telefonu Siemens ME75):

```
<!-- ME75 -->
<device user_agent=‘‘SIE-ME75’’ actual_device_root=‘‘true’’
                      fall_back=‘‘upgui_generic’’ id=‘‘sie_me75_ver1’’>

    <group id=‘‘product_info’’>
        <capability name=‘‘brand_name’’ value=‘‘Siemens’’/>
        <capability name=‘‘model_name’’ value=‘‘ME75’’/>
    </group>

    <group id=‘‘display’’>
```

```
        <capability name=‘‘resolution_width’’ value=‘‘132’’/>
        <capability name=‘‘resolution_height’’ value=‘‘176’’/>
        <capability name=‘‘max_image_width’’ value=‘‘132’’/>
        <capability name=‘‘max_image_height’’ value=‘‘176’’/>
    </group>

    <group id=‘‘image_format’’>
        <capability name=‘‘bmp’’ value=‘‘true’’/>
        <capability name=‘‘jpg’’ value=‘‘true’’/>
        <capability name=‘‘gif’’ value=‘‘true’’/>
        <capability name=‘‘png’’ value=‘‘true’’/>
        <capability name=‘‘colors’’ value=‘‘262144’’/>
    </group>
    ...
```

Návrh architektury webové služby

Webová služba se bude skládat z několika modulů zajišťujících jednotlivé činnosti nutné pro její provoz. Těmito moduly budou:

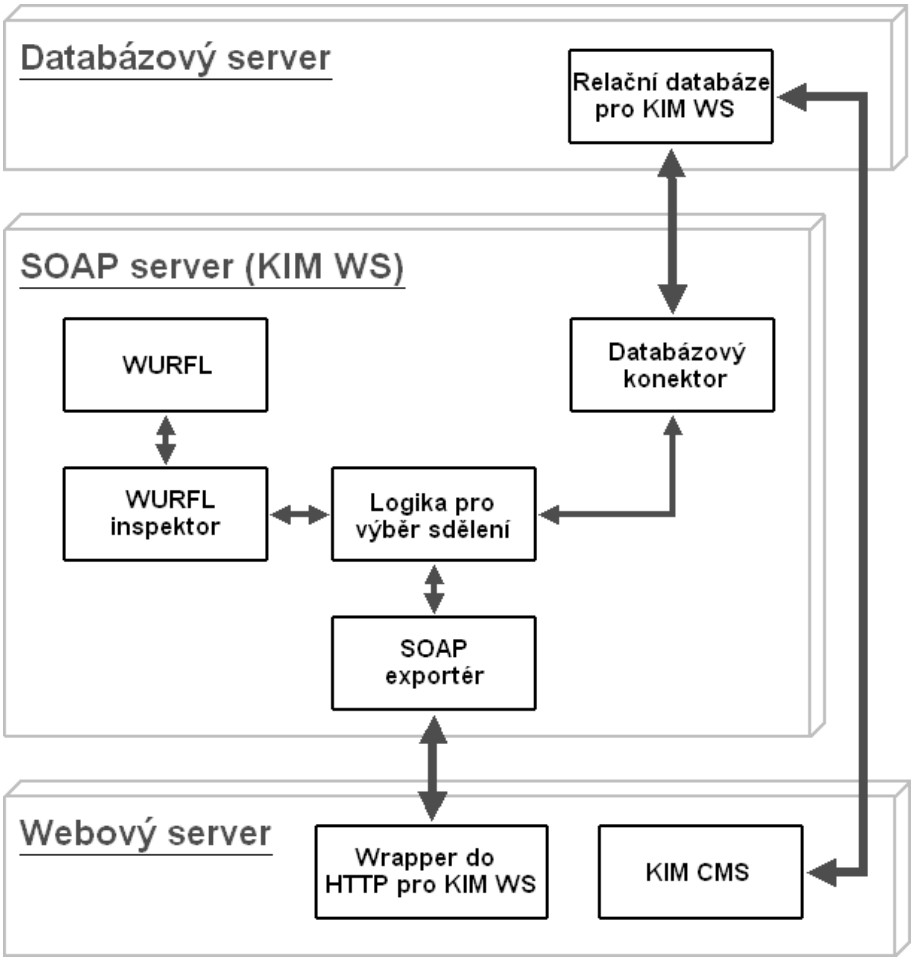
- databázový konektor,
- WURFL inspektor,
- logika pro výběr sdělení,
- SOAP exportér.

Na obrázku 4.4 je znázorněna struktura celé webové služby. Nyní si podrobněji popíšeme činnost jednotlivých modulů.

Databázový konektor bude tvořit vrstvu mezi webovou službou a daty uloženými v databázi. Tím zajistíme odstínění od konkretního databázového serveru. API databázového konektoru bude tedy poskytovat jediné možné rozhraní pro přístup k databazi. Jednotlivé funkce databázového API budou umožňovat získat, měnit, rušit a přidávat data v databázi. Příklad prototypů funkcí API databázového konektoru:

```
## vrací množinu sdělení, určených pro zařízení daného výrobce
set getAdsByManuf(manuf)

## uloží záznam o zobrazení sdělení na mobilním zařízení
void insertShowLog(ad, site, locat, platf, manuf, chann, bid, ua, ip)
```



Obrázek 4.4: Struktura webové služby

Tyto funkce budou jednoduché a budou vždy zajišťovat pouze manipulaci s data v databázi. Veškerá logika bude implementována v jiném modulu. Jelikož volání těchto funkcí bude na jedno zavolání webové služby řádově desítky a volání samotné webové služby nekolik za sekundu je nutné aby se přístup k databázi využíval jednoho permanentního TCP spojení na databázový server.

WURFL konektor zpřístupní jednoduché API pro získání dat z WURFL XML souboru (podrobněji jsme se o WURFL zmiňovali výše).

Logika pro výběr sdělení na základě kontextu je srdcem celé aplikace. Na základě kontextových informací z požadavku obdrží z WURFL a databázového konektoru úplný kontext dotazu dle specifikace uvedené výše.

Algoritmus jakým se rozhodne jaké sdělení zobrazit danému mobilnímu zařízení, které se nachazí v jistém kontextu bude vypadat následovně:

1. krok: vyfiltruj sdělení odpovídající danému kontextu
2. krok: přiřad jednotlivým sdělením pozitivní body pro losování
3. krok: vylosuj vítězná sdělení, která se nakonec zobrazí
4. krok: připrav konkretní formát v jakém bude sdělení zasláno na mobilní zařízení
5. krok: zaloguj zobrazení

Algoritmus samotného filtrování bude postupně procházet jednotlivé parametry kontextu a vytvářet množiny odpovídajích sdělení pro každy parametr. Průnik těchto množin jsou pak sdělení zobrazitelná na daném zařízení.

SOAP exporter zajistí zpřístupnění API webové služby přes SOAP protokol. API bude popsáno v samostatné podkapitole níže.

HTTP wrapper slouží pouze k obalení webové služby poskytnuté do protokolu HTTP, který nám v tomto případě stačí a je nejjednodušší pro implementaci na straně poskytovatelů obsahu.

API webové služby

Navrhovaná webová služba bude mít API tvořené dvěmi funkcemi s následujícím prototypem:

```
string getShowContent( id, lat, lon, ua, ip, rua, rip )

string getClickedContent( impId )
```

Funkce `getShowContent` obdrží v parametrech kontext mobilního zařízení daný GPS koordináty (lat, lon), polem user-agent a IP adresou prohlížeče (ua, ip) a polem user-agent a ip adresou serveru (rua, rip). Dle kontextu vybere relevantní sdělení, které vratí ve formátu daném cílovou platformou. Implementace bude počítat s formátem pro WAP (text, obrázek) a Java2 Mobile Edition (obrázek) a pluginem pro platformu Apache/PHP (WAP). Plugin pro J2ME je vytvořen v rámci [3].

Funkce `getClickedContent` poslouží u sdělení, které jsou zároveň odkazem na WAPové stránky (v našem případě všechny uvažované případy). Odkaz v daném sdělení bude směrován na html stránku, která zavolá `getClickedContent()` a zaznamená tak proklik, poté vratí v HTTP hlavičce redirect na originální lokaci. Jediným parametrem je impID, které poslouží k vyhledání v tabulce zobrazení. Tabulka zobrazení obsahuje kromě jiných infromací i id sdělení, podle kterého je vyhledána cílová URL.

Příklad formátu textového sdělení, které je odkazem na WAPovou stránku se zaznamenáním prokliku:

`Titulek sdělení###http://www.adplaze.com/r/?i=X`

Znaky `###` zde hrají roli oddělovače. Plugin, který toto zpracovává na straně poskytovatelů obsahu musí tento řetězec analyzovat a sestavit příslušný html odkaz. Nabízí se poskytovat přímo html tag `<a href=...>`, ale tím bychom omezovali poskytovatele obsahu.

Webový správce

Webový systém pro správu kontextových dat není součástí této práce. V rámci projektu byl navržen a implementován jiným členem týmu.

4.2.3 Napojení webové služby na SOAP proxy server

Na obrázku 4.5 je struktura kompletního systému, kterým se zabývá tato práce. SOAP webová služba je překryta proxy serverem, který bude její činnost nezávisle monitorovat.

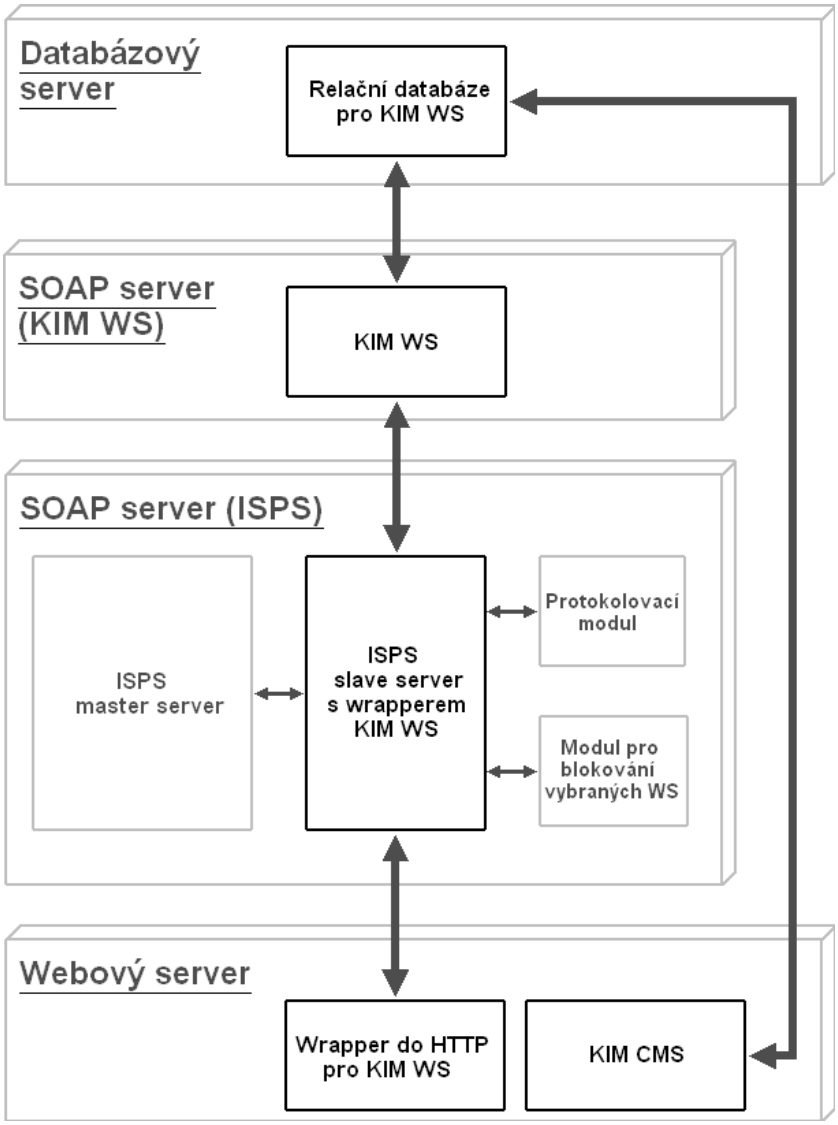
4.3 Bezpečnost

Tato kapitola stručně nastíní jednotlivá hlediska zabezpečení platformy představené v této práci. Nahlíží na platformu jako na celek včetně částí, které nebyly součástí této diplomové práce (napojení na bankovní systém, webový správce). V následujícím textu budou představeny bezpečné domény, modely útočníka a definice citlivých dat

4.3.1 Definice bezpečných domén

Celé řešení sestává ze síťovych aplikací komunikujících standardními Internetovými protokoly nad TCP/IP sití. Celý systém se skládá z následujících částí:

- hardware
  - množina serverů umístěných v zabezpečeném prostředí a klientská PC a notebooky pro vzdálenou správu,
  - množina externích serverů a klientských zařízení (PC, mobilní telefony), které přistupují ke službám systému z nezabezpečeného prostředí. Obecně jsou tyto subjekty zdrojem bezpečnostního rizika.
- software – množina aplikací zajišťujících provoz a správu služeb
- uživatelé
  - pracovníci firmy – přistupující k servisním funkcím,
  - pracovnící zákazníků - přistupující k webovému správci,
  - pracovníci partnerských firem.



Obrázek 4.5: Úplný návrh architektury

V systému je možné identifikovat dvě bezpečné domény:

1. Servery: fyzicky umístěny v zabezpečené místnosti u ISP. Přístup do místnosti se servery mají administrátoři serveru a pracovníci ISP.

2. Klientská PC a notebooky: PC umístěny v zabezpečených kancelářích, notebooky jsou pod zodpovědností pracovníka, který je má v užívání.

4.3.2 Definice modelu útočníka

Rozdělení útočníka podle prostředí

- konkurenční firma – motivace: konkurenční boj

- partnerské firmy – motivace: zvýšení příjmů umělým generovaním falešných manipulací s kontextovými sděleními
- uživatelé služeb partnerů – motivace: poškození zadavatele sdělení generovaním falešných manipulací s kontextovými sděleními
- náhodný hacker – motivace: požitek z prolamování zabezpečení cizích systémů

Rozdělení útočníka podle komunikačních protokolů

Platforma poskytuje služby pro externí subjekty prostřednictvím protokolů HTTP, HTTPS a SOAP. Interně systém využívá protokoly SQL, SSH a SOAP. Pro identifikaci modelů útočníků vytvoříme 3 skupiny podle veřejně přístupných komunikačních protokolů:

- HTTP
  - uživatelé webového rozhraní
  - uživatelé webových služeb
  - náhodný hacker
- HTTPS
  - uživatelé webového rozhraní
  - náhodný hacker
- SOAP
  - uživatelé webových služeb
  - náhodný hacker

4.3.3 Definice citlivých dat

Platforma pracuje s následujícími citlivými daty:

Domácí data

Důvěrná data organizace, se kterými pracuje tento systém jsou zejména přístup na bankovní účet (bezpečnostní certifikát, klientské číslo, heslo) a osobní údaje o pracovnících firmy. Přístup na bankovní účet je přenášen na servery banky (eBanka) zabezpečeným protokolem HTTPS. Osobní údaje o pracovnících firmy neopouští bezpečnou doménu žadným způsobem.

Data partnerů

Důvěrná data partnerských organizaci – poskytovatelů prostoru pro kontextová sdělení – jsou zejména osobní informace, čísla bankovních účtů, historie příjmů a identifikace prostorů poskytnutých pro umísťování kontextových sdělení. Jsou přístupná pouze přes protokol HTTPS vždy jen partnerské firmě, které patří. Partnerská firma by měla přistupovat k těmto informací ze svých bezpečných domén. Kompromitace účtu jedné partnerské firmy nemá vliv na data ostatních uživatelů.

Data zadavatelů

Mezi data zadavatelů patří zejména osobní data, čísla bankovních účtů, historie výdajů za umisťování kontextových sdělení, historie práce se systémem a obsah kontextových sdělení připravovaných informačních kampaní. Tato data jsou přístupná skrze protokol HTTPS vždy jen zadavateli, kterému patří. Zadavatel by měl přistupovat k těmto informacím ze svých bezpečných domén. Kompromitace účtu jednoho zadavatele nemá vliv na data ostatních uživatelů. Veškerá citlivá data jsou uložena na serverech v rámci bezpečné domény.

4.3.4 Návrh bezpečnostních mechanismů

Tato kapitola formálně popisuje některé možné bezpečnostní incidenty, současný stav ochrany proti nim a mechanismy jakými se na daný incident bude reagovat. Pro každý uváděný incident je připravena tabulka popisující současný stav zabezpečení dané části systému, ochranu před uvažovaným incidentem a reakci jaká nastane pokud se incident vyskytne. Tabulky 4.1 a 4.2 se zabývají dvěma fyzickými incidenty a to ztrátou serveru způsobenou například krádeží či živelnou pohromou a případnou ztrátou kontektivity do Internetu. Tabulky 4.3, 4.4 a 4.5 se zabývají útoky na vybrané provozované služby.

Ztráta serverů (krádež, živelná pohroma)	
Současný stav:	Servery jsou umístěny u ISP (Internet Sevice Provider). Budova je opatřena poplašným zařízením a vzdáleným 24/7 dozorem bezpečnostní agentury. Servery nemají fyzický 24/7 dohled. Neredundantní umístění serverů je v současné době jedna z bezpečnostních slabin platformy.
Ochrana:	Zálohování dat na klientská PC umístěná v kancelářích v jiném městě.
Reakce:	Nahlášení krádeže na policii, obnova dat ze záloh na klientských PC, instalace nových serverů.
Vylepšení:	Řešení tohoto problému bude, v případě komerční úspěšnosti projektu, realizováno zajištěním druhého redundantního serveru u jiného ISP.

Tabulka 4.1: Fyzické incidenty: ztráta serverů

Ztráta konektivity do Internetu	
Současný stav:	Připojení k Internetu zajišťuje pouze jeden ISP, jedná se optické připojení do sítě GTS Novera.
Ochrana:	V současné době není možná.
Reakce:	Při dlouhodobém výpadku je možné přemístit server k partnerskému poskytovateli Internetové konektivity.
Vylepšení:	Záložní připojení k Internetu zajištěné jiným poskytovatelem.

Tabulka 4.2: Fyzické incidenty: ztráta kontektivity do Internetu

Webové služby	
Současný stav:	Webové služby zajišťuje v současné době SOAP server Webrick a HTTP server Apache pro jednodušší implementaci klienta na některých platformách. Tyto služby byly vyvinuty speciálně pro tuto platformu a při jejich vývoji byli dodržovány obecné zásady psaní bezpečného kódu s přihlednutím k specifikům jazyka Ruby a jeho interpretru. Kód neprošel žádným interním ani externím bezpečnostním auditem.
Ochrana:	**DoS (Denial of Service):** Firewall na serveru, kde je umístěná služba, další dedikovaný firewall mezi serverem a Internetovou branou. **Spušťění cizího kódu:** V současné době ochrana pouze na úrovni omezených práv uživatele, pod kterým je spuštěn Webrick a Apache server.
Reakce:	**DoS** bude řešen automatizovanou detekcí na firewallu a při překročení nastaveného prahu množství TCP spojení na službu bude zablokován provoz z dané IP. **Spušťění cizího kódu:** Identifikace a oprava zneužité chyby, audit serveru z pohledu uživatele pod kterým běžela zneužitá služba.
Vylepšení:	Pokud bude spuštěn ostrý provoz, počítá se z upgradem HW, to umožní zpřísnit pravidla kontroly provádění server-side skriptů (mod_security), která jsou velmi náročná na zdroje, dále bude proveden interní audit kódu (později i externí audit), výměnna standardního distribučního linuxového jádra za jádro obohacené o bezpečnostní rozšíření (Linux Intrusion Detection System, Medusa, apd.).

Tabulka 4.3: Útok na služby: webové služby

Vzdálená správa	
Současný stav:	Na serveru běží aktuální distribuční verze OpenSSH, přístup na ni je firewallem omezen jen na vybrané IP.
Ochrana:	**DoS:** Firewall na serveru, kde je umístěná služba, další dedikovaný firewall mezi serverem a Internetovou branou. **Spušťění cizího kódu:** Pravidelné aktualizace. **Útok hrubou silou:** Omezení počtu pokusů o vložení hesla.
Reakce:	**DoS** bude řešen automatizovanou detekcí na firewallu a při překročení nastaveného prahu množství TCP spojení na službu bude zablokován provoz z dané IP. **Spušťění cizího kódu** by způsobilo kompromitaci serveru a nutnost instalace nového serveru s daty obnovenými ze záloh, pokud by se jednalo o zneužití známé a dosud neopravené chyby byla by služba spuštěna až po jejím opravení. **Útok hrubou silou** a následná kompromitace daného uživatele by vyvolala audit, který by měl odpovědět na otázku zda nedošlo k úplné kompromitaci serveru a určit tak kroky nezbytné k obnovení služeb.
Vylepšení:	Přihlašování pomocí bezpečnostního klíče uloženého na bezpečnostním modulu (USB klíčenka) či pomocí biometrického modulu.

Tabulka 4.4: Útok na služby: vzdálená správa

Webová aplikace	
Současný stav:	Webová aplikace byla speciálně vyvinuta přímo pro platformu, je postavena na technologií Apache/PHP a tak je obecně nachylná k bezpečnostním problémům této technologie. Při vývoji byli dodržovány zasády tvorby bezpečného kódu s přihlednutím ke specifickým problémům PHP. Kód neprošel žádným interním ani externím bezpečnostním auditem. Na serveru je nainstalován stavový firewall (iptables). Server je umístěn v DMZ za firewallem (Cisco).
Ochrana:	**DoS:** Firewall na serveru, kde je umístěná služba, další dedikovaný firewall mezi serverem a Internetovou branou. **Spušťění cizího kódu** je omezeno pomocí modulu mod_security na straně Apache.
Reakce:	**DoS** bude řešen automatizovanou detekcí na firewallu a při překročení nastaveného prahu množství TCP spojení na službu bude zablokován provoz z dané IP. **Spušťění cizího kódu:** Identifikace a oprava zneužité chyby, audit serveru z pohledu uživatele pod kterým běžela zneužitá služba.
Vylepšení:	Pokud bude spuštěn ostrý provoz, počítá se z upgradem HW, to umožní zpřísnit pravidla kontroly provádění PHP (mod_security), která jsou velmi náročná na zdroje, dále bude proveden interní audit kódu (později i externí audit), výměnna standardního distribučního linuxového jádra za jádro obohacené o bezpečnostní rozšíření (Linux Intrusion Detection System, Medusa, apd.).

Tabulka 4.5: Útok na služby: webový správce

Kapitola 5

Implementace

V následující kapitole bude probrána implementace obou součástí vyvíjené platformy. Nejprve bude představena implementace škálovatelného inverzního SOAP proxy serveru a následně i webové služby poskytující kontextové informace mobilním zařízením.

5.1 Modulární inverzní SOAP proxy server

Na základě návrhu modulárního inverzního SOAP proxy serveru, který byl představen v předchozím textu byla vytvořena implementace prototypu, kterou stručně vysvětlí tato kapitola. Tabulka 5.1 popisuje strukturu zdrojových souborů. Konkretní způsob implementace jednotlivých součástí ISPS budou probírat následující kapitoly.

Soubor	Obsah
sp_common.rb	pomocné funkce pro stavové výpisy serveru apd.
conf/sp_config.rb	konfigurační proměnné proxy serveru
proxyserver	shell skript pro start a ukončení běhu ISPS
run/soapProxyServer.pid	PID procesu serveru
sp_server.rb	třída spouštějící master proxy server jako SOAP server
sp_master.rb	master proxy server ovladajicí slave server
sp_slaves.rb	slave server zajišťující samotná spojení na registrované WS
mods/logger/	soubory modulu zajišťujícího protokolování dotazů a odpovědí na jednotlivé webové služby
mods/namecheck/	soubory modulu zajišťujícího blokování vybraných webových služeb
services/	adresář s wrappery jednotlivých webových služeb
triggers/	adresář s triggery jednotlivých modulů proxy serveru

Tabulka 5.1: Obsah jednotlivých souborů SOAP proxy serveru

5.1.1 Master server

Master server hraje v ISPS roli konfiguračního rozhraní. V rámci ISPS tvoří za jeho běhu samostatný proces, který poskytuje ISPS API ve formě webové služby nad protokolem SOAP. Zajišťuje tedy registraci resp. odregistraci jednotlivých webových služeb a triggerů modulů proxy serveru. Zároveň dle potřeby vytváří a ukončuje proces slave serveru, o kterém bude více uvedeno dále v textu.

Master server je implementovaný třídami `SP_ProxyMaster` a `SP_SOAPProxyMaster`, které jsou umístěné v souboru `sp_master.rb`.

Třída `SP_SOAPProxyMaster` zajišťuje pouze zpřístupnění ISPS API pomocí protokolu SOAP. Tato třída dědí třídu `SOAP::RPC::StandaloneServer` z knihovny soap4r a implementuje metodu `on_init()`, která je vyvolána při vytváření nové instance. Dále obsahuje metodu `slaveShutdown`, která se, jak již název napovídá, používá pro ukončení procesu slave serveru. `on_init()` pouze vytvoří nový objekt třídy `SP_ProxyMaster` a zaregistruje její metody tvořící ISPS API. Jak již bylo uvedeno v návrhu, ISPS API tvoří následující čtyři metody:

```
proxyServiceRegister(host, port, namespace, wsname, params)

proxyServiceUnregister(host, port, namespace, wsname)

proxyTriggerRegister(host, port, namespace, tgname, type)

proxyTriggerUnregister(host, port, namespace, tgname)
```

Metody `proxyServiceRegister` resp. `proxyServiceUnregister` slouží k registraci resp. odregistraci webových služeb, které má ISPS zpřístupňovat. Parametry registrační metody jsou adresa a port serveru poskytujícího danou WS (parametry `host` a `port`), jmenný prostor (`namespace`), jméno webové služby (`wsname`) a její parametry ve formě pole (proměnná `params` typu `SOAP::Array`).

Metody `proxyTriggerRegister` resp. `proxyTriggerUnregister` slouží k registraci resp. odregistraci triggerů modulů proxy serveru. Podobně jako předchozí metody pro (od)registraci WS jsou jejími parametry adresa, port, jmenný prostor a jméno WS, která zajišťuje určitou funkci proxy serveru. Metoda `proxyTriggerRegister` má jako poslední parametr typ triggeru (`type`) s možnými hodnotami:

`'before-block', 'before-noblock', 'ater-block', 'after-noblock'.`

Typ triggeru určí jaké parametry musí WS mít a jaké bude vracet hodnoty. Všechny čtyři metody pouze zavolají odpovídající metody objektu slave serveru, který je třídy `SP_SOAPProxySlave` a je reprezentován instanční proměnnou `@slaveServer`. Následně provedou restart jeho procesu. Instance, nad kterou je vytvořen nový proces slave serveru tak již obsahuje nově zaregistrované WS resp. triggery.

Více o slave serveru a formě jakou se dynamicky registrují nové metody jeho instance budou uvedeny v následující podkapitole.

5.1.2 Slave server

Slave server hraje v ISPS roli samotného inverzního proxy serveru. Poskytuje tedy všechny registrované WS jako stejnojmenné WS v rámci jednoho svého jmenného prostoru a na jednom portu. Implementace prototypu oproti návrhu uvažuje pouze jeden možný slave server umístěný na stejném počítači jako master server.

Jak bylo uvedeno v předchozí kapitole master server (od)registruje WS a triggery modulů proxy serveru pomocí odpovídajích metod slave serveru. Slave server je implementovaný třídami `SP_SOAPProxySlave` a `SP_ProxySlave` umístěnými v souboru `sp_slave.rb`. Třída `SP_SOAPProxySlave` dědí třídu `SOAP::RPC::StandaloneServer` z knihovny soap4r a stejně jako master server implementuje metodu `on_init()`, která je vyvolána při vytváření nové instance slave serveru. Tato metoda vytvoří objekt třídy `SP_ProxySlave` a zavolá své metody `registerStoredServices` a `registerStoredTriggers`.

Metoda `registerStoredServices` zaregistruje na server wrappery registrovaných WS. Tyto wrappery jsou uloženy v adresáři `services` v podobě zdrojových souborů v jazyku Ruby. Obdobně metoda `registerStoredTriggers` zaregistruje uložené triggery modulů proxy serveru, které jsou v adresáři `triggers`.

Vytvoření wrapperů pro WS, které mají být zpřístupněny přes ISPS zajistí metoda `registerService` s parametry `host`, `port`, `namespace`, `wsname` a `params`, která patří do třídy `SP_SOAPProxySlave`. Tato metoda dynamicky vytvoří a zadefinuje ve své instanci novou metodu pojmenovanou stějně jako registrovaná WS a se stejnými parametry. Nově zadefinovaná metoda je pak wrapper původní webové služby a je vytvořena na základě následující šablony:

```
def @slave.#{wsname}(#{strParams})
  @execParams['allow'] = true

  # make new thread and execute non-blocking triggers
  execBeforeNonblockingTriggers('#{host}', '#{port}', '#{namespace}',
    '#{wsname}', #{strParams})

  # execute blocking triggers chain
  execBeforeBlockingTriggers('#{host}', '#{port}', '#{namespace}',
    '#{wsname}', #{strParams})

  # call of original web service
  if @execParams['allow'] == true
    url=``http://#{host}:#{port}''
```

```
    srv = SOAP::RPC::Driver.new( url, '#{namespace}' )
    srv.add_method('#{wsname}' #{strParamsAM})
    @execParams['result'] = srv.#{wsname}(#{strParams})
  else
    prn ``Original WS (#{wsname}) not called!''
    @execParams['result'] = 0
  end

  # make new thread and execute after-nonblock triggers
  execAfterNonblockingTriggers('#{host}', '#{port}', '#{namespace}',
    '#{wsname}', @execParams['result'], #{strParams})

  # execute after-block triggers chain
  execAfterBlockingTriggers('#{host}', '#{port}', '#{namespace}',
    '#{wsname}', @execParams['result'], #{strParams})

  return @execParams['result']
end
add_method(@slave, \``#{wsname}\'' #{strParamsAM})
```

V této šabloně se nahradí proměnné uzavřené v `#{...}` za skutečné hodnoty. Tyto proměnné jsou parametry metody `registerService()` a byli vysvětleny již dříve. Důležitou proměnnou je předdefinované instanční asociativní pole `@execParams` s následující definicí:

```
@execParams = {
    'allow' => true,
    'params' => [],
    'result' => nil
}
```

Tato proměnná se používá ve funkcích spouštějící jednotlivé triggery a obsahuje parametry WS, návratovou hodnotu a příznak zda se WS bude volat či nikoliv (`'allow'`). Parametry a navratovou hodnotu mohou průběžně jednotlivé moduly proxy serveru upravovat.

Podobným způsobem jako jsou vytvářeny wrappery, vytvoří ISPS dynamicky i triggery modulů proxy serveru. Úkolem těchto triggerů je v době volání originální WS zpřístupněné přes ISPS, zavolat WS registrovaných modulů proxy serveru plnící určitou funkci ISPS. WS modulů proxy serveru mají předepsané parametry a návratové hodnoty. Jejich parametry jsou adresa, port, jmenný prostor, parametry a případně i návratová hodnota (pokud se jedná o "after" triggery) volané WS. Zaregistrované triggery jsou uložené v adresáři `triggers`. Na základě typu jsou pak jednotlivé triggery ukládány do odpovídajích instančních proměnných typu pole:

```
@beforeBlock, @beforeNonblock, @afterBlock, @afterNonblock
```

.

Tato pole jsou při zavolání WS v jejím wrapperu sekvenčně procházena a postupně jsou vyvolávány jednotlivé triggery v nich uložené.

5.1.3 Moduly proxy serveru

V rámci této práce byly také vytvořeny dva moduly proxy serveru v podobě webové služby nad protokolem SOAP, které budou použity zejména k otestování ISPS API a funkce wrapperů a triggerů. Modul pro protokolování SOAP transakcí otestuje implementaci neblokujících triggerů, modul pro blokování vybraných webových služeb otestuje funkci blokujících triggerů.

Modul pro protokolování SOAP transakcí

Modul pro protokolování SOAP transakcí je implementován v adresáři `mods/logger/`. Tento modul je samostatná webová služba poskytující dvě metody:

```
logSOAPCallStart(host,port,namespace,wsname,params)
```

```
logSOAPCallStop(host,port,namespace,wsname,result,params)
```

První metoda uloží do textového souboru s protokolem záznam s časem, jménem, adresou, portem a parametry volané webové služby. Druhá metoda uloží čas a výsledek po jejím návratu.

Modul pro blokování vybraných webových služeb

Modul pro blokování webových služeb je implementován v `mods/namecheck/`. Tento modul znepřístupní vyjmenované webové služby a vratí implicitní hodnotu namísto skutečné. Tato hodnota bude uvedena v konfiguračním souboru `mods/namecheck/nc_deny` vedle jména blokované služby. Funkčnost zajistí tyto dvě metody:

```
checkService(host,port,namespace,wsname,params)
```

```
createResult(host,port,namespace,wsname,result,params)
```

První z metod vrátí booleovskou hodnotu říkající zda se má či nemá originální metoda zavolat. Pokud se originální metoda nevolá vrací druhá jmenovaná metoda implicitní výsledek, který načtě v konfiguračním souboru, jinak vrací originální výsledek.

Tímto byla stručně představena implementace ISPS. Následující kapitola představuje implementaci WS pokytující kontextové informace mobilním zařízením.

5.2 WS poskytující kontextové informace mobilním zařízením

V této kapitole bude popsána implementace webové služby představné v návrhu. Tabulka 5.2 popisuje strukturu zdrojových souborů. Implementaci jednotlivých součástí webové služby probereme v následujících kapitolách.

Soubor	Obsah
ap_common.rb	pomocné funkce pro stavové výpisy serveru apd.
conf/ap_config.rb	konfigurační proměnné webové služby
ap_server.rb	server poskytující webovou službu nad protokolem SOAP
proxyserver	shell skript pro start a ukončení běhu KIM
run/ap_soapServer.pid	PID procesu serveru
ap_addirector.rb	logika pro výběr sdělení na základě kontextu
ap_mysql.rb	konektor k MySQL databázi
log/adserver.log	záznamy z provozu serveru
ap_wurfl.rb	konektor k WURFL databázi
wurfl/wurflhandset.rb	inspektor pro práci s WURFL daty
wurfl/wurfl.db	samotná WURFL databáze s technickými detaily jednotlivých mobilních zařízení
html/r.php	přesměrování na stránku z originálním obsahem, zaprotokolování prokliku
html/a.rbx	HTTP wrapper pro webovou službu

Tabulka 5.2: Obsah jednotlivých souborů webové služby

5.2.1 Logika pro výběr sdělení

Logika pro výběr sdělení je implementována v souboru `server/ap_addirector.rb`. Základem implementace je třída `ApDirector`. Tato třída poskytuje dvě veřejné metody tvořící API webové služby.

```
getShowContent(id, lat, lon, ua, ip, rua, rip)

getClickedContent(impId)
```

Metoda `getShowContent(id, lat, lon, ua, ip, rua, rip)` vybere na základě kontextových informací, které obdrží jako parametry jedno nebo více (záleží na platformě) sdělení. Tato sdělení pak ve speciálně formátovaném řetězci vrátí jako výsledek.

Celý mechanismus je rozdělen do tří kroků. Každý krok je reprezentován voláním jedné metody zajišťující danou činnost.

Jako první se volá metoda `selectMsgs(id, lat, lon, ua, ip)`, které jsou předány vstupní parametry veřejné metody `getShowContent()`. `selectMsgs()` nejprve získá z databáze podle jednotlivých parametrů postupně: kanál (`channel`) a typ mobilního obsahu

(`medium`) podle id obsahu, výrobce (`manuf`) a vlastnosti (`caps`) mobilního zařízení podle user-agent a mobilního operátora (`carrier`) podle IP adresy. Následně zavolá metodu `getAdsByTarget(id, channel, lat, lon, caps, carrier, manuf, medium)`, která pro jednotlivé kontextové parametry vytáhne z databáze množiny odpovídajících sdělení. Nad těmito množinami provede průnik čímž dostane množinu sdělení, která odpovídají všem kontextovým parametrům a skrz nadřazenou metodu vrátí až do funkce `getShowContent`.

Další krok reprezentuje funkce `evaluateAds(adIds, channel)`. Tato funkce přiřadí jednotlivým sdělením bonusové body, které zvýší šanci daného sdělení zvítězit v pozdějším losování.

Poslední krok zajistí funkce `buildAdContent(evalAdIds,id,lat,lon,ua,ip,rua,rip)`. Tato metoda vylosuje podle typu cílového obsahu (např. WAP, J2ME,...) 1 až N sdělení a sestaví je do formátu, který umí zpracovat plugin žádající sdělení. Tato funkce zároveň provede zápis do logů o každém sdělení, které bylo zásláno v odpovědi. Záznam o zobrazení obsahuje zejména kompletní kontext, čas zobrazení, id sdělení, id obsahu a další zajimavé informace. Samotná odpověď má, v případě WAPu, podobu textového odkazu, který je zformátovaný následujícím způsobem:

```
Text sdělení###http://www.adplaze.com/r/?i=X
```

Jak je z příkladu zřejmé, WAPový obsah obdrží vždy jen jedno sdělení, které je samo o sobě titulek a samotné URL hypertextového odkazu. Odkaz pak směřuje na webový skript, který zaloguje prokliknutí a vratí HTTP redirect na cílový WAP.

Zaprotokolování prokliku zajistí již výšše zmíněná metoda `getClickedContent(impId)`. Její jediný parametr je id zobrazení daného sdělení, který získá z CGI parametru `i` webového skriptu. Na základě id zobrazení může provést update daného řádku v tabulce záznamů o zobrazení, kde nastaví čas prokliku.

5.2.2 Databázový konektor

Jednotný přístup k databázi je implementován v souboru `ap_mysql.rb`. Jak již z názvu vyplývá jako databázový server byla zvolena MySQL (verze 5.0). Při výměně databáze se pak pouze implementuje odpovídající kontektor se stejným API. Jedinou třídou je zde třída `DBconnector`. Konstruktor této třídy zajistí vytvoření spojení s databází, které je reprezentováno třídní proměnnou `@@mconn`.

`DBconnector` poskytuje sadu veřejných metod, které zajišťují dané databázové operace. Příkladem může být například funkce `getMsgsByManuf(manuf)`, která pouze pomocí jednoduchého SQL selectu získá množinu sdělení odpovídajících danému výrobci. Obdobně vypadají i zbylé metody této třídy.

5.2.3 WURFL inspektor

Detaily o mobilních zařízeních zpřístupňuje pro webovou službu třída `CapMiner` implementovaná v souboru `ap_wurfl.rb`. Tato třída poskytuje dvě metody: `getCapabs(user_agent)`

a `getManuf(user_agent)`. První z nich naplní a vrátí hash s vlastnostmi telefonu, které nás zajimají (tyto vlastnosti jsou uložené v databázi a `CapMiner` dostane předpřipravený hash s jejich výčtem jako jediný parametr konstruktoru). Druha jak už je z názvu patrné vrátí tak též na základě user-agent výrobce telefonu.

Třída `CapMiner` využívá ke své činnosti třídu `WurflInspector` implementovanou ve stejném souboru. Nejdůležitější metodou této třídy je `search_handsets(proc)`, která prohledá záznamy jednotlivých zařízení a vrátí ty, které splňují danou podmínku. Tuto podmínku obdrží metoda `search_handsets()` jako svůj jediný parametr ve fromě procedury, představované objektem třídy `Proc` (patří do standardní knihovny jazyka Ruby).

5.2.4 SOAP exportér

Samotný SOAP server, zpřístupňující webovou službu, implementuje třída `Ap_SOAPServer` obsažená v souboru `ap_soapServer.rb`. Tato třída dědí z `SOAP::RPC::StandaloneServer` z knihovny soap4r a implementuje jedinou metodu `on_init()`, která je vyvolána při vytváření nové instance. `on_init()` pouze vytvoří nový objekt třídy `ApDirector` a zaregistruje její dvě metody na SOAP server, který následně nastartuje zdědenou metodou `start`. `ap_soapServer.rb` tedy slouží jako spouštěcí soubor aplikace. Pro jeho ovládání byl vytvořen shellovský skript `kimserver`.

Kapitola 6

Dosažené výsledky a další rozvoj projektu

V této kapitole budou představeny výsledky zátěžových testů ISPS a KIM a možnosti dalšího rozvoje projektu.

6.1 Výsledky zátěžových testů

Zátěžové testy mají za cíl ukázat výkonnost vyvinutých aplikací na referenčním serveru. Konfigurace serveru je následující:

- Intel Xeon 2,8GHz,
- 512 MB RAM, DDR2, 667MHz
- 2x 80GB harddisk, SATA, 10000RPM, SW RAID1
- GNU/Linux Debian 4.0 amd64, Linux 2.6.8-smp
- OpenVZ

Obě části systému (KIM, ISPS) běží na jednom virtuálním privátním stroji (VPS), který má přiděleno 100% zdrojů výšše uvedeného hardware. Testy byly prováděny opakovaným dotazem na obě metody WS KIM (`getShowContent()`, `getClickedContent()`). Každý test trval přesně 60 minut. Přehled dosažených výsledků ukazuje tabulka 6.1.

Nejprve byl proveden test samotné WS KIM. Zde bylo dosaženo výkonu 5,4 odpovědi/s. Tento test bude sloužit jako referenční pro následující testy, kde bude zapojen ISPS.

Ve druhém testu byl KIM dotazován skrze ISPS což sebou přineslo dvě TCP spojení na dotaz navíc. Výkon v tomto případě ale klesl pouze o 0,1 odpovědi/s, tedy jen velmi nepatrně.

Ve třetím testu byl na ISPS zaregistrován neblokující PSM. Zde se při každém dotazu tedy vytvářela dvě nová vlákna, která navazovala po jednom TCP spojení. Výkon klesl

na 3,02 odpovědi/s. Jelikož z předchozího testu je patrné, že samotné další TCP spojení výkon příliš neovlivní, úbytek způsobilo vytvoření nového vlákna, které se na OS Linux rovná novému procesu.

Ve čtvrtém testu byl na ISPS zaregistrován pouze blokující PSM, každý požadavek tedy vyvolal dvě další TCP spojení navíc v podobě volání dvou SOAP WS. I zde klesl výkon pouze nepatrně na 4,93 odpovědi/s.

Pátý test vyzkoušel systém s registrovanými blokujícími i neblokujícími triggery. Citelný úbytek výkonu lze přičíst opět vytváření dvou nových vláken při každém požadavku.

režim	pož.	pož./s	TCP spoj./pož.	vláken/pož.
(1) KIM	19410	5,40	1	0
(2) ISPS, KIM	18972	5,27	2	0
(3) ISPS, KIM (noblock)	10872	3,02	4	2
(4) ISPS, KIM (block)	17748	4,93	4	0
(5) ISPS, KIM (noblock, block)	9756	2,71	6	2

Tabulka 6.1: Zátěžové testy

Jak bylo uvedeno výše, testy byly prováděny pouze v rámci jednoho fyzického serveru. Nejvíce se tedy projevila režie vytváření nového procesu u neblokujících triggerů. Testy ukázali, že neblokující triggery mají smysl až v případě skutečně náročných modulů proxy serveru. Vytváření nového vlákna má několikanásobně větší režii jak TCP spojení (v případě rychlé konektivity) a provedení kratké operace jako např. zaprotokolování či kontrola podle jistých pravidel jsou tedy natolik krátké, že je pak lépe je registrovat jako blokující PSM. Neblokující triggery se tedy s výhodou použijí u volání PSM dostupných na nespolehlivé konektivitě či v případě, že provádějí déle trvající a náročnější operace.

6.2 Návrhy dalšího rozvoje projektu

Implementace ISPS je v této práci na úrovni funkčního prototypu a zůstává tak mnoho směrů, kterými by se jeho vývoj mohl ubírat. Jedním z nich je další rozvoj ISPS API, master a slave serverů.

Registrace je v současné době zajištěna uložením kódu wrapperů WS i triggerů PSM do souborů, nabízí se ale možnost ukládat je v rozumnější formě do relační databáze a připravit pro registraci webového správce. Další prostor pro práci představuje implementace wrapperů a triggerů, či rozšíření množiny modulů proxy serveru. Výsledky testů ukázali, že by bylo vhodné aby se vlákna pro neblokující triggery při volání WS nevytvářela ale ideálně již běžela po celou dobu běhu ISPS a pouze zpracovávala požadavky na spuštění konkretních neblokujících triggerů. Tím by se odstranil hlavní neduh systému neblokujících triggerů v podobě velké režie na vytvoření nového vlákna při každém požadavku.

KIM WS je naproti ISPS jednodušší na samotný návrh i implementaci. Prostor pro další rozvoj u ní představuje zejména vytvoření pluginů pro další platformy a upravení samotné WS pro práci s těmito platformami. Dalším směrem může být například rozšíření o možnost vkládání kontextových sdělení do multimediálních dat.

Kapitola 7

Závěr

Cílem této práce bylo vytvořit webovou službu poskytující kontextové informace mobilním zařízením a prototyp inverzního SOAP proxy serveru umožňujícího efektivní monitoring a management všech webových služeb organizace.

Webová služba poskytující kontextové informace mobilním zařízením byla navrhnuta a implementována nad protokolem SOAP a umožňuje výběr relevantního sdělení na základě mnoha informací o typu mobilního zařízení a stavu ve kterém se nachází. Výsledná implementace umožňuje zobrazovat tato sdělení na WAPových stránkách a mobilních aplikacích postavených na platformě J2ME, ale systém je připravený na jednoduché rozšíření o další aplikace, kde bude možné kontextové informace zobrazovat (například mobilní aplikace na OS Symbian).

Implementace inverzního SOAP proxy serveru pro webové služby se přes některé těžkosti podařilo dovést k funkčnímu prototypu. Jeho architektura umožňuje jednoduše škálovat funkčnost systému a provozovat ho v distribuovaném heterogenním prostředí. V rámci implementace bylo vytvořeno funkční jádro umožňující registraci webových služeb, který má proxy server poskytovat a funkčních modulů v podobě speciálních webových služeb zajišťujích vždy konkretní funkci ISPS. Pro jeho otestování byly vytvořeny dvě webové služby zajištující protokolování SOAP transakcí a možnost dočasně odepřít přístup k vybraným webovým službám.

Literatura

[1] Y. Amir. Distribuované operační systémy. http://www.cs.jhu.edu/ yairamir/cs437/week7.ps (listopad 2006).

[2] D. Box. Soap: Simple object access protocol. http://static.userland.com/xmlRpcCom/soap/SOAPv11.htm (duben 2007).

[3] T. Bátrla. *Distribuce místně závislých informací v mobilních zařízeních (diplomová práce)*. FIT VUT Brno, 2007.

[4] J. Košek. *Inteligentní podpora navigace na WWW s využitím XML (diplomová práce)*. VŠE Praha, 2002.

[5] M. Kuba. Web services. http://www.ics.muni.cz/zpravodaj/articles/269.html (listopad 2006).

[6] WWW stránky. Calling methods dynamically. http://www.rubycentral.com/articles/dynacall.html (duben 2007).

[7] WWW stránky. Documentation for the ruby programming language. http://www.ruby-doc.org/ (duben 2007).

[8] WWW stránky. Extensible markup language (xml) 1.0 (fourth edition). http://www.w3.org/TR/xml/ (říjen 2006).

[9] WWW stránky. Grokking idiomatic dynamic ruby. http://wanderingbarque.com/ruby/DynamicRuby.html (duben 2007).

[10] WWW stránky. Http - hypertext transfer protocol. http://www.w3.org/Protocols/ (říjen 2006).

[11] WWW stránky. Otevřená encyklopedie wikipedia. http://en.wikipedia.org/ (listopad 2006).

[12] WWW stránky. Soa terminology overview, part 1: Service, architecture, governance, and business terms. http://www-128.ibm.com/developerworks/webservices/library/ws-soa-term1 (listopad 2006).

[13] WWW stránky. Soa terminology overview, part 2: Development processes, models, and assets. http://www-128.ibm.com/developerworks/webservices/library/ws-soa-term2 (listopad 2006).

[14] WWW stránky. Soa terminology overview, part 3: Analysis and design. http://www-128.ibm.com/developerworks/webservices/library/ws-soa-term3 (listopad 2006).

[15] WWW stránky. Soap version 1.2. http://www.w3.org/TR/soap/ (listopad 2006).

[16] WWW stránky. Web services and service-oriented architectures. http://www.service-architecture.com/ (listopad 2006).

[17] WWW stránky. Web services architecture. http://www.w3.org/TR/ws-arch/ (říjen 2006).

[18] WWW stránky. Web services description language (wsdl) 1.1. http://www.w3.org/TR/wsdl (říjen 2006).

Seznam tabulek

Seznam obrázků