VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ADRESÁŘ WEBOVÝCH SLUŽEB

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE JIŘÍ PAGÁČ
AUTHOR

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ADRESÁŘ WEBOVÝCH SLUŽEB

WEB SERVICE BROKER

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — JIŘÍ PAGÁČ

AUTHOR

VEDOUCÍ PRÁCE — Ing. PETR WEISS

SUPERVISOR

BRNO 2008

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Pagáč Jiří**
Obor: Informační technologie
Téma: **Adresář webových služeb**
Kategorie: Softwarové inženýrství

Pokyny:

1. Seznamte se s problematikou webových služeb (WS) - XML, SOAP, WSDL, HTTP, UDDI.
2. Prostudujte možnosti adresování a vyhledávání WS a způsoby uchovávání a poskytování těchto údajů.
3. Navrhněte adresář WS.
4. Implementujte navržený adresář jako ukázkovou aplikaci. Na ukázkovém příkladě demonstrujete funkčnost zvoleného postupu.
5. Diskutujte dosažené výsledky a další postup.

Literatura:

- Erl, T.: *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, 2005, ISBN 0-13-185858-0.
- Newcomer, E.: *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. Addison-Wesley, 2002, ISBN 0-201-75081-3.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3 zadání.

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

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% 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 nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Weiss Petr, Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

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

L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Jiří Pagáč**
Id studenta: 78817
Bytem: Dolní 17, 794 01 Krnov
Narozen: 13. 11. 1985, Karviná
(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):
   bakalářská práce

Název VŠKP: Adresář webových služeb
Vedoucí/školitel VŠKP: Weiss Petr, Ing.
Ústav: Ústav informačních 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

Abstrakt

Práce se zabývá studiem technologií webových služeb, možností jejich vyhledávání a implementací systémů určených k registraci a vyhledávání webových služeb, tzv. adresářů. Blíže se věnuje adresáři UDDI (*The Universal Description, Discovery and Integration*), jeho datovým strukturám, programovému rozhranní a problematice registrace služby a jejího vyhledání s využitím UDDI. Hlavní část práce je zaměřena na implementaci části standardu UDDI v.3.0.2 pro účely demonstrace registrace a vyhledání služby v adresáři typu UDDI. Výsledná aplikace má sloužit při výuce problematiky webových služeb. Aplikace je napsána v programovacím jazyce C++ s využitím SŘBD Firebird, knihoven pro komunikaci s databází IBPP a C++ toolkitu pro vývoj webových služeb v C++ – gSOAP.

Klíčová slova

webové služby, WS, vyhledávání služeb, registrace služeb, popis webové služby, XML, WSDL, SOAP, UDDI, WSIL, adresář webových služeb, gSOAP, architektura orientovaná na služby, SOA

Abstract

The thesis deals with the study of the technology of web services and how to search for them, and with the implementation of systems used for registration of and searching for web services – directories. Greater attention is paid to the UDDI (*The Universal Description, Discovery and Integration*) directory, its data structures, program interface and problems related to registration of the service and searches using UDDI. The main part of the study deals with the implementation of a part of UDDI standard version 3.0.2 for the demonstration of the registration and search of the service in a UDDI type directory. The resultant application should be used for teaching problems concerning web services. It is written in C++ programming language using Firebird, libraries for communication with Firebird database IBPP and C++ toolkit for the development of web services in C++ – gSOAP.

Keywords

web services, WS, service discover, service publishing, web service description, XML, WSDL, SOAP, UDDI, WSIL, web service broker, gSOAP, service oriented architecture, SOA

Citace

Jiří Pagáč: Adresář webových služeb, bakalářská práce, Brno, FIT VUT v Brně, 2008

Adresář webových služeb

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Petra Weisse.

. . . . . .
Jiří Pagáč
13. května 2008

Poděkování

Zde bych rád poděkoval vedoucímu práce Ing. Petru Weissovi za poskytnutou odbornou pomoc během tvorby mé práce.

Obsah

Úvod

Webové služby jsou dnes velmi používaným nástrojem pro komunikaci aplikací. Název, jak je z něj patrné, si vysloužily díky tomu, že jsou založeny na stávajících technologiích World Wide Webu. Samotná myšlenka, spouštění metod na jiném systému, není nová, ale až právě tato implementace vzdáleného volání funkcí v distribuovaných prostředích (jakým je např. internet nebo rozsáhlé vnitropodnikové sítě) se dočkala velmi širokého využití. Právě technologie webu dokázaly zjednodušit celý proces vývoje komponent a komunikace mezi dvěma programy.

Na rozdíl od starších systémů, např. CORBA (*Common Object Request Broker Architecture*), se webové služby dokázaly prosadit ve firemním prostředí a to hlavně díky jejich flexibilitě. Ta se projevuje, mimojiné tím, že díky jazyku XML a standardním přenosovým protokolům (např. HTTP), dokázala tato technologie spojit části podnikových systémů, které jsou napsány v různých jazycích a fungují na různých platformách, což se pozitivně projevilo hlavně při vzájemném propojování firemních prostředí.

S nástupem servisně orientované architektury (*SOA Service Oriented Architecture*) v podnicích a globalizace společnosti, roste důležitost a možnost uplatnění lidí, schopných tuto problematiku pochopit a zvládnout.

Kromě všeobecně známých technologií, jako je jazyk XML nebo protokol HTTP, se ve spojení s webovými službami používají i podpůrné systémy, které mají podpořit využívání webových služeb. Mezi takové technologie patří i adresáře webových služeb. Ty mají umožnit jejich registraci, následné vyhledání a tím pádem jednodušší znovupoužitelnost. Adresáře, neboli registry, jsou také důležitou součástí SOA. Protože službu můžeme použít jen tehdy, pokud ji umíme najít a současně s tím získat informace, které umožní implementovat klientskou aplikaci služby.

Cílem této práce je vytvořit aplikaci adresáře, která umožní princip registrace a vyhledání služby demonstrovat a pomůže zájemcům o tuto problematiku pochopit tento princip. Dalším cílem práce je popularizovat téma webových služeb a rozšířit počet dostupných materiálů pro další studium této problematiky.

Práce se skládá z úvodní části, která představuje motivaci v podobě výstavby SOA architektury a zabývá se spojením SOA s technologií webových služeb a jejich adresářů. V druhé kapitole je stručně popsána teorie webových služeb a souvisejících technologií. Představuje nezbytné minimum při pochopení toho, jak webové služby fungují.

Následující kapitola se zabývá cílem této práce, tedy adresářem webových služeb. Rozebírá dva hlavní produkty v této oblasti a naznačuje jejich využití v praxi. U adresáře typu UDDI se hlouběji zabývá popisem vnitřních struktur a způsobem reprezentace dat uvnitř tohoto typu adresáře.

V další kapitole je popis nástrojů použitých při implementaci a také shrnutí licencí pod kterými jsou šířeny a které upravují způsob jejich použití.

Pátá kapitola obsahuje obsah mé práce na aplikaci adresáře. Opakuje požadavky na

aplikaci, rozebírá její návrh a problémy při implementaci. Dokumentuje jakým způsobem aplikace funguje a jak je možné registrovat službu. Nakonec shrnuje přínosy práce a směry, jakými by se dala aplikace v budoucnu rozšířit.

Ná pátou kapitolu navazuje konkrétní popis programové části aplikace v kapitole následující. Popisuje také důvody pro volbu použitého programovacího jazyka a slovní popis algoritmu, ve kterém se provádí obsluha příchozích požadavků.

Sedmá a poslední kapitola obsahuje závěr a shrnutí výsledků práce.

Kapitola 1

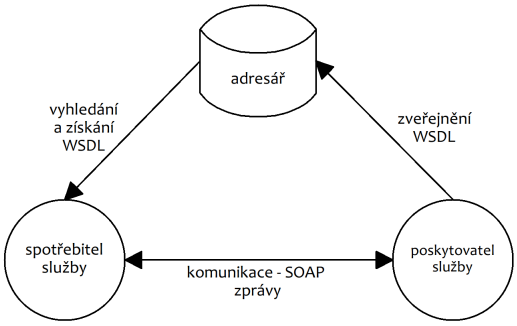
Servisně orientovaná architektura

Servisně orientovaná architektura (*Service Oriented Architecture*) je stále ve fázi vývoje a neexistuje žádná přesná definice. SOA tak lze popsat několika různými způsoby [10].

Současné SOA reprezentuje otevřenou, živou, rozšiřitelnou, sdruženou a uspořádanou architekturu, složenou z autonomních, škálovatelných, na poskytovateli nezávislých, spolupracujících, vyhledatelných a případně znovupoužitelných celků (služeb), implementovaných s pomocí webových služeb.

Další charakteristika říká, že SOA umožňuje vytvořit abstraktní vrstvu mezi podnikovými procesy a technologií.

Tato architektura je založená na technologiích webových služeb a jedna z prvních implementací byla založena na využití tří základních komponent – spotřebiteli služeb, poskytovateli služeb a na adresáři služeb. Toto se někdy označuje jako primitivní SOA (obrázek 1.1).



Obrázek 1.1: Primitivní SOA [10]

1.1 SOA a WS

Webové služby jsou základním stavebním blokem SOA. Představují samostatné jednotky, které mohou být distribuovány v síti a využívány podle aktuální zátěže. Jsou na sobě také nezávislé a obecné do té míry, že mohou být znovu využitelné v jiném systému – nebo business procesu. To umožňuje snížit cenu vývoje a jednoduše propojit stávající systémy bez nutnosti složitých úprav. [2]

Kapitola 2

Webové služby

Následující kapitola představuje teoretický úvod do problematiky webových služeb (WS) a souvisejících technologií.

2.1 Definice WS

Webové služby jsou (podle [1]) softwarovým systémem navrženým k podpoře interakce mezi stroji přes prostředí sítě. Mají rozhraní popsané ve strojově zpracovatelném formátu (obrázek 2.4.4). Ostatní systémy interagují s webovou službou způsobem popsaným tímto popisem s využitím SOAP zpráv, s obsahem ve formátu XML, transportovaných protokolem HTTP *Hypertext Transport Protocol*.

2.2 Vlastnosti WS

Mezi podstatné vlastnosti webových služeb patří jejich otevřenost, ve smyslu používání otevřených standardů. To umožnilo pravou kompatibilitu a přenositelnost mezi platformami i mezi použitými programovacími jazyky. Celý koncept staví na technologiích, které byly a jsou běžně používány a známy, takže jejich implementace i použití je snadné.

Z pohledu koncového uživatele je podstatné, že technologie webových služeb dokáže spojit různé, již existující, softwarové systémy. Toto spojení umožňuje použití standardních transportních protokolů (HTTP, aj.).

2.3 Interakce s WS

Standardy webových služeb používají dva hlavní typy interakce – RPC (*Remote Procedure Call*) a „Document oriented“ styl.

Styl RPC [7] je určen pro volání metod, u kterého jsou parametry metody zabaleny do elementu, jehož název určuje volanou metodu. Styl document se používá k předávání libovolných XML dokumentů.

2.4 Technologie WS

Jádro technologie WS tvoří tyto základní komponenty:

- *přenosové protokoly* – v první generaci WS to byl většinou protokol HTTP, ale rozšíření *WS – Addressing* umožnilo nezávislost na konkrétním přenosovém protokolu u služeb druhé generace. V mé práci se budu věnovat protokolu HTTP.
- *XML (eXtensible Markup Language)* – značkovací jazyk XML
- *SOAP (Simple Object Access Protocol)* – protokol pro přenos strukturovaných dat
- *WSDL (Web Services Description Language)* – jazyk pro popis rozhraní služeb
- *UDDI (Universal Description, Discovery and Integration), WSIL (Web Services Inspection Language)* – adresář podporující registraci a nalezení služeb

2.4.1 Transportní protokol HTTP

HTTP (*Hypertext Transfer Protocol*) je transportní protokol široce používaný při komunikaci mezi webovými službami. Ve spojení s webovými službami slouží pro přenos XML požadavků ve zprávách SOAP protokolu.

Definice

HTTP je protokol [8], pracující na aplikační vrstvě a určený pro distribuované, spolupracující, hypertextové informační systémy. Zprávy HTTP protokolu sestávají z požadavků od klienta k serveru a odpovědi od serveru ke klientovi. Požadavky i odpovědi jsou složeny z hlaviček a těla, ve kterém jsou přenášena data. Typ dat se určuje podle MIME (*Multipurpose Internet Mail Extension*) typu v hlavičce *Content-Type* (obrázek 2.1).

```
POST /uddi/ HTTP/1.1
Host: www.example.com:9000
User-Agent: gSOAP/2.7
Content-Type: text/xml; charset=utf-8
Content-Length: 1141
Connection: close
SOAPAction: ""
```

Obrázek 2.1: Hlavička HTTP požadavku přenášejícího SOAP zprávu.

2.4.2 XML

XML je značkovací jazyk velmi podobný HTML. Byl vytvořen pracovní skupinou konsorcia W3C – XML Working Group. Na rozdíl od HTML umožňuje definovat vlastní (počtem neomezené) elementy a nedefinuje jak mají být elementy zobrazeny, ale jaká data elementy XML dokumentu obsahují.

Definice

XML dokument (obrázek 2.2) představuje stromovou datovou strukturu s právě jedním kořenem. Uzly tohoto stromu se nazvývají *tagy*. Listy stromu mohou být tagy, atributy nebo textový obsah tagů. Atributy *id* mohou odkazovat na tagy a vytvořit tak obecný orientovaný graf. [7]

Vlastnosti a cíle XML

Kromě definice vlastních elementů, můžeme definovat i typy dat, které elementy obsahují. To je velmi důležité, protože bez definice datových typů by mohlo dojít k různé interpretaci stejných dat mezi dvěma komunikujícími systémy. Proto se zavedla tzv. XML schémata, která obsahují definice a sémantiku jednotlivých elementů. Pokud tedy budou mít obě strany přístup ke stejnému XML schéma, bude zaručeno, že data budou interpretována stejně.

XML tvoří jádro na kterém jsou postaveny klíčové technologie webových služeb. Jedná se především o popis služby v jazyce WSDL a protokol SOAP. Obojí je založeno na XML.

```
<?xml version="1.0" encoding="UTF-8" ?>
<předmět ukončení="Zk">
  <název>Web Services</název>
  <cíle>Studenti se naučí zvládat problematiku Webových služeb.</cíle>
    <studenti>
       <student ročník="2">Jan Novák</student>
       <student ročník="3">Jiří Pagáč</student>
    </studenti>
    <zkoušky termíny="2">
       <řádná datum="2008-10-05T11:00:00" />
       <opravná datum="2008-20-05T09:00:00" />
    </zkoušky>
</předmět>
```

Obrázek 2.2: Ukázka XML dokumentu.

2.4.3 SOAP

Komunikace mezi službami je založena na výměně zpráv. Proto musí být protokol pro jejich výměnu standardizován tak, aby všechny služby používaly stejný formát a transportní protokoly. [10]

V první generaci webových služeb byl k přenášení SOAP zpráv využit protokol HTTP, později byla vydána specifikace *WS – Addressing*, která umožňuje využít i jiné protokoly.

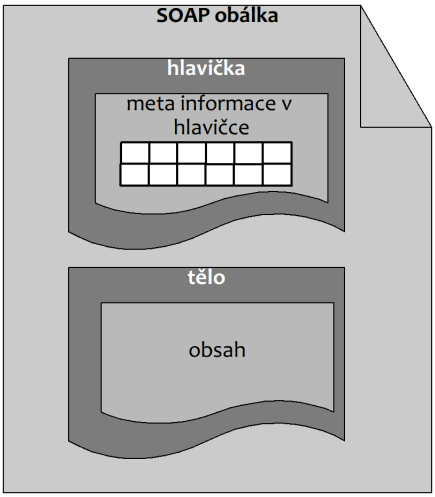
Se SOAP protokolem většinou nezachází programátor přímo, ale většinou přes nějakou knihovnu třetí strany.

Definice

SOAP (*Simple Object Access Protocol*) je protokol určený k výměně strukturovaných informací v decentralizovaných a distribuovaných prostředích. Využívá XML (kapitola 2.4.2) a poskytuje prostředky k sestavení zprávy, kterou lze předat přes velké množství transportních protokolů (nejčastěji HTTP, viz. kapitola 2.4.1). Systém je navržen tak, aby byl nezávislý na jakémkoliv určitém programovém modelu nebo specifické implementaci. [6]

Formát zpráv

Každá SOAP zpráva je zabalená do obálky (*envelope*), která obsahuje ostatní části zprávy (obrázek 2.3). Dále může každá zpráva obsahovat *hlavičku*, což je oblast, která je určena



Obrázek 2.3: Základní struktura SOAP zprávy [10]

pro meta informace. Samotný obsah je umístěný v těle zprávy. [10]

Volání metody služby

Metodou POST protokolu HTTP (nebo jiným způsobem) je zpráva přenesena na server služby. Každá metoda služby musí být v nějakém jmenném prostoru, který je určen nějakým URI (*Uniform Resource Identifier*). Datové typy metod jsou určeny pomocí XML schéma.

Na serveru se služba obvykle vyhledá podle jména a jmenného prostoru, ke kterému patří. Další možností je určení služby podle URL (*Uniform Resource Locator*) na kterou přišel požadavek nebo podle hlavičky SOAPAction. Konkrétní způsob není předepsán. [7]

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:uddicall="urn:uddi-org:api_v3_call"
xmlns:uddi="urn:uddi-org:api_v3">
<SOAP-ENV:Body>
<uddicall:save_business>
  <uddi:save_business>
    <uddi:businessEntity>
      <uddi:name>Jmeno businessEntity</uddi:name>
      <uddi:description>Popis businessEntity</uddi:description>
      </uddi:businessEntity>
  </uddi:save_business>
</uddicall:save_business>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Obrázek 2.4: Ukázka SOAP volání metody `save_business`.

2.4.4 WSDL

WSDL je jazyk založený na XML, sloužící k jednotnému popisu rozhraní webových služeb a šíří se ve formě XML dokumentů.

Definice

WSDL popisuje místo, kde lze službu kontaktovat, tzv. komunikační koncový bod (*service endpoint*, nebo také jen *endpoint*) a poskytuje formální definici rozhranní tohoto bodu tak, aby klient služby přesně věděl, jak má strukturovat požadavek. Také poskytuje fyzické umístění (adresu) služby. [10]

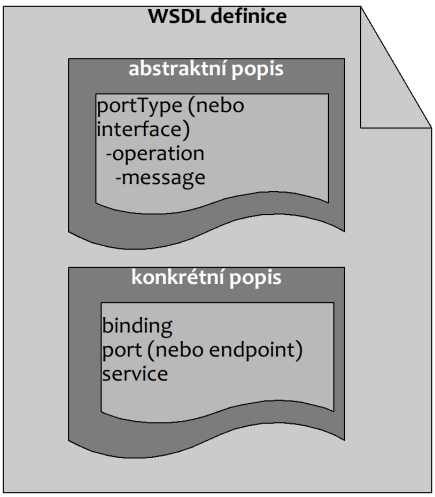
Aby se dala služba vzdáleně zavolat je třeba o ní znát především:

- Kde lze službu kontaktovat (URL, přenosové protokoly, atd.)
- Jak lze navázat komunikaci se službou
- Popis metod, které služba poskytuje (jména funkcí, názvy a typy parametrů, typy návratových hodnot)

Struktura WSDL

WSDL popis je rozdělen na dvě části (obrázek 2.5):

- Abstraktní popis
- Konkrétní popis



Obrázek 2.5: Struktura WSDL dokumentu. [10]

Abstraktní popis

Abstraktní popis charakterizuje rozhraní služby bez návaznosti na technická specifika hostitelského systému nebo způsobu jakým bude komunikovat. Oddělením těchto informací je

udržena integrita popisu nezávisle na případných změnách platformy nebo způsobu komunikace. [10]

Abstraktní popis sestává ze tří hlavních částí:

- portType (ve verzi 2.0 specifikace byl portType přejmenován na *interface*)
- operation
- message

Konkrétní popis

Představuje spojení mezi abstraktním popisem služby a reálnou implementací. Také je zde popsáno propojení na fyzický transportní protokol. [10]

Tvoří jej tři části:

- binding
- port (ve verzi 2.0 specifikace byl port přejmenován na *endpoint*)
- service

2.4.5 Popis elementů WSDL

Popis jednotlivých elementů WSDL (podle [3] a [7]):

- *datové typy (types)* – definice datových typů ve formě XML schémat nebo jiného mechanizmu.
- *zpráva (message)* – definice zpráv, obsahuje jednu nebo více částí (*part*), které odpovídají parametrům a návratovým hodnotám
- *operace (operation)* – metody služby, každá definuje většinou vstupní a výstupní zprávu
- *rozhraní (portType)* – souhrn operací (operation)
- *vazba (binding)* – definuje, jak se volá daná brána (port), např. SOAP přes HTTP
- *brána (port)* – kombinace vazeb (binding) a síťové adresy pro přístup ke službě
- *služba (service)* – kolekce souvisejících

2.4.6 Styly WSDL

Vazby (binding) ve WSDL specifikují pro každou operaci atributy *style* a *use*. *Style* může obsahovat hodnoty *rpc* nebo *document* (kapitola 2.3). Stylem document lze předávat jen XML dokumenty.

Atribut *use* určuje, jestli se mají parametry zapsat pomocí typového systému SOAP (encoded) nebo XML schéma (literal) a může nabývat hodnot *literal* nebo *encoded*.

Pro udržení kompatibility mezi různými implementacemi se dnes používá převážně kombinace document/literal.

V případě použití RPC se používá tzv. document/literal wrapped styl, což je volání simulované XML dokumentem s jedním hlavním elementem se jménem shodným s volanou metodou. Používání tohoto stylu se dnes doporučuje. [7]

Kapitola 3

Adresáře webových služeb

Adresáře služeb jsou podstatnou součástí konceptu webových služeb. Aby mohl klient službu využít, tak musí znát její umístění, službou poskytované metody, formáty zpráv a definici datových typů. Adresáře tak reprezentují koncept registrace (také se užívá termín *publikace*) a vyhledání (*nalezení*) služby.

Jejich primárním účelem je poskytování odkazů na popisy (WSDL) služeb a umožnit jejich snadné vyhledání podle zadaných kritérií. To umožňuje prezentaci organizace, vyhledání obchodních partnerů a možnost snadného propojení s jejich infrastrkturou skrze webové služby. [10]

Během vývoje celého konceptu webových služeb se většina použitých technologií ustálila do více, či méně, stabilní podoby. Jedinou výjimkou jsou právě adresáře. Časem se vyvinuly dva koncepty. Adresáře typu UDDI (*The Universal Description, Discovery and Integration*) a WSIL (*Web Service Inspection Language*).

3.1 WSIL

WSIL se ukázal jako bezpečnější systém v prostředí internetu a pro koncové spotřebitele. Není náhradou UDDI, ale spíše jeho doplňkem. Nespoléhá se na koncept centrálního registru, řízeného operátorem třetí strany. Místo toho si každá organizace umístí do kořenového adresáře svého webového serveru soubor `inspection.wsil`, který obsahuje popis služeb poskytovaných organizací. Každá organizace si proto kontroluje záznamy o svých službách sama a uživatel tak má jistotu, že záznam bude aktuální a že není podvržený. A protože je umístění souboru `inspection.wsil` dobře známo, lze použít existující webové vyhledávače k prohledávání těchto souborů a k vyhledání služeb. [7]

Uživatel tak postupuje opačně, než u UDDI. Místo aby hledal službu a podle toho vybíral poskytovatele, si nejdříve vybere poskytovatele a teprve pak si vybere mezi nabízenými službami.

Tato práce je zaměřena na adresáře typu UDDI, proto se nebudu WSIL dále zabývat.

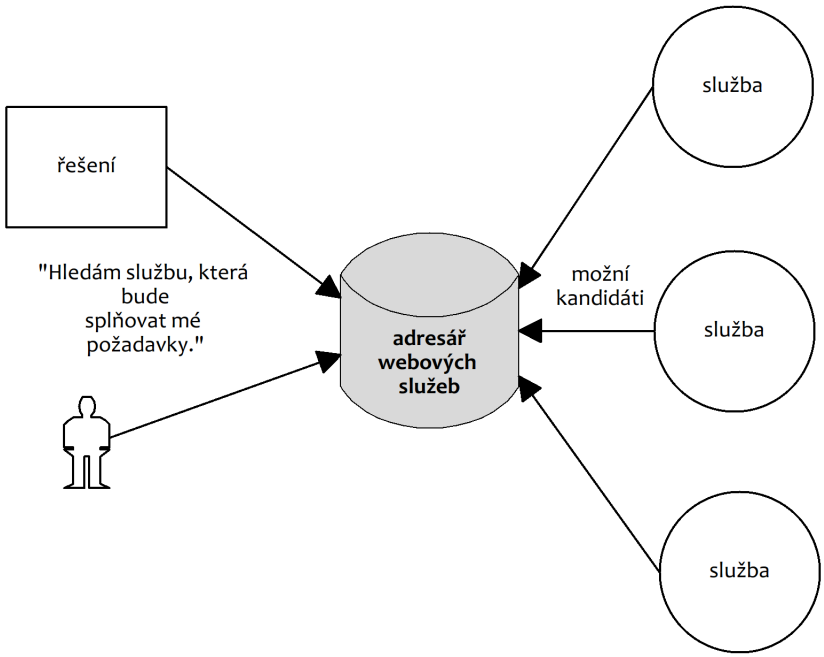
3.2 UDDI

Koncept UDDI je starší. Na jeho vytvoření se podílely firmy jako Microsoft, IBM nebo Ariba. Je založen na myšlence centrálního adresáře, kde se budou soustřeďovat informace (obrázek 3.1) o poskytovateli služby, odkazy na popisy (WSDL, viz. kapitola 2.4.4) služeb, nebo jejich přesná reprezentace pomocí tModelů.

V UDDI lze uchovat také velké množství doprovodných informací, podle kterých se dá organizace kontaktovat, nebo vyhledávat podle různých charakteristik (kategorizací), např. podle čísla v evidenci daňového úřadu.

Pojem služby je v UDDI adresářích chápán velmi obecně. V adresáři může být u organizace uvedeno např. jen telefonní nebo faxové číslo. Služba tak nemusí být přímo webová služba a její zavolání a výsledek nemá nic společného se zasíláním SOAP zpráv.

Možnosti typů a identifikace objektů uvnitř UDDI lze téměř neomezeně rozšiřovat s pomocí tModelů.



Obrázek 3.1: Centrální registr. [10]

UDDI standardizuje organizace OASIS a v současnosti existuje verze 3.0.2 specifikace, která standardizuje chování UDDI uzlů. Tato specifikace je rozdělena na související části, kterým se říká *API*[1] *set*.

Pro následující text byla zdrojem specifikace UDDI [5], pokud není uvedeno jinak.

3.2.1 Privátní a veřejné adresáře

Adresáře lze použít buď jako veřejné, nebo privátní.

Veřejné adresáře přijímají registrace od jakékoliv organizace. Veřejný provoz této služby ale nesplnil očekávání, a proto byla většina velkých veřejných adresářů uzavřena.

Místo veřejného použití se pro UDDI našlo využití v podnicích a organizacích, především při implementaci SOA (kapitola 1). Adresář zde plní funkci centrálního repozitáře pro popisy všech služeb, které organizace spravuje. [10]

3.2.2 Application Program Interface

UDDI Specifikace poskytuje následující API (*Application Program Interface*). Tato API jsou používána při komunikaci s adresářem.

[1] application programming interface – rozhraní pro programování aplikací

- UDDI Inquiry – vyhledání organizace, webové služby, konkrétní vazby (binding) služby nebo tModelu
- UDDI Publication – registrace organizace, webové služby, konkrétní vazby (binding) nebo tModelu
- UDDI Security – práce s bezpečnostním tokenem
- UDDI Custody Transfer
- UDDI Subscription
- UDDI Replication
- UDDI Subscription Listener
- UDDI Value Set

3.2.3 Registry a uzly

Jeden adresář UDDI může být decentralizován mezi UDDI uzly. UDDI uzel odpovídá jedné (nebo více) webové službě, která implementuje alespoň jedno UDDI API a splňuje tato pravidla:

- uzel musí být členem právě jednoho UDDI adresáře
- uzel má přístup (a právo modifikace) k vlastní, kompletní, logické kopii databáze UDDI adresáře, jehož je členem.

Pokud se adresář skládá z více než jednoho uzlu, musí všechny uzly implementovat API pro replikaci dat. Adresář musí povinně obsahovat alespoň jeden uzel, který implementuje Inquiry API a měl by (volitelně) obsahovat alespoň jeden uzel implementující Publication API.

Uzly v adresáři spravují stejná data (každý musí mít svou kopii), která jsou synchronizována pomocí metody z Replication API.

Jednotlivé uzly jsou vlastně webové služby. A tyto služby musí být také registrovány v adresáři, jehož jsou členy.

3.2.4 Registrace služeb

Nejdříve je třeba v UDDI registrovat svou organizaci a teprve potom je zde možné registrovat samotné služby. Způsob, jakým je klientovi přiřazeno právo zapisovat do adresáře, je určen politikami adresáře, pokud je obsahuje. Přesné chování a způsob řešení politik, nebo řízení přístupu, není specifikací předepsán.

Vkládaným entitám jsou přiřazeny jedinečné identifikátory – klíče. Je možné vložit záznam s vlastním klíčem.

3.2.5 Vyhledání služeb

Lze vyhledávat podle klíčových slov v názvu nebo popisu daného objektu, podle přiřazených klíčů, kategorií (*categoryBag*, nebo identifikátorů (*identifierBag*).

3.2.6 Klíče

Od verze 3 specifikace UDDI je preferováno používání doménových a odvozených klíčů. Do verze 3 byly používány *UUID* identifikátory (obrázek 3.3), které nebyly dobře zapamatovatelné člověkem.

Základem je doménový klíč a z něj generované odvozené klíče. Unikátnost je zajištěna pomocí konceptu *generátoru klíčů* (*keyGenerator*). [9]

Generátor klíčů je tModel, jehož vlastnictví (nebo přiřazení k entitě) umožňuje generovat nový klíč z hodnoty odvozeného klíče. V příkladu (obrázek 3.2) odpovídá odvozenému klíči řetězec „uddi:example.com:registry“ a řetězec „sales“ je tzv. *key specific string*. Takto lze vytvářet jedinečné klíče, které si lze zároveň snadno zapamatovat.

Hodnota KSS nesmí být nikdy „keyGenerator“. Takový klíč označuje *partition* pro generování klíčů, kde se místo řetězce „:keyGenerator“ dosadí požadované KSS. Přidělování generátoru klíčů určuje vždy vlastník nadřazeného generátoru.

```
uddi:example.com:registry:sales
```

Obrázek 3.2: Ukázka doménového klíče.

```
uddi:4CD7E4BC-648B-426D-9936-443EAAC8AE23
```

Obrázek 3.3: Ukázka UUID klíče.

3.2.7 Datové struktury v UDDI

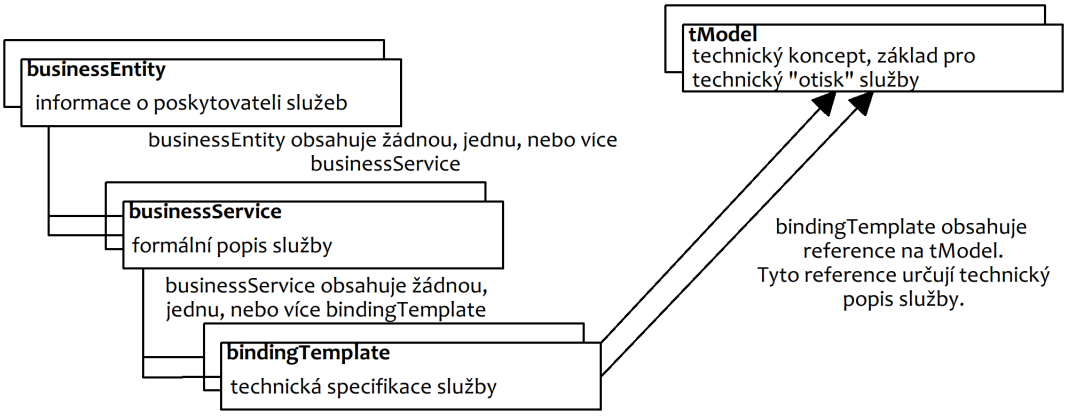
Pro reprezentaci dat se v UDDI využívají následující datové struktury (obrázek 3.4):

- businessEntity – Struktura obsahuje jméno a formální popis poskytovatele služeb (podnik, škola, atd.), může obsahovat kontaktní informace a klasifikaci v podobě přiřazených identifikátorů (identifierBag) a kategorií (categoryBag). Může obsahovat vnořené objekty businessService a bindingTemplate.
- businessService – Formální popis služby (jméno, slovní popis). Odkazuje se na nadřazenou entitu (businessEntity), obsahuje jméno, popis a seznam kategorií (categoryBag). Může obsahovat zanořené entity bindingTemplate.
- bindingTemplate – Technický popis služby. Spojuje tModel a technické kategorie s webovou službou. Obsahuje odkaz na nadřazenou entitu (businessService).
- tModel – Technický model. Je to reprezentace technického konceptu (kapitola 3.2.8).

3.2.8 tModel

Technický model (tModel) je v UDDI použit k reprezentaci konceptů (např. tModel HTTP protokolu, viz. obrázek 3.5, WSDL popis, telefonní číslo, e-mail, atd.). Umožňují jednotnou reprezentaci, a tím i jednotnou interpretaci těchto konceptů napříč adresářem.

Šablony vazeb (bindingTemplate) odkazují na stejné struktury tModelů. Tato vlastnost umožňuje sestavit tzv. *technický otisk*. Tento otisk představuje typ objektu a je určen všemi



Obrázek 3.4: UDDI struktury. [10]

tModely, na které se objekt odkazuje. Pokud šablony odkazují na stejné tModely, mají i stejný otisk, a proto mají stejný typ.

Definice tModelu není uložena uvnitř adresáře. Adresář obsahuje jen odkaz na umístění (adresu) dokumentu, který popisuje co daný tModel představuje a typ tohoto dokumentu. Většinou se jedná o slovní popis v prostém textu.

Struktura tModelu v UDDI obsahuje navíc také jedinečný klíč (kapitola 3.2.6).

```
<tModel tModelKey="uddi:uddi.org:transport:http">
     <name>uddi-org:http</name>
     <description> A~Web service that uses HTTP transport</description>
     <overviewDoc>
    <overviewURL useType="text">
      http://uddi.org/pubs/uddi_v3.htm#overHTTP
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference keyName="uddi-org:types:transport"
      keyValue="transport"
      tModelKey="uddi:uddi.org:categorization:types"/>
  </categoryBag>
</tModel>
```

Obrázek 3.5: Ukázka tModelu reprezentujícího HTTP protokol.

Použití tModelů představuje základní způsob, jak v UDDI reprezentovat data a metadata. Specifikace určuje sadu běžně používaných tModelů, které musí každý UDDI uzel podporovat. Pokud by bylo třeba, lze přidat vlastní tModel.

3.2.9 Klasifikační a identifikační systémy

Objekt v UDDI lze klasifikovat (kategorizovat) pomocí struktur, kterým se říká *categoryBag*. Tato struktura obsahuje seznam kategorií, které v určitém ohledu klasifikují organizaci (nebo obecně jakýkoliv objekt v UDDI, který categoryBag podporuje). Například to

může být obor ve kterém organizace podniká, jaký produkt vyrábí, případně zeměpisné souřadnice.

Podobně funguje identifikace. Ta je realizována strukturou *identifierBag*. Ta, na rozdíl od categoryBag, poskytuje dodatečnou identifikaci businessEntity nebo tModelu (jiné typy objektů tuto vlastnost nemají) v rámci adresáře. Jinými slovy – businessEntity je identifikována svým doménovým klíčem. IdentifierBag přidává další možnost identifikace, např. podle identifikačního čísla u daňového úřadu.

Kombinace categoryBag a identifierBag se označuje jako *Value Set.*.

Tyto systémy lze použít ke zpřesnění vyhledávání.

3.2.10 Existující implementace UDDI adresáře

Mezi open source implementace můžeme zařadit jUDDI (*http://ws.apache.org/juddi/*), což je projekt pod záštitou Apache Software Foundation. Poskytuje rozhraní kompatibilní se specifikací UDDI verze 2. Projekt je napsán v jazyce JAVA.

Svou komerční aplikaci má také Microsoft. UDDI komponenta je součástí produktu Microsoft Windows Server 2003.

IBM má svou implementaci pod názvem IBM Business Registry.

Svou implementaci UDDI v2 má zabudovánu také gSOAP (kapitola 4).

Kapitola 4

Použité nástroje

Seznam knihoven a nástrojů třetích stran, které jsem použil při implementaci.

4.1 gSOAP

gSOAP (*http://www.cs.fsu.edu/ engelen/soap.html*) je sada nástrojů, která umožňuje vygenerovat z WSDL dokumentu proxy/stub (klientská část) a skeleton (serverová část) kód v jazyce C/C++ a naopak (tedy z C/C++ hlavičkových souborů vygenerovat WSDL dokument). Dále umí mapovat zabudované i uživatelské C/C++ datové typy na ekvivalentní datové typy z XML schéma. Obsahuje vestavěný HTTP 1.1 server a možnost rozšíření pomocí zásuvných modulů (*plug-in*). Například modul pro výpisy zpráv na standardní výstup nebo modul pro zpracování HTTP GET požadavků.

Většina ostatních nástrojů umožňuje dynamické skládání dotazu. gSOAP funguje tak, že vygeneruje jednoúčelový kód. Ten se vyznačuje svou rychlostí, protože gSOAP implicitně generuje optimalizovaný kód pro zpracování XML dat. Nevýhodou je nízká pružnost.

Bohužel gSOAP obsahuje některá nepříjemná omezení. Nejvíce problematická je identifikace volané služby. Pro gSOAP platí, že jedna služba okupuje jeden port. Z toho plynou nepříjemnosti při implementaci více služeb v rámci jednoho serveru.

Jednotlivé části gSOAP jsou licencovány třemi možnými způsoby. Jedná se o licence GPL, gSOAP Public License (založená na Mozilla Public License 1.1) nebo pod proprietární licencí.

4.2 Firebird

Firebird (*http://www.firebirdsql.org/*) je relační databázový systém. Vychází z databázového systému InterBase společnosti Borland.

Firebird plně podporuje uložené procedury a triggery, využívá ACID transakcí, má malé nároky na hw, je multiplatformní a má embedded verzi.

Jako embedded se označuje taková databáze, kterou je možné připojit k programu a šířit s programem. Není ji třeba instalovat.

Firebird je šířen pod licencí Initial Developer's Public License (IDPL), která je odvozena od licence Mozilla Public License. To znamená, že je možné projekt volně používat i v kombinaci s uzavřeným softwarem (software šířený bez možnosti získat zdrojové kódy a s restriktivní licencí).

4.3 IBPP

IBPP *http://www.ibpp.org/* je knihovna poskytující rozhraní pro přístup k Firebird databázi pro programy v jazyce C++. Je jednoduchá na použití a umožňuje provádět většinu operací nad databází. To zahrnuje především spuštění transakce, její potvrzení a vrácení. Vytvoření databáze. A práci s daty v databázi pomocí dotazovacího jazyka SQL.

IBPP je *free software*[1] projekt s dostupnými zdrojovými kódy. Je šířen pod svou vlastní licencí IBPP License v1.1.

[1] „free as in free speech“

Kapitola 5

Analýza a návrh aplikace adresáře

Kapitola obsahuje analýzu aplikace, rozbor problémů a směry případného budoucího rozšíření programu.

5.1 Zadání

Adresář má sloužit pro demonstraci základního konceptu registrace a vyhledání služby.

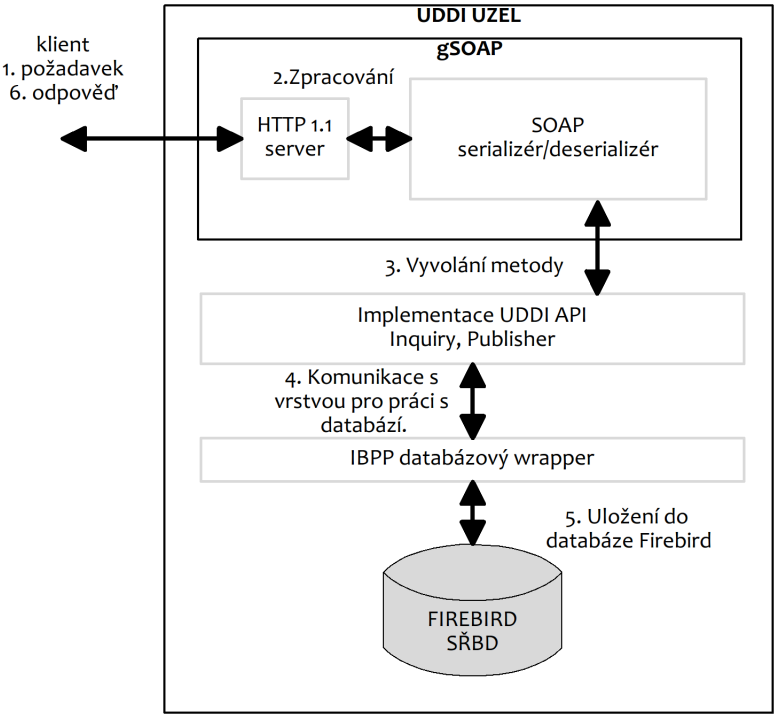
Z toho vyplývají následující požadavky:

- Adresář musí implementovat podmnožinu specifikace UDDI, konkrétně Inquiry a Publication API.
- Adresář pro účely demonstrace nevyžaduje zvláštní řízení přístupu.
- Aplikace musí zobrazovat přijaté a odeslané SOAP zprávy.
- Datový model musí obsahovat všechny důležité struktury.
- Jednoduchá inicializace uzlu/adresáře.
- Pro jednoduchost bude adresář tvořit jeden uzel.
- Aplikace by měla umět zpracovat více konkurentních spojení.
- Komunikace s uzlem přes SOAP-over-HTTP.

5.2 Konceptuální model

Uzel musí obsahovat komponentu HTTP serveru, která bude vstupní a výstupní branou pro SOAP zprávy. Ty předá SOAP procesoru, který se postará o serializaci/deserializaci zprávy a vyvolá požadovanou metodu se zadanými parametry. Metoda provede svůj kód, jejímž výsledkem bude výběr nebo vložení do databáze, poté vrátí výsledek přes SOAP procesor a HTTP server zpět klientovi.

HTTP server a SOAP procesor obstarává knihovna projektu gSOAP. Databáze je embedded verze Firebird a komunikaci s databází obstarává IBPP knihovna. Znázornění aplikace je na obrázku 5.1.



Obrázek 5.1: Konceptuální model UDDI uzlu. [10]

5.3 Datový model

Datové struktury UDDI odpovídají spíše způsobu uložení v XML dokumentu. Proto je třeba provést dekompozici na relační model dat, jak je vidět na obrázku 5.2.

UDDI datový model je příliš komplexní a rozsáhlý. Jeho plná implementace by zabrala velice hodně času. Proto jsem se rozhodl pro zjednodušení. Tato zjednodušení jsou v souladu se specifikací UDDI. Jeden z prvků, které jsem z důvodu zjednodušení nezohlednil při návrhu databáze, je internacionalizace. Každá entita může mít svůj popis a jméno. Tento element se v SOAP zprávě může vyskytovat vícekrát. Buď pokaždé pro jiný jazyk, nebo více verzí jmen a popisu ve stejném jazyce. Rozhodl jsem se, že pro účely aplikace bude stačit vždy jen první zadaná hodnota elementu v nespecifikovaném jazyce.

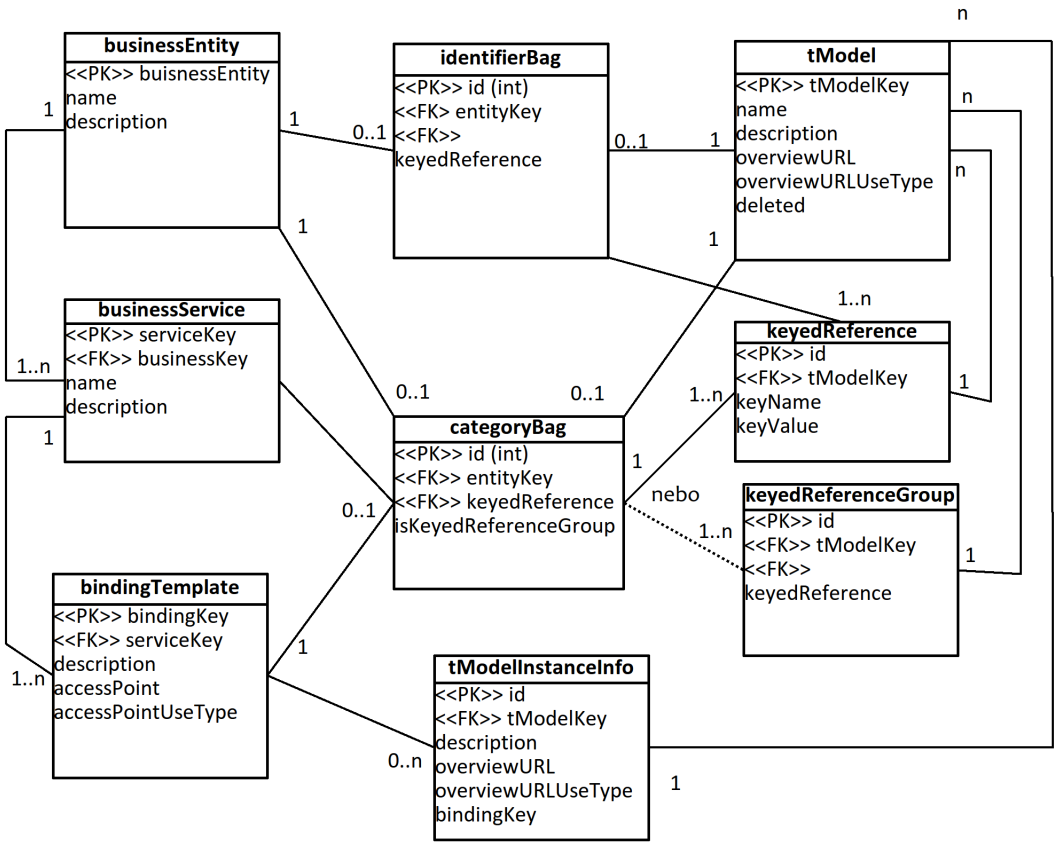
Po domluvě s Ing. Weissem jsem vynechal i některé jiné „nepodstatné“ elementy, např. kontaktní informace.

5.4 Zpracování více požadavků

Aplikace využívá knihovnu *pthreads* a konkurentní požadavky zpracovává v oddělených vláknech.

5.5 Inicializace databáze

Jelikož se databázové schéma nemění často, tak je součástí programu, včetně základních dat v podobě sady tModelů, které musí uzel nabídnout.



Obrázek 5.2: ER diagram databázového schéma. [10]

Při inicializaci se stará databáze smaže a vytvoří se nová. Dochází ke ztrátě dat. Při inicializaci také dojde k zapsání webové služby uzlu do databáze. V ukázce na obrázku 5.3 je naznačeno, jak taková struktura, implementující Publication API, může vypadat.

```
<uddi:businessEntity
businessKey="uddi:example.com:uddinode">
  <uddi:name>uddinode</uddi:name>
  <uddi:description>This is business entity which
owns this uddi node.</uddi:description>
  <uddi:businessServices>
    <uddi:businessService
serviceKey="uddi:example.com:uddinode:publication"
businessKey="uddi:example.com:uddinode">
      <uddi:bindingTemplates>
        <uddi:bindingTemplate
bindingKey="uddi:example.com:uddinode:publication:binding_publish_v3"
        serviceKey="uddi:example.com:uddinode:publication">
        <uddi:description>
This binding supports the UDDI
Programmer API SPecification for publication.
</uddi:description>
<uddi:tModelInstanceDetails>
  <uddi:tModelInstanceInfo tModelKey="uddi:uddi.org:v3_publication">
    <uddi:description>
This binding supports the UDDI Version 3.0
Programmer API Specification for publication.
    </uddi:description>
    <uddi:accessPoint useType="endPoint">
http://example.com:9000/uddi/
    </uddi:accessPoint>
</uddi:bindingTemplate>
     </uddi:bindingTemplates>
  </uddi:businessService>
</uddi:businessEntity>
```

Obrázek 5.3: Záznam webové služby implementující Publication API v UDDI uzlu.

5.6 Popis UDDI API

WSDL popis pro UDDI API poskytuje přímo organizace OASIS. Ale kvůli omezení ze strany gSOAP (jedna služba = jeden port) bylo nutné upravit WSDL dokument tak, aby byly všechna API sloučena pod jednu službu.

Zažitá konvence říká, že WSDL dokument má být přístupný přes HTTP GET požadavek, a to tak, že se za URL serveru připojí parametr *?wsdl*. To zařídil zásuvný modul gSOAP knihoven pro zpracování HTTP GET požadavků (kapitola 4).

5.7 Zobrazení výstupu

Zobrazení SOAP zpráv je klíčovou vlastností aplikace. Zajišťuje ji zásuvný modul gSOAP, který vypisuje zprávy na standardní výstup (*stdin*) včetně hlaviček protokolu HTTP.

5.8 Publication API

Metody pro registraci služeb jsou zajímavé tím, že zapisují do databáze. Modifikace databáze mohou v případě selhání způsobit porušení referenční integrity dat v databázi. Aby se tomuto zamezilo, spouští se pro každou metodu nová databázová transakce. V případě chyby způsobí transakční zpracování vrácení veškerých změn.

Registrace služby probíhá obecně tak, že server příjme od klinta SOAP zprávu, která vyvolá některou z metod určených k registraci entity do adresáře. To může být business entita, služba, šablona vazeb nebo tModel. Pokud klient nezadá atribut klíče, znamená to, že se bude vytvářet nová entita. V tom případě se z doménového klíče vygeneruje odpovídající nová hodnota klíče.

5.9 Klíče

V teoretické části (kapitola 3.2.6) jsem uvedl, že generování klíčů se řídí politikou a generátor klíčů přiřazuje vždy nadřazený subjekt. Vzhledem k povaze aplikace a absenci politik se klíče generují pevně zadaným způsobem (obrázek 5.4). Uvedený klíč odpovídá vazební šabloně s indexem 0 (ta se zvětšují globálně v celé databázi, je to hodnota generovaná databázovým systémem), která patří službě s názvem „jménoBusinessSlužby0“ a ta patří organizaci s názvem „jménoBusinessEntity0“. Jména se odvozují od hodnot předaných v SOAP požadavku.

Klient může vložit také svou hodnotu klíče. V takovém případě se hodnota klíče nekontroluje.

```
uddi:doména.tld:jménoBusinessEntity0:jménoBusinessSlužby0:bindingTemplate0
```

Obrázek 5.4: Ukázka doménového klíče.

Další problém, vyplývající z absence pokročilých funkcí, je vlastnictví záznamu a přístup k záznamům. Protože se tyto informace nijak nevedou (a specifikace to také nevyžaduje), nemůže být totožnost klienta ověřena a v případě zadání klíče s hodnotou, která už v databázi vystupuje, dojde ke kompletnímu nahrazení celé entity.

Toto jsou oblasti, které by si určitě zasloužily dále rozvést.

5.10 Inquiry API

UDDI je opravdu velmi rozsáhlé. Nejvíce to lze pozorovat na propracovanosti specifikace v oblasti vyhledávání v adresáři.

Podle specifikace lze vyhledávat podle kategorií, rozšířených identifikátorů a způsob vyhledávání lze dále specifikovat tzv. kvalifikátory vyhledávání (*find qualifier*). Já jsem z časových důvodů implementoval pouze základní vyhledávání podle klíčového slova, které je nezávislé na velikosti písmen.

Dále jsem implementoval metody, které vrací kompletní informace o hledaném objektu, podle jeho zadaného klíče.

5.11 Registrace služby

Jedna z možností, jak registrovat službu, je uložit do adresáře vazební šablonu (bindingTemplate) s přístupovým bodem služby (*accessPoint*), což je její URL adresa. A tModel reprezentující umístění jejího WSDL dokumentu. Ukázka, jak to může vypadat, je na obrázku 5.5.

Ve specifikaci UDDI verze 3 přibyla ještě jedna možnost, jak vystavit v registru pouze odkaz na samotný WSDL dokument služby. Klient, který pak tuto službu vyhledá, si musí z WSDL popisu sám zjistit informace potřebné k registraci služby. Relevantní ukázka je na obrázku 5.6. Použití wsdlDeployment je omezeno jen na UDDI v3. Jde ale o nejrychlejší a nejjednodušší způsob, jak v adresáři umístit odkaz na službu a nevyžaduje hlubší znalosti datových struktur UDDI.

5.12 Problémy

Problémy při implementaci vycházely hlavně z rozsáhlosti specifikace a nepřítomnosti podpůrných částí v programu, jako jsou politiky (širší nastavení chování uzlu) a systém pro ověření uživatele. Kvůli tomu bylo nutné některé části programu zjednodušit natolik, že nemusí přesně odpovídat specifikaci. Takovou částí je například generování klíčů a vyhledávání.

S problémy jsem se také setkal při vracení odpovědí přes serializér gSOAP. Problém byl způsoben nesprávným použitím funkce pro inicializaci paměti. GSOAP vyžaduje použití vlastních funkcí pro alokaci dynamické paměti. Tyto speciální funkce si navíc samy hlídají využití paměti a samy ji při skončení programu uvolní, takže není třeba se starat o její ruční uvolnění.

Také jsem narazil na problémy s Firebird embedded databází, která se na některých systémech odmítala spustit. Zřejmě vinou nějaké nestandardní specifické konfigurace na daném serveru. Chybu jsem neobjevil. Stejná chyba se ale projevovala i u oficiálního klientského programu pro manipulaci s databází, takže je pravděpodobné, že jde o chybu v knihovnách databáze. Při instalaci a nastavení databáze jsem postupoval podle dokumentace k databázovému serveru.

5.13 Oblasti pro vylepšení

Na práci by šlo navázat hlavně v oblasti širší implementace vyhledávání v adresáři, s podporou kvalifikátorů.

Zajímavá by byla také demonstrace replikace a spolupráce více samostatných uzlů v jednom adresáři.

Při další práci na tomto programu by také bylo vhodné implementovat nějaké politiky pro nastavení adresáře, ověření a kontrolu přístupu uživatele.

```
<tModel tModelKey="...">
   <name>StockQuote Service</name>
   <description>
      WSDL description of a standard stock quote service interface
   </description>
   <overviewDoc>
      <description>
         WSDL source document.
 </description>
 <overviewURL>
         http://example.com/stockquote.wsdl
 </overviewURL>
   </overviewDoc>
   <categoryBag>
 <keyedReference tModelKey="uddi:uddi.org:categorization:types"
    keyName="uddi-org:types"
    keyValue="wsdlSpec"/>
   </categoryBag>
</tModel>

<businessService businessKey="..." serviceKey="...">
   <name>StockQuoteService</name>
   <description> (...) </description>
   <bindingTemplates>
      <bindingTemplate>
         (...)
         <accessPoint urlType="http">
            http://example.com/stockquote
         </accessPoint>
         <tModelnstanceDetails>
            <tModelnstanceInfo tModelKey="...">
            </tModelnstanceInfo>
         <tModelnstanceDetails>
      </bindingTemplate>
   </bindingTemplates>
</businessService>
```

Obrázek 5.5: Struktura služby referuje na tModel, který reprezentuje umístění WSDL. [4]

```
<bindingTemplate bindingKey="uddi:example.org:catalog">
   <description xml:lang="en">
       Browse catalog Web service
   </description>
   <accessPoint useType="wsdlDeployment">
       http://www.example.org/CatalogWebService/catalog.wsdl
   </accessPoint>
   <categoryBag>
       <keyedReference keyName="uddi-org:types:wsdl"
       keyValue="wsdlDeployment"
       tModelKey="uddi:uddi.org:categorization:types"/>
   </categoryBag>
</bindingTemplate>
```

Obrázek 5.6: Použití wsdlDeployment. [5]

5.14 Přínosy práce

Hlavní přínos práce vidím v tom, že aplikace nevyžaduje ke svému běhu žádný dodatečný software, ani speciální nastavení. Většina dnešních UDDI serverů je vytvářena s tím, že budou fungovat nad aplikačním serverem. V prostředí výuky může být obtížné provozovat aplikační server. Kompletní UDDI servery jsou také velké a náročné na konfiguraci.

Vzhledem k tomu, že k mému programu jsou všechny knihovny linkovány staticky, není problém přenést už hotovou binární aplikaci na jakýkoliv jiný počítač s Linuxem a spustit jej. Databáze je připojena k programu, takže je možné jednoduše udržovat několik připravených databází a pro každou příležitost použít jinou, prostým nakopírováním ze zálohy.

Další přínos práce vidím v seznámení zájemců o problematiku webových služeb s tématem adresářů. Adresáře webových služeb jsou často opomíjeným tématem. Rozsáhlost specifikace UDDI také činí tuto technologii nepřehlednou.

Dále ve spojení s webovými službami kolegů, lze krásně demonstrovat interakci služeb napsaných v různých programovacích jazycích a fungujících na různých platformách, což je jeden ze základních charakteristik webových služeb.

Kapitola 6

Implementace

Následuje stručné vysvětlení struktury a implementace programu.

Program je napsán v jazyce C++, překládán byl překladačem g++ a odladěn v operačním systému Linux.

Z důvodů odlišností od POSIX normy v implementaci knihovny pthreads, v operačním systému Windows, je program nepřenosný na tuto platformu. Testován byl pouze v operačním systému Linux.

6.1 C++

Jazyk použitý pro implementaci byl C++. Vybral jsem ho s ohledem na dostupné nástroje pro práci s webovými službami – gSOAP. gSOAP umí generovat také kód v čistém C. C++ má oproti C výhodu v možnosti zpracování chyb pomocí odchycení výjimek a v objektovém přístupu.

Bohužel gSOAP negeneruje zcela čistý objektový kód, takže celá aplikace není psána s využitím objektů.

Využití C++ má také výhodu při sestavení programu, protože stačí jen překladač jazyka C++ a doprovodné knihovny, které jsou většinou součástí Linuxových distribucí.

Dále nejsou třeba žádné podpůrné prostředky, např. aplikační server, nebo běhový prostředí (*runtime*) jako u implementací v jazyce Java nebo .NET. Oproti jiným jazykům (.NET, C#) je ale implementace náročnější a vygenerovaný kód je také málo přehledný.

6.2 Uživatelské rozhranní

Aplikace se ovládá základní sadou parametrů (pro nápovědu stačí programu zadat parametr `-h`) z příkazové řádky.

Výsledky své činnosti program vypisuje na standardní výstup. Chybové stavy vypisuje na standardní chybový výstup.

6.3 Implementace serveru

Server je implementován v souboru `uddinode.cpp`. Obsahuje implementaci konkurentního HTTP serveru.

Při překladu je možné redefinicí makra `MAX_THR`, změnit maximální počet současně běžících konkurentních vláken. Redefinicí makra `MAX_QUEUE` zase maximální velikost fronty

požadavků. Výchozí hodnoty jsou 10 pro počet souběžně spuštěných vláken a 1000 pro velikost fronty požadavků na spojení.

Po spuštění serveru se inicializuje běhové prostředí gSOAP a spustí se vestavěný webový server na zadaném portu (výchozí je 9000). Webová služba serveru má svůj endpoint na adrese `<hostname>:<port>/uddi/`. Poté čeká v nekonečné smyčce na příchozí spojení. Komunikace s webovým serverem je synchronní, takže po příchodu dotazu následuje okamžitě odpověď.

Po příchodu spojení, jej vyjme z fronty příchozích spojení a zkopíruje běhové prostředí gSOAP do nového vlákna. Ve vlákně se provede vyvolání funkce a vlákno se ukončí. Metodu služby, která se spustí, určuje kód vygenerovaný gSOAP, programátor má omezené možnosti, jak toto řídit. Při změně parametrů nebo WSDL je třeba opět vygenerovat nový proxy/stub a skeleton kód.

Jediné co lze měnit bez nutnosti znovu vygenerovat kód gSOAPem, je přístupový bod služby a port na kterém běží.

Celý modul serveru je založený na kódu z dokumentace k gSOAP [11] a je to doporučený postup zpracování spojení ve více vláknech.

6.4 Rozložení modulů

Jednotlivé API jsou v modulech v adresáři `apis`.

IBPP a gSOAP mají své knihovny ve stejně pojmenovaných adresářích.

V adresáři `soap` jsou umístěny moduly a hlavičkové soubory proxy/stub a skeleton kódu. Jsou zde také ukázkové SOAP zprávy pro všechny metody. gSOAP serializér a deserializér jsou z pohledu programátora černou skříňkou.

V modulech `keyGenerator.cpp` a `tools.cpp` jsou podpůrné funkce, např. šablony funkcí pro převod mezi datovými typy jazyka C++, nebo generátor klíčů.

V adresáři `database` jsou knihovny embedded databáze, samotná databáze se jménem `uddidb.fdb` a můj pomocný modul `databasewrapper.cpp`, který dále zapouzdřuje volání IBPP funkcí pro snadnější výměnu databáze (pokud by to bylo třeba).

6.5 Testovací klientská aplikace

Přiložen je i program, který funguje jako jednoduchý testovací program `uddinodeclient` v modulu `uddiclient.cpp`.

Umí zasílat zprávy s předdefinovaným obsahem. Pro více informací stačí program spustit bez parametrů.

Kapitola 7

Závěr

Cílem práce bylo navrhnout a implementovat adresář webových služeb typu UDDI pro potřeby demonstrace při výuce a nastudování s tím spojených technologií.

V praktické části jsem popsal WS technologie zahrnující servisně orientovanou architekturu *SOA*, komunikační protokoly, protokol pro předávání zpráv SOAP, jazyk XML – na kterém je většina WS technologií založena a hlavně adresáře webových služeb.

Problematika adresářů je však mnohem rozsáhlejší, zejména těch, které fungují na konceptu UDDI, a proto nešlo a nejde tuto oblast plně popsat jednou prací.

V praktické části jsem nastínil, jak také může vývoj webových služeb vypadat a jaké nástroje je možné použít. Sada nástrojů gSOAP mi velmi ulehčila práci, i když použití není zrovna triviální.

Původní záměr zadání, nepočítal s rozsáhlostí a způsoby reprezentace popisů služeb v adresáři. Taková implementace by neodpovídala skutečnému použití adresáře. Proto jsem implementoval základní struktury UDDI tak, aby bylo možné prezentovat koncept registrace a vyhledání tak, aby odpovídal realitě. To zahrnuje především kategorizaci a identifikaci entit pomocí *categoryBag* a *identifierBag* struktur. A dále pak využití struktur tModelu uvnitř adresáře. Adresář samotný pak umožňuje přijímat a spouštět své metody, které umožňují registrovat a vyhledávat organizaci, službu i doprovodné struktury. Také je možné tyto entity smazat.

Bohužel rozsáhlost implementace se podepsala na implementaci vyhledávání, která se omezuje jen na vyhledání podle klíčového slova ve jméně entity a na získání kompletních uložených informací o službě podle jejího klíče.

Další slabinou je nedostatečné zabezpečení (spočívající v nepodpoře protokolu HTTPS), ověření uživatele a řízení přístupu k datům v databázi. Tady by si aplikace také zasloužila vylepšit. Ale myslím, že pro účely demonstrace při výuce to není příliš podstatné.

Klientskou část plní webové služby kolegů, které se do adresáře registrují a jejich protějšky se je pokusí vyhledat.

Právě implementace jiného způsobu ručního prohledávání adresáře, než jen s pomocí API, může představovat další možné rozšíření práce. Mohlo by být realizováno například jako webová služba, která využívá API uzlu k získání informací a jejich zobrazení uživateli v prostředí webového prohlížeče.

Nabízí se také implementace dalších UDDI API. Zajímavá by mohla být demonstrace replikace mezi uzly UDDI adresáře.

Přes tyto nedostatky si myslím, že aplikace splňuje zadání a je použitelná pro účely uvedené v zadání.

Při vypracování práce jsem se seznámil s celou řadou technologií, které představují velký

„boom“ v dnešní firemní praxi. Servisně orientovaná architektura je budoucnost firemních informačních systémů a s tím jak roste povědomí o těchto technologiích, rostou i možnosti využití a stabilizuje se technologie.

Konkrétně adresáře UDDI mají, mezi veřejností, velmi špatné jméno. Zklamání z veřejného provozu služby a jejího následného uzavření se uvádí jako hlavní argument proti použití UDDI. Tyto události však pouze změnily způsob, jakým se tyto adresáře začaly používat. Z prostředí internetu se dostaly do vnitrpodnikových sítí, kde plní podstatnou roli při výstavbě SOA a správě podnikových webových služeb.

Tato technologie také ztrácí kvůli své obecnosti a tím také velké složitosti. Možnosti UDDI jsou totiž obrovské a v mnoha směrech nedoceněné a určitě stojí za to se tomuto konceptu a vůbec celé technologii webových služeb, věnovat i nadále.

Literatura

[1] Booth D., Haas H., and McCabe F. Web services architecture. [online] <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>, Navštíveno 9. 5. 2008.

[2] Kubát D. Rozhraní webových služeb. Master's thesis, FIT VUT BRNO, 2007.

[3] Newcomer E. Understanding web services: Xml, wsdl, soap, and uddi, 2002. ISBN 0-201-75081-3.

[4] Curbera F., Ehnebuske D., and Rogers D. Using wsdl in a uddi registry. [online] <http://www.uddi.org/pubs/wsdlbestpractices-V1.07-Open-20020521.pdf>, Navštíveno 9. 5. 2008.

[5] Clement L., Hately A., and von Riegen C. Uddi version 3.0.2. [online] <http://uddi.org/pubs/uddi-v3.0.2-20041019.htm>, Navštíveno 9. 5. 2008.

[6] Gudgin M., Hadley M., and Mendelsohn N. Soap version 1.2 part 1: Messaging framework. [online] <http://www.w3.org/TR/2007/REC-soap12-part1-20070427/>, Navštíveno 9. 5. 2008.

[7] Kuba M. Web services. [online] <http://www.ics.muni.cz/~makub/soap/MartinKuba_WebServices_Datakon2006_clanek.pdf>, Navštíveno 9. 5. 2008.

[8] Fielding R., Irvine U.C., and Gettys J. Hypertext transfer protocol – http/1.1. [online] <ftp://ftp.isi.edu/in-notes/rfc2616.txt>, Navštíveno 9. 5. 2008.

[9] Capell S. and Rogers T. Understanding key partitions. [online] <http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-understandingkeypartitions-20050330.doc>, Navštíveno 9. 5. 2008.

[10] Erl T. Service-oriented architecture: Concepts, technology, and design, 2006. ISBN 0-13-185858-0.

[11] van Engelen R. gsoap 2.7.10 user guide. [online] <http://www.cs.fsu.edu/~engelen/soapdoc2.pdf>, Navštíveno 9. 5. 2008.

Dodatek A

Manuál a návod k použití

A.1 Sestavení

V adresáři se zdrojovými kódy stačí spustit překlad a sestavení příkazem `make`. Výsledkem překladu je serverová část (aplikace `uddinodeserver`) a jednoduchá klientská testovací aplikace (`uddinodeclient`).

A.2 Spouštění

Ke spuštění programu jsou dodány připravené skripty, které nastaví proměnné prostředí a spustí aplikaci se stejnými parametry, jaké byly předány skriptu. Skripty jsou psány pro `sh` shell.

- run-uddinode.sh – spustí server
- run-isql.sh – spustí aplikaci pro manipulaci s databází Firebird. Ta je dodána jako součást embedded databáze a umožňuje nad databází vykonávat SQL dotazy.
- run-config.sh – nastaví cestu k projektu do konfiguračního souboru Firebird databáze. Tento skript není potřeba spouštět, protože tento úkon se automaticky provede před každým spuštěním serveru.

Testovací klientskou aplikaci lze spouštět přímo příkazem `./uddinodeclient`. Spuštění bez parametrů způsobí výpis nápovědy.

A.3 Popis parametrů

- `--help | -h` – zobrazí nápovědu)
- `--verbose | -v` – zapne zobrazování stavových informací)
- `--Debug | -D` – zapne zobrazování ladících informací
- `--soap | -S` – zapne zobrazování příchozích a odchozích SOAP zpráv na standardní výstup
- `--port | -p` – spustí server na jiném, než výchozím portu, výchozí port je 9000

- `--domain | -d` – nastaví jinou, než výchozí doménu, výchozí je „example.com“, což je doménové jméno rezervované pro demonstrační účely
- `--initdb | -i` – inicializace databáze a uzlu, parametr je nutné zadat při prvním použití, nebo při nastavení výchozího stavu databáze (smaže všechny data uvnitř)
- `--nodename` – název uzlu, jednoslovný řetězec, výchozí hodnota je „uddinode“.

A.4 Použití

Nejjednoduší použití vypadá následovně:

```
./run-uddinode
```

Dojde ke spuštění serveru na portu 9000 s doménovým jménem example.com, bez inicializace databáze a s potlačeným výstupem.

Doporučené použití:

```
./run-uddinode -v -i -S
```

A.5 Komunikace

Klientský program musí implementovat volání funkcí pro Inquiry a Publication API.

Endpoint služby je `<hostname>:<port>/uddi/`

Dodatek B

Datová příloha

B.1 Popis

CD s kompletními zdrojovými kódy, přeloženou aplikací, ukázkami SOAP zpráv a použitými knihovnami.

B.2 Popis adresářů

- src (adresář se zdrojovými kódy)
  - apis (obsahuje moduly UDDI api)
  - database (knihovny databáze, databázové schémata, soubor databáze a modul wrapperu IBPP)
  - gsoap (knihovny a moduly gSOAP)
  - ibpp (knihovny pro práci s Firebird databází)
  - schemas (obsahuje XSD schémata)
  - soap (soubory proxy/stub a skeleton kódu)
  - wsdl (obsahuje WSDL soubor)
- messages (ukázky SOAP zpráv pro každou metodu)